МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №44

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Д. В. Куртяник |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| СОЗДАНИЕ API ДЛЯ БРОНИРОВАНИЯ |
| по дисциплине: Основы программирования |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4319 |  | 14.12.2024 |  | А. М. Емельянов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕОшибка: источник перекрёстной ссылки не найден](#_Toc516226212)

* 1. Анализ требований к API для бронирования 4
  2. Проектирование архитектуры API
  3. Реализация API для бронирования

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А. Примеры запросов и ответов API (в формате JSON)

ПРИЛОЖЕНИЕ Б. SQL-запросы для работы с базой данных

ПРИЛОЖЕНИЕ В. Примеры кода эндпоинтов API

**ВВЕДЕНИЕ**

Современные технологии играют ключевую роль в автоматизации процессов во многих сферах человеческой деятельности. Одной из таких сфер является бронирование — будь то гостиницы, столики в ресторанах или аренда строительных инструментов. Успешное управление такими процессами требует создания эффективных и надёжных решений, обеспечивающих быструю обработку данных и удобный доступ для пользователей.

API (Application Programming Interface) — это программный интерфейс, который позволяет взаимодействовать между различными системами. Создание API для бронирования предоставляет возможность централизовать обработку запросов, интегрировать внешние системы и улучшить пользовательский опыт за счёт высокой скорости и доступности данных. Такие интерфейсы позволяют автоматизировать бронирование, снижая вероятность ошибок и повышая оперативность работы.

Целью данной курсовой работы является разработка программного интерфейса (API), который обеспечит выполнение ключевых задач в системе бронирования. В рамках работы будут рассмотрены современные подходы к проектированию API, а также продемонстрирован процесс разработки, тестирования и развертывания решения.

Создание API для бронирования требует понимания архитектуры веб-приложений, работы с базами данных, а также использования современных инструментов и фреймворков. В качестве основы будет использован язык программирования Python с применением фреймворка FastAPI, который зарекомендовал себя как надёжное и производительное решение для создания RESTful API.

Практическое значение данной работы заключается в том, что разработанное API может быть адаптировано для различных систем бронирования, а его функционал легко расширяется под требования конкретных задач. Итоговый результат будет представлен в виде полностью функционирующего интерфейса с задокументированными эндпоинтами, поддерживающего основные операции CRUD (Create, Read, Update, Delete).

1 АНАЛИЗ ТРЕБОВАНИЙ К API ДЛЯ БРОНИРОВАНИЯ

1.1 Определение функциональных и нефункциональных требований

Функциональные требования:

- Возможность создания нового бронирования;

- Получение информации о существующих бронированиях;

- Изменение деталей бронирования;

- Удаление бронирования;

- Авторизация пользователей.

Нефункциональные требования:

- Быстродействие при обработке запросов;

- Надёжность и отказоустойчивость системы;

- Удобство интеграции с другими системами.

#### 1.2 Постановка задач и определение ограничений

Основные задачи:

- Разработать структуру базы данных для хранения данных о бронированиях;

- Реализовать маршруты API для выполнения операций с бронированиями;

- Обеспечить безопасность данных через авторизацию и валидацию запросов.

Ограничения:

- Используемый стек технологий — Python, FastAPI, PostgreSQL.

- Проект ориентирован на локальную среду разработки.

---

### 2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ API

#### 2.1 Выбор технологий и инструментов разработки

Для реализации API выбраны следующие инструменты:

- \*\*FastAPI\*\* — фреймворк для создания API на Python;

- \*\*PostgreSQL\*\* — реляционная база данных для хранения данных;

- \*\*SQLAlchemy\*\* — инструмент для работы с базой данных;

- \*\*Docker\*\* — для контейнеризации приложения.

#### 2.2 Проектирование структуры данных и базы данных

Основные сущности:

- Пользователь (User): включает данные о пользователях, такие как имя, email и пароль;

- Бронирование (Booking): включает информацию о бронированиях, такие как дата, время, пользователь и статус.

Пример структуры базы данных:

```sql

CREATE TABLE users (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100) UNIQUE,

password VARCHAR(255)

);

CREATE TABLE bookings (

id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id),

date TIMESTAMP,

status VARCHAR(50)

);

```

[Вставь сюда структуру своей базы данных, если она отличается]

#### 2.3 Разработка схемы маршрутов (эндпоинтов)

Пример маршрутов:

- `POST /bookings` — создание нового бронирования;

- `GET /bookings/{id}` — получение информации о бронировании по ID;

- `PUT /bookings/{id}` — обновление бронирования;

- `DELETE /bookings/{id}` — удаление бронирования.

[Опиши свои эндпоинты, если они отличаются]

---

### 3. РЕАЛИЗАЦИЯ API ДЛЯ БРОНИРОВАНИЯ

#### 3.1 Настройка и развертывание окружения

1. Установите необходимые зависимости:

```bash

pip install fastapi uvicorn sqlalchemy psycopg2

```

2. Настройте Docker Compose для работы с PostgreSQL:

```yaml

version: '3.9'

services:

db:

image: postgres:15

environment:

POSTGRES\_USER: user

POSTGRES\_PASSWORD: password

POSTGRES\_DB: bookings

ports:

- '5432:5432'

```

[Добавь сюда свой конфигурационный файл Docker Compose, если он отличается]

#### 3.2 Реализация основных модулей API

##### 3.2.1 Эндпоинт для создания бронирования

```python

from fastapi import FastAPI, Depends

from sqlalchemy.orm import Session

from database import get\_db

app = FastAPI()

@app.post("/bookings")

def create\_booking(booking: BookingCreate, db: Session = Depends(get\_db)):

# [Вставь сюда свой код для создания бронирования]

pass

```

##### 3.2.2 Эндпоинт для получения информации о бронировании

```python

@app.get("/bookings/{id}")

def get\_booking(id: int, db: Session = Depends(get\_db)):

# [Вставь сюда свой код для получения бронирования]

pass

```

[Добавь сюда реализацию остальных эндпоинтов]

#### 3.3 Обеспечение валидации данных и обработки ошибок

Используйте Pydantic для валидации входных данных.

```python

from pydantic import BaseModel

class BookingCreate(BaseModel):

user\_id: int

date: str

status: str

@app.exception\_handler(Exception)

def handle\_exception(request, exc):

# [Добавь свой код для обработки ошибок]

pass

```

---

### 4. РУЧНАЯ ПРОВЕРКА API

#### 4.1 Методика ручного тестирования функциональности

- Используйте Postman или curl для отправки запросов на эндпоинты.

- Проверьте:

- Создание бронирования;

- Получение информации;

- Обновление и удаление бронирования.

#### 4.2 Примеры сценариев тестирования

1. Создание бронирования:

```json

POST /bookings

{

"user\_id": 1,

"date": "2024-12-15T10:00:00",

"status": "confirmed"

}

```

2. Получение информации о бронировании:

```bash

GET /bookings/1

```

[Добавь свои тестовые сценарии]

---

### 5. ДОКУМЕНТИРОВАНИЕ API

#### 5.1 Автоматическая генерация документации в FastAPI

FastAPI автоматически создаёт документацию в формате OpenAPI. Для доступа откройте:

- `/docs` — Swagger UI;

- `/redoc` — ReDoc.

#### 5.2 Руководство по использованию API

- Примеры запросов и ответов можно найти в `/docs`.

[Добавь описание специфических инструкций, если нужно]

---

### ЗАКЛЮЧЕНИЕ

В рамках курсовой работы был создан API для бронирования, обеспечивающий основные операции с данными. Работа включает проектирование, реализацию и ручное тестирование функционала. Данное API предоставляет возможность легко адаптировать его под различные сценарии бронирования, что делает его универсальным инструментом для автоматизации подобных процессов.

---

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[Добавьте список литературы, статей или документации, использованных в работе]

---

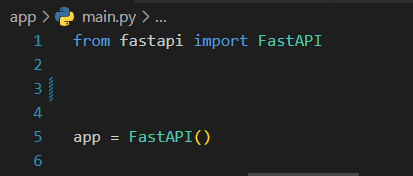
### ПРИЛОЖЕНИЕ

А. Примеры запросов и ответов API (в формате JSON)

