САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 1

Выполнила:

Афанасьева Ирина Максимовна

Группа:

К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

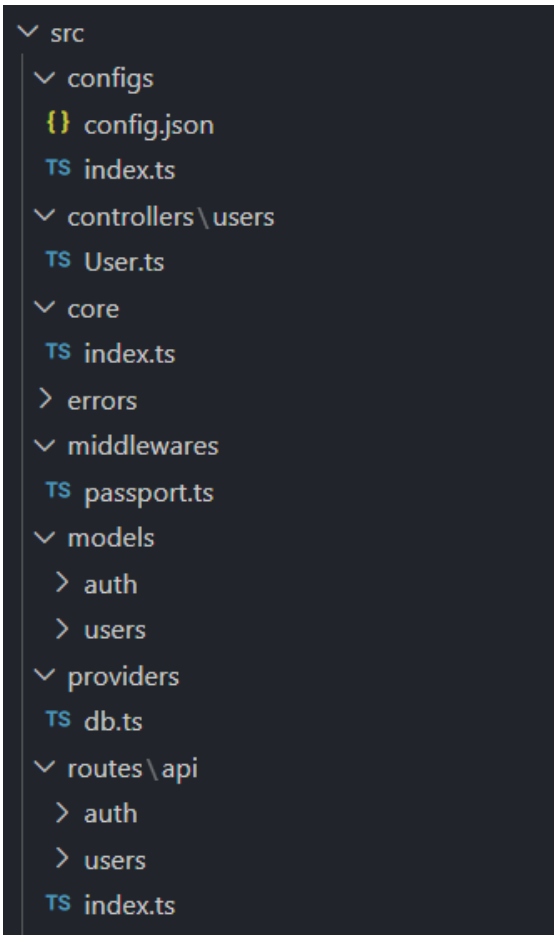
Задание
Написать свой boilerplate на express + sequelize / TypeORM + typescript.
Должно быть явное разделение на:

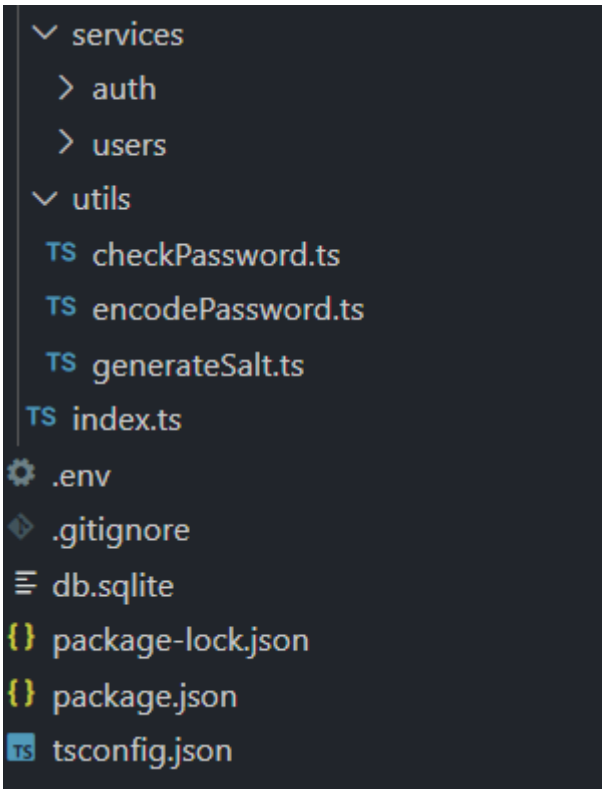
- модели
- контроллеры
- роуты
- сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы

ORM – sequelize.

Структура проекта:





- configs: Папка с файлами конфигурации.
- controllers: Папка с контроллерами, отвечающими за обработку запросов.
- core: Папка с основными компонентами приложения.
- errors: Папка с классами ошибок приложения.

- middlewares: Папка с промежуточными обработчиками (middlewares).
- models: Папка с моделями базы данных.
- providers: Папка с провайдерами, отвечающими за подключение к базе данных и другие сервисы.
- routes: Папка с маршрутами (routes).
- services: Папка с сервисами, которые предоставляют бизнес-логику.
- utils: Папка с утилитами, вспомогательными функциями.

