Министерство науки и высшего образования  
 Российской Федерации   
 федеральное государственное автономное  
 образовательное учреждение высшего образования  
 «Самарский национальный исследовательский университет имени академика С.П. Королева»

Институт информатики, математики и электроники

Факультет информатики  
 Кафедра технической кибернетики

**ИТОГОВЫЙ ОТЧЕТ**

по разработке полноценного веб-приложения

Дисциплина «Технологии сетевого программирования»

##### 

Работу выполнили студенты

группы 6301-010302D

Иванова Анна

Рамазанова Милана

**САМАРА  
 2025**

# Выполнение заданий

**Стек технологий**

1. PostgreSQL

2. Visual Studio Code

3. Git, GitHub

4. OpenAPI/Swagger

5. SQLAlchemy, Postman, Curl, Docker

6. HTML, CSS

5. [Draw.io](http://draw.io)

**Концепция приложения.**

1. Регистрация нового пользователя:

· Клиент отправляет POST-запрос на сервер с данными регистрации

· Сервер проверяет данные и создает нового пользователя в базе данных

· Сервер отправляет только зарегистрированного пользователя на страницу входа

2. Вход зарегистрированного пользователя:

· Клиент далее использует логин и пароль, которые ввел в регистрации.

· Сервер проверяет логин и пароль и является ли пользователь админом, и возвращает токен аутентификации (например, JWT) при успешной проверке.

· Сервер проверяет данные и возвращает токен аутентификации (например, JWT) при успешной проверке.

· Сервер отправляет пользователя на его страницу.

3. Получение списка психологов:

· Если сервер перенаправляет пользователя на страницу админа, то на странице появляется страница список психологов, их добавление и регистрация.

4. Создание нового сеанса:

· Клиент отправляет POST-запрос на сервер с данными о новом сеансе (выбранный психолог, дата и время).

· Сервер добавляет новый сеанс в базу данных и возвращает подтверждение клиенту.

5. Уведомления:

· Сервер проверяет, что пользователь успешно зашел на свою страницу и отправляет на его страницу маленькое окошко с фразами, которые заранее прописаны в сервере, для поднятия настроения

**Проектирование архитектуры**

Была разработана логическая схема базы данных, которая показана на рис. 1.

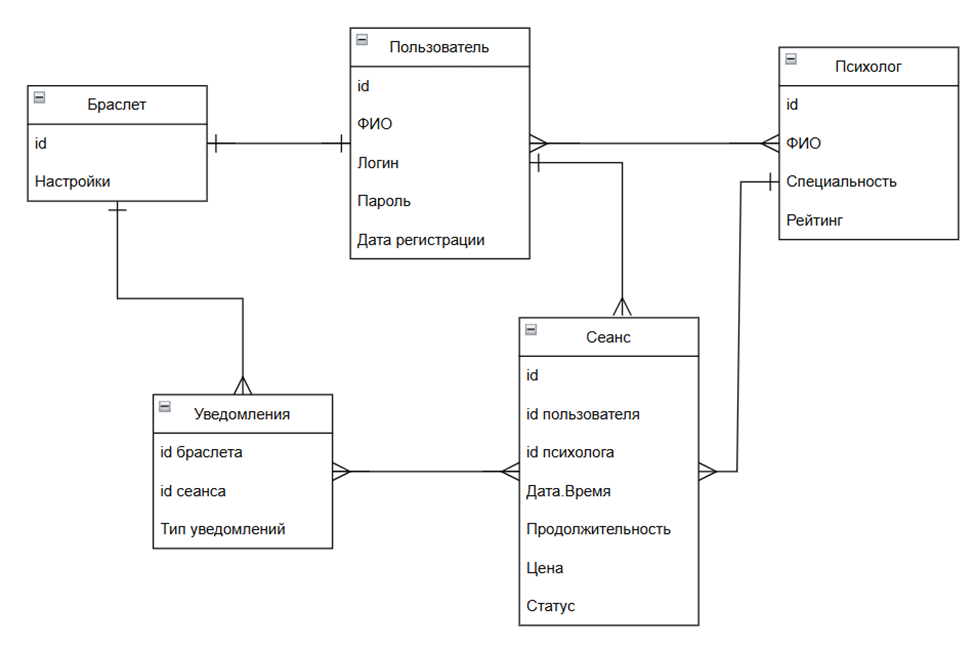


Рисунок 1 - Схема базы данных

**Структура API**

Таблица 1 - Структура API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Метод** | **URL** | **Формат запроса** | **Формат ответа** | **Описание** |
| POST | /api/token |  | {  "access\_token": "eyJhbG...",  "token\_type": "bearer",  "is\_admin": false,  "redirect\_url": "/api/user/dashboard"  } | Аутентификация |
| POST | /api/users/ | { "full\_name": "Иван Иванов",  "username": "ivanov",  "password": "securepassword123"  } | { "id": 1,  "full\_name": "Иван Иванов",  "username": "ivanov",  "registration\_date": "2023-10-25T14:30:00"} | Регистрация нового пользователя в системе. |
| GET | /api/users/14 |  | { "id": 1,  "full\_name": "Иван Иванов",  "username": "ivanov"} | Получение информации о конкретном пользователе. |
| PUT | /api/users/1 | { "full\_name": "Иван Иванов",  "username": "ivanov\_new",  "password": "newpassword123"} | { "id": 1,  "full\_name": "Иван Иванов",  "username": "ivanov\_new"  } | Обновление информации о пользователе. |
| DELETE | /api/users/21 |  | { "message": "User deleted successfully"  } | Удаление пользователя |
| POST | /users/change-password | { "old\_password": "oldpassword123",  "new\_password": "newsecurepassword456"  } | {  "id": 1,  "full\_name": "Иван Иванов",  "username": "ivanov"  } | Смена пароля текущего пользователя. |
| POST | /users/logout |  | {  "message": "You have successfully logged out"  } | Добавление токена в черный список (инвалидация токена). |
| POST | /sessions/ | { "psychologist\_id": 1,  "date\_time": "2023-11-15T14:00:00",  "duration": 60,  "price": 2500,  "status": "pending",  "notes": "Тема: тревожность"  } | {  "id": 1,  "user\_id": 5,  "psychologist\_id": 1,  "date\_time": "2023-11-15T14:00:00",  "status": "pending"  } | Создание сессии |
| GET | /sessions/{session\_id} |  | { "id": 1,  "psychologist\_id": 1,  "date\_time": "2023-11-15T14:00:00",  "status": "confirmed",  "notes": "Первая консультация"  } | Получение сессии по id |
| PUT | /sessions/{session\_id} | {  "status": "cancelled",  "notes": "Перенесено по просьбе клиента"  } | {  "id": 1,  "status": "cancelled",  "notes": "Перенесено по просьбе клиента"  } | Обновление сессии |
| DELETE | /sessions/{session\_id} |  | {  "message": "Sessions deleted successfully"  } | Удаление сессии |
| POST | /psychologists/ | {  "name": "Анна Петрова",  "specialization": "Когнитивно-поведенческая терапия",  "bio": "Опыт работы 10 лет...",  "photo\_url": "https://example.com/photo.jpg"  } | {  "id": 1,  "name": "Анна Петрова",  "specialization": "Когнитивно-поведенческая терапия"  } | Создание психолога |
| GET | /psychologists/ |  | [  {  "id": 1,  "name": "Анна Петрова",  "specialization": "КПТ"  },  {  "id": 2,  "name": "Иван Сидоров",  "specialization": "Гештальт-терапия"  }  ] | Получение списка психологов |
| PUT | /psychologists/{psychologist\_id} | {  "name": "Анна Петрова",  "specialization": "Тревожные расстройства"  } | {  "id": 1,  "name": "Анна Петрова",  "specialization": "Тревожные расстройства"  } | Обновление данных психолога |
| DELETE | /psychologists/{psychologist\_id} |  | {  "message": "Psychologist deleted successfully"  } | Удаление психолога |
| POST | /notifications/ | {  "bracelet\_id": 123,  "session\_id": 456,  "message\_type": "emergency\_alert"  } | {  "id": 1,  "bracelet\_id": 123,  "session\_id": 456,  "timestamp": "2023-11-20T14:30:00",  "message\_type": "emergency\_alert"  } | Создание уведомления |
| GET | /notifications/{notification\_id} |  | {  "id": 1,  "bracelet\_id": 123,  "session\_id": 456,  "message\_type": "vital\_warning",  "timestamp": "2023-11-20T14:30:00"  } | Получение уведомление по id |
| PUT | /notifications/{notification\_id} | {  message\_type": "false\_alarm"  } | {  "id": 1,  "message\_type": "false\_alarm"  } | Обновление уведомления |
| DELETE | /notifications/{notification\_id} |  | {  "message": "Notification deleted successfully"  } | Удаление уведомления |
| POST | /bracelets/ | {  "settings": "{\"alert\_threshold\": 120}",  "user\_id": 1  } | {  "id": 1,  "settings": "{\"alert\_threshold\": 120}",  "user\_id": 1,  "registration\_date": "2023-11-20T12:00:00"  } | Создание браслета |
| GET | /bracelets/{bracelet\_id} |  | {  "id": 1,  "settings": "{\"vibration\":true}",  "user\_id": 1,  "last\_sync": "2023-11-20T12:30:00"  } | Получение браслета по id |
| PUT | /bracelets/{bracelet\_id} | {  "settings": "{\"alert\_threshold\":130}"  } | {  "id": 1,  "settings": "{\"alert\_threshold\":130}"  } | Обновление браслета |
| DELETE | /bracelets/{bracelet\_id} |  | {  "message": "Bracelets deleted successfully"  } | Удаление браслета |

