САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 5

Выполнил:

Григорян Самвел

Группа K3440

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Разработать спроектированные микросервисы:

- выделить самостоятельные модули в приложении;
- провести разделение API на микросервисы;
- настроить сетевое взаимодействие между микросервисами.

Проект: сервис для аренды недвижимости.

Ход работы

1. Выделение самостоятельных модулей

Ранее реализованное монолитное приложение (сущности User/Building/Apartment/Contract и REST API) было декомпозировано на 3 независимых доменных модуля, соответствующих ключевым бизнес-сущностям и потокам: пользователи, недвижимость, контракты.

Выделенные микросервисы:

- User Service (порт 3001) — управление пользователями, регистрация, роли (агенты/клиенты). Основная сущность: User.
- Property Service (порт 3002) — управление недвижимостью, поиск/фильтрация объектов, фото/описания. Сущности: Building, Apartment.
- Contract Service (порт 3003) — создание и управление договорами аренды, валидация зависимостей, статусы. Сущность: Contract.

2. Реализация микросервисов и структура проектов

Каждый сервис реализован как отдельное Node.js/Express + TypeScript приложение со своей структурой исходников и точкой входа. Использована унифицированная структура каталогов: config, controllers, entities, middleware, routes, services (для межсервисного взаимодействия) и др.

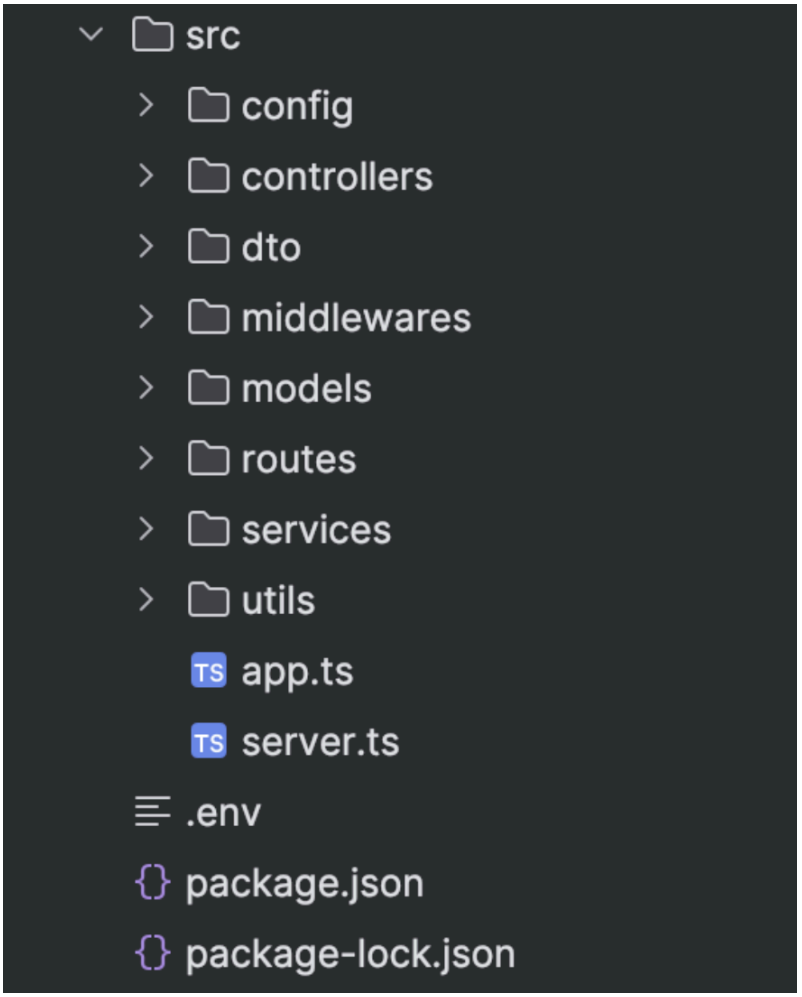

Рисунок 1 - Файловая структура проекта

В исходном API были реализованы основные REST-эндпоинты (аутентификация, CRUD для пользователей/зданий/квартир/контрактов). Эти эндпоинты были распределены по микросервисам согласно ответственности домена.

Для защиты маршрутов использована JWT-аутентификация через middleware: токен извлекается из заголовка Authorization: Bearer ..., валидируется и по userId извлекается пользователь из БД. Также реализован optionalAuthMiddleware.

Для межсервисного взаимодействия каждый микросервис предоставляет внутренний HTTP-эндпоинт POST /api/internal, принимающий { service, action, data }. Это позволяет Contract Service обращаться к другим сервисам по унифицированному контракту.

В Contract Service реализованы адаптеры (классы UserService и PropertyService) на базе axios. Адреса сервисов берутся из переменных окружения USER_SERVICE_URL и PROPERTY_SERVICE_URL.

