САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1: Boilerplate на express + sequelize / TypeORM + typescript

Выполнила:
Балдина Д. Д.
Группа K33401

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Необходимо 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

1. **Первый этап работы над boilerplate - создание файла package.json со всеми необходимыми зависимостями и скриптами для работы приложения.**

   Содержимое package.json:

```
{
    "name": "lr_1",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "prestart": "npm run build",
        "start": "nodemon dist/index.js",
        "build": "npx tsc",
        "lint": "npx eslint . --ext .ts",
        "migrate": "npx sequelize db:migrate"
    },
    "author": "",
    "license": "ISC",
    "dependencies": {
        "bcrypt": "^5.1.0",
        "body-parser": "^1.20.2",
        "cors": "^2.8.5",
        "express": "^4.18.2",
        "passport": "^0.6.0",
        "passport-jwt": "^4.0.1",
        "reflect-metadata": "^0.1.13",
        "sequelize": "^6.30.0",
        "sequelize-typescript": "^2.1.5",
        "sequelize-typescript-migration-v2": "^0.0.2-beta.6",
        "sqlite3": "^5.1.6",
        "tcs": "^10.0.2",
        "typeorm": "^0.3.13",
        "uuid": "^9.0.0"
    },
```

```
    "devDependencies": {
        "@types/bcrypt": "^5.0.0",
        "@types/cors": "^2.8.13",
        "@types/dotenv": "^8.2.0",
        "@types/express": "^4.17.17",
        "@types/express-session": "^1.17.7",
        "@types/flat": "^5.0.2",
        "@types/node": "^18.15.11",
        "@types/passport": "^1.0.12",
        "@types/passport-jwt": "^3.0.8",
        "@types/styled-components": "^5.1.26",
        "@types/styled-system": "^5.1.16",
        "@types/uuid": "^9.0.1",
        "@types/validator": "^13.7.14",
        "@typescript-eslint/eslint-plugin": "^5.57.1",
        "@typescript-eslint/parser": "^5.57.1",
        "dotenv": "^16.0.3",
        "eslint": "^8.38.0",
        "express-list-endpoints": "^6.0.0",
        "nodemon": "^2.0.22",
        "sequelize-cli": "^6.6.0",
        "ts-node": "^10.9.1",
        "typescript": "^5.0.4"
    }
}
```

Далее с помощью npm i были установлены все необходимые зависимости.

Также кроме файла package.json, были созданы необходимые файлы конфигурации окружения, а именно:

- .eslintrc.js - файл конфигурации ESLint

```
{
    "root": true,
    "env": {
        "node": true,
        "commonjs": true},
    "parser": "@typescript-eslint/parser",
    "plugins": [
        "@typescript-eslint"
    ],
    "extends": [
        "eslint:recommended",
        "plugin:@typescript-eslint/eslint-recommended",
        "plugin:@typescript-eslint/recommended"
    ],
    "rules": {
        "@typescript-eslint/no-explicit-any": "off",
        "@typescript-eslint/no-non-null-assertion": "off",
        "@typescript-eslint/ban-ts-comment": "off"
```

```
    }
}
```

- .sequelizerc - файл конфигурации sequelize

```
const path = require('path')

module.exports = {
  'config': path.resolve('src', 'configs/db.js'),
  'models-path': path.resolve('src', 'models'),
  'seeders-path': path.resolve('src', 'seeders'),
  'migrations-path': path.resolve('src', 'migrations')
}
```

- nodemon.json - конфигурационный файл пакета nodemon

```
{
    "watch": ["src"],
    "ext": "ts",
    "ignore": ["src/**/*.spec.ts", "node_modules"],
    "exec": "ts-node ./src/index.ts"
  }
```

- tsconfig.json - файл конфигурации TypeScript для настройки компилятора ЯП

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "strictPropertyInitialization": false,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "allowJs": true,
  },
}
```

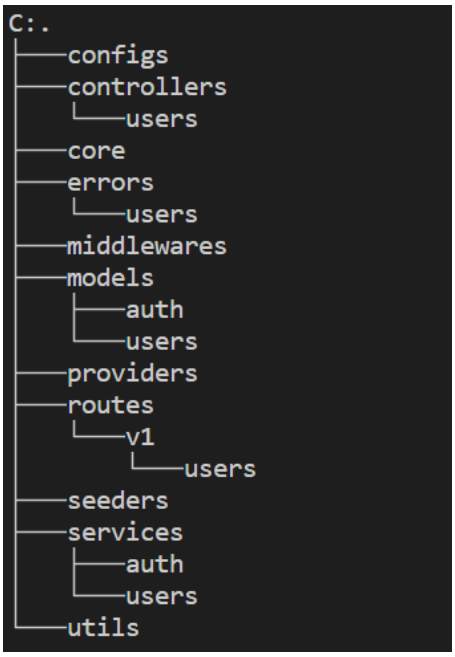
- .env – файл с переменными окружения

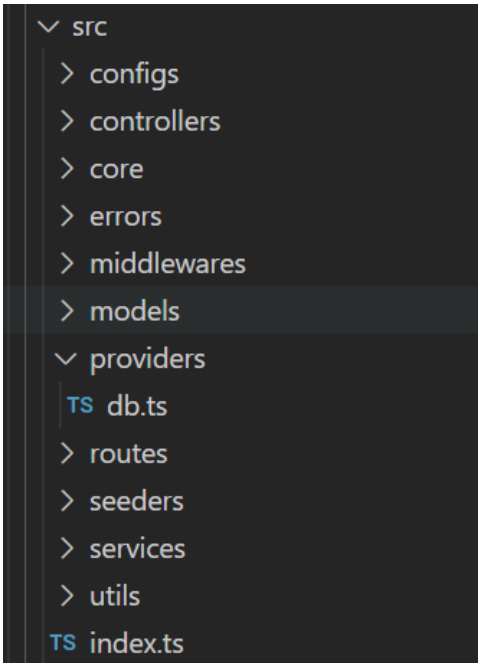
```
# DATABASE
NAME = "some_db"
USERNAME = "root"
DIALECT = "sqlite"
PASSWORD = ""
STORAGE = "db.sqlite"
```

