САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа: Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

Выполнил:

Хайрнасов А.К.

K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

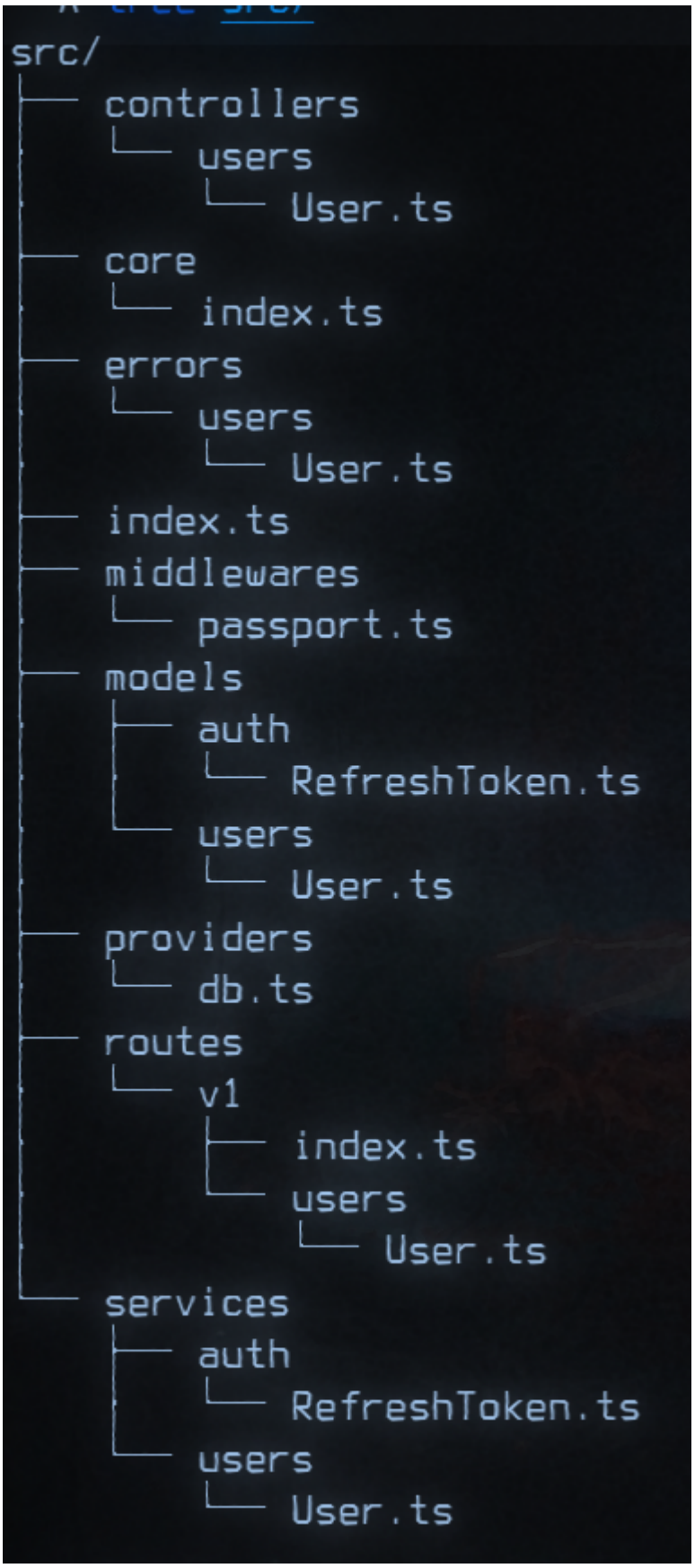
Задача
Нужно написать свой boilerplate на express + sequelize / TypeORM + typescript.

Должно быть явное разделение на:

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

Ход работы

Структура сурцов


src/
controllers
users
User.ts
core
index.ts
errors
users
User.ts
index.ts
middlewares
passport.ts
models
auth
RefreshToken.ts
users
User.ts
providers
db.ts
routes
v1
index.ts
users
User.ts
services
auth
RefreshToken.ts
users
User.ts

Модельки

File: src/models/auth/RefreshToken.ts

```ts
import { Table, Column, Model, Unique, AllowNull, ForeignKey } from 'sequelize-typescript'
import User from '../users/User'

@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull(false)
    @Column
    token: string;

    @ForeignKey(() => User)
    @Column
    userId: number;
}

export default RefreshToken;
```