Листинг 1 - user-service: внутренний обработчик /api/internal.

```
import { Router } from 'express';

import { AppDataSource } from '../typeorm/data-source';

import { User } from '../entities/User';

import { InternalRequest, InternalResponse } from '../../common/internal-api';



export const internalRouter = Router();



internalRouter.post('/internal', async (req, res) => {

 const body = req.body as InternalRequest<{ userId: number }>;



 try {

   if (body.service !== 'user-service') {

     const resp: InternalResponse = { success: false, error: 'Wrong service target'
};

     return res.status(400).json(resp);

   }



   switch (body.action) {

     case 'getUserById': {

       const repo = AppDataSource.getRepository(User);

       const user = await repo.findOne({ where: { UserID: body.data.userId } });



       const resp: InternalResponse = user
```

```
        ? { success: true, data: user }
        : { success: false, error: 'User not found' };

      return res.json(resp);
    }

    default:
      return res.status(400).json({ success: false, error: 'Unknown action' }
satisfies InternalResponse);
  }
 } catch (e) {
  return res.status(500).json({ success: false, error: 'Internal error' }
satisfies InternalResponse);
 }
});
```

Листинг 2 — property-service: внутренний /api/internal для квартир.

```
import { Router } from 'express';
import { AppDataSource } from '../typeorm/data-source';
import { Apartment } from '../entities/Apartment';
import { InternalRequest, InternalResponse } from '../../common/internal-api';

export const internalRouter = Router();

internalRouter.post('/internal', async (req, res) => {
 const body = req.body as InternalRequest<{ apartmentId: number }>;
```

```
 try {

   if (body.service !== 'property-service') {

     return res.status(400).json({ success: false, error: 'Wrong service target' }
satisfies InternalResponse);

   }



   switch (body.action) {

     case 'getApartmentById': {

       const repo = AppDataSource.getRepository(Apartment);

       const apt = await repo.findOne({ where: { ApartmentID: body.data.apartmentId
} });



       return res.json(

         apt ? ({ success: true, data: apt } satisfies InternalResponse)

             : ({ success: false, error: 'Apartment not found' } satisfies
InternalResponse)

       );

     }



     default:

       return res.status(400).json({ success: false, error: 'Unknown action' }
satisfies InternalResponse);

   }

 } catch {

   return res.status(500).json({ success: false, error: 'Internal error' }
satisfies InternalResponse);

 }

});
```

Листинг 3 — contract-service: клиент для user-service (services/userClient.ts).

```
import axios from 'axios';

import { InternalRequest, InternalResponse } from '../../common/internal-api';



export interface UserDto {

 UserID: number;

 username: string;

 first_name: string;

 last_name: string;

 role: 'agent' | 'client';

}



export class UserClient {

 constructor(private readonly baseUrl: string) {}



 async getUserById(userId: number): Promise<UserDto | null> {

   const payload: InternalRequest<{ userId: number }> = {

     service: 'user-service',

     action: 'getUserById',

     data: { userId },

   };



   const resp = await
axios.post<InternalResponse<UserDto>>(`${this.baseUrl}/api/internal`, payload);
```

```
    if (resp.data.success && resp.data.data) return resp.data.data;

    return null;

  }

}
```

Листинг 4 - contract-service: клиент для property-service (services/propertyClient.ts).

```
import axios from 'axios';

import { InternalRequest, InternalResponse } from '../../common/internal-api';



export interface ApartmentDto {

  ApartmentID: number;

  Number: number;

  Square: number;

  Price: number;

  BuildingID: number;

}



export class PropertyClient {

  constructor(private readonly baseUrl: string) {}



  async getApartmentById(apartmentId: number): Promise<ApartmentDto | null> {

    const payload: InternalRequest<{ apartmentId: number }> = {

      service: 'property-service',

      action: 'getApartmentById',

      data: { apartmentId },

    };
```

```
   const resp = await
axios.post<InternalResponse<ApartmentDto>>(`${this.baseUrl}/api/internal`,
payload);



   if (resp.data.success && resp.data.data) return resp.data.data;

   return null;

 }

}
```

Вывод

В ходе выполнения лабораторной работы монолитное приложение сервиса аренды недвижимости было разделено на 3 независимых микросервиса: User Service, Property Service и Contract Service. Была настроена структура каждого сервиса, распределены REST-эндпоинты по доменам, реализована JWT-аутентификация через middleware, а также настроено межсервисное взаимодействие через внутренний HTTP-эндпоинт POST /api/internal и адаптеры на базе axios. 🆓🆓🆓

В результате микросервисы могут развиваться и масштабироваться независимо, при этом Contract Service выполняет агрегацию данных и валидацию межсервисных зависимостей (проверка пользователя и объекта недвижимости) перед операциями с контрактами.