САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №3

Выполнил:

Корчагин Вадим

Группа
К3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Тема

Платформа для фитнес-тренировок и здоровья.

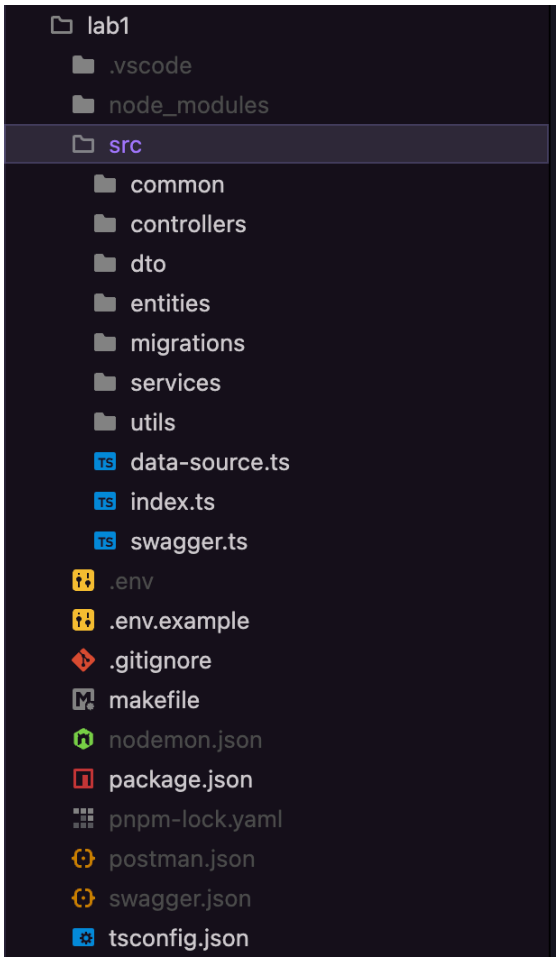
Задача

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

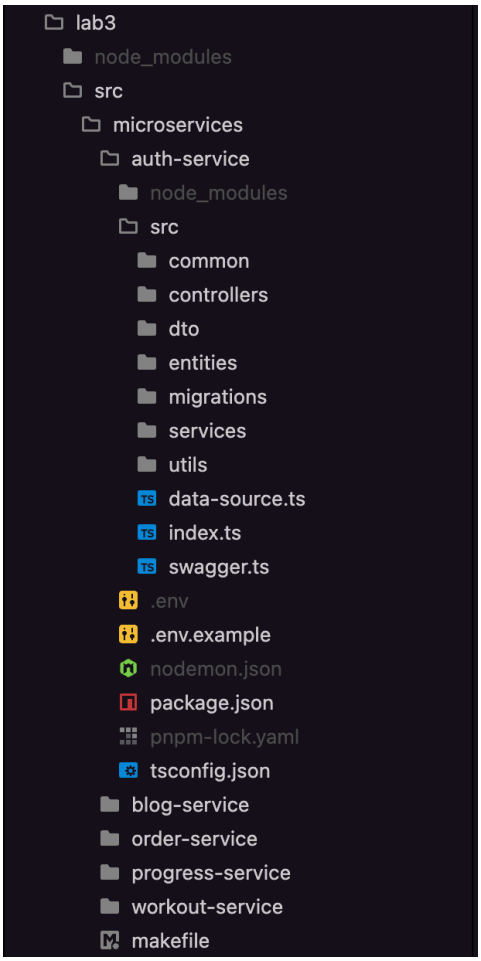
Ход работы

Выделение самостоятельных микросервисов

В процессе разработки приложения была проведена декомпозиция системы на независимые модули. Каждый модуль отвечает за свой домен, а все зависимости между ними минимизированы и оформлены через REST API. Ниже приведён список модулей, их назначение, ответственность и ключевые сущности.



Было - ЛР2



Стало - ЛР3

1. auth-service — сервис аутентификации и управления пользователями

Назначение: отвечает за регистрацию, вход, хранение пользователей, их ролей и генерацию JWT-токенов.