File: src/models/users/User.ts

```ts
import {Table, Column, Model, Unique, AllowNull, BeforeCreate, BeforeUpdate} from 'sequelize-typescript'
import bcrypt from "bcrypt";

@Table
class User extends Model {
    @Unique
    @Column
    username: string;

    @Unique
    @Column
    email: string;

    @AllowNull(false)
    @Column
    password: string;

    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const {password} = instance;

        if (instance.changed('password')) {
            instance.password = bcrypt.hashSync(password, bcrypt.genSaltSync(128));
        }
    }
}

export default User;
```

Сервисы

File: src/models/users/User.ts

```ts
import {Table, Column, Model, Unique, AllowNull, BeforeCreate, BeforeUpdate} from 'sequelize-typescript'
import bcrypt from "bcrypt";

@Table
class User extends Model {
    @Unique
    @Column
    username: string;

    @Unique
    @Column
    email: string;

    @AllowNull(false)
    @Column
    password: string;

    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const {password} = instance;

        if (instance.changed('password')) {
            instance.password = bcrypt.hashSync(password, bcrypt.genSaltSync(128));
        }
    }
}

export default User;
```

```ts

    isRefreshTokenExpired = async (token: string): Promise<{ userId: number | null, isExpired: boolean }> => {
        const refreshToken = await RefreshToken.findOne({where: {token}});

        if (refreshToken) {
            const tokenData = refreshToken.toJSON();
            const timeDelta = new Date().getTime() - tokenData.createdAt.getTime();

            if (timeDelta > 0 && timeDelta < parseInt(process.env.REFRESH_TOKEN_LIFETIME!)) {
                return {userId: tokenData.userId, isExpired: false};
            }

            return {userId: null, isExpired: true};
        }

        return {userId: null, isExpired: true};
    }
}

export default RefreshTokenService;
```

File: src/services/users/User.ts

```ts
import User from '../../models/users/User'
import UserError from '../../errors/users/User'
import bcrypt from "bcrypt";

class UserService {
    async getById(id: number): Promise<User> {
        const user = await User.findByPk(id);

        if (user) return user.toJSON();

        throw new UserError(`User not fount UwU`);
    }

    async create(userData: Partial<User>): Promise<User> {
        try {
            const user = await User.create(userData);

            return user.toJSON();
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message);

            throw new UserError(errors);
        }
    }

    async checkPassword(email: string, password: string): Promise<any> {
        const user = await User.findOne({where: {email}});

        if (user) return {user: user.toJSON(), checkPassword: bcrypt.compareSync(password, user.password)};
```

Контроллеры

File: src/routes/v1/users/User.ts

```ts
import express from "express"
import UserController from "../../../controllers/users/User"
import passport from "../../../middlewares/passport"

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

const controller: UserController = new UserController();

router.route('/create')
    .post(controller.post);

router.route('/login')
    .post(controller.auth);

router.route('/auth')
    .get(passport.authenticate('jwt', {session: false}), controller.me);

router.route('/refresh')
    .post(controller.refreshToken);

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

router.route('/')
    .get(controller.getAllUsers);

export default router;
```

```
        response.send(request.user);
    };

    auth = async (request: any, response: any) => {
        const {body} = request;
        const {email, password} = body;

        try {
            const {user, checkPassword} = await this.userService.checkPassword(email, password)

            if (checkPassword) {
                const payload = {id: user.id};
                console.log('payload is', payload);
                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey);
                const refreshTokenService = new RefreshTokenService(user);
                const refreshToken = await refreshTokenService.generateRefreshToken();
                response.send({accessToken, refreshToken});
            } else {
                throw new Error('Login or password is incorrect!');
            }
        } catch (e: any) {
            response.status(401).send({"error": e.message});
        }
    };

    refreshToken = async (request: any, response: any) => {
        const {body} = request;
        const {refreshToken} = body;
        const refreshTokenService = new RefreshTokenService();

        try {
            const {userId, isExpired} = await refreshTokenService
                .isRefreshTokenExpired(refreshToken);
```

Роуты

File: src/routes/v1/users/User.ts

```ts
import express from "express"
import UserController from "../../../controllers/users/User"
import passport from "../../../middlewares/passport"

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

const controller: UserController = new UserController();

router.route('/create')
    .post(controller.post);

router.route('/login')
    .post(controller.auth);

router.route('/auth')
    .get(passport.authenticate('jwt', {session: false}), controller.me);

router.route('/refresh')
    .post(controller.refreshToken);

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

router.route('/')
    .get(controller.getAllUsers);

export default router;
```

Вывод

В ходе работы был создан шаблон приложения с использованием Express.js, Sequelize/TypeORM и TypeScript. Успешно проведено разделение на модели, контроллеры, роуты и сервисы. Реализован паттерн "репозиторий" для работы с моделями. Результат работы - гибкая, масштабируемая система, готовая к дальнейшему использованию и разработке.