**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

# Лабораторная работа №1

Выполнил:

Захарчук Александр

К3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

**Задача**

Нужно написать свой boilerplate на express + TypeORM + typescript.

**Ход работы**

Первым этапом стало добавление JWT-авторизации с помощью middleware. Его код представлен на рисунке 1.

```
import { Request, Response, NextFunction } from "express";
import * as jwt from "jsonwebtoken";

export const checkJwt = (req: Request, res: Response, next: NextFunction) => {
  const authHeader = <string>req.headers["authorization"];
  if (!authHeader) {
    res.status(401).send({detail: "Unathorized"});
    return;
  }

  const token = authHeader.split(" ")[1];
  if (!token) {
    res.status(401).send({detail: "Unathorized"});
    return;
  }

  try {
    const jwtPayload = <any>jwt.verify(token, process.env.JWT_SECRET);
    res.locals.jwtPayload = jwtPayload;
  } catch (error) {
    res.status(401).send({detail: "Unathorized"});
    return;
  }
  next();
};
```

Рисунок 1 - Middleware для авторизации

Также был добавлен endpoint для получения токена. Его код представлен на рисунке 2.

```
userRouter.post("/login", async function (req: Request, res: Response) {
    const { username, password } = req.body;
    const user = await userRepository.findOneBy({username});
    if (!user) {
        res.status(404).json({ detail: `User with username ${username} not found` });
        return;
    }

    const passwordMatch = await comparePassword(user.password_hash, password);

    if (!passwordMatch){
        res.status(401).json({detail: "Unathorized"});
    }

    const token = jwt.sign(
        {username},
        process.env.JWT_SECRET,
        {expiresIn: process.env.JWT_EXPIRES_IN},
    )

    res.send({token});
})
```

Рисунок 2 - Endpoint для получения токена

Для хеширования паролей была применена библиотека bcrypt. Код вспомогательных методов для работы с паролями представлен на рисунке 3.

```
const hashPassword = async (password: string): Promise<string> => {
    const salt = await bcrypt.genSalt();
    const hash = await bcrypt.hash(password, salt);
    return hash;
}

const comparePassword = async (hash: string, password: string): Promise<boolean> => {
    const isMatch = await bcrypt.compare(password, hash);
    return isMatch;
}
```

Рисунок 3 - Методы для работы с паролями

На основные API-методы была добавлена авторизация с помощью реализованного ранее middleware. В обработчике из расшифрованного токена извлекается имя пользователя, которое используется для установления владельца сущности. Пример создания рецепта представлен на рисунке 4.

```
recipeRouter.post("/", [checkJwt], async function (req: Request, res: Response) {
    const username = res.locals.jwtPayload.username;
    const user = await userRepository.findOneBy({username})

    if (!user) {
        res.status(404).json({detail: `User with username ${username} not found`});
        return;
    }

    const recipe = recipeRepository.create({...req.body, user});
    const results = await recipeRepository.save(recipe);
    res.send(results);
})
```

Рисунок 4 - Endpoint для создания рецепта

Для конфигурации сервиса были применены переменные окружения и библиотека dotenv. Пример конфигурации подключения к базе данных представлен на рисунке 5.

```
export const dataSource = new DataSource({
    type: "postgres",
    host: process.env.DB_HOST,
    port: parseInt(process.env.DB_PORT),
    username: process.env.DB_USERNAME,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
    entities: ["src/entities/*.ts"],
    logging: true,
    synchronize: true,
});
```

Рисунок 5 - Конфигурация подключения к базе данных

**Вывод**

В ходе выполнения данной лабораторной работы была реализована JWT авторизация с помощью middleware, добавлен endpoint для получения токена, а также применены переменные окружения для конфигурации сервиса. Все это позволило доработать код приложения до соответствия boilerplate.