Основные сущности: User — пользователь, Role — роль пользователя.

2. blog-service — сервис управления блогом

Назначение: публикация блог-постов, добавление комментариев, получение материалов.

Основные сущности: BlogPost — пост в блоге, BlogComment — комментарий к посту.

- Содержит связи OneToMany/ManyToOne между постами и комментариями.
- Автоматическое добавление автора/комментатора через REST-запрос к auth-service.

3. order-service — сервис обработки заказов и платежей

Назначение: хранение информации о заказах пользователей и об их оплате.

Основные сущности: Order — заказ с привязкой к пользователю, Payment — статус оплаты заказа

- Связь: Order → Payment
- Получение пользователя по user_id через auth-service

4. workout-service — сервис тренировок и тренировочных планов

Назначение: управление программами тренировок, отдельными упражнениями и составлением планов.

Основные сущности: Workout — упражнение, TrainingPlan — тренировочный план, TrainingPlanWorkout — связь между планом и упражнениями

- Содержит только внутренние связи с сущностями

5. progress-service — сервис пользовательского прогресса

Назначение: отслеживание прогресса пользователя (вес, шаги, план).

Основные сущности: UserProgress — текущее состояние пользователя (вес, шаги, вода), UserTrainingPlan — связи пользователя с активными тренировочными планами

- Получение пользователя по user_id через auth-service

Итог

Каждый сервис:

- Изолирован (имеет свою БД, DTO, сущности, миграции)
- Подключается через Express и routing-controllers
- Документируется Swagger через routing-controllers-openapi
- Сервисы не делят ORM-связи — общение только через REST-запросы с использованием axios.

Пример изменений на микросервисе blogService

1. entities/: Удаление внешних связей

До:

- В модели BlogPost была @ManyToOne(() => User) — она нарушает границу микросервиса.

После:

- Все внешние связи на другие микросервисы (например, User) удалены.
- Оставлены только локальные связи, например:
    - BlogPost ↔ BlogComment (OneToMany, ManyToOne)
- Это позволяет сохранить референциальную целостность внутри домена без нарушения SRP.

```
@Entity("blog_posts")
export class BlogPost {
  @PrimaryGeneratedColumn()
  id: number

  @Column("int")
  author_id: number

  @ManyToOne(() => User, (user) => user.blogPosts)
  @JoinColumn({ name: "author_id" })
  author: User

  @Column("varchar", { length: 255 })
  title: string

  @Column("text")
  content: string

  @CreateDateColumn()
  created_at: Date

  @UpdateDateColumn()
  updated_at: Date

  @OneToMany(() => BlogComment, (comment) => comment.post)
  comments: BlogComment[]
}
```

Было - ЛР2

```
@Entity("blog_posts")
export class BlogPost {
  @PrimaryGeneratedColumn()
  id: number

  @Column("int")
  author_id: number

  @Column("varchar", { length: 255 })
  title: string

  @Column("text")
  content: string

  @CreateDateColumn()
  created_at: Date

  @UpdateDateColumn()
  updated_at: Date

  @OneToMany(() => BlogComment, (comment) => comment.post)
  comments: BlogComment[]
}
```

Стало - ЛР3

2. services/: Замена ORM-связей на HTTP-запросы

Что изменено:

Добавлен axios для вызова auth-service:

```
const response = await axios.get(`http://localhost:3000/users/id/${post.author_id}`);
```

Методы findAllWithRelations() и findOneWithRelations(id) теперь обогащают сущности извне, добавляя информацию о пользователе (author) или комментаторе (user). Нет жёсткой зависимости от структуры внешних таблиц. Можно кешировать ответы, масштабировать сервис независимо.

```
export class BlogPostService extends BaseService<BlogPost> {
  constructor() {
    super(BlogPost);
  }

  async findAllWithRelations() {
    return this.repository.find({ relations: ["author"] });
  }

  async findOneWithRelations(id: number) {
    return this.repository.findOne({ where: { id }, relations: ["author"] });
  }
}
```

