

# JavaEE2024-WMS-stage2

文档摘要（强烈建议使用飞书链接查看文档 [📖 JavaEE2024-WMS-stage2](#)）

💡 此文档说明了[项目简介（项目使用的技术栈）](#)、[需求分析与功能设计](#)、[数据持久层设计](#)、[API与功能详细设计](#)、[测试](#)、[阶段总结](#)

## 一、项目简介（技术栈介绍）

这是Automatic Warehouse Management System的第2阶段开发（后端），完成了下面所有项目要求。

### ***Basic Requirements:***

- 1, Follow the Restful APIs defined in Assignment 1.
- 2, Implement the API in Java with Spring WebFlux API.
- 3, Implement the reactive service layer and dao layer.
- 4, Testing with boot testing framework and WebClient/WebTestClient API.
- 5, API authentication, using spring security and JWT will be given credit.
- 6, Using functional api instead of annotation will be given credit.

### ***Credit Implementations:***

- 1, Caching
- 2, Session Control
- 3, Log
- 4, Rate Limiting
- 5, and so on……..

## **1.1 基本要求（Basic Requirements）**

### **1.1.1 Follow the Restful APIs defined in Assignment 1.**

定义并编写一套API（可为Json或Open API doc）。

此要求已实现，详见[第四部分API与功能详细设计](#)。

### **1.1.2 Implement the API in Java with Spring WebFlux API.**

此要求已实现，详见[第四部分API与功能详细设计](#)。

WebFlux API 是一种基于 Java 的 Web 应用程序框架，它采用了反应式编程的理念，旨在简化 Web 应用程序的开发。Spring WebFlux 提供了一组可重用的组件和工具，包括反应式路由、处理器、数据绑

定和表单标签库等，使得开发人员可以更加高效地构建可扩展、可维护的 Web 应用程序。

本项目基于Spring Boot框架构建，Spring Boot 是一个基于 Spring 框架的快速开发框架，它简化了 Spring 应用的初始搭建和开发过程。Spring Boot 提供了一套默认的配置，减少了开发人员对配置文件的需求，同时也提供了各种插件和工具，使得开发、部署和监控 Spring 应用变得更加容易。通过 Spring Boot，开发者可以更专注于业务逻辑的实现，而不必过多关注框架本身的配置和管理。

Spring WebFlux 支持完全非阻塞的反应式编程模型，能够处理大量并发请求，适用于高吞吐量和低延迟的应用场景。它基于 Project Reactor 实现，提供了丰富的 API 用于创建和操作异步数据流。通过 Spring WebFlux，开发者可以构建响应式的 RESTful 服务和实时数据流应用，为用户提供更好的性能和体验。

### 1.1.3 Implement the reactive service layer and dao layer.

此要求已实现，详见[第三部分数据持久层设计](#)。

服务层包含业务逻辑，并与DAO层交互以执行操作。在反应式服务层中，我们返回 `Mono` 或 `Flux` 对象，分别表示单个或多个异步结果。

本项目使用的关系型数据库为MySQL，同时使用Mybatis-plus实现对数据库之间的连接。

MySQL是一种开源的关系型数据库管理系统，广泛用于Web应用程序的开发和数据存储。它具有良好的性能、稳定性和可靠性，支持标准的SQL语言，同时也提供了丰富的特性和功能，如事务支持、索引、视图、存储过程等。

MyBatis-Plus是一个基于MyBatis的增强工具，提供了许多便捷的功能来简化对数据库的操作。它可以与Spring框架无缝集成，提供了简单而强大的CRUD（增删改查）操作，支持通过注解或XML配置SQL语句，还提供了分页、条件构造器、逻辑删除、乐观锁等功能。MyBatis-Plus简化了开发者对数据库的操作，提高了开发效率，同时也降低了出错的可能性，是开发基于MyBatis的应用程序的利器之一。

### 1.1.4 Testing with boot testing framework and WebClient/WebTestClient API.

此要求已实现，详见[第五部分测试](#)。

### 1.1.5 API鉴权（API authentication） API authentication, using spring security and JWT will be given credit.

此要求已实现。本项目实现了API的鉴权，使用Spring Security用于提供身份验证和授权功能，可以帮助我们实现各种精细化的安全控制。

具体来说，使用Spring Security针对用户登录生成token，当用户访问其他接口时，需要对用户的token进行鉴权，如果用户可以访问当前接口，那么鉴权成功，如果用户没有权限访问当前接口，那么鉴权失败。同时，使用Spring Security实现对不同接口的访问规则，从鉴权角度可以分为接口本身是否需要鉴权，从用户角度可以分为用户的身份是否可以使用。

详细实现节选如下：

```

1 package com.java.warehousemanagementsystem.config.filter;
2
3
4 import com.java.warehousemanagementsystem.config.SecurityUserDetails;
5 import com.java.warehousemanagementsystem.service.SecurityUserDetailsService;
6 import com.java.warehousemanagementsystem.utils.JwtUtils;
7 import io.micrometer.common.util.StringUtils;
8 import jakarta.annotation.Resource;
9 import jakarta.servlet.FilterChain;
10 import jakarta.servlet.ServletException;
11 import jakarta.servlet.http.HttpServletRequest;
12 import jakarta.servlet.http.HttpServletResponse;
13 import
    org.springframework.security.authentication.UsernamePasswordAuthenticationToken
    ;
14 import org.springframework.security.core.context.SecurityContextHolder;
15 import
    org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
16 import org.springframework.stereotype.Component;
17 import org.springframework.web.filter.OncePerRequestFilter;
18
19 import java.io.IOException;
20
21
22 @Component
23 public class MyAuthenticationFilter extends OncePerRequestFilter {
24
25     @Resource
26     private SecurityUserDetailsService securityUserDetailsService;
27
28     @Override
29     protected void doFilterInternal(HttpServletRequest request,
30                                     HttpServletResponse response,
31                                     FilterChain filterChain) throws
    ServletException, IOException {
32         String requestToken =
    request.getHeader(JwtUtils.getCurrentConfig().getHeader());
33         // 读取请求头中的token
34         if (StringUtils.isNotBlank(requestToken)) {
35             // 判断token是否有效
36             boolean verifyToken = JwtUtils.isValidToken(requestToken);
37             if (!verifyToken) {
38                 filterChain.doFilter(request, response);
39             }
40
41             // 解析token中的用户信息
42             String subject = JwtUtils.getSubject(requestToken);

