САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа №1

Выполнила:
Самчук Анита
К34402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

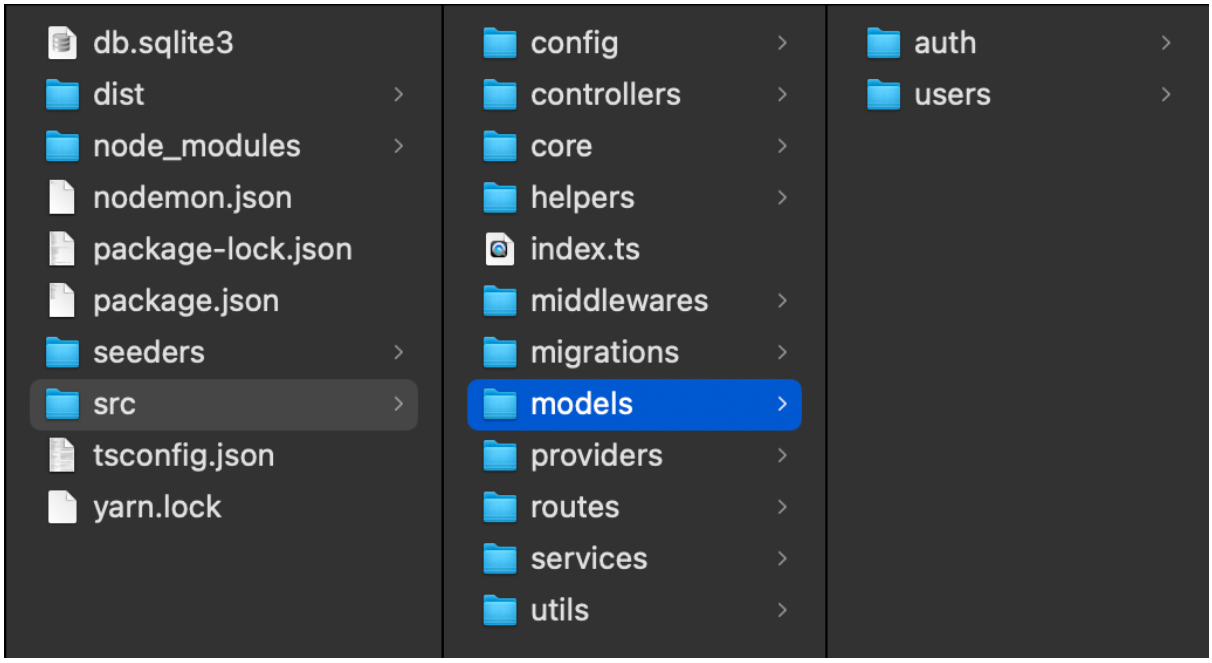
Задача

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

Написанный мной boilerplate имеет следующую структуру



- Core — основной класс для работы с приложением, в нём происходит подключение роутов и базы данных;
- providers — класс для подключения к БД;
- Routes — роуты;
- Middlewares — middleware для проверки авторизованности пользователя;
- Models — модели (User, RefreshToken);
- Controllers, services, repositories — классы для управления данными, реализуют одноименный паттерн.

Дополнительно есть файл .env с переменными окружениям:

```
# DB
NAME="db"
USERNAME="root"
DIALECT="sqlite"
PASSWORD=""
STORAGE="db.sqlite3"
# JWT
ACCESS_TOKEN_LIFETIME=3000000
REFRESH_TOKEN_LIFETIME=3600000
# SERVER
HOST="localhost"
PORT=5555
```

Были созданны модели данных, которые представляют сущности в приложении. Sequelize позволил определить схемы и отношения между моделями, обеспечивая структурированное хранение данных в базе данных.

Модель пользователя:

```
@Table
class User extends Model {
    @Column
    firstName!: string

    @Column
    lastName!: string

    @Column
    patronymic?: string

    @Unique
    @Column
    username!: string

    @Unique
    @Column
    email!: string

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

    6+ usages  Anita Samchuk
    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) : void {...}
}
```

Модель RefreshToken

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

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

Были определены маршруты, которые связывают конечные точки API с соответствующими контроллерами. Это обеспечило навигацию в приложении и управление потоком данных.