Б. SQL-запросы для работы с базой данных

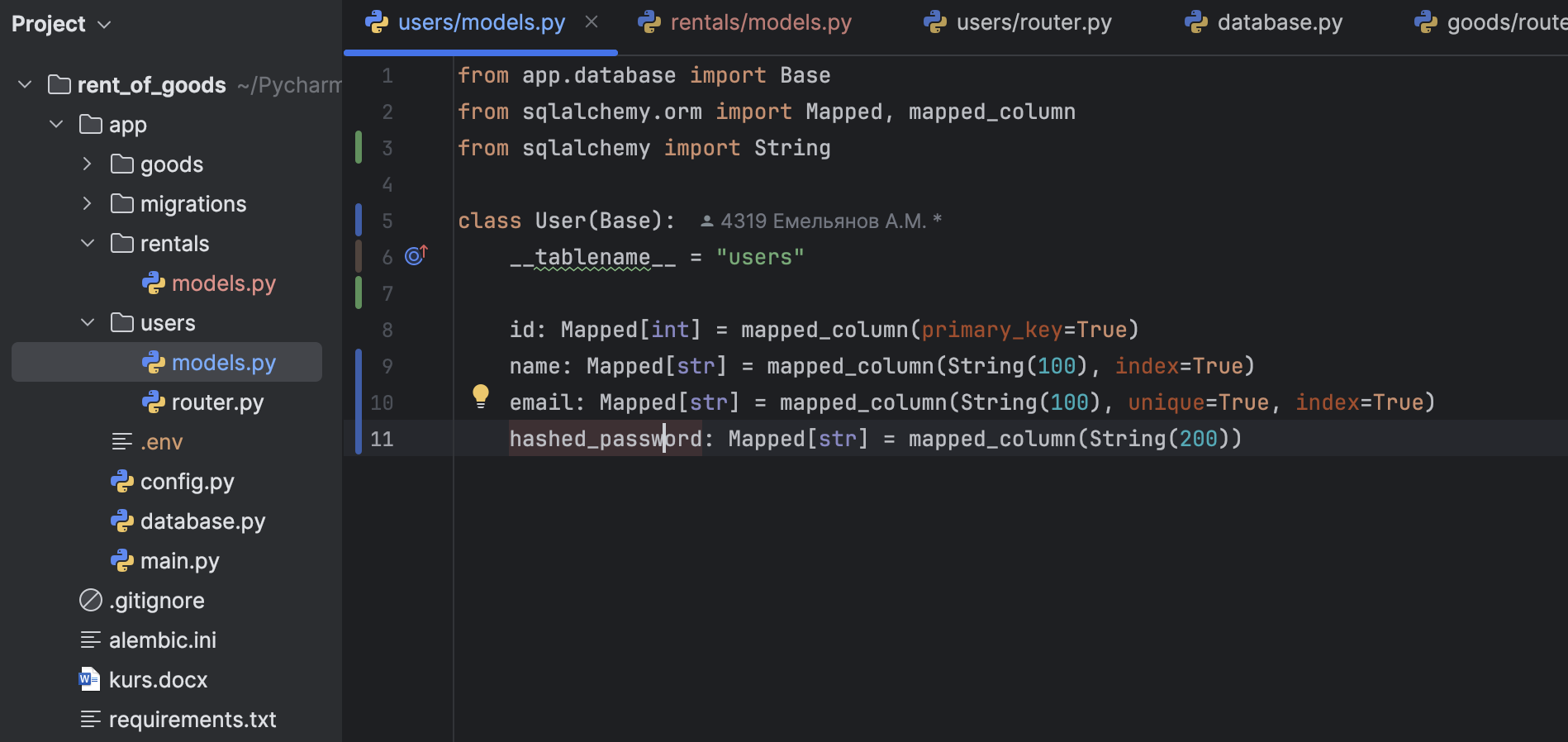
В. Код эндпоинтов API

Создал первый файл (main.py) в нём поместил

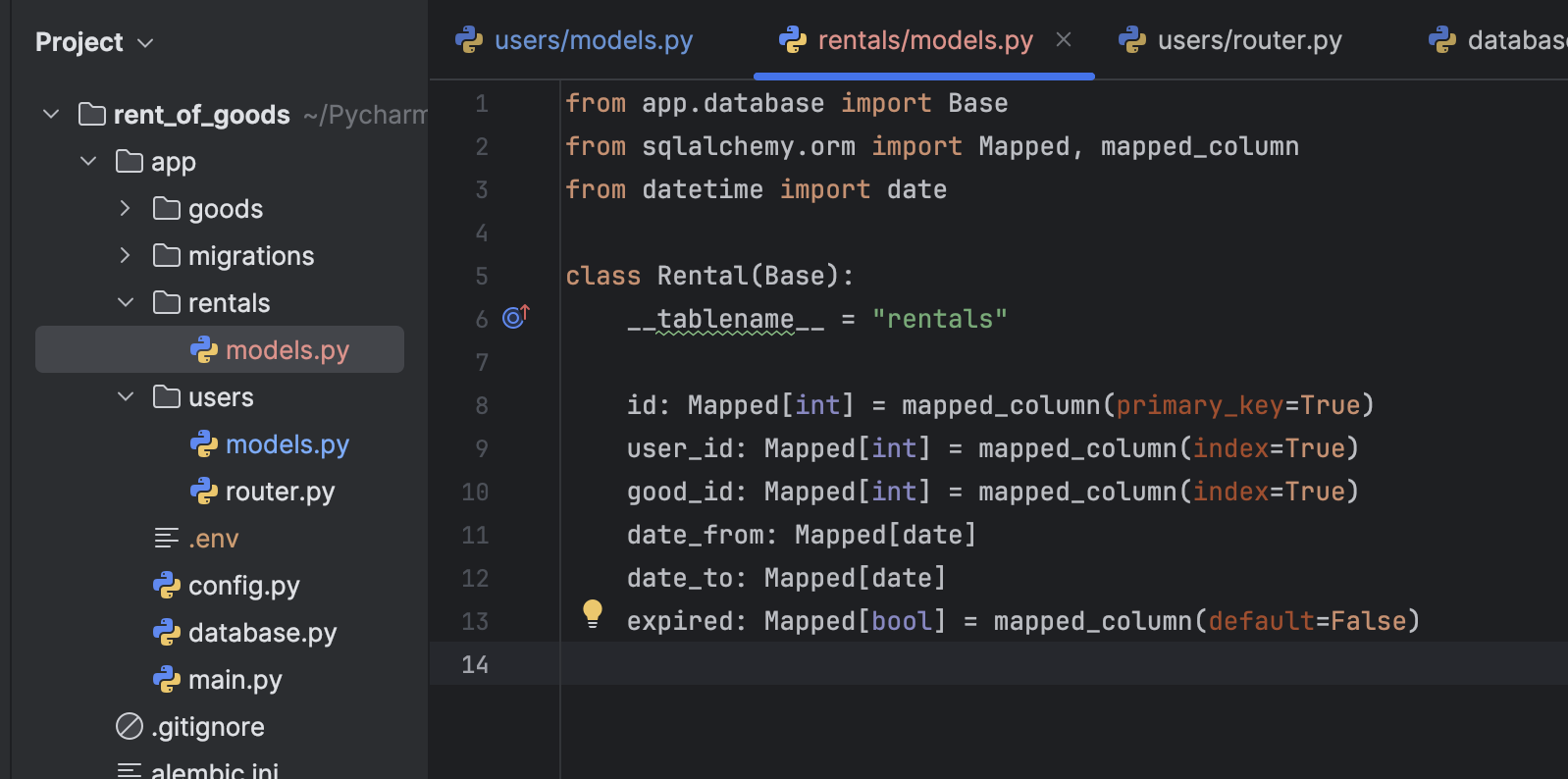
Создал файл requirements.txt для настройки зависимостей