Реализованы 2 модели: User и RefreshToken и 5 эндпоинтов: логин, регистрация, создание пользователя, получение текущего пользователя, изменение текущего пользователя. Последние 2 эндпоинта требуют авторизации при помощи JWT.

Реализованные методы класса UserController:

POST - обрабатывает POST-запрос для создания нового пользователя.

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

GET - обрабатывает GET-запрос для получения информации о текущем пользователе. Если пользователь аутентифицирован, возвращается информация о пользователе. В противном случае возвращается статус 401 (Unauthorized) с сообщением об отсутствии аутентификации.

```
get = async (request: any, response: any) => {
    const {user} = request
    if (user) {
        response.send(user)
    } else {
        response.status(401).send({'detail': 'Not authenticated'})
    }
}
```

PUT - обрабатывает PUT-запрос для обновления информации о текущем пользователе. Если пользователь аутентифицирован, вызывается метод update из UserService для обновления информации о пользователе. Если операция успешна, возвращается статус 200 (OK) и обновленный пользователь в ответе.

```
put = async (request: any, response: any) => {
    const {body, user} = request
    if (user) {
        try {
            const updatedUser: User | APIError = await this.userService.update(user.id, body)
            response.status(200).send(updatedUser)
        } catch (error: any) {
            response.status(400).send({'detail': error.message})
        }
    } else {
        response.status(401).send({'detail': 'Not authenticated'})
    }
}
```

LOGIN - обрабатывает POST-запрос для аутентификации пользователя. Входные данные из запроса (электронная почта и пароль) извлекаются из request.body. Затем вызывается метод checkPassword из UserService, который проверяет правильность пароля и возвращает пользователя и флаг checkPassword. Если пароль верный, генерируется JWT-токен доступа и обновления.

```
login = 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}
            const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
            const refreshTokenService = new RefreshTokenService(user)
            const refreshToken = await refreshTokenService.generateRefreshToken()

            response.send({accessToken, refreshToken})
        } else {
            response.status(400).send({'detail': 'Invalid credentials'})
        }
    } catch (e: any) {
        response.status(400).send({'detail': e.message})
    }
}
```

REFRESHTOKEN - обрабатывает POST-запрос для обновления токенов.

```
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) {
            try {
                const user = await this.userService.getById(userId)
                // @ts-ignore
                const payload = {id: user.id}
                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
                // @ts-ignore
                const refreshTokenService = new RefreshTokenService(user)
                const refreshToken = await refreshTokenService.generateRefreshToken()
                response.send({accessToken, refreshToken})
            } catch (e: any) {
                response.status(400).send({'detail': e.message})
            }
        } else {
            response.status(401).send({'error': 'Invalid credentials'})
        }
    } catch (e: any) {
        response.status(401).send({'error': e.message})
    }
}
```

Реализованные методы класса UserService:

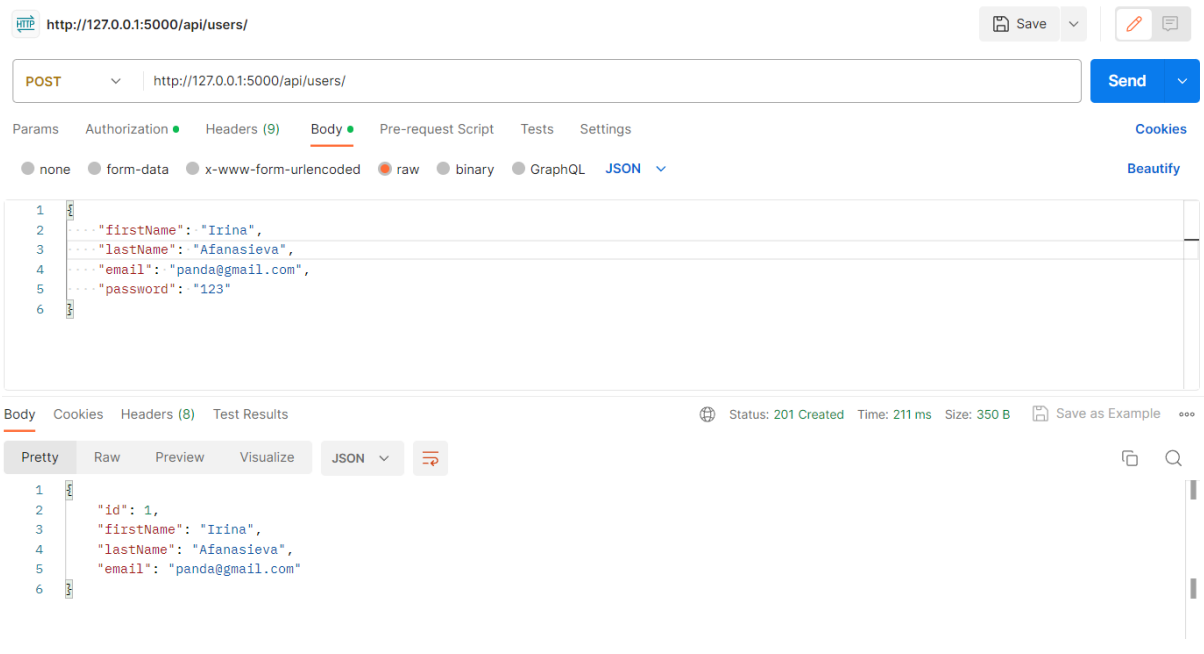
```
async getById(id: number): Promise<User | APIError> {
    const user = await User.findByPk(id)
    if (user) {
        return user.toJSON()
    }
    throw new APIError('User not found')
}
```

