

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
Факультет инфокоммуникационных технологий

ОТЧЕТ

по Лабораторной работе № 1

Специальность:

09.03.03 Мобильные и сетевые технологии

Проверил:

Добряков Д. И. _____

Дата: «__» _____ 202__ г.

Оценка _____

Выполнил:

студенты группы К33401

Ковалев В. М.

Санкт-Петербург

2023

ЦЕЛЬ РАБОТЫ

Написать свой boilerplate на express + sequelize / TypeORM + typescript.

Должно быть явное разделение на:

- модели
- контроллеры
- роуты
- сервисы для работы с моделями (реализуем паттерн “репозиторий”)

ВЫПОЛНЕНИЕ

В своей работе я использовал ORM Prisma.

1. Инициализировал проект, установил nodemon автоматического перезапуска проекта при изменении, составил Makefile.

```
{
  "name": "rest-express",
  "version": "1.0.0",
  "license": "MIT",
  "scripts": {
    "start": "node ./dist/src/index.js",
    "start:dev": "node --inspect-brk=0.0.0.0 ./dist/src/index.js",
    "build": "tsc",
    "dev": "nodemon -L -e ts --exec \"yarn run build && yarn start\""
  },
  "dependencies": {
    "@prisma/client": "4.13.0",
    "express": "4.18.2"
  },
  "devDependencies": {
    "@types/express": "4.17.17",
    "@types/node": "18.16.0",
    "nodemon": "^2.0.22",
    "prisma": "4.13.0",
    "ts-node": "10.9.1",
    "typescript": "4.9.5"
  },
  "prisma": {
    "seed": "ts-node prisma/seed.ts"
  }
}
```

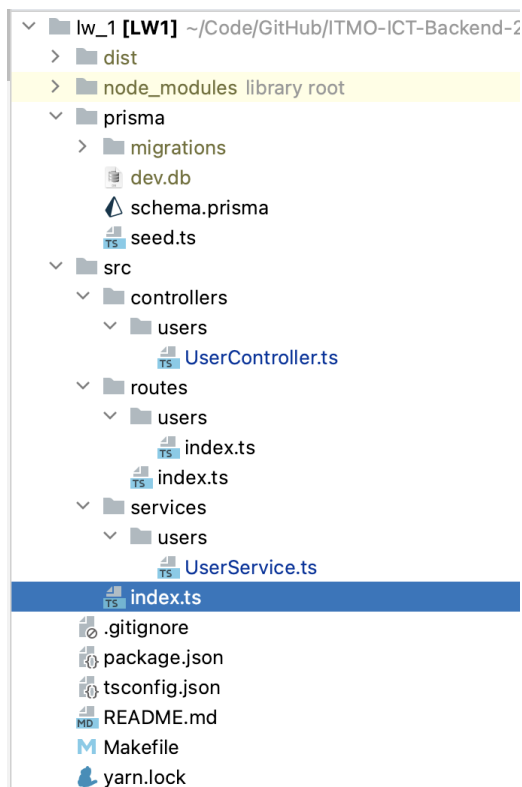
```
.PHONE: dev
dev:
  yarn prisma migrate dev
  yarn run dev

.PHONE: build
build:
  yarn run build

.PHONE: start
start:
  yarn prisma migrate dev
  yarn run start

.PHONE: init
init:
  yarn init
```

2. Задал структуру проекта. Модели хранятся в schema.prisma.



3. Описал модель User

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = "file:./dev.db"
}

model User {
  id      Int      @id @default(autoincrement())
  email   String   @unique
  name    String?
}
```

4. Класс UserService

```
import {PrismaClient, User} from "@prisma/client";

const prisma = new PrismaClient()

interface UserModel extends Omit<User, "id">{}

class UserService {
  createUser(user: UserModel) {
    return prisma.user.create({data: user})
  }
  getById(id: number) {
    return prisma.user.findUniqueOrThrow( args: {where: {id}})
  }
  getByEmail(email: string){
    return prisma.user.findUniqueOrThrow( args: {where:{email}})
  }
  getUsers() {
    return prisma.user.findMany()
  }
}

export default UserService;
```

5. Класс UserController

```
class UserController{
    private userService: UserService

    constructor() {
        this.userService = new UserService()
    }

    get = async (request: Request, response: Response) => {
        try {
            const user = await this.userService.getById(Number(request.params.id))
            response.send(user)
        } catch (e:any) {
            response.status( code: 404).send( body: { "error": e.name })
        }
    }