**Работающее приложение**

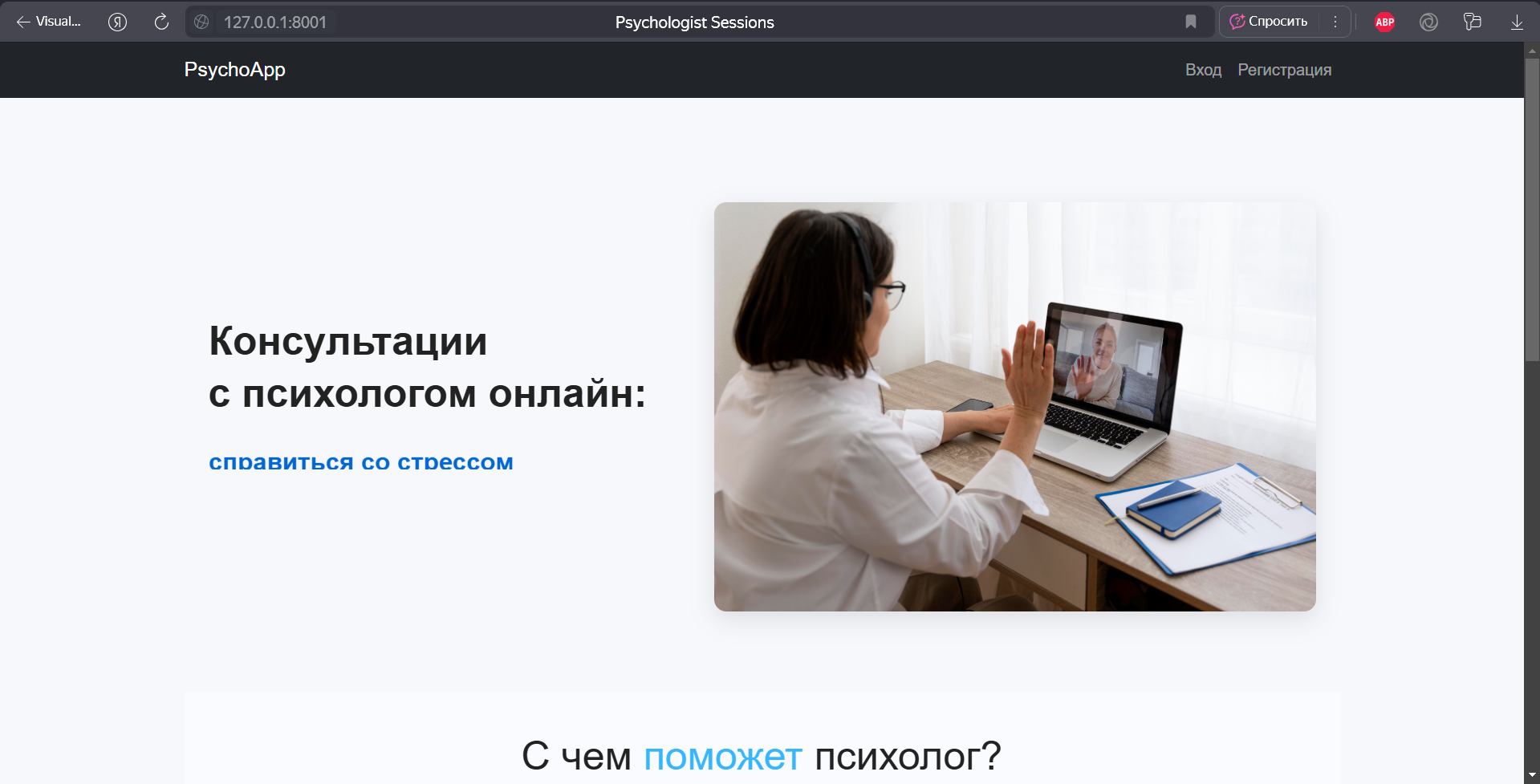
****

Рисунок 2 - Основная страница приложения

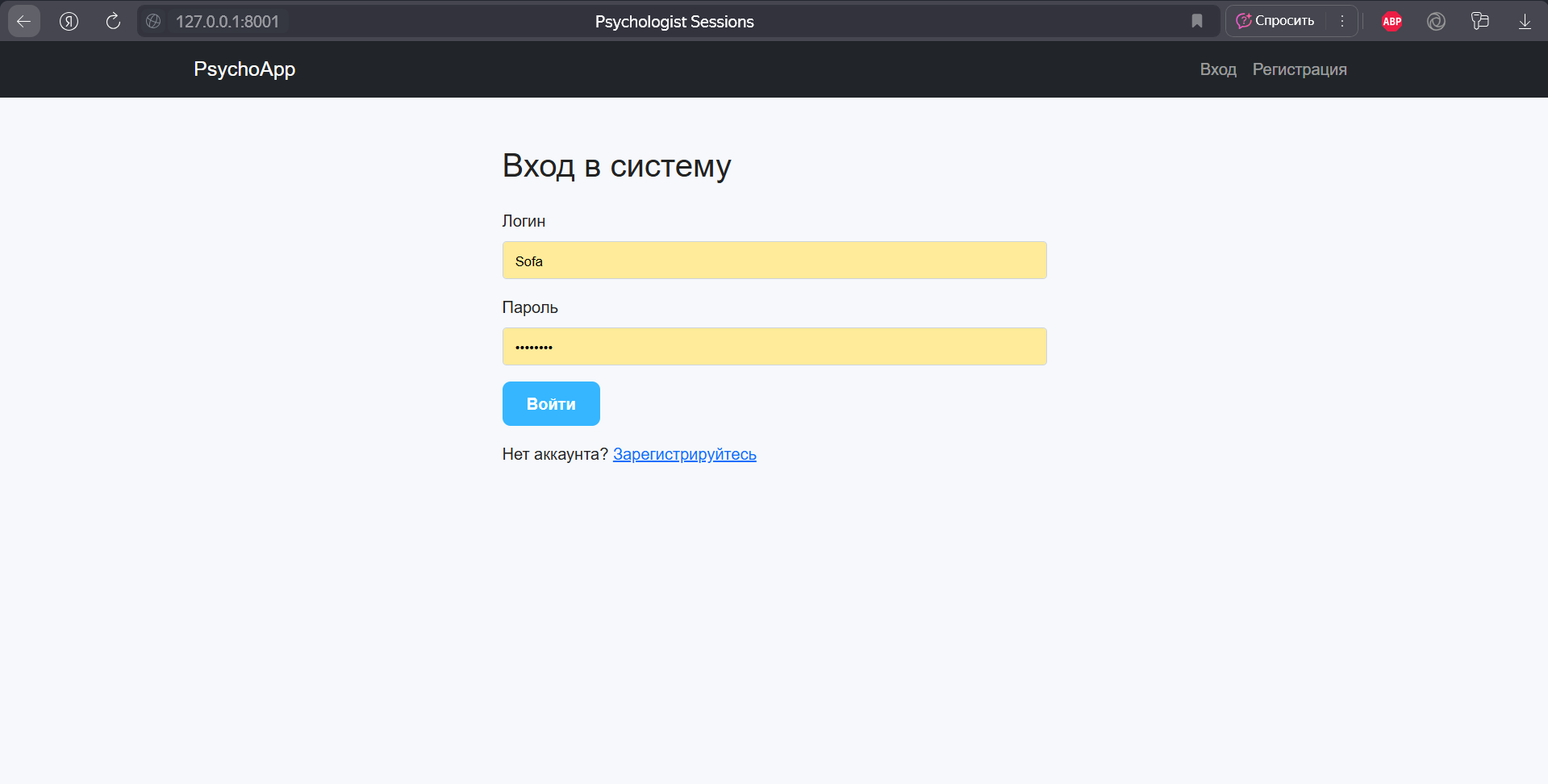


Рисунок 3 - Вход пользователя

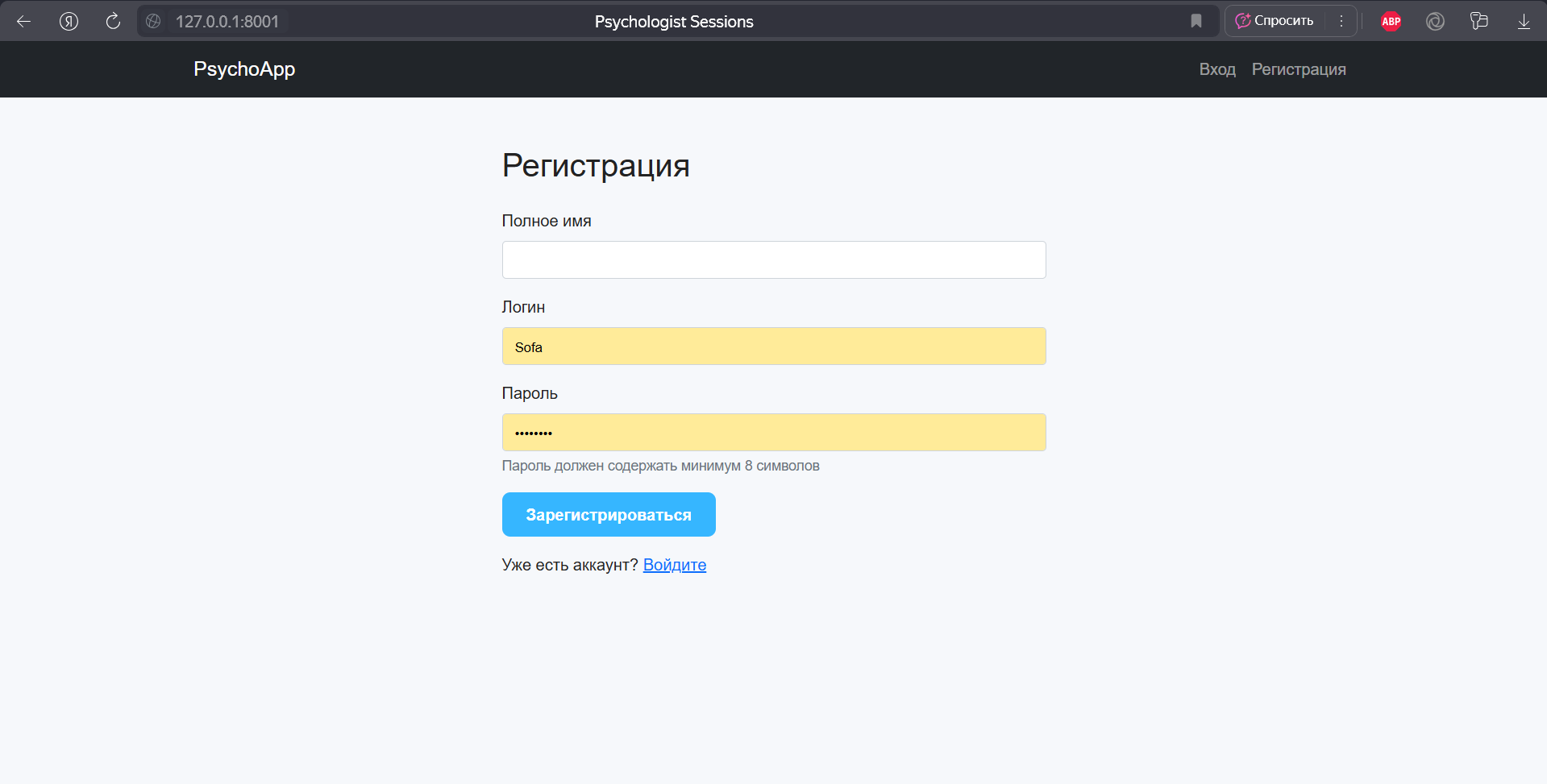


Рисунок 4 - Регистрация пользователя

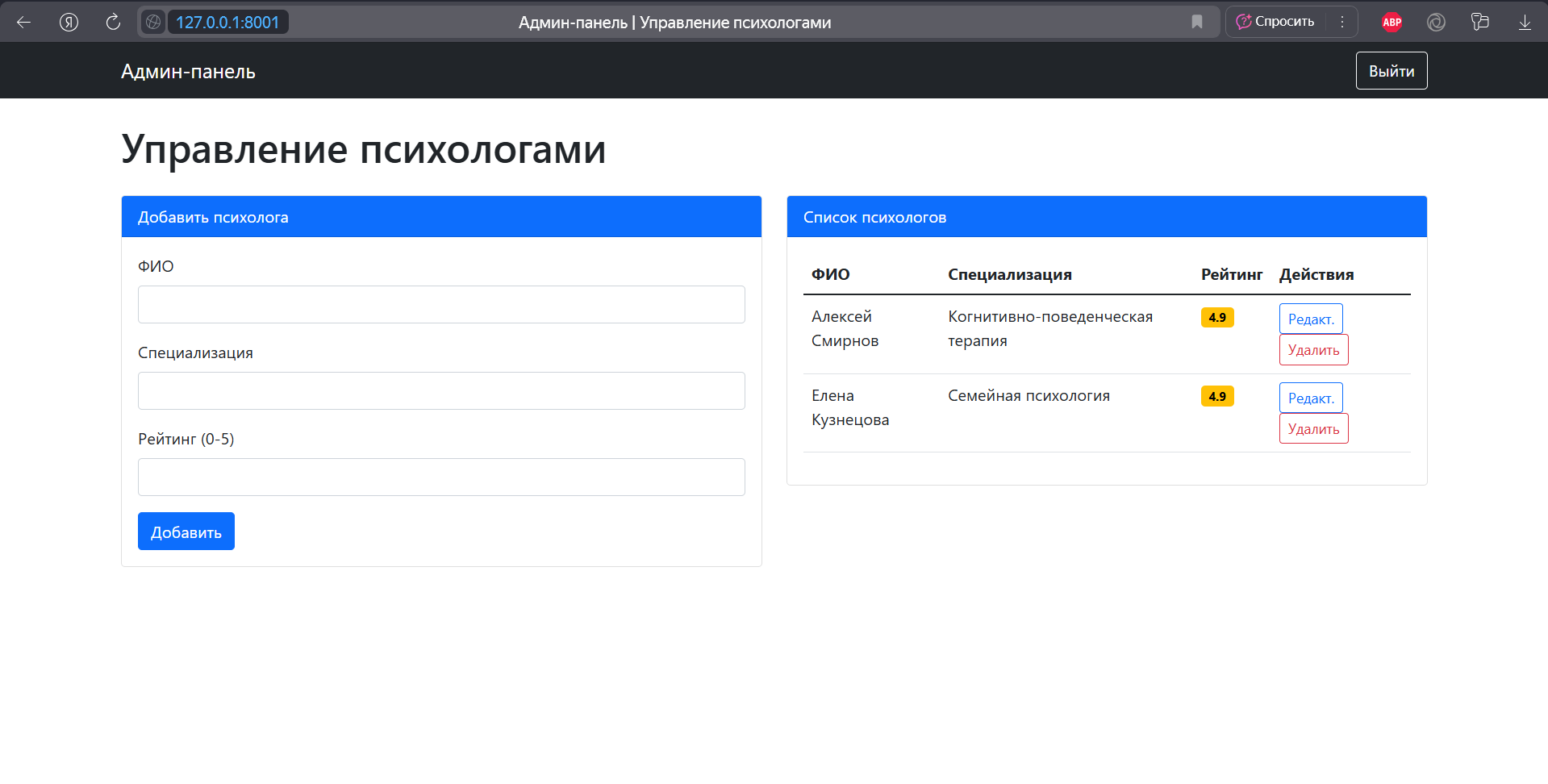


Рисунок 5 - Страница админа

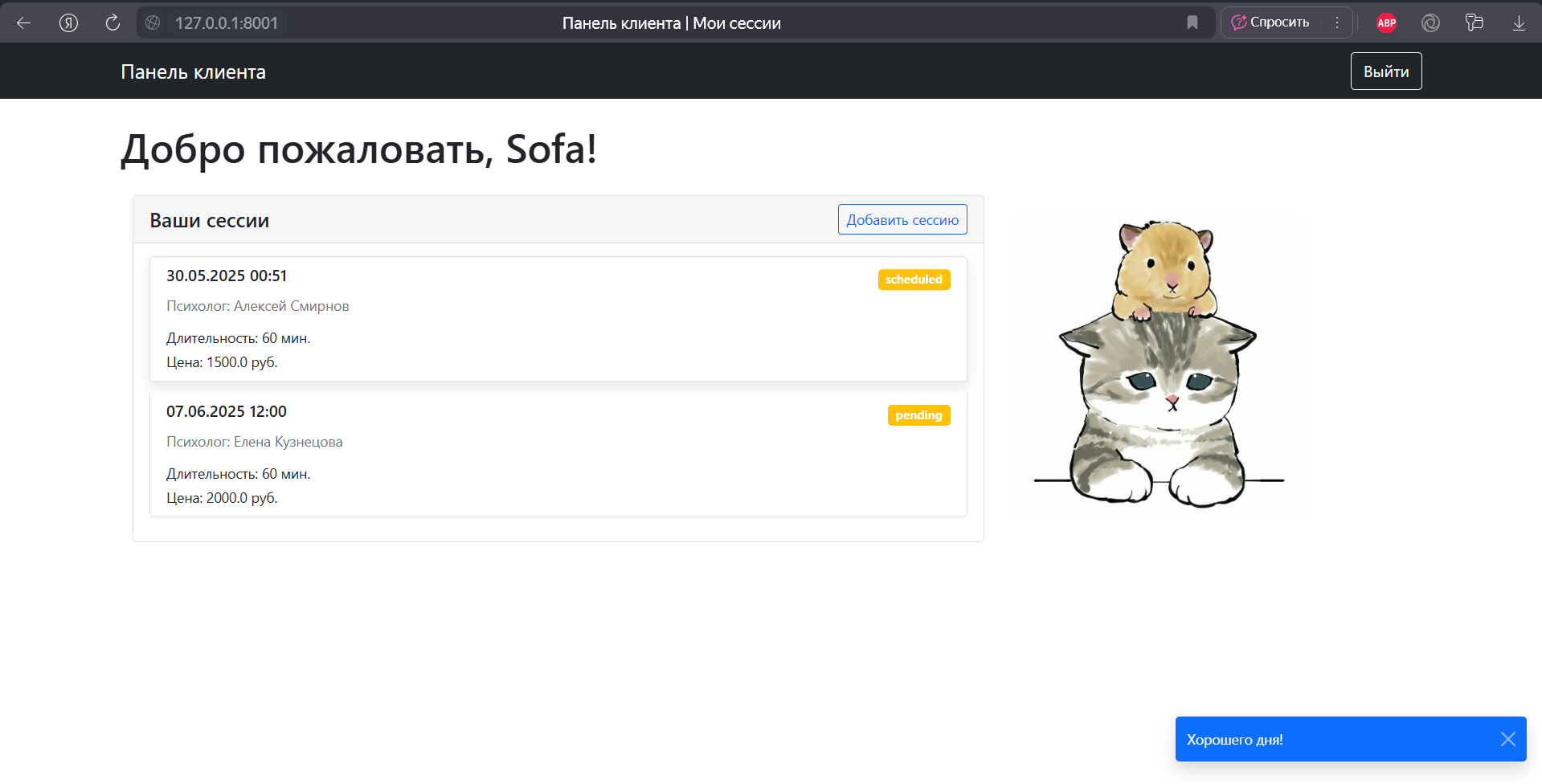


Рисунок 6 - Страница пользователя

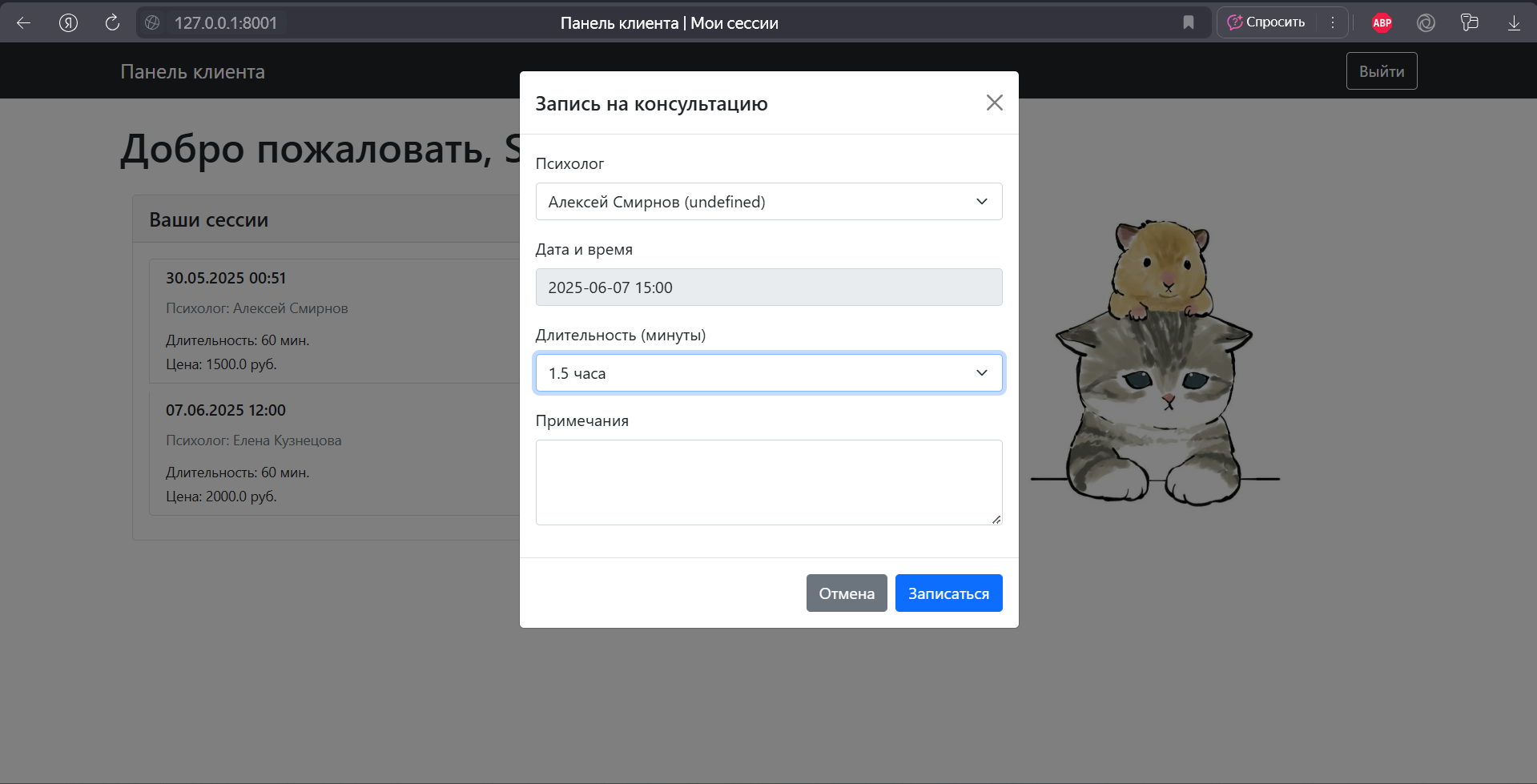


Рисунок 7 - Добавление сессии пользователей

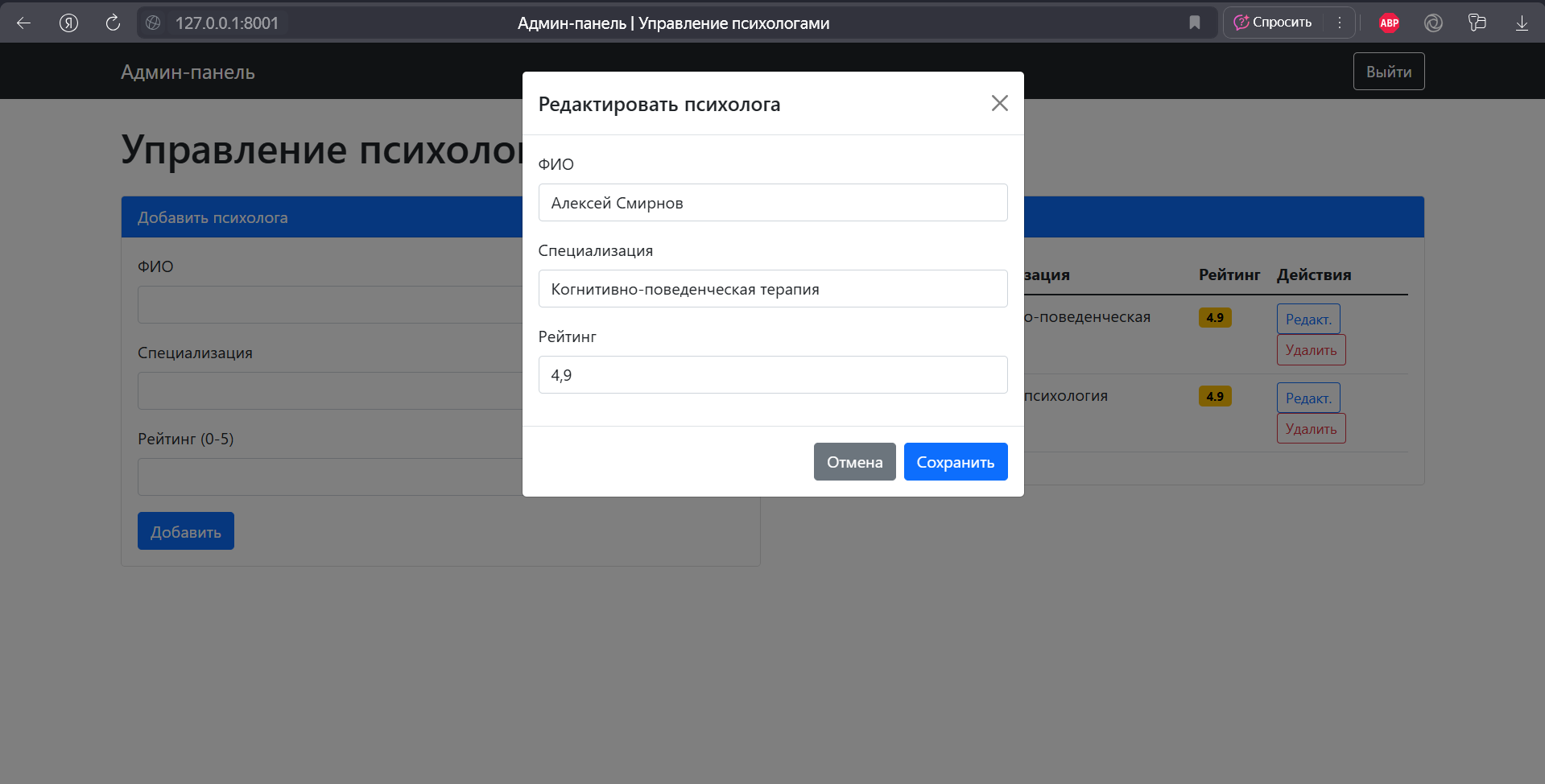


Рисунок 8 - Редактирование психолога админом

# ПРИЛОЖЕНИЕ

1. **.gitignore**

# Файлы окружения

.env

\*.env

.env.example

# Logs

logs

\*.log

npm-debug.log\*

yarn-debug.log\*

yarn-error.log\*

lerna-debug.log\*

.pnpm-debug.log\*

# Diagnostic reports (https://nodejs.org/api/report.html)

report.[0-9]\*.[0-9]\*.[0-9]\*.[0-9]\*.json

# Runtime data

pids

\*.pid

\*.seed

\*.pid.lock

# Directory for instrumented libs generated by jscoverage/JSCover

lib-cov

# Coverage directory used by tools like istanbul

coverage

\*.lcov

# nyc test coverage

.nyc\_output

# Grunt intermediate storage (https://gruntjs.com/creating-plugins#storing-task-files)

.grunt

# Bower dependency directory (https://bower.io/)

bower\_components

# node-waf configuration

.lock-wscript

# Compiled binary addons (https://nodejs.org/api/addons.html)

build/Release

# Dependency directories

node\_modules/

jspm\_packages/

# Snowpack dependency directory (https://snowpack.dev/)

web\_modules/

# TypeScript cache

\*.tsbuildinfo

# Optional npm cache directory

.npm

# Optional eslint cache

.eslintcache

# Optional stylelint cache

.stylelintcache

# Microbundle cache

.rpt2\_cache/

.rts2\_cache\_cjs/

.rts2\_cache\_es/

.rts2\_cache\_umd/

# Optional REPL history

.node\_repl\_history

# Output of 'npm pack'

\*.tgz

# Yarn Integrity file

.yarn-integrity

# dotenv environment variable files

.env

.env.development.local

.env.test.local

.env.production.local

.env.local

# parcel-bundler cache (https://parceljs.org/)

.cache

.parcel-cache

# Next.js build output

.next

out

