Задание. Backend, Python

Необходимо реализовать асинхронное веб приложение в парадигме REST API.

Стек:

* База данных - postgresql
* sqlalchemy - для работы с базой данных
* sanic - веб фреймворк
* docker compose

Необходимо реализовать работу со следующими сущностями:

1. Пользователь
2. Администратор
3. Счет - имеет баланс, привязан к пользователю
4. Платеж(пополнение баланса) - хранит уникальный идентификатор и сумму пополнения счета пользователя

Пользователь должен иметь следующие возможности:

1. Авторизоваться по email/password
2. Получить данные о себе(id, email, full\_name)
3. Получить список своих счетов и балансов
4. Получить список своих платежей

Администратор должен иметь следующие возможности:

1. Авторизоваться по email/password
2. Получить данные о себе (id, email, full\_name)
3. Создать/Удалить/Обновить пользователя
4. Получить список пользователей и список его счетов с балансами

Для работы с платежами должен быть реализован роут эмулирующий обработку вебхука от сторонней платежной системы.

Структура json-объекта для обработки вебхука должна состоять из следующих полей:

* transaction\_id - уникальный идентификатор транзакции в “сторонней системе”
* account\_id - уникальный идентификатор счета пользователя
* user\_id - уникальный идентификатор счета пользователя
* amount - сумма пополнения счета пользователя
* signature - подпись объекта

signature должна формироваться через SHA256 хеш, для строки состоящей из конкатенации значений объекта в алфавитном порядке ключей и “секретного ключа” хранящегося в конфигурации проекта ({account\_id}{amount}{transaction\_id}{user\_id}{secret\_key}).

Пример, для secret\_key gfdmhghif38yrf9ew0jkf32:

{

  "transaction\_id": "5eae174f-7cd0-472c-bd36-35660f00132b",

  "user\_id": 1,

  "account\_id": 1,

  "amount": 100,

  "signature": "7b47e41efe564a062029da3367bde8844bea0fb049f894687cee5d57f2858bc8"

}

При обработке вебхука необходимо:

1. Проверить подпись объекта
2. Проверить существует ли у пользователя такой счет - если нет, его необходимо создать
3. Сохранить транзакцию в базе данных
4. Начислить сумму транзакции на счет пользователя

Транзакции являются уникальными, начисление суммы с одним transaction\_id должно производиться только один раз.

Для тестирования приложения в миграции должен быть создан:

1. Тестовый пользователь
2. Счет тестового пользователя
3. Тестовый администратор

Для развертывания проекта необходимо реализовать docker compose конфигурацию состоящую из сервиса postgresql и сервиса приложения.

К реализованному заданию должна прилагаться краткая инструкция по запуску проекта в двух вариантах - с использованием docker compose и без него. В инструкции также должны быть предоставлены email/password для пользователя и администратора по умолчанию созданных в миграции.

**На данный момент, структура проекта следующая:**

1. app
   1. routes
      1. \_\_init\_\_.py
      2. crud\_routes.py
      3. payment\_routes.py
      4. user\_routes.py
   2. templates
      1. forbidden.html
      2. index.html
      3. user-accounts.html
      4. user-create.html
      5. user-delete.html
      6. user-lk.html
      7. user-payments.html
      8. user-update.html
      9. users-list.html
   3. \_\_init\_\_.py
   4. config.py
   5. database.py
   6. encrypting.py
   7. main.py
   8. models.py
   9. schemas.py
2. migrations
3. alembic.ini
4. docker-compose.yml
5. Dockerfile
6. requirements.txt

**модуль main.py:**

from sanic import Sanic  
from sanic\_session import Session, RedisSessionInterface  
from sanic\_jinja2 import SanicJinja2  
from sanic.response import json, redirect, html  
from pathlib import Path  
from app.routes import user\_bp, admin\_bp, payment\_bp # Импортируем Blueprints  
from app.routes.login import login\_bp # Импортируем Blueprints  
import logging  
import aioredis  
  
