САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа

Выполнила:

Олейникова Полина

Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Нужно написать свой boilerplate на express + sequelize + typescript.

Должно быть явное разделение на:

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

Ход работы

Написали файл для запуска сервера.

```ts
import express from "express"
import { createServer, Server } from "http"
import sequelize from "../providers/db"
import { Sequelize } from 'sequelize-typescript'
import bodyParser from "body-parser"
import cors from "cors";
import routes from "../routes/v1/index"

require('dotenv').config()

export default class App {
    public port: number
    public host: string

    private app: express.Application
    private server: Server
    private sequelize: Sequelize

    constructor(port = 8000, host = "localhost") {
        this.port = Number(process.env.PORT) || port
        this.host = process.env.HOST || host

        this.app = this.createApp()
        this.server = this.createServer()
        this.sequelize = sequelize
    }

    private createApp(): express.Application {
        const app = express()
        app.use(cors())
        app.use(bodyParser.json())
        app.use('/v1', routes)
        return app
    }
```

Создали и подключились к БД.

```js
require('dotenv').config();

module.exports = {
  "development": {
    "username": process.env.DB_USERNAME,
    "password": process.env.DB_PASSWORD,
    "database": process.env.DB_NAME,
    "host": process.env.HOST,
    "dialect": "postgres"
  },
  "test": {
    "username": process.env.DB_USERNAME,
    "password": process.env.DB_PASSWORD,
    "database": process.env.DB_NAME,
    "host": process.env.HOST,
    "dialect": "postgres"
  },
  "production": {
    "username": process.env.DB_USERNAME,
    "password": process.env.DB_PASSWORD,
    "database": process.env.DB_NAME,
    "host": process.env.HOST,
    "dialect": "postgres"
  }
}
```

```ts
import { Sequelize } from 'sequelize-typescript'
import User from '../models/users/User'
import { RefreshToken } from "../models/auth/RefreshToken"

require('dotenv').config();

const sequelize = new Sequelize({
    "username": process.env.DB_USERNAME,
    "password": process.env.DB_PASSWORD,
    "database": process.env.DB_NAME,
    "host": process.env.HOST,
    "dialect": "postgres"
})
const models = [User, RefreshToken]
sequelize.addModels(models)

// sequelize
//     .sync()
//     .then(() => {
//         console.log('synced models')
//     })
//     .catch((e) => console.log(e));

async function testConnection() {
    try {
        await sequelize.authenticate();
        console.log('Connection has been established successfully.');
    } catch (error) {
        console.error('Unable to connect to the database:', error);
    }
}

testConnection()

export default sequelize
```

Обработали пользователя: создали модель, указали маршруты, добавили контроллер и сервис.

User.ts

```ts
import hashPassword from '../../utils/hashPassword'
import { Table, Column, Model, Unique, AllowNull, BeforeCreate, BeforeUpdate, PrimaryKey, DataType } from 'sequelize-typescript'
import { Optional } from "sequelize";

export type UserAttributes = {
    id: string,
    name: string,
    email: string;
    password: string;
};

export type UserCreationAttributes = Optional<UserAttributes, 'id'>;


@Table
export class User extends Model<UserAttributes, UserCreationAttributes> {
    @PrimaryKey
    @Column({
        type: DataType.UUID,
        defaultValue: DataType.UUIDV4,
    })
    id: string;

    @Column
    name: string;

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
    password: string
```

UserRoutes.ts

```ts
import express from "express"
import UserController from "../../../controllers/users/UserController"


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

const controller: UserController = new UserController()

router.get('/me', controller.me)
router.route('/')
    .get(controller.get)
    .patch(controller.update)
    .delete(controller.delete)
router.route('/password') .patch(controller.changePassword)

export default router
```

UserController.ts

```ts
import UserService from '../../services/users/UserService'

export default class UserController {
    private userService: UserService

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

    get = async (request: any, response: any) => {
        try {
            const user = await this.userService.getAll()
            response.status(201).send(user)
        } catch (error: any) {
            response.status(404).send({ "error": error.toString() })
        }
    }

    update = async (request: any, response: any) => {
        try {
            const user = await this.userService.update(request.user, request.body)
            response.status(201).send(user)
        } catch (error: any) {
            response.status(400).send({ "error": error.toString() })
        }
    }
```

UserService.ts

```ts
import User, { UserAttributes } from '../../models/users/User'
import checkPassword from '../../utils/checkPassword'
import hashPassword from '../../utils/hashPassword'

class UserService {
    async getById(id: number): Promise<User> {
        try {
            const user = await User.findByPk(id)
            if (!user) {
                throw new Error('Not Found');
            }
            return user
        }
        catch (error) {
            throw error;
        }
    }

    async getAll(): Promise<User[]> {
        try {
            const user = await User.findAll()
            return user
        } catch (error) {
            throw error;
        }
    }
```

