МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Лабораторная работа №5

По дисциплине “**Современные платформы программирования**”

Выполнил: студент группы ПО-11  
Сымоник И.А.  
Проверил:

Козик И. Д.

Брест 2025

**Цель:** приобрести практические навыки разработки API и баз данных.

**Вариант 18**

**Задание.**

1. Реализовать базу данных из не менее 5 таблиц на заданную тематику. При реализации продумать типизацию полей и внешние ключи в таблицах;

2. Визуализировать разработанную БД с помощью схемы, на которой отображены все таблицы и связи между ними

3. На языке Python с использованием SQLAlchemy реализовать подключение к БД;

4. Реализовать основные операции с данными (выборку, добавление, удаление, модификацию);

5. Для каждой реализованной операции с использованием FastAPI реализовать отдельный эндпойнт;

База данных Туристические агентства

Схема БД:

Изображение выглядит как текст, снимок экрана, Параллельный, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Код программы:

Api/\_\_init\_\_.py

from fastapi import APIRouter

from .agency import router as agency\_router

from .tour import router as tour\_router

from .client import router as client\_router

from .employee import router as employee\_router

from .booking import router as booking\_router

router = APIRouter()

router.include\_router(agency\_router)

router.include\_router(tour\_router)

router.include\_router(client\_router)

router.include\_router(employee\_router)

router.include\_router(booking\_router)

\_\_all\_\_ = ["router"]

Api/agency.py

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from typing import List

from database import get\_db

from schemas.agency import Agency, AgencyCreate

from crud.agency import (

    get\_agency, get\_agencies, create\_agency,

    update\_agency, delete\_agency

)

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

@router.post("/", response\_model=Agency, status\_code=201)

def create\_agency\_endpoint(agency: AgencyCreate, db: Session = Depends(get\_db)):

    return create\_agency(db=db, agency=agency)

@router.get("/", response\_model=List[Agency])

def read\_agencies\_endpoint(

    skip: int = 0,

    limit: int = 100,

    db: Session = Depends(get\_db)

):

    return get\_agencies(db, skip=skip, limit=limit)

@router.get("/{agency\_id}", response\_model=Agency)

def read\_agency\_endpoint(agency\_id: int, db: Session = Depends(get\_db)):

    db\_agency = get\_agency(db, agency\_id=agency\_id)

    if db\_agency is None:

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

    return db\_agency

@router.put("/{agency\_id}", response\_model=Agency)

def update\_agency\_endpoint(

    agency\_id: int,

    agency: AgencyCreate,

    db: Session = Depends(get\_db)

):

    db\_agency = update\_agency(db, agency\_id=agency\_id, agency\_data=agency)

    if db\_agency is None:

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

    return db\_agency

@router.delete("/{agency\_id}")

def delete\_agency\_endpoint(agency\_id: int, db: Session = Depends(get\_db)):

    success = delete\_agency(db, agency\_id=agency\_id)

    if not success:

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

    return {"message": "Agency deleted successfully"}

api/booking.py

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from typing import List

from datetime import date

from database import get\_db

from schemas.booking import Booking, BookingCreate

from crud.booking import (

    get\_booking, get\_bookings, get\_bookings\_by\_client,

    get\_bookings\_by\_employee, get\_upcoming\_bookings,

    create\_booking, update\_booking, delete\_booking

)

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

@router.post("/", response\_model=Booking, status\_code=201)

def create\_booking\_endpoint(

    booking: BookingCreate,

    db: Session = Depends(get\_db)

):

    return create\_booking(db=db, booking=booking)

@router.get("/", response\_model=List[Booking])

def read\_bookings\_endpoint(

    skip: int = 0,

    limit: int = 100,

    db: Session = Depends(get\_db)

):

    return get\_bookings(db, skip=skip, limit=limit)