# Nuxt.js build / generate output

.nuxt

dist

# Gatsby files

.cache/

# Comment in the public line in if your project uses Gatsby and not Next.js

# https://nextjs.org/blog/next-9-1#public-directory-support

# public

# vuepress build output

.vuepress/dist

# vuepress v2.x temp and cache directory

.temp

.cache

# Docusaurus cache and generated files

.docusaurus

# Serverless directories

.serverless/

# FuseBox cache

.fusebox/

# DynamoDB Local files

.dynamodb/

# TernJS port file

.tern-port

# Stores VSCode versions used for testing VSCode extensions

.vscode-test

# yarn v2

.yarn/cache

.yarn/unplugged

.yarn/build-state.yml

.yarn/install-state.gz

.pnp.\*

1. [**main.py**](http://main.py)

from fastapi import FastAPI, Depends, Request

from fastapi.middleware.cors import CORSMiddleware

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker, Session

from contextlib import contextmanager

from sqlalchemy import text

import logging

from fastapi.staticfiles import StaticFiles

from fastapi.templating import Jinja2Templates

from pathlib import Path

from server.app.dataBase.base import settings

from fastapi.middleware.cors import CORSMiddleware

from server.app.middleware.auth\_middleware import AuthMiddleware

# Настройка логгера

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

# Подключение к БД

DATABASE\_URL = "postgresql://postgres:123@localhost:5433/DataBase"

engine = create\_engine(DATABASE\_URL, pool\_size=20, max\_overflow=10)

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

# Dependency для получения сессии

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

def check\_connection():

try:

with engine.connect() as conn:

conn.execute(text("SELECT 1"))

logger.info("✓ Подключение к БД успешно")

except Exception as e:

logger.error(f"× Ошибка подключения: {e}")

raise

def create\_tables():

from server.app.dataBase.base import Base

try:

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

logger.info("✓ Таблицы созданы/проверены")

except Exception as e:

logger.error(f"× Ошибка создания таблиц: {e}")

raise

app = FastAPI(

title="Psychologist Session API",

version="1.0.0"