# Создаем приложение Sanic  
app = Sanic("MyApp")  
logger = logging.Logger(\_\_name\_\_)  
  
# Настройка сессий (инициализируем позже!)  
session = Session()  
  
# Настройка Jinja2  
jinja = SanicJinja2(  
 app,  
 pkg\_name="app",  
 enable\_async=True, # Включаем асинхронный режим  
)  
  
  
# ⚡ Создаём подключение к Redis перед стартом сервера  
@app.before\_server\_start  
async def setup\_redis(app, \_):  
 app.ctx.redis = await aioredis.from\_url("redis://redis:6379")  
  
 # ✅ Проверяем соединение с Redis  
 try:  
 pong = await app.ctx.redis.ping()  
 if pong:  
 logger.warning("✅ Redis подключен!")  
 print("✅ Redis подключен!")  
 except Exception as e:  
 logger.warning(f"❌ Ошибка подключения к Redis: {e}")  
 print(f"❌ Ошибка подключения к Redis: {e}")  
  
 session.init\_app(app, interface=RedisSessionInterface(app.ctx.redis))  
  
  
# ⚡ Закрываем соединение с Redis при выключении сервера  
@app.after\_server\_stop  
async def close\_redis(app, \_):  
 await app.ctx.redis.close()  
  
  
# Регистрируем Blueprints  
app.blueprint(user\_bp)  
app.blueprint(admin\_bp)  
app.blueprint(payment\_bp)  
app.blueprint(login\_bp)  
  
  
# # Основной маршрут  
@app.route('/', name="index")  
async def index(request):  
 logger.warning(f"request.ctx: {request.ctx}")  
  
 # Проверяем, существует ли request.ctx.session  
 if hasattr(request.ctx, "session"):  
 logger.warning(f"request.ctx.session: {request.ctx.session}")  
 user = request.ctx.session.get("user", None)  
 else:  
 user = None  
  
 logger.warning(f"user: {user}")  
  
 with open(Path('app/templates/index.html'), 'r') as f:  
 template = f.read()  
 if user:  
 template = template.replace("{% if user %}", "").replace("{% endif %}", "")  
 template = template.replace("{{ user.full\_name }}", user['full\_name'])  
 template = template.replace("{{ user.email }}", user['email'])  
 template = template.replace("{{ user.id }}", str(user['id']))  
 else:  
 template = template.replace("{% if user %}", "").replace("{% endif %}", "")  
 template = template.replace("<p>Вы вошли как: {{ user.full\_name }} ({{ user.email }})</p>", "")  
 template = template.replace("<a href=\"/user/{{ user.id }}\"><button>Перейти в личный кабинет</button></a>",  
 "")  
 template = template.replace("<a href=\"/logout\"><button>Выйти</button></a>", "")  
  
 return html(template)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host="0.0.0.0", port=8000)

**модуль config.py:**

import os  
from dotenv import load\_dotenv  
  
load\_dotenv()  
  
class Config:  
 POSTGRES\_USER = os.getenv("POSTGRES\_USER", "postgres")  
 POSTGRES\_PASSWORD = os.getenv("POSTGRES\_PASSWORD", "postgres")  
 POSTGRES\_DB = os.getenv("POSTGRES\_DB", "postgres")  
 # POSTGRES\_HOST = os.getenv("POSTGRES\_HOST", "localhost")  
 POSTGRES\_HOST = os.getenv("POSTGRES\_HOST", "postgres")  
 POSTGRES\_PORT = os.getenv("POSTGRES\_PORT", "5432")  
 # DATABASE\_URL = f"postgresql+asyncpg://{POSTGRES\_USER}:{POSTGRES\_PASSWORD}@{POSTGRES\_HOST}:{POSTGRES\_PORT}/{POSTGRES\_DB}"  
 # DATABASE\_URL = "postgresql+asyncpg://postgres:postgres@localhost:5432/postgres"  
 REDIS\_HOST = os.getenv("REDIS\_HOST", "redis")  
 REDIS\_PORT = os.getenv("REDIS\_PORT", "6379")  
 DATABASE\_URL = "postgresql+asyncpg://postgres:postgres@postgres:5432/postgres"  
 SECRET\_KEY = os.getenv("SECRET\_KEY", "gfdmhghif38yrf9ew0jkf32")  
 JWT\_SECRET = os.getenv("JWT\_SECRET", "super-secret-key")