Было - ЛР2

```ts
export class BlogPostService extends BaseService<BlogPost> {
  constructor() {
    super(BlogPost);
  }

  async findAllWithRelations() {
    const posts = await this.repository.find({ relations: ["comments"] });

    return await Promise.all(
      posts.map(async (post) => {
        const author = await this.fetchUser(post.author_id);
        return { ...post, author };
      })
    );
  }

  async findOneWithRelations(id: number) {
    const post = await this.repository.findOne({
      where: { id },
      relations: ["comments"],
    });
    if (!post) return null;

    const author = await this.fetchUser(post.author_id);
    return { ...post, author };
  }

  private async fetchUser(userId: number) {
    try {
      const response = await axios.get(`http://localhost:3000/users/id/${userId}`);
      return response.data;
    } catch {
      return null;
    }
  }
}
```

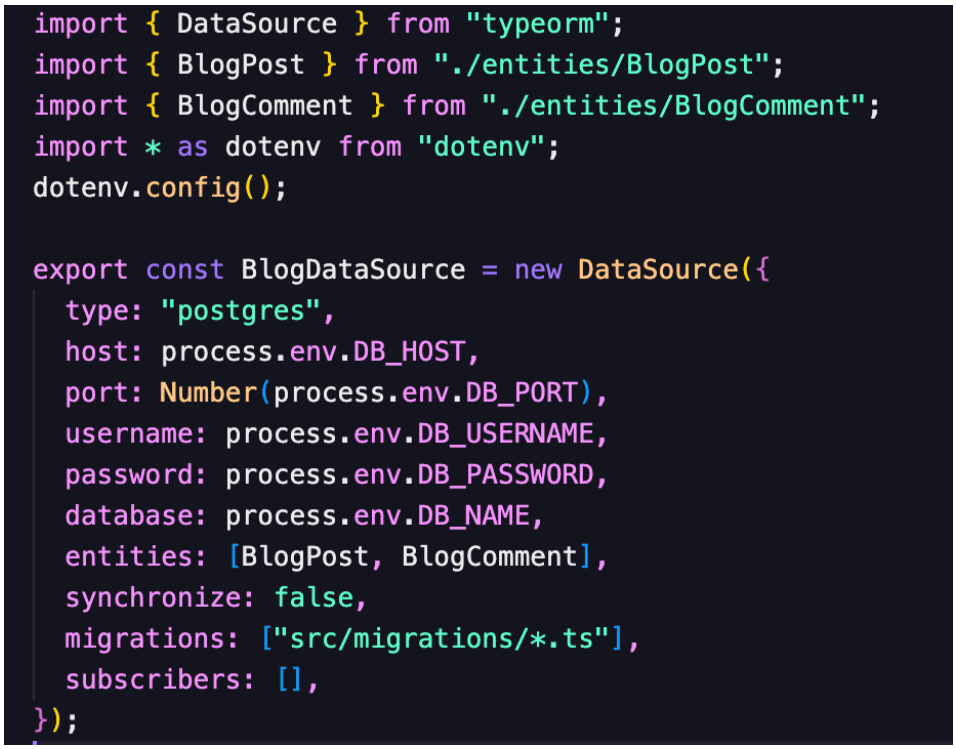
Стало - ЛР3

3. data-source.ts: Изолированная конфигурация базы

Изменения:

Убраны общие таблицы: в entities подключены только локальные (BlogPost, BlogComment).