```

```

43         if (StringUtils.isNotBlank(subject) &&
SecurityContextHolder.getContext().getAuthentication() == null) {
44
45             SecurityUserDetails userDetails = (SecurityUserDetails)
securityUserDetailsService.loadUserByUsername(subject);
46             // 保存用户信息到当前会话
47             UsernamePasswordAuthenticationToken authentication =
48                 new UsernamePasswordAuthenticationToken(
49                     userDetails,
50                     null,
51                     userDetails.getAuthorities());
52             // 将authentication填充到安全上下文
53             authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
54
SecurityContextHolder.getContext().setAuthentication(authentication);
55         }
56     }
57     filterChain.doFilter(request, response);
58 }
59 }

```

## 1.1.6 Using functional api instead of annotation will be given credit.

## 1.2 额外实现（Credit Implementations）

### 1.2.1 缓存（Caching）

本项目实现了缓存，即在用户访问相关接口时，系统首先检查是否存在缓存数据。如果缓存中已有请求的数据，则直接从缓存中获取并返回给用户，避免了对数据库的直接查询，从而减少了数据库的负载并提高了系统的响应速度。如果缓存中没有用户请求的数据，系统则会执行正常的数据库查询，将查询结果不仅返回给用户，同时也保存到缓存中，以供后续相同请求使用。

缓存数据库采用Redis数据库。Redis 是一个开源的高性能键值对数据库，常用作数据结构服务器。它支持多种类型的数据结构，如字符串（strings）、列表（lists）、集合（sets）、索引半径查询等。通过将不同接口的请求信息序列化存入Redis数据库实现高效的缓存。

在Springboot中，采用建立切面的方式可以在接口方法执行前后进行一些操作，比如在方法执行前检查缓存中是否存在数据，如果存在则直接返回缓存数据，同样可以将缓存逻辑与业务逻辑分离，提高了代码的可维护性和可读性。

详细实现节选如下：

```

1 @Aspect
2 @Component

```

```

3 public class CacheLoggingAspect {
4
5     private static final Logger logger =
        LoggerFactory.getLogger(CacheLoggingAspect.class);
6
7     @Around("@annotation(org.springframework.cache.annotation.Cacheable)")
8     public Object cacheableAdvice(ProceedingJoinPoint joinPoint) throws
        Throwable {
9         MethodSignature signature = (MethodSignature) joinPoint.getSignature();
10        String methodName = signature.getMethod().getName();
11        Object[] args = joinPoint.getArgs();
12        String key = args.length > 0 ? args[0].toString() : "No Key";
13
14        logger.info("Attempting to access cache for method: {}, key: {}",
            methodName, key);
15        Object result = null;
16        try {
17            result = joinPoint.proceed();
18            if (result != null) {
19                logger.info("Cache hit for method: {}, key: {}", methodName,
                key);
20            } else {
21                logger.info("Cache miss for method: {}, key: {}", methodName,
                key);
22            }
23        } catch (Throwable t) {
24            logger.error("Error accessing cache for method: {}, key: {}",
                methodName, key);
25            throw t;
26        }
27        return result;
28    }
29
30    @Around("@annotation(org.springframework.cache.annotation.CachePut) ||
        @annotation(org.springframework.cache.annotation.CacheEvict)")
31    public Object cachePutEvictAdvice(ProceedingJoinPoint joinPoint) throws
        Throwable {
32        MethodSignature signature = (MethodSignature) joinPoint.getSignature();
33        String methodName = signature.getMethod().getName();
34        Object[] args = joinPoint.getArgs();
35        String key = args.length > 0 ? args[0].toString() : "No Key";
36        Object result = joinPoint.proceed();
37
38        if
            (signature.getMethod().isAnnotationPresent(org.springframework.cache.annotation
                .CachePut.class)) {

```

```

39         logger.info("Cache updated for method: {}, key: {}", methodName,
key);
40     } else if
(signature.getMethod().isAnnotationPresent(org.springframework.cache.annotation
.CacheEvict.class)) {
41         logger.info("Cache evicted for method: {}, key: {}", methodName,
key);
42     }
43
44     return result;
45 }
46 }

```

### 1.2.2 会话控制（Session Control）

本项目实现了对用户登录状态的会话控制。会话控制是指管理用户在服务器上的会话状态的各种方式。这通常涉及到用户登录后的身份验证状态和用户的特定数据。在本项目中，我们使用JWT Token实现会话管理，当Token有效期失效后，用户需要重新登录以获得最新的Token。同时，当用户主动退出登录后，Token会立即失效。

在Token的生成机制中，我们加入了用户的version id。通过对用户登入登出version id的变化，实现对用户的会话控制。

详细实现节选如下：

```

1  package com.java.warehousemanagementsystem.utils;
2
3
4  import com.auth0.jwt.JWT;
5  import com.auth0.jwt.JWTVerifier;
6  import com.auth0.jwt.algorithms.Algorithm;
7  import com.auth0.jwt.exceptions.JWTVerificationException;
8  import com.auth0.jwt.interfaces.DecodedJWT;
9  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
10 import com.java.warehousemanagementsystem.aspect.CacheLoggingAspect;
11 import com.java.warehousemanagementsystem.mapper.UserMapper;
12 import com.java.warehousemanagementsystem.pojo.User;
13 import lombok.Data;
14 import org.apache.commons.lang3.StringUtils;
15 import org.slf4j.Logger;
16 import org.slf4j.LoggerFactory;
17 import org.springframework.util.CollectionUtils;
18
19 import java.util.Collections;
20 import java.util.Date;
21 import java.util.List;

```

```

22
23 /**
24  * JWT工具类
25  */
26 public class JwtUtils {
27
28     private static final String VERSION_CLAIM = "version";
29     private static final Logger logger =
LoggerFactory.getLogger(CacheLoggingAspect.class);
30
31     /**
32     * 默认JWT标签头
33     */
34     public static final String HEADER = "Authorization";
35
36     /**
37     * JWT配置信息
38     */
39     private static JwtConfig jwtConfig;
40
41     private JwtUtils() {
42
43     }
44
45     /**
46     * 初始化参数
47     *
48     * @param header          JWT标签头
49     * @param tokenHead       Token头
50     * @param issuer          签发者
51     * @param secretKey       密钥 最小长度: 4
52     * @param expirationTime  Token过期时间 单位: 秒
53     * @param issuers         签发者列表 校验签发者时使用
54     * @param audience        接受者
55     */
56     public static void initialize(String header, String tokenHead, String
issuer, String secretKey, long expirationTime, List<String> issuers, String
audience) {
57
58         jwtConfig = new JwtConfig();
59         jwtConfig.setHeader(StringUtils.isNotBlank(header) ? header : HEADER);
60         jwtConfig.setTokenHead(tokenHead);
61         jwtConfig.setIssuer(issuer);
62         jwtConfig.setSecretKey(secretKey);
63         jwtConfig.setExpirationTime(expirationTime);
64         if (CollectionUtils.isEmpty(issuers)) {
65             issuers = Collections.singletonList(issuer);
66         }
67         jwtConfig.setIssuers(issuers);
68         jwtConfig.setAudience(audience);
69         jwtConfig.setAlgorithm(Algorithm.HMAC256(jwtConfig.getSecretKey()));

