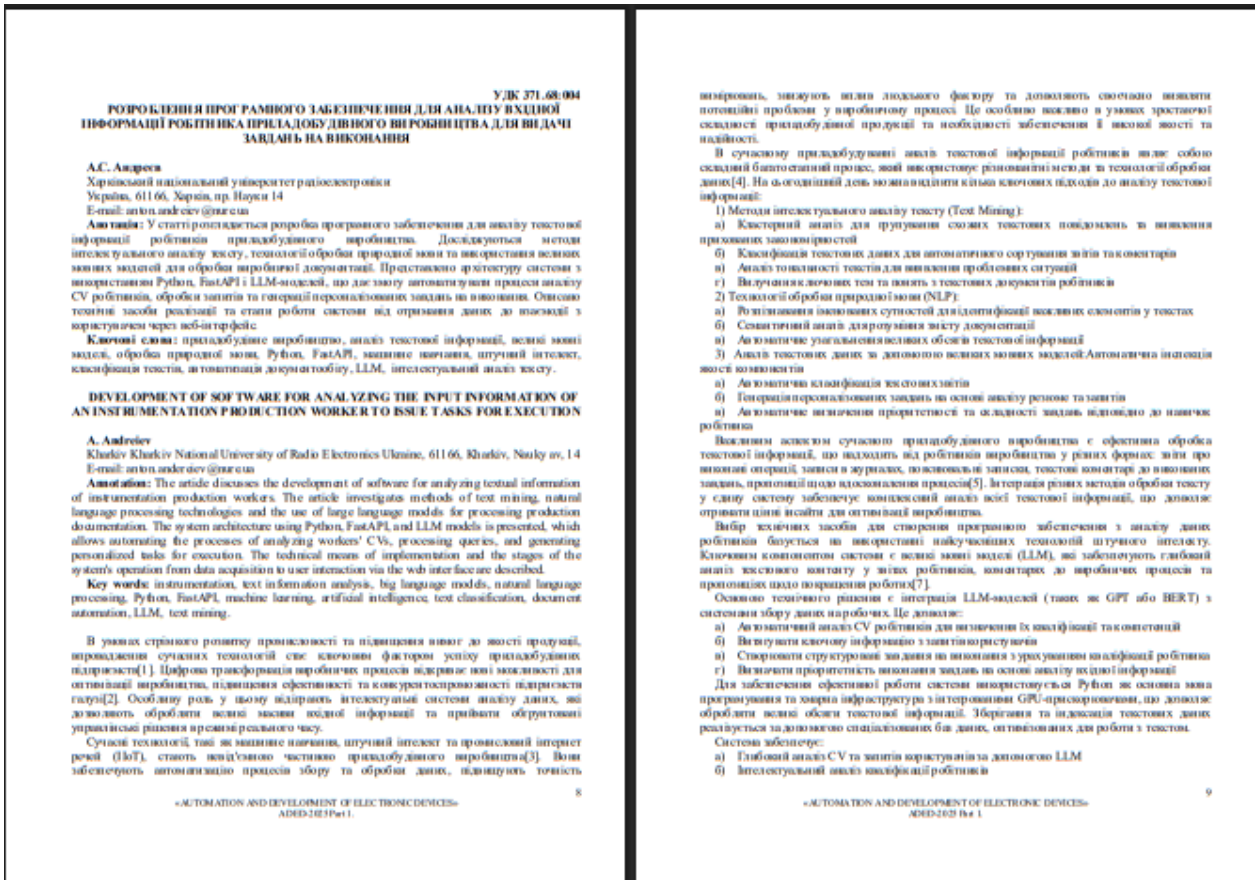
ДОДАТОК А
Апробація результатів роботи

Рисунок А.1 – Титульний аркуш

Рисунок А.2 – Зміст конференції



УДК 371.68:004

РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУ ВХІДНОЇ ІНФОРМАЦІЇ РОБІТНИКА ПРИЛАДОБУДІВНОГО ВИРОБНИЦТВА ДЛЯ ВИДАЧІ ЗАВДАНЬ НА ВИКОНАННЯ

А.С. Андрєєв
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки 14
E-mail: anton.andreiev@nure.ua

Анотація: У статті розглядається розробка програмного забезпечення для аналізу текстової інформації робітників приладобудівного виробництва. Досліджуються методи інтелектуального аналізу тексту, технології обробки природної мови та використання великих мовних моделей для обробки виробничої документації. Представлено архітектуру системи з використанням Python, FastAPI і LLM-моделей, що дає змогу автоматизувати процеси аналізу CV робітників, обробки запитів та генерації персоналізованих завдань на виконання. Описано технічні засоби реалізації та етапи роботи системи від отримання даних до взаємодії з користувачем через веб-інтерфейс.

Ключові слова: приладобудівне виробництво, аналіз текстової інформації, великі мовні моделі, обробка природної мови, Python, FastAPI, машинне навчання, штучний інтелект, класифікація тексту, автоматизація документообігу, LLM, інтелектуальний аналіз тексту.

DEVELOPMENT OF SOFTWARE FOR ANALYZING THE INPUT INFORMATION OF AN INSTRUMENTATION PRODUCTION WORKER TO ISSUE TASKS FOR EXECUTION

A. Andreiev
Kharkiv Kharkiv National University of Radio Electronics Ukraine, 61166, Kharkiv, Nauky av, 14
E-mail: anton.andreiev@nure.ua

Annotation: The article discusses the development of software for analyzing textual information of instrumentation production workers. The article investigates methods of text mining, natural language processing technologies and the use of large language models for processing production documentation. The system architecture using Python, FastAPI, and LLM models is presented, which allows automating the processes of analyzing workers' CVs, processing queries, and generating personalized tasks for execution. The technical means of implementation and the stages of the system's operation from data acquisition to user interaction via the web interface are described.

Key words: instrumentation, text information analysis, big language models, natural language processing, Python, FastAPI, machine learning, artificial intelligence, text classification, document automation, LLM, text mining.

В умовах стрімкого розвитку промисловості та підвищення вимог до якості продукції, впровадження сучасних технологій стає ключовим фактором успіху приладобудівних підприємств[1]. Цифрова трансформація виробничих процесів відкриває нові можливості для оптимізації виробництва, підвищення ефективності та конкурентоспроможності підприємств галузі[2]. Особливу роль у цьому відіграють інтелектуальні системи аналізу даних, які дозволяють обробляти великі масиви вхідної інформації та приймати обґрунтовані управлінські рішення в режимі реального часу.

Сучасні технології, такі як машинне навчання, штучний інтелект та промисловий інтернет речей (IIoT), стають невід'ємною частиною приладобудівного виробництва[3]. Вони забезпечують автоматизацію процесів збору та обробки даних, підвищують точність

вимірювань, знижують вплив людського фактору та дозволяють своєчасно виявляти потенційні проблеми у виробничому процесі. Це особливо важливо в умовах зростаючої складності приладобудівної продукції та необхідності забезпечення її високої якості та надійності.

В сучасному приладобудуванні аналіз текстової інформації робітників являє собою складний багатоетапний процес, який використовує різноманітні методи та технології обробки даних[4]. На сьогоднішній день можна виділити кілька ключових підходів до аналізу текстової інформації:

1) Методи інтелектуального аналізу тексту (Text Mining):
а) Кластерний аналіз для групування схожих текстових повідомлень та виявлення прихованих закономірностей
б) Класифікація текстових даних для автоматичного сортування звітів та коментарів
в) Аналіз тональності текстів для виявлення проблемних ситуацій
г) Вилучення ключових тем та понять з текстових документів робітників
2) Технології обробки природної мови (NLP):
а) Розпізнавання іменованих сутностей для ідентифікації важливих елементів у текстах
б) Семантичний аналіз для розуміння змісту документації
в) Автоматичне узагальнення великих обсягів текстової інформації
3) Аналіз текстових даних за допомогою великих мовних моделей:Автоматична індексація якості компонентів
а) Автоматична класифікація текстових звітів
б) Генерація персоналізованих завдань на основі аналізу резюме та звітів
в) Автоматичне визначення пріоритетності та складності завдань відповідно до навичок робітника

Важливим аспектом сучасного приладобудівного виробництва є ефективна обробка текстової інформації, що надходить від робітників виробництва у різних формах: звіти про виконані операції, записи в журналах, пояснювальні записки, текстові коментарі до виконаних завдань, пропозиції щодо вдосконалення процесів[5]. Інтеграція різних методів обробки тексту у єдину систему забезпечує комплексний аналіз всієї текстової інформації, що дозволяє отримати цінні інсайти для оптимізації виробництва.

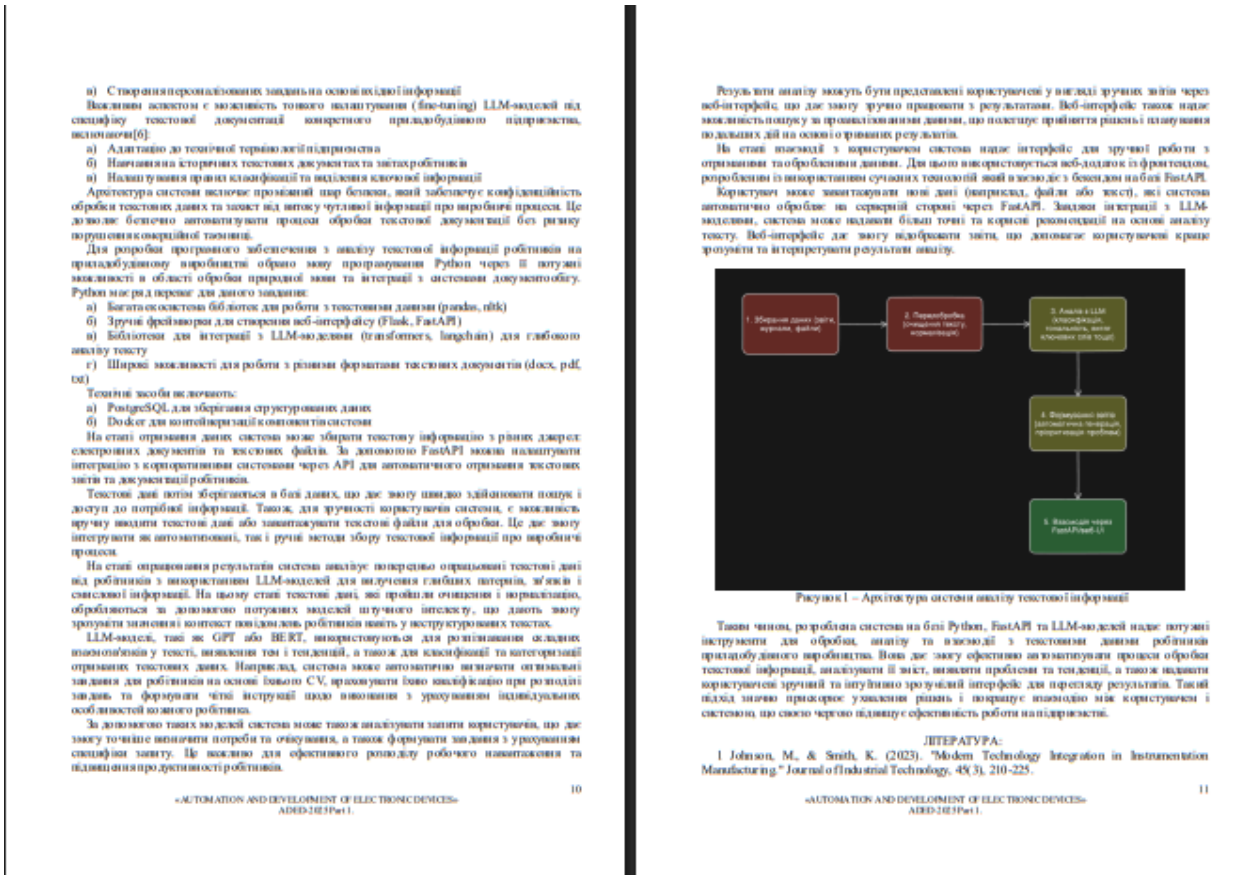
Вибір технічних засобів для створення програмного забезпечення з аналізу даних робітників базується на використанні найсучасніших технологій штучного інтелекту. Ключовим компонентом системи є великі мовні моделі (LLM), які забезпечують глибокий аналіз текстового контенту у звітах робітників, коментарях до виробничих процесів та пропозиціях щодо покращення роботи[7].

Основою технічного рішення є інтеграція LLM-моделей (таких як GPT або BERT) з системами збору даних виробничих. Це дозволяє:
а) Автоматичний аналіз CV робітників для визначення їх кваліфікації та компетенцій
б) Витягувати ключову інформацію з звітів користувачів
в) Створювати структуровані завдання на виконання з урахуванням кваліфікації робітника
г) Визначати пріоритетність виконання завдань на основі аналізу вхідної інформації

Для забезпечення ефективної роботи системи використовується Python як основна мова програмування та хмарна інфраструктура з інтегруванням GPU-прискорювачами, що дозволяє обробляти великі обсяги текстової інформації. Зберігання та індексація текстових даних реалізується за допомогою спеціалізованих баз даних, оптимізованих для роботи з текстом.

