САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа № 3

“DI, IoC, Развёртывание, микросервисы, CI/CD”

Выполнил:
Ле Хоанг Чыонг
Чан Дык Минь

Группа:
K33392

Проверил:
Добряков Д. И.

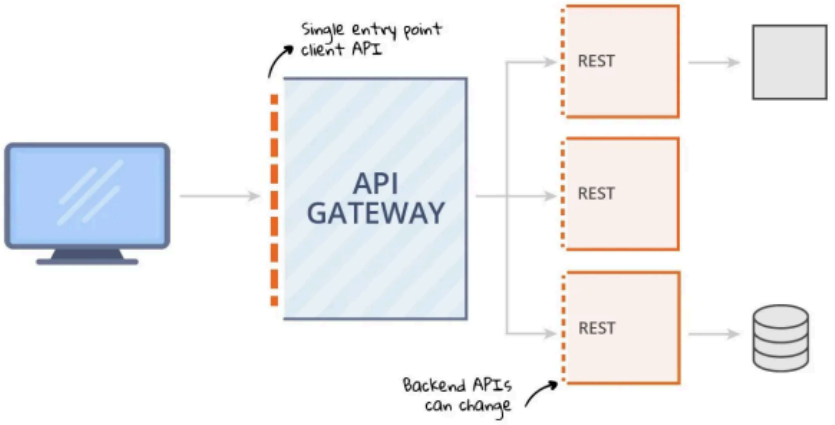
Санкт-Петербург
2024 г.

Задача

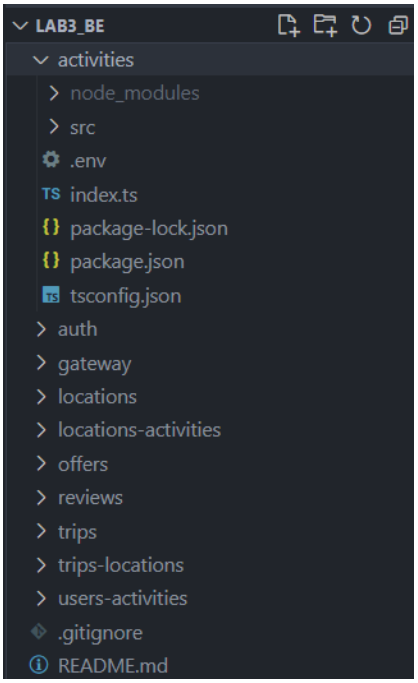
Необходимо реализовать отдельный микросервис, выполняющий какую-либо содержательную функцию из всего арсенала функций вашего приложения

Ход работы

По сути, архитектура микросервисов — это архитектурный стиль, в котором приложение будет разделено на множество более мелких и независимых сервисов, называемых микросервисами или сервисами, причем сервисы соединяются вместе, образуя большое приложение.
В своем проекте «планирование поездки» я перейду с монолитной на микросервисную архитектуру согласно схеме ниже. Сервисы авторизации, путешествий, определения местоположения и активности создаются как микросервисы. Общая схема микросервиса с использованием портов:



Структура проекта:



Мой проект был разделен на различные сервисы, такие как сервисы пользователей, сервисы поездок, сервисы определения местоположения...

Каждый сервис имеет свою собственную базу данных и контроллеры, а также отдельные маршруты для независимой работы.

Gateway

```
import express from "express";
import proxy from "express-http-proxy";

const router: express.Router = express.Router();

router.use('/users', proxy('http://localhost:8001'));
router.use('/trips', proxy('http://localhost:8002'));
router.use('/activities', proxy('http://localhost:8003'));
router.use('/users-activities', proxy('http://localhost:8004'));
router.use('/reviews', proxy('http://localhost:8005'));
router.use('/locations', proxy('http://localhost:8006'));
router.use('/locations-activities', proxy('http://localhost:8007'));
router.use('/trips-locations', proxy('http://localhost:8008'));
router.use('/offers', proxy('http://localhost:8009'));

export default router;
```

Каждый маршрут в gateway направляет запросы к службе через прокси-сервер Express HTTP, что создает гибкий механизм управления запросами.