Используются только локальные .env файлы у каждого сервиса.



```
import { DataSource } from "typeorm";
import { BlogPost } from "./entities/BlogPost";
import { BlogComment } from "./entities/BlogComment";
import * as dotenv from "dotenv";
dotenv.config();

export const BlogDataSource = new DataSource({
  type: "postgres",
  host: process.env.DB_HOST,
  port: Number(process.env.DB_PORT),
  username: process.env.DB_USERNAME,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  entities: [BlogPost, BlogComment],
  synchronize: false,
  migrations: ["src/migrations/*.ts"],
  subscribers: [],
});
```

4. index.ts: Инициализация микросервиса

Что сделано:

- Инициализация БД (через BlogDataSource.initialize()).
- Swagger-документация развернута на /docs с генерацией через routing-controllers-openapi.
- Каждый сервис запускается на своём порту (в данном случае 3004).
- cors: true — позволяет другим микросервисам обращаться к API.

```
const app = express();
const port = 3004;

BlogDataSource.initialize().then(() => {
  console.log("Blog DB connected");

  useExpressServer(app, {
    controllers: [BlogPostController, BlogCommentController],
    routePrefix: "",
    cors: true,
    defaultErrorHandler: true,
  });

  useSwagger(app, {
    controllers: [BlogPostController, BlogCommentController],
    serviceName: "Blog Service",
    port
  });

  app.listen(port, () => {
    console.log(`🚀 Blog service running at http://localhost:${port}`);
  });
});
```