)

app.add\_middleware(

CORSMiddleware,

allow\_origins=["http://localhost:8000", "http://127.0.0.1:8000"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

expose\_headers=["\*"]

)

app.add\_middleware(AuthMiddleware)

# Инициализация при старте

@app.on\_event("startup")

def startup():

logger.info("Запуск инициализации...")

check\_connection()

create\_tables()

logger.info("Инициализация завершена")

# Подключение роутеров (синхронные версии)

from server.app.api.endpoints import user, auth, psychologist, session, notification, bracelet # и другие

# Подключаем роутеры

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

app.include\_router(user.router, prefix="/api/users", tags=["Пользователи"])

app.include\_router(psychologist.router, prefix="/api/psychologists", tags=["Психологи"])

app.include\_router(session.router, prefix="/api/sessions", tags=["Сессии"])

app.include\_router(notification.router, prefix="/api/notifications", tags=["Уведомления"])

app.include\_router(bracelet.router, prefix="/api/bracelets", tags=["Браслеты"])

@app.get("/", include\_in\_schema=False)

async def home(request: Request):

return templates.TemplateResponse("index.html", {"request": request})

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

import uvicorn

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

print("Текущие настройки:")

print(f"Algorithm: {settings.ALGORITHM}")

print(f"Token expires in: {settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES} minutes")

# Настройка путей

BASE\_DIR = Path(\_\_file\_\_).resolve().parent

templates = Jinja2Templates(directory=str(BASE\_DIR / "templates"))

app.mount("/static", StaticFiles(directory=str(BASE\_DIR / "static")), name="static")

@app.get("/login")

async def login\_page(request: Request):

return templates.TemplateResponse("auth/login.html", {"request": request})

@app.get("/register")

async def register\_page(request: Request):

return templates.TemplateResponse("auth/register.html", {"request": request})

1. **/models/user.py**

from sqlalchemy import create\_engine, Column, Integer, String, Float, DateTime, ForeignKey, Table, Boolean

from sqlalchemy.orm import declarative\_base

from sqlalchemy.orm import relationship

from datetime import datetime

from passlib.context import CryptContext

from zoneinfo import ZoneInfo

from ..base import Base

# Настроим хеширование паролей

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

# Модель Пользователь (User)

class User(Base):

\_\_tablename\_\_ = 'users'

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

full\_name = Column(String(100), nullable=False)

username = Column(String(50), unique=True, nullable=False)

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

registration\_date = Column(DateTime, nullable=False)

is\_admin = Column(Boolean, default=False)

is\_active = Column(Boolean, default=True)

# Связи

bracelet = relationship('Bracelet', back\_populates='user', uselist=False)

psychologists = relationship('Psychologist', secondary='user\_psychologist', back\_populates='users')

sessions = relationship('Session', back\_populates='user')

# Добавляем property для совместимости

@property

def hashed\_password(self):

return self.password

def set\_password(self, password):

self.password = pwd\_context.hash(password)

def verify\_password(self, password):

return pwd\_context.verify(password, self.password)

1. **/models/psychologist**[**.py**](http://user.py)

from sqlalchemy import create\_engine, Column, Integer, String, Float, DateTime, ForeignKey, Table

from sqlalchemy.orm import declarative\_base

from sqlalchemy.orm import relationship

from datetime import datetime

from passlib.context import CryptContext

from zoneinfo import ZoneInfo

from ..base import Base

# Модель Психолог (Psychologist)

class Psychologist(Base):

\_\_tablename\_\_ = 'psychologists'

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

full\_name = Column(String(100), nullable=False)

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

rating = Column(Float, nullable=False)

# Связи

users = relationship('User', secondary='user\_psychologist', back\_populates='psychologists')

sessions = relationship('Session', back\_populates='psychologist')

def repr(self):

return f"<Psychologist(id={self.id}, full\_name={self.full\_name})>"

# Таблица для связи многие-ко-многим между User и Psychologist

user\_psychologist = Table(

'user\_psychologist', Base.metadata,

Column('user\_id', Integer, ForeignKey('users.id')),

Column('psychologist\_id', Integer, ForeignKey('psychologists.id'))

)

1. **/models/session**[**.py**](http://user.py)

from sqlalchemy import create\_engine, Column, Integer, String, Float, DateTime, ForeignKey, Table

from sqlalchemy.orm import declarative\_base

from sqlalchemy.orm import relationship

from datetime import datetime

from passlib.context import CryptContext

from zoneinfo import ZoneInfo

from ..base import Base

# Модель Сеанс (Session)

class Session(Base):

\_\_tablename\_\_ = 'sessions'

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

user\_id = Column(Integer, ForeignKey('users.id'))

psychologist\_id = Column(Integer, ForeignKey('psychologists.id'))

date\_time = Column(DateTime, nullable=False)

duration = Column(Integer, nullable=False)

price = Column(Float, nullable=False)

status = Column(String(50), nullable=False)

notes = Column(String, nullable=True)

# Связи

user = relationship('User', back\_populates='sessions')

psychologist = relationship('Psychologist', back\_populates='sessions')

notifications = relationship('Notification', back\_populates='session')

def \_\_repr\_\_(self):

return f"<Session(id={self.id}, status={self.status})>"

1. **/models/notification**[**.py**](http://user.py)

from sqlalchemy import create\_engine, Column, Integer, String, Float, DateTime, ForeignKey, Table

from sqlalchemy.orm import declarative\_base

from sqlalchemy.orm import relationship

from datetime import datetime

from passlib.context import CryptContext

from zoneinfo import ZoneInfo

from ..base import Base

# Модель Уведомления (Notification)

class Notification(Base):

\_\_tablename\_\_ = 'notifications'

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

bracelet\_id = Column(Integer, ForeignKey('bracelets.id'))

session\_id = Column(Integer, ForeignKey('sessions.id'))

message\_type = Column(String(1000), nullable=False)

# Связи

bracelet = relationship('Bracelet', back\_populates='notifications')

session = relationship('Session', back\_populates='notifications')

@property

def user(self):

"""Получаем пользователя через связанный браслет"""

return self.bracelet.user if self.bracelet else None

def \_\_repr\_\_(self):

return f"<Notification(id={self.id})>"

1. **/models/bracelet**[**.py**](http://user.py)

from sqlalchemy import create\_engine, Column, Integer, String, Float, DateTime, ForeignKey, Table

from sqlalchemy.orm import declarative\_base

from sqlalchemy.orm import relationship

from datetime import datetime

from passlib.context import CryptContext

from zoneinfo import ZoneInfo

from ..base import Base

# Модель Браслет (Bracelet)

class Bracelet(Base):

\_\_tablename\_\_ = 'bracelets'

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

settings = Column(String(255), nullable=False)

user\_id = Column(Integer, ForeignKey('users.id'), unique=True)

# Связи

user = relationship('User', back\_populates='bracelet')

notifications = relationship('Notification', back\_populates='bracelet')

def \_\_repr\_\_(self):

return f"<Bracelet(id={self.id}, settings={self.settings})>"

1. **/endpoints/auth**[**.py**](http://user.py)

from fastapi import APIRouter, Depends, HTTPException, Request, Form, status, Response, Cookie

from fastapi.responses import RedirectResponse, HTMLResponse, JSONResponse

from fastapi.security import OAuth2PasswordBearer, HTTPBearer

from fastapi.templating import Jinja2Templates

from sqlalchemy.orm import Session

from datetime import timedelta, datetime

from jose import JWTError, jwt

from passlib.context import CryptContext

import os

from dotenv import load\_dotenv

from pathlib import Path

# Импорты из вашего проекта

from server.app.main import get\_db

from server.app.dataBase.models.user import User

from server.app.api.deps import add\_to\_blacklist

from server.app.api.deps import get\_current\_user

from server.app.dataBase.models.session import Session as SessionModel

load\_dotenv()

router = APIRouter(tags=["auth"])

security = HTTPBearer()

# Настройка шаблонов

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent.parent.parent

templates = Jinja2Templates(directory=str(BASE\_DIR / "app" / "templates"))

# Конфигурация аутентификации

SECRET\_KEY = os.getenv("SECRET\_KEY")

if not SECRET\_KEY:

raise RuntimeError("SECRET\_KEY не задан в .env!")

ALGORITHM = "HS256"

ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="api/token")

def verify\_password(plain\_password: str, hashed\_password: str) -> bool:

return pwd\_context.verify(plain\_password, hashed\_password)

def authenticate\_user(username: str, password: str, db: Session) -> User | None:

user = db.query(User).filter(User.username == username).first()

if not user or not verify\_password(password, user.password):

return None

return user

def create\_access\_token(data: dict, expires\_delta: timedelta = None) -> str:

to\_encode = data.copy()

expire = datetime.utcnow() + (expires\_delta or timedelta(minutes=15))

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

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

def get\_user\_from\_token(token: str, db: Session) -> User | None:

try:

payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])

username = payload.get("sub")

if username is None:

return None

return db.query(User).filter(User.username == username).first()

except JWTError:

return None

@router.post("/token")

async def login\_for\_access\_token(

username: str = Form(...),

password: str = Form(...),

db: Session = Depends(get\_db)

):

user = authenticate\_user(username, password, db)

if not user:

raise HTTPException(

status\_code=401,

detail="Неверное имя пользователя или пароль",

headers={"WWW-Authenticate": "Bearer"}

)

access\_token = create\_access\_token(

data={"sub": user.username, "user\_id": user.id, "is\_admin": user.is\_admin},

expires\_delta=timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)

)

# Возвращаем JSON с токеном и информацией для редиректа

return {

"access\_token": access\_token,

"token\_type": "bearer",

"is\_admin": user.is\_admin,

"redirect\_url": "/api/admin/dashboard" if user.is\_admin else "/api/user/dashboard"

}

@router.post("/login", response\_class=HTMLResponse)

async def login\_post(

request: Request,

username: str = Form(...),

password: str = Form(...),

db: Session = Depends(get\_db)

):

user = authenticate\_user(username, password, db)

if not user:

return templates.TemplateResponse(

"auth/login.html",

{"request": request, "error": "Неверный логин или пароль"}

)

access\_token = create\_access\_token(

data={"sub": user.username, "user\_id": user.id, "is\_admin": user.is\_admin},

expires\_delta=timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)

)

redirect\_url = "/api/admin/dashboard" if user.is\_admin else "/api/user/dashboard"

response = RedirectResponse(url=redirect\_url, status\_code=303)

response.set\_cookie(

key="access\_token",

value=f"Bearer {access\_token}",

httponly=True,

max\_age=ACCESS\_TOKEN\_EXPIRE\_MINUTES \* 60,

secure=False,

samesite="lax",

path="/"

)

response.set\_cookie(

key="user\_id",

value=str(user.id),

httponly=False, # ❗ Доступен в JS

max\_age=ACCESS\_TOKEN\_EXPIRE\_MINUTES \* 60,

secure=False,

samesite="lax",

path="/"

)

return response

# Добавляем новые роуты для разных типов пользователей

@router.get("/admin/dashboard", response\_class=HTMLResponse)

async def admin\_dashboard(

request: Request,

access\_token: str = Cookie(None),

db: Session = Depends(get\_db)

):

if not access\_token:

return RedirectResponse(url="/api/login", status\_code=303)

try:

token = access\_token.replace("Bearer ", "")

payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])

# Проверяем, что пользователь админ

if not payload.get("is\_admin"):

return RedirectResponse(url="/api/login", status\_code=303)

username = payload.get("sub")

user = db.query(User).filter(User.username == username).first()

return templates.TemplateResponse(

"admin\_dashboard.html", # Ваш шаблон для админа

{"request": request, "user": user}

)

except JWTError:

return RedirectResponse(url="/api/login", status\_code=303)

@router.get("/user/dashboard", response\_class=HTMLResponse)

async def user\_dashboard(

request: Request,

access\_token: str = Cookie(None),

db: Session = Depends(get\_db)

):

if not access\_token:

return RedirectResponse(url="/api/login", status\_code=303)

try:

token = access\_token.replace("Bearer ", "")

payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])

username = payload.get("sub")

user = db.query(User).filter(User.username == username).first()

if not user:

return RedirectResponse(url="/api/login", status\_code=303)

sessions = db.query(SessionModel).filter(SessionModel.user\_id == user.id).all()

return templates.TemplateResponse(

"dashboard.html", # Обычный дашборд

{"request": request, "user": user, "sessions": sessions}

)

except JWTError:

return RedirectResponse(url="/api/login", status\_code=303)

@router.get("/login", response\_class=HTMLResponse)

async def login\_page(request: Request, registration: str = None):

context = {"request": request}

if registration == "success":

context["success"] = "Регистрация прошла успешно! Теперь вы можете войти."

return templates.TemplateResponse("auth/login.html", context)

@router.post("/register", response\_class=RedirectResponse)

async def register\_user(

full\_name: str = Form(...),

username: str = Form(...),

password: str = Form(...),

db: Session = Depends(get\_db)

):

# Проверка длины пароля

if len(password) < 8:

raise HTTPException(

status\_code=400,

detail="Пароль должен содержать минимум 8 символов"

)

# Проверка уникальности логина

existing\_user = db.query(User).filter(User.username == username).first()

if existing\_user:

raise HTTPException(

status\_code=400,

detail="Пользователь с таким логином уже существует"

)

# Хеширование пароля

hashed\_password = pwd\_context.hash(password)