Также написали функции для хеширования пароля:

checkPassword.ts

```ts
import bcrypt from "bcrypt"
import { UserAttributes } from "../models/users/User"

export default (user: UserAttributes, password: string) => {
    return bcrypt.compareSync(password, user.password)
}
```

hashPassword.ts

```ts
import bcrypt from 'bcrypt'

export default (password: string): string => bcrypt.hashSync(password, bcrypt.genSaltSync(8))
```

Сделали авторизацию и регистрацию, написали middleware

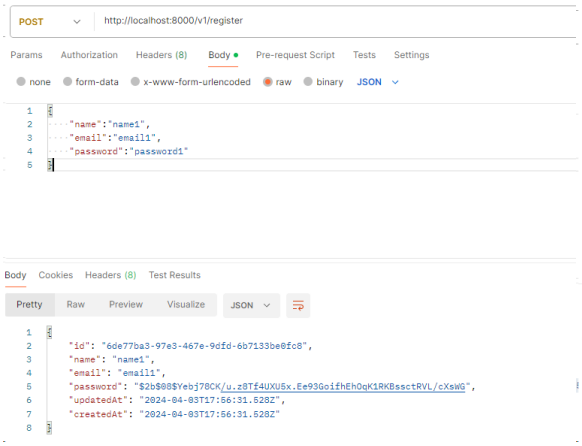
AuthService.ts

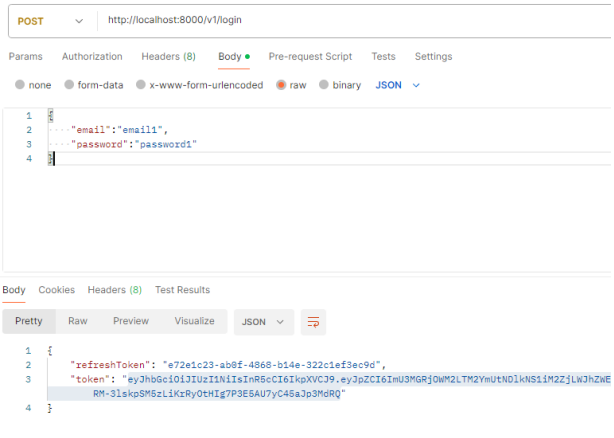
```ts
async login(email: string, password: string): Promise<{ token: string, refreshToken: string }> {
    try {
        const user = await User.findOne({ where: { email } })
        if (!user || !checkPassword(user, password)) {
            throw new Error('Email or password is not correct');
        }
        const refreshTokenService = new RefreshTokenService(user)
        const refreshToken = await refreshTokenService.generateRefreshToken()
        const token = jwt.sign({ id: user.id }, SECRET_KEY, { expiresIn: '2 days' });
        return { refreshToken: refreshToken, token: token };
    } catch (error) {
        throw error;
    }
}

async refreshToken(refreshToken: string): Promise<{ token: string, refreshToken: string }> {
    const refreshTokenService = new RefreshTokenService()
    try {
        const { userId, isExpired } = await refreshTokenService.isRefreshTokenExpired(refreshToken)
        if (!isExpired && userId) {
            const user = await User.findByPk(userId)
            const refreshTokenService = new RefreshTokenService(user)
            const refreshToken = await refreshTokenService.generateRefreshToken()
            const accessToken = jwt.sign({ id: userId }, SECRET_KEY, { expiresIn: '2 days' });
            return { refreshToken: refreshToken, token: accessToken };
        } else {
            throw new Error('Invalid credentials')
        }
    } catch (error) {
        throw error;
    }
}
}
```