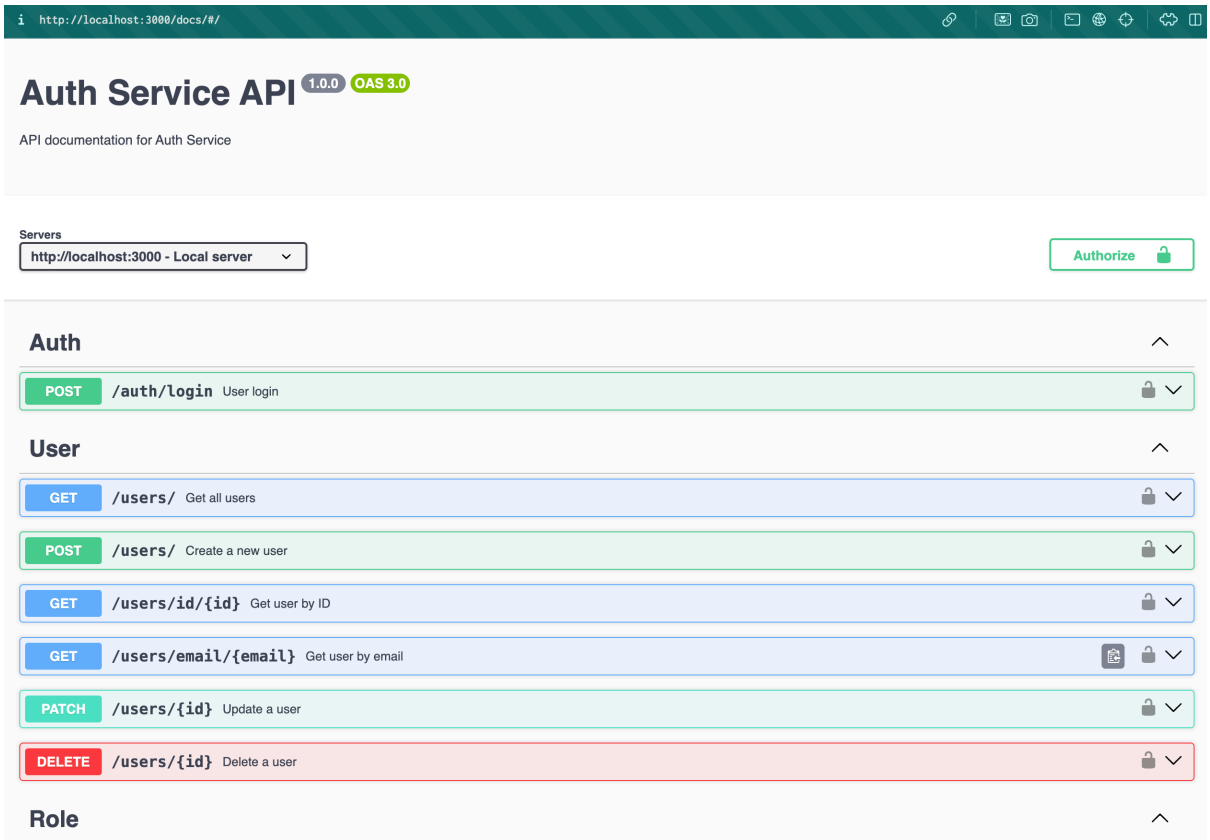
Сетевое взаимодействие

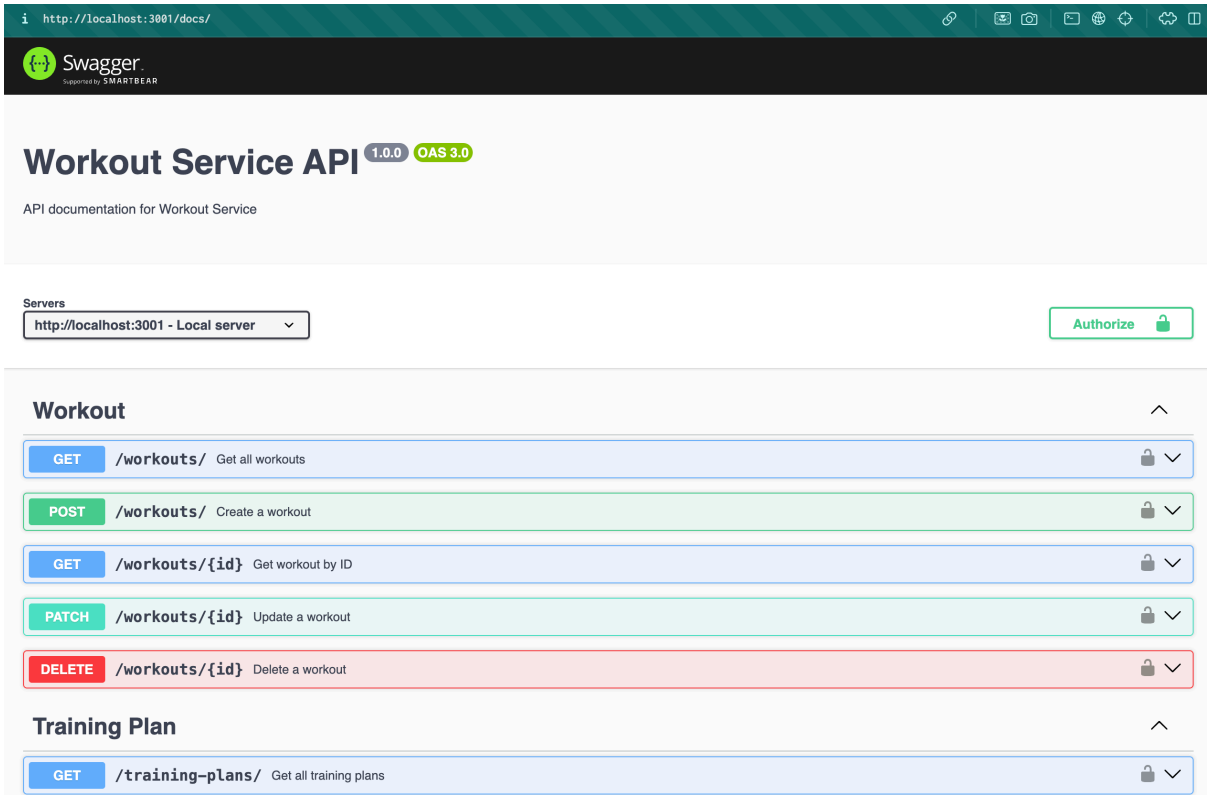
Все микросервисы взаимодействуют через HTTP/REST API.

