МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Дисциплина «Проектный практикум»

Отчёт по проектному практикуму

Web-backend для канбан-доски

Выполнил:

Студент группы БВТ2402

Шпильчин А. В.

Москва, 2025 г.

<https://github.com/zachem7/kanban>

**main.py**

from fastapi import FastAPI

from .database import Base, engine

from .routers import users, projects, columns, tasks

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

app = FastAPI(title="Kanban Backend (modular)")

app.include\_router(users.router)

app.include\_router(projects.router)

app.include\_router(columns.router)

app.include\_router(tasks.router)

**database.py**

from sqlalchemy import create\_engine, MetaData

from sqlalchemy.orm import sessionmaker, scoped\_session, declarative\_base

DATABASE\_URL = "sqlite:///./kanban.db"

engine = create\_engine(DATABASE\_URL, connect\_args={"check\_same\_thread": False})

metadata = MetaData()

SessionLocal = scoped\_session(sessionmaker(autocommit=False, autoflush=False, bind=engine))

Base = declarative\_base(metadata=metadata)

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

**models.py**

from datetime import datetime

from sqlalchemy import Column, DateTime, ForeignKey, Integer, String, Text, Table

from sqlalchemy.orm import relationship

from .database import Base

# Ассоциативная таблица "многие ко многим" между пользователями и проектами

user\_project\_table = Table(

"user\_project",

Base.metadata,

Column("user\_id", Integer, ForeignKey("users.id"), primary\_key=True),

Column("project\_id", Integer, ForeignKey("projects.id"), primary\_key=True),

)

class User(Base):

\_\_tablename\_\_ = "users"

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

username = Column(String, unique=True, index=True, nullable=False)

projects = relationship("Project", secondary=user\_project\_table, back\_populates="users")

class Project(Base):

\_\_tablename\_\_ = "projects"

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

title = Column(String, nullable=False)

description = Column(Text, default="")

columns = relationship("KanbanColumn", back\_populates="project", cascade="all, delete-orphan")

users = relationship("User", secondary=user\_project\_table, back\_populates="projects")

class KanbanColumn(Base):

\_\_tablename\_\_ = "columns"

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

title = Column(String, nullable=False)

position = Column(Integer, default=0)

project\_id = Column(Integer, ForeignKey("projects.id", ondelete="CASCADE"), nullable=False)

project = relationship("Project", back\_populates="columns")

tasks = relationship("Task", back\_populates="column", cascade="all, delete-orphan")

class Task(Base):

\_\_tablename\_\_ = "tasks"

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

title = Column(String, nullable=False)

description = Column(Text, default="")

priority = Column(Integer, default=0)

assignee\_id = Column(Integer, ForeignKey("users.id"), nullable=True)

created\_at = Column(DateTime, default=datetime.utcnow)

updated\_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

column\_id = Column(Integer, ForeignKey("columns.id", ondelete="CASCADE"), nullable=False)

column = relationship("KanbanColumn", back\_populates="tasks")

assignee = relationship("User")

logs = relationship("TaskLog", back\_populates="task", cascade="all, delete-orphan")

class TaskLog(Base):

\_\_tablename\_\_ = "task\_logs"

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

task\_id = Column(Integer, ForeignKey("tasks.id", ondelete="CASCADE"), nullable=False)

message = Column(Text, nullable=False)

created\_at = Column(DateTime, default=datetime.utcnow)

task = relationship("Task", back\_populates="logs")

**schemas.py**

from datetime import datetime

from typing import List, Optional

from pydantic import BaseModel, Field

class UserCreate(BaseModel):

username: str = Field(..., min\_length=1)

class UserRead(BaseModel):

id: int

username: str

class Config:

orm\_mode = True

class ProjectCreate(BaseModel):

title: str

description: Optional[str] = ""

class ProjectRead(BaseModel):

id: int

title: str

description: Optional[str]

users: List[UserRead] = []

class Config:

orm\_mode = True

class ColumnCreate(BaseModel):

title: str

position: Optional[int] = 0

project\_id: int

class ColumnRead(BaseModel):

id: int

title: str

position: int

project\_id: int

class Config:

orm\_mode = True

class TaskCreate(BaseModel):

title: str

description: Optional[str] = ""

priority: Optional[int] = 0

assignee\_id: Optional[int] = None

column\_id: int

class TaskUpdate(BaseModel):

title: Optional[str] = None

description: Optional[str] = None

priority: Optional[int] = None

assignee\_id: Optional[int] = None

column\_id: Optional[int] = None

class TaskRead(BaseModel):

id: int

title: str

description: Optional[str]

priority: int

assignee: Optional[UserRead] = None

column\_id: int

created\_at: datetime

updated\_at: datetime

class Config:

orm\_mode = True

class TaskLogRead(BaseModel):

id: int

task\_id: int

message: str

created\_at: datetime

class Config:

orm\_mode = True

**utils.py**

from sqlalchemy.orm import Session

from .models import Task, TaskLog

def create\_task\_log(db: Session, task: Task, message: str):

log = TaskLog(task\_id=task.id, message=message)

db.add(log)

db.commit()

db.refresh(log)

return log

**columns.py**

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from .. import models, schemas

from ..database import get\_db

router = APIRouter(prefix="/columns", tags=["Columns"])