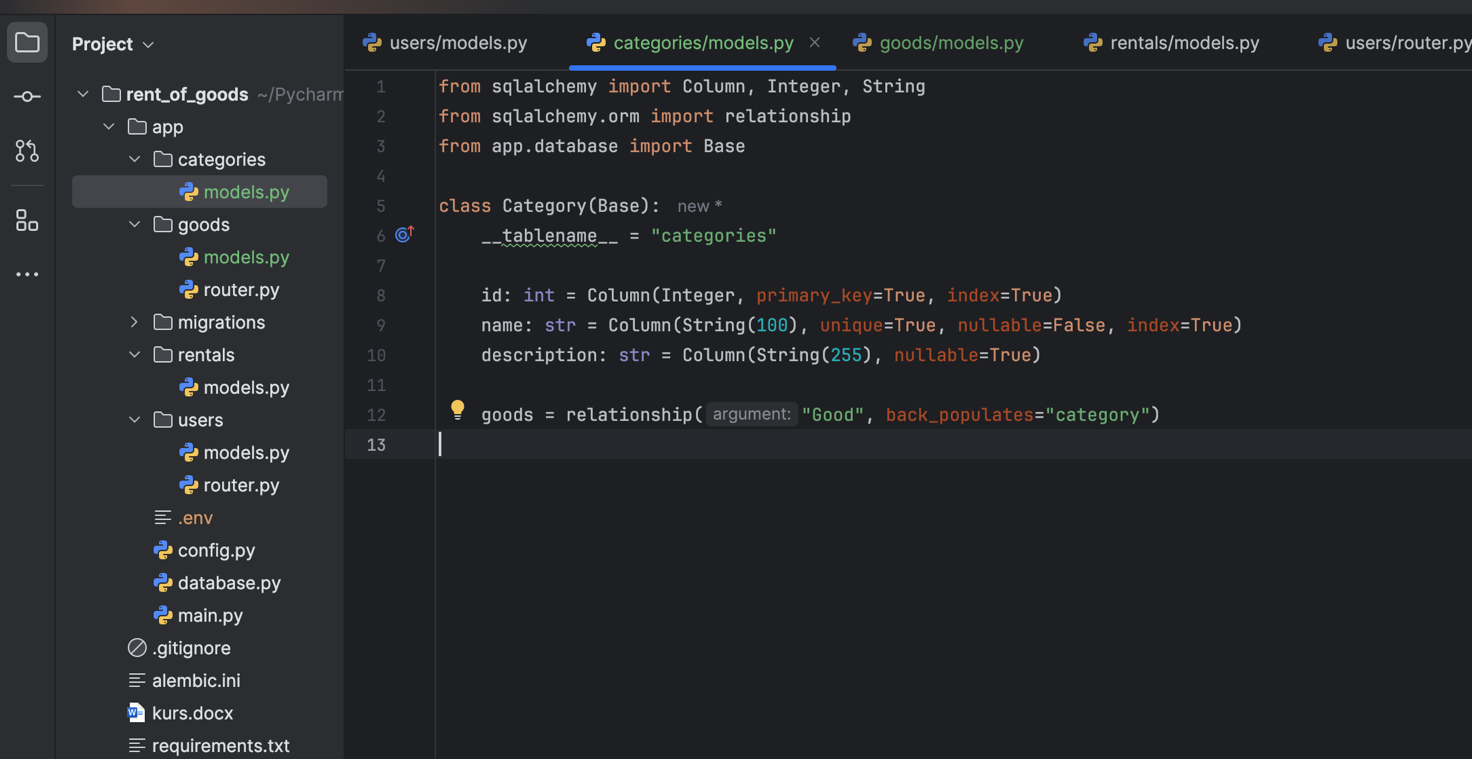
Изменил users/models.py

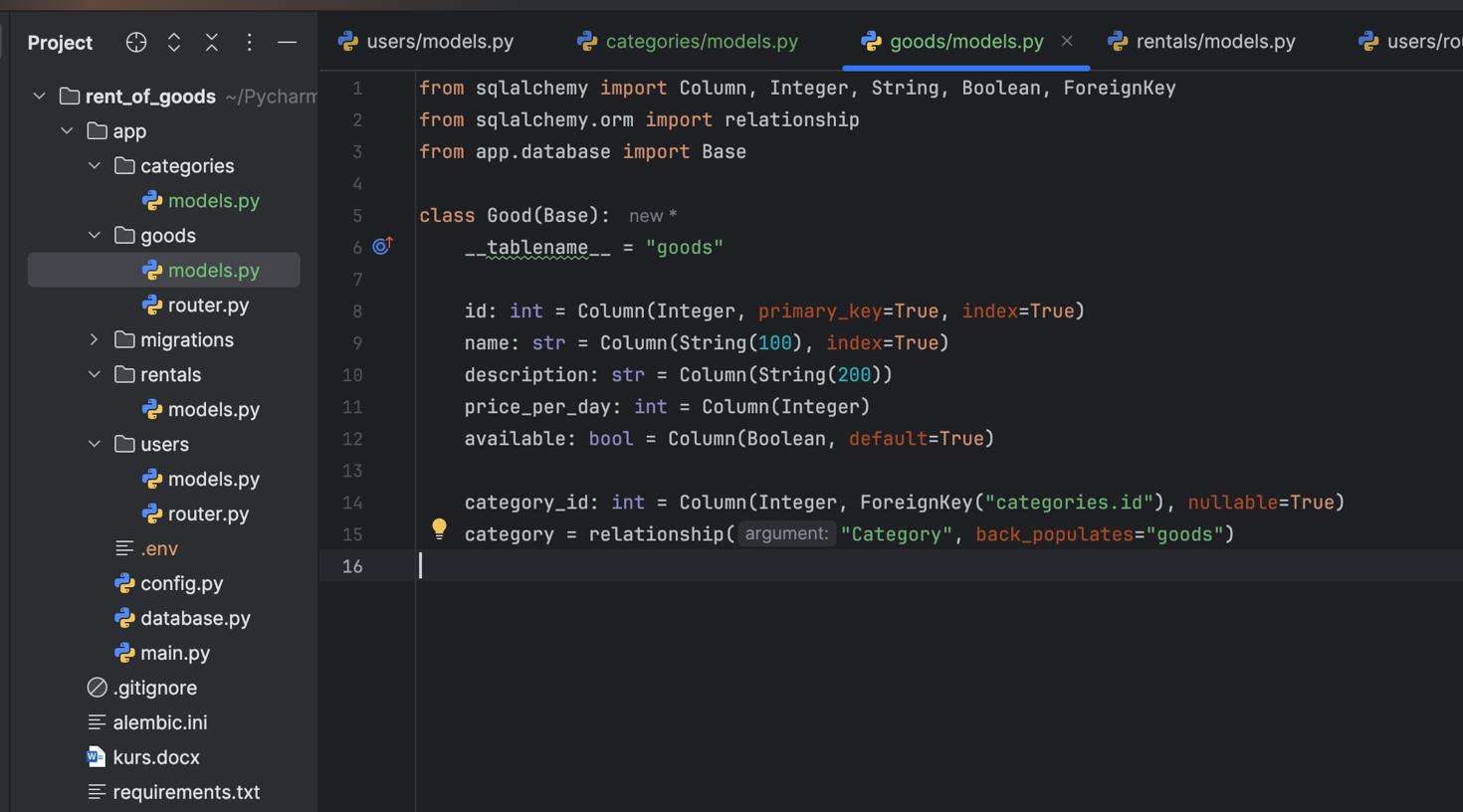


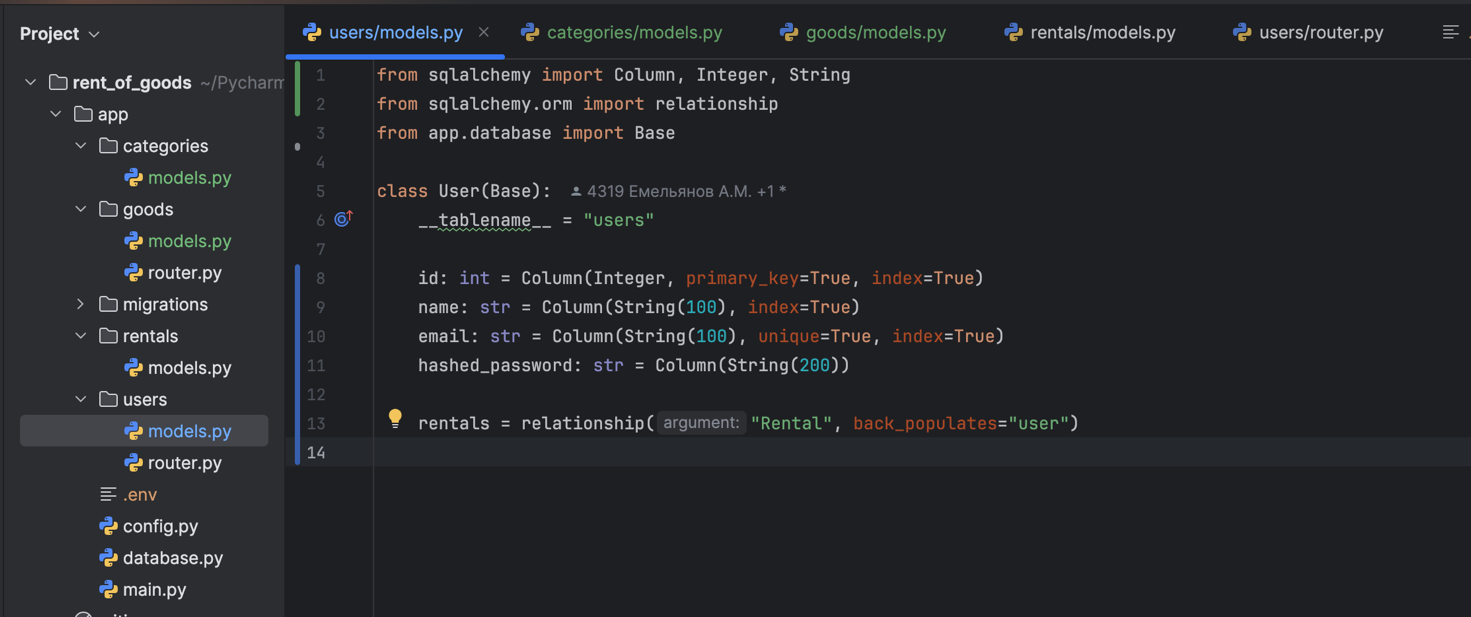
Изменил rental/models.py

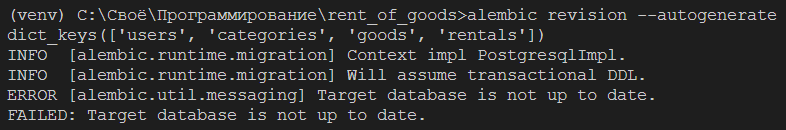
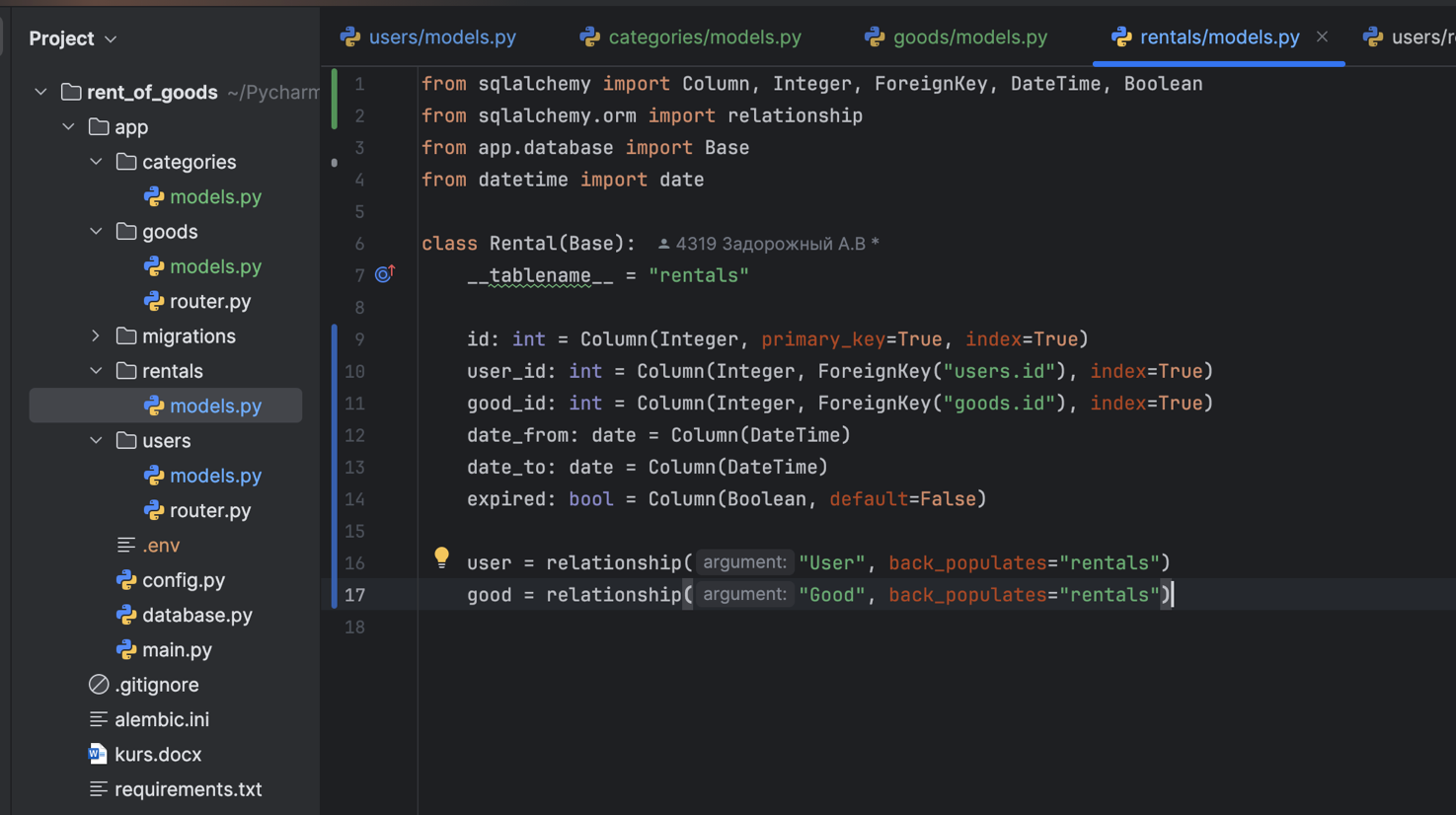


Добавил каталог categories  + models.py

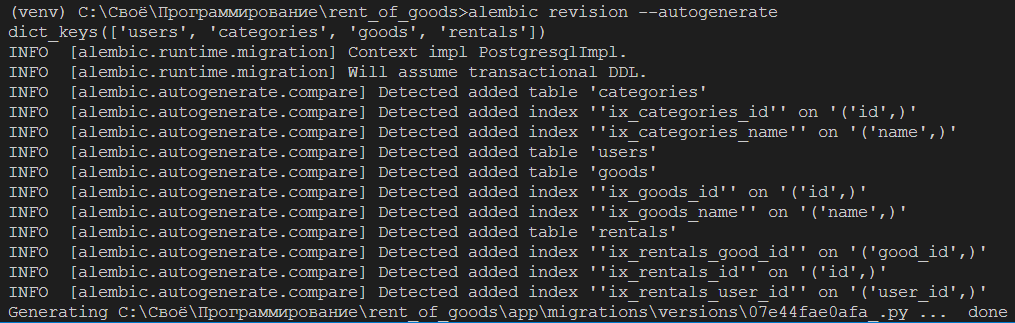


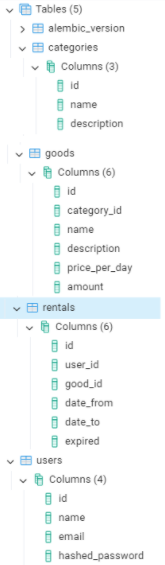
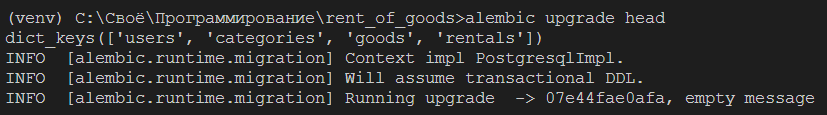
Обновил models.py для /users /rentals /goods (Установил связи между таблицами)