**модуль database.py:**

from sqlalchemy.ext.asyncio import create\_async\_engine, AsyncSession  
from sqlalchemy.orm import sessionmaker, declarative\_base  
from app.config import Config # Импортируем конфигурацию  
  
# Создаем асинхронный движок для подключения к PostgreSQL  
engine = create\_async\_engine(  
 Config.DATABASE\_URL, # URL базы данных из конфигурации  
 echo=True, # Логирование SQL-запросов (можно отключить в production)  
 future=True, # Используем новые возможности SQLAlchemy 2.0  
)  
  
# Создаем фабрику сессий  
async\_session = sessionmaker(  
 bind=engine,  
 class\_=AsyncSession,  
 expire\_on\_commit=False, # Отключаем автоматическое завершение транзакций  
)  
  
# Базовый класс для моделей  
Base = declarative\_base()  
  
  
# Функция для получения сессии  
async def get\_session() -> AsyncSession:  
 async with async\_session() as session:  
 yield session

**модуль models.py:**

from sqlalchemy import Column, Integer, String, Float, ForeignKey, DateTime  
from sqlalchemy.sql import func  
from app.database import Base # Импортируем Base из database.py  
  
  
class User(Base):  
 \_\_tablename\_\_ = "users"  
 id = Column(Integer, primary\_key=True)  
 email = Column(String, unique=True, nullable=False)  
 password = Column(String, nullable=False)  
 full\_name = Column(String, nullable=False)  
 is\_admin = Column(Integer, default=0)  
  
  
class Account(Base):  
 \_\_tablename\_\_ = "accounts"  
 id = Column(Integer, primary\_key=True)  
 user\_id = Column(Integer, ForeignKey("users.id"), nullable=False)  
 balance = Column(Float, default=0.0)  
  
  
class Payment(Base):  
 \_\_tablename\_\_ = "payments"  
 id = Column(Integer, primary\_key=True)  
 transaction\_id = Column(String, unique=True, nullable=False)  
 account\_id = Column(Integer, ForeignKey("accounts.id"), nullable=False)  
 user\_id = Column(Integer, ForeignKey("users.id"), nullable=False)  
 amount = Column(Float, nullable=False)  
 created\_at = Column(DateTime, default=func.now())

**модуль admin\_routes.py:**

from sanic import Blueprint  
from sanic.response import json  
from sanic.exceptions import NotFound  
from sqlalchemy import select  
from app.database import async\_session  
from app.models import User, Account  
  
# Создаем Blueprint для маршрутов администраторов  
admin\_bp = Blueprint("admin\_routes", url\_prefix="/admin")  
  
  
@admin\_bp.route('/<admin\_id:int>', methods=['GET'])  
async def get\_admin(request, admin\_id):  
 async with async\_session() as session:  
 result = await session.execute(select(User).where(User.id == admin\_id, User.is\_admin == 1))  
 admin = result.scalars().first()  
 if not admin:  
 raise NotFound("Admin not found")  
 return json({"id": admin.id, "email": admin.email, "full\_name": admin.full\_name})  
  
  
@admin\_bp.route('/users', methods=['GET'])  
async def get\_users(request):  
 async with async\_session() as session:  
 result = await session.execute(select(User))  
 users = result.scalars().all()  
 return json([{"id": user.id, "email": user.email, "full\_name": user.full\_name} for user in users])  
  
  
@admin\_bp.route('/users/<user\_id:int>/accounts', methods=['GET'])  
async def get\_user\_accounts(request, user\_id):  
 async with async\_session() as session:  
 result = await session.execute(select(Account).where(Account.user\_id == user\_id))  
 accounts = result.scalars().all()  
 return json([{"id": acc.id, "balance": acc.balance} for acc in accounts])