@router.get("/upcoming", response\_model=List[Booking])

def read\_upcoming\_bookings\_endpoint(db: Session = Depends(get\_db)):

    bookings = get\_upcoming\_bookings(db)

    if not bookings:

        raise HTTPException(

            status\_code=404,

            detail="No upcoming bookings found"

        )

    return bookings

@router.get("/client/{client\_id}", response\_model=List[Booking])

def read\_bookings\_by\_client\_endpoint(

    client\_id: int,

    db: Session = Depends(get\_db)

):

    bookings = get\_bookings\_by\_client(db, client\_id=client\_id)

    if not bookings:

        raise HTTPException(

            status\_code=404,

            detail="No bookings found for this client"

        )

    return bookings

@router.get("/employee/{employee\_id}", response\_model=List[Booking])

def read\_bookings\_by\_employee\_endpoint(

    employee\_id: int,

    db: Session = Depends(get\_db)

):

    bookings = get\_bookings\_by\_employee(db, employee\_id=employee\_id)

    if not bookings:

        raise HTTPException(

            status\_code=404,

            detail="No bookings found for this employee"

        )

    return bookings

@router.get("/{booking\_id}", response\_model=Booking)

def read\_booking\_endpoint(

    booking\_id: int,

    db: Session = Depends(get\_db)

):

    db\_booking = get\_booking(db, booking\_id=booking\_id)

    if db\_booking is None:

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

    return db\_booking

@router.put("/{booking\_id}", response\_model=Booking)

def update\_booking\_endpoint(

    booking\_id: int,

    booking: BookingCreate,

    db: Session = Depends(get\_db)

):

    db\_booking = update\_booking(db, booking\_id=booking\_id, booking\_data=booking)

    if db\_booking is None:

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

    return db\_booking

@router.delete("/{booking\_id}")

def delete\_booking\_endpoint(

    booking\_id: int,

    db: Session = Depends(get\_db)

):

    success = delete\_booking(db, booking\_id=booking\_id)

    if not success:

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

    return {"message": "Booking deleted successfully"}

api/client.py

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from typing import List

from database import get\_db

from schemas.client import Client, ClientCreate

from crud.client import (

    get\_client, get\_client\_by\_passport, get\_clients,

    create\_client, update\_client, delete\_client

)

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

@router.post("/", response\_model=Client, status\_code=201)

def create\_client\_endpoint(client: ClientCreate, db: Session = Depends(get\_db)):

    db\_client = get\_client\_by\_passport(db, passport\_number=client.passport\_number)

    if db\_client:

        raise HTTPException(

            status\_code=400,

            detail="Client with this passport already exists"

        )

    return create\_client(db=db, client=client)

@router.get("/", response\_model=List[Client])

def read\_clients\_endpoint(

    skip: int = 0,

    limit: int = 100,

    db: Session = Depends(get\_db)

):

    return get\_clients(db, skip=skip, limit=limit)

@router.get("/{client\_id}", response\_model=Client)

def read\_client\_endpoint(client\_id: int, db: Session = Depends(get\_db)):

    db\_client = get\_client(db, client\_id=client\_id)

    if db\_client is None:

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

    return db\_client

@router.get("/by-passport/{passport\_number}", response\_model=Client)

def read\_client\_by\_passport\_endpoint(

    passport\_number: str,

    db: Session = Depends(get\_db)

):

    db\_client = get\_client\_by\_passport(db, passport\_number=passport\_number)

    if db\_client is None:

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

    return db\_client

@router.put("/{client\_id}", response\_model=Client)

def update\_client\_endpoint(

    client\_id: int,

    client: ClientCreate,

    db: Session = Depends(get\_db)

):

    db\_client = update\_client(db, client\_id=client\_id, client\_data=client)

    if db\_client is None:

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

    return db\_client

@router.delete("/{client\_id}")

def delete\_client\_endpoint(client\_id: int, db: Session = Depends(get\_db)):

    success = delete\_client(db, client\_id=client\_id)

    if not success:

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

    return {"message": "Client deleted successfully"}

api/employee.py

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from typing import List

from database import get\_db

from schemas.employee import Employee, EmployeeCreate

from crud.employee import (

    get\_employee, get\_employees, get\_employees\_by\_agency,

    create\_employee, update\_employee, delete\_employee

)

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

@router.post("/", response\_model=Employee, status\_code=201)

def create\_employee\_endpoint(

    employee: EmployeeCreate,

    db: Session = Depends(get\_db)

):

    return create\_employee(db=db, employee=employee)

@router.get("/", response\_model=List[Employee])

def read\_employees\_endpoint(

    skip: int = 0,

    limit: int = 100,

    db: Session = Depends(get\_db)

):

    return get\_employees(db, skip=skip, limit=limit)

@router.get("/agency/{agency\_id}", response\_model=List[Employee])

def read\_employees\_by\_agency\_endpoint(

    agency\_id: int,

    db: Session = Depends(get\_db)

):

    employees = get\_employees\_by\_agency(db, agency\_id=agency\_id)

    if not employees:

        raise HTTPException(

            status\_code=404,

            detail="No employees found for this agency"

        )

    return employees

@router.get("/{employee\_id}", response\_model=Employee)

def read\_employee\_endpoint(

    employee\_id: int,

    db: Session = Depends(get\_db)

):

    db\_employee = get\_employee(db, employee\_id=employee\_id)

    if db\_employee is None:

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

    return db\_employee

@router.put("/{employee\_id}", response\_model=Employee)

def update\_employee\_endpoint(

    employee\_id: int,

    employee: EmployeeCreate,

    db: Session = Depends(get\_db)

):

    db\_employee = update\_employee(db, employee\_id=employee\_id, employee\_data=employee)

    if db\_employee is None:

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

    return db\_employee

@router.delete("/{employee\_id}")

def delete\_employee\_endpoint(

    employee\_id: int,

    db: Session = Depends(get\_db)

):

    success = delete\_employee(db, employee\_id=employee\_id)

    if not success:

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

    return {"message": "Employee deleted successfully"}

api/tour.py

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from typing import List

from database import get\_db

from schemas.tour import Tour, TourCreate

from crud.tour import (

    get\_tour, get\_tours, get\_tours\_by\_agency,

    create\_tour, update\_tour, delete\_tour

)

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

@router.post("/", response\_model=Tour, status\_code=201)

def create\_tour\_endpoint(tour: TourCreate, db: Session = Depends(get\_db)):

    return create\_tour(db=db, tour=tour)

@router.get("/", response\_model=List[Tour])

def read\_tours\_endpoint(

    skip: int = 0,

    limit: int = 100,

    db: Session = Depends(get\_db)

):

    return get\_tours(db, skip=skip, limit=limit)

@router.get("/agency/{agency\_id}", response\_model=List[Tour])

def read\_tours\_by\_agency\_endpoint(

    agency\_id: int,

    db: Session = Depends(get\_db)

):

    tours = get\_tours\_by\_agency(db, agency\_id=agency\_id)

    if not tours:

        raise HTTPException(

            status\_code=404,

            detail="No tours found for this agency"

        )

    return tours

@router.get("/{tour\_id}", response\_model=Tour)

def read\_tour\_endpoint(tour\_id: int, db: Session = Depends(get\_db)):

    db\_tour = get\_tour(db, tour\_id=tour\_id)

    if db\_tour is None:

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

    return db\_tour

@router.put("/{tour\_id}", response\_model=Tour)

def update\_tour\_endpoint(

    tour\_id: int,

    tour: TourCreate,

    db: Session = Depends(get\_db)

):

    db\_tour = update\_tour(db, tour\_id=tour\_id, tour\_data=tour)

    if db\_tour is None:

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

    return db\_tour

@router.delete("/{tour\_id}")

def delete\_tour\_endpoint(tour\_id: int, db: Session = Depends(get\_db)):

    success = delete\_tour(db, tour\_id=tour\_id)

    if not success:

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

    return {"message": "Tour deleted successfully"}

crud/\_\_init\_\_.py

from .agency import (

    get\_agency, get\_agencies, create\_agency,

    update\_agency, delete\_agency

)

from .tour import (

    get\_tour, get\_tours, get\_tours\_by\_agency,

    create\_tour, update\_tour, delete\_tour

)

from .client import (

    get\_client, get\_client\_by\_passport, get\_clients,

    create\_client, update\_client, delete\_client

)

from .employee import (

    get\_employee, get\_employees, get\_employees\_by\_agency,

    create\_employee, update\_employee, delete\_employee

)

from .booking import (

    get\_booking, get\_bookings, get\_bookings\_by\_client,

    get\_bookings\_by\_employee, get\_upcoming\_bookings,

    create\_booking, update\_booking, delete\_booking

)

\_\_all\_\_ = [

    # Agency

    "get\_agency", "get\_agencies", "create\_agency",

    "update\_agency", "delete\_agency",

    # Tour

    "get\_tour", "get\_tours", "get\_tours\_by\_agency",

    "create\_tour", "update\_tour", "delete\_tour",

    # Client

    "get\_client", "get\_client\_by\_passport", "get\_clients",

    "create\_client", "update\_client", "delete\_client",

    # Employee

    "get\_employee", "get\_employees", "get\_employees\_by\_agency",

    "create\_employee", "update\_employee", "delete\_employee",

    # Booking

    "get\_booking", "get\_bookings", "get\_bookings\_by\_client",

    "get\_bookings\_by\_employee", "get\_upcoming\_bookings",

    "create\_booking", "update\_booking", "delete\_booking"

]

Crud/agency.py

from sqlalchemy.orm import Session

from models.agency import Agency

from schemas.agency import AgencyCreate

def get\_agency(db: Session, agency\_id: int):

    return db.query(Agency).filter(Agency.id == agency\_id).first()

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

    return db.query(Agency).offset(skip).limit(limit).all()

def create\_agency(db: Session, agency: AgencyCreate):

    db\_agency = Agency(\*\*agency.dict())

    db.add(db\_agency)

    db.commit()

    db.refresh(db\_agency)

    return db\_agency

def update\_agency(db: Session, agency\_id: int, agency\_data: AgencyCreate):

    db\_agency = db.query(Agency).filter(Agency.id == agency\_id).first()

    if db\_agency:

        for key, value in agency\_data.dict().items():

            setattr(db\_agency, key, value)

        db.commit()

        db.refresh(db\_agency)

    return db\_agency

def delete\_agency(db: Session, agency\_id: int):

    db\_agency = db.query(Agency).filter(Agency.id == agency\_id).first()

    if db\_agency:

        db.delete(db\_agency)

        db.commit()

    return db\_agency

crud/booking.py

from sqlalchemy.orm import Session

from models.booking import Booking

from schemas.booking import BookingCreate

from datetime import date

def get\_booking(db: Session, booking\_id: int):

    return db.query(Booking).filter(Booking.id == booking\_id).first()

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

    return db.query(Booking).offset(skip).limit(limit).all()

def get\_bookings\_by\_client(db: Session, client\_id: int):

    return db.query(Booking).filter(Booking.client\_id == client\_id).all()

def get\_bookings\_by\_employee(db: Session, employee\_id: int):

    return db.query(Booking).filter(Booking.employee\_id == employee\_id).all()

def get\_upcoming\_bookings(db: Session):

    return db.query(Booking).filter(Booking.departure\_date >= date.today()).all()

def create\_booking(db: Session, booking: BookingCreate):

    db\_booking = Booking(\*\*booking.dict())

    db.add(db\_booking)

    db.commit()

