Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни «Основи WEB - технологій»

Тема: «Створення власного API на Python (FastAPI)»

Перевірив:
Доц. Голубєв Л. П.

Виконав:
студент групи ІМ-22
Балахон Михайло
Варіант 11

Київ - 2025

Завдання.

Створити web-додаток з використанням технології FastAPI, який буде працювати з інформацією відповідно вашому варіанту (Додаток 1. Таблиця 1)

Варіант 11. Спроектувати базу даних технологічних карт: деталь, вид обробки, тривалість обробки.

Хід роботи:

1. Підготовка проекту

1.1. Створення структури проекту

```
mkdir lab6

cd lab6

mkdir static
```

1.2. Створення віртуального середовища

```
python -m venv venv

source venv/bin/activate  # На Windows: venv\Scripts\activate
```

1.3. Встановлення залежностей

```
pip install fastapi uvicorn[standard] sqlalchemy pydantic python-dotenv
python-multipart
```

Встановлені пакети:

- `fastapi` (0.104.1) - сучасний веб-фреймворк для створення API
- `uvicorn` (0.24.0) - ASGI сервер для запуску FastAPI
- `sqlalchemy` (2.0.23) - ORM для роботи з базами даних
- `pydantic` (2.5.2) - валідація даних та створення схем

- `python-dotenv` (1.0.0) - робота зі змінними середовища
- `python-multipart` (0.0.6) - підтримка multipart форм

2. Налаштування бази даних

2.1. Файл database.py

```
from sqlalchemy import create_engine

from sqlalchemy.ext.declarative import declarative_base

from sqlalchemy.orm import sessionmaker



DATABASE_URL = "sqlite:///./technical_cards.db"



engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()



def get_db():

    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()
```

2.2. Переваги SQLite для навчального проекту

- Не потребує окремого серверу БД
- Легке розгортання та переносимість
- Достатня функціональність для CRUD операцій
- Підтримка транзакцій

3. Створення моделей даних

3.1. SQLAlchemy модель (models.py)

```
class ProcessingType(str, enum.Enum):

    TURNING = "Токарна"

    MILLING = "Фрезерна"

    DRILLING = "Свердлильна"

    GRINDING = "Шліфувальна"

    WELDING = "Зварювальна"

    ASSEMBLY = "Складальна"

    PAINTING = "Фарбування"

    THERMAL = "Термічна"



class TechnicalCard(Base):

    __tablename__ = "technical_cards"



    id = Column(Integer, primary_key=True, index=True)

    detail_name = Column(String, nullable=False, index=True)

    processing_type = Column(Enum(ProcessingType), nullable=False, index=True)

    processing_duration = Column(Integer, nullable=False)

    created_at = Column(DateTime(timezone=True), server_default=func.now())

    updated_at = Column(DateTime(timezone=True), onupdate=func.now())
```

3.2. Pydantic схеми (schemas.py)

```
class TechnicalCardBase(BaseModel):

    detail_name: str = Field(..., min_length=1, max_length=200)
```

```
    processing_type: ProcessingType

    processing_duration: int = Field(..., gt=0)


class TechnicalCardCreate(TechnicalCardBase):

    pass


class TechnicalCard(TechnicalCardBase):

    id: int

    created_at: datetime

    updated_at: Optional[datetime]


    model_config = ConfigDict(from_attributes=True)
```

4. Реалізація CRUD операцій

4.1. Функції роботи з БД (crud.py)

CREATE - Створення технічної карти:

```
def create_technical_card(db: Session, card: schemas.TechnicalCardCreate):

    db_card = models.TechnicalCard(**card.model_dump())

    db.add(db_card)

    db.commit()

    db.refresh(db_card)

    return db_card
```

READ - Отримання карт з фільтрацією:

```
def get_technical_cards(db: Session, skip: int = 0, limit: int = 100,
```

```
                        filters: Optional[schemas.TechnicalCardFilter] = None):
    query = db.query(models.TechnicalCard)

    if filters:
        if filters.processing_type:
            query = query.filter(models.TechnicalCard.processing_type ==
filters.processing_type)
        if filters.min_duration is not None:
            query = query.filter(models.TechnicalCard.processing_duration >=
filters.min_duration)
        # інші фільтри...

    return query.offset(skip).limit(limit).all()
```