Система забезпечує:
а) Глибокий аналіз CV та звітів користувачів за допомогою LLM
б) Інтелектуальний аналіз кваліфікації робітників

Рисунок А.3 – Тезиси, перша частина



в) Створення персоналізованих завдань на основі вхідної інформації

Важливим аспектом є можливість тонкого налаштування (fine-tuning) LLM-моделей під специфіку текстової документації конкретного приладобудівного підприємства, включаючи[6]:
а) Адаптацію до технічної термінології підприємства
б) Навчання на історичних текстових документах та звітах робітників
в) Налаштування правил класифікації та виділення ключової інформації

Архітектура системи включає прошарок щодо безпеки, який забезпечує конфіденційність обробки текстових даних та захист від витоку чутливої інформації про виробничі процеси. Це дозволяє безпечно автоматизувати процеси обробки текстової документації без ризику порушення конфіденційності.

Для розробки програмного забезпечення з аналізу текстової інформації робітників на приладобудівному виробництві обрано мову програмування Python через її потужні можливості в області обробки природної мови та інтеграції з системами документообігу. Python має ряд переваг для даного завдання:
а) Багата екосистема бібліотек для роботи з текстовими даними (pandas, nltk)
б) Зручні фреймворки для створення веб-інтерфейсу (Flask, FastAPI)
в) Бібліотеки для інтеграції з LLM-моделями (transformers, langchain) для глибокого аналізу тексту
г) Широкі можливості для роботи з різними форматами текстових документів (docx, pdf, txt)

Технічні засоби включають:
а) PostgreSQL для зберігання структурованих даних
б) Docker для контейнеризації компонентів системи

На етапі отримання даних система може збирати текстову інформацію з різних джерел: електронних документів та текстових файлів. За допомогою FastAPI можна налаштувати інтеграцію з корпоративними системами через API для автоматичного отримання текстових звітів та документації робітників.

Текстові дані потім зберігаються в базі даних, що дає змогу швидко здійснювати пошук і доступ до потрібної інформації. Також, для зручності користувачів системи, є можливість вручну вводити текстові дані або завантажувати текстові файли для обробки. Це дає змогу інтегрувати як автоматизовані, так і ручні методи збору текстової інформації про виробничі процеси.

На етапі опрацювання результатів система аналізує попередньо опрацьовані текстові дані від робітників з використанням LLM-моделей для вилучення глибших патернів, зв'язків і смислової інформації. На цьому етапі текстові дані, які пройшли очищення і нормалізацію, обробляються за допомогою потужних моделей штучного інтелекту, що дають змогу зрозуміти значення і контекст повідомлень робітників навіть у неструктурованих текстах.

LLM-моделі, такі як GPT або BERT, використовуються для розпізнавання складних взаємозв'язків у тексті, виявлення тем і тенденцій, а також для класифікації та категоризації отриманих текстових даних. Наприклад, система може автоматично визначати оптимальні завдання для робітника на основі його CV, враховувати його кваліфікацію при розподілі завдань та формувати чіткі інструкції щодо виконання з урахуванням індивідуальних особливостей кожного робітника.

За допомогою таких моделей система може також аналізувати запити користувачів, що дає змогу точніше визначити потреби та очікування, а також формувати завдання з урахуванням специфіки запиту. Це важливо для ефективного розподілу робочого навантаження та підвищення продуктивності робітників.

Результати аналізу можуть бути представлені користувачеві у вигляді зручних звітів через веб-інтерфейс, що дає змогу зручно працювати з результатами. Веб-інтерфейс також надає можливість пошуку за проаналізованими даними, що полегшує прийняття рішень і планування подальших дій на основі отриманих результатів.

На етапі взаємодії з користувачем система надає інтерфейс для зручної роботи з отриманими та обробленими даними. Для цього використовується веб-додаток із фронтендом, розробленим із використанням сучасних технологій який взаємодіє з бекендом на базі FastAPI.

Користувач може завантажувати нові дані (наприклад, файли або текст), які система автоматично обробляє на серверній стороні через FastAPI. Завдяки інтеграції з LLM-моделями, система може надавати більш точні та корисні рекомендації на основі аналізу тексту. Веб-інтерфейс дає змогу відображати звіти, що допомагає користувачеві краще зрозуміти та інтерпретувати результати аналізу.

Рисунок 1 – Архітектура системи аналізу текстової інформації

Таким чином, розроблена система на базі Python, FastAPI та LLM-моделей надає потужні інструменти для обробки, аналізу та взаємодії з текстовими даними робітників приладобудівного виробництва. Вона дає змогу ефективно автоматизувати процеси обробки текстової інформації, аналізувати її зміст, виявляти проблеми та тенденції, а також надавати користувачеві зручний та інтуїтивно зрозумілий інтерфейс для перегляду результатів. Такий підхід значно прискорює ухвалення рішень і покращує взаємодію між користувачем і системою, що своєю чергою підвищує ефективність роботи на підприємстві.

ЛІТЕРАТУРА:

1 Johnson, M., & Smith, K. (2023). "Modern Technology Integration in Instrumentation Manufacturing." Journal of Industrial Technology, 45(3), 210-225.

Рисунок А.4 – Тезиси, друга частина

2 Thompson, R. (2023). "Digital Transformation of Production Processes." International Journal of Production Research, 61(4), 1287-1301.

3 Zhang, Y., & Liu, Q. (2022). "Machine Learning Applications in Instrumentation Manufacturing." Journal of Intelligent Manufacturing, 33(6), 1625-1642.

4 Li, X., & Park, J. (2022). "Text Analysis Methods for Manufacturing Documentation." Journal of Computing and Information Science in Engineering, 22(3), 031007.

5 Lee, H., & Collins, T. (2023). "Semantic Understanding of Technical Documentation." Applied Artificial Intelligence, 37(1), 2150007.

6 Zhao, Q., & Morgan, A. (2022). "Integrated Text Processing Systems for Production Optimization." Journal of Manufacturing Technology Management, 33(9), 1548-1565.

7 Wang, B., & Gonzalez, C. (2023). "Large Language Models in Worker Report Analysis." Artificial Intelligence in Engineering, 16, 100133.

8. Vladyslav, Y., & Bronnikov, A. (2020, October). ANALYSIS OF THE CMMI MODEL APPLICATION FOR SOLVING THE TASKS OF CPPS CONTROL PROCESSES AUTOMATION DEVELOPMENT. In The 4 th International scientific and practical conference "Actual trends of modern scientific research"(October 11-13, 2020) MDPC Publishing, Munich, Germany. 2020. 386 p. (p. 128).

9. Yevsieiev, V. V., & Bronnikov, A. I. (2020). Development of databases interconnection "essences" information model for cyber-physical production systems additive cyber design creation automation. Збірник Наукових Праць НУК, №3. С.56-62. DOI https://doi.org/10.15589/znp2020.3(481).7

10. Yevsieiev V., Bronnikov A. Information systems development methodologies application analysis for cyber-physical production systems development. III International scientific-practical conference "Theory, science and practice" (Japan, Tokyo, 5–8 October 2020). P. 398–401. DOI: 10.46299/ISG.2020.II.III

11. Yevsieiev V., Bronnikov A. Analysis of the cyber-physical production systems implementation impact to achieve the goals of lean production. The IIth International scientific and practical conference «Development of scientific and practical approaches in the era of globalization» (USA, Boston, 28–30 September. 2020). P.221–226. DOI:10.46299/ISG.2020.II.II.

12. Attar, H., & et al.. (2022). Zoomorphic Mobile Robot Development for Vertical Movement Based on the Geometrical Family Caterpillar. Computational Intelligence and Neuroscience, 2022, Article ID 3046116, https://doi.org/10.1155/2022/3046116.

13. Abu-Jassar AT, Attar H, Amer A, et al. Remote Monitoring System of Patient Status in Social IoT Environments Using Amazon Web Services (AWS) Technologies and Smart Health Care. International Journal of Crowd Science, 2024, https://doi.org/10.26599/IJCS.2023.9100019

14. Abu-Jassar AT, Attar H, Amer A, et al. Development and Investigation of Vision System for a Small-Sized Mobile Humanoid Robot in a Smart Environment. International Journal of Crowd Science, 2024, https://doi.org/10.26599/IJCS.2023.9100018

15. Chala, O., Yevsieiev, V., Maksymova, S., & Abu-Jassar, A. (2025). MATHEMATICAL MODEL BASED ON MULTI-AGENT REINFORCEMENT LEARNING (MARL) AND PARTIALLY OBSERVABLE MARKOV DECISION PROCESS (POMDP) FOR MODELING CARGO MOVEMENT FOR A MOBILE ROBOTS GROUP. *Multidisciplinary Journal of Science and Technology*, *5*(4), 480-489.

Науковий керівник: *Бронніков Артем Ігорович, доц., к.т.н., доцент кафедри КІТАР Харківського національного університету радіоелектроніки.*

Рисунок А.5 – Перелік джерел посилання

ДОДАТОК Б
Код програми

alembic/script.py.mako
```
"""${message}

Revision ID: ${up_revision}
Revises: ${down_revision | comma,n}
Create Date: ${create_date}

"""
from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa
${imports if imports else ""}

# revision identifiers, used by Alembic.
revision: str = ${repr(up_revision)}
down_revision: Union[str, None] = ${repr(down_revision)}
branch_labels: Union[str, Sequence[str], None] = ${repr(branch_labels)}
depends_on: Union[str, Sequence[str], None] = ${repr(depends_on)}


def upgrade() -> None:
    ${upgrades if upgrades else "pass"}


def downgrade() -> None:
    ${downgrades if downgrades else "pass"}
```

