САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Реализация приложения на базе микросервисной архитектуры

Выполнил:

Петухов Семён

Группа
К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Цель
Реализовать систему на базе микросервисной архитектуры

Ход работы

В ходе работы был реализован backend для системы поиска вакансий на базе микросервисной архитектуры

Общая архитектура приложения

Состав системы:

- **gateway**: API-шлюз, единственная точка входа для клиента.
- **user-service**: сервис пользователей
- **resume-service**: сервис резюме
- **vacancy-service**: сервис вакансий.
- **skill-service**: сервис навыков.
- **postgres**: единая БД resume_finder для всех сервисов

Gateway
Данный сервис обеспечивает корректное взаимодействие со всеми модулями системы, позволяет проксировать запросу к нужному микросервису.

```
const app = express();

app.use("/user-service", createProxyMiddleware({ target: "http://user-
service:3001", changeOrigin: true }));
app.use("/resume-service", createProxyMiddleware({ target: "http://resume-
service:3002", changeOrigin: true }));
app.use("/vacancy-service", createProxyMiddleware({ target: "http://vacancy-
service:3003", changeOrigin: true }));
app.use("/skill-service", createProxyMiddleware({ target: "http://skill-
service:3005", changeOrigin: true }));

app.listen(3000, () => {
    console.log("API Gateway running on port 3000");
});
```

После настройки прокси-маршрутов запускается сервер на 3000 порте

Resume-service

Назначение и ответственность

- Управление резюме соискателей
- Управление уровнями образования
- Управление опытом работы
- Связывание резюме с навыками

Реализует данные сущности:

- Education — уровни образования,
- Resume — резюме пользователя,
- WorkExperience — отдельные записи опыта работы,
- ResumeSkills — связь резюме с навыками из внешнего skill-service.

Микросервис реализует CRUD-запросы к БД для сущностей относящихся к резюме.

Пример одного из контроллеров (educationController.ts)

```
import { Request, Response } from "express";
import { AppDataSource } from "../data-source";
import { Education } from "../models/educationModel";

const repo = AppDataSource.getRepository(Education);

// Получить все образовательные записи
export const getAllEducations = async (_: Request, res: Response) => {
    const items = await repo.find();
    res.json(items);
};

// Получить образовательную запись по ID
export const getEducationById = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        res.status(400).json({ message: "Invalid ID format" });
        return;
    }

    const item = await repo.findOneBy({ id });
    if (!item) {
        res.status(404).json({ message: "Education not found" });
        return;
    }

    res.json(item);
};

// Создать образовательную запись
export const createEducation = async (req: Request, res: Response) => {
    const { education_level } = req.body;

    if (!education_level || typeof education_level !== "string" ||
education_level.trim() === "") {
        res.status(400).json({
            message: "Missing or invalid required field: education_level"
        });
        return;
    }

    try {
        const education = repo.create({ education_level:
education_level.trim() });
        await repo.save(education);
        res.status(201).json(education);
        return;
    } catch (error) {
        console.error("Error creating education:", error);
        res.status(500).json({
            message: "Internal server error"
        });
```

```
        return;
    }
};

// Обновить образовательную запись
export const updateEducation = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        res.status(400).json({ message: "Invalid ID format" });
        return;
    }

    const item = await repo.findOneBy({ id });
    if (!item) {
        res.status(404).json({ message: "Education not found" });
        return;
    }

    const { education_level } = req.body;
    if (!education_level) {
        res.status(400).json({ message: "Missing required field:
education_level" });
        return;
    }

    item.education_level = education_level;
    await repo.save(item);
    res.json(item);
};

// Удалить образовательную запись
export const deleteEducation = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        res.status(400).json({ message: "Invalid ID format" });
        return;
    }

    const result = await repo.delete(id);
    if (result.affected === 0) {
        res.status(404).json({ message: "Education not found" });
        return;
    }

    res.json({ deleted: result.affected });
};
```

Взаимодействие с другими сервисами

- Зависимость от User Service: проверка существования пользователя
- Зависимость от Skill Service: проверка существования навыков при создании связей
- Используется Vacancy Service: при создании откликов проверяется существование резюме