```



```

66     }
67
68     /**
69      * 初始化参数
70      */
71     public static void initialize(String header, String issuer, String
secretKey, long expirationTime) {
72         initialize(header, null, issuer, secretKey, expirationTime, null,
null);
73     }
74
75     /**
76      * 初始化参数
77      */
78     public static void initialize(String header, String tokenHead, String
issuer, String secretKey, long expirationTime) {
79         initialize(header, tokenHead, issuer, secretKey, expirationTime, null,
null);
80     }
81
82
83     /**
84      * 生成 Token
85      *
86      * @param subject 主题
87      * @return Token
88      */
89     public static String generateToken(String subject, Integer versionId) {
90         return generateToken(subject, versionId,
jwtConfig.getExpirationTime());
91     }
92
93     /**
94      * 生成 Token
95      *
96      * @param subject 主题
97      * @param expirationTime 过期时间
98      * @return Token
99      */
100    public static String generateToken(String subject, Integer versionId, long
expirationTime) {
101        Date now = new Date();
102        Date expiration = new Date(now.getTime() + expirationTime * 1000);
103
104        return JWT.create()
105            .withClaim(VERSION_CLAIM, versionId)
106            .withSubject(subject)

```

```

107         .withIssuer(jwtConfig.getIssuer())
108         .withAudience(jwtConfig.getAudience())
109         .withIssuedAt(now)
110         .withExpiresAt(expiration)
111         .sign(jwtConfig.getAlgorithm());
112     }
113
114     /**
115      * 获取Token数据体
116      */
117     public static String getTokenContent(String token) {
118         if (StringUtils.isNotBlank(jwtConfig.getTokenHead())) {
119             token = token.substring(jwtConfig.getTokenHead().length()).trim();
120         }
121         return token;
122     }
123
124     /**
125      * 验证 Token
126      *
127      * @param token token
128      * @return 验证通过返回true, 否则返回false
129      */
130     public static boolean isValidToken(String token) {
131         try {
132             String subject = getSubject(token);
133             token = getTokenContent(token);
134             QueryWrapper<User> queryWrapper = new QueryWrapper<>();
135             queryWrapper.eq("username", subject);
136             UserMapper userMapper =
ContextUtils.getApplicationContext().getBean(UserMapper.class);
137             User user = userMapper.selectOne(queryWrapper);
138             DecodedJWT jwt = JWT.decode(token);
139             int version = jwt.getClaim(VERSION_CLAIM).asInt();
140             System.out.println(version);
141             System.out.println(subject);
142             logger.info("version: " + version);
143             logger.info("user version: " + user.getVersion());
144
145
146             if (version != user.getVersion()) return false;
147
148             Algorithm algorithm = Algorithm.HMAC256(jwtConfig.getSecretKey());
149             JWTVerifier verifier = JWT.require(algorithm).build();
150             verifier.verify(token);
151             return true;
152         } catch (JWTVerificationException exception) {

```

```
153         // Token验证失败
154         return false;
155     }
156 }
157
158 /**
159  * 判断Token是否过期
160  *
161  * @param token token
162  * @return 过期返回true, 否则返回false
163  */
164 public static boolean isTokenExpired(String token) {
165     try {
166         token = getTokenContent(token);
167         Algorithm algorithm = Algorithm.HMAC256(jwtConfig.secretKey);
168         JWTVerifier verifier = JWT.require(algorithm).build();
169         verifier.verify(token);
170
171         Date expirationDate = JWT.decode(token).getExpiresAt();
172         return expirationDate != null && expirationDate.before(new Date());
173     } catch (JWTVerificationException exception) {
174         // Token验证失败
175         return false;
176     }
177 }
178
179 /**
180  * 获取 Token 中的主题
181  *
182  * @param token token
183  * @return 主题
184  */
185 public static String getSubject(String token) {
186     token = getTokenContent(token);
187     return JWT.decode(token).getSubject();
188 }
189
190 /**
191  * 获取当前Jwt配置信息
192  */
193 public static JwtConfig getCurrentConfig() {
194     return jwtConfig;
195 }
196
197 @Data
198 public static class JwtConfig {
199     /**
```

```

200      * JwtToken Header标签
201      */
202     private String header;
203     /**
204      * Token头
205      */
206     private String tokenHead;
207     /**
208      * 签发者
209      */
210     private String issuer;
211     /**
212      * 密钥
213      */
214     private String secretKey;
215     /**
216      * Token 过期时间
217      */
218     private long expirationTime;
219     /**
220      * 签发者列表
221      */
222     private List<String> issuers;
223     /**
224      * 接受者
225      */
226     private String audience;
227     /**
228      * 加密算法
229      */
230     private Algorithm algorithm;
231 }
232 }

```

### 1.2.3 日志 (Log)

本项目实现了日志的记录与管理，可以帮助开发人员追踪应用程序的运行状态、排查问题、分析性能等。

在本项目中，我们使用了SLF4J、logback的方式实现日志的记录，实现了日志切片，在方法执行前后记录日志，以实现日志的统一管理和格式化输出。比如我们会记录当前执行的接口的入参、出参等信息，方便后续进行问题的排查。

详细实现节选如下：

```

2 @Component
3 public class LoggingAspect {
4     @Around("@annotation(org.springframework.web.bind.annotation.GetMapping)
5         || " +
6             "@annotation(org.springframework.web.bind.annotation.PostMapping)
7         || " +
8             "@annotation(org.springframework.web.bind.annotation.PutMapping)
9         || " +
10            "@annotation(org.springframework.web.bind.annotation.DeleteMapping)")
11     public Object logMethod(ProceedingJoinPoint joinPoint) throws Throwable {
12         ObjectMapper objectMapper = new ObjectMapper();
13         objectMapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS,
14             false);
15         objectMapper.setSerializationInclusion(JsonInclude.Include.NON_NULL);
16
17         Logger logger =
18             LoggerFactory.getLogger(joinPoint.getTarget().getClass());
19
20         // 只记录基础信息和简单类型的参数
21         Object[] args = joinPoint.getArgs();
22         if (args != null) {
23             for (Object arg : args) {
24                 if (arg instanceof Serializable) { // 确保只记录可序列化的简单对象
25                     logger.info("Request argument: {}",
26                         objectMapper.writeValueAsString(arg));
27                 }
28             }
29         }
30
31         Object result = joinPoint.proceed(); // 调用原方法
32
33         if (result != null && result instanceof Serializable) {
34             logger.info("Response: {}",
35                 objectMapper.writeValueAsString(result));
36         }
37
38         return result;
39     }
40 }