Переадресация на auth микросервис при проверке авторизации

```
export const auth = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');

    if (!token) {
      throw new Error('Missing token');
    }

    const decoded = jwt.verify(token, SECRET_KEY) as JwtPayload;
    console.log(decoded.id)
    const userData = await axios.post('http://localhost:8000/users/v1/user', {
      id: decoded.id
    });


    if (!userData.data) {
      throw new Error('User not found');
    }

    (req as CustomRequest).userId = userData.data.id;
    next();
  } catch (err) {
    console.error('Authentication error:', err);
    res.status(401).send('Please authenticate');
  }
};
```

Для остальных сервисов (Поездка, Местоположение, Активность, Предложение) я реализовал контроллеры и сервисы как в предыдущей лабе 3 и ничего особенного.

Например, Trip Service:

1. routes/index.ts

```ts
import express from "express"
import TripController from "../../../controllers/trips/trip"
import { auth } from "../../../middlewares/auth"
const router: express.Router = express.Router()

const controller: TripController = new TripController()

router.post('/',auth, controller.create);
router.get('/',auth, controller.getAll);
router.get('/trip', controller.getById);
router.delete('/:id', controller.delete);
router.put('/:trip_id', controller.update);


export default router
```

2. services/trip/trip.ts

```ts
import { Sequelize } from 'sequelize-typescript';
import { Trip } from '../../models/trips/trip';
import { Op } from 'sequelize';

class TripService {

    async create(tripData: any): Promise<Trip> {
        try {
            const trip = await Trip.create(tripData);
            console.log("tripData: ", tripData)
            return trip;
        } catch (error) {
            throw error;
        }
    }

    async getById(id: string): Promise<Trip> {
        try {
            const trip = await Trip.findByPk(id, {
                include: [{ association: 'locations' }]
            });
            if (!trip)
                throw new Error('Trip not found');
            return trip;
        } catch (error) {
            throw error;
        }
    }

    async getAll(): Promise<Trip[]> {
        try {
            const trips = await Trip.findAll();
            return trips;
        } catch (error) {
            throw error;
        }
    }
```

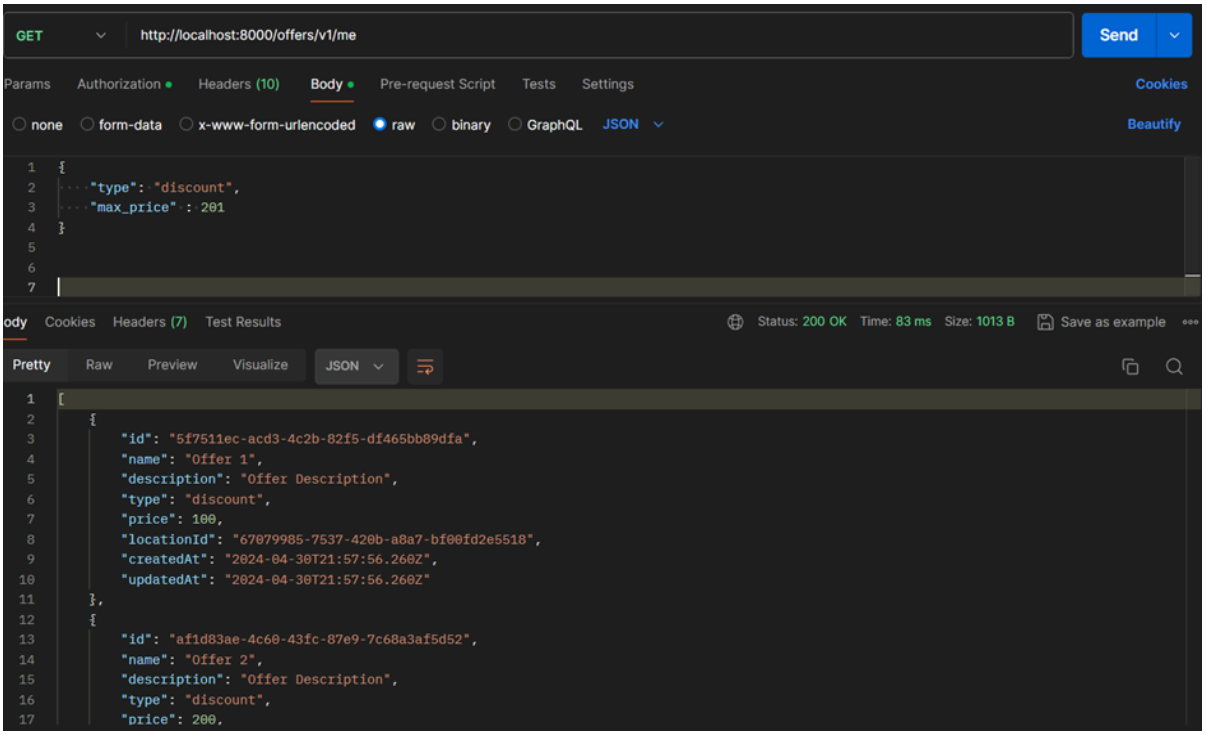
3. controllers/trip/trip.ts

```
export default class TripController {
    private tripService: TripService;

    constructor() {
        this.tripService = new TripService();
    }

    create = async (request: any, response: any) => {
        console.log(request.body);
        const dateStartString = request.body.dateStart;
        const dateEndString = request.body.dateEnd;
        const dateStart = new Date(dateStartString);
        const dateEnd = new Date(dateEndString);
        try {
            const { body } = request;
            const trip: any = await this.tripService.create({ ...body, dateStart: dateStart, dateEnd: dateEnd });
            response.status(201).send(trip)
        } catch (error: any) {
            response.status(404).send(getErrorMessage(error))
        }
    }

    getAll = async (request: any, response: any) => {
        try {
            const trips: any = await this.tripService.getAll()

            response.status(201).send(trips)
        } catch (error: any) {
            response.status(404).send(getErrorMessage(error))
        }
    }
```

