

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Работа с TypeORM

Выполнил:

Петухов Семён

Группа  
К3339

Проверил:  
Добряков Д. И.

Санкт-Петербург

2022 г.

## **Задача**

- Реализовать все модели данных, спроектированные в рамках ДЗ1
- Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
- Реализовать API-эндпоинт для получения пользователя по id/email

## **Ход работы**

Модели и эндпоинты реализовывались по схеме ниже

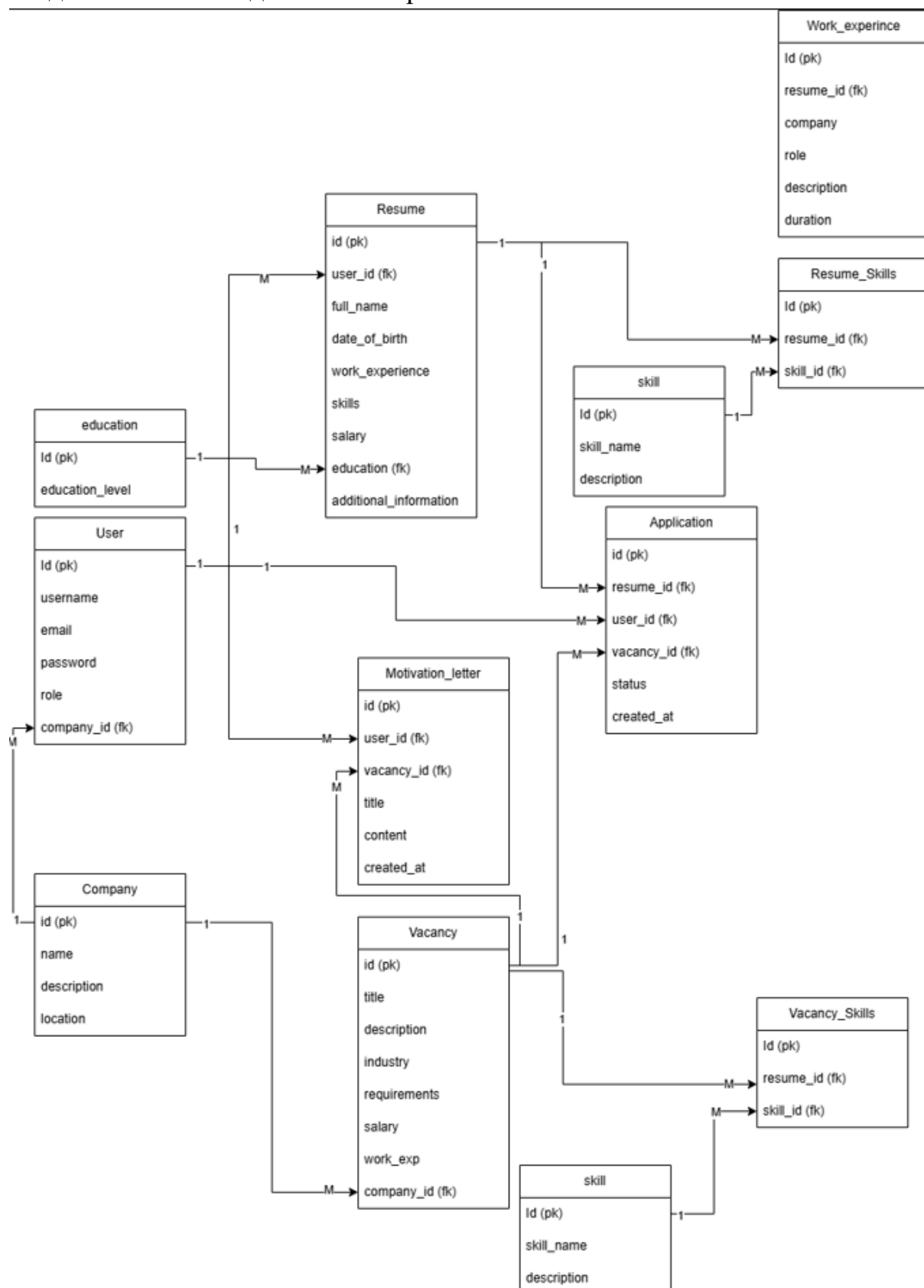


Рисунок 1 – модель базы данных

## 1. Описание моделей базы данных

Первым делом были описаны модели базы данных в файле «schema.prisma»