```

## 1.2.4 接口限流（Rate Limiting）

本项目实现了对接口的限流，旨在控制访问频率以防止服务被过度使用，保障系统稳定性并优化用户体验。限流是通过对接口请求进行计数和分析，确保在给定的时间窗口内，访问次数不超过预设的阈值。

在本项目中，我们采取了固定速率的令牌桶算法，令牌桶算法是通过一个填充令牌的桶来控制数据的流入流量。桶每隔一定时间生成一定数量的令牌，请求必须消耗一定数量的令牌才能被处理。如果桶中的令牌不足，请求就直接被拒绝。令牌桶的令牌数量存储于Redis数据库中，保证存取的速度与原子性。

同时，项目在用户的维度上进行限流，即每一个用户在一分钟之内只能请求15次，超过次数的请求会被拒绝。用户令牌的数量key采用当前登录所使用的token。

详细实现节选如下：

```
1 @Component
2 public class RateLimitInterceptor implements HandlerInterceptor {
3
4     @Autowired
5     private StringRedisTemplate redisTemplate;
6
7     @Override
8     public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) {
9         if (!(handler instanceof HandlerMethod)) {
10             // 如果不是HandlerMethod实例，直接放行
11             return true;
12         }
13
14         HandlerMethod handlerMethod = (HandlerMethod) handler;
15         Method method = handlerMethod.getMethod();
16         RateLimit rateLimitByToken = method.getAnnotation(RateLimit.class);
17         if (rateLimitByToken == null) return true;
18
19         String bearerToken = extractBearerToken(request);
20         if (bearerToken == null || bearerToken.isEmpty()) throw new
RuntimeException("Bearer token is required");
21
22         String key = "rate_limit:" + method.getDeclaringClass().getName() + ":"
+ method.getName() + ":" + bearerToken;
23         Long currentCount = redisTemplate.opsForValue().increment(key);
24         if (currentCount == 1) redisTemplate.expire(key,
rateLimitByToken.timeout(), TimeUnit.SECONDS);
25
26         if (currentCount > rateLimitByToken.limit()) {
27             response.setContentType("application/xml;charset=UTF-8");
28             String str = "请求次数过多！请稍后再试！";
29             try {
30                 response.getWriter().write(str);
31             } catch (IOException e) {
32                 throw new RuntimeException(e);
33             }
34         }
35     }
36 }
```

```
33         }
34         return false;
35     }
36
37     return true;
38 }
39
40 private String extractBearerToken(HttpServletRequest request) {
41     String authorizationHeader = request.getHeader("Authorization");
42     if (authorizationHeader != null &&
        authorizationHeader.startsWith("Bearer ")) {
43         return authorizationHeader.substring(7);
44     }
45     return null;
46 }
47 }
```

## 二、需求分析与功能设计

### 2.1 需求分析

#### 2.1.1 用户管理

系统需要支持用户管理功能，包括管理员和普通用户。管理员可以管理用户信息，如添加用户、删除用户、更新用户信息等操作。普通用户可以登录系统，并根据权限进行相应的操作。

#### 2.1.2 仓库管理

系统需要支持仓库管理功能，包括添加仓库、删除仓库、更新仓库信息等操作。每个仓库都有自己的位置、容量等信息，管理员可以对这些信息进行管理。

#### 2.1.3 物品管理

系统需要支持物品管理功能，包括添加物品、删除物品、更新物品信息等操作。每个物品都有自己的名称、描述、数量、价格等信息。管理员可以对这些信息进行管理，并在仓库管理中关联物品与仓库的信息，以便于订单管理时使用。

#### 2.1.4 订单管理

系统需要支持订单管理功能，包括创建订单、删除订单、更新订单信息等操作。每个订单都包括商品信息、客户信息、订单状态等内容。同时用户可以往订单内添加不同仓库的物品，订单总价、时间等信息会自动更新。

### 2.2 功能设计

下面简述当前第一阶段系统实现的功能逻辑。

用户注册，用户登录，新增仓库，新增物品，新增订单，向订单添加物品，登出。

## 三、数据持久层设计

### 3.1 MySQL数据库设计

#### 3.1.1 user

用户表，用于存储系统中的用户信息，包括管理员和普通用户。每个用户具有唯一的ID作为主键，用户名和密码用于登录系统。版本号用于乐观锁控制并发访问。

字段：

- id：整数型，自增，主键，用于唯一标识用户。
- username：文本型，存储用户的用户名。
- password：文本型，存储用户的密码。
- version：整数型，表示当前版本，主要用于token鉴权。

#### 3.1.2 warehouse

仓库表，用于存储系统中的仓库信息，包括仓库的名称、地址、管理员、描述和创建时间。

字段：

- id：整数型，自增，主键，用于唯一标识仓库。
- name：文本型，存储仓库的名称。
- address：文本型，存储仓库的地址。
- manager：文本型，存储仓库的管理员。
- description：文本型，存储仓库的描述信息。
- create\_time：日期型，存储仓库的创建时间。

#### 3.1.3 item

物品表，用于存储系统中的物品信息，包括物品的名称、描述、数量、价格、所属仓库ID、创建时间和更新时间。

字段：

- id：整数型，自增，主键，用于唯一标识物品。
- name：文本型，存储物品的名称。
- description：文本型，存储物品的描述信息。



- quantity: 文本型, 存储物品的数量。
- price: 双精度浮点型, 存储物品的价格。
- warehouseId: 整数型, 外键, 关联到仓库表的主键, 表示物品所属的仓库。
- createTime: 日期型, 存储物品的创建时间。
- updateTime: 日期型, 存储物品的更新时间。

### 3.1.4 orders

订单表, 用于存储系统中的订单信息, 包括订单的ID、用户名、状态、总价、地址、创建时间和更新时间。

字段:

- id: 整数型, 自增, 主键, 用于唯一标识订单。
- username: 文本型, 存储下单用户的用户名。
- status: 文本型, 存储订单的状态, 如待支付、已支付、已发货等。
- total\_price: 双精度浮点型, 存储订单的总价。
- address: 文本型, 存储订单的配送地址。
- create\_time: 日期型, 存储订单的创建时间。
- update\_time: 日期型, 存储订单的更新时间。

### 3.1.4 order\_item

订单物品关联表, 用于存储订单和物品之间的关联关系, 包括关联ID、订单ID、物品ID、物品数量和物品总价。

字段:

- id: 整数型, 自增, 主键, 用于唯一标识关联关系。
- order\_id: 整数型, 外键, 关联到订单表的主键, 表示订单ID。
- item\_id: 整数型, 外键, 关联到物品表的主键, 表示物品ID。
- item\_quantity: 整数型, 表示订单中该物品的数量。
- item\_total\_price: 双精度浮点型, 表示订单中该物品的总价。