Реализованы другие сложные API: Вместо прямого импорта контроллеров, как в предыдущей лабораторной работе, я буду совершать вызовы API из разных сервисов.

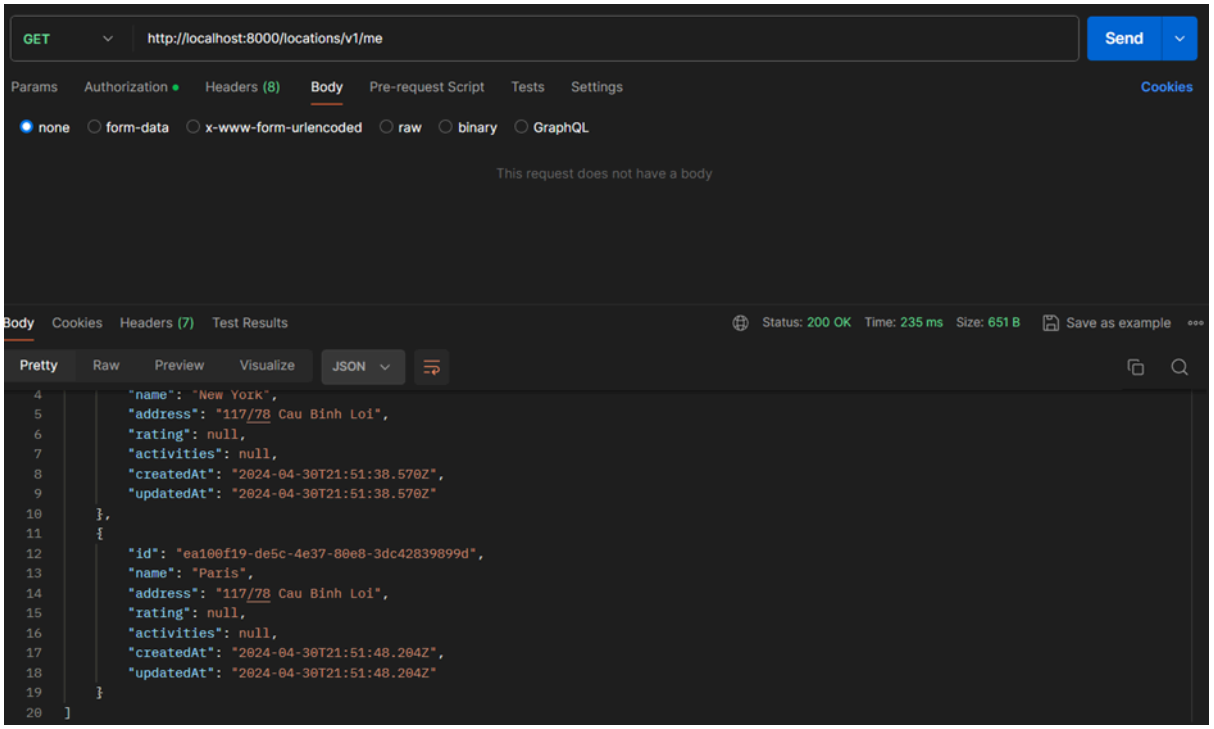
- Найдите предложения, которые актуальны для текущего пользователя

```
async getOfferForUser(userId: string, type?: string, price?: number): Promise<Offer[]> {
    try {
        const response = await axios.post('http://localhost:8000/users-activities/v1/all-user-activity', {
            userId: userId
        });
        const activities = response.data
        if (!activities || activities.length === 0) {
            return [];
        }
        const activityIds = activities.map((activity: { activityId: string; }) => activity.activityId);

        const locationSet: Set<string> = new Set();

        const locationPromises = activityIds.map(async (activityId: string) => {
            try {
                const response = await axios.post('http://localhost:8000/locations-activities/v1/activity', {
                    activityId: activityId
                });
                const locationsWithActivity = response.data;
                locationsWithActivity.forEach((location: { location_id: string; }) => {
                    locationSet.add(location.location_id);
                });
            } catch (error) {
                console.error(`Error fetching locations for activityId ${activityId}:`, error);
            }
        });

        await Promise.all(locationPromises);


        const locationIds: string[] = [...locationSet];

        const offerPromises: Promise<Offer[]>[] = [];

        locationIds.forEach(locationId => {
            let whereCondition: any = { locationId: locationId };
            if (type) {
                whereCondition.type = type;
            }
            if (price) {
                whereCondition.price = { [Op.lte]: price };
            }
```



