САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:
Кривцов Павел
Группа К33402

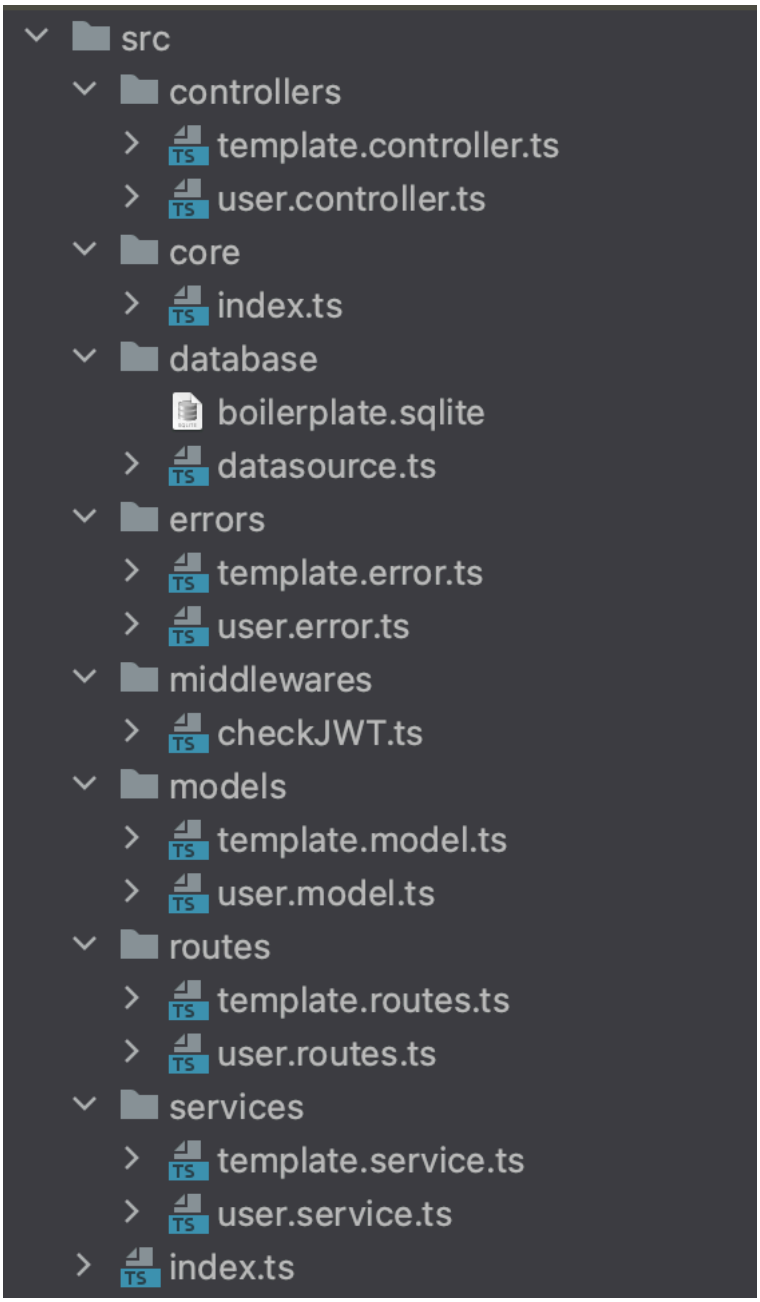
Проверил:
Добряков Д. И.

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
Структура boilerplate:



Модель пользователя:

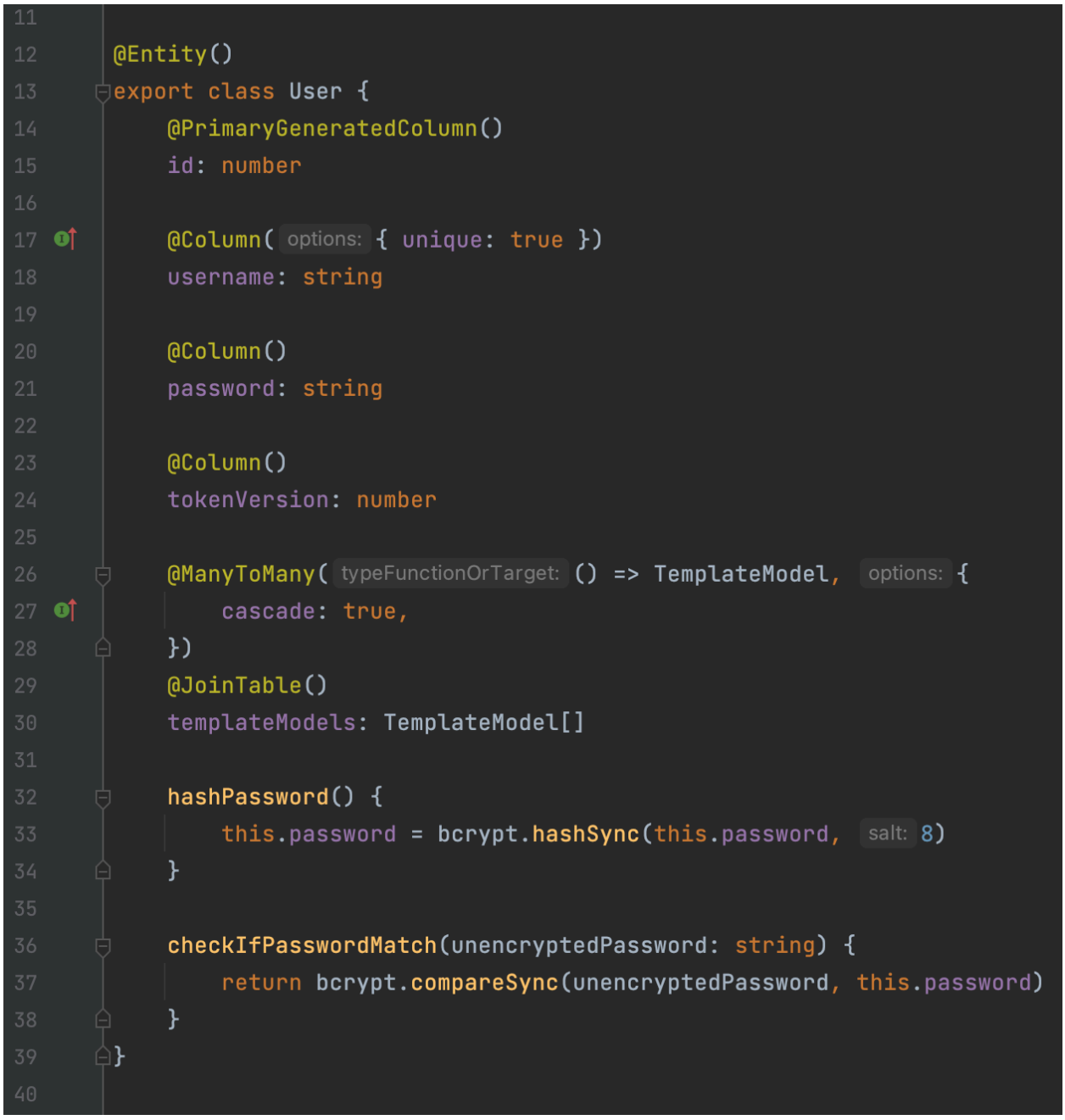
```
const userService = new UserService()

class UserController {
    login = async (request: Request, response: Response) => {
        const { username, password } = request.body

        if (!(username && password)) {
            return response
                .status( code: 400)
                .send( body: { error: 'Empty name or password' })
        }

        try {
            const user = await userService.getByUsername(username)
            if (!user.checkIfPasswordMatch(password)) {
                return response.status( code: 401).send( body: { error: 'Wrong password' })
            }
            const newTokenVersion = await userService.updateUserTokenVersion(
                user.username
            )
            const token = jwt.sign(
                 payload: {
                    userId: user.id,
                    username: user.username,
                    v: newTokenVersion,
                },
                process.env.JWT_SECRET as string,
                 options: { expiresIn: process.env.JWT_LIFETIME as string }
            )
            response.send(token)
        } catch (error) {
            return response.status( code: 401).send( body: { error: 'User does not exist' })
        }
    }
```