Основные маршруты

```
const router :Router  = Router()

router.use('/users', userRouter)
```

Маршруты связанные с пользователем

```
export const userRouter :Router  = Router()
const controller: UserController = new UserController()

// Выводит всех юзеров
userRouter.get( path: "/", controller.getAll)

// Регистрация нового пользователя
userRouter.post( path: "/reg", controller.post)

// Вход в аккаунт
userRouter.post( path: "/login", controller.auth)

// Аккаунт авторизированного пользователя
userRouter.get( path: '/account', passport.authenticate( strategy: 'jwt', options: {session: false}), controller.me)

// Аккаунт пользователя по id
userRouter.get( path: "/:id", controller.get)

// Аккаунт пользователя по username
userRouter.get( path: "/:username", controller.getByUsername)

userRouter.delete( path: "/delete/:id", controller.delete)

// Обновление токена
userRouter.post( path: "/refresh", controller.refreshToken)
```

Для обработки HTTP-запросов реализован контроллер, который принимает входные данные от клиентов, обращается к соответствующим сервисам и возвращает ответы. Это обеспечило четкую сегрегацию бизнес-логики от обработки запросов.

```
export class UserController {
    private userService: UserService

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

    2 usages  Anita Samchuk
    post = async (request: Request, response: Response) : Promise<void>  => {
        const {body} = request
        try {
            const user: User | UserError = await this.userService.createUser(body)

            response.status( code: 201).json(user)
        } catch (error: any) {
            response.status( code: 404).json( body: {error: error.message})
        }
    }

    5+ usages  Anita Samchuk
    get = async (request: Request, response: Response) : Promise<void>  => {
        try {
            const user: User | UserError = await this.userService.getById(Number(request.params.id))
            response.status( code: 200).json(user)
        } catch (error: any) {
            response.status( code: 404).json( body: {error: error.message})
        }
    }
```

```
    getAll = async (request: Request, response: Response) : Promise<void>  => {
        const users: User[] = await this.userService.getAllUsers()
        response.status( code: 200).json(users)
    }
    1 usage  Anita Samchuk
    getByUsername = async (request: Request, response: Response) : Promise<void>  => {
        try {
            const user: User | UserError = await this.userService.getByUsername(request.params.username)
            response.status( code: 200).json(user)
        } catch (error: any) {
            response.status( code: 404).json( body: {error: error.message})
        }
    }

    2 usages  Anita Samchuk
    delete = async (request: Request, response: Response) : Promise<void>  => {
        try {
            await this.userService.deleteUser(Number(request.params.id))
            response.status( code: 204)
        } catch (error: any) {
            response.status( code: 404).json( body: {error: error.message})
        }
    }
```

```
    auth = async (request: Request, response: Response) : 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( body: {accessToken, refreshToken});
            }
        } catch (e) {

        }
    }
```

```
    refreshToken = async (request: Request, response: Response) : 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( body: {accessToken, refreshToken});
            } else {
                throw new Error('Invalid credentials');
            }
        } catch (e) {
            response.status( code: 401).send( body: {'error': 'Invalid credentials'});
        }
    }

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

Для работы с моделями данных мы реализовали сервисы. Это позволило нам инкапсулировать все операции с данными внутри сервисов и обеспечить более гибкий и чистый доступ к данным моделей.

UserService

```
export class UserService {

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

            return user.toJSON()
        } catch (error: any) {
            const errors = error.errors.map((error: any) => error.message)
            throw new UserError(errors)
        }
    }

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

        return []
    }
```

```
    async getById(id: number): Promise<User> {
        const user : User | null  = await User.findByPk(id)
        if (user) return user.toJSON()

        throw new UserError('User is not found')
    }

    1 usage  Anita Samchuk
    async deleteUser(id: number) : Promise<void>  {
        try {
            const isDeleted : number  = await User.destroy( options: {where: {id: id}})
        } catch (e) {
            throw new UserError('Could not delete user')
        }
    }

    1 usage  Anita Samchuk
    async getByUsername(username: string): Promise<User> {
        const user : User | null  = await User.findOne( options: {where: {username: username}})
        if (user) return user.toJSON()

        throw new UserError('User with such username is not found')
    }
```

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

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

RefreshTokenService

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

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

    2 usages  Anita Samchuk
    generateRefreshToken = async (): Promise<string> => {
        const token : `${string}-${string}-${string}...  = randomUUID()

        const userId = this.user?.id

        await RefreshToken.create( values: {token, userId})

        return token
    }
```

```
    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}
    }
}
```

Вывод

В результате выполнения лабораторной работы мы успешно разработали boilerplate приложение на основе Express, Sequelize и TypeScript. Мы четко разделили его на модели, контроллеры, роуты и сервисы. Это обеспечило нам структурированность, чистоту кода и удобство дальнейшей разработки подобных приложений.