САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнила:

Мухина Юлия

Группа К34401

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

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

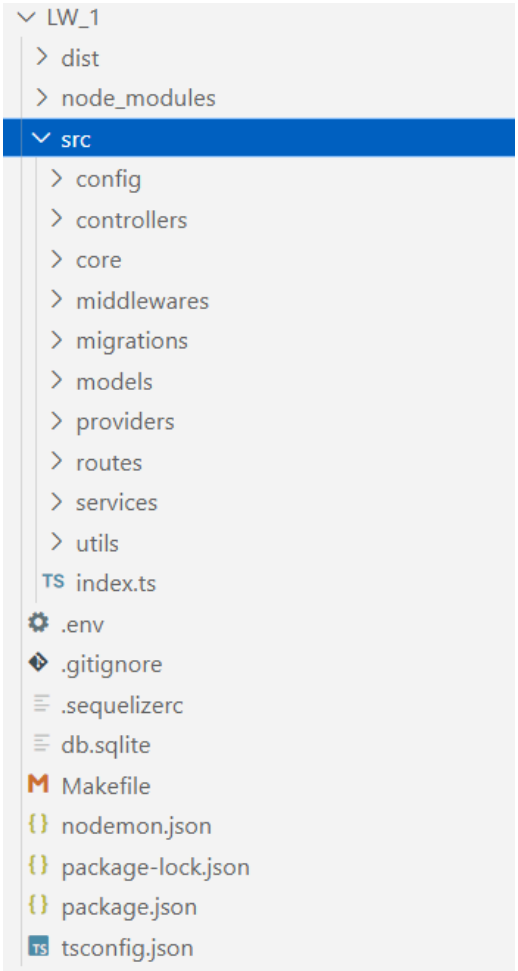
Должно быть явное разделение на:

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

Ход работы

За основу данного boilerplate взят boilerplate, продемонстрированный на лекции.

Структура boilerplate.



Реализована модель пользователя

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

@Table({
    createdAt: false,
    updatedAt: false,
})
class User extends Model {
    @AllowNull(false)
    @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
```

Реализована модель токена

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

@Table({
    createdAt: false,
    updatedAt: false,
})
class Token extends Model {
    @Unique
    @AllowNull(false)
    @Column
    token: string

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

export default Token
```

Реализован контроллер юзера, в котором есть метод получения юзера по id

```
import User from '../models/user.model'
import UserService from '../services/user.service'
import jwt from 'jsonwebtoken'
import {jwtOptions} from '../middlewares/passport'
import TokenService from '../services/token.service'

class UserController {
    private userService: UserService

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

    get = async (request: any, response: any) => {
        try {
            const user: User | Error = await this.userService.getById(
                Number(request.params.id)
            )
            response.send(user)
        } catch (error: any) {
            response.status(404).send({"error": error.message})
        }
    }
```

Метод создания юзера

```

    post = async (request: any, response: any) => {
        const {body} = request
        try {
            const user: User | Error = await this.userService.create(body)
            response.status(201).send(user)
        } catch (error: any) {
            response.status(400).send({"error": error.message})
        }
    }
```

Метод получения профиля юзера

```

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

```

Аутентификация

```
38
39      auth = async (request: any, response: any) => {
40          const {body} = request
41          const {email, password} = body
42          try {
43              const {user, checkPassword} = await this.userService.checkPassword(email, password)
44              if (checkPassword) {
45                  const payload = {id: user.id}
46                  const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
47                  const tokenService = new TokenService(user)
48                  const token = await tokenService.generateToken()
49                  response.send({accessToken})
50              } else {
51                  throw new Error('Login or password is incorrect!')
52              }
53          } catch (e: any) {
54              response.status(401).send({"error": e.message})
55          }
56      }
```

Обновление токена

```
57
58      refreshToken = async (request: any, response: any) => {
59          const {body} = request
60          const {token} = body
61          const tokenService = new TokenService()
62          try {
63              const {userId, isExpired} = await tokenService.isTokenExpired(token)
64              if (!isExpired && userId) {
65                  const user = await this.userService.getById(userId)
66                  const payload = {id: user.id}
67                  const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
68                  const refreshTokenService = new TokenService(user)
69                  const refreshToken = await refreshTokenService.generateToken()
70                  response.send({accessToken, refreshToken})
71              } else {
72                  throw new Error('Invalid credentials')
73              }
74          } catch (e) {
75              response.status(401).send({'error': 'Invalid credentials'})
76          }
77      }
78  }
79
80  export default UserController
```

Роуты юзера

```ts
import express from "express"
import UserController from "../controllers/user.controller"
import passport from "../middlewares/passport"

const userRoutes: express.Router = express.Router()
const controller: UserController = new UserController()

userRoutes.route('/').post(controller.post)
userRoutes.route('/profile').get(passport.authenticate('jwt', {session: false}), controller.me)
userRoutes.route('/login').post(controller.auth)
userRoutes.route('/refresh').post(controller.refreshToken)

export default userRoutes
```

```ts
import express from "express"
import userRoutes from "./user.routes"

const router: express.Router = express.Router()
router.use('/users', userRoutes)

export default router
```

Сервис юзера

```ts
import User from '../models/user.model'
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 Error('Not found!')
    }

    async create(userData: any): Promise<User | Error> {
        const user = await User.create(userData)
        return user.toJSON()
    }

    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 Error('Incorrect login/password!')
    }
}

export default UserService
```

Сервис токена

```ts
import Token from "../models/token.model"
import User from "../models/user.model"
import {randomUUID} from "crypto"

class TokenService {
    private user: User | null

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

    generateToken = async (): Promise<string> => {
        const token = randomUUID()
        const userId = this.user?.id
        await Token.create({token, userId})
        return token
    }

    isTokenExpired = async (token: string): Promise<{ userId: number | null, isExpired: boolean }> => {
        const tokenInstance = await Token.findOne({where: {token}})
        if (tokenInstance) {
            const tokenData = tokenInstance.toJSON()
            const currentDate = new Date()
            const timeDelta = currentDate.getTime() - tokenData.createdAt.getTime()
            if (timeDelta > 0 && timeDelta < parseInt(process.env.REFRESH_TOKEN_LIFETIME!, 10)) {
                return {userId: tokenData.userId, isExpired: false}
            }
            return {userId: null, isExpired: true}
        }
        return {userId: null, isExpired: true}
    }
}

export default TokenService
```

Также прописаны переменные окружения

```
# SEQUELIZE
USERNAME="root"
PASSWORD=""
DATABASE="boilerplate"
HOST="localhost"
PORT=8000
DIALECT="sqlite"
STORAGE="db.sqlite"

# JWT
ACCESS_TOKEN_LIFETIME=300000
REFRESH_TOKEN_LIFETIME=3600000
```

Вывод

Написан boilerplate на express + sequelize + typescript, который можно использовать в последующем для создания приложения.