```
# JWT
ACCESS_TOKEN_LIFITIME = 300000 ; in milliseconds
REFRESH_TOKEN_LIFITIME = 3600000

# SERVER
HOST = "localhost"
PORT = 8000
```

2. Структура приложения





Корень исходных файлов **src** включает в себя следующие директории:

- **core** – точка входа в приложение, объединяющая в себе все составляющие;
- **configs** – файлы конфигурации (файл для подключения к БД);
- **controllers** – контроллеры, отвечающие за логику обработки http-запросов;
- **models** – модели sequelize;
- **providers** – точки доступа к данным;
- **routes** – описание маршрутов express;
- **seeders** – управление всеми миграциями;

- **services** – службы, которые содержат запросы к базе данных и возвращают объекты или выдают ошибки;
- **utils** – вспомогательные файлы, которые используются во всем приложении
- **middlewares** - содержит аутентификацию с использованием passport.js.

3. Модели

User.ts – модель пользователя

```
import { Table, Column, Model, Unique, AllowNull, BeforeCreate, BeforeUpdate }
from 'sequelize-typescript'
import hashPassword from '../../utils/hashPassword'

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

    @Column
    firstName: string

    @Column
    lastName: 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 = hashPassword(password)
        }
    }
}

export default User
```

RefreshToken.ts - модель хранения токенов

```
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
```

4. Контроллеры

```
import User from '../../models/users/User'
import UserService from '../../services/users/User'
import UserError from '../../errors/users/User'
import jwt from 'jsonwebtoken'
import { jwtOptions } from '../../middlewares/passport'
import RefreshTokenService from '../../services/auth/RefreshToken'

class UserController {
    private userService: UserService

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

    get = async (request: any, response: any) => {
        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 })
        }
    }

    post = async (request: any, response: any) => {
        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 })
        }
    }

    me = async (request: any, response: any) => {
        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 })
        }
    }

    getAll = async (request: any, response: any) => {
        try {
            const users = await this.userService.getAll()

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

```
    getByUsername = async (request: any, response: any) => {
        try {
            const user = await this.userService.getByUsername(
                request.params.username
            )

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

            if (!isExpired && userId) {
                const user = await this.userService.getById(userId)

                const payload = { id: user.id }

                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('Invalid credentials')
            }
        } catch (e) {
            response.status(401).send({ 'error': 'Invalid credentials' })
        }
    }
}

export default UserController
```

Методы класса UserController:

- get: находит пользователя по id;
- post: создание нового пользователя;
- me: возвращает данные о пользователе, полученные из объекта запроса;
- auth: генерирует новый токен доступа и токен обновления, если пользователь залогинился;
- refreshToken: генерирует новый JWT токен;
- getAll: получает информацию по всем пользователям
- getByUsername: находит пользователя

5. Services

User.ts

```
import User from '../../models/users/User'
import UserError from '../../errors/users/User'
import checkPassword from '../../utils/checkPassword'

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

        if (user) return user.toJSON()

        throw new UserError('Not found!')
    }

    async create(userData: any) {
        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 getAll() {
```

```
        const users = await User.findAll()

        if (users) return users

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

    async getByUsername(username: string) {
        const user = await User.findOne({where: {username: username}})

        if (user) return user.toJSON()

        throw new UserError('User with this username not found')
    }

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

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

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

export default UserService
```

RefreshToken.ts

```
import RefreshToken from "../../models/auth/RefreshToken"
import User from "../../models/users/User"
import { randomUUID } from "crypto"


class RefreshTokenService {
    private user: User | null

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

    generateRefreshToken = async () : Promise<string> => {
        const token = randomUUID()

        const userId = this.user?.id

        await RefreshToken.create({ token, userId })

        return token
```

```
    }

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

        if (refreshToken) {
            const tokenData = refreshToken.toJSON()

            const currentDate = new Date()
            const timeDelta = 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 }
    }
}

export default RefreshTokenService
```

6. Routes

```
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('/reg')
    .post(controller.post)

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

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

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

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

```
router.route('/accounts')
    .get(controller.getAll)

router.route('/accounts/:username')
    .get(controller.getByUsername)

export default router
```

Вывод

В ходе первой лабораторной работы был разработан boilerplate с продуманной структурой и необходимыми конфигурационными файлами с помощью следующих инструментов: express, sequelize/TypeORM, typescript.