    db.refresh(db\_booking)

    return db\_booking

def update\_booking(db: Session, booking\_id: int, booking\_data: BookingCreate):

    db\_booking = db.query(Booking).filter(Booking.id == booking\_id).first()

    if db\_booking:

        for key, value in booking\_data.dict().items():

            setattr(db\_booking, key, value)

        db.commit()

        db.refresh(db\_booking)

    return db\_booking

def delete\_booking(db: Session, booking\_id: int):

    db\_booking = db.query(Booking).filter(Booking.id == booking\_id).first()

    if db\_booking:

        db.delete(db\_booking)

        db.commit()

    return db\_booking

crud/client.py

from sqlalchemy.orm import Session

from models.client import Client

from schemas.client import ClientCreate

def get\_client(db: Session, client\_id: int):

    return db.query(Client).filter(Client.id == client\_id).first()

def get\_client\_by\_passport(db: Session, passport\_number: str):

    return db.query(Client).filter(Client.passport\_number == passport\_number).first()

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

    return db.query(Client).offset(skip).limit(limit).all()

def create\_client(db: Session, client: ClientCreate):

    db\_client = Client(\*\*client.dict())

    db.add(db\_client)

    db.commit()

    db.refresh(db\_client)

    return db\_client

def update\_client(db: Session, client\_id: int, client\_data: ClientCreate):

    db\_client = db.query(Client).filter(Client.id == client\_id).first()

    if db\_client:

        for key, value in client\_data.dict().items():

            setattr(db\_client, key, value)

        db.commit()

        db.refresh(db\_client)

    return db\_client

def delete\_client(db: Session, client\_id: int):

    db\_client = db.query(Client).filter(Client.id == client\_id).first()

    if db\_client:

        db.delete(db\_client)

        db.commit()

    return db\_client

crud/employee.py

from sqlalchemy.orm import Session

from models.employee import Employee

from schemas.employee import EmployeeCreate

def get\_employee(db: Session, employee\_id: int):

    return db.query(Employee).filter(Employee.id == employee\_id).first()

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

    return db.query(Employee).offset(skip).limit(limit).all()

def get\_employees\_by\_agency(db: Session, agency\_id: int):

    return db.query(Employee).filter(Employee.agency\_id == agency\_id).all()

def create\_employee(db: Session, employee: EmployeeCreate):

    db\_employee = Employee(\*\*employee.dict())

    db.add(db\_employee)

    db.commit()

    db.refresh(db\_employee)

    return db\_employee

def update\_employee(db: Session, employee\_id: int, employee\_data: EmployeeCreate):

    db\_employee = db.query(Employee).filter(Employee.id == employee\_id).first()

    if db\_employee:

        for key, value in employee\_data.dict().items():

            setattr(db\_employee, key, value)

        db.commit()

        db.refresh(db\_employee)

    return db\_employee

def delete\_employee(db: Session, employee\_id: int):

    db\_employee = db.query(Employee).filter(Employee.id == employee\_id).first()

    if db\_employee:

        db.delete(db\_employee)

        db.commit()

    return db\_employee

curd/tour.py

from sqlalchemy.orm import Session

from models.tour import Tour

from schemas.tour import TourCreate

def get\_tour(db: Session, tour\_id: int):

    return db.query(Tour).filter(Tour.id == tour\_id).first()