# Создание пользователя

new\_user = User(

full\_name=full\_name,

username=username,

password=hashed\_password,

registration\_date=datetime.utcnow(),

is\_active=True,

is\_admin=False # По умолчанию обычный пользователь

)

try:

db.add(new\_user)

db.commit()

except Exception as e:

db.rollback()

raise HTTPException(

status\_code=500,

detail="Ошибка при создании пользователя"

)

return RedirectResponse(

url="/login?registration=success",

status\_code=303

)

@router.get("/register", response\_class=HTMLResponse)

async def register\_page(request: Request):

return templates.TemplateResponse("auth/register.html", {"request": request})

@router.get("/check-auth")

async def check\_auth(

access\_token: str = Cookie(None),

db: Session = Depends(get\_db)

):

if not access\_token:

raise HTTPException(status\_code=401)

try:

token = access\_token.replace("Bearer ", "")

jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])

return {"status": "authenticated"}

except JWTError:

raise HTTPException(status\_code=401)

@router.post("/logout")

def logout(response: Response,

access\_token: str = Cookie(None)

):

if not access\_token:

raise HTTPException(status\_code=401, detail="Not authenticated")

token = access\_token.replace("Bearer ", "")

try:

# Добавьте токен в blacklist, если нужно

add\_to\_blacklist(token)

except Exception as e:

print("[LOGOUT ERROR]", e)

response = JSONResponse({"message": "You have successfully logged out"})

response.delete\_cookie("access\_token")

return response

1. **/endpoints/**[**user.py**](http://user.py)

from fastapi import APIRouter, Depends, HTTPException, status

from sqlalchemy.orm import Session

from typing import List

from datetime import datetime

from fastapi.security import OAuth2PasswordBearer

from jose import JWTError, jwt

from server.app.api.deps import add\_to\_blacklist

from server.app.dataBase.base import settings

from server.app.dataBase.sessions import get\_db

from server.app.dataBase.models.user import User

from server.app.api.schemas import UserCreate, User as UserSchema, PasswordChange

from server.app.api.deps import get\_current\_user, get\_admin\_user, check\_self\_or\_admin

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="api/token")

router = APIRouter()

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

def create\_user(user: UserCreate, db: Session = Depends(get\_db)):

# Проверяем, существует ли пользователь

existing\_user = db.query(User).filter(User.username == user.username).first()

if existing\_user:

raise HTTPException(

status\_code=400,

detail="Username already registered"

)

# Создаем нового пользователя

db\_user = User(

full\_name=user.full\_name,

username=user.username,

registration\_date=datetime.now(),

)

db\_user.set\_password(user.password)

db.add(db\_user)

db.commit()

db.refresh(db\_user)

return db\_user

@router.get("/", response\_model=List[UserSchema], dependencies=[Depends(get\_admin\_user)])

def read\_users(

skip: int = 0,

limit: int = 100,

db: Session = Depends(get\_db)

):

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

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

def read\_user(

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

current\_user: User = Depends(get\_current\_user)

):

if not current\_user.is\_admin and current\_user.id != user\_id:

raise HTTPException(

status\_code=403,

detail="You can only access your own data"

)

user = db.query(User).filter(User.id == user\_id).first()

if not user:

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

return user

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

def update\_user(

user\_id: int,

user\_data: UserCreate,

db: Session = Depends(get\_db),

current\_user: User = Depends(get\_current\_user)

):

if not current\_user.is\_admin and current\_user.id != user\_id:

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail="You can only update your own profile"

)

db\_user = db.query(User).filter(User.id == user\_id).first()

if not db\_user:

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

# Проверяем, не занят ли username другим пользователем

if user\_data.username != db\_user.username:

existing\_user = db.query(User).filter(

User.username == user\_data.username

).first()

if existing\_user:

raise HTTPException(

status\_code=400,

detail="Username already taken"

)

db\_user.full\_name = user\_data.full\_name

db\_user.username = user\_data.username

if user\_data.password:

db\_user.set\_password(user\_data.password)

db.commit()

db.refresh(db\_user)

return db\_user

@router.delete("/{user\_id}", dependencies=[Depends(get\_admin\_user)])

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

user = db.query(User).filter(User.id == user\_id).first()

if not user:

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

db.delete(user)

db.commit()

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

@router.post("/change-password", response\_model=UserSchema)

def change\_password(

password\_data: PasswordChange,

current\_user: User = Depends(get\_current\_user),

db: Session = Depends(get\_db)

):

# Проверяем старый пароль

if not current\_user.verify\_password(password\_data.old\_password):

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail="Incorrect old password"

)

# Обновляем пароль

current\_user.set\_password(password\_data.new\_password)

db.commit()

db.refresh(current\_user)

return current\_user

@router.post("/logout")

def logout(

token: str = Depends(oauth2\_scheme),

current\_user: User = Depends(get\_current\_user)

):

add\_to\_blacklist(token)

return {"message": "You have successfully logged out"}

1. **/endpoints/psychologist**[**.py**](http://user.py)

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from server.app.dataBase.sessions import get\_db

from server.app.api.schemas import PsychologistCreate, Psychologist

from sqlalchemy import select

from typing import List

from server.app.dataBase.models.psychologist import Psychologist as PsychologistModel

from server.app.api.schemas import Psychologist as PsychologistSchema

router = APIRouter()

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

def create\_psychologist(

psychologist: PsychologistCreate,

db: Session = Depends(get\_db)

):

db\_psychologist = PsychologistModel(\*\*psychologist.dict())

db.add(db\_psychologist)

db.commit()

db.refresh(db\_psychologist)

return db\_psychologist

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

def read\_psychologists(

skip: int = 0,

limit: int = 100,

db: Session = Depends(get\_db)

):

psychologists = db.query(PsychologistModel).offset(skip).limit(limit).all()

return psychologists

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

def read\_psychologist(

psychologist\_id: int,

db: Session = Depends(get\_db)

):

psychologist = db.get(PsychologistModel, psychologist\_id)

if psychologist is None:

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

return psychologist

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

def update\_psychologist(

psychologist\_id: int,

psychologist\_data: PsychologistCreate,

db: Session = Depends(get\_db)

):

db\_psychologist = db.get(PsychologistModel, psychologist\_id)

if db\_psychologist is None:

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

# Обновляем поля

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

setattr(db\_psychologist, key, value)

db.commit()

db.refresh(db\_psychologist)

return db\_psychologist

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

def delete\_psychologist(

psychologist\_id: int,

db: Session = Depends(get\_db)

):

psychologist = db.get(PsychologistModel, psychologist\_id)

if psychologist is None:

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

db.delete(psychologist)

db.commit()

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

1. **/endpoints/session**[**.py**](http://user.py)

from fastapi import APIRouter, Depends, HTTPException, status, Request

from sqlalchemy.orm import Session

from typing import List, Optional

from datetime import datetime

from server.app.dataBase.sessions import get\_db

from server.app.dataBase.models.session import Session as SessionModel

from server.app.dataBase.models.user import User

from server.app.dataBase.models.psychologist import Psychologist

from server.app.api.schemas import SessionCreate, SessionSchema, SessionUpdate, SessionStatus

from server.app.api.deps import get\_current\_user\_from\_cookie # Новый импорт

router = APIRouter(tags=["sessions"])

@router.post("/", response\_model=SessionSchema, status\_code=status.HTTP\_201\_CREATED)

async def create\_session(

request: Request,

session\_data: SessionCreate,

db: Session = Depends(get\_db),

current\_user: User = Depends(get\_current\_user\_from\_cookie)

):

"""

Создание новой терапевтической сессии

- Требуется аутентификация через куки

- Статус по умолчанию: 'pending'

"""

try:

# Проверка существования психолога

psychologist = db.query(Psychologist).get(session\_data.psychologist\_id)

if not psychologist:

raise HTTPException(

status\_code=status.HTTP\_404\_NOT\_FOUND,

detail="Психолог не найден"

)

db\_session = SessionModel(

user\_id=current\_user.id,

psychologist\_id=session\_data.psychologist\_id,

date\_time=session\_data.date\_time,

duration=session\_data.duration,

price=session\_data.price,

status=session\_data.status.value,

notes=session\_data.notes

)

db.add(db\_session)

db.commit()

db.refresh(db\_session)

return db\_session

except ValueError as e:

db.rollback()

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail=str(e)

)

except Exception as e:

db.rollback()

raise HTTPException(

status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR,

detail="Ошибка сервера при создании сессии"

)

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

async def read\_sessions(

request: Request,

skip: int = 0,

limit: int = 100,

status: Optional[SessionStatus] = None,

db: Session = Depends(get\_db),

current\_user: User = Depends(get\_current\_user\_from\_cookie)

):

"""

Получение списка сессий текущего пользователя

- Фильтрация по статусу (опционально)

- Пагинация через skip/limit

"""

query = db.query(SessionModel).filter(SessionModel.user\_id == current\_user.id)

if status:

query = query.filter(SessionModel.status == status.value)

return query.order\_by(SessionModel.date\_time)\

.offset(skip)\

.limit(limit)\

.all()

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

async def read\_session(

request: Request,

session\_id: int,

db: Session = Depends(get\_db),

current\_user: User = Depends(get\_current\_user\_from\_cookie)

):

"""

Получение детальной информации о сессии

- Доступ только для владельца или администратора

"""

session = db.query(SessionModel).get(session\_id)

if not session:

raise HTTPException(

status\_code=status.HTTP\_404\_NOT\_FOUND,

detail="Сессия не найдена"

)

if session.user\_id != current\_user.id and not current\_user.is\_admin:

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail="Нет доступа к этой сессии"

)

return session

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

async def update\_session(

request: Request,

session\_id: int,

session\_data: SessionUpdate,

db: Session = Depends(get\_db),

current\_user: User = Depends(get\_current\_user\_from\_cookie)

):

"""

Обновление информации о сессии

- Доступ только для владельца или администратора

- Частичное обновление разрешено

"""

session = db.query(SessionModel).get(session\_id)

if not session:

raise HTTPException(status\_code=404, detail="Сессия не найдена")

if session.user\_id != current\_user.id and not current\_user.is\_admin:

raise HTTPException(status\_code=403, detail="Нет прав для обновления")

try:

update\_data = session\_data.dict(exclude\_unset=True)

for field, value in update\_data.items():

if field == 'status' and value:

setattr(session, field, value.value)

else:

setattr(session, field, value)

db.commit()

db.refresh(session)

return session

except Exception as e:

db.rollback()

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail=f"Ошибка обновления: {str(e)}"

)

@router.delete("/{session\_id}", status\_code=status.HTTP\_204\_NO\_CONTENT)

async def delete\_session(

request: Request,

session\_id: int,

db: Session = Depends(get\_db),

current\_user: User = Depends(get\_current\_user\_from\_cookie)

):

"""

Удаление сессии

- Доступ только для владельца или администратора

"""

session = db.query(SessionModel).get(session\_id)

if not session:

raise HTTPException(status\_code=404, detail="Сессия не найдена")

if session.user\_id != current\_user.id and not current\_user.is\_admin:

raise HTTPException(status\_code=403, detail="Нет прав для удаления")

try:

db.delete(session)

db.commit()

except Exception as e:

db.rollback()

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail=f"Ошибка удаления: {str(e)}"

)

1. **/endpoints/notification**[**.py**](http://user.py)

from fastapi import APIRouter, Depends, HTTPException

from sqlalchemy.orm import Session

from sqlalchemy import select

from typing import List

from server.app.dataBase.sessions import get\_db

from server.app.dataBase.models.notification import Notification as NotificationModel

from server.app.api.schemas import NotificationCreate, Notification as NotificationSchema

router = APIRouter()

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

def create\_notification(

notification: NotificationCreate,

db: Session = Depends(get\_db)

):

# Создаем уведомление

db\_notification = NotificationModel(

bracelet\_id=notification.bracelet\_id,

session\_id=notification.session\_id,

message\_type=notification.message\_type

)

db.add(db\_notification)

db.commit()

db.refresh(db\_notification)

return db\_notification

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

def read\_notifications(

skip: int = 0,

limit: int = 100,

db: Session = Depends(get\_db)

):

notifications = db.query(NotificationModel).offset(skip).limit(limit).all()

return notifications

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

def read\_notification(

notification\_id: int,

db: Session = Depends(get\_db)

):

notification = db.get(NotificationModel, notification\_id)

if notification is None:

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

return notification

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

def update\_notification(

notification\_id: int,

notification\_data: NotificationCreate,

db: Session = Depends(get\_db)

):

db\_notification = db.get(NotificationModel, notification\_id)

if db\_notification is None:

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

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

setattr(db\_notification, key, value)

db.commit()

db.refresh(db\_notification)

return db\_notification

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

def delete\_notification(

notification\_id: int,

db: Session = Depends(get\_db)

):

notification = db.get(NotificationModel, notification\_id)

if notification is None:

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

db.delete(notification)

db.commit()

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

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

def get\_user\_notifications(

user\_id: int,

db: Session = Depends(get\_db)

):

notifications = db.query(NotificationModel).filter(NotificationModel.bracelet\_id == user\_id).all()

return notifications

1. **/endpoints/bracelet**[**.py**](http://user.py)

from fastapi import APIRouter, Depends, HTTPException, status

from sqlalchemy.orm import Session

from sqlalchemy import select

from typing import List

from server.app.dataBase.sessions import get\_db

from server.app.dataBase.models.bracelet import Bracelet as BraceletModel

from server.app.api.schemas import BraceletCreate, Bracelet as BraceletSchema

router = APIRouter()

@router.post(

"/",

response\_model=BraceletSchema,

status\_code=status.HTTP\_201\_CREATED,

summary="Create a new bracelet"

)

def create\_bracelet(

bracelet: BraceletCreate,

db: Session = Depends(get\_db)