Сервисы для пользователя:

```ts
const userRepository = AppDataSource.getRepository(User)
const templateService = new TemplateService()

class UserService {
    async getAll() {
        return await userRepository.find( options: {
            select: ['id', 'username'],
        })
    }

    async getByUsername(username: string) {
        return await userRepository.findOneOrFail( options: {
            select: ['id', 'username', 'password', 'tokenVersion'],
            relations: {
                templateModels: true,
            },
            where: { username: username },
        })
    }

    async create(username: string, password: string) {
        const user = new User()
        user.username = username
        user.password = password
        user.tokenVersion = 1
        user.hashPassword()
        return await userRepository.save(user)
    }

    async addOrDeleteTemplate(
        username: string,
        templateId: number,
        add: boolean
    ) {
        const template = await templateService.getById(templateId)
        const user = await this.getByUsername(username)

        if (add) {
            await userRepository
                .createQueryBuilder() SelectQueryBuilder<User>
                .relation(User, propertyPath: 'templateModels') RelationQueryBuilder<User>
                .of(user) RelationQueryBuilder<User>
                .add(template)
        } else {
            await userRepository
                .createQueryBuilder() SelectQueryBuilder<User>
                .relation(User, propertyPath: 'templateModels') RelationQueryBuilder<User>
                .of(user) RelationQueryBuilder<User>
                .remove(template)
        }
        return await this.getByUsername(username)
    }

    async updateUserTokenVersion(username: string) {
        const user = await this.getByUsername(username)
        user.tokenVersion += 1
        await userRepository.save(user)
        return user.tokenVersion
    }
}
```

Контроллеры:

- логин пользователя



```
@Entity()
export class User {
    @PrimaryGeneratedColumn()
    id: number

    @Column( options: { unique: true })
    username: string

    @Column()
    password: string

    @Column()
    tokenVersion: number

    @ManyToMany( typeFunctionOrTarget: () => TemplateModel, options: {
        cascade: true,
    })
    @JoinTable()
    templateModels: TemplateModel[]

    hashPassword() {
        this.password = bcrypt.hashSync(this.password, salt: 8)
    }

    checkIfPasswordMatch(unencryptedPassword: string) {
        return bcrypt.compareSync(unencryptedPassword, this.password)
    }
}
```

- регистрация пользователя

```
    getAll = async (request: Request, response: Response) => {
        const allUsers = await userService.getAll()
        return response.send(allUsers)
    }

    addTemplate = async (request: Request, response: Response) => {
        const templateId = Number(request.params.id)
        const username = response.locals.jwtPayload.username
        const user = await userService.addOrDeleteTemplate(
            username,
            templateId,
            add: true
        )
        return response.send(user)
    }

    removeTemplate = async (request: Request, response: Response) => {
        const templateId = Number(request.params.id)
        const username = response.locals.jwtPayload.username
        const user = await userService.addOrDeleteTemplate(
            username,
            templateId,
            add: false
        )
        return response.send(user)
    }

    me = async (request: Request, response: Response) => {
        const username = response.locals.jwtPayload.username
        const user = await userService.getByUsername(username)
        return response.send(user)
    }
}
```