User Service

Назначение и ответственность

- Регистрация и аутентификация пользователей

- Управление профилями пользователей
- Выдача JWT токенов для доступа к защищенным ресурсам
- Валидация ролей пользователей (соискатель/работодатель)

authMiddleWare.ts

```
import { Request, Response, NextFunction } from "express";
import jwt from "jsonwebtoken";

const JWT_SECRET = process.env.JWT_SECRET || "test_jwt_secret";

export const verifyToken = (req: Request, res: Response, next: NextFunction)
=> {
    const authHeader = req.headers.authorization;
    console.log(authHeader);
    if (!authHeader || !authHeader.startsWith("Bearer ")) {
        res.status(401).json({ message: "Authorization token missing or
invalid" });
        return;
    }

    const token = authHeader.split(" ")[1];

    try {
        const decoded = jwt.verify(token, JWT_SECRET);
        (req as any).user = decoded;
        next();
    } catch (error) {
        res.status(403).json({ message: "Invalid or expired token" });
        return;
    }
};
```

userController.ts

```
import { Request, Response } from "express";
import { AppDataSource } from "../data-source";
import { User, UserRole } from "../models/userModel";
import bcrypt from "bcrypt";
import axios from "axios";
import jwt from "jsonwebtoken";

const userRepo = AppDataSource.getRepository(User);
const JWT_SECRET = process.env.JWT_SECRET || "test_jwt_secret";

// Получение всех пользователей
export const getAllUsers = async (req: Request, res: Response): Promise<void>
=> {
    try {
        const users = await userRepo.find();
        res.json(users);
    } catch (error) {
        console.error("Error fetching users:", error);
        res.status(500).json({ message: "Internal server error" });
    }
};

// Получение пользователя по ID
export const getUserById = async (req: Request, res: Response): Promise<void>
=> {
    try {
        const user = await userRepo.findOneBy({ id: parseInt(req.params.id)
});
        if (!user) {
            res.status(404).json({ message: "User not found" });
```

```
            return;
        }
        res.json(user);
    } catch (error) {
        console.error("Error fetching user:", error);
        res.status(500).json({ message: "Internal server error" });
    }
};

// Создание пользователя
export const createUser = async (req: Request, res: Response) => {
    try {
        const { username, email, password, role, companyId } = req.body;

        if (!username || !email || !password) {
            res.status(400).json({
                message: "Missing required fields: username, email, or
password",
            });
            return;
        }

        const existingUser = await userRepo.findOneBy({ email: email.trim()
});
        if (existingUser) {
            res.status(409).json({ message: "User with this email already
exists" });
            return;
        }

        if (role && !Object.values(UserRole).includes(role)) {
            res.status(400).json({ message: `Invalid role. Valid roles:
${Object.values(UserRole).join(", ")}` });
            return;
        }

        if (companyId) {
            const companyResponse = await axios.get(`http://vacancy-
service:3003/vacancy-service/company/${companyId}`);
            if (companyResponse.status !== 200 || !companyResponse.data) {
                res.status(400).json({ message: "Invalid company ID" });
                return;
            }
        }

        const hashedPassword = await bcrypt.hash(password, 10);

        const user = userRepo.create({
            username: username.trim(),
            email: email.trim(),
            password: hashedPassword,
            role: role || UserRole.SEEKER,
            companyId: companyId || null,
        });

        await userRepo.save(user);
        res.status(201).json(user);
        return;
    } catch (error: any) {
        if (axios.isAxiosError(error) && error.response?.status === 404) {
            res.status(400).json({ message: "Invalid company ID" });
            return;
        }
        console.error("Error creating user:", error);
        res.status(500).json({ message: "Internal server error", error });
        return;
```

```
    }
};

// Обновление пользователя
export const updateUser = async (req: Request, res: Response): Promise<void>
=> {
    try {
        const user = await userRepo.findOneBy({ id: parseInt(req.params.id)
});
        if (!user) {
            res.status(404).json({ message: "User not found" });
            return;
        }

        const { companyId, ...rest } = req.body;

        if (companyId !== undefined) {
            if (companyId !== null) {
                const companyResponse = await axios.get(`http://vacancy-
service:3003/vacancy-service/company/${companyId}`);
                if (companyResponse.status !== 200 || !companyResponse.data)
{
                    res.status(400).json({ message: "Invalid company ID" });
                    return;
                }
            }
            user.companyId = companyId;
        }

        userRepo.merge(user, rest);

        if (rest.password) {
            user.password = await bcrypt.hash(rest.password, 10);
        }

        await userRepo.save(user);
        res.json(user);
    } catch (error: any) {
        if (axios.isAxiosError(error) && error.response?.status === 404) {
            res.status(400).json({ message: "Invalid company ID" });
        }
        console.error("Error updating user:", error);
        res.status(500).json({ message: "Internal server error" });
    }
};

// Удаление пользователя
export const deleteUser = async (req: Request, res: Response) => {
    try {
        const result = await userRepo.delete(parseInt(req.params.id));
        if (result.affected === 0) {
            res.status(404).json({ message: "User not found" });
            return;
        }
        res.json({ deleted: result.affected });
    } catch (error) {
        console.error("Error deleting user:", error);
        res.status(500).json({ message: "Internal server error" });
    }
};

export const loginUser = async (req: Request, res: Response) => {
    try {
        const { email, password } = req.body;

        if (!email || !password) {
```

```
            res.status(400).json({ message: "Email and password are required"
});
            return;
        }

        const user = await userRepo.findOneBy({ email });
        if (!user) {
            res.status(401).json({ message: "Invalid email or password" });
            return;
        }

        const isPasswordValid = await bcrypt.compare(password,
user.password);
        if (!isPasswordValid) {
            res.status(401).json({ message: "Invalid email or password" });
            return;
        }

        const token = jwt.sign(
            {
                id: user.id,
                role: user.role,
                email: user.email,
            },
            JWT_SECRET,
            { expiresIn: "1d" }
        );

        res.json({
            message: "Login successful",
            token,
            user: {
                id: user.id,
                username: user.username,
                email: user.email,
                role: user.role,
            },
        });
    } catch (error) {
        console.error("Login error:", error);
        res.status(500).json({ message: "Internal server error" });
    }
};
```