):

"""

Create a new bracelet with:

- \*\*settings\*\*: JSON string with bracelet configuration

- \*\*user\_id\*\*: ID of associated user (must be unique)

"""

try:

db\_bracelet = BraceletModel(\*\*bracelet.dict())

db.add(db\_bracelet)

db.commit()

db.refresh(db\_bracelet)

return db\_bracelet

except Exception as e:

db.rollback()

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail=f"Error creating bracelet: {str(e)}"

)

@router.get(

"/",

response\_model=List[BraceletSchema],

summary="Get list of bracelets"

)

def read\_bracelets(

skip: int = 0,

limit: int = 100,

db: Session = Depends(get\_db)

):

"""

Retrieve bracelets with pagination:

- \*\*skip\*\*: Number of items to skip

- \*\*limit\*\*: Maximum items to return

"""

bracelets = db.query(BraceletModel)\

.order\_by(BraceletModel.id)\

.offset(skip)\

.limit(limit)\

.all()

return bracelets

@router.get(

"/{bracelet\_id}",

response\_model=BraceletSchema,

summary="Get bracelet by ID"

)

def read\_bracelet(

bracelet\_id: int,

db: Session = Depends(get\_db)

):

"""

Get detailed bracelet information by ID

"""

bracelet = db.get(BraceletModel, bracelet\_id)

if not bracelet:

raise HTTPException(

status\_code=status.HTTP\_404\_NOT\_FOUND,

detail="Bracelet not found"

)

return bracelet

@router.put(

"/{bracelet\_id}",

response\_model=BraceletSchema,

summary="Update bracelet information"

)

def update\_bracelet(

bracelet\_id: int,

bracelet\_data: BraceletCreate,

db: Session = Depends(get\_db)

):

"""

Update bracelet configuration by ID

"""

db\_bracelet = db.get(BraceletModel, bracelet\_id)

if not db\_bracelet:

raise HTTPException(

status\_code=status.HTTP\_404\_NOT\_FOUND,

detail="Bracelet not found"

)

try:

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

setattr(db\_bracelet, key, value)

db.commit()

db.refresh(db\_bracelet)

return db\_bracelet

except Exception as e:

db.rollback()

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail=f"Error updating bracelet: {str(e)}"

)

@router.delete(

"/{bracelet\_id}",

status\_code=status.HTTP\_204\_NO\_CONTENT,

summary="Delete a bracelet"

)

def delete\_bracelet(

bracelet\_id: int,

db: Session = Depends(get\_db)

):

"""

Delete bracelet by ID

"""

bracelet = db.get(BraceletModel, bracelet\_id)

if not bracelet:

raise HTTPException(

status\_code=status.HTTP\_404\_NOT\_FOUND,

detail="Bracelet not found"

)

try:

db.delete(bracelet)

db.commit()

except Exception as e:

db.rollback()

raise HTTPException(

status\_code=status.HTTP\_400\_BAD\_REQUEST,

detail=f"Error deleting bracelet: {str(e)}"

)

1. **api/**[**deps.py**](http://deps.py)

from fastapi import Depends, HTTPException, status, Request

from fastapi.security import OAuth2PasswordBearer

from jose import JWTError, jwt

from sqlalchemy.orm import Session

from typing import Annotated, Dict, Optional

from datetime import datetime, timedelta

from pydantic import ValidationError

from fastapi.responses import JSONResponse

from server.app.dataBase.base import settings

from server.app.dataBase.models.user import User

from server.app.dataBase.sessions import get\_db

from server.app.api.schemas import TokenData

# Для JWT-аутентификации через заголовки

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="api/token")

# Для cookie-based аутентификации

def get\_current\_user\_from\_cookie(

request: Request,

db: Session = Depends(get\_db)

) -> User:

"""Получаем пользователя из куки access\_token"""

credentials\_exception = HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Could not validate credentials",

headers={"WWW-Authenticate": "Bearer"},

)

# Получаем куку

cookie = request.cookies.get("access\_token")

if not cookie:

raise credentials\_exception

# Обрабатываем токен

token = None

try:

# Удаляем "Bearer " если есть

token = cookie.replace("Bearer ", "") if cookie.startswith("Bearer ") else cookie

# Проверка на отозванный токен

if is\_token\_revoked(token):

raise credentials\_exception

payload = jwt.decode(

token,

settings.SECRET\_KEY.get\_secret\_value(),

algorithms=[settings.ALGORITHM]

)

print("Декодированный payload:", payload) # Для отладки

username: str = payload.get("sub")

if username is None:

raise credentials\_exception

token\_data = TokenData(username=username)

except (JWTError, ValidationError) as exc:

print(f"[AUTH ERROR] Cookie token validation failed: {exc}")

print(f"Проблемный токен: {token}") # Выводим токен для отладки

raise credentials\_exception

except Exception as exc:

print(f"[UNEXPECTED AUTH ERROR]: {exc}")

raise credentials\_exception

user = db.query(User).filter(User.username == token\_data.username).first()

if user is None:

print(f"[AUTH ERROR] User {token\_data.username} not found")

raise credentials\_exception

if not user.is\_active:

print(f"[AUTH WARNING] Inactive user {user.username} tried to access")

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail="Inactive user"

)

return user

# Черный список токенов

token\_blacklist: Dict[str, datetime] = {}

def is\_token\_revoked(token: str) -> bool:

"""Проверяет, отозван ли токен"""

return token in token\_blacklist

async def get\_current\_user(

token: Annotated[str, Depends(oauth2\_scheme)],

db: Session = Depends(get\_db)

) -> User:

credentials\_exception = HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Could not validate credentials",

headers={"WWW-Authenticate": "Bearer"},

)

try:

# Проверка на отозванный токен

if is\_token\_revoked(token):

raise credentials\_exception

payload = jwt.decode(

token,

settings.SECRET\_KEY.get\_secret\_value(),

algorithms=[settings.ALGORITHM]

)

username: str = payload.get("sub")

if username is None:

raise credentials\_exception

# Валидация через Pydantic

token\_data = TokenData(username=username)

except (JWTError, ValidationError) as exc:

print(f"[AUTH ERROR] Token validation failed: {exc}")

raise credentials\_exception

user = db.query(User).filter(User.username == token\_data.username).first()

if user is None:

print(f"[AUTH ERROR] User {token\_data.username} not found")

raise credentials\_exception

return user

def add\_to\_blacklist(token: str, expires\_delta: Optional[timedelta] = None) -> bool:

"""Добавляет токен в черный список с обработкой ошибок"""

try:

payload = jwt.decode(

token,

settings.SECRET\_KEY.get\_secret\_value(),

algorithms=[settings.ALGORITHM]

)

exp = payload.get("exp")

if exp:

token\_blacklist[token] = datetime.fromtimestamp(exp)

elif expires\_delta:

token\_blacklist[token] = datetime.utcnow() + expires\_delta

else:

token\_blacklist[token] = datetime.utcnow() + timedelta(

minutes=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES

)

return True

except JWTError as exc:

print(f"[BLACKLIST ERROR] Failed to add token: {exc}")

return False

def get\_admin\_user(current\_user: User = Depends(get\_current\_user)) -> User:

"""Проверка прав администратора"""

if not current\_user.is\_admin:

print(f"[AUTH WARNING] User {current\_user.username} attempted admin access")

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail="Admin privileges required"

)

return current\_user

def check\_self\_or\_admin(

user\_id: int,

current\_user: User = Depends(get\_current\_user)

) -> User:

"""Проверка доступа к ресурсу (владелец или администратор)"""

if not current\_user.is\_admin and current\_user.id != user\_id:

print(f"[AUTH WARNING] User {current\_user.username} attempted unauthorized access to user {user\_id}")

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail="You can only access your own account"

)

return current\_user

1. **api/schemas**[**.py**](http://deps.py)

from pydantic import BaseModel, field\_validator, Field

from datetime import datetime

from typing import List, Optional

from pydantic import BaseModel, Field, validator

from enum import Enum

class UserBase(BaseModel):

full\_name: str = Field(..., min\_length=2, max\_length=100)

username: str = Field(..., min\_length=3, max\_length=50, pattern="^[a-zA-Z0-9\_]+$")

password: str

class UserCreate(UserBase):

password: str = Field(..., min\_length=8)

#is\_admin: bool = False

@field\_validator('password')

def validate\_password(cls, v):

if len(v) < 8:

raise ValueError("Password must be at least 8 characters")

return v

class User(UserBase):

id: int

registration\_date: datetime

class Config:

from\_attributes = True

class PasswordChange(BaseModel):

old\_password: str

new\_password: str

class PsychologistBase(BaseModel):

full\_name: str

specialty: str

rating: float

class PsychologistCreate(PsychologistBase):

pass

class Psychologist(PsychologistBase):

id: int

class Config:

from\_attributes = True

class SessionStatus(str, Enum):

PENDING = "pending"

CONFIRMED = "confirmed"

CANCELLED = "cancelled"

COMPLETED = "completed"

class SessionBase(BaseModel):

psychologist\_id: int = Field(..., description="ID психолога")

date\_time: datetime = Field(..., description="Дата и время сеанса")

duration: int = Field(..., gt=0, description="Длительность в минутах")

price: float = Field(..., gt=0, description="Стоимость сеанса")

notes: Optional[str] = Field(None, max\_length=1000, description="Дополнительные заметки")

@validator('date\_time')

def validate\_future\_date(cls, v):

if v < datetime.now():

raise ValueError("Дата сеанса должна быть в будущем")

return v

class SessionCreate(SessionBase):

status: Optional[SessionStatus] = Field(SessionStatus.PENDING, description="Статус сеанса")

class SessionUpdate(BaseModel):

date\_time: Optional[datetime] = None

duration: Optional[int] = Field(None, gt=0)

price: Optional[float] = Field(None, gt=0)

status: Optional[SessionStatus] = None

notes: Optional[str] = None

class SessionSchema(SessionBase):

id: int

user\_id: int

psychologist\_id: int

status: SessionStatus

#created\_at: datetime

class Config:

from\_attributes = True

json\_encoders = {

datetime: lambda v: v.isoformat()

}

class BraceletBase(BaseModel):

settings: str

user\_id: int

class BraceletCreate(BraceletBase):

pass

class Bracelet(BraceletBase):

id: int

class Config:

from\_attributes = True

class NotificationBase(BaseModel):

bracelet\_id: int

session\_id: int

message\_type: str

class NotificationCreate(NotificationBase):

pass

class Notification(NotificationBase):

id: int

class Config:

from\_attributes = True

class Token(BaseModel):

access\_token: str

token\_type: str

class TokenData(BaseModel):

username: Optional[str] = None

1. **dataBase/base**[**.py**](http://deps.py)

from sqlalchemy.ext.declarative import declarative\_base

from pydantic\_settings import BaseSettings

from datetime import timedelta

from pydantic import SecretStr

import os

from dotenv import load\_dotenv

# Загружаем переменные окружения из .env файла

load\_dotenv()

# Базовый класс для моделей SQLAlchemy

Base = declarative\_base()

# Настройки JWT

class Settings(BaseSettings):

SECRET\_KEY: SecretStr

ALGORITHM: str = os.getenv("ALGORITHM", "HS256")

ACCESS\_TOKEN\_EXPIRE\_MINUTES: int = int(os.getenv("ACCESS\_TOKEN\_EXPIRE\_MINUTES", 30))

REFRESH\_TOKEN\_EXPIRE\_DAYS: int = int(os.getenv("REFRESH\_TOKEN\_EXPIRE\_DAYS", 7))

class Config:

case\_sensitive = True

env\_file = ".env"

env\_file\_encoding = "utf-8"

extra = "forbid" # Запретит неизвестные переменные

settings = Settings()

def get\_token\_expires():

return timedelta(minutes=settings.ACCESS\_TOKEN\_EXPIRE\_MINUTES)

1. **dataBase/sessions**[**.py**](http://deps.py)

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker, Session

from .base import Base

# Синхронный URL подключения

DATABASE\_URL = "postgresql://postgres:123@localhost:5433/DataBase"

# Создаем синхронный engine

engine = create\_engine(

DATABASE\_URL,

pool\_size=20,

max\_overflow=10,

pool\_timeout=30,

echo=True # Логирование SQL-запросов

)

# Создаем фабрику сессий

SessionLocal = sessionmaker(

bind=engine,

autocommit=False,

autoflush=False,

expire\_on\_commit=False

)

def get\_db():

"""

Генератор синхронной сессии для зависимостей.