UPDATE - Оновлення карти:

```
def update_technical_card(db: Session, card_id: int, card_update:
schemas.TechnicalCardUpdate):
    db_card = get_technical_card(db, card_id)
    if db_card:
        update_data = card_update.model_dump(exclude_unset=True)
        for field, value in update_data.items():
            setattr(db_card, field, value)
        db.commit()
        db.refresh(db_card)
    return db_card
```

DELETE - Видалення карти:

```
def delete_technical_card(db: Session, card_id: int):
    db_card = get_technical_card(db, card_id)
    if db_card:
        db.delete(db_card)
        db.commit()
    return db_card
```

5. Створення API endpoints

5.1. Основні маршрути (main.py)

```
@app.get("/technical-cards", response_model=List[schemas.TechnicalCard])
def read_technical_cards(
    skip: int = Query(0, ge=0),
    limit: int = Query(100, ge=1, le=1000),
    processing_type: Optional[models.ProcessingType] = None,
    min_duration: Optional[int] = Query(None, ge=0),
    max_duration: Optional[int] = Query(None, ge=0),
    detail_name_contains: Optional[str] = None,
    db: Session = Depends(get_db)
):
    filters = schemas.TechnicalCardFilter(
        processing_type=processing_type,
        min_duration=min_duration,
        max_duration=max_duration,
        detail_name_contains=detail_name_contains
    )
    cards = crud.get_technical_cards(db, skip=skip, limit=limit, filters=filters)
```

```
    return cards
```

5.2. Статистичні endpoints

```
@app.get("/stats", response_model=schemas.GeneralStats)
def get_statistics(db: Session = Depends(get_db)):
    return crud.get_general_stats(db)

@app.get("/stats/processing-types", response_model=List[schemas.ProcessingStats])
def get_processing_statistics(db: Session = Depends(get_db)):
    return crud.get_processing_stats(db)
```

6. HTML інтерфейс

6.1. Структура інтерфейсу

- **Статистика:** відображення загальних показників
- **Форма:** додавання/редагування технічних карт
- **Фільтри:** пошук за різними критеріями
- **Таблиця:** відображення всіх карт з можливістю дій

6.2. JavaScript функціональність

```
// Завантаження технічних карт
async function loadTechnicalCards(filters = {}) {
    const params = new URLSearchParams();
    if (filters.processing_type) params.append('processing_type',
filters.processing_type);
    // інші параметри...

    const response = await fetch(`/technical-cards?${params}`);
    const cards = await response.json();
```

```
    displayTechnicalCards(cards);
}

// CRUD операції через AJAX
async function editCard(id) {
    const response = await fetch(`/technical-cards/${id}`);
    const card = await response.json();
    // заповнення форми...
}

async function deleteCard(id) {
    if (!confirm('Ви впевнені?')) return;
    await fetch(`/technical-cards/${id}`, { method: 'DELETE' });
    loadTechnicalCards();
}
```

7. Фільтрація та статистика

7.1. Реалізація фільтрів

- **За типом обробки:** випадаючий список з enum значеннями
- **За тривалістю:** мінімальне та максимальне значення
- **За назвою деталі:** текстовий пошук з підтримкою часткового співпадіння

7.2. Статистичні показники

- Загальна кількість технічних карт
- Сумарний час обробки всіх деталей
- Середній час обробки
- Розподіл по видах обробки з деталізацією

8. Автоматична документація API

8.1. Swagger UI

FastAPI автоматично генерує інтерактивну документацію:

- Доступна за адресою `/docs`
- Можливість тестування всіх endpoints
- Відображення схем даних
- Приклади запитів та відповідей

8.2. ReDoc

Альтернативна документація:

- Доступна за адресою `/redoc`
- Більш структурований вигляд
- Зручна для друку

9. Особливості реалізації

9.1. Валідація даних

- Pydantic автоматично валідує вхідні дані
- Field валідатори для перевірки обмежень
- Інформативні повідомлення про помилки

9.2. Dependency Injection

```
def get_db():

    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()



# Використання
```

```
@app.get("/technical-cards")

def read_cards(db: Session = Depends(get_db)):

    # автоматичне управління сесією БД
```

9.3. Обробка помилок

```
@app.get("/technical-cards/{card_id}")

def read_technical_card(card_id: int, db: Session = Depends(get_db)):

    db_card = crud.get_technical_card(db, card_id=card_id)

    if db_card is None:

        raise HTTPException(status_code=404, detail="Технічна карта не знайдена")

    return db_card
```