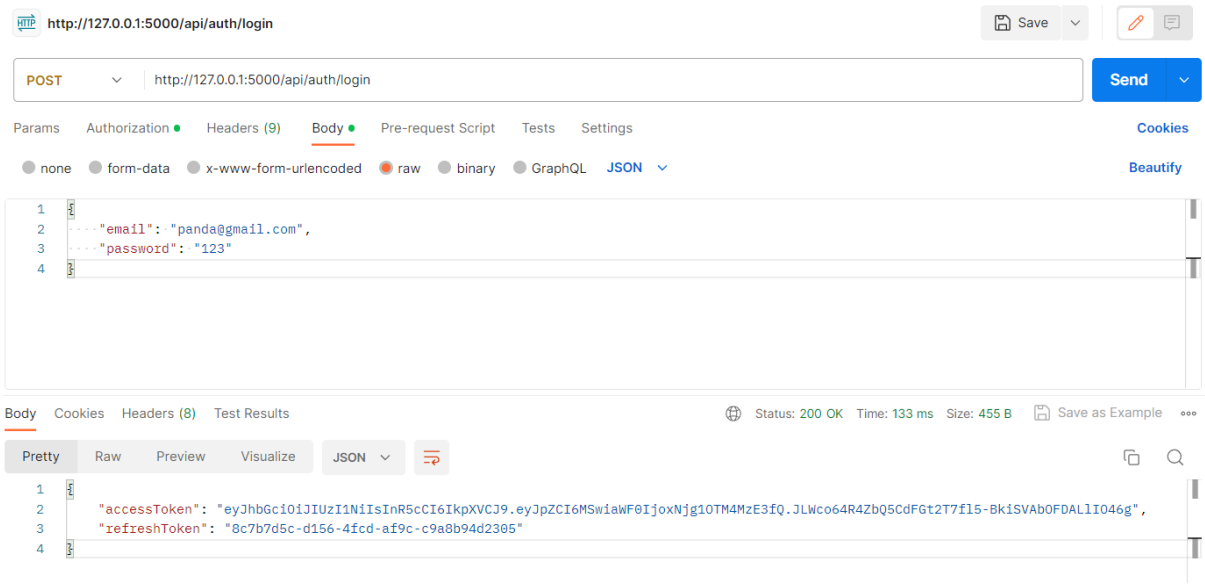
```
async create(userData: any): Promise<User | APIError> {
    try {
        const user = await User.create(userData)
        await user.reload()
        return user.toJSON()
    } catch (e: any) {
        const errors = e.errors.map((error: any) => error.message)
        throw new APIError(errors)
    }
}
```