Использование:

db: Session = Depends(get\_db)

"""

db = SessionLocal()

try:

yield db

finally:

db.close()

1. **middleware/auth\_middleware**[**.py**](http://deps.py)

from fastapi import Request, HTTPException

from fastapi.responses import JSONResponse

from jose import JWTError, jwt

from typing import Optional, List

from starlette.middleware.base import BaseHTTPMiddleware

import re

from server.app.dataBase.base import settings

from server.app.api.deps import is\_token\_revoked

class AuthMiddleware(BaseHTTPMiddleware):

def \_\_init\_\_(self, app, exempt\_routes: Optional[List[str]] = None):

super().\_\_init\_\_(app)

self.exempt\_routes = exempt\_routes or [

"/",

"/docs",

"/redoc",

"/openapi.json",

"/api/token",

"/api/register",

"/login",

"/register",

r"/static/.\*", # Регулярка для статических файлов

r"/favicon\.ico"

]

self.compiled\_patterns = [re.compile(pattern) for pattern in self.exempt\_routes]

async def dispatch(self, request: Request, call\_next):

path = request.url.path

# Пропускаем exempt routes

if any(pattern.fullmatch(path) for pattern in self.compiled\_patterns):

return await call\_next(request)

token = self.extract\_token(request)

if not token:

raise HTTPException(

status\_code=401,

detail="Not authenticated",

headers={"WWW-Authenticate": "Bearer"}

)

if is\_token\_revoked(token):

raise HTTPException(

status\_code=401,

detail="Token revoked"

)

try:

payload = jwt.decode(

token,

settings.SECRET\_KEY.get\_secret\_value(),

algorithms=[settings.ALGORITHM]

)

request.state.user\_id = payload.get("user\_id") # Более полезно чем sub

request.state.is\_admin = payload.get("is\_admin", False)

except JWTError as e:

raise HTTPException(

status\_code=401,

detail=f"Invalid token: {str(e)}"

)

return await call\_next(request)

def extract\_token(self, request: Request) -> Optional[str]:

"""Извлекает токен из кук или заголовка"""

# 1. Проверяем куки

if "access\_token" in request.cookies:

cookie = request.cookies["access\_token"]

if cookie.startswith("Bearer "):

return cookie[7:] # Удаляем "Bearer "

return cookie

# 2. Проверяем Authorization header

auth\_header = request.headers.get("Authorization")

if auth\_header and auth\_header.startswith("Bearer "):

return auth\_header.split(" ")[1]

return None

1. **templates/index.html**

<!-- templates/index.html -->

{% extends "base.html" %}

{% block content %}

<section class="banner">

<div class="container banner\_\_container" style="display: flex; align-items: center; gap: 24px;">

<div class="banner\_\_content" style="flex: 1;">

<h1 class="banner\_\_title">Консультации<br> с психологом онлайн:<br>

<marquee behavior="scroll" direction="up" scrollamount="2" class="slow-blue-marquee">

повысить качество жизни<br>

справиться со стрессом<br>

наладить отношения с близкими<br>

найти своё призвание<br>

принять себя<br>

обрести спокойствие

</marquee>

</h1>

</div>

<div class="banner\_\_image" style="flex-shrink: 0;">

<img src="\static\images\1.jpg" alt="Психолог"

style="max-width: 600px; width: 100%; height: auto; border-radius: 12px;">

</div>

</div>

</section>

<section class="psycho-help">

<div class="container">

<h2 class="section-title">С чем <span>поможет</span> психолог?</h2>

<div class="help-grid">

<!-- Первая строка -->

<div class="help-row">

<!-- Колонка 1 -->

<div class="help-card">

<div class="help-image">

<img src="/static/images/2.png" alt="Отношения">

</div>

<div class="help-content">

<h3>Гармоничные отношения</h3>

<p>Наладить отношения с близкими людьми</p>

</div>

</div>

<!-- Колонка 2 -->

<div class="help-card">

<div class="help-image">

<img src="/static/images/3.png" alt="Стресс">

</div>

<div class="help-content">

<h3>Стресс и тревога</h3>

<p>Справиться со стрессом, тревогой и страхом</p>

</div>

</div>

<!-- Колонка 3 -->

<div class="help-card">

<div class="help-image">

<img src="/static/images/4.png" alt="Самооценка">

</div>

<div class="help-content">

<h3>Принятие себя</h3>

<p>Принять себя и повысить самооценку</p>

</div>

</div>

</div>

<!-- Вторая строка -->

<div class="help-row">

<!-- Колонка 1 -->

<div class="help-card">

<div class="help-image">

<img src="/static/images/5.png" alt="Границы">

</div>

<div class="help-content">

<h3>Личные границы</h3>

<p>Научиться говорить "нет" и отстаивать себя</p>

</div>

</div>

<!-- Колонка 2 -->

<div class="help-card">

<div class="help-image">

<img src="/static/images/6.png" alt="Перемены">

</div>

<div class="help-content">

<h3>Жизненные перемены</h3>

<p>Пережить переезд, расставание, потерю работы</p>

</div>

</div>

<!-- Колонка 3 -->

<div class="help-card">

<div class="help-image">

<img src="/static/images/7.png" alt="Работа">

</div>

<div class="help-content">

<h3>Рабочие вопросы</h3>

<p>Понять причины проблем и отсутствия мотивации</p>

</div>

</div>

</div>

</div>

</div>

</section>

{% endblock %}

1. **templates/base.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>{% block title %}Psychologist Sessions{% endblock %}</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">

<!-- Подключение CSS -->

<link rel="stylesheet" href="/static/css/styles.css">

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container">

<a class="navbar-brand" href="/">PsychoApp</a>

<div class="navbar-nav">

<a class="nav-link" href="/login">Вход</a>

<a class="nav-link" href="/register">Регистрация</a>

</div>

</div>

</nav>

<div class="container mt-4">

{% block content %}{% endblock %}

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>

<!-- Скрипт для автоматической подстановки токена -->

<script>

const token = localStorage.getItem('access\_token');

if (token) {

const originalFetch = window.fetch;

window.fetch = async (url, options = {}) => {

options.headers = {

...options.headers,

'Authorization': `Bearer ${token}`

};

return originalFetch(url, options);

};

}

</script>

{% block scripts %}{% endblock %}

</body>

</html>

1. **templates/dashboard.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Панель клиента | Мои сессии</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/flatpickr/dist/flatpickr.min.css">

<style>

.badge-rating { background-color: #ffc107; color: #000; }

.session-card { transition: all 0.3s ease; }

.session-card:hover { transform: translateY(-3px); box-shadow: 0 5px 15px rgba(0,0,0,0.1); }

.add-session-btn {

position: fixed; bottom: 30px; right: 30px;

width: 60px; height: 60px; border-radius: 50%;

font-size: 24px; box-shadow: 0 4px 8px rgba(0,0,0,0.2);

}

</style>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-4">

<div class="container">

<a class="navbar-brand" href="#">Панель клиента</a>

<button id="logout-btn" class="btn btn-outline-light ms-auto">Выйти</button>

</div>

</nav>

<div class="container">

<h1 class="mb-4">Добро пожаловать, {{ user.full\_name }}!</h1>

<div class="container banner\_\_container" style="display: flex; align-items: center; gap: 24px;">

<div class="col-md-8">

<div class="card">

<div class="card-header d-flex justify-content-between align-items-center">

<h5 class="mb-0">Ваши сессии</h5>

<button class="btn btn-sm btn-outline-primary" data-bs-toggle="modal" data-bs-target="#addSessionModal">

Добавить сессию

</button>

</div>

<div class="card-body">

{% if sessions %}

<div class="list-group">

{% for session in sessions %}

<div class="list-group-item session-card mb-2">

<div class="d-flex justify-content-between">

<div>

<h6>{{ session.date\_time.strftime('%d.%m.%Y %H:%M') }}</h6>

<small class="text-muted">Психолог: {{ session.psychologist.full\_name }}</small>

</div>

<div>

<span class="badge bg-{{ 'success' if session.status == 'confirmed' else 'warning' }}">

{{ session.status }}

</span>

</div>

</div>

<div class="mt-2">

<small>Длительность: {{ session.duration }} мин.</small><br>

<small>Цена: {{ session.price }} руб.</small>

</div>

</div>

{% endfor %}

</div>

{% else %}

<div class="alert alert-info">У вас пока нет запланированных сессий</div>

{% endif %}

</div>

</div>

</div>

<div class="banner\_\_image" style="flex-shrink: 0;">

<img src="\static\images\9.jfif" alt="Психолог"

style="max-width: 300px; width: 100%; height: auto; border-radius: 12px;">

</div>

</div>

</div>

<div class="modal fade" id="addSessionModal" tabindex="-1" aria-hidden="true">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Запись на консультацию</h5>

<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div class="modal-body">

<form id="addSessionForm">

<div class="mb-3">

<label for="psychologistSelect" class="form-label">Психолог</label>

<select class="form-select" id="psychologistSelect" required>

<option value="" selected disabled>Выберите психолога</option>

</select>

</div>

<div class="mb-3">

<label for="sessionDate" class="form-label">Дата и время</label>

<input type="datetime-local" class="form-control" id="sessionDate" required>

</div>

<div class="mb-3">

<label for="duration" class="form-label">Длительность (минуты)</label>

<select class="form-select" id="duration" required>

<option value="30">30 минут</option>

<option value="45">45 минут</option>

<option value="60" selected>1 час</option>

<option value="90">1.5 часа</option>

</select>

</div>

<div class="mb-3">

<label for="notes" class="form-label">Примечания</label>

<textarea class="form-control" id="notes" rows="3"></textarea>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Отмена</button>

<button type="button" class="btn btn-primary" id="submitSessionBtn">Записаться</button>

</div>

</div>

</div>

</div>

<button class="btn btn-primary add-session-btn d-md-none" data-bs-toggle="modal" data-bs-target="#addSessionModal">+</button>

<div id="toastContainer" class="position-fixed bottom-0 end-0 p-3" style="z-index: 9999;"></div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/flatpickr"></script>

<script src="https://cdn.jsdelivr.net/npm/flatpickr/dist/l10n/ru.js"></script>

<script>

document.addEventListener('DOMContentLoaded', () => {

const greetings = ["Привет!","Будь счастлив!","Хорошего дня!","Улыбнись — сегодня отличный день!","Желаем удачи!","Ты молодец, так держать!","Пусть всё складывается отлично!"];

const randomIndex = Math.floor(Math.random() \* greetings.length);

showToast(greetings[randomIndex], 'primary');

flatpickr("#sessionDate", {

enableTime: true,

dateFormat: "Y-m-d H:i",

time\_24hr: true,

minDate: "today",

locale: "ru"

});

checkAuth();

document.getElementById('logout-btn')?.addEventListener('click', logout);

loadPsychologists();

document.getElementById('submitSessionBtn')?.addEventListener('click', createSession);

});

async function checkAuth() {

try {

const response = await fetch('/api/check-auth', { credentials: 'include' });

if (!response.ok) throw new Error('Not authenticated');

const data = await response.json();

if (data.status !== "authenticated") redirectToLogin();

} catch (error) {

redirectToLogin();

}

}

async function loadPsychologists() {

try {

const response = await fetch('/api/psychologists', { credentials: 'include' });

if (!response.ok) throw new Error("Ошибка загрузки");

const data = await response.json();

const select = document.getElementById('psychologistSelect');

select.innerHTML = '<option value="" selected disabled>Выберите психолога</option>';

data.forEach(psych => {

const option = document.createElement('option');

option.value = psych.id;

option.textContent = `${psych.full\_name} (${psych.specialization})`;

select.appendChild(option);

});

} catch (error) {

showToast('Не удалось загрузить список психологов', 'danger');

}

}

async function createSession() {

const psychologistId = document.getElementById('psychologistSelect').value;

const dateTime = document.getElementById('sessionDate').value;

const duration = parseInt(document.getElementById('duration').value);

const notes = document.getElementById('notes').value;

if (!psychologistId || !dateTime || !duration) {

showToast('Заполните все поля', 'warning');

return;

}

const prices = {30: 1500, 45: 1800, 60: 2000, 90: 2800};

const price = prices[duration];

if (!price) {

showToast('Неверная длительность', 'danger');

return;

}

try {

const response = await fetch('/api/sessions/', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

credentials: 'include', // Это обязательно!

body: JSON.stringify({

psychologist\_id: parseInt(psychologistId),

date\_time: dateTime,

duration: duration,

price: price,

status: 'pending',

notes: notes

})

});

if (!response.ok) {

const error = await response.json();

throw new Error(error.detail || 'Ошибка сервера');

}

showToast('Сессия создана успешно!', 'success');

setTimeout(() => window.location.reload(), 1500);

} catch (error) {

showToast(error.message || 'Ошибка создания сессии', 'danger');

if (error.message.includes('Unauthorized')) {

setTimeout(() => window.location.href = '/login', 2000);

}

}

}

async function logout() {

try {

const response = await fetch('/api/logout', {

method: 'POST',

credentials: 'include'

});

if (response.ok) redirectToLogin();

else showToast('Ошибка выхода', 'danger');

} catch (error) {

showToast('Ошибка сети', 'danger');

}

}

function redirectToLogin() {

window.location.href = '/login?error=session\_expired';

}

function showToast(message, type = 'success') {

const toastContainer = document.getElementById('toastContainer');

const toast = document.createElement('div');

toast.className = `toast align-items-center text-white bg-${type} border-0 mb-2`;

toast.setAttribute('role', 'alert');

toast.innerHTML = `

<div class="d-flex">

<div class="toast-body">${message}</div>

<button type="button" class="btn-close btn-close-white me-2 m-auto" data-bs-dismiss="toast"></button>

</div>

`;

toastContainer.appendChild(toast);

new bootstrap.Toast(toast, { delay: 5000 }).show();

}

</script>

</body>

</html>

1. **templates/admin\_dashboard.html**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Админ-панель | Управление психологами</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">

<style>

.badge-rating {

background-color: #ffc107;

color: #000;