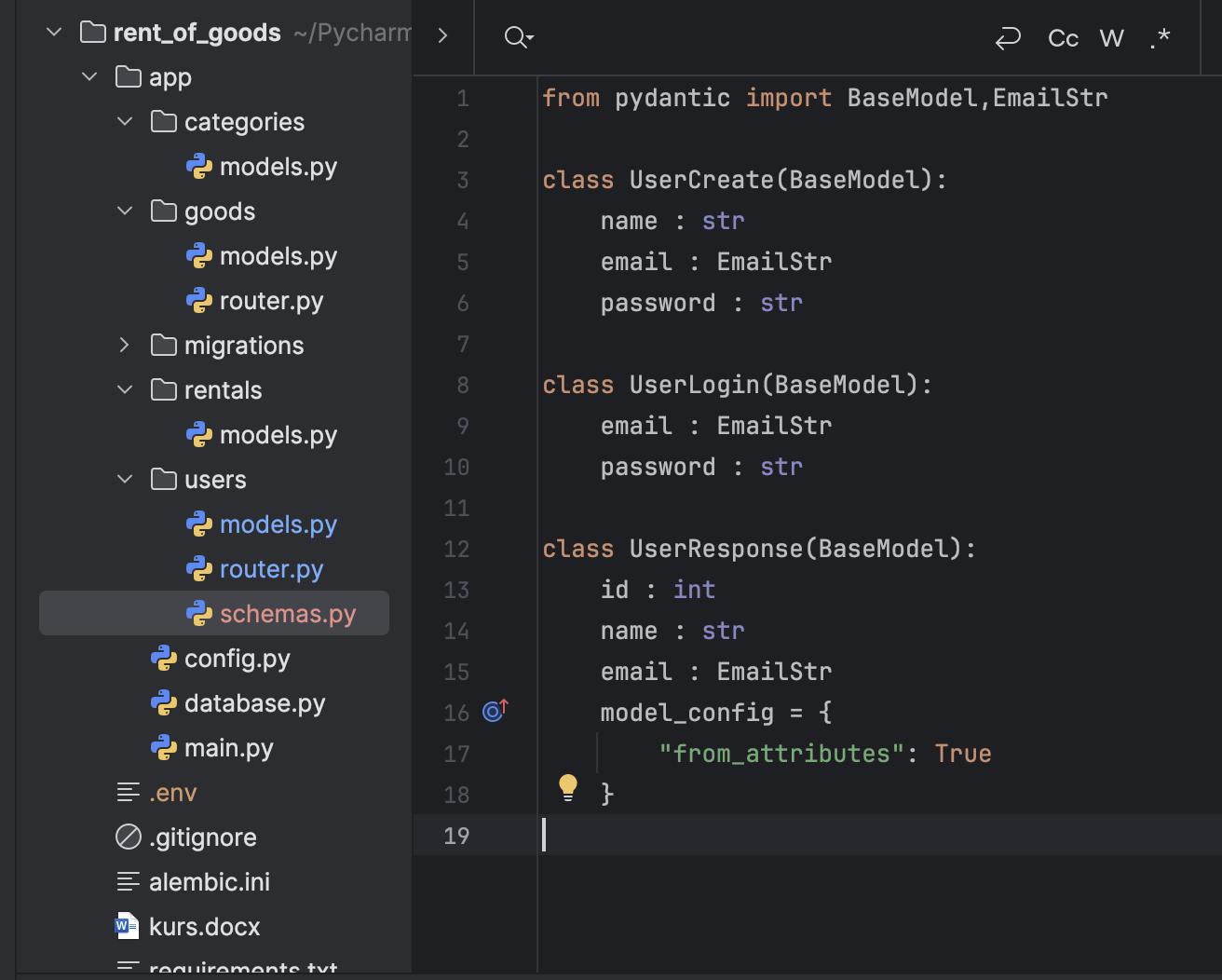


Возникла проблема при миграции. Пытаюсь решить

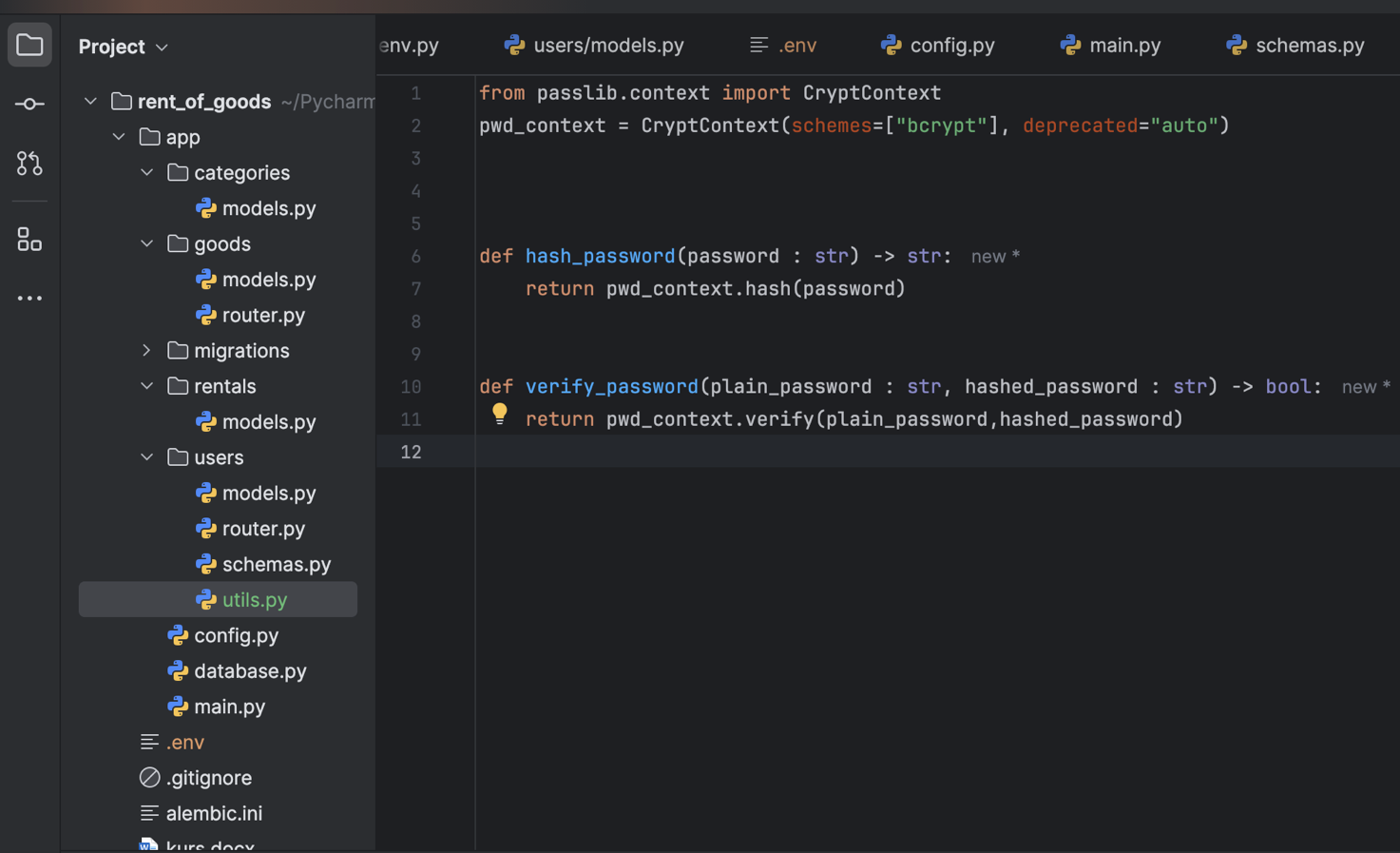
Проблема была в другой миграции. Удалил её и создалась новая миграция

Далее прогоняю миграцию и создаю все необходимые таблицы

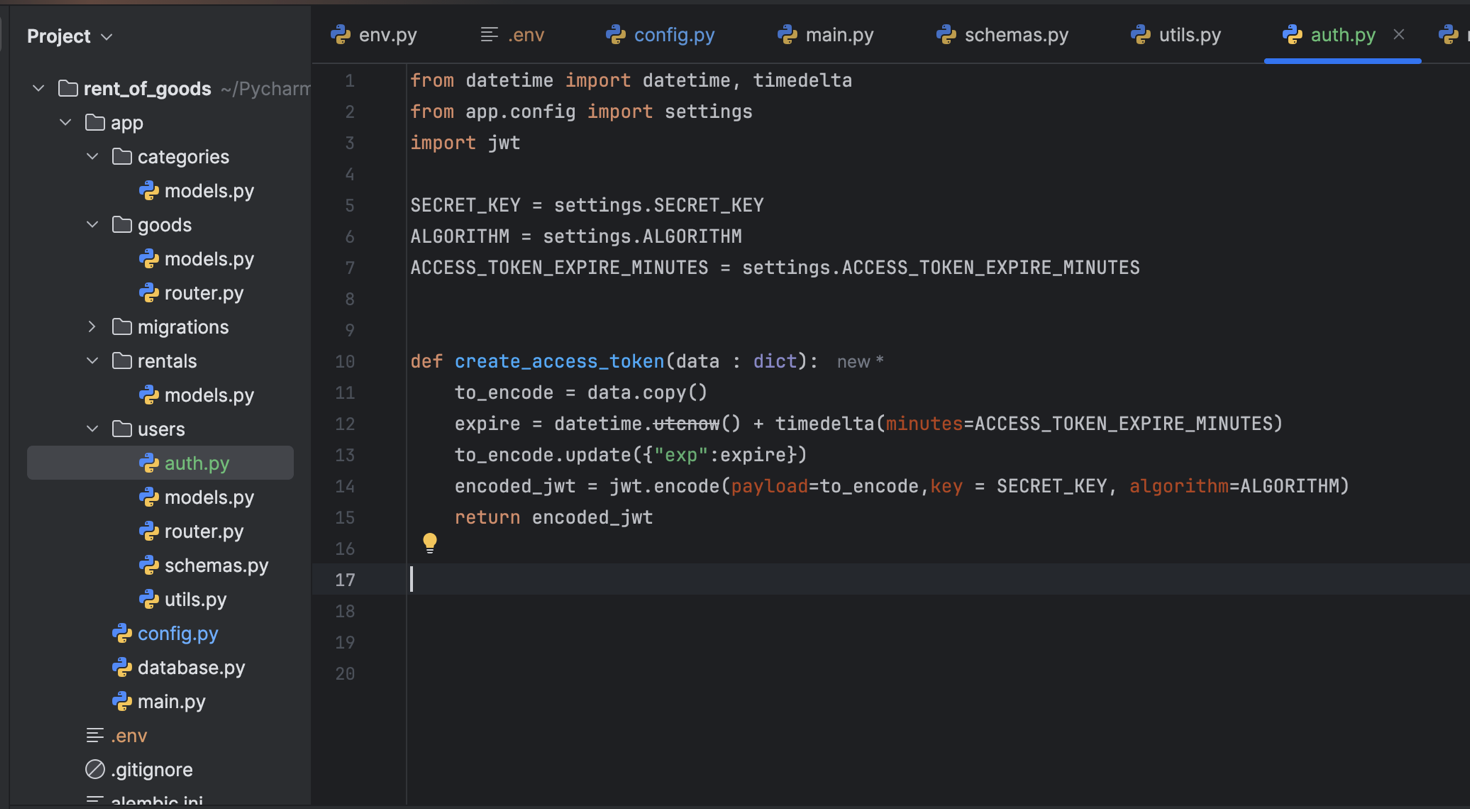
Создал несколько схем для аутентификации