**модуль login.py:**

from sanic import Blueprint  
from sanic.response import html, redirect, json  
from sanic.exceptions import NotFound, Unauthorized  
from sanic\_session import Session  
from sqlalchemy import select  
from app.database import async\_session  
from app.models import User  
from pathlib import Path  
import logging  
  
login\_bp = Blueprint('login', url\_prefix='/')  
logger = logging.Logger(\_\_name\_\_)  
  
  
@login\_bp.get('/login')  
async def login\_get(request):  
 with open(Path('app/templates/login.html'), 'r') as f:  
 return html(f.read())  
  
  
@login\_bp.post('/login')  
async def login\_post(request):  
 email = request.form.get("email")  
 password = request.form.get("password")  
  
 logger.warning(f"login\_post(request).ctx: {request.ctx}")  
 async with async\_session() as session:  
 result = await session.execute(select(User).where(User.email == email))  
 user = result.scalars().first()  
  
 if not user or user.password != password:  
 raise Unauthorized("Invalid email or password")  
  
 request.ctx.session['user'] = {  
 "id": user.id,  
 "email": user.email,  
 "full\_name": user.full\_name,  
 "is\_admin": user.is\_admin  
 }  
 logger.warning(f"request.ctx.session: {request.ctx.session}")  
  
 # Перенаправляем на главную страницу  
 return redirect('/')  
  
  
@login\_bp.route('/logout', methods=['GET'], name="logout")  
async def logout(request):  
 # Очищаем сессию  
 if 'user' in request.ctx.session:  
 del request.ctx.session['user']  
 return redirect('/')

**модуль payment\_routes.py:**

from sanic import Blueprint  
from sanic.response import json  
from sqlalchemy import select  
from app.database import async\_session  
from app.models import Payment, Account  
from app.config import Config  
import hashlib  
  
# Создаем Blueprint для маршрутов платежей  
payment\_bp = Blueprint("payment\_routes", url\_prefix="/payment")  
  
  
@payment\_bp.route('/webhook', methods=['POST'])  
async def webhook(request):  
 data = request.json  
 signature = data.pop('signature')  
 sorted\_data = sorted(data.items(), key=lambda x: x[0])  
 concat\_values = ''.join([str(v) for k, v in sorted\_data]) + Config.SECRET\_KEY  
 expected\_signature = hashlib.sha256(concat\_values.encode()).hexdigest()  
  
 if signature != expected\_signature:  
 return json({"error": "Invalid signature"}, status=400)  
  
 async with async\_session() as session:  
 result = await session.execute(select(Account).where(Account.id == data['account\_id']))  
 account = result.scalars().first()  
 if not account:  
 account = Account(id=data['account\_id'], user\_id=data['user\_id'], balance=0.0)  
 session.add(account)  
  
 result = await session.execute(select(Payment).where(Payment.transaction\_id == data['transaction\_id']))  
 if result.scalars().first():  
 return json({"error": "Transaction already processed"}, status=400)  
  
 payment = Payment(  
 transaction\_id=data['transaction\_id'],  
 account\_id=data['account\_id'],  
 user\_id=data['user\_id'],  
 amount=data['amount']  
 )  
 session.add(payment)  
 account.balance += data['amount']  
 await session.commit()  
  
 return json({"success": True})

**модуль user\_routes.py:**

from sanic import Blueprint  
from sanic.response import json, html  
from sanic.exceptions import NotFound  
from sqlalchemy import select  
from app.models import User, Account, Payment  
from app.database import async\_session  
  
# Создаем Blueprint для маршрутов пользователей  
user\_bp = Blueprint("user\_routes", url\_prefix="/user")  
  
  
@user\_bp.route('/<user\_id:int>', methods=['GET'])  
async def get\_user(request, user\_id):  
 async with async\_session() as session:  
 result = await session.execute(select(User).where(User.id == user\_id))  
 user = result.scalars().first()  
 if not user:  
 raise NotFound("User not found")  
  