- Найдите места, которые актуальны для текущих пользователей

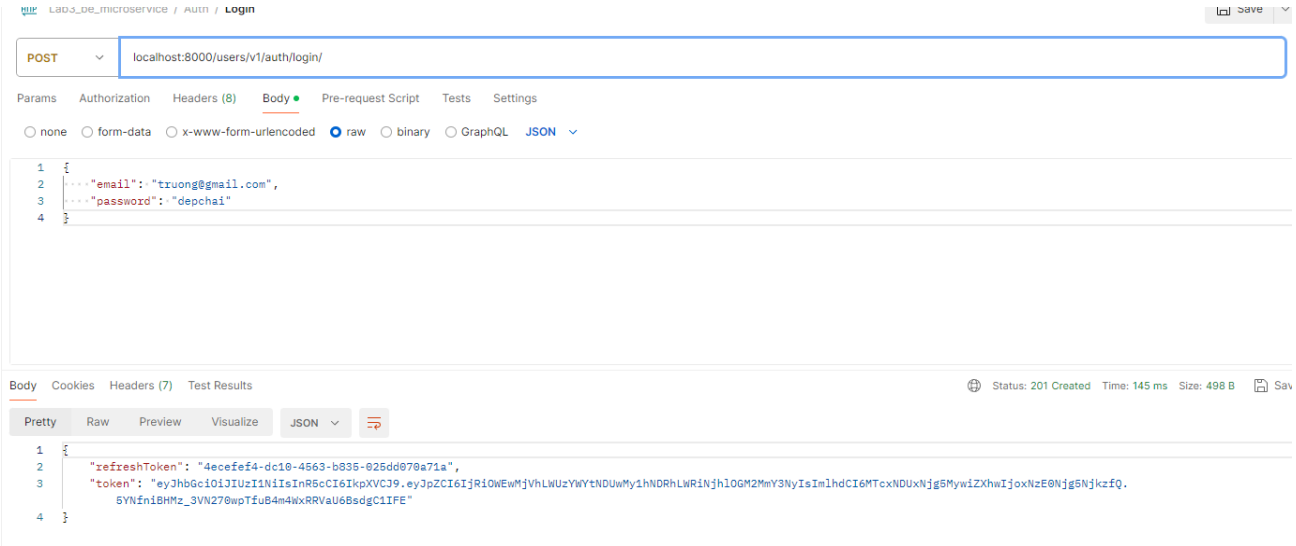
```
async getLocationForUser(token : string): Promise<Location[]> {
    try {
        const response = await axios.get('http://localhost:8000/users-activities/v1/all-user-activity',  {
            headers: {
                Authorization: `Bearer ${token}`
            }
        });
        const activities = response.data
        if (!activities || activities.length === 0) {
            return [];
        }
        const activityIds = activities.map((activity: { activityId: string; }) => activity.activityId);

        const locationSet: Set<string> = new Set();
        console.log(activityIds)
        const locationPromises = activityIds.map(async (activityId: string) => {
            try {
                const response = await axios.post('http://localhost:8000/locations-activities/v1/activity', {
                    activityId: activityId
                }, {
                    headers: {
                        Authorization: `Bearer ${token}`
                    }
                });
                const locationsWithActivity = response.data;
                locationsWithActivity.forEach((location: { locationId: string; }) => {
                    locationSet.add(location.locationId);
                });
            } catch (error) {
                console.error(`Error fetching locations for activityId ${activityId}:`, error);
            }
        });
```