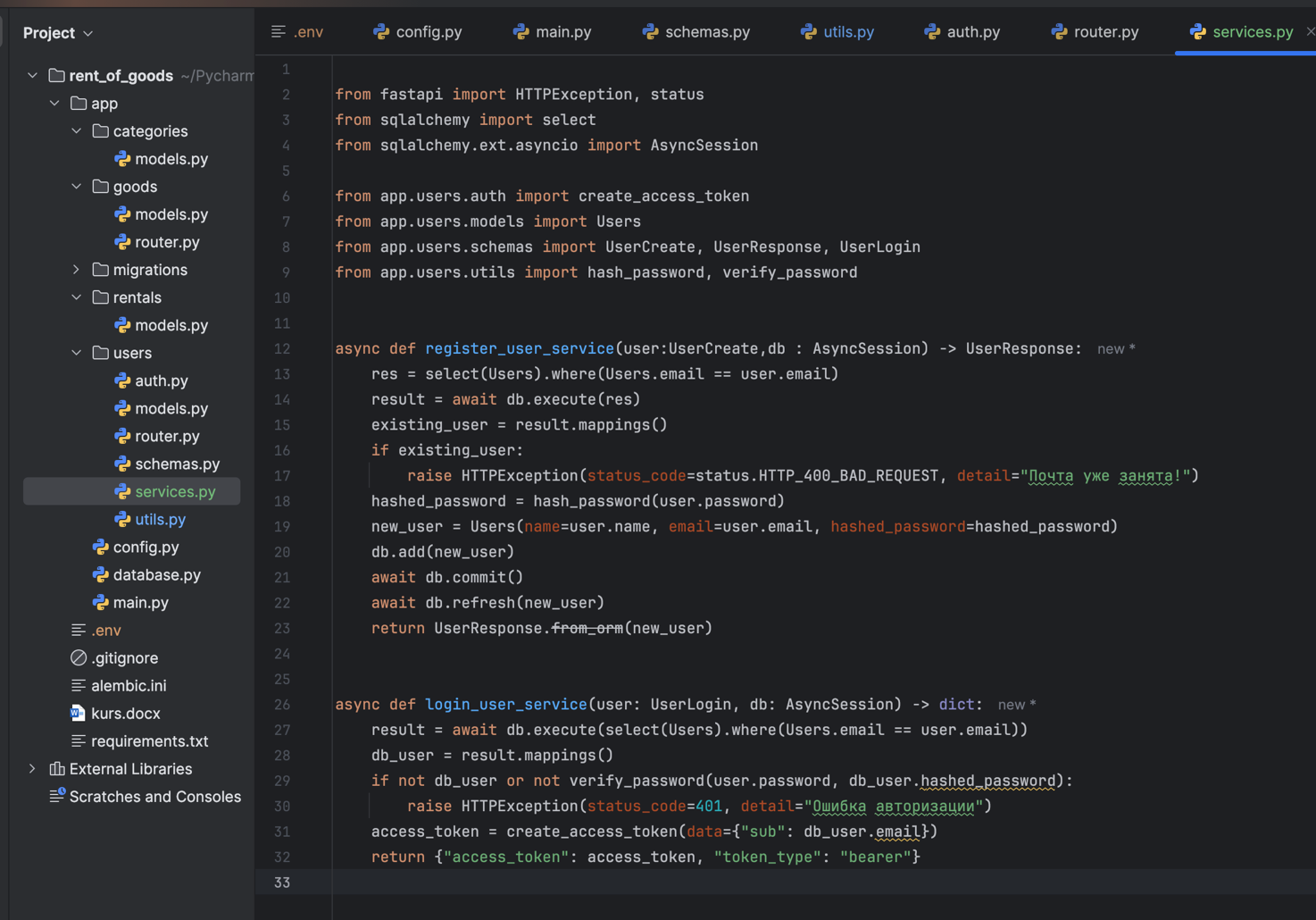
Создал utils.py для хэширования паролей



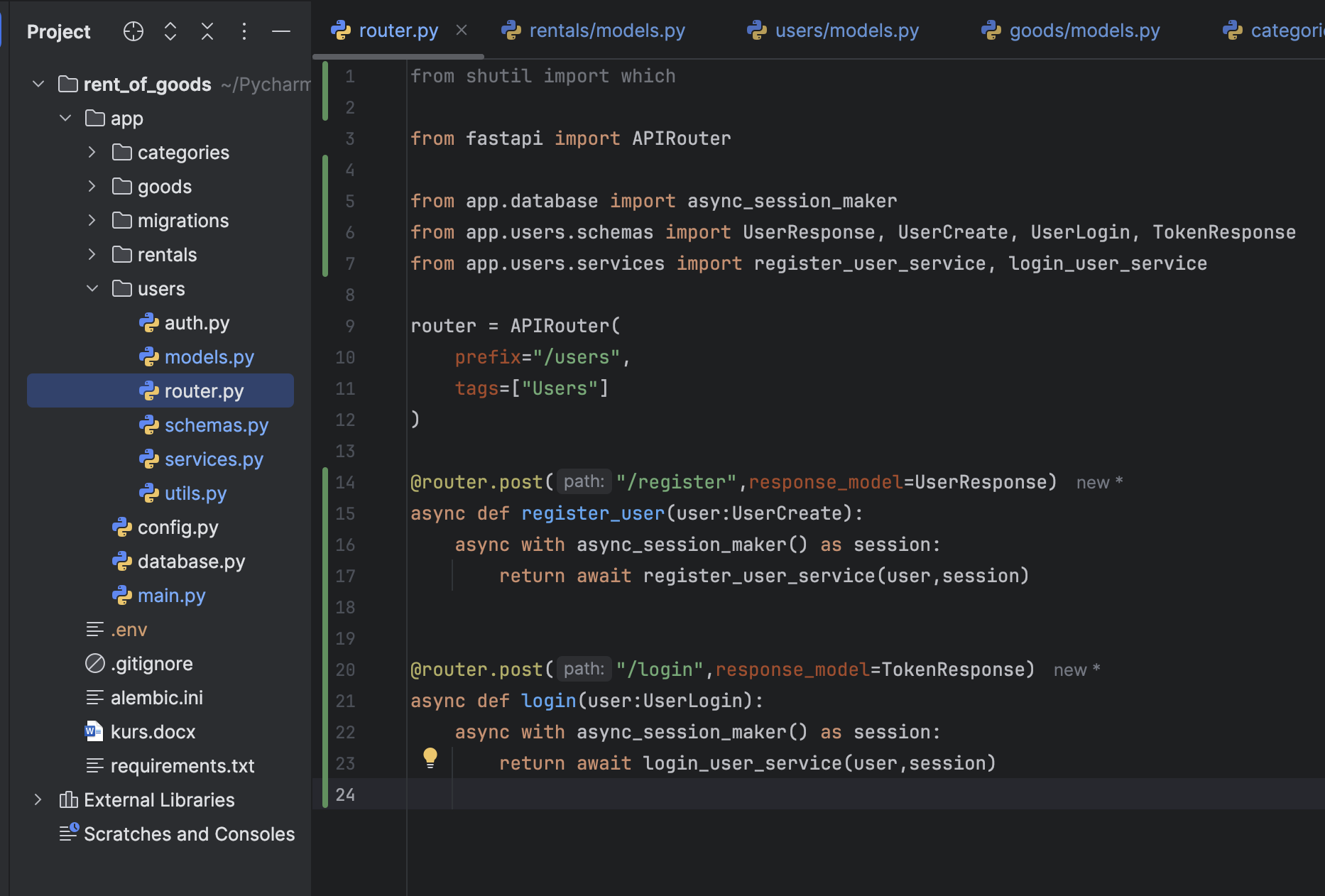
Создал auth.py для получения jwt токена, скрыл важные данные