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

    return db.query(Tour).offset(skip).limit(limit).all()

def get\_tours\_by\_agency(db: Session, agency\_id: int):

    return db.query(Tour).filter(Tour.agency\_id == agency\_id).all()

def create\_tour(db: Session, tour: TourCreate):

    db\_tour = Tour(\*\*tour.dict())

    db.add(db\_tour)

    db.commit()

    db.refresh(db\_tour)

    return db\_tour

def update\_tour(db: Session, tour\_id: int, tour\_data: TourCreate):

    db\_tour = db.query(Tour).filter(Tour.id == tour\_id).first()

    if db\_tour:

        for key, value in tour\_data.dict().items():

            setattr(db\_tour, key, value)

        db.commit()

        db.refresh(db\_tour)

    return db\_tour

def delete\_tour(db: Session, tour\_id: int):

    db\_tour = db.query(Tour).filter(Tour.id == tour\_id).first()

    if db\_tour:

        db.delete(db\_tour)

        db.commit()

    return db\_tour

models/agency.py

from sqlalchemy import Column, Integer, String

from sqlalchemy.orm import relationship

from database import Base

class Agency(Base):

    \_\_tablename\_\_ = 'agencies'

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

    name = Column(String(100), nullable=False)

    address = Column(String(200))

    phone = Column(String(20))

    email = Column(String(100))

    employees = relationship("Employee", back\_populates="agency")

    tours = relationship("Tour", back\_populates="agency")

    def \_\_repr\_\_(self):

        return f"<Agency(id={self.id}, name='{self.name}')>"

from pydantic import BaseModel

from typing import Optional

class AgencyBase(BaseModel):

    name: str

    address: Optional[str] = None

    phone: Optional[str] = None

    email: Optional[str] = None

class AgencyCreate(AgencyBase):

    pass

class Agency(AgencyBase):

    id: int

    class Config:

        orm\_mode = True

models/booking.py

from sqlalchemy import Column, Integer, Date, String, ForeignKey

from sqlalchemy.orm import relationship

from database import Base

from datetime import date

class Booking(Base):

    \_\_tablename\_\_ = 'bookings'

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

    booking\_date = Column(Date, default=date.today())

    departure\_date = Column(Date)

    return\_date = Column(Date)

    status = Column(String(20), default='confirmed')

    client\_id = Column(Integer, ForeignKey('clients.id'))

    tour\_id = Column(Integer, ForeignKey('tours.id'))

    employee\_id = Column(Integer, ForeignKey('employees.id'))

    client = relationship("Client", back\_populates="bookings")

    tour = relationship("Tour", back\_populates="bookings")

    employee = relationship("Employee", back\_populates="bookings")

    def \_\_repr\_\_(self):

        return f"<Booking(id={self.id}, tour\_id={self.tour\_id}, status='{self.status}')>"

from pydantic import BaseModel

from datetime import date

from typing import Optional

class BookingBase(BaseModel):

    departure\_date: date

    return\_date: date

    client\_id: int

    tour\_id: int

    employee\_id: int

    status: Optional[str] = 'confirmed'

class BookingCreate(BookingBase):

    pass

class Booking(BookingBase):

    id: int

    booking\_date: date

    class Config:

        orm\_mode = True

models/client.py

from sqlalchemy import Column, Integer, String

from sqlalchemy.orm import relationship

from database import Base

class Client(Base):

    \_\_tablename\_\_ = 'clients'

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

    first\_name = Column(String(50), nullable=False)

    last\_name = Column(String(50), nullable=False)

    passport\_number = Column(String(50), unique=True)

    phone = Column(String(20))

    email = Column(String(100))

    bookings = relationship("Booking", back\_populates="client")

    def \_\_repr\_\_(self):

        return f"<Client(id={self.id}, name='{self.first\_name} {self.last\_name}')>"

from pydantic import BaseModel

from typing import Optional

class ClientBase(BaseModel):

    first\_name: str

    last\_name: str

    passport\_number: Optional[str] = None

    phone: Optional[str] = None

    email: Optional[str] = None

class ClientCreate(ClientBase):

    pass

class Client(ClientBase):

    id: int

    class Config:

        orm\_mode = True

models/employee.py

from sqlalchemy import Column, Integer, String, ForeignKey

from sqlalchemy.orm import relationship

from database import Base

class Employee(Base):

    \_\_tablename\_\_ = 'employees'