alembic/env.py
```
import asyncio
import os
from logging.config import fileConfig

from sqlalchemy import pool
from sqlalchemy.engine import Connection
from sqlalchemy.ext.asyncio import async_engine_from_config

from alembic import context
from asyncpg import Connection

from app.models.base import Base
from app.config import settings

# це об'єкт Alembic Config, який надає
# доступ до значень у використовуваному файлі .ini.
config = context.config
```

```
# Інтерпретація конфігураційного файлу для ведення журналу Python.
# Цей рядок в основному налаштовує логери.
fileConfig(config.config_file_
name)  # type: ignore

# додайте об'єкт MetaData вашої моделі тут
# для підтримки «автогенерації
# з myapp import mymodel
# target_metadata = mymodel.Base.metadata
target_metadata = Base.metadata


def get_url():
    return settings.database_config.DB_CONFIG[0]

# інші значення з конфігурації, визначені потребами env.py,
# можна встановити:
# my_important_option = config.get_main_option("my_important_option")
# ... etc.


def run_migrations_offline() -> None:
    """Запускайте міграції в автономному режимі.

    Це дозволяє налаштувати контекст лише за допомогою URL-адреси
    а не двигун, хоча двигун може бути прийнятним
    і тут також.  Пропустивши створення рушія
    нам навіть не потрібен DBAPI, щоб бути доступними.

    Виклик context.execute() тут виводить заданий рядок на екран виведення сценарію.

    """
    # url = config.get_main_option("sqlalchemy.url")
    url = get_url()
    context.configure(
        url=url,
        target_metadata=target_metadata,
        literal_binds=True,
        dialect_opts={"paramstyle": "named"},
    )

    with context.begin_transaction():
        context.run_migrations()


def do_run_migrations(connection: Connection) -> None:
    """
        Запустити процес міграції
    :param connection:
    :return:
    """
    context.configure(connection=connection, target_metadata=target_metadata)
```

```
    with context.begin_transaction():
        context.run_migrations()


async def run_async_migrations() -> None:
    """
        У цьому сценарії нам потрібно створити двигун
        і асоціювати зв'язок з контекстом.
    """

    connectable = async_engine_from_config(
        config.get_section(config.config_ini_section, {}),
        prefix="sqlalchemy.",
        poolclass=pool.NullPool,
    )

    async with connectable.connect() as connection:
        await connection.run_sync(do_run_migrations)

    await connectable.dispose()


async def run_migrations_online():
    """Запускайте міграції в режимі «онлайн».
    У цьому сценарії нам потрібно створити двигун
    і асоціювати зв'язок з контекстом.
    """
    configuration = config.get_section(config.config_ini_section)
    configuration["sqlalchemy.url"] = get_url()
    connectable = async_engine_from_config(
        configuration,
        prefix="sqlalchemy.",
        poolclass=pool.NullPool,
    )

    async with connectable.connect() as connection:
        await connection.run_sync(do_run_migrations)

    await connectable.dispose()



if context.is_offline_mode():
    run_migrations_offline()
else:
    asyncio.run(run_migrations_online())


alembic/README
Generic single-database configuration with an async dbapi.
```

```
alembic/versions/5dd18f391b75_your_message_here.py
"""Your message here

Revision ID: 5dd18f391b75
Revises:
Create Date: 2025-04-19 23:12:29.130143

"""
from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa
from sqlalchemy.dialects import postgresql

# revision identifiers, used by Alembic.
revision: str = '5dd18f391b75'
down_revision: Union[str, None] = None
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table('service_users',
    sa.Column('username', sa.String(length=64), nullable=False),
    sa.Column('email', sa.String(length=255), nullable=False),
    sa.Column('hashed_password', sa.String(), nullable=True),
    sa.Column('created', sa.DateTime(), nullable=False),
    sa.Column('email_confirmed', sa.Boolean(), nullable=False),
    sa.Column('banned', sa.Boolean(), nullable=False),
    sa.Column('role', sa.Enum('user', 'admin', name='user_roles'), nullable=False),
    sa.Column('password_reset_token', sa.String(length=64), nullable=True),
    sa.Column('password_reset_expire', sa.DateTime(), nullable=True),
    sa.Column('id', sa.Uuid(), nullable=False),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_service_users_email'), 'service_users', ['email'], unique=True)
    op.create_index(op.f('ix_service_users_username'), 'service_users', ['username'], unique=True)
    op.create_table('service_auth_tokens',
    sa.Column('secret', sa.String(length=1024), nullable=False),
    sa.Column('expiration', sa.DateTime(), nullable=False),
    sa.Column('created', sa.DateTime(), nullable=False),
    sa.Column('token_type', sa.Enum('refresh', name='token_types'), nullable=False),
    sa.Column('user_id', sa.Uuid(), nullable=True),
    sa.Column('id', sa.Uuid(), nullable=False),
    sa.ForeignKeyConstraint(['user_id'], ['service_users.id'], ondelete='CASCADE'),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_service_auth_tokens_secret'), 'service_auth_tokens', ['secret'],
unique=True)
    op.create_table('service_logs',
    sa.Column('log_type', sa.String(length=64), nullable=False),
```

```
    sa.Column('target_id', sa.Uuid(), nullable=True),
    sa.Column('created', sa.DateTime(), nullable=False),
    sa.Column('data', postgresql.JSONB(astext_type=sa.Text()), nullable=False),
    sa.Column('user_id', sa.Uuid(), nullable=True),
    sa.Column('id', sa.Uuid(), nullable=False),
    sa.ForeignKeyConstraint(['user_id'], ['service_users.id'], ),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_service_logs_created'), 'service_logs', ['created'], unique=False)
    op.create_index(op.f('ix_service_logs_log_type'), 'service_logs', ['log_type'], unique=False)
    op.create_table('task_requests',
    sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
    sa.Column('user_id', sa.Uuid(), nullable=False),
    sa.Column('worker_info', sa.Text(), nullable=False),
    sa.Column('generated_info', sa.Text(), nullable=False),
    sa.Column('created_at', sa.DateTime(), nullable=False),
    sa.ForeignKeyConstraint(['user_id'], ['service_users.id'], ),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_task_requests_user_id'), 'task_requests', ['user_id'], unique=False)
    # ### end Alembic commands ###


def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_index(op.f('ix_task_requests_user_id'), table_name='task_requests')
    op.drop_table('task_requests')
    op.drop_index(op.f('ix_service_logs_log_type'), table_name='service_logs')
    op.drop_index(op.f('ix_service_logs_created'), table_name='service_logs')
    op.drop_table('service_logs')
    op.drop_index(op.f('ix_service_auth_tokens_secret'), table_name='service_auth_tokens')
    op.drop_table('service_auth_tokens')
    op.drop_index(op.f('ix_service_users_username'), table_name='service_users')
    op.drop_index(op.f('ix_service_users_email'), table_name='service_users')
    op.drop_table('service_users')
    # ### end Alembic commands ###

app/api/dependencies/auth.py
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:
    """
        Залежність FastAPI для ендпоінтів, щоб перевіряти авторизацію користвуача перед
запуском функції
        яка підв'язана під ендпоїнт
    :param current_user: залаженість з модуля для отриманння користувача
    :return: Модель БД користувача
    """
    return current_user


async def validate_signup(signup: Signup, db_session: DBSessionDep) -> Signup:
    """
    Залежність FastAPI перевіряє користувача реєстраційних даних.

    :param signup: Реєстраційні дані, які потрібно підтвердити.
    :type signup: Pydantic схема Реєстрація

    :param db_session: Залежність від сеансу бази даних.

    :return: Signup: Підтверджені реєстраційні дані.

    :raise:
        HTTPException: Якщо ім'я користувача невірне, він вже існує,
            або якщо адреса вже існує.
    """
    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


async def validate_login(
        request: Request,
        login: LoginArgs,
        db_session: DBSessionDep
) -> LoginValidationResult:
    """
```

```
    Залежність FastAPI яка перевіряє користувача за даними логін форми.
    :param request: Request об'єкт
    :param login: Pydantic схема Login
    :param db_session: Сесія БД
    :return:
    """
    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)
        # if not check_auth_user_from_token_by_payload(ACCESS_TOKEN_TYPE, login.email,
payload):
        #     root_logger.warning(f"Invalid access token for user {login.email}. Refresh token ID:
{refresh_token_id}")



    return LoginValidationResult(user=user, refresh_token_id=refresh_token_id)



def validate_password_reset(
        current_user: CurrentUserDep
) -> User:
    """
    Залежність FastAPI перед зміною паролю
    :param current_user: залаженість з модуля для отриманння користувача
    :return:
    """
    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
```

app/api/dependencies/core.py

```
from typing import Annotated
from fastapi import Depends
from sqlalchemy.ext.asyncio import AsyncSession
```

```
from app.database import get_db_session

DBSessionDep = Annotated[AsyncSession, Depends(get_db_session)]

app/api/dependencies/user.py
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:
    """
    Залежність FastAPI для перевірки авторизації користувача проходячи різні етапи через
сервіс auth
    :param token: Access токен із сессії користувача
    :param db_session: Сессія БД
    :return: Модель БД користувача
    """
    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


CurrentUserDep = Annotated[models.User, Depends(get_current_user)]


async def get_current_user_with_refresh_token_id(token: Annotated[str,
Depends(get_access_token)], db_session: DBSessionDep) ->
CurrentUserDataWithRefreshTokenID:
    """
    Залежність FastAPI для отримання рефреш токена - автентифікованого користувача
    :param token: Access токен із сессії користувача
    :param db_session: Сессія БД
```

```
    :return: Pydantic схема 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)


CurrentRefreshDataUser = Annotated[CurrentUserDataWithRefreshTokenID,
Depends(get_current_user_with_refresh_token_id)]


async def get_admin_user(token: Annotated[str, Depends(get_access_token)], db_session:
DBSessionDep) -> models.User:
    """
    Залежність FastAPI для перевірки авторизації ADMIN користувача проходячи різні
етапи через сервіс auth
    :param token: Access токен із сессії користувача
    :param db_session: Сессія БД
    :return: Модель БД користувача
    """
    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


CurrentAdminDep = Annotated[models.User, Depends(get_admin_user)]
```

app/api/routers/ai.py

```
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(...)
):
    """
    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
    """
    feedback, content = await get_worker_statistics(file)
    await create_request_log(db_session, current_user, content, feedback)

    return feedback


@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_cv2(
    current_user: CurrentUserDep,
    db_session: DBSessionDep,
    file: UploadFile = File(...)
):
    """
    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
    """
    feedback, content = await get_task_by_cv(file=file)
    await create_request_log(db_session, current_user, content, feedback)
    return feedback


@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()
):
    """
    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
    """
    feedback = await get_worker_data(worker_info)

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

    return feedback


@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
):
    """
```

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

    - **current_user**: Авторизований користувач
    - **Повертає**: Список словників з `worker_info`, `generated_info` та `created_at`
    """
    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
    ]
```

app/api/routers/auth.py

```
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),
):
    """
    Обробляє реєстрацію користувачів.

    - **signup**: Підтверджені дані для реєстрації (email, пароль, ім'я користувача тощо.)
    - **Returns**: Ідентифікатор користувача, ім'я користувача та мітка часу створення.
    """
    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
    }


@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)
):
    """
    Перевіряє автентичність облікових даних користувача.
```

```
    - **login_result**: Містить екземпляр користувача та оновлений ідентифікатор токена після валідації.
    - **Returns**: Вітальне повідомлення та встановлює токен доступу до сеансу.
    """
    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"


@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
):
    """
    Виводить користувача з системи, анулюючи токен оновлення.

    - **refresh_data**: Включає поточний ідентифікатор користувача та ідентифікатор токена оновлення.
    - **Returns**: Підтвердження повідомлення.
    """
    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"


@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
):
    """
    Генерує новий токен доступу з дійсного токена оновлення.

    - **refresh_data**: Містить ідентифікатор користувача та його токену оновлення.
    - **Returns**: Новий рядок токену доступу.
    """
    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

app/api/routers/users.py
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
)

router = APIRouter(
    prefix="/api/users",
    tags=["Users Control System"],
    responses={404: {"description": "Ресурс не знайдено"}}
)


@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):
    """
    Отримує дані авторизованого користувача.

    - **current_user**: Об'єкт користувача з Depends.
    - **db_session**: Сесія бази даних.
    - **Повертає**: Pydantic-схему `AuthorizedUser`.
    """
    await create_log(db_session, "me", current_user)
    return current_user


@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
):
    """
    Змінює особисті дані поточного користувача.

    - **profile_update**: Нові дані профілю (ім'я, прізвище тощо)
    - **current_user**: Авторизований користувач
    - **Повертає**: Оновлений об'єкт користувача
    """
    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


@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
):
    """
    Створює токен для скидання пароля користувача.

    - **current_user**: Поточний користувач.
    - **Повертає**: Текстове повідомлення про успішність операції.
    """
    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"


@router.post(
    "/reset_password",
    summary="Скинути пароль",
    description="Скидає пароль користувача за допомогою дійсного токена скидання.",
```

```
    response_description="Повідомлення про успішне скидання пароля",
    responses={
        200: {"description": "Пароль успішно змінено"},
        400: {"description": "Невірний токен або помилкові дані"},
        401: {"description": "Недійсний токен скидання"},
        500: {"description": "Помилка при зміні пароля"}
    }
)
async def reset_password(
        reset_password_args: ResetPasswordArgs,
        db_session: DBSessionDep,
        current_user: User = Depends(validate_password_reset),
):
    """
    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}`
    """
    await create_new_password(db_session, current_user, reset_password_args)
    await create_log(db_session, "reset_password", current_user)
    return {"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):
    """
    Повертає інформацію про адміністратора.

    - **current_admin**: Поточний адміністратор
    - **Повертає**: Pydantic-схему `AuthorizedUser`
    """
    return current_admin
```

app/crud/auth.py

```
from fastapi import HTTPException
from sqlalchemy.ext.asyncio import AsyncSession
from sqlalchemy import select
from datetime import timedelta
```

```
from typing import Optional

from app.models import User as DBModelUser
from app.models import AuthToken
from app.services.auth import create_refresh_token
from app.utils.auth import utc_now
from app.errors import TokenInvalidError


async def create_refresh_auth_token(db_session: AsyncSession, user: DBModelUser) ->
AuthToken:
    """
    Створіть новий токен авторизації для поточного користувача

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param user: Користувач, якому призначено токен.


    :return: Оновити токен автора
    """
    now = utc_now()
    secret = create_refresh_token(user)
    expiration_time = now + timedelta(days=15)


    token = AuthToken(
        **{
            "secret": secret,
            "expiration": expiration_time,
            "created": now,
            "user": user,
            "token_type": "refresh",
        }
    )

    db_session.add(token)
    await db_session.commit()

    return token


async def get_refresh_auth_token(db_session: AsyncSession, user: DBModelUser, token_id: str)
-> AuthToken:
    """
    Отримайте токен refresh auth за його ідентифікатором і перевірте, чи належить він
користувачеві.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param token_id: Токен ID.
    :param user: Користувач, якому призначено токен.

    :return: Знайдено токен або Немає.
```

```
    """
    query = select(AuthToken).where(
        AuthToken.id == token_id,
        AuthToken.user_id == user.id,
        AuthToken.token_type == "refresh"
    )
    result = await db_session.execute(query)
    token = result.scalar_one_or_none()
    return token


async def get_refresh_auth_token_without_user(db_session: AsyncSession, token_id: str) ->
AuthToken:
    """
    Отримати токен оновлення авторизації за його ідентифікатором, не перевіряючи
користувача.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param token_id: Токен ID.

    :return: Found token or None.
    """
    query = select(AuthToken).where(
        AuthToken.id == token_id,
        AuthToken.token_type == "refresh"
    )
    result = await db_session.execute(query)
    token = result.scalar_one_or_none()
    return token


async def update_refresh_auth_token(db_session: AsyncSession, user: DBModelUser, token_id:
str) -> AuthToken:
    """
        Оновити існуючий токен refresh author за його ідентифікатором.

        :param db_session: Асинхронний сеанс роботи з базою даних.
        :param user: Користувач, якому належить токен.
        :param token_id: Ідентифікатор токена, який потрібно оновити.

        :return: Оновлений об'єкт AuthToken.
        :raises HTTPException: Якщо токен не знайдено або він не належить користувачеві.
    """

    token = await get_refresh_auth_token(db_session, user, token_id)

    if not token:
        raise TokenInvalidError(f"Token with ID {token_id} not found or does not belong to user
{user.id}")

    now = utc_now()
    token.secret = create_refresh_token(user)
```

```
    token.expiration = now + timedelta(days=15)
    token.created = now

    await db_session.commit()

    return token


async def delete_refresh_auth_token(db_session: AsyncSession, user: DBModelUser, token_id:
str) -> None:
    """
        Видалити токен оновлення (refresh token), який прив'язаний до конкретного
користувача.