SQL语句如下 (由IDEA导出) :

```
1 create table if not exists item
2 (
3     id            int auto_increment
4     primary key,
5     name          text null,
```

```
6      description text    null,
7      quantity    text    null,
8      price        double null,
9      warehouseId int     null,
10     createTime   date    null,
11     updateTime   date    null
12 );
13
14 create table if not exists `order`
15 (
16     id            int auto_increment
17     primary key,
18     username      text    null,
19     status        text    null,
20     total_price   double null,
21     address       text    null,
22     create_time   date    null,
23     update_time   date    null
24 );
25
26 create table if not exists order_item
27 (
28     id            int auto_increment
29     primary key,
30     order_id      int     null,
31     item_id       int     null,
32     item_quantity int     null,
33     item_total_price double null
34 );
35
36 create table if not exists user
37 (
38     id            int auto_increment
39     primary key,
40     username      text    null,
41     password      text    null,
42     version       int     null
43 );
44
45 create table if not exists warehouse
46 (
47     id            int auto_increment
48     primary key,
49     name          text    null,
50     address       text    null,
51     manager       text    null,
52     description   text    null,
```

```
53     create_time date null
54 );
```

## 3.2 Mapper与MyBatis-Plus

### Mybatis-plus特点：

- 1、无侵入：Mybatis-Plus 在 Mybatis 的基础上进行扩展，只做增强不做改变，引入 Mybatis-Plus 不会对您现有的 Mybatis 构架产生任何影响，而且 MP 支持所有 Mybatis 原生的特性
- 2、依赖少：仅仅依赖 Mybatis 以及 Mybatis-Spring
- 3、损耗小：启动即会自动注入基本CRUD，性能基本无损耗，直接面向对象操作
- 4、通用CRUD操作：内置通用 Mapper、通用 Service，仅仅通过少量配置即可实现单表大部分 CRUD 操作，更有强大的条件构造器，满足各类使用需求
- 5、多种主键策略：支持多达4种主键策略（内含分布式唯一ID生成器），可自由配置，完美解决主键问题
- 6、支持ActiveRecord：支持 ActiveRecord 形式调用，实体类只需继承 Model 类即可实现基本 CRUD 操作
- 7、支持代码生成，支持自定义全局通用操作：支持全局通用方法注入( Write once, use anywhere )
- 8、内置分页插件：基于Mybatis物理分页，开发者无需关心具体操作，配置好插件之后，写分页等同写基本List查询
- 9、内置性能分析插件：可输出Sql语句以及其执行时间，建议开发测试时启用该功能，能有效解决慢查询
- 10、内置全局拦截插件：提供全表 delete 、 update 操作智能分析阻断，预防误操作

下面以items为例，mapper包下应用MyBatis-Plus的实现如下：

```
1 package com.java.warehousemanagementsystem.mapper;
2
3 import com.baomidou.mybatisplus.core.mapper.BaseMapper;
4 import com.java.warehousemanagementsystem.pojo.Item;
5 import org.apache.ibatis.annotations.Mapper;
6
7 @Mapper
8 public interface ItemMapper extends BaseMapper<Item> {
9 }
```

## 四、API与功能详细设计

## 4.1 用户API

### 1. 用户注册：

- 请求类型：POST
- 路径： `/user`
- 参数：用户名、密码、确认密码
- 功能：调用 `userService.register()` 方法进行用户注册，注册成功返回成功信息，失败返回失败信息。

### 2. 更新用户数据：

- 请求类型：PUT
- 路径： `/user`
- 参数：用户id、用户名、密码、确认密码
- 功能：调用 `userService.updateUser()` 方法更新用户数据，更新成功返回成功信息，失败返回失败信息。

### 3. 根据id查找用户：

- 请求类型：GET
- 路径： `/user/{id}`
- 参数：用户id
- 功能：调用 `userService.findUserById()` 方法根据id查找用户，找到用户返回用户信息，未找到返回失败信息。

### 4. 获取用户列表：

- 请求类型：GET
- 路径： `/user`
- 参数：无
- 功能：调用 `userService.findAllUser()` 方法获取所有用户列表，返回用户列表信息。

### 5. 删除用户：

- 请求类型：DELETE
- 路径： `/user/{id}`
- 参数：用户id
- 功能：调用 `userService.deleteUser()` 方法删除用户，删除成功返回成功信息，失败返回失败信息。

详细实现代码如下：

```
1 package com.java.warehousemanagementsystem.controller;
2
3 import com.java.warehousemanagementsystem.pojo.User;
4 import com.java.warehousemanagementsystem.service.UserService;
5 import com.java.warehousemanagementsystem.vo.ResponseResult;
6 import io.swagger.v3.oas.annotations.Operation;
7 import io.swagger.v3.oas.annotations.Parameter;
8 import io.swagger.v3.oas.annotations.responses.ApiResponse;
9 import io.swagger.v3.oas.annotations.tags.Tag;
10 import org.slf4j.Logger;
11 import org.slf4j.LoggerFactory;
12 import org.springframework.web.bind.annotation.*;
13 import reactor.core.publisher.Flux;
14 import reactor.core.publisher.Mono;
15
16 @Tag(name = "用户管理", description = "用户管理的相关操作")
17 @RestController
18 @RequestMapping("/user")
19 public class UserController {
20     private static final Logger logger =
21         LoggerFactory.getLogger(UserController.class);
22
23     private final UserService userService;
24
25     public UserController(UserService userService) {
26         this.userService = userService;
27     }
28
29     @Operation(summary = "用户注册")
30     @PostMapping()
31     @ApiResponse(responseCode = "200", description = "注册成功")
32     @ApiResponse(responseCode = "400", description = "注册失败")
33     @ResponseBody
34     public Mono<ResponseResult<String>> register(
35         @RequestParam @Parameter(description = "用户名") String username,
36         @RequestParam @Parameter(description = "密码") String password,
37         @RequestParam @Parameter(description = "确认密码") String
38         confirmedPassword) {
39         return userService.register(username, password, confirmedPassword)
40             .map(success -> success ? ResponseResult.success("用户注册成功")
41 : ResponseResult.failure(400, "用户注册失败"));
42     }
43
44     @Operation(summary = "更新用户数据")
45     @PutMapping()
46     @ApiResponse(responseCode = "200", description = "用户数据更新成功")
```

```

44     @ApiResponse(responseCode = "400", description = "用户数据更新失败")
45     @ResponseBody
46     public Mono<ResponseResult<String>> updateUser(@RequestBody User user) {
47         return userService.updateUser(user)
48             .map(success -> success ? ResponseResult.success("用户数据更新成功") : ResponseResult.failure(400, "用户数据更新失败"));
49     }
50
51     @Operation(summary = "根据ID查找用户")
52     @GetMapping("/{id}")
53     @ApiResponse(responseCode = "200", description = "用户查找成功")
54     @ApiResponse(responseCode = "400", description = "用户查找失败")
55     @ResponseBody
56     public Mono<ResponseResult<User>> getUserById(@PathVariable Integer id) {
57         return userService.findUserById(id)
58             .map(ResponseResult::success);
59     }
60
61     @Operation(summary = "获取所有用户")
62     @GetMapping()
63     @ApiResponse(responseCode = "200", description = "用户列表获取成功")
64     @ApiResponse(responseCode = "400", description = "用户列表获取失败")
65     @ResponseBody
66     public Flux<User> getAllUsers() {
67         return userService.findAllUser();
68     }
69
70     @Operation(summary = "删除用户")
71     @DeleteMapping("/{id}")
72     @ApiResponse(responseCode = "200", description = "用户删除成功")
73     @ApiResponse(responseCode = "400", description = "用户删除失败")
74     @ResponseBody
75     public Mono<ResponseResult<String>> deleteUser(@PathVariable Integer id) {
76         return userService.deleteUser(id)
77             .map(success -> success ? ResponseResult.success("用户删除成功")
78                 : ResponseResult.failure(400, "用户删除失败"));
79     }

