САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1: Boilerplate на express + sequelize / TypeORM + typescript

Выполнил:

Конев Антон

Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 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

Models

Модель пользователя

```
7+ usages
@Table
class User extends Model {
    @Unique
    @Column
    username: string;

    @Unique
    @Column
    email: string;

    @AllowNull( allowNull: false)
    @Column
    password: string;

    no usages
    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) : void  {
        const {password : string } = instance;

        if (instance.changed( key: 'password')) {
            instance.password = hashPassword(password);
        }
    }
}
```

Модель RefreshToken

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

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

Controllers

```
1 usage
getAllUsers = async (request: any, response: any) : Promise<void>  => {
    try {
        const users : User[]  = await this.userService.getAllUsers();

        response.send(users);
    } catch (error: any) {
        response.status(404).send({"error": error.message});
    }
};
```

```typescript
class UserController {
    private userService: UserService;

    1 usage
    constructor() {
        this.userService = new UserService();
    }

    1 usage
    get = async (request: any, response: any) : Promise<void>  => {
        try {
            const user: User | UserError = await this.userService.getById(
                Number(request.params.id)
            );

            response.send(user);
        } catch (error: any) {
            response.status(404).send({"error": error.message});
        }
    };

    1 usage
    post = async (request: any, response: any) : Promise<void>  => {
        const {body} = request;

        try {
            const user: User | UserError = await this.userService.create(body);

            response.status(201).send(user);
        } catch (error: any) {
            response.status(400).send({"error": error.message});
        }
    };

    1 usage
    me = async (request: any, response: any) : Promise<void>  => {
        response.send(request.user);
    };
```

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

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

        if (checkPassword) {
            const payload : {id: any}  = {id: user.id};
            console.log('payload is', payload);
            const accessToken : string  = jwt.sign(payload, jwtOptions.secretOrKey);
            const refreshTokenService : RefreshTokenService  = new RefreshTokenService(user);
            const refreshToken : string  = 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});
    }
};

1 usage
refreshToken = async (request: any, response: any) : Promise<void>  => {
    const {body} = request;
    const {refreshToken} = body;
    const refreshTokenService : RefreshTokenService  = new RefreshTokenService();

    try {
        const {userId : number | null , isExpired : boolean } = await refreshTokenService
            .isRefreshTokenExpired(refreshToken);

        if (!isExpired && userId) {
            const user : User  = await this.userService.getById(userId);
            const payload : {id: any}  = {id: user.id};
            const accessToken : string  = jwt.sign(payload, jwtOptions.secretOrKey);
            const refreshTokenService : RefreshTokenService  = new RefreshTokenService(user);
            const refreshToken : string  = await refreshTokenService.generateRefreshToken();
            response.send({accessToken, refreshToken});
        } else {
            throw new Error('Invalid credentials');
        }
    } catch (e) {
        response.status(401).send({'error': 'Invalid credentials'});
    }
};
```

Router

```
router.route( prefix: '/create')
    .post(controller.post);

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

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

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

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

router.route( prefix: '/')
    .get(controller.getAllUsers);
```

Services

UserService

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

        if (user) return user.toJSON();

        throw new UserError(`User with id = ${id} not found :(`);
    }

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

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

            throw new UserError(errors);
        }
    }

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

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

        throw new UserError('Incorrect login/password!');
    }

    1 usage
    async getAllUsers() : Promise<User[]>  {
        const users : User[]  = await User.findAll();

        if (users) return users;

        throw new UserError('Users are not found');
    }
}
```

RefreshTokenService

```
class RefreshTokenService {
    private user: User | null;

    3 usages
    constructor(user: User | null = null) {
        this.user = user;
    };

    2 usages
    generateRefreshToken = async (): Promise<string> => {
        const token : `${string}-${string}-${string}... = randomUUID();
        const userId = this.user?.id;
        await RefreshToken.create( values: {token, userId});

        return token;
    };

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

        if (refreshToken) {
            const tokenData = refreshToken.toJSON();
            const currentDate : Date = new Date();
            const timeDelta : number = currentDate.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};
```

Структура проекта

```
└──src
   ├──configs
   ├──controllers
   │  └──users
   ├──core
   ├──errors
   │  └──users
   ├──middlewares
   ├──models
   │  ├──auth
   │  └──users
   ├──providers
   ├──routes
   │  └──v1
   │     └──users
   ├──services
   │  ├──auth
   │  └──users
   └──utils
```

Вывод

В ходе выполнения лабораторной работы был реализован boilerplate с использованием express + sequelize + typescript для быстрого начала разработки приложения.