        :param db_session: Асинхронна сесія взаємодії з базою даних.
        :param user: Користувач, якому належить токен.
        :param token_id: Унікальний ідентифікатор токена, який необхідно видалити.

        :return: None
        :raises HTTPException: Якщо токен не знайдено або він не належить зазначеному
користувачеві.
    """
    token = await get_refresh_auth_token(db_session, user, token_id)

    if not token:
        raise HTTPException(status_code=404, detail="Refresh token not found or does not belong
to the user.")

    await db_session.delete(token)
    await db_session.commit()



async def delete_refresh_auth_token_without_user(db_session: AsyncSession, token_id: str) ->
None:
    """
        Видалити токен оновлення без перевірки на приналежність до конкретного
користувача.

        Ця функція корисна у випадках, коли користувач уже не визначений (наприклад,
після втрати контексту авторизації),
        але потрібно видалити токен з бази даних за його ID.

        :param db_session: Асинхронна сесія бази даних.
        :param token_id: Ідентифікатор токена, який необхідно видалити.

        :return: None
        :raises HTTPException: Якщо токен не знайдено у базі даних.
    """
    token = await get_refresh_auth_token_without_user(db_session, token_id)

    if not token:
```

```
        raise HTTPException(status_code=404, detail="Refresh token not found or does not belong
to the user.")

    await db_session.delete(token)
    await db_session.commit()



async def handle_refresh_token(
    db_session: AsyncSession, user: DBModelUser, refresh_token_id: Optional[str]
) -> AuthToken:
    """
        Обробити логіку оновлення токену: або створити новий, або оновити наявний,
        або видалити некоректний і створити новий.

        - Якщо `refresh_token_id` не вказано — створюється новий токен.
        - Якщо токен дійсний — він оновлюється.
        - Якщо токен недійсний — він видаляється, і створюється новий.

        :param db_session: Сесія бази даних.
        :param user: Користувач, для якого створюється або оновлюється токен.
        :param refresh_token_id: Ідентифікатор токена, що підлягає оновленню.

        :return: Об'єкт нового або оновленого `AuthToken`.
    """
    if not refresh_token_id:
        return await create_refresh_auth_token(db_session, user)
    try:
        return await update_refresh_auth_token(db_session, user, refresh_token_id)
    except TokenInvalidError:
        await delete_refresh_auth_token_without_user(db_session, refresh_token_id)
        return await create_refresh_auth_token(db_session, user)
```

app/crud/get_jobs.py

```
def get_jobs():
    return """[
  {
    "title": "Інженер-конструктор",
    "company": "UFORCE",
    "location": "Київ",
    "salary": null,
    "requirements": [
      "Вища технічна освіта (електроніка, машинобудування, промисловий дизайн,
конструювання)",
      "Досвід проектування корпусних конструкцій для електронних пристроїв",
      "Володіння САПР-програмами (SolidWorks, AutoCAD, CATIA, Siemens NX або
аналогічними)",
      "Розуміння матеріалів та технологій виробництва (3D-друк, лиття, механічна обробка,
штампування)",
      "Досвід роботи з електронними компонентами та інтеграцією електронних схем у
корпусні рішення",
      "Базові знання електротехніки та теплового моделювання",
```

```
      "Вміння працювати з технічною документацією, стандартами (ГОСТ, ISO, IPC)",
      "Досвід розробки прототипів та взаємодії з виробництвом",
      "Навички командної роботи, уважність до деталей, відповідальність"
    ],
    "responsibilities": [
      "Проектування корпусів, механічних частин та конструкцій для високотехнологічних пристроїв",
      "Розробка 3D-моделей та конструкторської документації відповідно до стандартів",
      "Вибір матеріалів та технологій виробництва для виготовлення корпусних деталей",
      "Оптимізація конструкції під вимоги механічної міцності, ваги, ергономіки та тепловідведення",
      "Тестування та вдосконалення прототипів, усунення конструктивних недоліків",
      "Взаємодія з виробничими підрозділами, постачальниками та суміжними відділами",
      "Підготовка технічної документації, креслень та специфікацій",
      "Участь у створенні та розробці дизайну нових продуктів",
      "Робота з зовнішніми підрядниками для виробництва корпусів і комплектуючих",
      "Впровадження нових підходів у проектуванні та використання сучасних матеріалів і технологій"
    ],
    "source": "https://jobs.dou.ua/companies/uforce/vacancies/297799/"
  },
  {
    "title": "Інженер-технолог з механоскладальних робіт",
    "company": "ПОЖМАШИНА",
    "location": "Чернігівська обл., с.м.т. Ладан",
    "salary": null,
    "requirements": [
      "Вища освіта",
      "Досвід роботи від 1 року",
      "Вміння оперативно вирішувати поставлені завдання",
      "Відсутність шкідливих звичок",
      "Знання основ технології машинобудування (розмірний аналіз; система допусків і посадок, технічні вимірювання)",
      "Знання основних технологічних операцій складання і ремонту автомобільної техніки, промислового устаткування і механізмів",
      "Знання основ роботи із автоматизованим проектуванням технологічних процесів",
      "Вільне володіння базовими комп’ютерними програмами і оргтехнікою",
      "Робота в CAD системі"
    ],
    "responsibilities": [
      "Розробка технологічних процесів складання спеціальної автомобільної техніки",
      "Трудове нормування технологічних операцій технологічного процесу",
      "Розробка технологічних карт і схем складання (в тому числі і в CAD-системі)",
      "Контроль за дотриманням технології виготовлення на виробничих дільницях",
      "Супровід виробництва виготовлення нових та серійних виробів",
      "Підбір інструмента і пристосувань для виконання слюсарно-складальних робіт",
      "Розробка ТЗ на оснастку, обладнання",
      "Розробка технологічних інструкцій за напрямком",
      "Знання та досвід застосування принципів LEAN"
    ],
    "source": "https://pkpm.com.ua/uk/vacanc/%D1%96%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%
```

```
D1%80-%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3-
%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BE%D0%B1%D1%83%D0%B4%D1
%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F/"
  },
  {
    "title": "Інженер-програміст верстатів з ЧПК",
    "company": "ПОЖМАШИНА",
    "location": "Чернігівська обл., с.м.т. Ладан",
    "salary": null,
    "requirements": [
      "Досвід роботи інженером-програмістом верстатів з ЧПК протягом щонайменше 1
року",
      "Знання програмування на мові G-Code та здатність до його використання",
      "Вміння працювати з CAD/CAM-програмами",
      "Вміння читати технічні креслення та розуміння виробничого процесу",
      "Відповідальність, орієнтованість на результат та здатність працювати в команді"
    ],
    "responsibilities": [
      "Налагодження та програмування верстатів з ЧПК з системою FANUC, НЦ-31 для
виготовлення деталей на металообробних верстатах"
    ]"""

def get_tasks():
    return """[
        {
          "id": 1,
          "title": "Розробка креслень",
          "description": "Створення технічних креслень приладів відповідно до ТЗ",
          "tools": ["SolidWorks", "AutoCAD"],
          "deadline": "2025-04-20"
        },
        {
          "id": 2,
          "title": "3D моделювання",
          "description": "Моделювання корпусу приладу у SolidWorks",
          "tools": ["SolidWorks"],
          "deadline": "2025-04-22"
        },
        {
          "id": 3,
          "title": "Підготовка технічної документації",
          "description": "Підготовка пакету документації згідно з ДСТУ",
          "tools": ["MS Word", "PDF редактор"],
          "deadline": "2025-04-25"
        },
        {
          "id": 4,
          "title": "Калібрування прототипу",
          "description": "Проведення вимірювань та калібрування дослідного зразка",
          "tools": ["Мультиметр", "Осцилограф"],
          "deadline": "2025-04-28"
        },
```

```
{
 "id": 5,
 "title": "Звіт по результатам тестування",
 "description": "Аналіз тестових даних та створення звіту",
 "tools": ["Excel", "Word"],
 "deadline": "2025-04-30"
},
{
 "id": 6,
 "title": "Розробка ПЗ (Embedded)",
 "description": "Розробка вбудованого ПЗ для контролера приладу",
 "tools": ["C", "C++", "SCADA", "PLC"],
 "deadline": "2025-05-01"
},
{
 "id": 7,
 "title": "Backend розробка",
 "description": "Розробка backend API для обліку та управління пристроями",
 "tools": ["Python", "FastAPI", "PostgreSQL"],
 "deadline": "2025-05-05"
},
{
 "id": 8,
 "title": "Frontend інтерфейс",
 "description": "Розробка інтерфейсу користувача для керування пристроями",
 "tools": ["React", "TypeScript", "TailwindCSS"],
 "deadline": "2025-05-07"
},
{
 "id": 9,
 "title": "Інтеграція з IoT",
 "description": "Інтеграція сенсорів та пристроїв в систему обліку",
 "tools": ["MQTT", "Raspberry Pi", "Python"],
 "deadline": "2025-05-08"
},
{
 "id": 10,
 "title": "Тестування ПЗ",
 "description": "Написання unit та integration тестів для критичного функціоналу",
 "tools": ["PyTest", "Jest", "Postman"],
 "deadline": "2025-05-09"
},
{
 "id": 11,
 "title": "CI/CD автоматизація",
 "description": "Налаштування CI/CD для автоматичного деплою на сервери",
 "tools": ["GitHub Actions", "Docker", "Kubernetes"],
 "deadline": "2025-05-12"
},
{
 "id": 12,
 "title": "Документація API",
```

```
  "description": "Створення Swagger-документації для REST API",
  "tools": ["Swagger", "Redoc", "OpenAPI"],
  "deadline": "2025-05-14"
},
{
  "id": 13,
  "title": "Міграція бази даних",
  "description": "Створення міграцій, оптимізація структури даних",
  "tools": ["Alembic", "PostgreSQL", "SQLAlchemy"],
  "deadline": "2025-05-16"
},
{
  "id": 14,
  "title": "Прибирання приміщень",
  "description": "Щоденне вологе прибирання офісів, лабораторій і санвузлів",
  "tools": ["Пилосос", "Швабра", "Мийні засоби"],
  "deadline": "2025-04-19"
},
{
  "id": 15,
  "title": "Охорона об'єкту",
  "description": "Цілодобове чергування на контрольно-пропускному пункті",
  "tools": ["Камери", "Сигналізація", "Радіостанція"],
  "deadline": "2025-12-31"
},
{
  "id": 16,
  "title": "Закупівля канцтоварів",
  "description": "Формування списку та замовлення офісних матеріалів",
  "tools": ["Excel", "Маркетплейс"],
  "deadline": "2025-05-02"
},
{
  "id": 17,
  "title": "Доставка компонентів",
  "description": "Прийом, перевірка та розподіл деталей по відділам",
  "tools": ["Візок", "Накладні", "Термінал"],
  "deadline": "2025-04-24"
},
{
  "id": 18,
  "title": "Організація харчування",
  "description": "Контроль роботи їдальні та оновлення меню",
  "tools": ["Google Sheets", "Месенджери"],
  "deadline": "2025-04-25"
},
{
  "id": 19,
  "title": "Проведення інструктажу з охорони праці",
  "description": "Інструктаж для нових співробітників з правил безпеки",
  "tools": ["Презентація", "Журнал інструктажів"],
  "deadline": "2025-05-03"
```

```
    },
    {
      "id": 20,
      "title": "Архівування документів",
      "description": "Сканування, упорядкування та зберігання юридичних документів",
      "tools": ["Сканер", "PDF редактор", "Папки"],
      "deadline": "2025-05-10"
    }
  ]"""

app/crud/log.py
from sqlalchemy.ext.asyncio import AsyncSession
from uuid import UUID

from app.models import User as DBModelUser
from app.models.log import Log
from app.utils.auth import utc_now


async def create_log(
    session: AsyncSession,
    log_type: str,
    user: DBModelUser | None,
    target_id: UUID | None = None,
    data: dict = None
):
  now = utc_now()

  log = Log(
    **{
      "created": now,
      "target_id": target_id,
      "log_type": log_type,
      "user": user,
      "data": data,
    }
  )

  session.add(log)
  await session.commit()

  return log

app/crud/request_log.py
from sqlalchemy import select
from sqlalchemy.ext.asyncio import AsyncSession

from app.models import User as DBModelUser, TaskRequestLog


async def get_request_logs_user(
  db_session: AsyncSession,
```

```
    current_user: DBModelUser,
):
    """
    Отримання усіх запитів та результатів користувача

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param current_user: Поточний користувач, який редагує дані.
    :return: logs: Усі записи користувача
    """
    result = await db_session.execute(
        select(TaskRequestLog).where(TaskRequestLog.user_id == current_user.id)
    )
    logs = result.scalars().all()

    return logs


async def create_request_log(
        db_session: AsyncSession,
        current_user: DBModelUser,
        worker_info: str,
        feedback: str
) -> TaskRequestLog:
    """
    Створити новий запис у TaskRequestLog

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param current_user: Поточний користувач, який редагує дані.
    :param worker_info: Текст користувача - prompt
    :param feedback: Відповідь від моделі