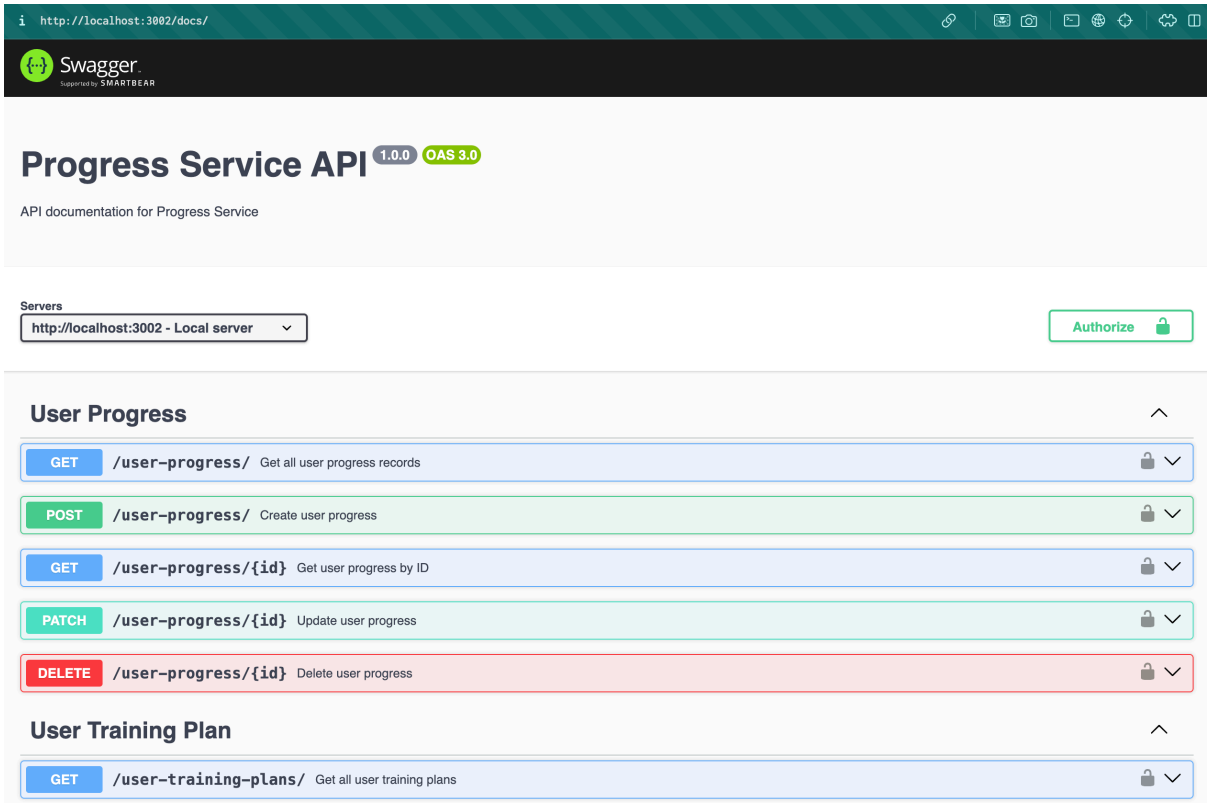
Используется библиотека axios во всех сервисах, где требуется доступ к данным других сервисов.