 # Генерация HTML-страницы  
 html\_content = f"""  
 <html>  
 <body>  
 <h1>Ваш id: {user.id}</h1>  
 <h1>Пароль: {user.password}</h1>  
 <h1>ФИО: {user.full\_name}</h1>  
 <br>  
 <a href="/"><button>Вернуться на стартовую страницу</button></a>  
 <a href="/user/{user.id}/accounts/"><button>Получить список своих счетов и балансов</button></a>  
 <a href="/user/{user.id}/payments/"><button>Получить список своих платежей</button></a>  
 <a href="/logout"><button>Выйти из аккаунта</button></a>  
 </body>  
 </html>  
 """  
 return html(html\_content)  
  
  
@user\_bp.route('/<user\_id:int>/accounts', methods=['GET'])  
async def get\_user\_accounts(request, user\_id):  
 async with async\_session() as session:  
 result = await session.execute(select(Account).where(Account.user\_id == user\_id))  
 accounts = result.scalars().all()  
 return json([{"id": acc.id, "balance": acc.balance} for acc in accounts])  
  
  
@user\_bp.route('/<user\_id:int>/payments', methods=['GET'])  
async def get\_user\_payments(request, user\_id):  
 async with async\_session() as session:  
 result = await session.execute(select(Payment).where(Payment.user\_id == user\_id))  
 payments = result.scalars().all()  
 return json([{"id": pay.id, "transaction\_id": pay.transaction\_id, "amount": pay.amount} for pay in payments])

**файл alembic.ini:**

[alembic]  
script\_location = migrations  
file\_template = %%(year)d\_%%(month).2d\_%%(day).2d\_%%(hour).2d%%(minute).2d-%%(rev)s\_%%(slug)s  
prepend\_sys\_path = .  
timezone = UTC  
truncate\_slug\_length = 40  
revision\_environment = false  
sourceless = false  
version\_locations = %(here)s/migrations/versions  
version\_path\_separator = os  
recursive\_version\_locations = false  
output\_encoding = utf-8  
sqlalchemy.url = postgresql://postgres:postgres@postgres:5432/postgres  
  
[post\_write\_hooks]  
hooks = ruff  
ruff.type = exec  
ruff.executable = %(here)s/.venv/bin/ruff  
ruff.options = --fix REVISION\_SCRIPT\_FILENAME  
  
# Logging configuration  
[loggers]  
keys = root,sqlalchemy,alembic  
  
[handlers]  
keys = console  
  
[formatters]  
keys = generic  
  
[logger\_root]  
level = WARNING  
handlers = console  
qualname =  
  
[logger\_sqlalchemy]  
level = WARNING  
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

**файл docker-compose.yml:**

version: "3.8"  
services:  
 postgres:  
 image: postgres:13  
 environment:  
 POSTGRES\_USER: postgres  
 POSTGRES\_PASSWORD: postgres  
 POSTGRES\_DB: postgres  
 ports:  
 - "5432:5432"  
 volumes:  
 - postgres\_data:/var/lib/postgresql/data  
 healthcheck:  
 test: [ "CMD-SHELL", "pg\_isready -U postgres -d postgres" ]  
 interval: 5s  
 timeout: 5s  
 retries: 5  
 networks:  
 - app-network  
  
 redis:  
 image: redis:7.4.2  
 restart: always  
 ports:  
 - "6379:6379"  
 volumes:  
 - redis\_data:/data  
 networks:  
 - app-network  
  
 app:  
 build: .  
 ports:  
 - "8000:8000"  
 environment:  
 POSTGRES\_HOST: postgres  
 POSTGRES\_USER: postgres  
 POSTGRES\_PASSWORD: postgres  
 POSTGRES\_DB: postgres  
 REDIS\_HOST: redis  
 depends\_on:  
 postgres:  
 condition: service\_healthy  
 redis:  
 condition: service\_started  
 networks:  
 - app-network  
  
volumes:  
 postgres\_data:  
 redis\_data:  
  
networks:  
 app-network:  
 driver: bridge

**файл Dockerfile:**

# Используем официальный образ Python  
FROM python:3.10-slim  
  
# Устанавливаем системные зависимости  
RUN apt-get update && apt-get install -y \  
 libpq-dev \  
 gcc \  
 && rm -rf /var/lib/apt/lists/\*  
  