```

## 4.2 仓库API

### 1. 创建新仓库：

- 请求类型：POST
- 路径： `/warehouse`

- 参数：仓库名称、仓库位置、管理员、仓库介绍
  - 功能：调用 `warehouseService.addWarehouse()` 方法创建新的仓库，创建成功返回成功信息和创建的仓库对象。
2. 更新仓库信息：
- 请求类型：PUT
  - 路径： `/warehouse/{id}`
  - 参数：仓库id、仓库名称、仓库位置、管理员、仓库介绍
  - 功能：调用 `warehouseService.updateWarehouse()` 方法更新指定id的仓库信息，更新成功返回成功信息和更新后的仓库对象。
3. 根据id查找仓库：
- 请求类型：GET
  - 路径： `/warehouse/{id}`
  - 参数：仓库id
  - 功能：调用 `warehouseService.selectWarehouse()` 方法根据id查找仓库，找到仓库返回成功信息和仓库对象，未找到返回失败信息。
4. 获取仓库列表：
- 请求类型：GET
  - 路径： `/warehouse`
  - 参数：仓库名称、页码、每页数量
  - 功能：调用 `warehouseService.selectWarehouse()` 方法获取仓库列表，根据名称模糊查询，返回指定页数和每页数量的数据。
5. 删除仓库：
- 请求类型：DELETE
  - 路径： `/warehouse/{id}`
  - 参数：仓库id
  - 功能：调用 `warehouseService.deleteWarehouse()` 方法删除指定id的仓库，删除成功返回成功信息，失败返回失败信息。

详细实现代码如下：

```
1 package com.java.warehousemanagementsystem.controller;
2
3 import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
4 import com.java.warehousemanagementsystem.pojo.Warehouse;
```

```
5 import com.java.warehousemanagementsystem.service.WarehouseService;
6 import com.java.warehousemanagementsystem.vo.ResponseResult;
7 import io.swagger.v3.oas.annotations.Operation;
8 import io.swagger.v3.oas.annotations.Parameter;
9 import io.swagger.v3.oas.annotations.responses.ApiResponse;
10 import io.swagger.v3.oas.annotations.tags.Tag;
11 import org.slf4j.Logger;
12 import org.slf4j.LoggerFactory;
13 import org.springframework.web.bind.annotation.*;
14 import reactor.core.publisher.Flux;
15 import reactor.core.publisher.Mono;
16
17 @Tag(name = "仓库管理", description = "仓库管理的相关操作")
18 @RestController
19 @RequestMapping("/warehouse")
20 public class WarehouseController {
21     private static final Logger logger =
22         LoggerFactory.getLogger(WarehouseController.class);
23
24     private final WarehouseService warehouseService;
25
26     public WarehouseController(WarehouseService warehouseService) {
27         this.warehouseService = warehouseService;
28     }
29
30     @Operation(summary = "创建新仓库")
31     @PostMapping()
32     @ApiResponse(responseCode = "200", description = "仓库创建成功")
33     @ApiResponse(responseCode = "400", description = "仓库创建失败")
34     @ResponseBody
35     public Mono<ResponseResult<Warehouse>> createWarehouse(
36         @RequestParam @Parameter(description = "仓库名称") String name,
37         @RequestParam @Parameter(description = "仓库位置") String location,
38         @RequestParam @Parameter(description = "管理员") String manager,
39         @RequestParam @Parameter(description = "仓库介绍") String
40         description) {
41         return warehouseService.addWarehouse(name, location, manager,
42             description)
43             .map(ResponseResult::success);
44     }
45
46     @Operation(summary = "更新仓库信息")
47     @PutMapping("/{id}")
48     @ApiResponse(responseCode = "200", description = "仓库更新成功")
49     @ApiResponse(responseCode = "400", description = "仓库更新失败")
50     @ResponseBody
51     public Mono<ResponseResult<Warehouse>> updateWarehouse(
```



```

49         @PathVariable Integer id,
50         @RequestParam @Parameter(description = "仓库名称") String name,
51         @RequestParam @Parameter(description = "仓库位置") String location,
52         @RequestParam @Parameter(description = "管理员") String manager,
53         @RequestParam @Parameter(description = "仓库介绍") String
description) {
54         return warehouseService.updateWarehouse(id, name, location, manager,
description)
55             .map(ResponseResult::success);
56     }
57
58     @Operation(summary = "删除仓库")
59     @DeleteMapping("/{id}")
60     @ApiResponse(responseCode = "200", description = "仓库删除成功")
61     @ApiResponse(responseCode = "400", description = "仓库删除失败")
62     @ResponseBody
63     public Mono<ResponseResult<Void>> deleteWarehouse(@PathVariable Integer id)
{
64         return warehouseService.deleteWarehouse(id)
65             .thenReturn(ResponseResult.success());
66     }
67
68     @Operation(summary = "根据ID查找仓库")
69     @GetMapping("/{id}")
70     @ApiResponse(responseCode = "200", description = "仓库查找成功")
71     @ApiResponse(responseCode = "400", description = "仓库查找失败")
72     @ResponseBody
73     public Mono<ResponseResult<Warehouse>> getWarehouseById(@PathVariable
Integer id) {
74         return warehouseService.selectWarehouseById(id)
75             .map(ResponseResult::success);
76     }
77
78     @Operation(summary = "分页查询仓库")
79     @GetMapping()
80     @ApiResponse(responseCode = "200", description = "仓库查找成功")
81     @ApiResponse(responseCode = "400", description = "仓库查找失败")
82     @ResponseBody
83     public Flux<Warehouse> getWarehouses(
84         @RequestParam @Parameter(description = "仓库名称") String name,
85         @RequestParam @Parameter(description = "页码") Long pageNo,
86         @RequestParam @Parameter(description = "每页数量") Long pageSize) {
87         return warehouseService.selectWarehouse(name, pageNo, pageSize);
88     }
89 }