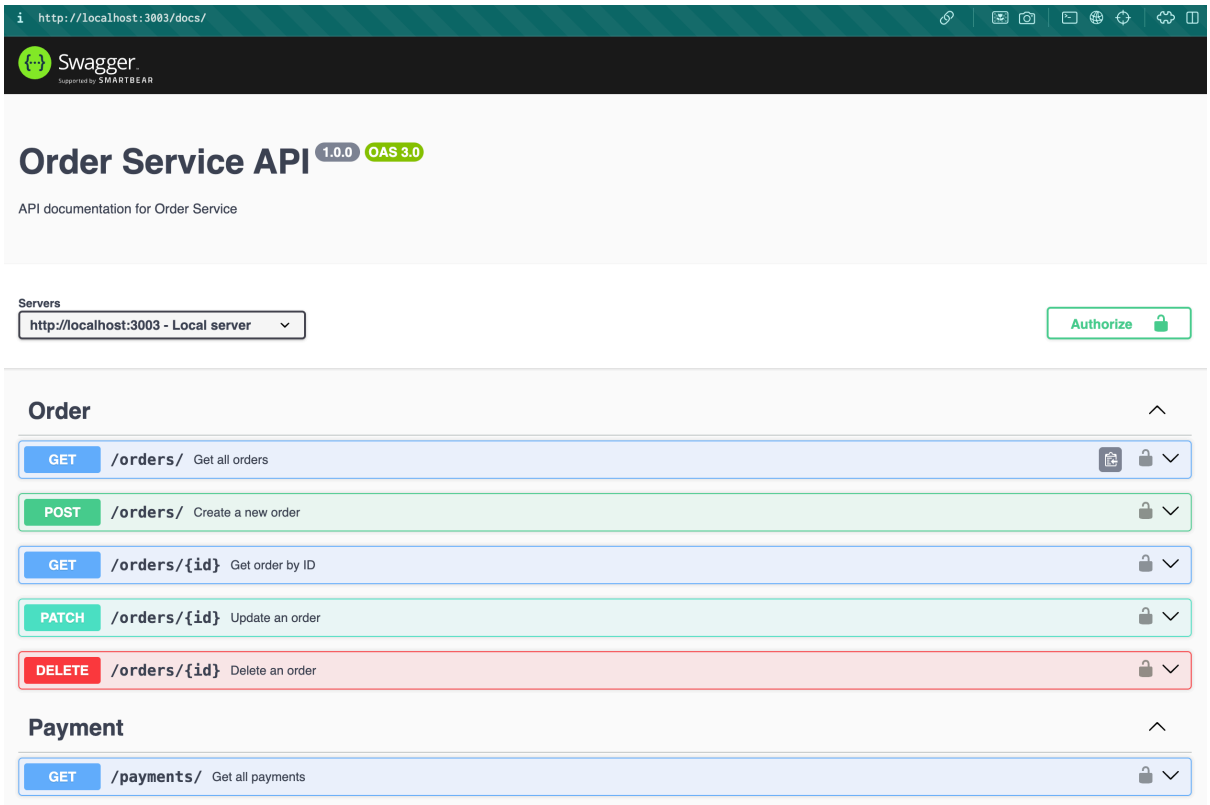
Каждый сервис изолирован и работает на собственном порту:

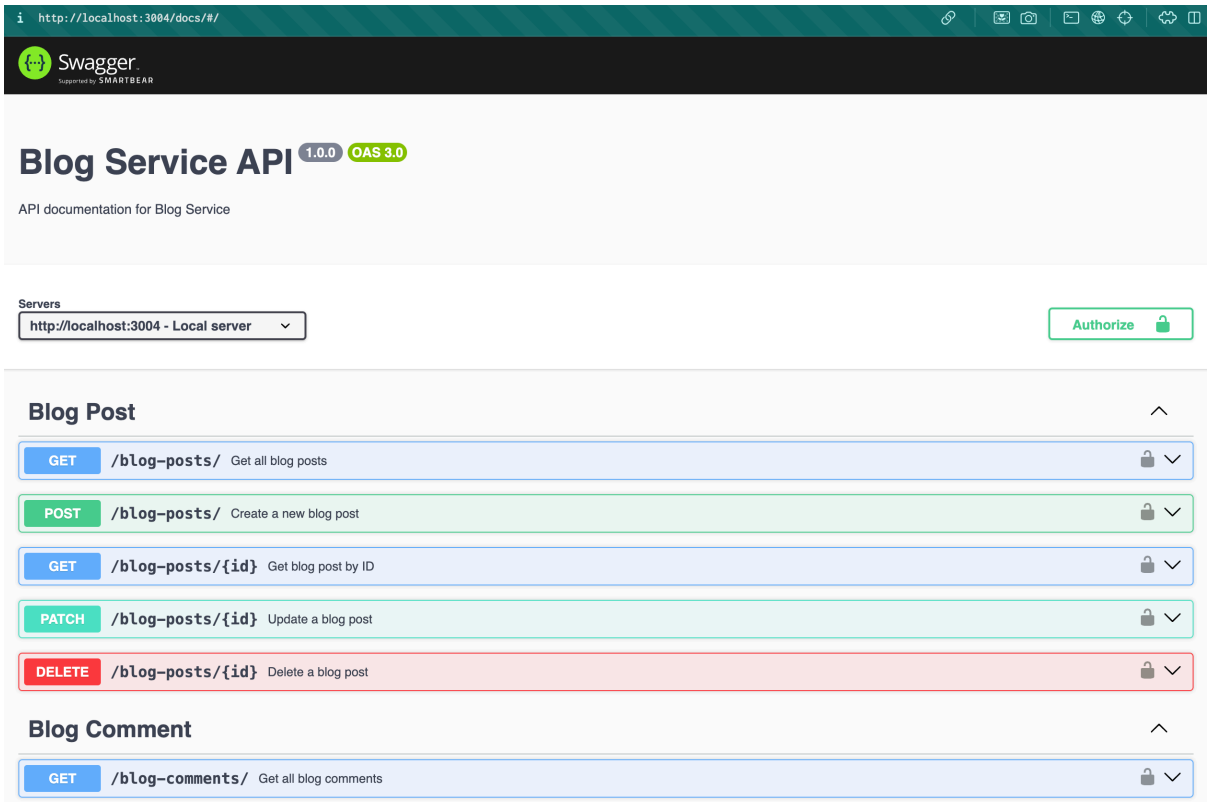
- auth-service: 3000
- workout-service: 3001
- progress-service: 3002
- order-service: 3003
- blog-service: 3004