    :return: Створений об'єкт користувача.
    """

    log_entry = TaskRequestLog(
        user_id=current_user.id,
        worker_info=worker_info,
        generated_info=str(feedback)
    )
    db_session.add(log_entry)
    await db_session.commit()

    return log_entry
```

app/crud/user.py

```
import logging
from typing import List
from uuid import UUID

from fastapi import HTTPException
from sqlalchemy import select, func
from sqlalchemy.ext.asyncio import AsyncSession
```

```
from sqlalchemy.orm import selectinload
from datetime import timedelta

from app.models import User as DBModelUser
from app.schemas.auth import Signup
from app.schemas.user import UpdateProfile, ResetPasswordArgs
from app.utils.auth import hash_password, utc_now, is_protected_username, verify_password
from app.services.auth import new_token


logging.basicConfig(level=logging.DEBUG)


async def get_all_users(db_session: AsyncSession) -> List[DBModelUser]:
    """
    Отримати список усіх користувачів у базі даних.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :return: Список об'єктів користувачів (DBModelUser).
    """
    stmt = select(DBModelUser)
    result = await db_session.execute(stmt)
    users = result.scalars().all()
    return users



async def get_user(db_session: AsyncSession, user_id: UUID):
    """
    Отримати користувача за його UUID.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param user_id: Унікальний UUID користувача.
    :return: Об'єкт користувача або None, якщо не знайдено.
    """
    return await db_session.scalar(
        select(DBModelUser).filter(DBModelUser.id == user_id)
    )


async def get_user_by_username(db_session: AsyncSession, username: str) -> DBModelUser |
None:
    """
    Отримати користувача за його іменем користувача (username), нечутливим до регістру.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param username: Ім'я користувача для пошуку.
    :return: Об'єкт користувача або None.
    """
    return await db_session.scalar(
        select(DBModelUser).filter(func.lower(DBModelUser.username) == username.lower())
    )
```

```
async def get_user_by_email(db_session: AsyncSession, email: str) -> DBModelUser | None:
    """
    Отримати користувача за email, нечутливим до регістру.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param email: Email користувача.
    :return: Об'єкт користувача або None.
    """
    return await db_session.scalar(
        select(DBModelUser).filter(func.lower(DBModelUser.email) == email.lower())
    )


async def get_admin_user_by_email(db_session: AsyncSession, email: str) -> DBModelUser |
None:
    """
    Отримати адміністратора за email, якщо його роль — 'admin'.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param email: Email користувача.
    :return: Об'єкт адміністратора або None.
    """
    return await db_session.scalar(
        select(DBModelUser).filter(
            func.lower(DBModelUser.email) == email.lower(),
            DBModelUser.role == "admin"
        )
    )


async def create_user(db_session: AsyncSession, signup: Signup) -> DBModelUser:
    """
    Створити нового користувача у системі.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param signup: Об'єкт зі схемою Signup (email, пароль, ім'я).
    :return: Створений об'єкт користувача.
    """
    password_hash = hash_password(signup.password)
    now = utc_now()

    user = DBModelUser(
        username=signup.username,
        email=signup.email,
        hashed_password=password_hash,
        created=now,
    )

    db_session.add(user)
    await db_session.commit()
```

```
    return user


async def update_user_profile(db_session: AsyncSession, current_user: DBModelUser,
profile_update: UpdateProfile):
    """
    Оновити профіль поточного користувача (email або username).

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param current_user: Поточний користувач, який редагує дані.
    :param profile_update: Дані для оновлення (email, username).
    :return: Оновлений об'єкт користувача.
    :raises HTTPException: Якщо username/email уже зайнятий або недопустимий.
    """
    if profile_update.username:
        if is_protected_username(profile_update.username):
            logging.debug(f"Invalid username detected: {profile_update.username}")
            raise HTTPException(status_code=400, detail="Invalid username")

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

        current_user.username = profile_update.username

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

        current_user.email = profile_update.email

    await db_session.commit()
    return current_user


async def create_password_token(db_session: AsyncSession, user: DBModelUser):
    """
    Створити токен для скидання пароля користувача, дійсний протягом 1 години.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param user: Користувач, для якого створюється токен.
    :return: Оновлений об'єкт користувача з токеном.
    """
    user.password_reset_expire = utc_now() + timedelta(hours=1)
    user.password_reset_token = new_token()

    db_session.add(user)
    await db_session.commit()

    return user
```

```
async def create_new_password(db_session: AsyncSession, user: DBModelUser,
reset_password_args: ResetPasswordArgs):
    """
    Встановити новий пароль користувачу після перевірки токена та старого пароля.

    :param db_session: Асинхронний сеанс роботи з базою даних.
    :param user: Користувач, який скидає пароль.
    :param reset_password_args: Аргументи: старий пароль і новий пароль.
    :return: None
    :raises HTTPException:
        - Якщо токен скидання пароля не встановлено.
        - Якщо старий пароль невірний.
    """
    if user.password_reset_token is None:
        raise HTTPException(status_code=400, detail="Need password reset token")

    if not verify_password(reset_password_args.old_password, user.hashed_password):
        raise HTTPException(status_code=400, detail="Incorrect old password")

    user.hashed_password = hash_password(reset_password_args.password)
    user.password_reset_expire = None
    user.password_reset_token = None

    db_session.add(user)
    await db_session.commit()
```

app/models/enums/user.py

```
from enum import Enum


class UserRole(Enum):
    user = "user"
    admin = "admin"
```

app/models/__init__.py

```
from .user import User
from .auth import AuthToken
from .log import Log
from .request_log import TaskRequestLog
```

app/models/auth.py

```
from sqlalchemy import String, ForeignKey, Enum
from sqlalchemy.orm import mapped_column, relationship, Mapped
from datetime import datetime
from .base import Base


class AuthToken(Base):
    __tablename__ = "service_auth_tokens"
```

```
    secret: Mapped[str] = mapped_column(String(1024), unique=True, index=True)
    expiration: Mapped[datetime]
    created: Mapped[datetime]

    token_type: Mapped[str] = mapped_column(Enum("refresh", name="token_types"))

    user_id = mapped_column(ForeignKey("service_users.id", ondelete="CASCADE"))
    user: Mapped["User"] = relationship("User", back_populates="refresh_tokens")
```

app/models/base.py

```
from sqlalchemy.ext.hybrid import hybrid_property
from sqlalchemy.orm import Mapped, mapped_column
from sqlalchemy.ext.asyncio import AsyncAttrs
from sqlalchemy.orm import DeclarativeBase
from uuid import UUID, uuid4


class Base(AsyncAttrs, DeclarativeBase):
    id: Mapped[UUID] = mapped_column(primary_key=True, default=uuid4)

    @hybrid_property
    def reference(self):
        return str(self.id)
```

app/models/log.py

```
from sqlalchemy.dialects.postgresql import JSONB
from sqlalchemy.orm import mapped_column, relationship, Mapped
from sqlalchemy import ForeignKey, String
from datetime import datetime
from .base import Base
from uuid import UUID

class Log(Base):
    __tablename__ = "service_logs"

    log_type: Mapped[str] = mapped_column(String(64), index=True)
    target_id: Mapped[UUID] = mapped_column(nullable=True)
    created: Mapped[datetime] = mapped_column(index=True)
    data: Mapped[dict] = mapped_column(JSONB, default={})

    user_id = mapped_column(ForeignKey("service_users.id"))
    user: Mapped["User"] = relationship(foreign_keys=[user_id])
```

app/models/request_log.py

```
from sqlalchemy.orm import Mapped, mapped_column, relationship
from sqlalchemy import String, Text, DateTime, ForeignKey, Integer
from datetime import datetime

from app.models.base import Base
```

```
class TaskRequestLog(Base):
    __tablename__ = "task_requests"

    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)

    user_id: Mapped[int] = mapped_column(ForeignKey("service_users.id"), index=True)

    worker_info: Mapped[str] = mapped_column(Text)
    generated_info: Mapped[str] = mapped_column(Text)

    created_at: Mapped[datetime] = mapped_column(default=datetime.utcnow)

    # зв'язок назад до користувача
    user: Mapped["User"] = relationship("User", back_populates="task_requests")
```

app/models/user.py

```
from sqlalchemy.orm import Mapped, mapped_column, relationship
from sqlalchemy import String, Enum, Numeric
from datetime import datetime

from .base import Base


class User(Base):
    __tablename__ = "service_users"

    username: Mapped[str] = mapped_column(String(64), index=True, unique=True)
    email: Mapped[str] = mapped_column(String(255), index=True, unique=True)
    hashed_password: Mapped[str] = mapped_column(nullable=True)
    created: Mapped[datetime]

    email_confirmed: Mapped[bool] = mapped_column(default=False)
    banned: Mapped[bool] = mapped_column(default=False)

    role: Mapped[str] = mapped_column(Enum("user", "admin", name="user_roles"),
default="user")

    password_reset_token: Mapped[str] = mapped_column(String(64), nullable=True)
    password_reset_expire: Mapped[datetime] = mapped_column(nullable=True)


    refresh_tokens: Mapped[list["AuthToken"]] = relationship("AuthToken",
back_populates="user", cascade="all, delete-orphan")

    task_requests: Mapped[list["TaskRequestLog"]] = relationship(
        "TaskRequestLog", back_populates="user", cascade="all, delete-orphan"
    )
```

app/schemas/__init__.py

```
from datetime import datetime
from decimal import Decimal
```

```
from pydantic import BaseModel, Field, EmailStr, field_validator, PlainSerializer
from typing import Annotated

from app.utils.auth import to_timestamp

datetime_pd = Annotated[
    datetime,
    PlainSerializer(lambda x: to_timestamp(x),
                    return_type=int,
                    ),
]


class UsernameArgs(BaseModel):
    username: str = Field(pattern="^[A-Za-z][A-Za-z0-9_]{4,63}$", examples=["Antony"],
description="Унікальне ім'я користувача")


class NameArgs(BaseModel):
    name: str = Field(pattern="^[A-Za-z][A-Za-z0-9_]{4,63}$", examples=["Crew"])


class EmailArgs(BaseModel):
    email: EmailStr = Field(examples=["your_email@gmail.com"], description="Електронна
пошта користувача")

    @field_validator("email")
    @classmethod
    def check_email(cls, value: EmailStr) -> EmailStr:
        if "+" in value:
            raise ValueError("Email contains unacceptable characters")

        return value


class PasswordArgs(BaseModel):
    password: str = Field(min_length=8, max_length=128, examples=["password"],
description="Пароль користувача")


class Title(BaseModel):
    title: str = Field(min_length=1, max_length=255, examples=["FPW Dron 6s"])


class Price(BaseModel):
    price: Decimal = Field(gt=0, decimal_places=2)


class Total(BaseModel):
    total: Decimal = Field(gt=0, decimal_places=2)
```

app/schemas/auth.py

```
from . import BaseModel, datetime_pd, UsernameArgs, PasswordArgs, EmailArgs
from app.models.user import User

from pydantic import Field


class Token(BaseModel):
    access_token: str
    token_type: str


class TokenInfo(BaseModel):
    access_token: str
    refresh_token: str | None = None


class TokenData(BaseModel):
    email: str


class TokenResponse(BaseModel):
    expiration: datetime_pd = Field(examples=[1686088809])
    created: datetime_pd = Field(examples=[1686088809])
    secret: str = Field(
        examples=["CQE-CTXVFCYoUpxz_6VKrHhzHaUZv68XvxV-3AvQbnA"]
    )


class Signup(UsernameArgs, PasswordArgs, EmailArgs):
    pass


class LoginArgs(PasswordArgs, EmailArgs):
    pass


class LoginValidationResult(BaseModel):
    user: User
    refresh_token_id: str | None = None

    class Config:
        arbitrary_types_allowed = True
```

app/schemas/user.py

```
from pydantic import BaseModel, ConfigDict, EmailStr, Field, constr, field_validator
from typing import List, Optional
from datetime import datetime
from uuid import UUID
from decimal import Decimal

from . import UsernameArgs, PasswordArgs, EmailArgs
```

```
from app.models import User as DB_User

class User(PasswordArgs, EmailArgs):
    pass


class AuthorizedUser(UsernameArgs, EmailArgs):
    created: datetime


class AuthorizedUserWithBalance(UsernameArgs, EmailArgs):
    created: datetime
    balance: Decimal = Field(..., max_digits=10, decimal_places=2)

    class Config:
        # Настройки Pydantic для поддержки Decimal
        json_encoders = {Decimal: lambda v: str(v)}

class UpdateProfile(UsernameArgs, EmailArgs):
    username: Optional[constr(pattern="^[A-Za-z][A-Za-z0-9_]{4,63}$")] = Field(
        None,
        examples=["New Username or None"]
    )

    email: Optional[EmailStr] = Field(None, examples=["New Email or None"])

    @field_validator("email")
    @classmethod
    def check_email(cls, value: EmailStr) -> EmailStr:
        if value is None:
            return value
        if "+" in value:
            raise ValueError("Email contains unacceptable characters")

        return value


class ResetPasswordArgs(PasswordArgs):
    old_password: str = Field(min_length=8, max_length=128, examples=["old_password"])


class UserDeleteResponse(BaseModel):
    success: bool


class CurrentUserDataWithRefreshTokenID(BaseModel):
    user: DB_User
    refresh_token_id: Optional[str]

    class Config:
        arbitrary_types_allowed = True
```

app/services/auth.py

```
import os
import uuid

import jwt
import secrets
from fastapi import Depends, HTTPException, Request
from datetime import datetime, timezone, timedelta
from starlette import status
from sqlalchemy.ext.asyncio import AsyncSession

from app.config import settings
from app.errors import credentials_exception
from app.models import AuthToken
from app.schemas.user import User
from app.constants import TOKEN_TYPE_FIELD, ACCESS_TOKEN_TYPE,
REFRESH_TOKEN_TYPE
from app.models.user import User as DB_User
from app.utils.auth import utc_now

ACCESS_TOKEN_SECRET_KEY = settings.auth_jwt.access_token_secret_key
ACCESS_TOKEN_ALGORITHM = settings.auth_jwt.access_token_algorithm


def new_token():
    """
        Генерує нових випадкових токен

    :returns: The new random token
    """
    return secrets.token_urlsafe(32)


def get_access_token(request: Request):
    """
    Отримує access-токен із сесії користувача.