# Устанавливаем рабочую директорию  
WORKDIR /app  
  
# Копируем зависимости  
COPY requirements.txt .  
  
# Устанавливаем зависимости  
RUN pip install --no-cache-dir -r requirements.txt  
  
# Копируем исходный код  
COPY . .  
  
# Указываем команду для запуска приложения  
CMD ["sanic", "app.main:app", "--host=0.0.0.0", "--port=8000"]

**файл requirements.txt:**

aiofiles==24.1.0  
aioredis==2.0.1  
alembic==1.14.1  
async-timeout==5.0.1  
asyncio-redis==0.16.0  
asyncpg==0.30.0  
bcrypt==4.2.1  
cffi==1.17.1  
cryptography==44.0.0  
ecdsa==0.19.0  
greenlet==3.1.1  
html5tagger==1.3.0  
httptools==0.6.4  
Jinja2==3.1.5  
Mako==1.3.8  
MarkupSafe==3.0.2  
multidict==6.1.0  
psycopg2==2.9.10  
pyasn1==0.6.1  
pycparser==2.22  
PyJWT==2.10.1  
python-dotenv==1.0.1  
python-jose==3.3.0  
PyYAML==6.0.2  
redis==5.2.1  
redis-cli==1.0.1  
rsa==4.9  
ruff==0.9.4  
sanic==24.12.0  
sanic-jinja2==2022.11.11  
sanic-jwt==1.8.0  
sanic-routing==23.12.0  
sanic-session==0.8.0  
six==1.17.0  
SQLAlchemy==2.0.37  
tracerite==1.1.1  
typing\_extensions==4.12.2  
tzdata==2025.1  
ujson==5.10.0  
websockets==14.2

**файл index.html:**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Главная страница</title>  
</head>  
<body>  
 <h1>Добро пожаловать!</h1>  
 {% if user %}  
 <p>Вы вошли как: {{ user.full\_name }} ({{ user.email }})</p>  
 <a href="/user/{{ user.id }}"><button>Перейти в личный кабинет</button></a>  
 <a href="/logout"><button>Выйти</button></a>  
 {% else %}  
 <a href="/login"><button>Войти</button></a>  
 {% endif %}  
</body>  
</html>

**файл login.html:**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Login</title>  
</head>  
<body>  
 <h1>Login</h1>  
 <form action="/login" method="POST">  
 <label for="email">Email:</label>  
 <input type="email" id="email" name="email" required>  
 <br>  
 <label for="password">Password:</label>  
 <input type="password" id="password" name="password" required>  
 <br>  
 <button type="submit">Login</button>  
  
 </form>  
</body>  
</html>

**файл user-lk.html:**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Login</title>  
</head>  
 <body>  
 <h1>Ваш id: {user.id}</h1>  
 <h1>Пароль: {user.password}</h1>  
 <h1>ФИО: {user.full\_name}</h1>  
 <br>  
 <a href="/"><button>Вернуться на стартовую страницу</button></a>  
 <a href="/user/{user.id}/accounts/"><button>Получить список своих счетов и балансов</button></a>  
 <a href="/user/{user.id}/payments/"><button>Получить список своих платежей</button></a>  
 <a href="/logout"><button>Выйти из аккаунта</button></a>  
</body>  
</html>

На данный момент, при переходе по адресу "/" происходит следующая ошибка:

app-1 | Traceback (most recent call last):

app-1 | File "handle\_request", line 100, in handle\_request

app-1 | Extend = TypeVar("Extend", type) # type: ignore

app-1 | File "/app/app/routes/login.py", line 16, in index

app-1 | user = request.ctx.session.get('user', None)

app-1 | AttributeError: 'types.SimpleNamespace' object has no attribute 'session'

Исправь это.

