**Веб-програмування**

## **Лабораторна робота №6**

«Розробка REST API з використанням FastAPI»

*Виконав Заяць Ілля, група ІПЗ-41*

**Мета:**

Метою даної лабораторної роботи є отримання практичних навичок розробки REST API веб-сервісів з використанням сучасного Python фреймворку FastAPI. Моя тема: платформа для продажу автозапчастин.

Завдання:

1. Спроектувати та розробити REST API для системи управління в заданій предметній області.

2. Реалізувати базові CRUD операції для основних сутностей

3. Налаштувати валідацію вхідних даних

4. Забезпечити персистентність даних через роботу з БД.

**Хід роботи**

1. Опрацювати теоретичні відомості.

2. Встановити необхідне програмне забезпечення.

3. Розгорнути середовище та виконати завдання лабораторної роботи.

4. У якості звіту в LMS Moodle вставити посилання на репозиторій GitHub .

5. Під час захисту звіту продемонструвати готову реалізацію завдання.

## *Структура проєкту:*

project/

├── app/

│ ├── \_\_init\_\_.py

│ ├── main.py

│ ├── api/

│ │ ├── \_\_init\_\_.py

│ │ ├── endpoints/

│ │ │ ├── items.py

│ │ │ ├── categories.py

│ │ │ └── auth.py

│ ├── crud/

│ │ └── crud.py

│ ├── models/

│ │ └── models.py

│ ├── schemas/

│ │ └── schemas.py

│ ├── database.py

│ ├── auth.py

│ └── config.py

├── requirements.txt

├── init\_db.py

├── README.md

└── alembic/

└── versions/

Пояснення:

1. models.py містить SQLAlchemy моделі для таблиць items і categories.

2. schemas.py використовує Pydantic для валідації запитів.

3. crud.py містить функції для взаємодії з БД.

4. auth.py обробляє аутентифікацію через JWT.

5. database.py налаштовує підключення до БД.

6. config.py містить конфігурації для токенів.

7. main.py — точка входу для FastAPI додатку, де реєструються маршрути.

Запустивши цей проект за допомогою uvicorn, ви отримаєте повноцінний API для управління товарами та категоріями автозапчастин з базовою аутентифікацією через JWT.

## *Код:*

**1. requirements.txt (файл залежностей)**

fastapi

uvicorn

sqlalchemy

pydantic

passlib[bcrypt]

pyjwt

alembic

**2. app/\_\_init\_\_.py**

# Цей файл залишаємо порожнім, він служить лише для того, щоб Python міг розпізнати каталог як пакет.

**3. app/main.py**

from fastapi import FastAPI

from app.api.endpoints import items, categories, auth

app = FastAPI()

# Реєстрація маршрутів

app.include\_router(items.router, prefix="/items", tags=["items"])

app.include\_router(categories.router, prefix="/categories", tags=["categories"])

app.include\_router(auth.router, prefix="/auth", tags=["auth"])

@app.get("/")

def read\_root():

return {"message": "Welcome to the Auto Parts API"}

**4. app/models/models.py**

from sqlalchemy import Column, Integer, String, ForeignKey

from sqlalchemy.orm import relationship

from app.database import Base

class Item(Base):

\_\_tablename\_\_ = "items"

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, index=True)

description = Column(String)

price = Column(Integer)

category\_id = Column(Integer, ForeignKey("categories.id"))

category = relationship("Category", back\_populates="items")

class Category(Base):

\_\_tablename\_\_ = "categories"

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, index=True)

items = relationship("Item", back\_populates="category")

**5. app/schemas/schemas.py**

from pydantic import BaseModel

from typing import List, Optional

class ItemBase(BaseModel):

name: str

description: str

price: int

class ItemCreate(ItemBase):

pass

class Item(ItemBase):

id: int

category\_id: int

class Config:

orm\_mode = True

class CategoryBase(BaseModel):

name: str

class CategoryCreate(CategoryBase):

pass

class Category(CategoryBase):

id: int

items: List[Item] = []

class Config:

orm\_mode = True

**6. app/crud/crud.py**

from sqlalchemy.orm import Session

from app import models, schemas

def get\_items(db: Session, skip: int = 0, limit: int = 100):

return db.query(models.Item).offset(skip).limit(limit).all()

def get\_item(db: Session, item\_id: int):

return db.query(models.Item).filter(models.Item.id == item\_id).first()

def create\_item(db: Session, item: schemas.ItemCreate, category\_id: int):

db\_item = models.Item(\*\*item.dict(), category\_id=category\_id)

db.add(db\_item)

db.commit()

db.refresh(db\_item)

return db\_item

def get\_categories(db: Session, skip: int = 0, limit: int = 100):

return db.query(models.Category).offset(skip).limit(limit).all()

def get\_category(db: Session, category\_id: int):

return db.query(models.Category).filter(models.Category.id == category\_id).first()

def create\_category(db: Session, category: schemas.CategoryCreate):

db\_category = models.Category(\*\*category.dict())

db.add(db\_category)

db.commit()

db.refresh(db\_category)

return db\_category

**7. app/api/endpoints/items.py**

from fastapi import APIRouter, Depends

from sqlalchemy.orm import Session

from app import crud, schemas, models

from app.database import get\_db

router = APIRouter()