    :param request: HTTP-запит FastAPI з активною сесією.
    :return: Токен доступу (JWT) як рядок.
    :raises HTTPException: Якщо токен відсутній або недійсний.
    """
    token = request.session.get('access_token')

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

    return token



def get_email_from_token_payload(token: str | bytes) -> str:
    """
```

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

    :param token: JWT токен у вигляді рядка або байтів.
    :return: Email користувача (sub).
    :raises HTTPException: Якщо токен має неправильний тип або не містить email.
    """
    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


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

    :param token: JWT-токен у вигляді рядка або байтів.
    :param public_key: Публічний ключ для перевірки підпису.
    :param algorithm: Алгоритм шифрування (за замовчуванням RS256 або HS256).
    :param verify_exp: Чи потрібно перевіряти термін дії токена.
    :return: Розкодований словник payload.
    :raises jwt.InvalidTokenError: Якщо токен невалідний.
    """
    decoded = jwt.decode(
        token,
        public_key,
        algorithms=[algorithm],
        options={"verify_exp": verify_exp}
    )
    return decoded



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:
    """
    Закодувує JWT-токен з підписом, встановленням `exp`, `iat` і `jti`.
```

```
    :param payload: Основні дані токена.
    :param private_key: Приватний ключ для підпису.
    :param algorithm: Алгоритм шифрування (RS256, HS256).
    :param expire_minutes: Час життя токена у хвилинах (якщо не задано `expire_timedelta`).
    :param expire_timedelta: Час життя токена як об'єкт timedelta (перебиває
`expire_minutes`).
    :return: Підписаний JWT у вигляді рядка.
    """
    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



def get_token_payload(
        token: str,
        check_expired_token: bool = True
) -> dict:
    """
    Декодує JWT та отримати payload із токена з перевіркою терміну дії.

    :param token: JWT access/refresh токен.
    :param check_expired_token: Чи перевіряти термін дії (exp).
    :return: Payload у вигляді словника.
    :raises HTTPException: Якщо токен невалідний або підпис неправильний.
    """
    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
```

```
def validate_token_type(
        payload: dict,
        token_type: str,
) -> bool:
    """
    Перевіряє тип токена у payload — чи відповідає очікуваному.

    :param payload: Розкодований payload токена.
    :param token_type: Очікуваний тип токена (наприклад, "access", "refresh").
    :return: True, якщо тип відповідає.
    :raises HTTPException: Якщо тип токена не співпадає з очікуваним.
    """
    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}"
    )



def check_auth_user_from_token_by_payload(
    token_type: str,
    user_email: str,
    payload: dict,
) -> bool:
    """
    Перевіряє, чи співпадає email у payload з поточним користувачем і чи тип токена
валідний.

    :param token_type: Очікуваний тип токена ("access" або "refresh").
    :param user_email: Email користувача, що виконує запит.
    :param payload: Payload із JWT.
    :return: True, якщо токен валідний і email співпадає. False — інакше.
    """
    try:
        validate_token_type(payload, token_type)
        if payload.get("sub") != user_email:
            return False
        return True
    except Exception as e:
        return False



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:
    """
    Створює JWT-токен заданого типу з переданим payload.

    :param token_type: Тип токена — "access" або "refresh".
    :param token_data: Дані, які необхідно вбудувати у payload.
    :param expire_minutes: Термін дії у хвилинах (опціонально).
    :param expire_timedelta: Альтернативний спосіб задання терміну дії.
    :return: Створений підписаний JWT токен.
    """
    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,
    )



def create_access_token(user: User, refresh_token: AuthToken) -> str:
    """
    Створює JWT access-токен для вхідного користувача з посиланням на refresh-токен.

    :param user: Користувач, для якого створюється токен.
    :param refresh_token: Об'єкт refresh-токена.
    :return: Access-токен (JWT) у вигляді рядка.
    """
    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,
    )



def create_refresh_token(user: User) -> str:
    """
    Створює JWT refresh-токен для користувача.

    :param user: Користувач, email якого буде використано в payload.
    :return: Refresh-токен у вигляді підписаного JWT.
    """
    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_days),
    )
```

app/services/data_anls.py

```
import asyncio
import io
import json
import os

from fastapi import UploadFile
from pypdf import PdfReader
import re

from openai import OpenAI

from app.crud.get_jobs import get_jobs, get_tasks


async def get_worker_statistics(file: UploadFile):
    """
    Прочитати PDF-файл працівника, витягнути текст і передати його в OpenAI для аналізу.

    :param file: Завантажений файл (формат PDF), що містить опис працівника або його резюме.
    :return: Рекомендація 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
    :raises ValueError: Якщо OpenAI повертає неправильний формат або JSON не розпізнано.
    """
    content = await file.read()
    return await __get_worker_statistics(content)


async def get_task_by_cv(file: UploadFile):
    """
    Надіслати текстову інформацію про працівника до OpenAI для генерації персоналізованого завдання виходячи із CV.

    :param file: Завантажений файл (формат PDF), що містить опис працівника або його резюме.
    :return: Рекомендоване завдання у форматі JSON, підібране відповідно до навичок працівника.
    """
    content = await file.read()
    text = __byte_to_text(content)

    return await __get_worker_data(text, return_text=True)
```

```
async def __get_worker_statistics(content: bytes):
    """
    Внутрішня функція для обробки PDF та виконання запиту до OpenAI API.

    :param content: Вміст PDF-файлу у вигляді байтів.
    :return: JSON-об'єкт із рекомендацією щодо посади для працівника.
    """
    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


async def get_worker_data(text: str):
    """
    Надіслати текстову інформацію про працівника до OpenAI для генерації персоналізованого завдання.

    :param text: Інформація про працівника (вільна форма, текст або JSON).
    :return: Рекомендоване завдання у форматі JSON, підібране відповідно до навичок працівника.
    """
    return await __get_worker_data(text)
```

```
async def __get_worker_data(text: str, return_text: bool = False):
    """
    Внутрішня функція, яка виконує запит до OpenAI з описом працівника.

    :param text: Повна текстова інформація про працівника (включаючи навички, освіту тощо).
    :return: Завдання, яке найкраще підходить працівнику, у форматі JSON:
        - task_id: ID завдання
        - task_title: Назва завдання
        - reason: Чому саме це завдання пасує працівнику
    """
    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)


def __byte_to_text(content: bytes) -> str:
    """
        Видає текст PDF файлу виходячи с представлених байтів
    :param content: передані bytes документа
    :return: str: текст с PDF документа
    """
    reader = PdfReader(io.BytesIO(content))
    text = ""
    for page in reader.pages:
        text += page.extract_text()

    return text
```

```
# if __name__ == '__main__':
#     loop = asyncio.get_event_loop()
#     with open("C:/Users/Ajeszny/Downloads/Rezjume_Pruladobuduvannya_Ivanenko.pdf",
"rb") as f:
#         bt = f.read()
#     r = loop.run_until_complete(__get_worker_data("Сергій Іваненко має досвід роботи
інженером-конструктором у ТОВ 'ТехноПром', де успішно розробляв креслення та 3D-
моделі приладів, використовуючи SolidWorks. Його освіта у сфері приладобудування та
вміння працювати з технічною документацією підтверджують його високий рівень
кваліфікації. Додатково, Сергій має навички роботи з контрольно-вимірювальними
приладами, що є важливим аспектом для посади інженера-конструктора. Враховуючи
його досвід роботи та технічні навички, він буде цінним доповненням до команди,
спроможним внести свій внесок у розробку нових технологій та оптимізацію вже
існуючих рішень."))
#     print(r)
#     r = loop.run_until_complete(__get_worker_statistics(bt))
#     print(r)

app/utils/auth.py
import os
import jwt
import secrets
from datetime import datetime, timezone, timedelta
from fastapi.security import OAuth2PasswordBearer
from passlib.context import CryptContext

from app.config import settings

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

UTC = timezone.utc


def hash_password(password: str) -> str:
    """
    Хешує наданий пароль.

    :param password: Пароль до хешу.
    :type password: str

    :returns: Зашифрований пароль.
    :rtype: str
    """
    return pwd_context.hash(password)


def verify_password(plain_password: str, hashed_password: str) -> bool:
    """
    Перевіряє, чи збігається відкритий пароль з хешованим.

    :param plain_password: Простий пароль для підтвердження.
```

```
    :type plain_password: str

    :param hashed_password: Хешований пароль для порівняння.
    :type hashed_password: str

    :returns: True, якщо відкритий пароль збігається з хешованим, False у протилежному випадку.
    :rtype: bool
    """
    return pwd_context.verify(plain_password, hashed_password)


def is_authenticated(user, password: str) -> bool:
    """
    Перевіряє чи юзер автентифікований користуач через БД.
    :param user: Обьєкт користувача
    :param password: Пароль користувача
    :return: bool Автентифікований чи ні
    """
    if not user or not user.hashed_password:
        return False
    if not verify_password(password, user.hashed_password):
        return False
    return True


def to_timestamp(date: datetime | None) -> int | None:
    """
    Перетворення об'єкта часу даних у мітку часу Unix.

    :param date: Об'єкт дата-час для перетворення
    :type date: datetime | None

    :returns: Мітка часу Unix, що відповідає вхідному об'єкту,
            або None, якщо вхідне значення None
    :rtype: int | None
    """
    # Якщо дата не None, встановіть часовий пояс UTC,
    # в іншому випадку залиште дані як є
    date = date.replace(tzinfo=timezone.utc) if date else date
    return int(date.timestamp()) if date else None


def utc_now():
    """
        Повертає поточний час у форматі UTC без інформації про часовий пояс.
    :returns: Об'єкт часу даних, що представляє поточний час у форматі UTC без інформації про часовий пояс.
    """
    return datetime.now(UTC).replace(tzinfo=None)
```

```
def is_protected_username(username: str):
    """
    Перевірити, чи є задане ім'я користувача у списку захищених імен користувачів.

    Ця функція перевіряє, чи належить надане ім'я користувача до набору
    зарезервовані або системні імена користувачів, які не повинні використовуватися
звичайними користувачами.
    Він перевіряє заздалегідь визначений список імен користувачів, який включає загальні
    адміністративні, технічні та сервісні назви.

    :param username: Ім'я користувача для перевірки
    :type username: str

    :return: True, якщо ім'я користувача захищено, False в іншому випадку.
    :rtype: bool
    """
    usernames = [
        ["admin", "blog", "dev", "ftp", "mail", "pop", "pop3", "imap", "smtp"],
        ["stage", "stats", "status", "www", "beta", "about", "access"],
        ["account", "accounts", "add", "address", "adm"],
        ["administration", "adult", "advertising", "affiliate", "affiliates"],
        ["ajax", "analytics", "android", "anon", "anonymous", "api"],
        ["app", "apps", "archive", "atom", "auth", "authentication"],
        ["avatar", "backup", "banner", "banners", "bin", "billing", "blog"],
        ["blogs", "board", "bot", "bots", "business", "chat"],
        ["cache", "cadastro", "calendar", "campaign", "careers"],
        ["cgi", "client", "cliente", "code", "comercial", "compare", "config"],
        ["connect", "contact", "contest", "create", "code", "compras"],
        ["css", "dashboard", "data", "db", "design", "delete"],
        ["demo", "design", "designer", "dev", "devel", "dir"],
        ["directory", "doc", "docs", "domain", "download", "downloads"],
        ["edit", "editor", "email", "ecommerce", "forum", "forums"],
        ["faq", "favorite", "feed", "feedback", "flog", "follow"],
        ["file", "files", "free", "ftp", "gadget", "gadgets"],
        ["games", "guest", "group", "groups", "help", "home", "homepage"],
        ["host", "hosting", "hostname", "html", "http", "httpd"],
        ["https", "hpg", "info", "information", "image", "img", "images"],
        ["imap", "index", "invite", "intranet", "indice", "ipad", "iphone"],
        ["irc", "java", "javascript", "job", "jobs", "js"],
        ["knowledgebase", "log", "login", "logs", "logout", "list", "lists"],
        ["mail", "mail1", "mail2", "mail3", "mail4", "mail5"],
        ["mailer", "mailing", "mx", "manager", "marketing"],
        ["master", "me", "media", "message", "microblog", "microblogs"],
        ["mine", "mp3", "msg", "msn", "mysql", "messenger"],
        ["mob", "mobile", "movie", "movies", "music", "musicas"],
        ["my", "name", "named", "net", "network", "new"],
        ["news", "newsletter", "nick", "nickname", "notes", "noticias"],
        ["ns", "ns1", "ns2", "ns3", "ns4", "ns5", "ns6", "ns7", "ns8"],
        ["ns9", "ns10", "old", "online", "operator", "order", "orders"],
        ["page", "pager", "pages", "panel", "password", "perl", "pic", "pics"],
        ["photo", "photos", "photoalbum", "php", "plugin", "plugins", "pop"],
        ["pop3", "post", "postmaster", "postfix", "posts"],
```

```
        ["profile", "project", "projects", "promo", "pub", "public", "python"],
        ["random", "register", "registration", "root", "ruby", "rss"],
        ["sale", "sales", "sample", "samples", "script", "scripts", "secure"],
        ["send", "service", "shop", "sql", "signup", "signin", "search"],
        ["security", "settings", "setting", "setup", "site"],
        ["sites", "sitemap", "smtp", "soporte", "ssh", "stage", "staging"],
        ["start", "subscribe", "subdomain", "suporte", "support", "stat"],
        ["static", "stats", "status", "store", "stores", "system"],
        ["tablet", "tablets", "tech", "telnet", "test", "test1", "test2"],
        ["test3", "teste", "tests", "theme", "themes", "tmp"],
        ["todo", "task", "tasks", "tools", "tv", "talk", "update", "upload"],
        ["url", "user", "username", "usuario", "usage", "vendas"],
        ["video", "videos", "visitor", "win", "ww", "www"],
        ["www1", "www2", "www3", "www4", "www5", "www6", "www7"],
        ["wwww", "wws", "wwws", "web", "webmail", "website", "websites"],
        ["webmaster", "workshop", "xxx", "xpg", "you"],
    ]

    usernames = list(set(item for sublist in usernames for item in sublist))

    return username.lower() in usernames
```