    getAll = async (request: Request, response: Response) => {
        try {
            const {email} = request.query
            if(email){
                const {email} = request.query
                const user = await this.userService.getByEmail(String(email))
                response.send( body: [user])
            }
            const users = await this.userService.getUsers()
            response.send(users)
        } catch (e: any) {
            response.status( code: 404).send( body: {"error": e.message})
        }
    }

    post = async (request: Request, response: Response) => {
        try {
            const {email, name} = request.body
            const user = await this.userService.createUser( user: {email, name})
            response.send(user)
        } catch (e:any) {
            response.status( code: 404).send( body: {"error": e.message})
        }
    }
}
```

6. Роутинг

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

const controller: UserController = new UserController()

router.route( prefix:('/:id')
  .get(controller.get)

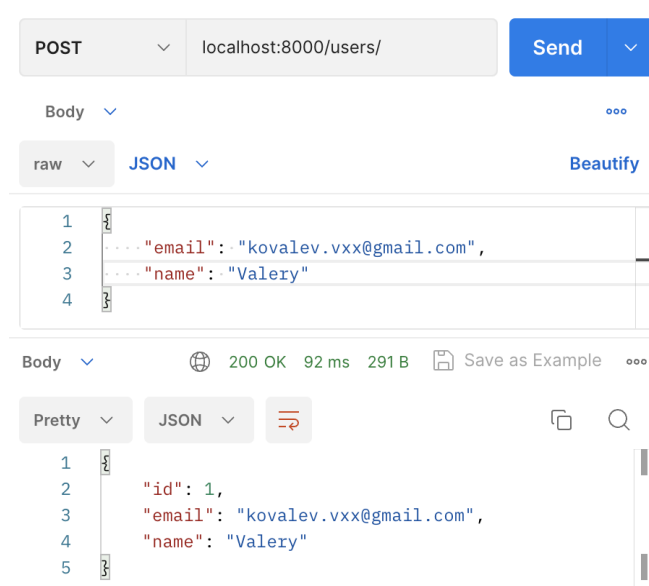
router.route( prefix: '/')
  .post(controller.post)

router.route( prefix: '/')
  .get(controller.getAll)

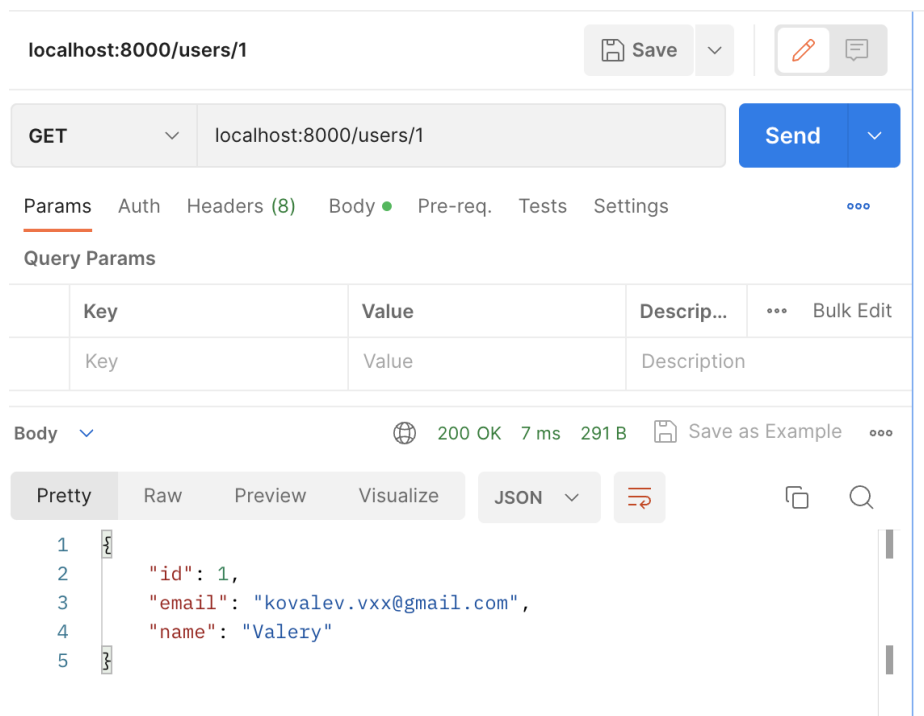
export default router;
```