```

## 4.3 物品API

### 1. 获取物品列表：

- 请求类型：GET
- 路径： `/item`
- 参数：无
- 功能：调用 `itemService.findAllItems()` 方法获取所有物品列表，返回物品列表信息。

### 2. 添加新物品：

- 请求类型：POST
- 路径： `/item`
- 参数：物品名称、物品描述、物品数量、物品价格、仓库ID
- 功能：调用 `itemService.addItem()` 方法添加新物品，添加成功返回成功信息，失败返回失败信息。

### 3. 根据ID获取物品信息：

- 请求类型：GET
- 路径： `/item/{id}`
- 参数：物品ID
- 功能：调用 `itemService.findItemById()` 方法根据物品ID查找物品信息，返回物品信息。

### 4. 更新物品信息：

- 请求类型：PUT
- 路径： `/item/{id}`
- 参数：物品ID、物品名称、物品描述、物品数量、物品价格、仓库ID
- 功能：调用 `itemService.updateItem()` 方法更新指定ID的物品信息，更新成功返回成功信息，失败返回失败信息。

### 5. 删除物品：

- 请求类型：DELETE
- 路径： `/item/{id}`
- 参数：物品ID
- 功能：调用 `itemService.deleteItem()` 方法删除指定ID的物品，删除成功返回成功信息，失败返回失败信息。

详细实现代码节选如下：

```
1 package com.java.warehousemanagementsystem.controller;
2
3 import com.java.warehousemanagementsystem.pojo.Item;
4 import com.java.warehousemanagementsystem.service.ItemService;
5 import com.java.warehousemanagementsystem.vo.ResponseResult;
6 import io.swagger.v3.oas.annotations.Operation;
7 import io.swagger.v3.oas.annotations.Parameter;
8 import io.swagger.v3.oas.annotations.responses.ApiResponse;
9 import io.swagger.v3.oas.annotations.tags.Tag;
10 import org.slf4j.Logger;
11 import org.slf4j.LoggerFactory;
12 import org.springframework.cache.annotation.CacheEvict;
13 import org.springframework.cache.annotation.Cacheable;
14 import org.springframework.web.bind.annotation.*;
15 import reactor.core.publisher.Flux;
16 import reactor.core.publisher.Mono;
17
18 @Tag(name = "物品管理", description = "物品管理的相关操作")
19 @RestController
20 @RequestMapping("/item")
21 public class ItemController {
22     private static final Logger logger =
23         LoggerFactory.getLogger(ItemController.class);
24
25     private final ItemService itemService;
26
27     public ItemController(ItemService itemService) {
28         this.itemService = itemService;
29     }
30
31     @Operation(summary = "获取物品列表")
32     @GetMapping()
33     @ApiResponse(responseCode = "200", description = "成功获取物品列表")
34     @ApiResponse(responseCode = "404", description = "未找到物品")
35     @ApiResponse(responseCode = "400", description = "获取物品列表失败")
36     @ResponseBody
37     @Cacheable(value = "itemList")
38     public Flux<Item> getAllItems() {
39         return itemService.findAllItems();
40     }
41
42     @Operation(summary = "添加新物品")
43     @PostMapping()
44     @ApiResponse(responseCode = "200", description = "物品添加成功")
45     @ApiResponse(responseCode = "400", description = "物品添加失败")
46     @ResponseBody
```

```

46     public Mono<ResponseResult<String>> addItem(
47         @RequestParam @Parameter(description = "物品名称") String name,
48         @RequestParam @Parameter(description = "物品描述") String
description,
49         @RequestParam @Parameter(description = "物品数量") Integer quantity,
50         @RequestParam @Parameter(description = "物品价格") Double price,
51         @RequestParam @Parameter(description = "仓库ID") Integer
warehouseId) {
52         return itemService.addItem(name, description, quantity, price,
warehouseId)
53             .map(success -> success ? ResponseResult.success("物品添加成功")
: ResponseResult.failure(400, "物品添加失败"));
54     }
55
56     @Operation(summary = "更新物品信息")
57     @PutMapping()
58     @ApiResponse(responseCode = "200", description = "物品信息更新成功")
59     @ApiResponse(responseCode = "400", description = "物品信息更新失败")
60     @ResponseBody
61     public Mono<ResponseResult<String>> updateItem(
62         @RequestParam @Parameter(description = "物品ID") Integer id,
63         @RequestParam @Parameter(description = "物品名称") String name,
64         @RequestParam @Parameter(description = "物品描述") String
description,
65         @RequestParam @Parameter(description = "物品数量") Integer quantity,
66         @RequestParam @Parameter(description = "物品价格") Double price,
67         @RequestParam @Parameter(description = "仓库ID") Integer
warehouseId) {
68         return itemService.updateItem(id, name, description, quantity, price,
warehouseId)
69             .map(success -> success ? ResponseResult.success("物品信息更新成
功") : ResponseResult.failure(400, "物品信息更新失败"));
70     }
71
72     @Operation(summary = "根据ID查找物品")
73     @GetMapping("/{id}")
74     @ApiResponse(responseCode = "200", description = "成功找到物品")
75     @ApiResponse(responseCode = "404", description = "未找到物品")
76     @ApiResponse(responseCode = "400", description = "查找物品失败")
77     @ResponseBody
78     public Mono<ResponseResult<Item>> getItemById(@PathVariable Integer id) {
79         return itemService.findItemById(id)
80             .map(ResponseResult::success);
81     }
82
83     @Operation(summary = "删除物品")
84     @DeleteMapping("/{id}")

```

```
85     @ApiResponse(responseCode = "200", description = "成功删除物品")
86     @ApiResponse(responseCode = "400", description = "删除物品失败")
87     @ResponseBody
88     @CacheEvict(value = "itemList", allEntries = true)
89     public Mono<ResponseResult<String>> deleteItem(@PathVariable Integer id) {
90         return itemService.deleteItem(id)
91             .map(success -> success ? ResponseResult.success("成功删除物品")
92               : ResponseResult.failure(400, "删除物品失败"));
93     }
```

## 4.4 订单API

### 1. 添加新订单：

- 请求类型：POST
- 路径： `/order`
- 参数：订单对象
- 功能：调用 `ordersService.addOrder()` 方法添加新订单，添加成功返回成功信息，失败返回失败信息。