Взаимодействие с другими сервисами

- Зависимость от Vacancy Service: проверка существования компании при создании/обновлении пользователя-работодателя
- Используется другими сервисами: Resume Service и Vacancy Service проверяют существование userId

Vacancy Service

Назначение и ответственность

- Управление компаниями-работодателями
- Управление вакансиями
- Обработка откликов соискателей
- Управление мотивационными письмами
- Связывание вакансий с навыками

Пример одного из контроллеров (applicationController.cs)

```
import { Request, Response } from "express";
import axios from "axios";
import { AppDataSource } from "../data-source";
import { Application } from "../models/applicationModel";
import { Vacancy } from "../models/vacancyModel";

const appRepo = AppDataSource.getRepository(Application);
const vacancyRepo = AppDataSource.getRepository(Vacancy);

// Получить все отклики
export const getAllApplications = async (_: Request, res: Response) => {
    const items = await appRepo.find({ relations: ["vacancy"] });
    res.json(items);
};

// Получить отклик по ID + данные пользователя и резюме
export const getApplicationById = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        return res.status(400).json({ message: "Invalid ID format" });
    }

    const app = await appRepo.findOne({ where: { id }, relations: ["vacancy"]
});
    if (!app) {
        return res.status(404).json({ message: "Application not found" });
    }

    try {
        const [user, resume] = await Promise.all([
            axios.get(`http://user-service:3001/user-
service/user/${app.userId}`),
            axios.get(`http://resume-service:3002/resume-
service/resume/${app.resumeId}`)
        ]);

        return res.json({
            ...app,
            user: user.data,
            resume: resume.data
        });
    } catch {
        return res.json({
            ...app,
            user: null,
            resume: null
        });
    }
};

// Создать отклик
export const createApplication = async (req: Request, res: Response) => {
    const { userId, resumeId, vacancyId, status } = req.body;

    if (!userId || !resumeId || !vacancyId || !status) {
        return res.status(400).json({
            message: "Missing required fields: userId, resumeId, vacancyId,
or status"
        });
    }

    try {
```

```
        // Проверка user/resume через микросервисы
        const [userRes, resumeRes] = await Promise.all([
            axios.get(`http://user-service:3001/user-
service/user/${userId}`),
            axios.get(`http://resume-service:3002/resume-
service/resume/${resumeId}`)
        ]);

        const vacancy = await vacancyRepo.findOneBy({ id: vacancyId });
        if (!vacancy) {
            return res.status(404).json({ message: `Vacancy with id
${vacancyId} not found` });
        }

        const application = appRepo.create({
            userId,
            resumeId,
            vacancy,
            status
        });

        await appRepo.save(application);
        return res.status(201).json(application);
    } catch (err) {
        console.error("Error creating application:", err);
        return res.status(500).json({ message: "Internal server error" });
    }
};

// Обновить отклик
export const updateApplication = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        return res.status(400).json({ message: "Invalid ID format" });
    }

    const item = await appRepo.findOneBy({ id });
    if (!item) {
        return res.status(404).json({ message: "Application not found" });
    }

    appRepo.merge(item, req.body);
    await appRepo.save(item);
    res.json(item);
};

// Удалить отклик
export const deleteApplication = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        return res.status(400).json({ message: "Invalid ID format" });
    }

    const result = await appRepo.delete(id);
    if (result.affected === 0) {
        return res.status(404).json({ message: "Application not found" });
    }

    res.json({ deleted: result.affected });
};
```