ТЕСТИРОВАНИЕ

1. Создание пользователя



2. Получение пользователя по ID



localhost:8000/users/1

GET localhost:8000/users/1

Params Auth Headers (8) Body ● Pre-req. Tests Settings

Query Params

Key	Value	Descrip...
Key	Value	Description

Body

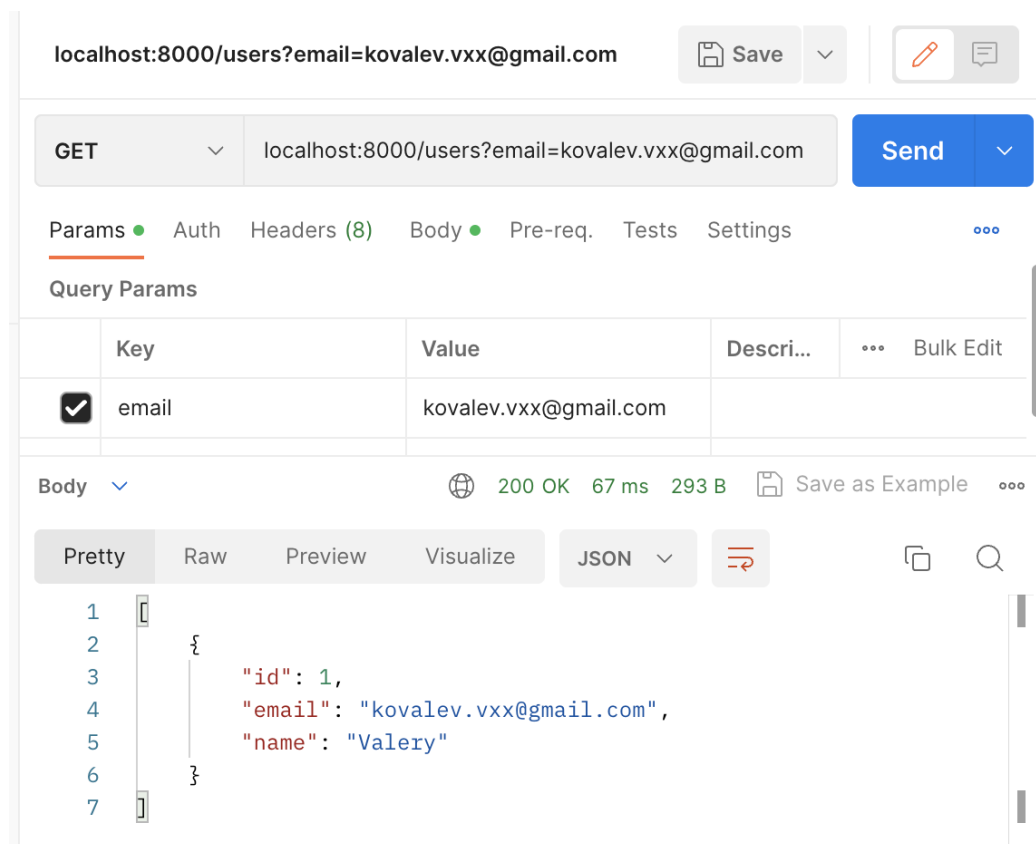
200 OK 7 ms 291 B

Save as Example

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": 1,  
3   "email": "kovalev.vxx@gmail.com",  
4   "name": "Valery"  
5 }
```

3. Поиск пользователя по email



localhost:8000/users?email=kovalev.vxx@gmail.com

GET localhost:8000/users?email=kovalev.vxx@gmail.com

Params ● Auth Headers (8) Body ● Pre-req. Tests Settings

Query Params

Key	Value	Descrip...
<input checked="" type="checkbox"/> email	kovalev.vxx@gmail.com	

Body

200 OK 67 ms 293 B

Save as Example

Pretty Raw Preview Visualize JSON

```
1 [  
2   {  
3     "id": 1,  
4     "email": "kovalev.vxx@gmail.com",  
5     "name": "Valery"  
6   }  
7 ]
```

4. Вывод всех пользователей

The screenshot shows a REST client interface with the following components:

- URL Bar:** `localhost:8000/users/` with a **Save** button and a dropdown arrow.
- Method and URL:** **GET** method selected, URL `localhost:8000/users/`, and a blue **Send** button.
- Tabs:** Params, Auth, Headers (8), Body (selected), Pre-req., Tests, Settings.
- Query Params Table:**

Key	Value	Description
Key	Value	Description
- Body Section:**
 - Status: 200 OK, 5 ms, 293 B.
 - Buttons: Save as Example, and a tooltip that says "Save this request to create examples."
 - Viewers: Pretty (selected), Raw, Preview, Visualize, JSON.
 - JSON Response:**

```
1 {
2   {
3     "id": 1,
4     "email": "kovalev.vxx@gmail.com",
5     "name": "Valery"
6   }
7 }
```

ВЫВОД

В результате выполнения лабораторной работы я получил boilerplate, который смогу использовать как основу для будущих приложений.