

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Ананьев Никита

К3340

Проверил:

Добряков Д. И.

Санкт-Петербург

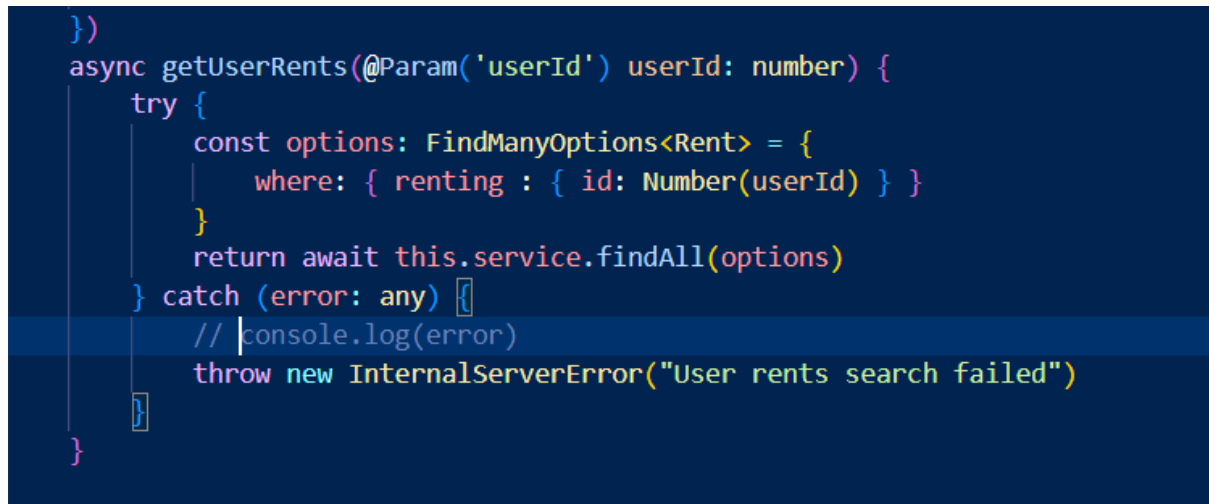
2025 г.

Задача

Написать свой boilerplate на express + TypeORM + typescript.

Ход работы

Для удобной работы с контроллерами и дальнейшей документации был использован пакет routing-controllers. Пример метода контроллера (рис. 1):

A screenshot of a code editor showing a TypeScript method for a controller. The code is as follows:

```
    })
    async getUserRents(@Param('userId') userId: number) {
        try {
            const options: FindManyOptions<Rent> = {
                where: { renting : { id: Number(userId) } }
            }
            return await this.service.findAll(options)
        } catch (error: any) {
            // console.log(error)
            throw new InternalServerError("User rents search failed")
        }
    }
}
```

Рисунок 1 - пример метода контроллера аренд

К API был добавлен ряд следующих роутов:

- [POST] /user/register - регистрирует юзера в системе.
- [POST] /user/login - проверяет данные авторизации и возвращает jwt токен в случае успеха.
- [GET] /messages/{userId}/sent - список сообщений, отправленных юзером.
- [GET] /messages/{userId}/received - список сообщений, полученных юзером.

Так как была добавлена логика авторизации, логичным шагом стало сделать те или иные роуты защищенными. С этой целью был реализован middleware, проверяющий наличие и валидность jwt-токена (см. рис. 2):

```
1 import { Request, Response, NextFunction } from 'express';
2 import { HttpCodes } from '../handlers/Codes';
3 import { AuthService } from '../auth';
4
5 export function AuthMiddleware(req: Request, res: Response, next: NextFunction): void {
6     const authHeader: string | undefined = req.headers.authorization
7     if (!authHeader) {
8         res.status(HttpCodes.UNAUTHORIZED).json({message: "authorization header is missing"})
9         return;
10    }
11    const token: string = authHeader.split(' ')[1];
12    try {
13        AuthService.verifyJWT(token)
14    } catch (error: any) {
15        res.status(HttpCodes.UNAUTHORIZED).json({message: "jwt verification failed"})
16        return;
17    }
18    next()
19 }
20
```

Рисунок 2 - middleware авторизации

Для работы с логикой формирования jwt, хэширования паролей и их проверки, был создан класс AuthService (см. рис. 3):

```
1 import jwt from "jsonwebtoken";
2 import bcrypt from 'bcryptjs';
3
4 const salt: number = 10;
5
6 export class AuthService {
7     static generateJWT(userId: number): string {
8         return jwt.sign({ userId }, process.env.JWT_SECRET!, { expiresIn: '1h' })
9     }
10
11     static verifyJWT(token: string) {
12         return jwt.verify(token, process.env.JWT_SECRET!)
13     }
14
15     static async hashPassword(password: string): Promise<string> {
16         return bcrypt.hash(password, salt);
17     }
18
19     static async comparePasswords(password: string, hash: string): Promise<boolean> {
20         return bcrypt.compare(password, hash)
21     }
22 }
23
```

Рисунок 3 - реализация класса AuthService

Для проверки наличия и валидации jwt в поступившем запросе используется middleware функция (см. рис. 4):

```
1 import { Request, Response, NextFunction } from 'express';
2 import { HttpCodes } from '../handlers/Codes';
3 import { AuthService } from '../auth';
4
5 export function AuthMiddleware(req: Request, res: Response, next: NextFunction): void {
6   const authHeader: string | undefined = req.headers.authorization
7   if (!authHeader) {
8     res.status(HttpCodes.UNAUTHORIZED).json({message: "authorization header is missing"})
9     return;
10  }
11  const token: string = authHeader.split(' ')[1];
12  try {
13    AuthService.verifyJWT(token)
14  } catch (error: any) {
15    res.status(HttpCodes.UNAUTHORIZED).json({message: "jwt verification failed"})
16    return;
17  }
18  next()
19 }
```

Рисунок 4 - middleware аутентификации

Вывод

В ходе выполнения работы удалось реализовать авторизацию с помощью JWT, заодно познакомиться и разобраться с использованием middleware и routing controllers.