Взаимодействие с другими сервисами

- Зависимость от User Service: проверка пользователей при создании

откликов и писем
- Зависимость от Resume Service: проверка резюме при создании откликов, агрегация данных
- Зависимость от Skill Service: проверка навыков при связывании с вакансиями
- Используется User Service: при создании пользователя-работодателя проверяется companyId

Skill Service

Назначение и ответственность
- Управление справочником навыков
- Предоставление единого каталога навыков для всей системы
- Валидация уникальности названий навыков

skillController.ts

```
import { Request, Response } from "express";
import { AppDataSource } from "../data-source";
import { Skill } from "../models/skillModel";

const skillRepo = AppDataSource.getRepository(Skill);

// Получить все навыки
export const getAllSkills = async (_: Request, res: Response) => {
    try {
        const skills = await skillRepo.find(); // Убраны relations
        res.json(skills);
    } catch (error) {
        res.status(500).json({ message: "Error retrieving skills", error });
    }
};

// Получить навык по ID
export const getSkillById = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        res.status(400).json({ message: "Invalid ID format" });
        return;
    }

    try {
        const skill = await skillRepo.findOneBy({ id }); // Без relations
        if (!skill) {
            res.status(404).json({ message: "Skill not found" });
            return;
        }
        res.json(skill);
    } catch (error) {
        res.status(500).json({ message: "Error retrieving skill", error });
    }
};

// Создать новый навык
export const createSkill = async (req: Request, res: Response) => {
    let { skill_name, description } = req.body;
```

```
    if (!skill_name || typeof skill_name !== "string" || skill_name.trim()
=== "") {
        return res.status(400).json({ message: "Missing or invalid required
field: skill_name" });
    }

    skill_name = skill_name.trim();
    description = typeof description === "string" ? description.trim() :
null;

    try {
        const existing = await skillRepo.findOneBy({ skill_name });
        if (existing) {
            return res.status(409).json({ message: "Skill with this name
already exists" });
        }

        const skill = skillRepo.create({ skill_name, description });
        await skillRepo.save(skill);

        return res.status(201).json(skill);
    } catch (error) {
        console.error("Error creating skill:", error);
        return res.status(500).json({ message: "Internal server error" });
    }
};

// Обновить навык
export const updateSkill = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        res.status(400).json({ message: "Invalid ID format" });
        return;
    }

    try {
        const skill = await skillRepo.findOneBy({ id });
        if (!skill) {
            res.status(404).json({ message: "Skill not found" });
            return;
        }

        // Обновляем только нужные поля с проверкой типа
        if (req.body.skill_name && typeof req.body.skill_name === "string" &&
req.body.skill_name.trim() !== "") {
            skill.skill_name = req.body.skill_name.trim();
        }
        if ("description" in req.body) {
            skill.description = typeof req.body.description === "string" ?
req.body.description.trim() : null;
        }

        await skillRepo.save(skill);
        res.json(skill);
    } catch (error) {
        res.status(500).json({ message: "Error updating skill", error });
    }
};

// Удалить навык
export const deleteSkill = async (req: Request, res: Response) => {
    const id = parseInt(req.params.id);
    if (isNaN(id)) {
        res.status(400).json({ message: "Invalid ID format" });
        return;
    }
```

```
    try {
        const result = await skillRepo.delete(id);
        if (result.affected === 0) {
            res.status(404).json({ message: "Skill not found" });
            return;
        }

        res.json({ deleted: result.affected });
    } catch (error) {
        res.status(500).json({ message: "Error deleting skill", error });
    }
};
```

Взаимодействие с другими сервисами

- Используется Resume Service: проверка существования навыков при создании ResumeSkills
- Используется Vacancy Service: проверка существования навыков при создании VacancySkills

Аутентификация

Авторизация реализуется при помощи JWT токена сроком действия 1 день. Токен передаётся в заголовке запроса и проверяется сервером на корректность.
Большинство маршрутов требуют токены

Выводы

В результате выполнения работы была реализована микросервисная система поиска подработок с получением нужных данных и реализованной аутентификацией