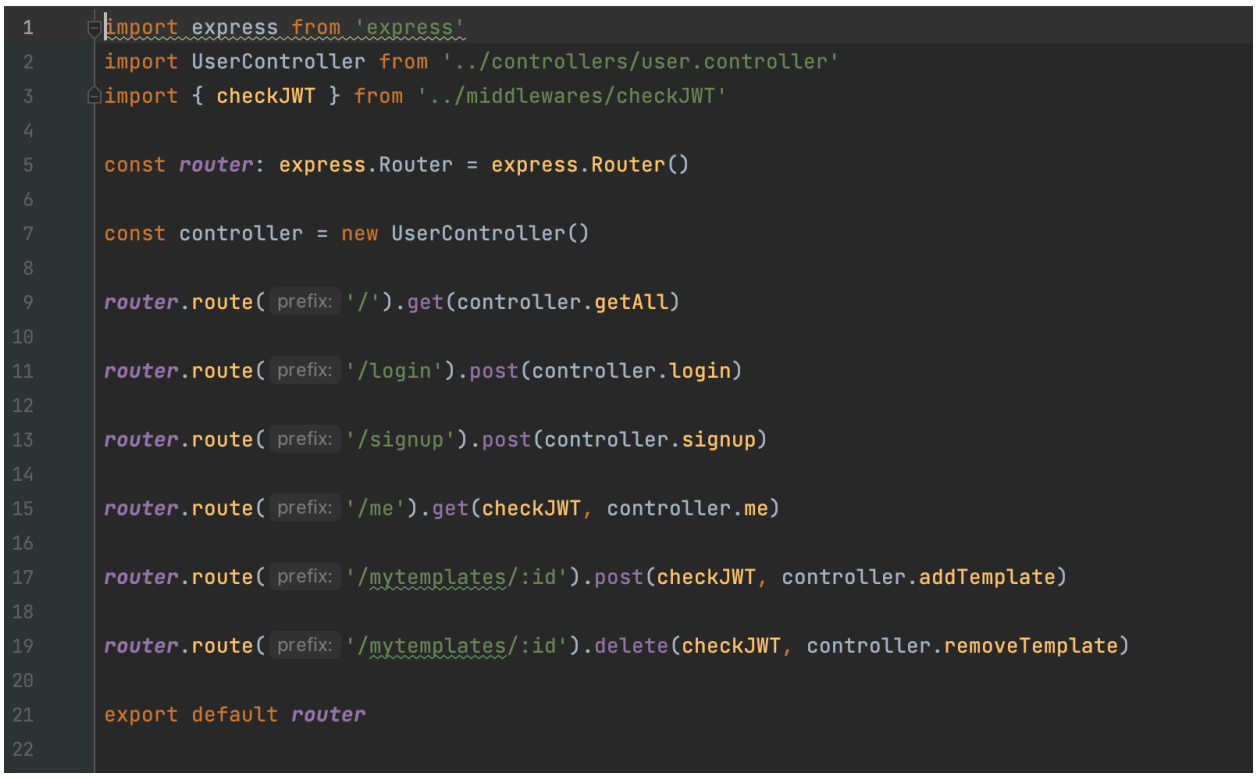
- остальные методы контроллера пользователя

```
    signup = async (request: Request, response: Response) => {
        const { username, password } = request.body
        try {
            await userService.create(username, password)
            response.status( code: 201).send( body: { msg: 'User created' })
        } catch (error) {
            return response.status( code: 409).send( body: { msg: 'Username already in use' })
        }
    }
```

Middleware для проверки авторизованности пользователя:

```ts
import { Request, Response, NextFunction } from 'express'

import * as jwt from 'jsonwebtoken'
import { JwtPayload } from 'jsonwebtoken'

import UserService from '../services/user.service'
import 'dotenv/config'

const userService = new UserService()

export const checkJWT = async (
    request: Request,
    response: Response,
    next: NextFunction
) => {
    const token = <string>request.headers['auth']
    let jwtPayload: JwtPayload | string

    try {
        jwtPayload = jwt.verify(
            token,
            process.env.JWT_SECRET as string
        ) as JwtPayload
        response.locals.jwtPayload = jwtPayload
        const user = await userService.getByUsername(jwtPayload.username)
        if (user.tokenVersion != jwtPayload.v) {
            throw { status: 404, message: 'Not Found' }
        }
    } catch (error) {
        response.status( code: 401).send( body: { error: 'Invalid token' })
        return
    }
    next()
}

```

Роуты:



```
import express from 'express'
import UserController from '../controllers/user.controller'
import { checkJWT } from '../middlewares/checkJWT'

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

const controller = new UserController()

router.route( prefix: '/').get(controller.getAll)

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

router.route( prefix: '/signup').post(controller.signup)

router.route( prefix: '/me').get(checkJWT, controller.me)

router.route( prefix: '/mytemplates/:id').post(checkJWT, controller.addTemplate)

router.route( prefix: '/mytemplates/:id').delete(checkJWT, controller.removeTemplate)

export default router

```

Вывод

В ходе работы был создан boilerplate с помощью express и TypeORM. Созданный шаблон можно использовать в дальнейших лабораторных работах.