**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэкенд разработка

# Отчет

## Лабораторная работа №2

Выполнил:

Фамилия Имя
Тишалович Леонид

Группа
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

**Задача:** необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

**Ход выполнения:**

В ходе выполнения лабораторной работы, проект приобрёл следующую структуру:

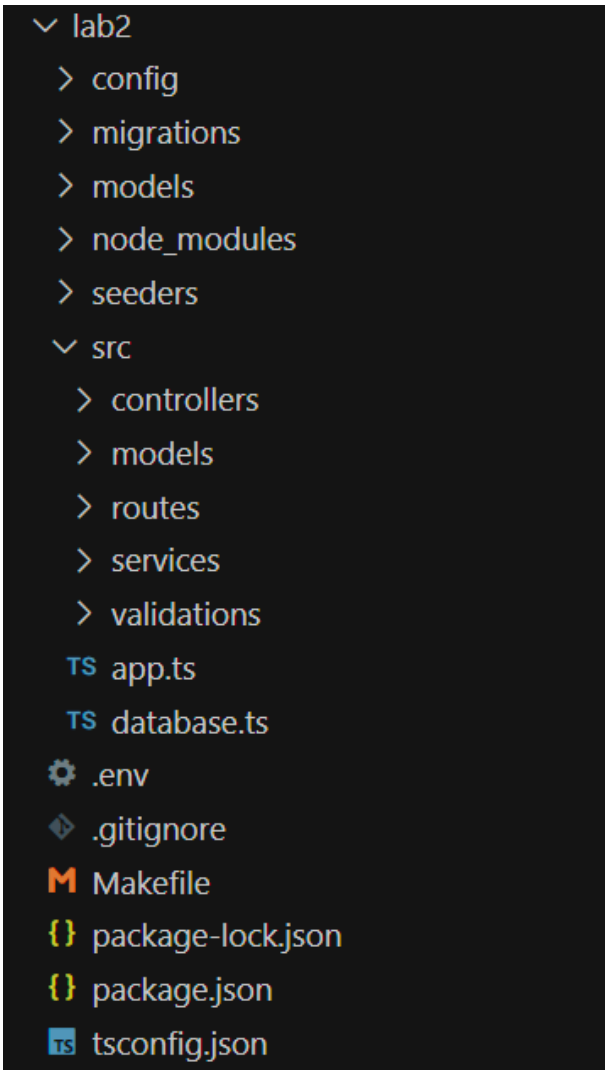


Рисунок 1 - структура проекта lab2

Был удален слой repository и возможность создавать миграции для обновления базы данных. Также, была добавлена папка validations, которая добавляет валидации для регистрации и авторизации. Сам флоу остался таким же, что и в первой лабораторной работе. В app.ts запускается проект и вызываются руты из папки routes, каждый рут отправляет определенный

запрос (GET, POST, PUT, DELETE) и вызывает функцию из контроллера и контроллер вызывает сервис, где и происходит манипуляция с бд через представителей классов моделей.

В качестве примера ниже приведена реализация регистрации и авторизации в приложении.

```
app.use("/api", apiRouter);
app.use("/auth", authRouter);

const start = async (): Promise<void> => {
  try {
    await sequelize.authenticate();
    await sequelize.sync();
  } catch (error) {
    console.log(error);
  }
  app.listen(PORT, () => {
    console.log(`Server is running on ${PORT} port`);
  });
};
```

Рисунок 2 - часть app.ts с использованием рута для авторизации

```
import express from "express";
import { registerValidation } from "../validations/auth";
import UserController from "../controllers/UserControllers";

const router = express.Router();

router.post("/login", UserController.login);
router.post("/register", registerValidation, UserController.register);

export default router;
```

Рисунок 3 - authRoutes.ts

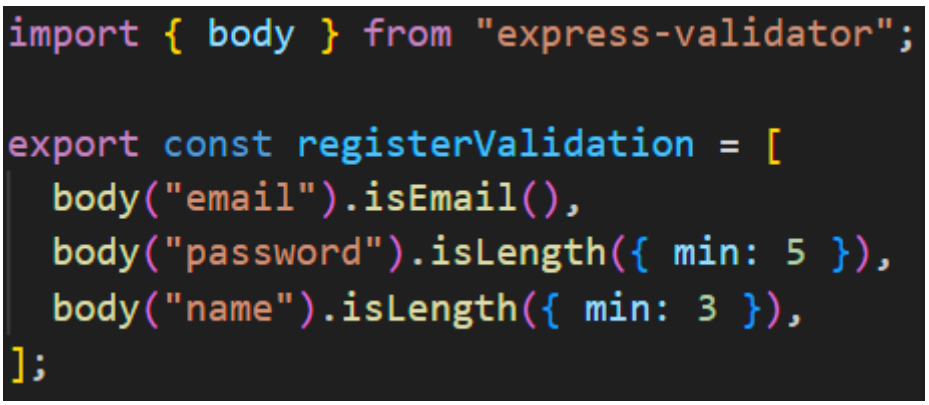


Рисунок 4 - валидации для регистрации

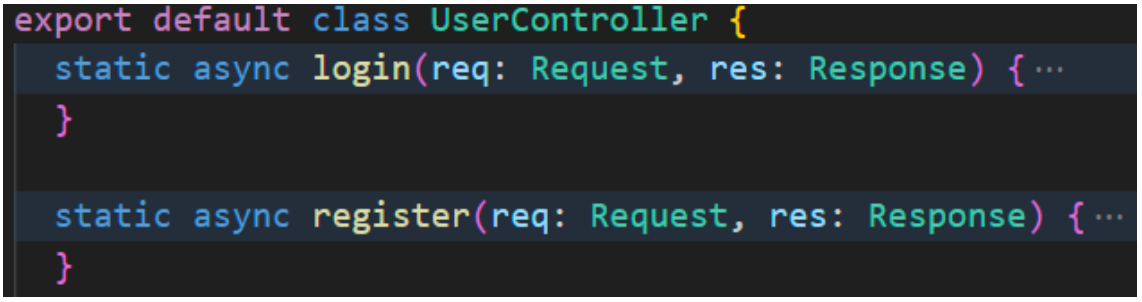


Рисунок 5 - часть UserController.ts

```ts
import { Sequelize } from "sequelize-typescript";
import User from "../models/User";
import { Repository } from "sequelize-typescript";

export default class UserService {
  private userRepository: Repository<User>;

  constructor(sequelizeInstance: Sequelize) {
    this.userRepository = sequelizeInstance.getRepository(User);
  }

  async getAllUsers() {
    return this.userRepository.findAll();
  }

  async getUserById(id: number) {
    return this.userRepository.findByPk(id);
  }

  async getUserByEmail(email: string) {
    return this.userRepository.findOne({
      where: { email: email },
    });
  }

  async createUser(userData: any) {
    return this.userRepository.create(userData);
  }

  async updateUser(id: number, userData: any) {
    const user = await this.userRepository.findByPk(id);
    if (!user) {
      throw new Error("User not found");
    }
    await user.update(userData);
    return user;
  }

  async deleteUser(id: number) {
    const user = await this.userRepository.findByPk(id);
    if (!user) {
      throw new Error("User not found");
    }
    await user.destroy();
  }
}
```

Рисунок 6 - UserService.ts

**Вывод:** в ходе данной лабораторной работы были получены навыки по созданию своего RESTful API.