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

    first\_name = Column(String(50), nullable=False)

    last\_name = Column(String(50), nullable=False)

    position = Column(String(50))

    phone = Column(String(20))

    email = Column(String(100))

    agency\_id = Column(Integer, ForeignKey('agencies.id'))

    agency = relationship("Agency", back\_populates="employees")

    bookings = relationship("Booking", back\_populates="employee")

    def \_\_repr\_\_(self):

        return f"<Employee(id={self.id}, name='{self.first\_name} {self.last\_name}')>"

from pydantic import BaseModel

from typing import Optional

class EmployeeBase(BaseModel):

    first\_name: str

    last\_name: str

    position: Optional[str] = None

    phone: Optional[str] = None

    email: Optional[str] = None

    agency\_id: int

class EmployeeCreate(EmployeeBase):

    pass

class Employee(EmployeeBase):

    id: int

    class Config:

        orm\_mode = True

models/tour.py

from sqlalchemy import Column, Integer, String, Float, ForeignKey

from sqlalchemy.orm import relationship

from database import Base

class Tour(Base):

    \_\_tablename\_\_ = 'tours'

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

    title = Column(String(100), nullable=False)

    description = Column(String(500))

    destination = Column(String(100), nullable=False)

    duration\_days = Column(Integer)

    price = Column(Float, nullable=False)

    agency\_id = Column(Integer, ForeignKey('agencies.id'))

    agency = relationship("Agency", back\_populates="tours")

    bookings = relationship("Booking", back\_populates="tour")

    def \_\_repr\_\_(self):

        return f"<Tour(id={self.id}, title='{self.title}', price={self.price}>"

from pydantic import BaseModel

from typing import Optional

class TourBase(BaseModel):

    title: str

    description: Optional[str] = None

    destination: str

    duration\_days: Optional[int] = None

    price: float

    agency\_id: int

class TourCreate(TourBase):

    pass

class Tour(TourBase):

    id: int

    class Config:

        orm\_mode = True

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from config import settings

# Используем settings.DATABASE\_URL (в верхнем регистре)

engine = create\_engine(settings.DATABASE\_URL)

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

Base = declarative\_base()

def get\_db():

    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()

main.py

from fastapi import FastAPI

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

import os

from dotenv import load\_dotenv

from models.agency import Agency

from models.tour import Tour

from models.client import Client

from models.employee import Employee

from models.booking import Booking

load\_dotenv()

app = FastAPI(

    title="Travel Agency API",

    description="API для управления туристическим агентством",

    version="1.0.0",

    docs\_url="/docs",

    redoc\_url="/redoc"

)

DATABASE\_URL = os.getenv("DATABASE\_URL", "sqlite:///./travel\_agency.db")

engine = create\_engine(DATABASE\_URL)

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

Base = declarative\_base()

from api.agency import router as agency\_router

from api.tour import router as tour\_router

from api.client import router as client\_router

from api.employee import router as employee\_router

from api.booking import router as booking\_router

from services.initialization import router as init\_router

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

def get\_db():

    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()

app.include\_router(agency\_router)

app.include\_router(tour\_router)

app.include\_router(client\_router)

app.include\_router(employee\_router)

app.include\_router(booking\_router)

app.include\_router(init\_router)

@app.get("/", tags=["Root"])

def read\_root():

    return {

        "message": "Добро пожаловать в API туристического агентства",

        "endpoints": {

            "agencies": "/agencies/",

            "tours": "/tours/",

            "clients": "/clients/",

            "employees": "/employees/",

            "bookings": "/bookings/",

            "init\_db": "/init-db/"

        },

        "documentation": {

            "swagger": "/docs",

            "redoc": "/redoc"

        }

    }

@app.get("/health", tags=["Utility"])

def health\_check():

    return {"status": "OK", "message": "Сервис работает нормально"}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run(app, host="0.0.0.0", port=8000)

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Вывод: приобрели практические навыки разработки API и баз данных.