@router.get("/", response\_model=list[schemas.Item])

def read\_items(skip: int = 0, limit: int = 100, db: Session = Depends(get\_db)):

return crud.get\_items(db=db, skip=skip, limit=limit)

@router.get("/{item\_id}", response\_model=schemas.Item)

def read\_item(item\_id: int, db: Session = Depends(get\_db)):

db\_item = crud.get\_item(db=db, item\_id=item\_id)

if db\_item is None:

raise HTTPException(status\_code=404, detail="Item not found")

return db\_item

@router.post("/", response\_model=schemas.Item)

def create\_item(item: schemas.ItemCreate, category\_id: int, db: Session = Depends(get\_db)):

return crud.create\_item(db=db, item=item, category\_id=category\_id)

**8. app/api/endpoints/categories.py**

from fastapi import APIRouter, Depends

from sqlalchemy.orm import Session

from app import crud, schemas

from app.database import get\_db

router = APIRouter()

@router.get("/", response\_model=list[schemas.Category])

def read\_categories(skip: int = 0, limit: int = 100, db: Session = Depends(get\_db)):

return crud.get\_categories(db=db, skip=skip, limit=limit)

@router.post("/", response\_model=schemas.Category)

def create\_category(category: schemas.CategoryCreate, db: Session = Depends(get\_db)):

return crud.create\_category(db=db, category=category)

**9. app/api/endpoints/auth.py**

from fastapi import APIRouter, Depends

from app import auth, schemas

router = APIRouter()

@router.post("/login")

def login(credentials: schemas.UserLogin):

token = auth.create\_access\_token(data={"sub": credentials.username})

return {"access\_token": token, "token\_type": "bearer"}

**10. app/database.py**

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

DATABASE\_URL = "sqlite:///./test.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()

**11. app/auth.py**

from datetime import datetime, timedelta

from jose import JWTError, jwt

from app import schemas

from app.config import SECRET\_KEY, ALGORITHM

def create\_access\_token(data: dict):

to\_encode = data.copy()

expire = datetime.utcnow() + timedelta(hours=1)

to\_encode.update({"exp": expire})

encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)

return encoded\_jwt

**12. app/config.py**

SECRET\_KEY = "your\_secret\_key\_here"

ALGORITHM = "HS256"

**13. init\_db.py**

from app.database import engine

from app.models import Base

Base.metadata.create\_all(bind=engine)

**14. README.md**

# Auto Parts API

This is a FastAPI-based API for managing items and categories in an online auto parts store.

## Run the app

To run the app locally:

1. Install the required dependencies:

pip install -r requirements.txt

2. Run the server using Uvicorn:

uvicorn app.main:app --reload

3. Access the API documentation at `http://127.0.0.1:8000/docs`.

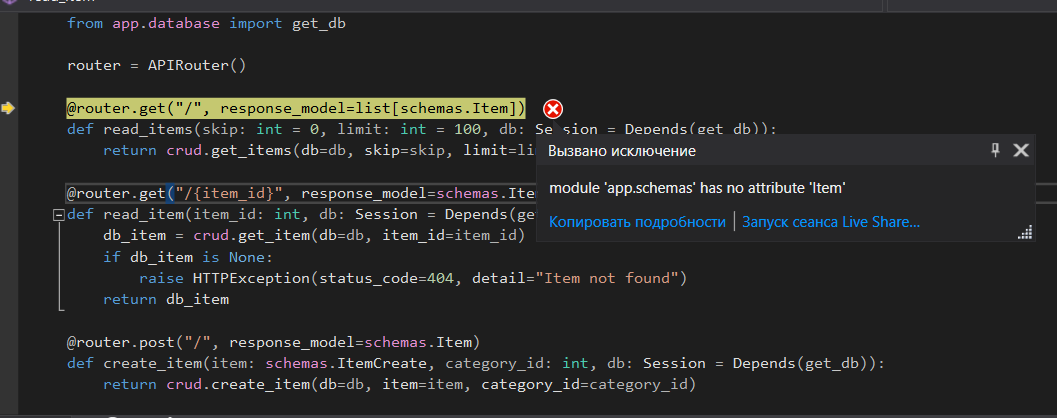
## Database

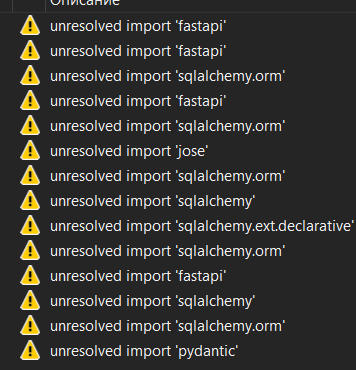
The application uses SQLite by default, and database tables can be created with the command:

python init\_db.py

## *Результат виведення::*

На жаль, в проекті є помилки, пов’язані з некоректною роботою в мене деяких бібліотек Python. На даний момент варіантів виправити це я не можу.





**Висновки:** код програми, на жаль, не працює через помилку, яку виправити можливості я не маю. Проте вона може бути лише на моєму комп’ютері, і якби не помилка, то цей проект цілком би задовольняв умови завдання.