}

</style>

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-4">

<div class="container">

<a class="navbar-brand" href="#">Админ-панель</a>

<button id="logout-btn" class="btn btn-outline-light ms-auto">Выйти</button>

</div>

</nav>

<div class="container">

<h1 class="mb-4">Управление психологами</h1>

<div class="row">

<div class="col-md-6 mb-4">

<div class="card">

<div class="card-header bg-primary text-white">

Добавить психолога

</div>

<div class="card-body">

<form id="add-psychologist-form">

<div class="mb-3">

<label for="full\_name" class="form-label">ФИО</label>

<input type="text" class="form-control" id="full\_name" required>

</div>

<div class="mb-3">

<label for="specialty" class="form-label">Специализация</label>

<input type="text" class="form-control" id="specialty" required>

</div>

<div class="mb-3">

<label for="rating" class="form-label">Рейтинг (0-5)</label>

<input type="number" class="form-control" id="rating" min="0" max="5" step="0.1"

required>

</div>

<button type="submit" class="btn btn-primary">Добавить</button>

</form>

</div>

</div>

</div>

<div class="col-md-6">

<div class="card">

<div class="card-header bg-primary text-white">

Список психологов

</div>

<div class="card-body">

<div class="table-responsive">

<table class="table">

<thead>

<tr>

<th>ФИО</th>

<th>Специализация</th>

<th>Рейтинг</th>

<th>Действия</th>

</tr>

</thead>

<tbody id="psychologists-list">

<!-- Данные будут загружены через JS -->

</tbody>

</table>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<!-- Модальное окно редактирования -->

<div class="modal fade" id="editModal" tabindex="-1" aria-hidden="true">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Редактировать психолога</h5>

<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div class="modal-body">

<form id="edit-form">

<input type="hidden" id="edit-id">

<div class="mb-3">

<label for="edit-full\_name" class="form-label">ФИО</label>

<input type="text" class="form-control" id="edit-full\_name" required>

</div>

<div class="mb-3">

<label for="edit-specialty" class="form-label">Специализация</label>

<input type="text" class="form-control" id="edit-specialty" required>

</div>

<div class="mb-3">

<label for="edit-rating" class="form-label">Рейтинг</label>

<input type="number" class="form-control" id="edit-rating" min="0" max="5" step="0.1"

required>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Отмена</button>

<button type="button" class="btn btn-primary" id="save-changes">Сохранить</button>

</div>

</div>

</div>

</div>

<div class="modal fade" id="editClientModal" tabindex="-1" aria-hidden="true">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-footer">

<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Отмена</button>

<button type="button" class="btn btn-primary" id="save-client-changes">Сохранить</button>

</div>

</div>

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js">

</script>

<script>

document.addEventListener('DOMContentLoaded', () => {

const editModal = new bootstrap.Modal('#editModal');

const editClientModal = new bootstrap.Modal('#editClientModal');

// Загрузка списка психологов

async function loadPsychologists() {

try {

const response = await fetch('/api/psychologists/');

const psychologists = await response.json();

renderPsychologists(psychologists);

} catch (error) {

console.error('Ошибка загрузки:', error);

}

}

// Отображение списка психологов

function renderPsychologists(psychologists) {

const tbody = document.getElementById('psychologists-list');

tbody.innerHTML = '';

psychologists.forEach(psychologist => {

const tr = document.createElement('tr');

tr.innerHTML = `

<td>${psychologist.full\_name}</td>

<td>${psychologist.specialty}</td>

<td><span class="badge badge-rating">${psychologist.rating.toFixed(1)}</span></td>

<td>

<button class="btn btn-sm btn-outline-primary edit-btn" data-id="${psychologist.id}">

Редакт.

</button>

<button class="btn btn-sm btn-outline-danger delete-btn" data-id="${psychologist.id}">

Удалить

</button>

</td>

`;

tbody.appendChild(tr);

});

document.querySelectorAll('.edit-btn').forEach(btn => {

btn.addEventListener('click', () => openEditForm(btn.dataset.id));

});

document.querySelectorAll('.delete-btn').forEach(btn => {

btn.addEventListener('click', () => deletePsychologist(btn.dataset.id));

});

}

// Добавление нового психолога

document.getElementById('add-psychologist-form').addEventListener('submit', async (e) => {

e.preventDefault();

const psychologist = {

full\_name: document.getElementById('full\_name').value,

specialty: document.getElementById('specialty').value,

rating: parseFloat(document.getElementById('rating').value)

};

try {

const response = await fetch('/api/psychologists/', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(psychologist)

});

if (response.ok) {

document.getElementById('add-psychologist-form').reset();

loadPsychologists();

}

} catch (error) {

console.error('Ошибка:', error);

}

});

// Открытие формы редактирования психолога

async function openEditForm(id) {

try {

const response = await fetch(`/api/psychologists/${id}`);

const psychologist = await response.json();

document.getElementById('edit-id').value = psychologist.id;

document.getElementById('edit-full\_name').value = psychologist.full\_name;

document.getElementById('edit-specialty').value = psychologist.specialty;

document.getElementById('edit-rating').value = psychologist.rating;

editModal.show();

} catch (error) {

console.error('Ошибка:', error);

}

}

// Сохранение изменений психолога

document.getElementById('save-changes').addEventListener('click', async () => {

const id = document.getElementById('edit-id').value;

const full\_name = document.getElementById('edit-full\_name').value;

const specialty = document.getElementById('edit-specialty').value;

const rating = parseFloat(document.getElementById('edit-rating').value);

try {

const response = await fetch(`/api/psychologists/${id}`, {

method: 'PUT',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ full\_name, specialty, rating })

});

if (response.ok) {

const editModalInstance = bootstrap.Modal.getInstance(document.getElementById('editModal'));

editModalInstance.hide();

loadPsychologists();

} else {

const errorData = await response.json();

alert('Ошибка: ' + (errorData.detail || 'Не удалось сохранить изменения'));

}

} catch (error) {

console.error('Ошибка при обновлении психолога:', error);

alert('Произошла ошибка при сохранении');

}

});

// Удаление психолога

async function deletePsychologist(id) {

if (!confirm('Удалить психолога?')) return;

try {

const response = await fetch(`/api/psychologists/${id}`, {

method: 'DELETE'

});

if (response.ok) {

loadPsychologists();

}

} catch (error) {

console.error('Ошибка:', error);

}

}

// Выход из системы

const logoutBtn = document.getElementById('logout-btn');

if (logoutBtn) {

logoutBtn.addEventListener('click', async () => {

try {

const response = await fetch('/api/logout', {

method: 'POST',

credentials: 'include'

});

if (response.ok) {

window.location.href = '/login';

} else {

alert('Ошибка при выходе');

}

} catch (error) {

console.error('Ошибка выхода:', error);

alert('Ошибка сети при выходе');

}

});

}

// Первоначальная загрузка данных

loadPsychologists();

});

</script>

</body>

</html>

1. **templates/auth/login.html**

{% extends "base.html" %}

{% block content %}

<div class="container mt-5">

<div class="row justify-content-center">

<div class="col-md-6">

<h2 class="mb-4">Вход в систему</h2>

{% if request.query\_params.get('error') == 'session\_expired' %}

<div class="alert alert-warning">

Ваша сессия истекла. Пожалуйста, войдите снова.

</div>

{% endif %}

{% if request.query\_params.get('registration') == 'success' %}

<div class="alert alert-success">

Регистрация прошла успешно! Теперь вы можете войти.

</div>

{% endif %}

{% if error %}

<div class="alert alert-danger">{{ error }}</div>

{% endif %}

<form id="login-form">

<div class="mb-3">

<label for="username" class="form-label">Логин</label>

<input type="text" class="form-control" id="username" name="username" required>

</div>

<div class="mb-3">

<label for="password" class="form-label">Пароль</label>

<input type="password" class="form-control" id="password" name="password" required>

</div>

<button type="submit" class="btn btn-primary">Войти</button>

</form>

<div id="error-message" class="text-danger mt-2"></div>

<div class="mt-3">

Нет аккаунта? <a href="/register">Зарегистрируйтесь</a>

</div>

</div>

</div>

</div>

{% endblock %}

{% block scripts %}

<script>

document.addEventListener('DOMContentLoaded', function() {

const loginForm = document.getElementById('login-form');

const errorMessageDiv = document.getElementById('error-message');

loginForm.addEventListener('submit', async function(e) {

e.preventDefault();

errorMessageDiv.textContent = '';

const username = document.getElementById('username').value;

const password = document.getElementById('password').value;

try {

const response = await fetch('/api/token', {

method: 'POST',

headers: {

'Content-Type': 'application/x-www-form-urlencoded',

},

body: new URLSearchParams({ username, password }),

credentials: 'include' // Важно для cookies

});

if (response.redirected) {

window.location.href = response.url;

return;

}

if (!response.ok) {

const errorData = await response.json();

errorMessageDiv.textContent = errorData.detail || 'Неверный логин или пароль';

return;

}

const data = await response.json();

// Устанавливаем cookie вручную, если сервер не делает это автоматически

document.cookie = `access\_token=${data.access\_token}; path=/; max-age=${30 \* 60}; SameSite=Lax`;

// Редирект на нужную страницу

if (data.is\_admin) {

window.location.href = '/api/admin/dashboard';

} else {

window.location.href = '/api/user/dashboard';

}

} catch (err) {

console.error('Ошибка входа:', err);

errorMessageDiv.textContent = 'Ошибка соединения с сервером';

}

});

});

</script>

{% endblock %}

1. **templates/auth/register.html**

{% extends "base.html" %}

{% block content %}

<div class="container mt-5">

<div class="row justify-content-center">

<div class="col-md-6">

<h2 class="mb-4">Регистрация</h2>

{% if error %}

<div class="alert alert-danger">{{ error }}</div>

{% endif %}

<form id="register-form" method="post" action="/api/register">

<div class="mb-3">

<label for="full\_name" class="form-label">Полное имя</label>

<input type="text" class="form-control" id="full\_name" name="full\_name" required>

</div>

<div class="mb-3">

<label for="username" class="form-label">Логин</label>

<input type="text" class="form-control" id="username" name="username" required>

<div id="username-error" class="invalid-feedback"></div>

</div>

<div class="mb-3">

<label for="password" class="form-label">Пароль</label>

<input type="password" class="form-control" id="password" name="password" required minlength="8">

<div class="form-text">Пароль должен содержать минимум 8 символов</div>

<div id="password-error" class="invalid-feedback"></div>

</div>

<button type="submit" class="btn btn-primary">Зарегистрироваться</button>

</form>

<div class="mt-3">

Уже есть аккаунт? <a href="/login">Войдите</a>

</div>

</div>

</div>

</div>

{% endblock %}

{% block scripts %}

<script>

document.addEventListener('DOMContentLoaded', function() {

const registerForm = document.getElementById('register-form');

const usernameInput = document.getElementById('username');

const passwordInput = document.getElementById('password');

registerForm.addEventListener('submit', async function(e) {

e.preventDefault();

// Клиентская валидация

let isValid = true;

// Проверка пароля

if (passwordInput.value.length < 8) {

passwordInput.classList.add('is-invalid');

document.getElementById('password-error').textContent = 'Пароль должен содержать минимум 8 символов';

isValid = false;

} else {

passwordInput.classList.remove('is-invalid');

}

if (!isValid) return;

try {

const formData = new FormData(registerForm);

const response = await fetch('/api/register', {

method: 'POST',

body: new URLSearchParams(formData),

headers: {

'Content-Type': 'application/x-www-form-urlencoded'

}

});

if (response.redirected) {

window.location.href = response.url;

return;

}

if (!response.ok) {

const errorData = await response.json();

if (errorData.detail.includes('логин')) {

usernameInput.classList.add('is-invalid');

document.getElementById('username-error').textContent = errorData.detail;

} else {

alert(errorData.detail);

}

return;

}

// Успешная регистрация

window.location.href = '/login?registration=success';

} catch (error) {

console.error('Registration error:', error);

alert('Ошибка соединения с сервером');

}

});

// Сброс ошибок при вводе

usernameInput.addEventListener('input', function() {

if (this.classList.contains('is-invalid')) {

this.classList.remove('is-invalid');

}

});

passwordInput.addEventListener('input', function() {

if (this.classList.contains('is-invalid')) {

this.classList.remove('is-invalid');

}

});

});

</script>

{% endblock %}

1. **migrations/env.py**

import sys

from os.path import abspath, dirname

from logging.config import fileConfig

from alembic import context

from sqlalchemy import engine\_from\_config, pool

# Добавляем путь к проекту в PYTHONPATH

sys.path.insert(0, dirname(dirname(abspath(\_\_file\_\_))))

# Эти импорты должны быть после добавления пути в sys.path

from app.dataBase.base import Base

from app.dataBase.models.user import User # Явный импорт всех моделей

from app.dataBase.models.psychologist import Psychologist

from app.dataBase.models.bracelet import Bracelet

from app.dataBase.models.session import Session

from app.dataBase.models.notification import Notification

# Это важно для autogenerate

target\_metadata = Base.metadata

# Конфигурация логгера (обычно уже есть в сгенерированном файле)

fileConfig(context.config.config\_file\_name)

def run\_migrations\_offline():

"""Запуск миграций в offline-режиме."""

url = context.config.get\_main\_option("sqlalchemy.url")

context.configure(

url=url,

target\_metadata=target\_metadata,

literal\_binds=True,

dialect\_opts={"paramstyle": "named"},

compare\_type=True,

compare\_server\_default=True,

)

with context.begin\_transaction():

context.run\_migrations()

def run\_migrations\_online():

"""Запуск миграций в online-режиме."""

connectable = engine\_from\_config(

context.config.get\_section(context.config.config\_ini\_section),

prefix="sqlalchemy.",

poolclass=pool.NullPool,

)

with connectable.connect() as connection:

context.configure(

connection=connection,

target\_metadata=target\_metadata,

compare\_type=True,

compare\_server\_default=True,

include\_schemas=True # Если используете схемы

)

with context.begin\_transaction():

context.run\_migrations()

if context.is\_offline\_mode():

run\_migrations\_offline()

else:

run\_migrations\_online()

1. **templates/index.html**