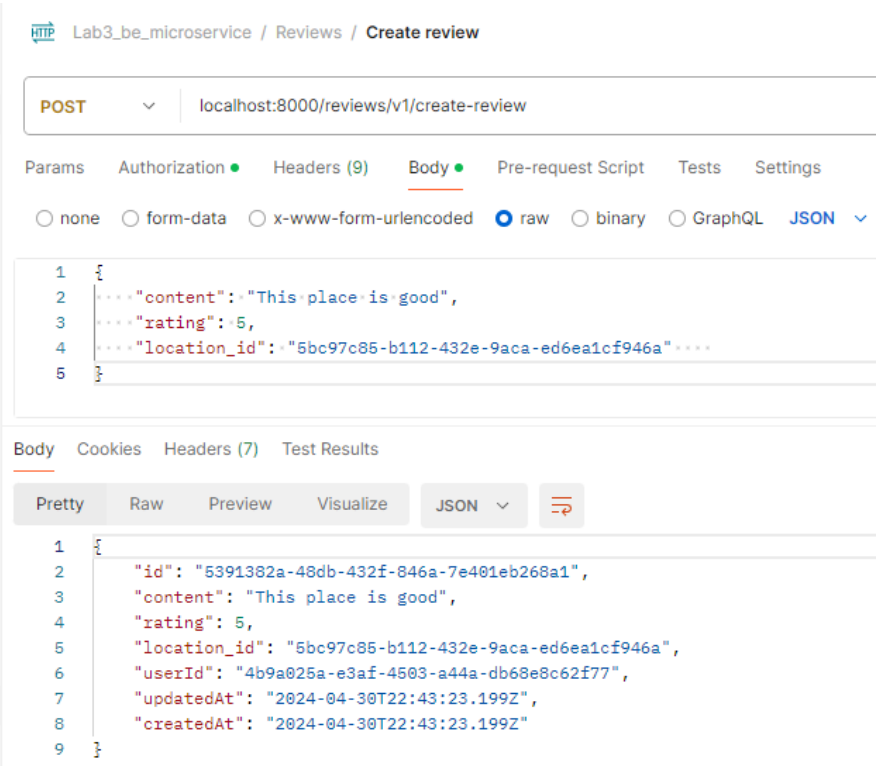
Протестируем работу API «Locations», «Reviews», «Аутентификация» и Gateway.

Сначала мы запустим API:

- Авторизация на порту 8001
- Reviews о порте 8005
- Locations находится на порту 8006.
- Gateway находится на порту 8000.



