САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа № 3

Выполнил:

Гуторова Инна

Группа К3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- выделить самостоятельные модули в вашем приложении;
- провести разделение своего API на микросервисы (минимум, их должно быть 3);
- настроить сетевое взаимодействие между микросервисами.

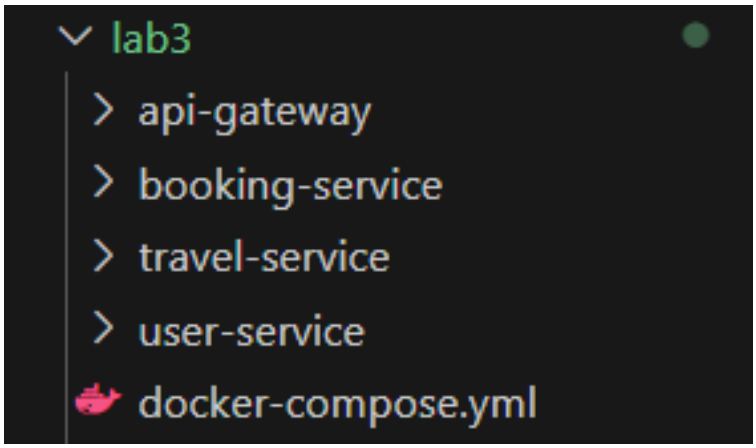
Ход работы

1. Выделение самостоятельных модулей

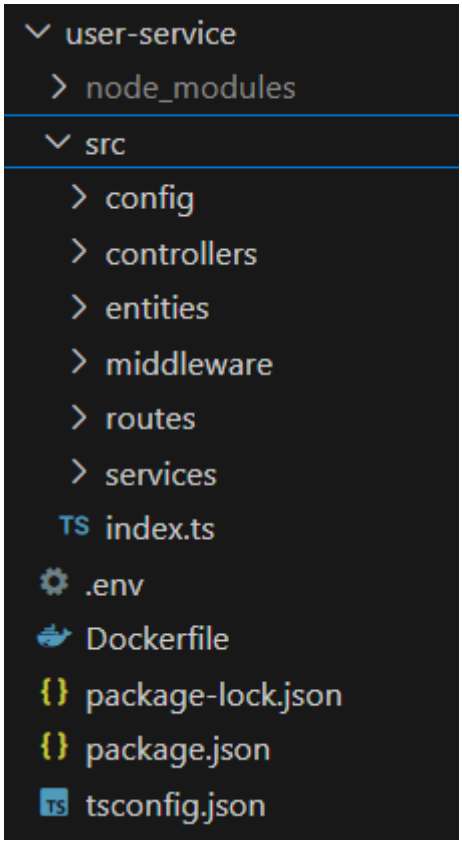
Исходное монолитное приложение было разделено на следующие микросервисы:

1. **User Service** – отвечает за аутентификацию и управление пользователями.
2. **Travel Service** – управляет данными о путешествиях, достопримечательностях, маршрутах и типах поездок.
3. **Booking Service** – обрабатывает бронирования и избранные записи пользователей.
4. **API Gateway** – обеспечивает единую точку входа для клиентов и маршрутизацию запросов к соответствующим сервисам.

Получившаяся структура всего проекта:



Каждый сервис (кроме api-gateway) в отдельности имеет вид:



2. Docker (подробнее в ЛР4)

Для каждого сервиса создан собственный Dockerfile и настроена оркестрация через docker-compose.yml.

Основные параметры:

- Каждый сервис работает в отдельном контейнере.
- Общая сеть travel-network для взаимодействия между сервисами.
- Использование PostgreSQL в качестве единой базы данных

3. Сетевое взаимодействие между сервисами

- Сервисы общаются через HTTP-запросы.
- API Gateway перенаправляет запросы на соответствующие сервисы (user-service, travel-service, booking-service).

- Для аутентификации используется JWT, проверяемый через user-service.

Для авторизации в travel-service и booking-service добавлен authClint. который отправляет запрос на верификацию токена в user-service.

```
class AuthClient {
    private readonly authServiceBaseUrl: string;

    constructor() {
        this.authServiceBaseUrl = process.env.USER_SERVICE_URL || 'http://user-service:3000';
    }

    async verifyToken(token: string): Promise<{ id: number; email: string; isAdmin: boolean }> {
        try {
            console.log(token)
            const response = await axios.get(`${this.authServiceBaseUrl}/api/auth/verify-token`, {
                headers: {
                    Authorization: `Bearer ${token}`,
                }
            });

            if (!response.data.user) {
                throw new Error('Invalid user data in response');
            }
```

В user-service добавлен эндпоинт:

authRoutes.get('/verify-token', AuthController.verifyToken);

В связи с этим адаптированы методы в middleware в сервисах кроме user-service

```
export const authenticate = async (req: Request, res: Response, next: NextFunction) => {
    try {
        const authHeader = req.headers.authorization;
        if (!authHeader || !authHeader.startsWith('Bearer ')) {
            return res.status(401).json({ message: 'Bearer token required' });
        }

        const token = authHeader.split(' ')[1];
        req.user = await authClient.verifyToken(token);
        next();
    } catch (error: any) {
        console.error('Authentication error:', error.message);
        res.status(401).json({
            message: error.message || 'Authentication failed',
            details: process.env.NODE_ENV === 'development' ? error.stack : undefined
        });
    }
};
```

```
export const authorizeAdmin = (req: Request, res: Response, next: NextFunction) => {
    if (!req.user?.isAdmin) {
        return res.status(403).json({ message: 'Admin access required' });
    }
    next();
};
```

Похожим образом реализована верификация данных для booking-service, добавлены userClient и travelClient, в которых по id можно получить информацию из других сервисов.

```
export class UserClient {
    private readonly baseUrl: string;

    constructor() {
        this.baseUrl = process.env.USER_SERVICE_URL || 'http://user-service:3000';
    }

    async getUserById(userId: number) {
        try {
            const response = await axios.get(`${this.baseUrl}/api/users/${userId}`, {
                timeout: 5000
            });
            return response.data;
        } catch (error) {
            throw new Error('Failed to fetch user data');
        }
    }
}

export const userClient = new UserClient();
```

4. Описание сервисов

4.1. User Service

Порт: 3000
Функционал:

- Регистрация и аутентификация пользователей.
- Управление профилями (CRUD).
- Генерация JWT-токенов.

Зависимости:

- PostgreSQL для хранения данных пользователей.

4.2. Travel Service

Порт: 3001
Функционал:

- Управление достопримечательностями (/api/attractions).
- Работа с медиафайлами (/api/media).
- Управление маршрутами (/api/routes).
- Типы путешествий (/api/travel-types).
- Поездки (/api/trips).

Зависимости:

- PostgreSQL для хранения данных.
- Интеграция с user-service для проверки аутентификации.

4.3. Booking Service

Порт: 3002
Функционал:

- Бронирования (/api/bookings).
- Избранные записи (/api/favorite).

Зависимости:

- PostgreSQL.
- Интеграция с user-service для аутентификации.

4.4. API Gateway

Порт: 3003
Функционал:

- Единая точка входа для всех запросов.
- Маршрутизация:

    - /api/auth, /api/users → user-service.
    - /api/attractions, /api/trips и др. → travel-service.
    - /api/bookings, /api/favorite → booking-service.
- Обработка ошибок и CORS.

6. Вывод

В ходе работы монолитное приложение было успешно разделено на микросервисы. Настроено взаимодействие между ними через API Gateway.

Итог: Микросервисная архитектура реализована, все сервисы работают корректно.