**Теперь модуль main имеет следующий код:**

from sanic import Sanic  
from sanic\_session import Session, RedisSessionInterface  
from sanic\_jinja2 import SanicJinja2  
from sanic.response import json, redirect, html  
from pathlib import Path  
from app.routes import user\_bp, admin\_bp, payment\_bp # Импортируем Blueprints  
from app.routes.login import login\_bp # Импортируем Blueprints  
import logging  
import aioredis  
  
# Создаем приложение Sanic  
app = Sanic("MyApp")  
logger = logging.Logger(\_\_name\_\_)  
  
# Настройка сессий (инициализируем позже!)  
session = Session()  
  
# Настройка Jinja2  
jinja = SanicJinja2(  
 app,  
 pkg\_name="app",  
 enable\_async=True, # Включаем асинхронный режим  
)  
  
  
# ⚡ Создаём подключение к Redis перед стартом сервера  
@app.before\_server\_start  
async def setup\_redis(app, \_):  
 app.ctx.redis = await aioredis.from\_url("redis://redis:6379")  
  
 # ✅ Проверяем соединение с Redis  
 try:  
 pong = await app.ctx.redis.ping()  
 if pong:  
 logger.warning("✅ Redis подключен!")  
 except Exception as e:  
 logger.warning(f"❌ Ошибка подключения к Redis: {e}")  
  
 session.init\_app(app, interface=RedisSessionInterface(app.ctx.redis))  
  
  
# ⚡ Закрываем соединение с Redis при выключении сервера  
@app.after\_server\_stop  
async def close\_redis(app, \_):  
 await app.ctx.redis.close()  
  
  
# Регистрируем Blueprints  
app.blueprint(user\_bp)  
app.blueprint(admin\_bp)  
app.blueprint(payment\_bp)  
app.blueprint(login\_bp)  
  
  
# # Основной маршрут  
@app.route('/', name="index")  
async def index(request):  
 logger.warning(f"request.ctx: {request.ctx}")  
  
 # Проверяем, существует ли request.ctx.session  
 if hasattr(request.ctx, "session"):  
 logger.warning(f"request.ctx.session: {request.ctx.session}")  
 user = request.ctx.session.get("user", None)  
 else:  
 user = None  
  
 logger.warning(f"user: {user}")  
  
 with open(Path('app/templates/index.html'), 'r') as f:  
 template = f.read()  
 if user:  
 template = template.replace("{% if user %}", "").replace("{% endif %}", "")  
 template = template.replace("{{ user.full\_name }}", user['full\_name'])  
 template = template.replace("{{ user.email }}", user['email'])  
 template = template.replace("{{ user.id }}", str(user['id']))  
 else:  
 template = template.replace("{% if user %}", "").replace("{% endif %}", "")  
 template = template.replace("<p>Вы вошли как: {{ user.full\_name }} ({{ user.email }})</p>", "")  
 template = template.replace("<a href=\"/user/{{ user.id }}\"><button>Перейти в личный кабинет</button></a>",  
 "")  
 template = template.replace("<a href=\"/logout\"><button>Выйти</button></a>", "")  
  
 return html(template)  
  
  
# Тестовый маршрут  
@app.route("/test/")  
async def test(request):  
 return json({"message": "Welcome to the API!"})  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host="0.0.0.0", port=8000)

**А модуль login имеет следующий код:**

from sanic import Blueprint  
from sanic.response import html, redirect, json  
from sanic.exceptions import NotFound, Unauthorized  
from sqlalchemy import select  
from app.database import async\_session  
from app.models import User  
from pathlib import Path  
import logging  
  
login\_bp = Blueprint('login', url\_prefix='/')  
logger = logging.Logger(\_\_name\_\_)  
  
  
@login\_bp.get('/login')  
async def login\_get(request):  
 with open(Path('app/templates/login.html'), 'r') as f:  
 return html(f.read())  
  
  
@login\_bp.post('/login')  
async def login\_post(request):  
 email = request.form.get("email")  
 password = request.form.get("password")  
  
 logger.warning(f"request.ctx: {request.ctx}")  
 async with async\_session() as session:  
 result = await session.execute(select(User).where(User.email == email))  
 user = result.scalars().first()  
  
 if not user or user.password != password:  
 raise Unauthorized("Invalid email or password")  
  
 logger.warning("🟡 session.init\_app() вызывается!")  
  