http://localhost:3001/docs/
Swagger
Supported by SMARTBEAR
Workout Service API
1.0.0
OAS 3.0
API documentation for Workout Service
Servers
http://localhost:3001 - Local server
Authorize
Workout
GET /workouts/ Get all workouts
POST /workouts/ Create a workout
GET /workouts/{id} Get workout by ID
PATCH /workouts/{id} Update a workout
DELETE /workouts/{id} Delete a workout
Training Plan
GET /training-plans/ Get all training plans


http://localhost:3002/docs/
Swagger
Supported by SMARTBEAR
Progress Service API
1.0.0
OAS 3.0
API documentation for Progress Service
Servers
http://localhost:3002 - Local server
Authorize
User Progress
GET /user-progress/ Get all user progress records
POST /user-progress/ Create user progress
GET /user-progress/{id} Get user progress by ID
PATCH /user-progress/{id} Update user progress
DELETE /user-progress/{id} Delete user progress
User Training Plan
GET /user-training-plans/ Get all user training plans


http://localhost:3003/docs/
Swagger
Supported by SMARTBEAR
Order Service API
1.0.0
OAS 3.0
API documentation for Order Service
Servers
http://localhost:3003 - Local server
Authorize
Order
GET /orders/ Get all orders
POST /orders/ Create a new order
GET /orders/{id} Get order by ID
PATCH /orders/{id} Update an order
DELETE /orders/{id} Delete an order
Payment
GET /payments/ Get all payments


http://localhost:3004/docs/#/
Swagger
Supported by SMARTBEAR
Blog Service API
1.0.0
OAS 3.0
API documentation for Blog Service
Servers
http://localhost:3004 - Local server
Authorize
Blog Post
GET /blog-posts/ Get all blog posts
POST /blog-posts/ Create a new blog post
GET /blog-posts/{id} Get blog post by ID
PATCH /blog-posts/{id} Update a blog post
DELETE /blog-posts/{id} Delete a blog post
Blog Comment
GET /blog-comments/ Get all blog comments

Выводы

В ходе лабораторной работы был выполнен переход от монолитной архитектуры к микросервисной. Каждый сервис получил свою зону ответственности, отдельную базу данных и API.

Связи между микросервисами реализованы через REST-запросы с использованием axios, что обеспечило слабую связанность и гибкость системы.

Проект стал модульным, масштабируемым и готовым к развертыванию в распределённой среде.