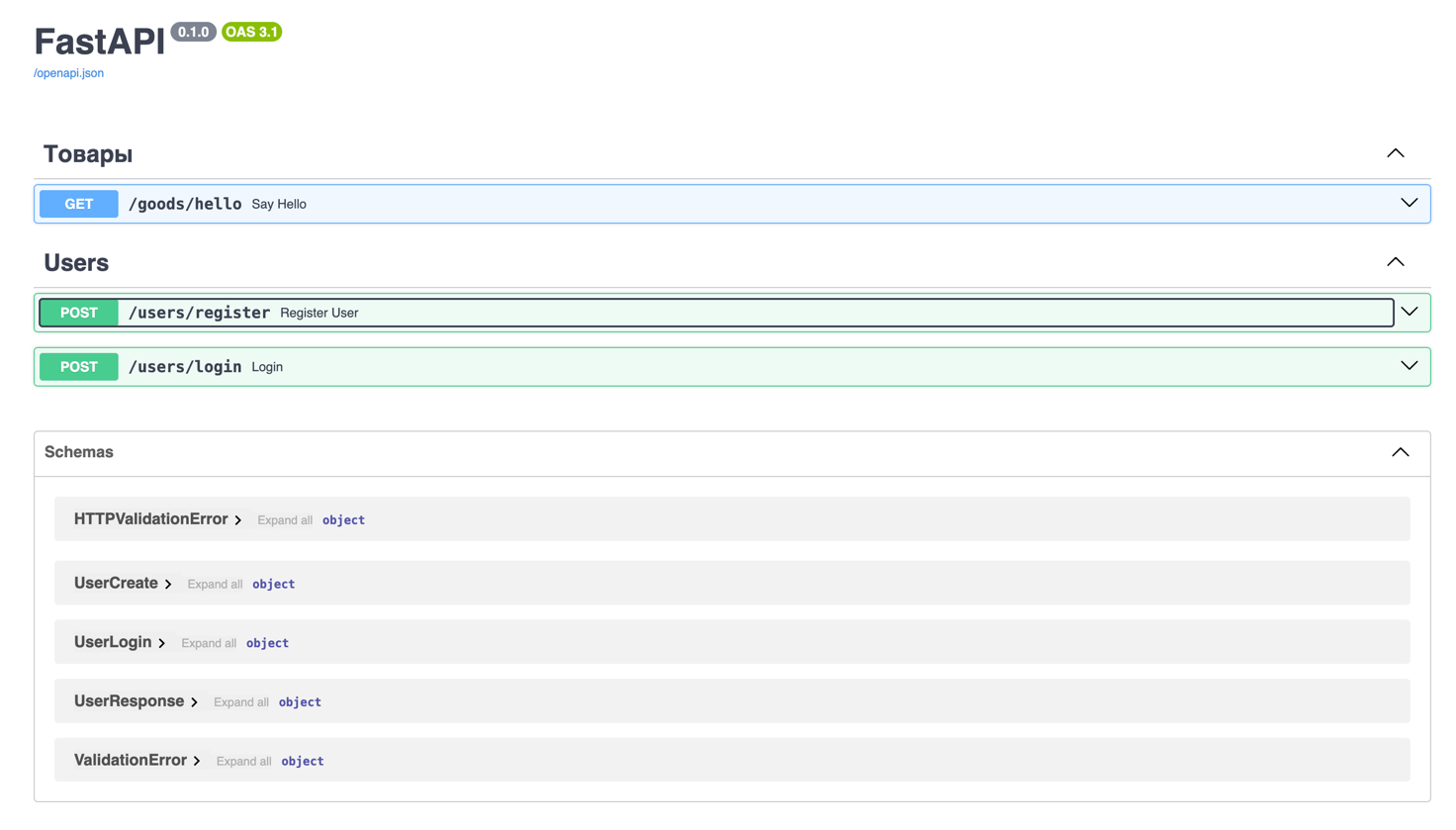
Добавил логику регистрации и входа



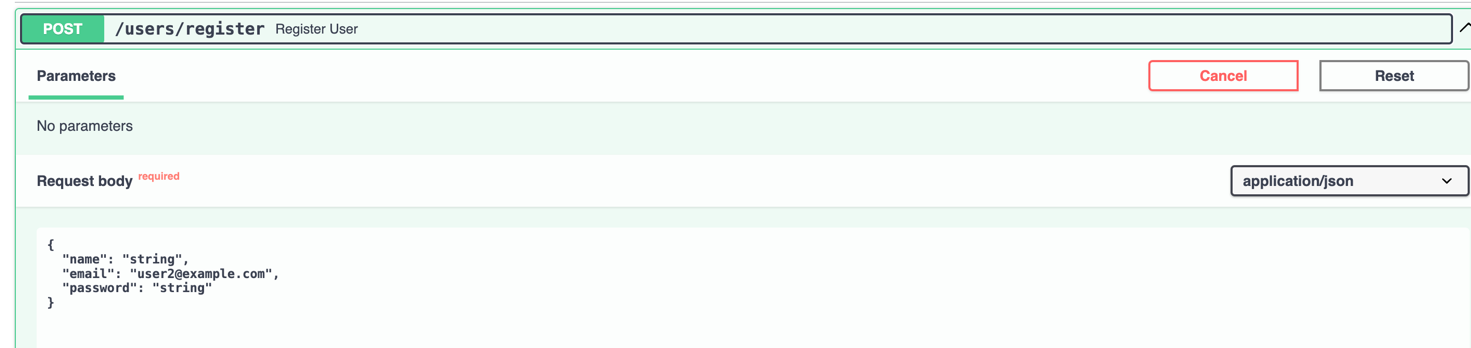
Настроил роутер для регистрации



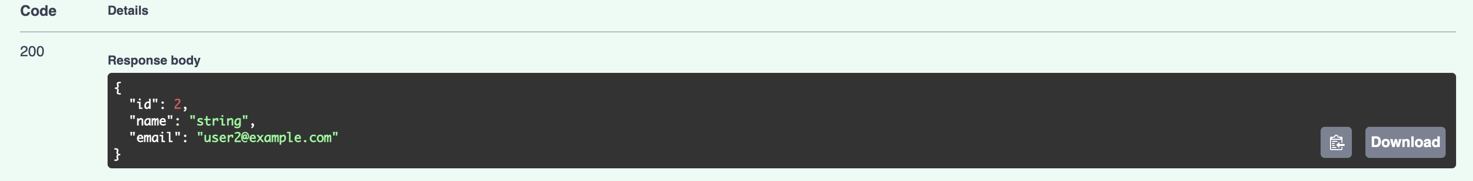
В результате появилось два эндпоинта для аутентификации и несколько схем



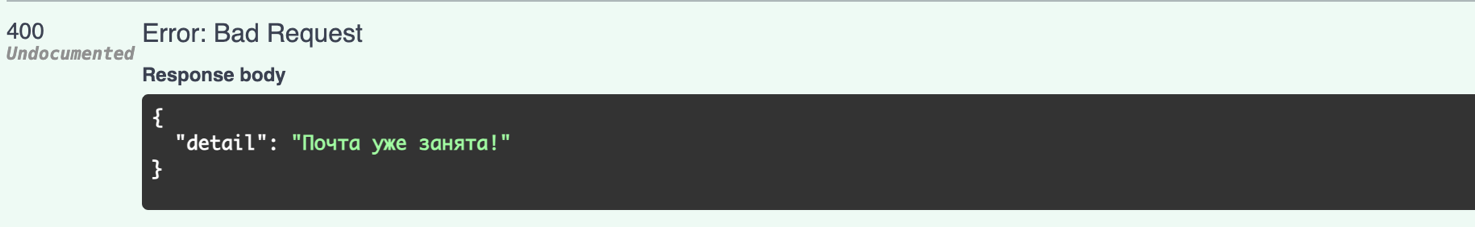
Пример теста энпоинта register



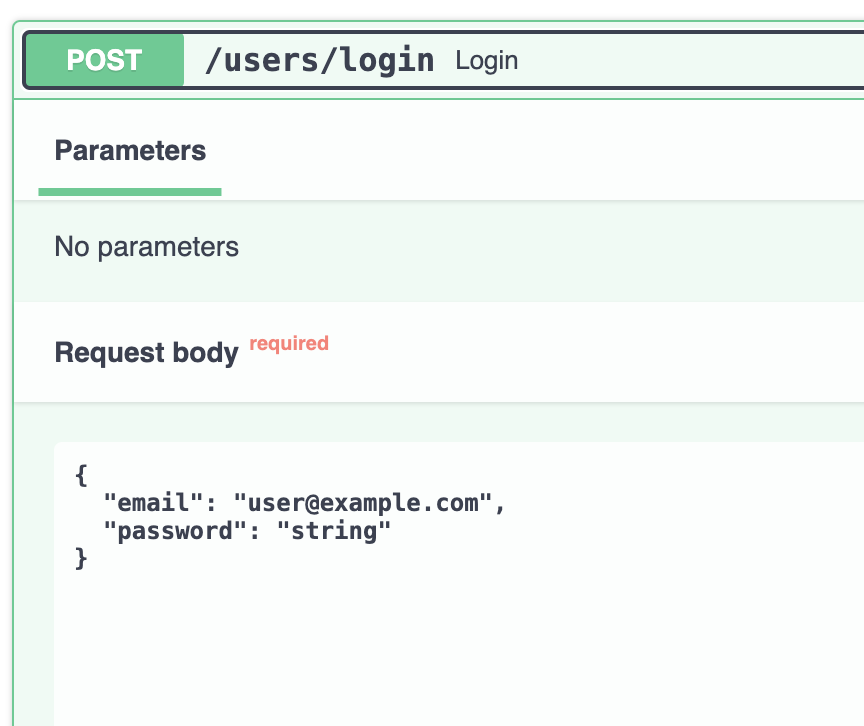
Результат



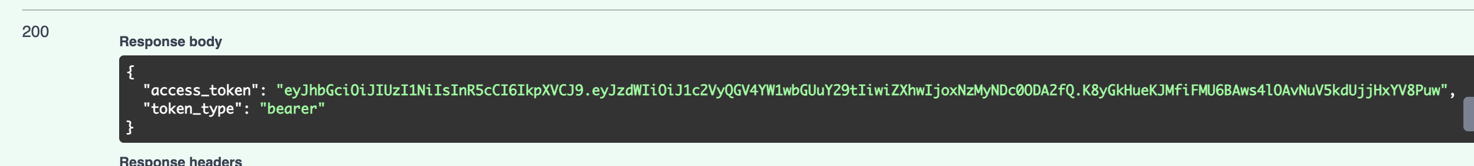
При повторной регистрации с занятой почтой



