САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1

Выполнил:
Байков Иван
К33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.
Должно быть явное разделение на:

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

Ход работы

Создаем entry point

```
import "dotenv/config";
import "reflect-metadata";

import cors from "cors";
import express from "express";
import { PostgresDataSource } from "./database/db.js";
import router from "./routes/index.js";

async function main() {
  try {
    console.log("Server started");
    await PostgresDataSource.initialize();
    const app = express();
    const port = process.env.PORT || 3000;

    app.use(cors());
    app.use(express.json());

    app.use(router);

    app.listen(port, () => {
      console.log(`Server is running at http://localhost:${port}`);
    });
  } catch (error) {
    console.log(error);
  }
}

main().catch((err) => console.error(err));
```

```ts
import { fileURLToPath } from "node:url";
import path, { dirname } from "node:path";
import { DataSource } from "typeorm";
import "dotenv/config";

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

export const PostgresDataSource = new DataSource({
  type: "postgres",
  host: process.env.PG_HOST,
  port: +process.env.PG_PORT,
  username: process.env.PG_USER,
  database: process.env.PG_DB,
  password: process.env.PG_PASSWORD,
  synchronize: true, //todo
  logging: false,
  logNotifications: false,
  applicationName: "itmo_backend",
  entities: [path.join(__dirname, "entities", "*.{ts,js}")],
  migrations: [path.join(__dirname, "migrations", "*.{ts,js}")],
});
```

Создаем модель

```ts
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ type: "varchar", unique: true })
  email: string;

  @Column({ type: "varchar", unique: true })
  password: string;
}
```

Роутер
index.ts

```
import { Router } from "express";

import usersRouter from "./users.js";
const apiVersion = "/api/v1";

const router = Router();

router.use(`${apiVersion}/users`, usersRouter);

router.use("*", (_, res) => {
  res.status(404).json("Endpoint not found");
});

export default router;
```

users.ts

```
import { Router } from "express";
import usersController from "../controllers/users.js";

const router = Router();

router.get("/:idOrEmail", usersController.get);
router.post("/:id", usersController.create);
router.put("/:id", usersController.update);
router.delete("/:id", usersController.delete);

export default router;
```

```ts
import { Request, Response } from "express";
import usersService from "../services/users.js";

const usersController = {
  get: async (req: Request, res: Response) => {
    try {
      const idOrEmail = req.params.idOrEmail;
      const user = await usersService.get(idOrEmail);
      if (user) {
        res.status(200).json(user);
      } else {
        res.status(404).json({ message: "User not found" });
      }
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },

  create: async (req: Request, res: Response) => {
    try {
      const { email, password } = req.body;
      if (!email || !password) {
        return res.status(400).json({ message: "Email and password are required" });
      }
      const user = await usersService.create({ email, password });
      res.status(201).json(user);
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },
```

```
update: async (req: Request, res: Response) => {
  try {
    const id = parseInt(req.params.id);
    const { email, password } = req.body;
    const updatedUser = await usersService.update(id, { email, password });
    if (updatedUser) {
      res.status(200).json(updatedUser);
    } else {
      res.status(404).json({ message: "User not found" });
    }
  } catch (error) {
    res.status(500).json({ message: "Internal server error" });
  }
},

delete: async (req: Request, res: Response) => {
  try {
    const id = parseInt(req.params.id);
    const deleted = await usersService.delete(id);
    if (deleted) {
      res.status(204).send();
    } else {
      res.status(404).json({ message: "User not found" });
    }
  } catch (error) {
    res.status(500).json({ message: "Internal server error" });
  }
},
```

```ts
import { userRepository } from "../database/repositories/User.js";

const usersService = {
  get: async (idOrEmail: string) => {
    try {
      const id = parseInt(idOrEmail, 10);

      if (!isNaN(id)) {
        return await userRepository.findOneBy({ id });
      } else {
        return await userRepository.findOneBy({ email: idOrEmail });
      }
    } catch (error) {
      throw new Error("Error fetching user");
    }
  },

  create: async (userData: { email: string; password: string }) => {
    try {
      const user = userRepository.create(userData);
      return await userRepository.save(user);
    } catch (error) {
      throw new Error("Error creating user");
    }
  },
```

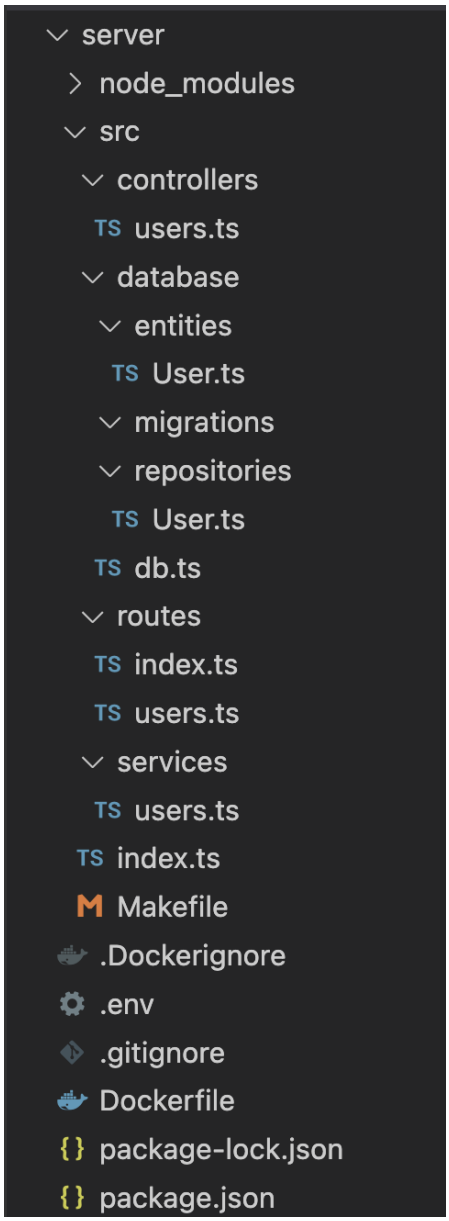
```ts
  update: async (id: number, updateData: { email?: string; password?: string }) => {
    try {
      const user = await userRepository.findOneBy({ id });
      if (!user) return null;

      user.email = updateData.email || user.email;
      user.password = updateData.password || user.password;

      return await userRepository.save(user);
    } catch (error) {
      throw new Error("Error updating user");
    }
  },

  delete: async (id: number) => {
    try {
      const result = await userRepository.delete(id);
      return result.affected > 0;
    } catch (error) {
      throw new Error("Error deleting user");
    }
  },
};
```

Структура проекта



Вывод

В ходе работы был реализован boilerplate на express + typeorm