10. Тестування функціональності

10.1. Тестування через Swagger UI

1. Створення технічних карт різних типів
2. Перевірка фільтрації за всіма параметрами
3. Оновлення існуючих записів
4. Видалення та перевірка каскадних операцій

10.2. Тестування через HTML інтерфейс

1. Заповнення форми та створення карт
2. Редагування через кнопки в таблиці
3. Застосування фільтрів
4. Перегляд статистики в реальному часі

10.3. Тестування через curl

```
# Створення карти

curl -X POST "http://localhost:8000/technical-cards" \

  -H "Content-Type: application/json" \
```

```
  -d '{"detail_name": "Вал", "processing_type": "TURNING", "processing_duration": 45}'


# Фільтрація

curl "http://localhost:8000/technical-cards?processing_type=MILLING&min_duration=30"
```

Структура проекту

```
lab6/

├── main.py              # Головний файл FastAPI додатку

├── models.py            # SQLAlchemy моделі даних

├── schemas.py           # Pydantic схеми валідації

├── database.py          # Налаштування підключення до БД

├── crud.py              # CRUD операції з БД

├── requirements.txt     # Python залежності

├── .env                 # Змінні середовища

├── technical_cards.db   # SQLite база даних (створюється автоматично)

├── static/              # Статичні файли

|   ├── index.html       # HTML інтерфейс

|   ├── style.css        # CSS стилі

|   └── script.js        # JavaScript логіка

├── .gitignore           # Ігноровані файли

└── README.md            # Документація проекту
```

Результати роботи

Створено повнофункціональний веб-додаток з наступними можливостями:

1. **RESTful API** з повним набором CRUD операцій
2. **Фільтрація даних** за множинними критеріями
3. **Статистичний модуль** з агрегацією даних
4. **HTML інтерфейс** для зручної роботи з API
5. **Автоматична документація** через Swagger UI та ReDoc
6. **Валідація даних** на всіх рівнях додатку
7. **База даних** з підтримкою індексів та timestamps

Скріншоти роботи додатку:

- Головна сторінка з формою та статистикою
- Таблиця технічних карт з фільтрами
- Swagger UI документація
- ReDoc документація
- Приклади API відповідей

Переваги FastAPI

1. **Висока продуктивність:** базується на Starlette та Pydantic
2. **Автоматична документація:** генерація OpenAPI (Swagger)
3. **Сучасний Python:** використання type hints
4. **Валідація даних:** автоматична перевірка вхідних/вихідних даних
5. **Асинхронність:** підтримка async/await
6. **Dependency Injection:** зручне управління залежностями

Порівняння з іншими фреймворками

Характеристика	FastAPI	Flask	Django

Швидкість	Дуже висока	Середня	Середня
Документація API	Автоматична	Потребує додатків	Потребує додатків
Type hints	Вбудовано	Опціонально	Опціонально
Async підтримка	Нативна	Обмежена	Є (3.0+)
Крива навчання	Середня	Низька	Висока