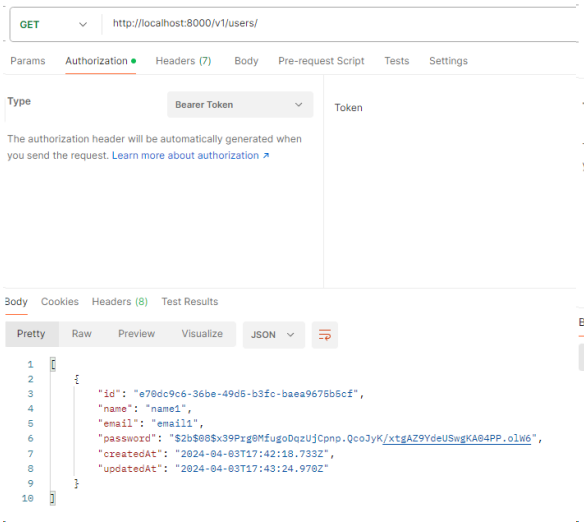
auth.ts

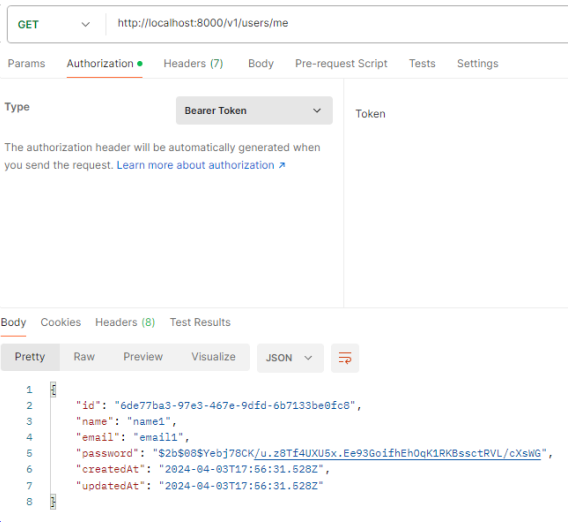
```ts
import jwt, { Secret, JwtPayload } from 'jsonwebtoken';
import { Request, Response, NextFunction } from 'express';
import UserService from '../services/users/UserService';
import { UserAttributes } from '../models/users/User';

require('dotenv').config();

if (!process.env.SECRET_KEY) {
  throw new Error('Missing SECRET_KEY in environment variables');
}

export const SECRET_KEY: Secret = process.env.SECRET_KEY;
export interface CustomRequest extends Request {
  user: UserAttributes;
}

const userService = new UserService();

export const auth = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');

    if (!token) {
      throw new Error('Missing token');
    }

    const decoded = jwt.verify(token, SECRET_KEY) as JwtPayload;
    const userData = await userService.getById(decoded.id);

    if (!userData) {
      throw new Error('User not found');
    }

    (req as CustomRequest).user = userData;
    next();
```

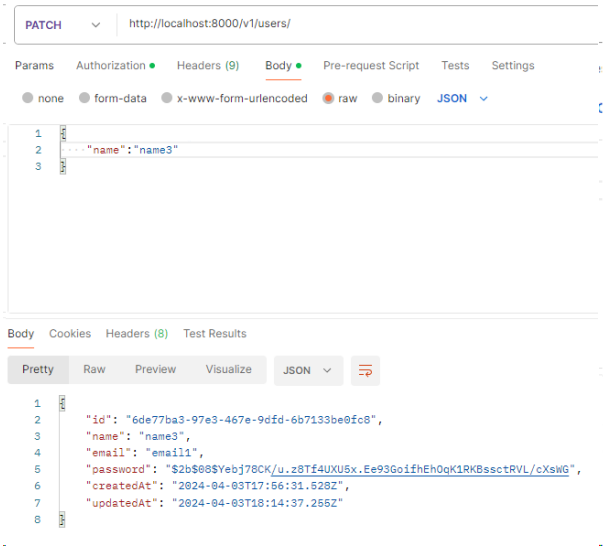
В итоге получились работающие запросы:

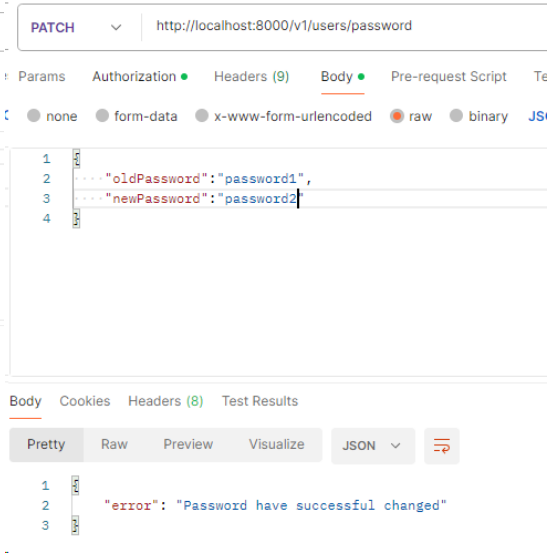




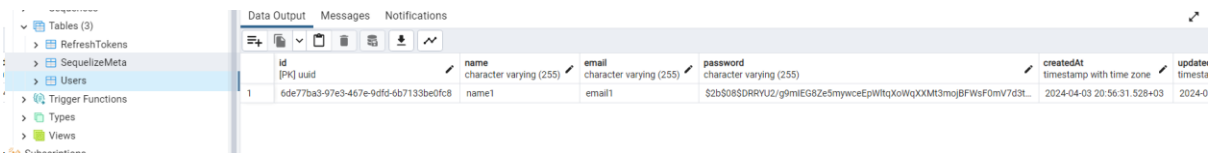








Вид в БД.



Вывод

В ходе данной работы был написан свой boilerplate на express + sequelize + typescript.