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

def create\_column(payload: schemas.ColumnCreate, db: Session = Depends(get\_db)):

project = db.query(models.Project).get(payload.project\_id)

if not project:

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

col = models.KanbanColumn(

title=payload.title,

position=payload.position,

project=project

)

db.add(col)

db.commit()

db.refresh(col)

return col

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

def get\_column(column\_id: int, db: Session = Depends(get\_db)):

col = db.query(models.KanbanColumn).get(column\_id)

if not col:

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

return col

@router.get("/project/{project\_id}", response\_model=list[schemas.ColumnRead])

def get\_columns\_by\_project(project\_id: int, db: Session = Depends(get\_db)):

"""Получить все колонки конкретного проекта"""

return db.query(models.KanbanColumn).filter(models.KanbanColumn.project\_id == project\_id).all()

**projects.py**

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from .. import models, schemas

from ..database import get\_db

router = APIRouter(prefix="/projects", tags=["Projects"])

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

def create\_project(payload: schemas.ProjectCreate, db: Session = Depends(get\_db)):

project = models.Project(title=payload.title, description=payload.description)

db.add(project)

db.commit()

db.refresh(project)

return project

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

def list\_projects(db: Session = Depends(get\_db)):

return db.query(models.Project).all()

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

def get\_project(project\_id: int, db: Session = Depends(get\_db)):

project = db.query(models.Project).get(project\_id)

if not project:

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

return project

@router.post("/{project\_id}/users/{user\_id}", response\_model=schemas.ProjectRead)

def add\_user\_to\_project(project\_id: int, user\_id: int, db: Session = Depends(get\_db)):

project = db.query(models.Project).get(project\_id)

user = db.query(models.User).get(user\_id)

if not project or not user:

raise HTTPException(status\_code=404, detail="project or user not found")

if user in project.users:

raise HTTPException(status\_code=400, detail="user already in project")

project.users.append(user)

db.commit()

db.refresh(project)

return project

@router.delete("/{project\_id}/users/{user\_id}", response\_model=schemas.ProjectRead)

def remove\_user\_from\_project(project\_id: int, user\_id: int, db: Session = Depends(get\_db)):

project = db.query(models.Project).get(project\_id)

user = db.query(models.User).get(user\_id)

if not project or not user:

raise HTTPException(status\_code=404, detail="project or user not found")

if user not in project.users:

raise HTTPException(status\_code=400, detail="user not in project")

project.users.remove(user)

db.commit()

db.refresh(project)

return project

**tasks.py**

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from typing import List

from .. import models, schemas

from ..database import get\_db

from ..utils import create\_task\_log

router = APIRouter(prefix="/tasks", tags=["Tasks"])

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

def create\_task(payload: schemas.TaskCreate, db: Session = Depends(get\_db)):

col = db.query(models.KanbanColumn).get(payload.column\_id)

if not col:

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

assignee = None

if payload.assignee\_id:

assignee = db.query(models.User).get(payload.assignee\_id)

if not assignee:

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

task = models.Task(

title=payload.title,

description=payload.description,

priority=payload.priority,

assignee=assignee,

column=col,

)

db.add(task)

db.commit()

db.refresh(task)

create\_task\_log(db, task, f"Task created: '{task.title}'")

return task

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

def get\_task(task\_id: int, db: Session = Depends(get\_db)):

task = db.query(models.Task).get(task\_id)

if not task:

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

return task

@router.get("/{task\_id}/logs", response\_model=List[schemas.TaskLogRead])

def get\_task\_logs(task\_id: int, db: Session = Depends(get\_db)):

return db.query(models.TaskLog).filter(models.TaskLog.task\_id == task\_id).all()

**users.py**

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from .. import models, schemas

from ..database import get\_db

router = APIRouter(prefix="/users", tags=["Users"])

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

def create\_user(payload: schemas.UserCreate, db: Session = Depends(get\_db)):

existing = db.query(models.User).filter(models.User.username == payload.username).first()

if existing:

raise HTTPException(status\_code=400, detail="username already exists")

user = models.User(username=payload.username)

db.add(user)

db.commit()

db.refresh(user)

return user

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

def get\_user(user\_id: int, db: Session = Depends(get\_db)):

user = db.query(models.User).get(user\_id)

if not user:

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

return user