app/config.py

```
import os
from pathlib import Path
from pydantic import BaseModel
from pydantic_settings import BaseSettings
from typing import Optional
from dotenv import load_dotenv


BASE_DIR = Path(__file__).resolve().parent

load_dotenv()


class Config(BaseModel):
    DB_CONFIG: str = os.getenv(
        "DB_CONFIG",

"postgresql+asyncpg://{DB_USER}:{DB_PASSWORD}@{DB_HOST}/{DB_NAME}".format
(
            DB_USER=os.getenv("DB_USER"),
            DB_PASSWORD=os.getenv("DB_PASSWORD"),
            DB_HOST=os.getenv("DB_HOST"),
            DB_NAME=os.getenv("DB_NAME"),
        ),
    ),
    TEST_DB_CONFIG: str = os.getenv(
        "TEST_DB_CONFIG",
```

```
"postgresql+asyncpg://{TEST_DB_USER}:{TEST_DB_PASSWORD}@{TEST_DB_HOST}/{
TEST_DB_NAME}".format(
        TEST_DB_USER=os.getenv("TEST_DB_USER"),
        TEST_DB_PASSWORD=os.getenv("TEST_DB_PASSWORD"),
        TEST_DB_HOST=os.getenv("TEST_DB_HOST"),
        TEST_DB_NAME=os.getenv("TEST_DB_NAME"),
    ),
  )


config = Config()


class AuthJWT(BaseModel):
  private_key_path: Path = BASE_DIR / "certs" / "jwt-private.pem"
  public_key_path: Path = BASE_DIR / "certs" / "jwt-public.pem"

  access_token_algorithm: str = os.getenv("ACCESS_TOKEN_ALGORITHM", "RS256")
  access_token_secret_key: str = os.getenv("ACCESS_TOKEN_SECRET_KEY")
  access_token_expire_minutes: int = int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES",
30))
  refresh_token_expire_days: int = int(os.getenv("REFRESH_TOKEN_EXPIRE_DAYS", 15))


auth_jwt_config = AuthJWT()


class Settings(BaseSettings):
  database_url: str = "postgresql+asyncpg://postgres:252525@localhost:5432/fast_poetry"
  database_config: Config = config
  echo_sql: bool = True
  test: bool = False
  project_name: str = "full_fast_api"

  oath_token_secret: Optional[str] = os.getenv("OATH_TOKEN_SECRET_KEY")
  auth_jwt: AuthJWT = auth_jwt_config

  log_level: str = "DEBUG"


settings = Settings()
```

app/constants.py

```
TOKEN_TYPE_FIELD = "type"
ACCESS_TOKEN_TYPE = "access"
REFRESH_TOKEN_TYPE = "refresh"
```

app/database.py

```
from typing import AsyncIterator
import contextlib
```

```
from sqlalchemy.ext.asyncio import (
    create_async_engine,
    async_sessionmaker,
    AsyncConnection,
    AsyncSession,
    AsyncEngine,
)


class DatabaseSessionManager:
    """
    Клас для керування асинхронними сесіями бази даних через SQLAlchemy.

    Забезпечує ініціалізацію підключення до бази, створення сесій, транзакційне з'єднання,
    та коректне закриття підключень. Працює як синглтон.
    """

    def __init__(self):
        self._sessionmaker: async_sessionmaker | None = None
        self._engine: AsyncEngine | None = None

    def init(self, host: str, echo: bool = False):
        """
        Ініціалізувати двигун бази даних та створити асинхронну фабрику сесій.

        :param host: Рядок підключення до бази даних (наприклад: 'postgresql+asyncpg://...').
        :param echo: Увімкнути SQL-логування у stdout (для дебагу).
        :raises: Exception, якщо вже ініціалізовано повторно.
        """

        self._engine = create_async_engine(host, echo=echo)

        self._sessionmaker = async_sessionmaker(
            autocommit=False,
            expire_on_commit=False,
            bind=self._engine,
        )

    async def close(self):
        """
        Коректно закрити підключення до бази даних та видалити фабрику сесій.

        :raises Exception: Якщо метод викликано до ініціалізації.
        """

        if self._engine is None:
            raise Exception("DatabaseSessionManager is not initialized")

        await self._engine.dispose()

        self._sessionmaker = None
        self._engine = None
```

```
    @contextlib.asynccontextmanager
    async def connect(self) -> AsyncIterator[AsyncConnection]:
        """
        Контекстний менеджер для створення низькорівневого транзакційного з'єднання з базою.

        :yield: Об'єкт AsyncConnection для ручного виконання SQL.
        :raises Exception: Якщо `DatabaseSessionManager` не ініціалізовано.
        """

        if self._engine is None:
            raise Exception("DatabaseSessionManager is not initialized")

        async with self._engine.begin() as connection:
            try:
                yield connection
            except Exception:
                await connection.rollback()
                raise

    @contextlib.asynccontextmanager
    async def session(self) -> AsyncIterator[AsyncSession]:
        """
        Контекстний менеджер для створення асинхронної сесії бази даних.

        Гарантує автоматичний `rollback` у випадку виключення і `close` сесії після використання.

        :yield: Об'єкт `AsyncSession` для взаємодії з ORM.
        :raises Exception: Якщо фабрика сесій не ініціалізована.
        """

        if self._sessionmaker is None:
            raise Exception("DatabaseSessionManager is not initialized")

        session = self._sessionmaker()

        try:
            yield session
        except Exception:
            await session.rollback()
            raise
        finally:
            await session.close()


sessionmanager = DatabaseSessionManager()


async def get_db_session():
    """
```

```
    FastAPI-залежність для надання сесії бази даних у кожному запиті.

    Забезпечує автоматичне відкриття та закриття сесії у scope-запиту FastAPI.

    :yield: `AsyncSession`, яку можна використовувати в ендпоінтах або сервісах.
    """

    async with sessionmanager.session() as session:
        yield session

app/errors.py
from fastapi import HTTPException, status


class Abort(Exception):
    def __init__(self, scope: str, message: str):
        self.scope = scope
        self.message = message


class TokenInvalidError(Exception):
    """Custom exception for invalid tokens."""

    def __init__(self, message: str, token_id: str = None, reason: str = None):
        super().__init__(message)
        self.token_id = token_id
        self.reason = reason

    def __str__(self):
        base_message = super().__str__()
        if self.token_id or self.reason:
            return f"{base_message} (Token ID: {self.token_id}, Reason: {self.reason})"
        return base_message


credentials_exception = HTTPException(
    status_code=status.HTTP_401_UNAUTHORIZED,
    detail="Could not validate credentials",
    headers={"WWW-Authenticate": "Bearer"},
)

app/main.py
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

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

root_logger = logging.getLogger("root_logger")


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

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

    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:**
- Впроваджено механізми:
  - 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"
    )
    app.add_middleware(SessionMiddleware, secret_key="some-random-string")

    @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)},
            )

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

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

    return app


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

tests/auth/test_login.py

```
async def test_login(client, register_user):
    login_data = {
        "email": "testuser@example.com",
        "password": "testpassword",
    }

    response = client.post("/auth/login", json=login_data)
    json_response = response.json()

    assert response.status_code == 200
    assert "access_token" in json_response
    assert "refresh_token" in json_response
```

tests/auth/test_signup.py

```
import pytest
import sys
import os
from pathlib import Path
from sqlalchemy import select

from app.models.user import User as DB_User
from app.models.auth import AuthToken


# Тест для регистрации пользователя
async def test_signup(client, test_session):
    signup_data = {
        "email": "testuser@example.com",
        "password": "testpassword",
        "username": "testuser"
    }

    # Отправляем запрос на регистрацию
    response = client.post("/auth/registration", json=signup_data)
    assert response.status_code == 200, f"Response: {response.json()}"

    # Проверяем созданного пользователя в базе данных
    user = await test_session.scalar(
        select(DB_User).filter(DB_User.email == "testuser@example.com")
```

```
    )
    refresh_token = await test_session.scalar(
        select(AuthToken).filter(AuthToken.user_id == user.id)
    )

    assert user.email is not None
    assert refresh_token is not None


async def test_signup_unavailable_username(client):
    signup_data = {
        "email": "testuser@example.com",
        "password": "testpassword",
        "username": "xxx"
    }

    response = client.post("/auth/registration", json=signup_data)

    assert response.status_code == 422


async def test_signup_existed_username(client, register_user):
    signup_data = {
        "email": "testuser2@example.com",
        "password": "testpassword",
        "username": "testuser"
    }

    response = client.post("/auth/registration", json=signup_data)

    assert response.status_code == 400

    json_response = response.json()
    assert "Username already exists" in json_response["detail"]


async def test_signup_existed_email(client, register_user):
    signup_data = {
        "email": "testuser@example.com",
        "password": "testpassword",
        "username": "testuser2"
    }

    response = client.post("/auth/registration", json=signup_data)

    assert response.status_code == 400

    json_response = response.json()
    assert "Email already exists" in json_response["detail"]
```

tests/conftest.py

```
import pytest
```

```
import sys
import asyncio
from pathlib import Path
from pytest_postgresql import factories
from pytest_postgresql.janitor import DatabaseJanitor
from contextlib import ExitStack
from fastapi.testclient import TestClient
from datetime import timedelta

from app.services.auth import new_token

sys.path.append(str(Path(__file__).resolve().parent.parent))


from app.models.base import Base
from app.schemas.auth import Signup, LoginArgs
from app.main import init_app
from app.database import sessionmanager, get_db_session
from app.models.user import User
from app.utils.auth import hash_password, utc_now


@pytest.fixture(autouse=True)
def app():
    with ExitStack():
        yield init_app(init_db=False)


# @pytest.fixture
# async def client(app):
#     async with TestClient(app) as client:
#         yield client

@pytest.fixture
async def client(app):
    return TestClient(app)

test_db = factories.postgresql_proc(
    port=None,
    dbname="test_fast_poetry",
)


@pytest.fixture(scope="session")
def event_loop(request):
    """
    Create an isolated event loop for the session scope.

    This fixture provides an isolated loop for test that require
    asynchronous execution. The loop is created once per test session to
    ensure test isolation and to prevent conflicts with the global event loop.
```

```
    :arg request: A pytest request object, not used in this func but
                  necessary for the fixture definition.