Висновки

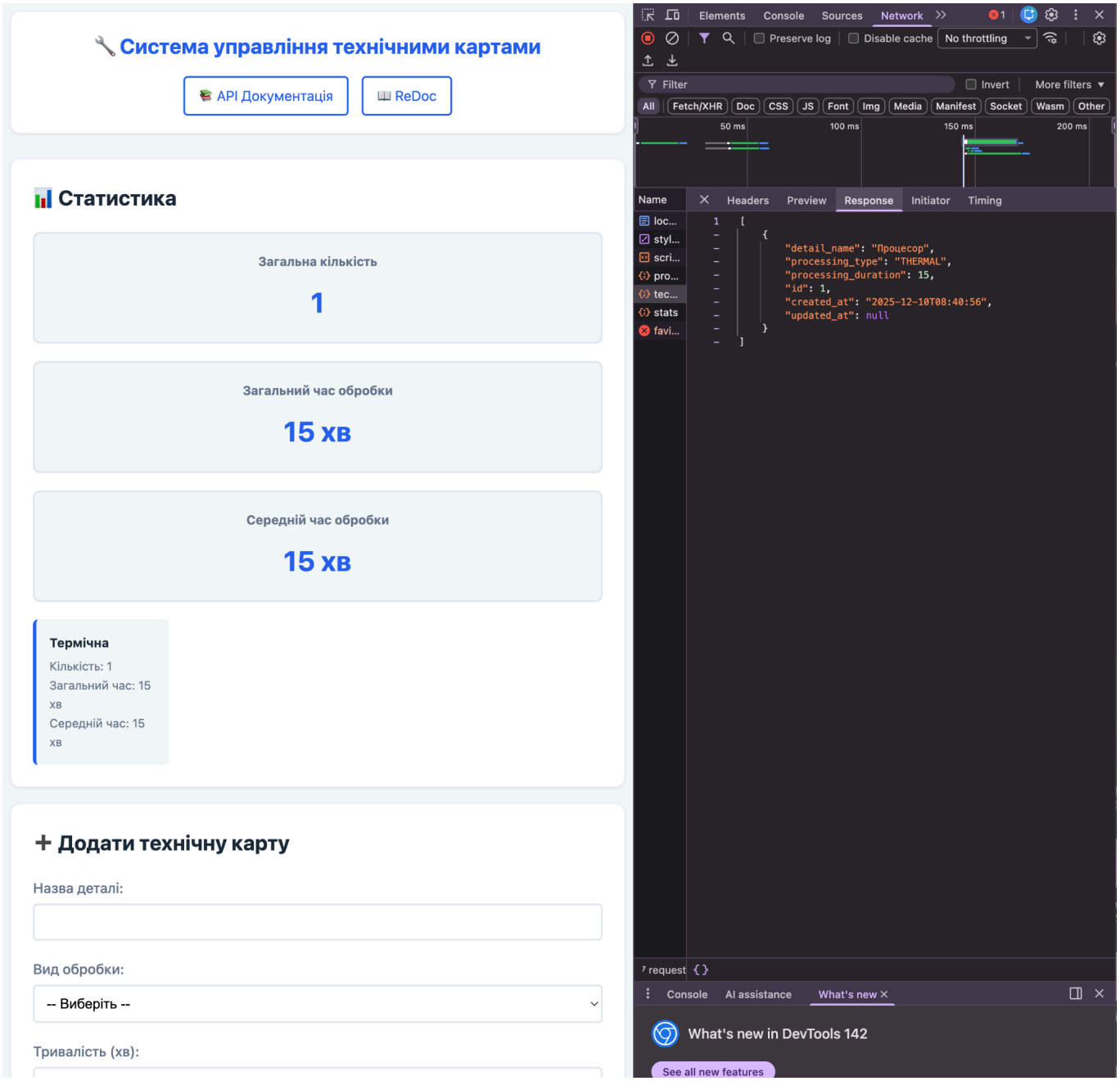
В ході виконання лабораторної роботи:

- Освоєно FastAPI - сучасний Python веб-фреймворк
- Реалізовано повний CRUD функціонал для технічних карт
- Створено систему фільтрації та статистики
- Впроваджено автоматичну валідацію даних через Pydantic
- Використано SQLAlchemy ORM для роботи з БД
- Створено зручний HTML інтерфейс для роботи з API
- Отримано досвід роботи з автоматичною документацією API

FastAPI показав себе як потужний та зручний фреймворк для створення сучасних веб-додатків. Автоматична генерація документації, вбудована валідація та висока продуктивність роблять його відмінним вибором для розробки API.

Проект демонструє практичне застосування сучасних Python технологій для створення професійних веб-додатків з мінімальною кількістю коду та максимальною функціональністю.

Скріншоти:


🔧 Система управління технічними картами
API Документація
ReDoc
📊 Статистика
Загальна кількість
1
Загальний час обробки
15 хв
Середній час обробки
15 хв
Термічна
Кількість: 1
Загальний час: 15 хв
Середній час: 15 хв
+ Додати технічну карту
Назва деталі:
Вид обробки:
-- Виберіть --
Тривалість (хв):
Elements
Console
Sources
Network
Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
Socket
Wasm
Other
50 ms
100 ms
150 ms
200 ms
Name
Headers
Preview
Response
Initiator
Timing
[
{
"detail_name": "Процесор",
"processing_type": "THERMAL",
"processing_duration": 15,
"id": 1,
"created_at": "2025-12-10T08:40:56",
"updated_at": null
}
]
Console
AI assistance
What's new
What's new in DevTools 142
See all new features