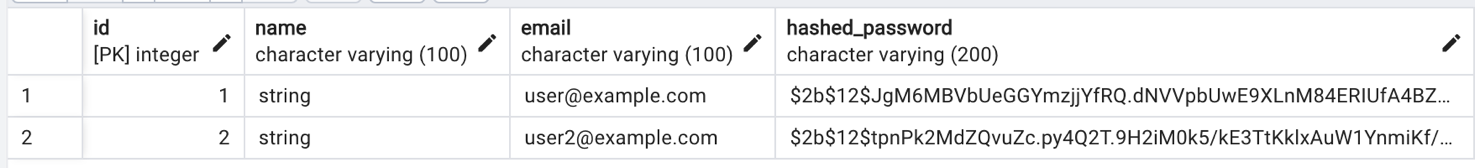
Пример теста эндпоинта login



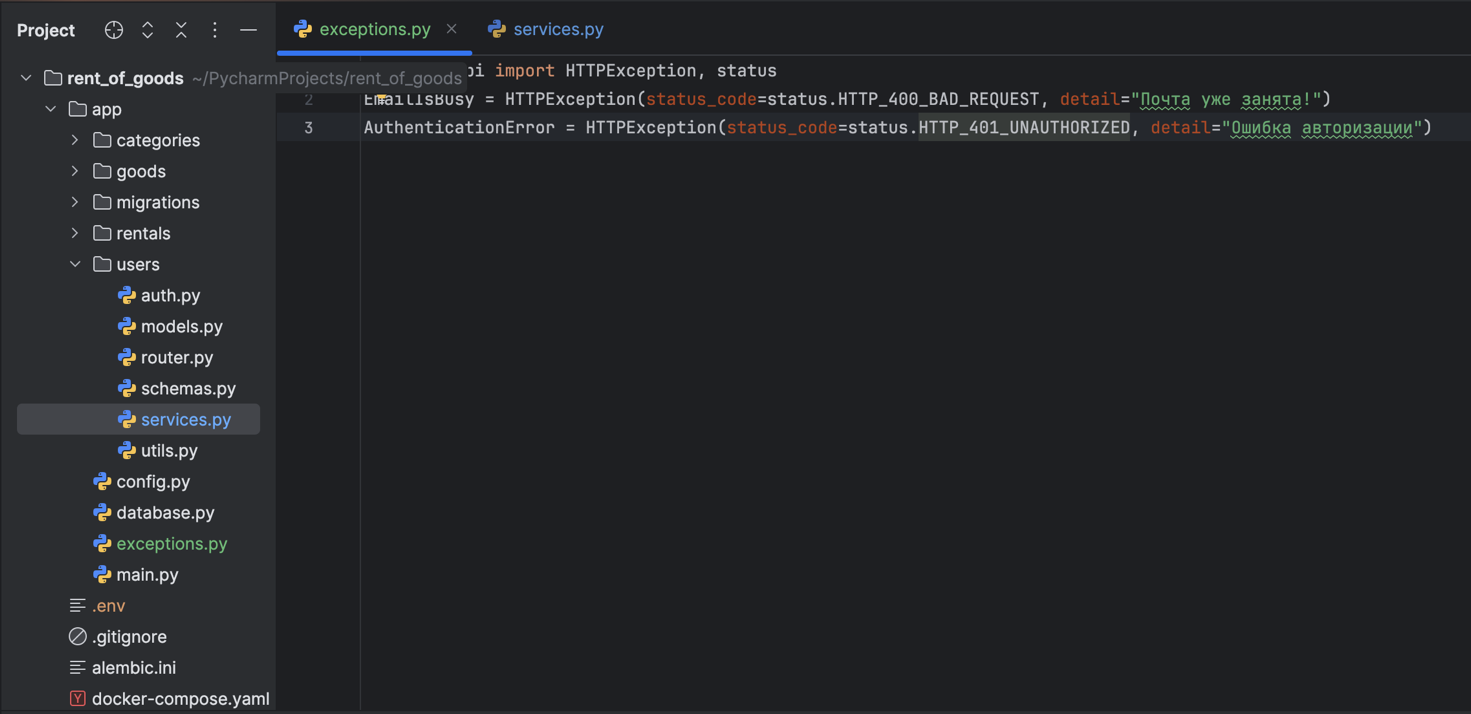
Результат



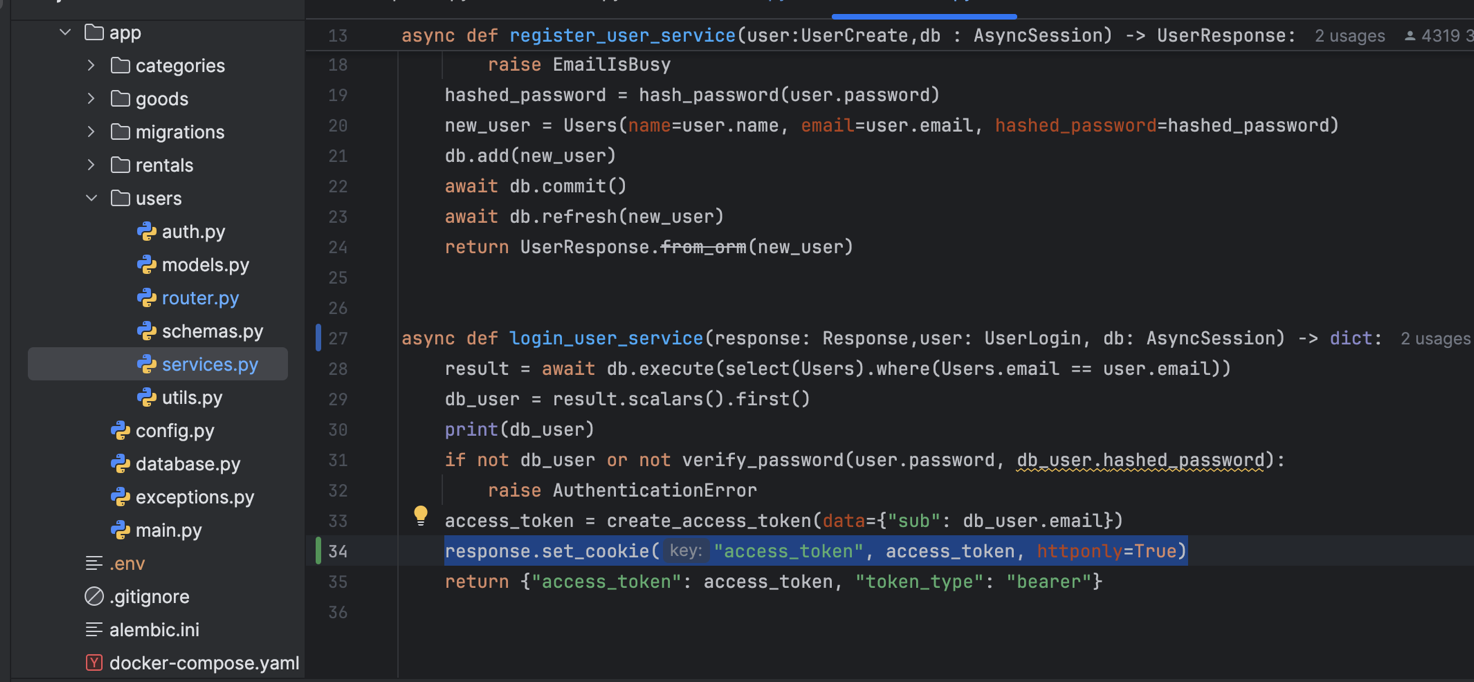
Отображается в бд



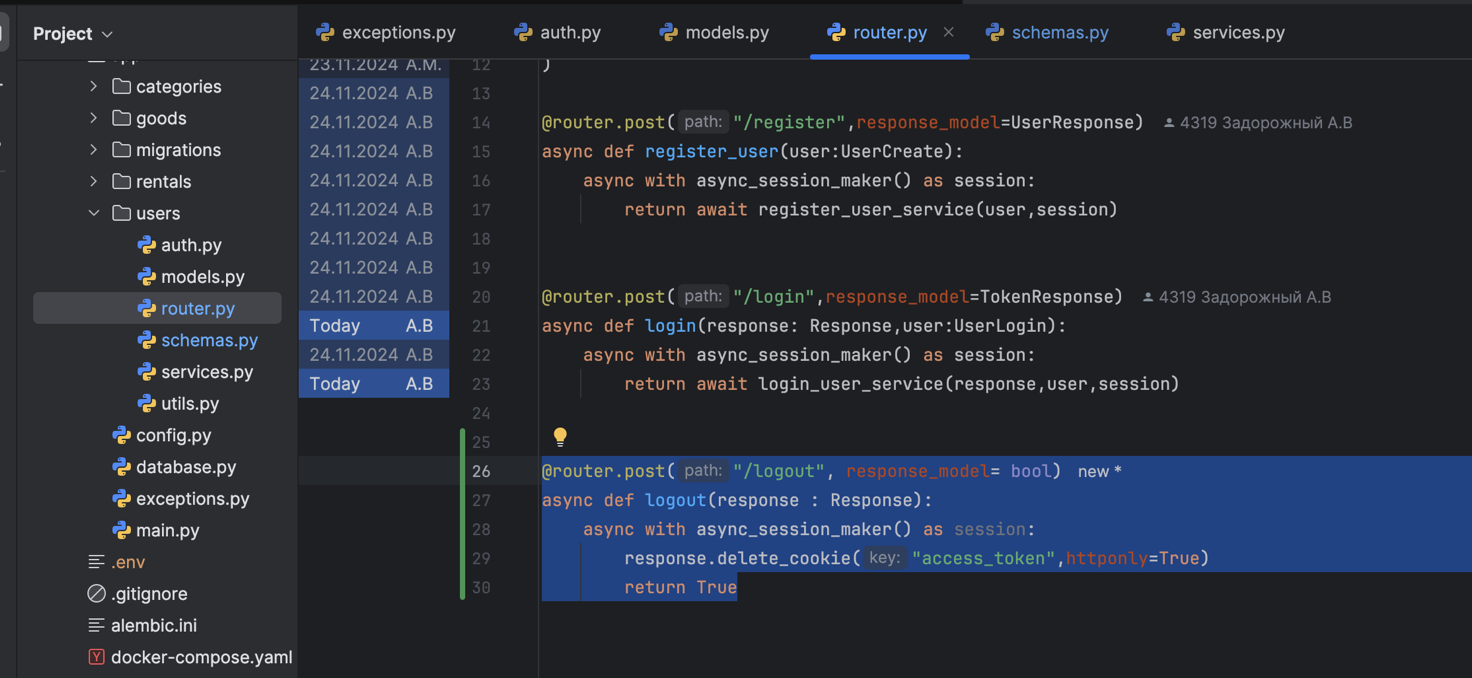
Вынес ошибки в отдельный файл



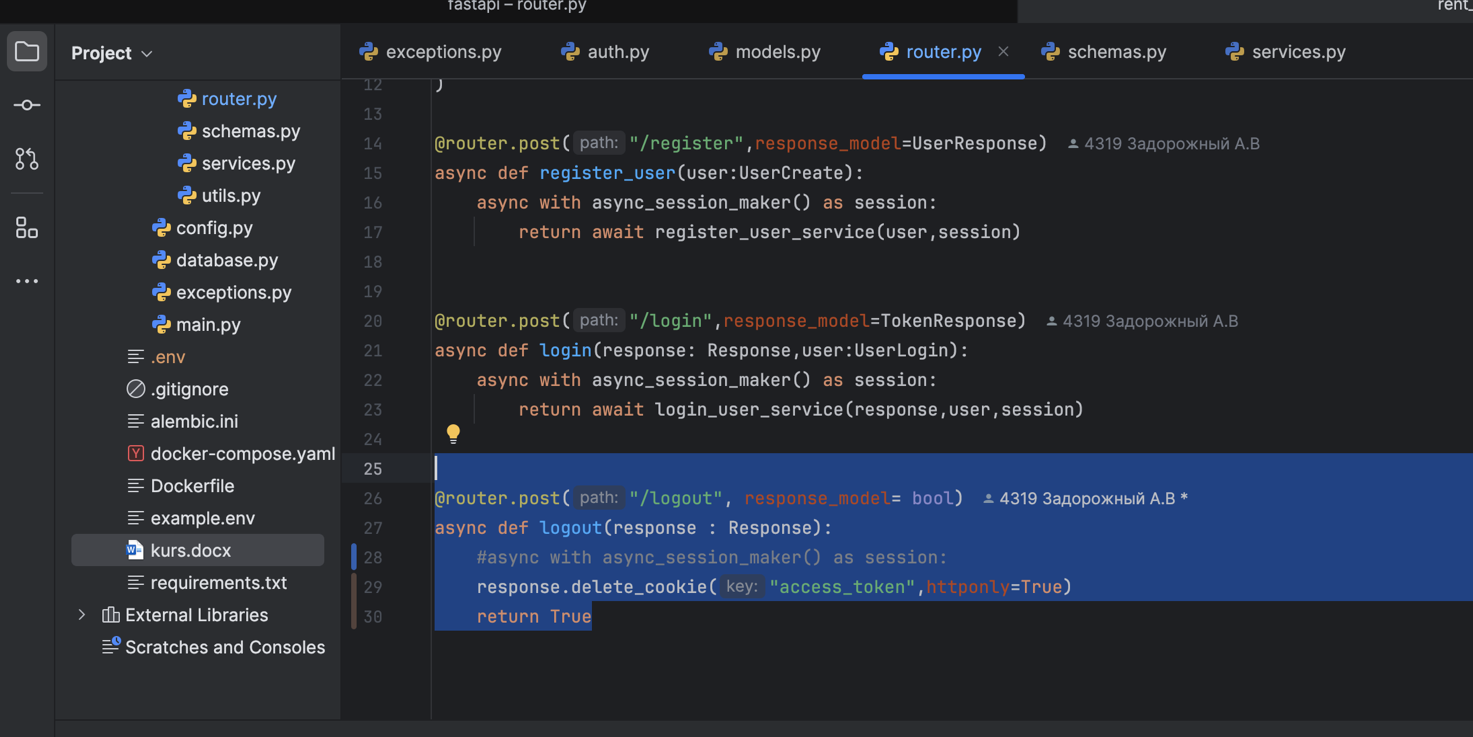
Добавил куки access\_token



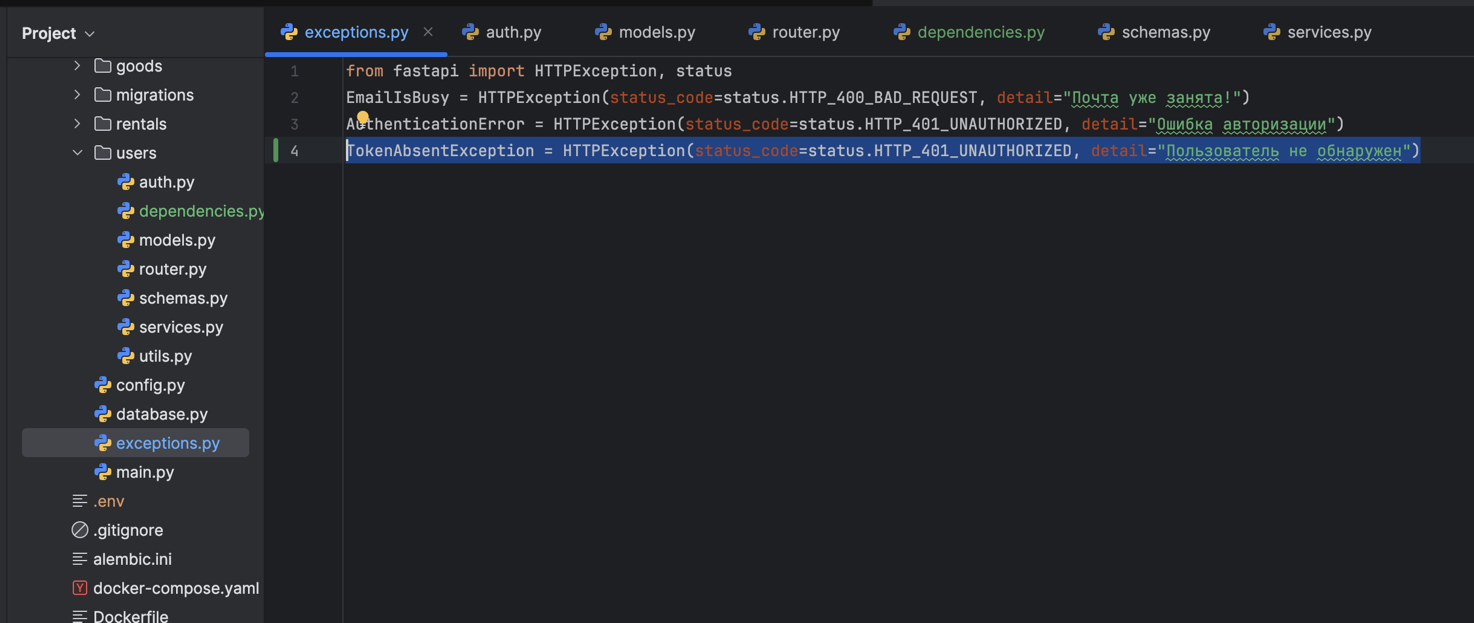
Добавил эндпоинт для логаута



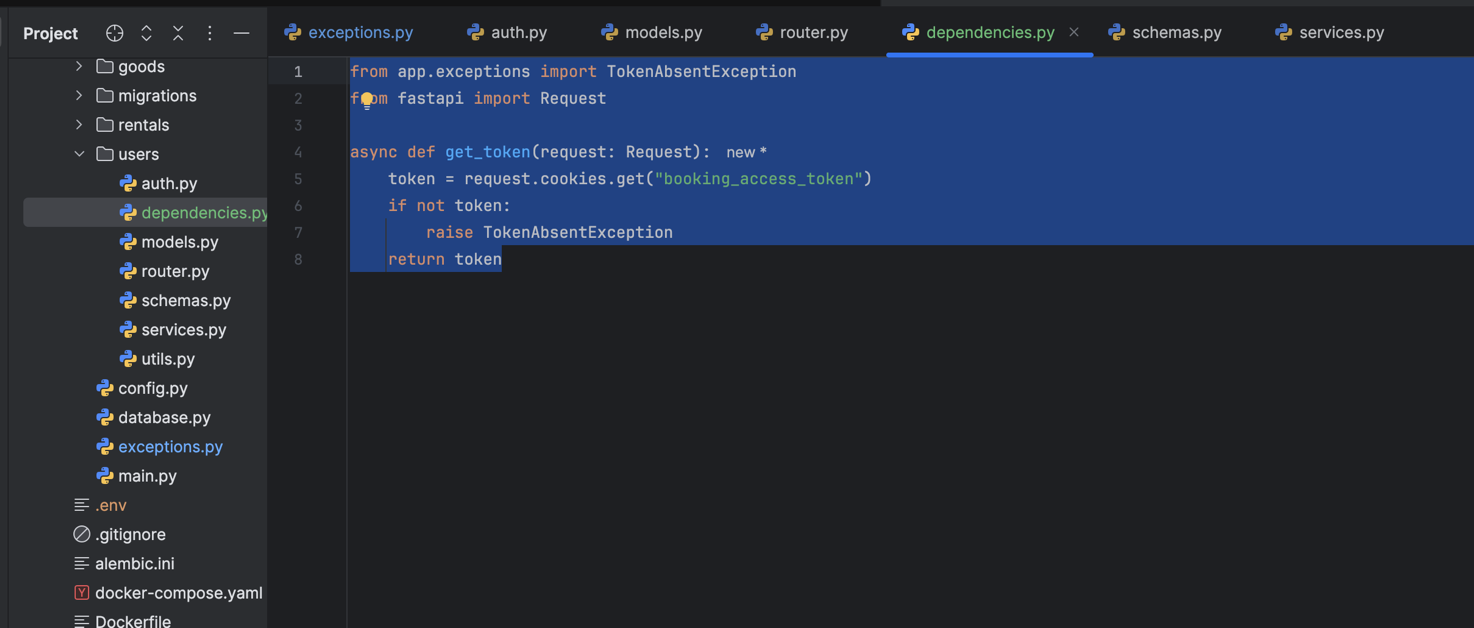