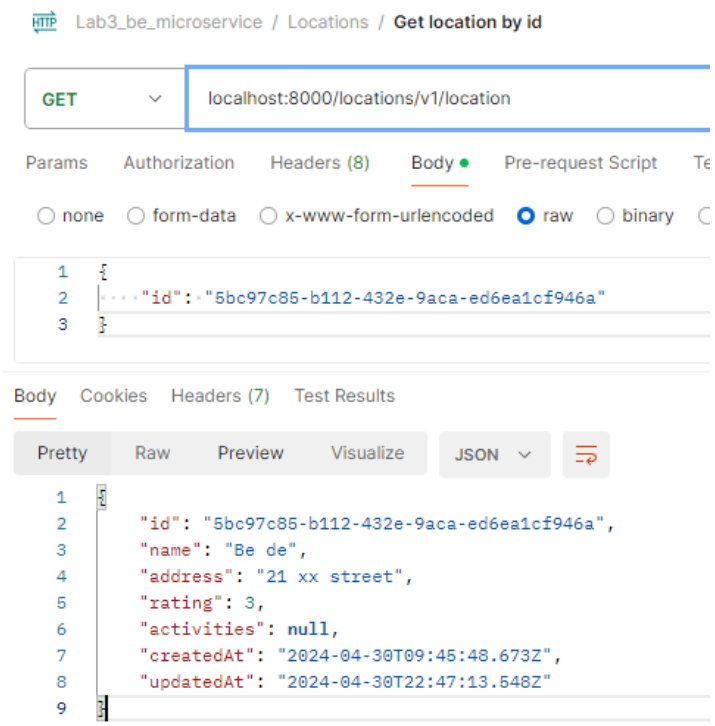
Мы видим, что при вызове localhost:8000/users маршрут будет перенаправлен на маршрут localhost:8001.



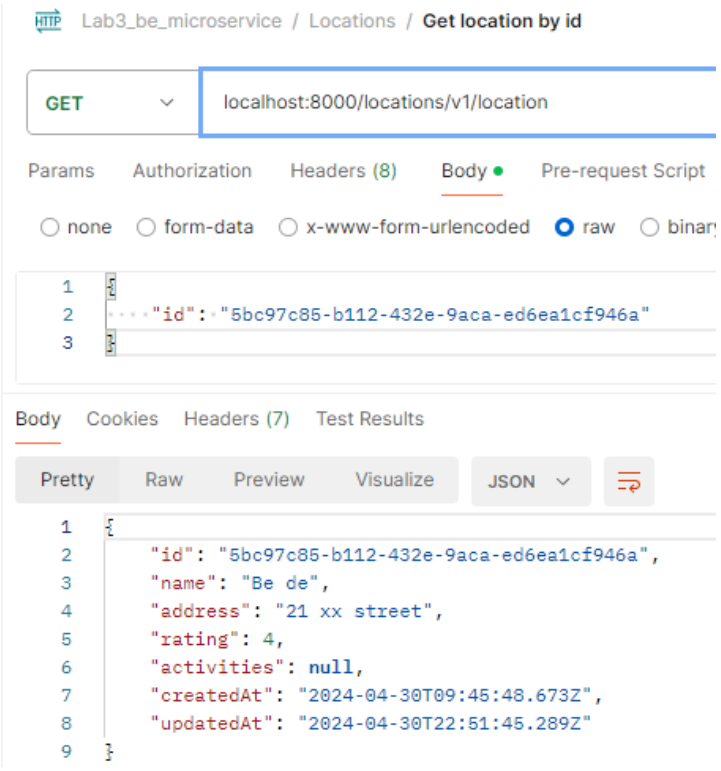
Аналогичным образом мы создаем обзор по маршруту localhost:8000/reviews/v1/create-review.

```
@AfterCreate
static async updateLocationRating(instance: Review) {
  try {
    // Fetch location data
    const response = await axios.get(`http://localhost:8000/locations/v1/location`, {
      data: {
        id: instance.location_id,
      },
    });
    const location = response.data;

    if (location) {
      const reviews = await Review.findAll({
        where: {
          location_id: instance.location_id,
        },
      });

      const totalRating = reviews.reduce((sum, review) => sum + review.rating, 0);
      const averageRating = totalRating / reviews.length;
      const roundedRating = Math.round(averageRating);

      // Update location rating
      await axios.put(`http://localhost:8000/locations/v1/update-location`, {
        id: instance.location_id,
        rating: roundedRating,
      });
    }
  } catch (error) {
    console.error('Error updating location rating:', error);
  }
}
```

При создании отзывов мы получим API-интерфейс местоположения по адресу localhost:8000/locations, чтобы рассчитать рейтинг местоположения.



Рейтинг локации до добавления 1 Оценка отзыва — 5



Рейтинг локации после добавления 1 Оценка отзыва — 5.

Вывод

В рамках третьей лабораторной работы я узнал о концепциях, преимуществах и недостатках стиля микросервисной архитектуры. Я научился это применять и переводить проект с монолитной архитектуры на микросервис.