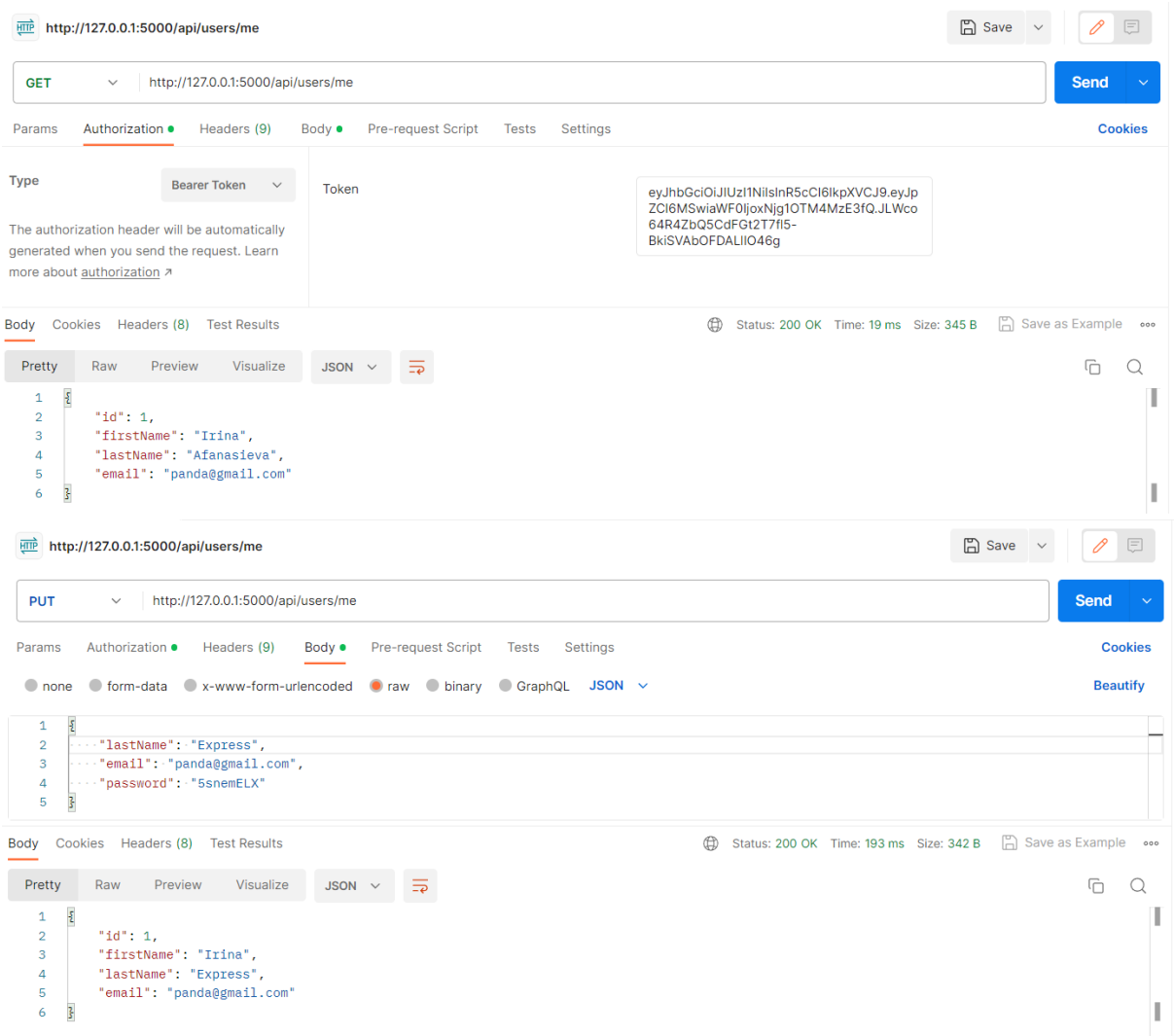
```
async update(id: number, userData: any): Promise<User | APIError> {
    let user = await User.findByPk(id)
    if (user) {
        try {
            user = await user.update(userData)
            await user.reload()
            return user.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new APIError(errors)
        }
    }
    throw new APIError('User not found')
```

```
async checkPassword(email: string, password: string): Promise<any> {
    const user = await User.scope('withPassword').findOne({where: {email}})
    if (user) {
        return {user: user.toJSON(), checkPassword: checkPassword(user, password)}
    }
    throw new APIError('Incorrect credentials')
}
```

Создадим запросы, чтобы проверить работу boilerplate:







Вывод

В ходе работы были получены практические навыки работы с express, sequelize-typescript и создан готовый шаблон boilerplate для последующей работы.