Fix логаута убрал работу с сессиями



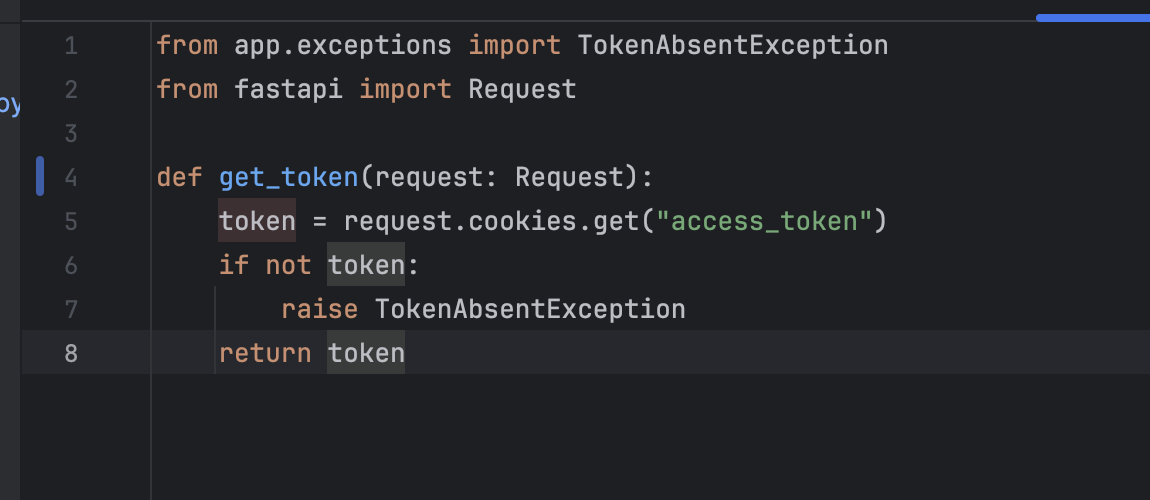
Добавил новую ошибки при поиске токена в куках (для дальнейших авторизации)



Добавил логику для поиска токена

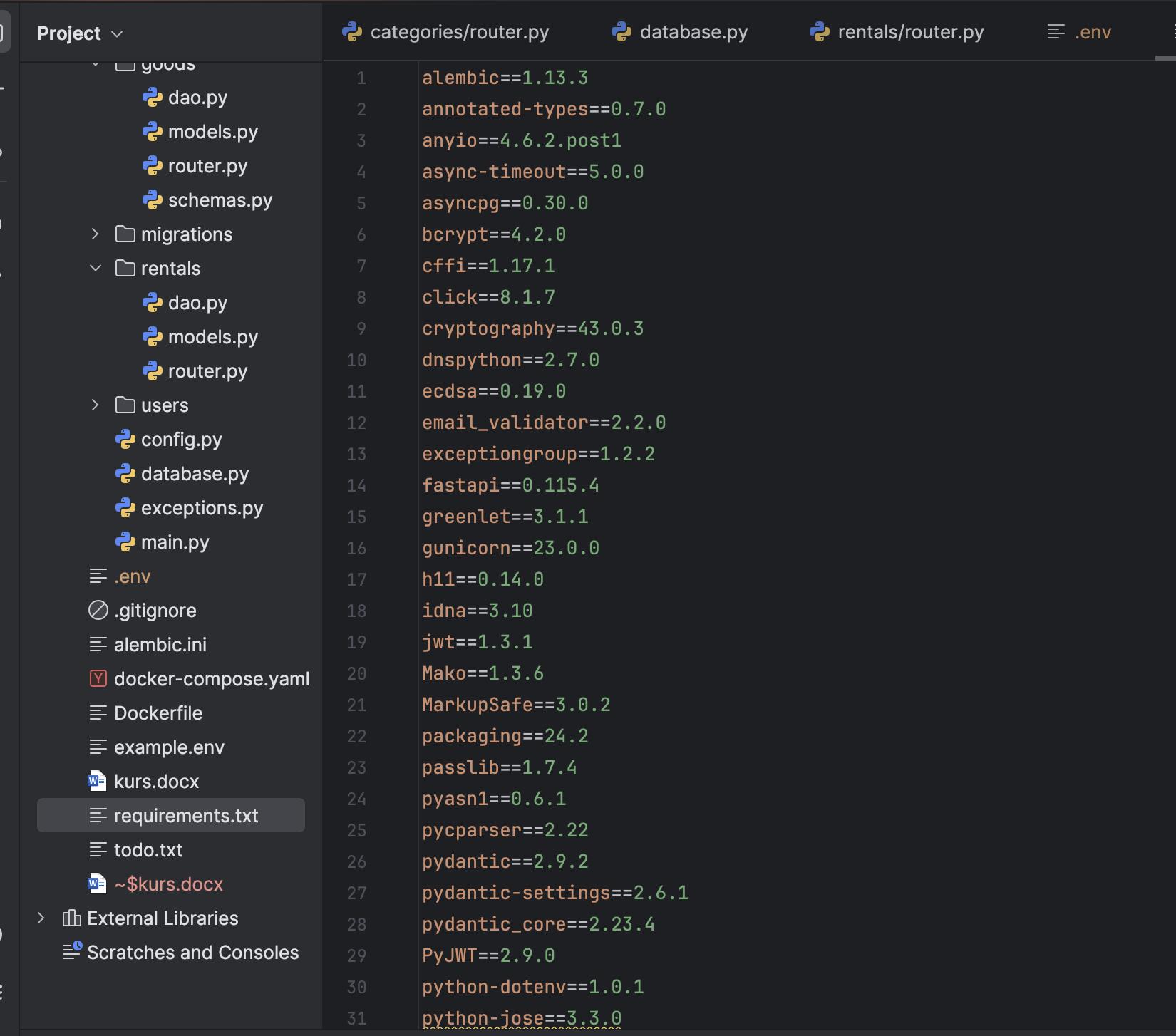


в get\_token() убрал  async

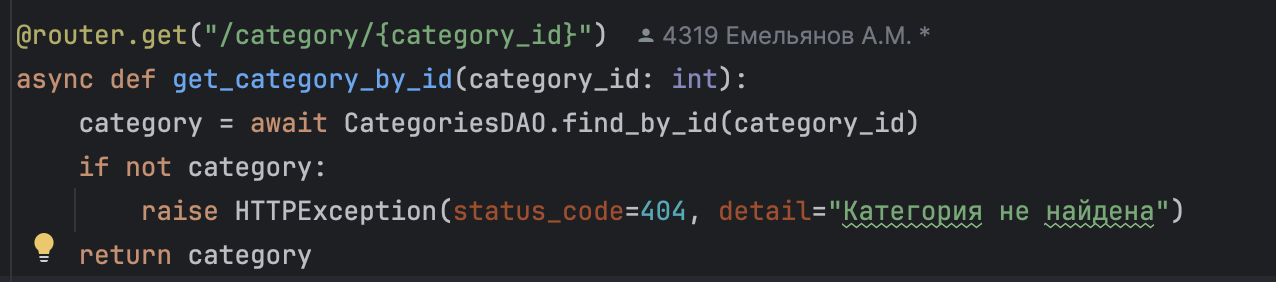




Генерируем список Python-зависимостей



Список всех зависимостей

Если категории нет, вызываем ошибку 404

Добавил проверку jwt токена для добавления