Ниже представлен пример модели для таблицы Resume

```
model Resume {
  id          Int      @id @default(autoincrement())
  user_id     Int      @unique
  full_name   String
  date_of_birth DateTime
  work_experience Int
  salary      Float
  education_id Int?
  additional_info String?

  user      User      @relation(fields: [user_id], references: [id])
  education Education? @relation(fields: [education_id], references: [id])
  skills    ResumeSkills[]
  work_experiences WorkExperience[]
  applications Application[]
}
```

Остальные модели были реализованы аналогично

## 2. Описание контроллеров для описания логики работы CRUD

Для каждой таблицы были реализованы контроллеры, содержащие логику обработки запроса

Возьмем как пример контроллер для user

```
import { Request, Response } from 'express';
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

// Создание нового пользователя
export const createUser = async (req: Request, res: Response) => {
  try {
    const { username, email, password, role, company_id } = req.body;
    const user = await prisma.user.create({
      data: {
        username,
        email,
        password, // в реальном проекте необходимо хэшировать
        role,
        company_id,
      },
    });
    res.status(201).json(user);
  } catch (error) {
    res.status(500).json({ error: 'Ошибка при создании пользователя' })
  }
}
```

```

});
  }
};

// Получение списка всех пользователей
export const getUsers = async (req: Request, res: Response) => {
  try {
    const users = await prisma.user.findMany();
    res.status(200).json(users);
  } catch (error) {
    res.status(500).json({ error: 'Ошибка при получении пользователей' });
  }
};

// Получение пользователя по id
export const getUserById = async (req: Request, res: Response) => {
  const { id } = req.params;
  try {
    const user = await prisma.user.findUnique({
      where: { id: parseInt(id) },
    });
    if (user) {
      res.status(200).json(user);
    } else {
      res.status(404).json({ error: 'Пользователь не найден' });
    }
  } catch (error) {
    res.status(500).json({ error: 'Ошибка при получении пользователя' });
  }
};

// Обновление пользователя
export const updateUser = async (req: Request, res: Response) => {
  const { id } = req.params;
  const { username, email, password, role, company_id } = req.body;
  try {
    const user = await prisma.user.update({
      where: { id: parseInt(id) },
      data: {
        username,
        email,
        password, // в реальном проекте необходимо хэшировать
        role,
        company_id,
      },
    });
    res.status(200).json(user);
  } catch (error) {
    res.status(500).json({ error: 'Ошибка при обновлении пользователя' });
  }
};

// Удаление пользователя
export const deleteUser = async (req: Request, res: Response) => {
  const { id } = req.params;
  try {
    const user = await prisma.user.delete({

```

```

        where: { id: parseInt(id) },
    });
    res.status(200).json({ message: 'Пользователь удалён' });
} catch (error) {
    res.status(500).json({ error: 'Ошибка при удалении пользователя'
});
}
};

// Получение пользователя по email
export const getUserByEmail = async (req: Request, res: Response) => {
    const { email } = req.query;

    if (!email || typeof email !== 'string') {
        return res.status(400).json({ error: 'Необходимо указать email' });
    }

    try {
        const user = await prisma.user.findUnique({
            where: { email },
        });

        if (user) {
            res.status(200).json(user);
        } else {
            res.status(404).json({ error: 'Пользователь не найден' });
        }
    } catch (error) {
        res.status(500).json({ error: 'Ошибка при получении пользователя по email' });
    }
};

```

Для каждого эндпоинта реализована индивидуальная функция с обработчиком запроса.

### 3. Реализация роутов

И наконец были реализованы роуты, которые привязывают операцию к конкретной ссылке

```

import express from 'express';
import { createUser, getUsers, getUserById, updateUser, deleteUser,
getUserByEmail } from '../controllers/userController';

const router = express.Router();

// Роуты для пользователей
router.post('/users', createUser); // создание пользователя
router.get('/users', getUsers); // получение всех пользователей
router.get('/users/:id', getUserById); // получение пользователя по ID
router.get('/users/email/search', getUserByEmail); // получение
пользователя по email

```

```
router.put('/users/:id', updateUser); // обновление пользователя
router.delete('/users/:id', deleteUser); // удаление пользователя

export default router;
```

## Вывод

По результатам работы были разобраны методы реализации базы данных и разработки CRUD-операций к ней, были реализованы модели и эндпоинты по схеме из домашней работы 1.