 # Сохраняем данные пользователя в сессии  
 request.ctx.session['user'] = {  
 "id": user.id,  
 "email": user.email,  
 "full\_name": user.full\_name,  
 "is\_admin": user.is\_admin  
 }  
  
 # Перенаправляем на главную страницу  
 return redirect('/')  
  
  
@login\_bp.route('/logout', methods=['GET'], name="logout")  
async def logout(request):  
 # Очищаем сессию  
 if 'user' in request.ctx.session:  
 del request.ctx.session['user']  
 return redirect('/')

**При нажатии на кнопку «Login» и последующим переходе по адресу "/" происходит следующая ошибка:**app-1 | Srv 0 2025-02-06 19:46:21 +0000 ERROR: Exception occurred while handling uri: 'http://localhost:8000/login'

app-1 | Traceback (most recent call last):

app-1 | File "handle\_request", line 100, in handle\_request

app-1 | Extend = TypeVar("Extend", type) # type: ignore

app-1 | File "/app/app/routes/login.py", line 39, in login\_post

app-1 | request.ctx.session['user'] = {

app-1 | AttributeError: 'types.SimpleNamespace' object has no attribute 'session'

**Теперь, при попытке запустить сервер, возникает следующая ошибка:**

app-1 | Traceback (most recent call last):

app-1 | File "/usr/local/bin/sanic", line 8, in <module>

app-1 | sys.exit(main())

app-1 | File "/usr/local/lib/python3.10/site-packages/sanic/\_\_main\_\_.py", line 12, in main

app-1 | cli.run(args)

app-1 | File "/usr/local/lib/python3.10/site-packages/sanic/cli/app.py", line 123, in run

app-1 | app = self.\_get\_app(app\_loader)

app-1 | File "/usr/local/lib/python3.10/site-packages/sanic/cli/app.py", line 218, in \_get\_app

app-1 | app = app\_loader.load()

app-1 | File "/usr/local/lib/python3.10/site-packages/sanic/worker/loader.py", line 92, in load

app-1 | module = import\_module(self.module\_name)

app-1 | File "/usr/local/lib/python3.10/importlib/\_\_init\_\_.py", line 126, in import\_module

app-1 | return \_bootstrap.\_gcd\_import(name[level:], package, level)

app-1 | File "<frozen importlib.\_bootstrap>", line 1050, in \_gcd\_import

app-1 | File "<frozen importlib.\_bootstrap>", line 1027, in \_find\_and\_load

app-1 | File "<frozen importlib.\_bootstrap>", line 1006, in \_find\_and\_load\_unlocked

app-1 | File "<frozen importlib.\_bootstrap>", line 688, in \_load\_unlocked

app-1 | File "<frozen importlib.\_bootstrap\_external>", line 883, in exec\_module

app-1 | File "<frozen importlib.\_bootstrap>", line 241, in \_call\_with\_frames\_removed

app-1 | File "/app/app/main.py", line 25, in <module>

app-1 | session = Session(app, interface=RedisSessionInterface(app.ctx.redis))

app-1 | AttributeError: 'types.SimpleNamespace' object has no attribute 'redis'

docker exec -it 8f030fac3c5b redis-cli ping

**Теперь, при ри нажатии на кнопку «Login» и последующим переходе по адресу "/" происходит следующая ошибка:**app-1 | Srv 0 2025-02-07 06:14:17 +0000 ERROR: Exception occurred while handling uri: 'http://localhost:8000/login'

app-1 | Traceback (most recent call last):

app-1 | File "handle\_request", line 100, in handle\_request

app-1 | Extend = TypeVar("Extend", type) # type: ignore

app-1 | File "/app/app/routes/login.py", line 34, in login\_post

app-1 | request.ctx.session['user'] = {

app-1 | AttributeError: 'types.SimpleNamespace' object has no attribute 'session'