    :return:
        Yields:
            An instance of asyncio event loop
    """
    loop = asyncio.get_event_loop_policy().new_event_loop()
    yield loop
    loop.close()


@pytest.fixture(scope="session", autouse=True)
async def connection_test(test_db, event_loop):
    with DatabaseJanitor(
            host=test_db.host,
            port=test_db.port,
            user=test_db.user,
            dbname=test_db.dbname,
            password=test_db.password,
            version='16',
    ):
        connection_str =
f"postgresql+psycopg://{test_db.user}:{test_db.password}@{test_db.host}:{test_db.port}/{test_
db.dbname}"
        sessionmanager.init(connection_str)
        yield
        await sessionmanager.close()


@pytest.fixture(scope="function", autouse=True)
async def create_tables(connection_test):
    async with sessionmanager.connect() as connection:
        await connection.run_sync(Base.metadata.drop_all)
        await connection.run_sync(Base.metadata.create_all)


@pytest.fixture(scope="function", autouse=True)
async def session_override(app, connection_test):
    async def get_db_override():
        async with sessionmanager.session() as session:
            yield session

    app.dependency_overrides[get_db_session] = get_db_override


@pytest.fixture
async def test_session():
    async with sessionmanager.session() as session:
        yield session
```

```
@pytest.fixture
async def admin_user(test_session):
    admin = User(
        email="admin@example.com",
        username="adminuser",
        hashed_password=hash_password("password"),
        role="admin",
        created=utc_now(),
        activation_token=new_token(),
        activation_expire=utc_now() + timedelta(hours=3)
    )

    test_session.add(admin)
    await test_session.commit()
    await test_session.refresh(admin)
    return admin


@pytest.fixture
async def register_user(client):
    signup_data = Signup(
        email="testuser@example.com",
        password="testpassword",
        username="testuser"
    ).model_dump()

    response = client.post("/auth/registration", json=signup_data)
    assert response.status_code == 200


@pytest.fixture
async def register_current_user(
        client,
        email: str = "testadmin@example.com",
        password: str = "password",
        username: str = "testuser"
):
    signup_data = Signup(
        email=email,
        password=password,
        username=username
    ).dict()

    response = client.post("/auth/registration", json=signup_data)
    assert response.status_code == 200


@pytest.fixture
async def login_admin_user(client, admin_user):
    login_data = LoginArgs(
        password="password",
        email=admin_user.email
```

```
    ).model_dump()

    response = client.post("/auth/login", json=login_data)
    assert response.status_code == 200
    json_response = response.json()
    assert "access_token" in json_response
    assert "refresh_token" in json_response
    return json_response["access_token"]
```

```
.gitignore
__pycache__/
*.py[cod]
*$py.class

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
*.egg-info/
.installed.cfg
*.egg

# IDE files
.idea/
.vscode/
```

```
alembic.ini
# Загальна, єдина конфігурація бази даних.

[alembic]
# шлях до скриптів міграції
script_location = alembic

# шаблон, який використовується для генерації назв файлів міграції; значення за
замовчуванням: %%(rev)s_%%(slug)s
# Відкоментуйте рядок нижче, якщо ви хочете, щоб до файлів додавалися дата і час
# file_template = %%(year)d_%%(month).2d_%%(day).2d_%%(hour).2d%%(minute).2d-
%%(rev)s_%%(slug)s

# sys.path, буде додано до sys.path, якщо він присутній.
# за замовчуванням вказує на поточну робочу директорію.
prepend_sys_path = .
```

```
# часовий пояс для відображення дати у файлі міграції
# а також ім'я файлу.
# Якщо вказано, потрібна бібліотека python>=3.9 або backports.zoneinfo.
# Усі необхідні пакунки можна встановити, додавши `alembic[tz]` до вимог pip
# строкове значення передається в ZoneInfo()
# залиште порожнім для місцевого часу
# timezone =

# максимальна довжина символів, які можна застосувати до
# "slug" field
# truncate_slug_length = 40

# встановлюється у значення «true», щоб запустити середовище під час
# команду 'revision', незалежно від автогенерації
# revision_environment = false

# встановлюється в 'true', щоб дозволити файли .pyc та .pyo без
# вихідний .py-файл, який буде виявлено як ревізії в
# versions/ directory
# sourceless = false

# специфікацію розташування версії; за замовчуванням це значення
# до alembic/версій.  При використанні декількох версій
# початкові ревізії слід вказувати за допомогою параметра --version-path.
# Роздільник шляху, що використовується тут, має бути роздільником, визначеним у параметрі «version_path_separator» нижче.
# version_locations = %(here)s/bar:%(here)s/bat:alembic/versions

# роздільник шляху до версії; як зазначено вище, це символ, який використовується для розділення
# version_locations. За замовчуванням у нових файлах alembic.ini використовується «os», який використовує os.pathsep.
# Якщо цей ключ опустити повністю, буде повернуто до успадкованої поведінки розбиття на пробіли та/або коми.
# Допустимими значеннями для version_path_separator є
#
# version_path_separator = :
# version_path_separator = ;
# version_path_separator = space
version_path_separator = os  # Використовуйте os.pathsep. Конфігурація за замовчуванням, яка використовується для нових проектів.

# встановлено у значення 'true' для рекурсивного пошуку вихідних файлів
# у кожному каталозі «version_locations»
# нове в Alembic версії 1.10
# recursive_version_locations = false

# вихідне кодування, яке використовується при створенні файлів ревізій
# написані з script.py.mako
# output_encoding = utf-8

# sqlalchemy.url = driver://user:pass@localhost/dbname
```

```
[post_write_hooks]
# post_write_hooks визначає скрипти або функції Python, які запускаються
# у новостворених сценаріях ревізій.  Зверніться до документації для отримання
додаткової
# деталі та приклади

# форматувати з використанням «чорного» - використати бігун console_scripts, проти
«чорної» точки входу
# hooks = black
# black.type = console_scripts
# black.entrypoint = black
# black.options = -l 79 REVISION_SCRIPT_FILENAME

# lint зі спробами виправити за допомогою «йоржика» - використовуємо бігун exec,
виконуємо двійковий файл
# hooks = ruff
# ruff.type = exec
# ruff.executable = %(here)s/.venv/bin/ruff
# ruff.options = --fix REVISION_SCRIPT_FILENAME

# Конфігурація ведення журналу
[loggers]
keys = root,sqlalchemy,alembic

[handlers]
keys = console

[formatters]
keys = generic

[logger_root]
level = WARN
handlers = console
qualname =

[logger_sqlalchemy]
level = WARN
handlers =
qualname = sqlalchemy.engine

[logger_alembic]
level = INFO
handlers =
qualname = alembic

[handler_console]
class = StreamHandler
args = (sys.stderr,)
level = NOTSET
formatter = generic
[formatter_generic]
```

```
format = %(levelname)-5.5s [%(name)s] %(message)s
datefmt = %H:%M:%S
```

ДОДАТОК В

Демонстраційний матеріал

Харківський національний університет радіоелектроники

Кафедра Кітар

Кваліфікаційна робота на тему:

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

Виконав:

Студент групи АКТСІ-21-2

Андреєв Антон Сергійович

Актуальність роботи

– Автоматизація роботи приладобудівного виробництва є критично важливими для приладобудівного виробництва.

– Програмного забезпечення дозволяють створювати гнучкі, надійні та масштабовані рішення.

– Рішення із використання ШІ зменшують людський фактор і підвищують автоматизацію.

– Передбачені функції системи які є актуальні на сьогодні.

Мета, Об’єкт та Предмет розробки

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

– Об’єкт розробки: автоматизація роботи приладобудівного виробництва.

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

Завдання:

– провести аналіз технологій штучного інтелекту та аналізу даних;

– розробити структурну схему макету;

– вибір технолгій та моделі ШІ;

– інтеграція авторизації та LLM моделі;

– проведення тестування.

Приклади автоматизації мого API

- Автоматичне роспізнавання мови
- Константна конструйована відповідь від моделі
- Автоматичне продовження сессії

Обрані технології для створення застосунку

- Python
- FastAPI
- PostgreSQL
- Sqlalchemy
- GPT-4o-mini

Модулі програми

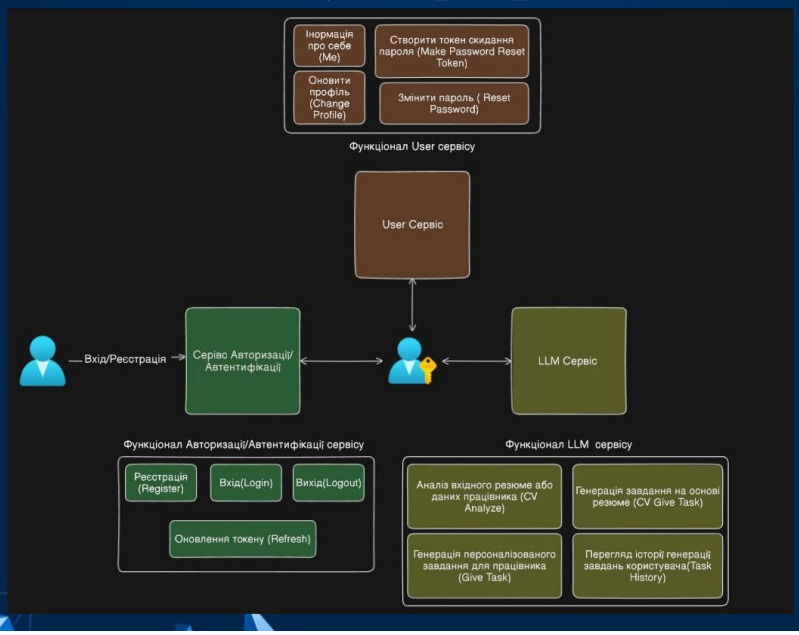
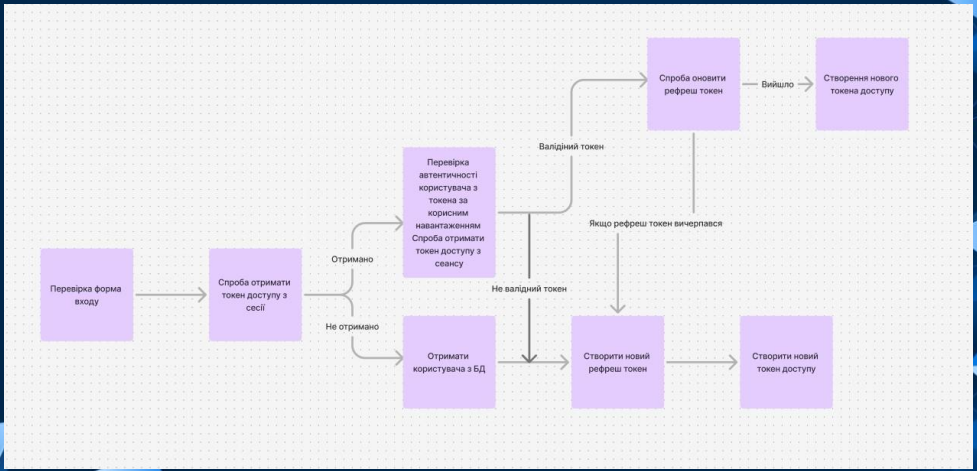
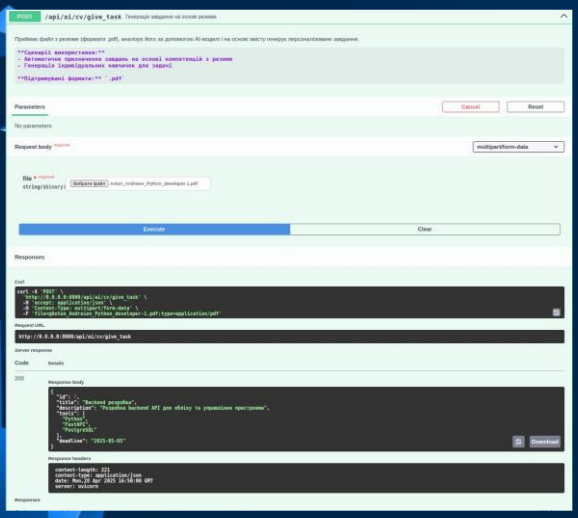
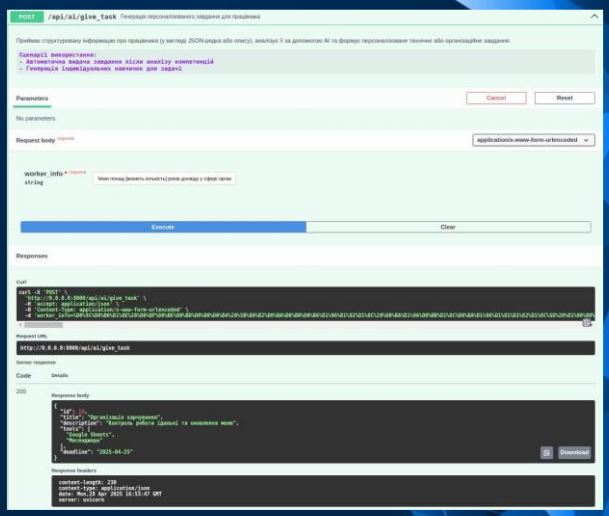


Схема Auth системи



Приклади генерації завдань





Висновки

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.

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 505400.015 ПЗ	Пояснювальна записка	А4, 100 с.
		Додаткові матеріали	
2		Апробація результатів	А4, 3 с.
		кваліфікаційної роботи	
3		Код програми	А4, 64 с.
4		Демонстраційний матеріал	А4, 10 с.

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК. 505400.015 ПЗ			
Розроб.		Андреєв А.С..			Розроблення програмного забезпечення для аналізу вхідної інформації робітника приладобудівного виробництва для видачі завдань на виконання	Літера	Аркуш	Аркушів
Перевір.		Бронніков А.І.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Стародубцев М.Г.						
Затв.		Невлюдов І.Ш.						