### 2. 添加物品：

- 请求类型：POST
- 路径： `/order/item/{id}`
- 参数：订单ID、物品ID
- 功能：调用 `ordersService.addItem()` 方法为指定订单添加物品，添加成功返回成功信息，失败返回失败信息。

### 3. 获取所有订单：

- 请求类型：GET
- 路径： `/order`
- 参数：无
- 功能：调用 `ordersService.findAllOrders()` 方法获取所有订单列表，返回订单列表信息。

### 4. 根据ID获取订单信息及其物品：

- 请求类型：GET
- 路径： `/order/{id}`
- 参数：订单ID

- 功能：调用 `ordersService.findOrderById()` 方法根据订单ID查找订单信息，同时调用 `ordersService.findItemsByOrderId()` 方法查找订单中的物品，返回订单信息及其物品信息。

#### 5. 根据用户ID获取订单信息：

- 请求类型：GET
- 路径： `/order/user/{userId}`
- 参数：用户ID
- 功能：调用 `ordersService.findOrdersByUserId()` 方法根据用户ID查找订单信息，返回订单列表信息。

#### 6. 根据订单状态获取订单信息：

- 请求类型：GET
- 路径： `/order/status/{status}`
- 参数：订单状态
- 功能：调用 `ordersService.findOrdersByStatus()` 方法根据订单状态查找订单信息，返回订单列表信息。

#### 7. 根据地址获取订单信息：

- 请求类型：GET
- 路径： `/order/address/{address}`
- 参数：地址
- 功能：调用 `ordersService.findOrdersByAddress()` 方法根据地址查找订单信息，返回订单列表信息。

#### 8. 更新订单信息：

- 请求类型：PUT
- 路径： `/order/{id}`
- 参数：订单ID、订单对象
- 功能：调用 `ordersService.updateOrder()` 方法更新指定ID的订单信息，更新成功返回成功信息，失败返回失败信息。

#### 9. 删除订单：

- 请求类型：DELETE
- 路径： `/order/{id}`
- 参数：订单ID

- 功能：调用 `ordersService.deleteOrder()` 方法删除指定ID的订单，删除成功返回成功信息，失败返回失败信息。

## 10. 删除订单物品：

- 请求类型：DELETE
- 路径： `/order/item/{id}`
- 参数：订单ID、物品ID
- 功能：调用 `ordersService.deleteItem()` 方法删除指定ID的订单中的指定物品，删除成功返回成功信息，失败返回失败信息。

详细实现代码节选如下：

```
1 package com.java.warehousemanagementsystem.controller;
2
3 import com.java.warehousemanagementsystem.pojo.Item;
4 import com.java.warehousemanagementsystem.pojo.Orders;
5 import com.java.warehousemanagementsystem.service.OrdersService;
6 import com.java.warehousemanagementsystem.vo.ResponseResult;
7 import io.swagger.v3.oas.annotations.Operation;
8 import io.swagger.v3.oas.annotations.Parameter;
9 import io.swagger.v3.oas.annotations.responses.ApiResponse;
10 import io.swagger.v3.oas.annotations.tags.Tag;
11 import org.slf4j.Logger;
12 import org.slf4j.LoggerFactory;
13 import org.springframework.web.bind.annotation.*;
14 import reactor.core.publisher.Flux;
15 import reactor.core.publisher.Mono;
16
17 @Tag(name = "订单管理", description = "订单管理的相关操作")
18 @RestController
19 @RequestMapping("/order")
20 public class OrdersController {
21     private static final Logger logger =
22         LoggerFactory.getLogger(OrdersController.class);
23
24     private final OrdersService ordersService;
25
26     public OrdersController(OrdersService ordersService) {
27         this.ordersService = ordersService;
28     }
29
30     @Operation(summary = "添加新订单")
31     @PostMapping
32     @ApiResponse(responseCode = "200", description = "订单添加成功")
```

```

32     @ApiResponse(responseCode = "400", description = "订单添加失败")
33     @ResponseBody
34     public Mono<ResponseResult<String>> addOrder(
35         @RequestParam @Parameter(description = "用户名") String username,
36         @RequestParam @Parameter(description = "地址") String address) {
37         Orders orders = new Orders();
38         orders.setUsername(username);
39         orders.setAddress(address);
40         return ordersService.addOrder(orders)
41             .map(success -> success ? ResponseResult.success("订单添加成功")
: ResponseResult.failure(400, "订单添加失败"));
42     }
43
44     @Operation(summary = "更新订单")
45     @PutMapping
46     @ApiResponse(responseCode = "200", description = "订单更新成功")
47     @ApiResponse(responseCode = "400", description = "订单更新失败")
48     @ResponseBody
49     public Mono<ResponseResult<String>> updateOrder(@RequestBody Orders orders)
50     {
51         return ordersService.updateOrder(orders)
52             .map(success -> success ? ResponseResult.success("订单更新成功")
: ResponseResult.failure(400, "订单更新失败"));
53     }
54
55     @Operation(summary = "根据ID查找订单")
56     @GetMapping("/{id}")
57     @ApiResponse(responseCode = "200", description = "订单查找成功")
58     @ApiResponse(responseCode = "400", description = "订单查找失败")
59     @ResponseBody
60     public Mono<ResponseResult<Orders>> getOrderById(@PathVariable Integer id)
61     {
62         return ordersService.findOrderById(id)
63             .map(ResponseResult::success);
64     }
65
66     @Operation(summary = "根据状态查找订单")
67     @GetMapping("/status")
68     @ApiResponse(responseCode = "200", description = "订单查找成功")
69     @ApiResponse(responseCode = "400", description = "订单查找失败")
70     @ResponseBody
71     public Flux<Orders> getOrdersByStatus(@RequestParam @Parameter(description
= "订单状态") String status) {
72         return ordersService.findOrdersByStatus(status);
73     }
74
75     @Operation(summary = "根据地址查找订单")

```



```

74     @GetMapping("/address")
75     @ApiResponse(responseCode = "200", description = "订单查找成功")
76     @ApiResponse(responseCode = "400", description = "订单查找失败")
77     @ResponseBody
78     public Flux<Orders> getOrdersByAddress(@RequestParam
79     @Parameter(description = "订单地址") String address) {
80         return ordersService.findOrdersByAddress(address);
81     }
82     @Operation(summary = "获取订单中的商品")
83     @GetMapping("/{id}/items")
84     @ApiResponse(responseCode = "200", description = "订单商品查找成功")
85     @ApiResponse(responseCode = "400", description = "订单商品查找失败")
86     @ResponseBody
87     public Flux<Item> getItemsByOrderId(@PathVariable Integer id) {
88         return ordersService.findItemsByOrderId(id);
89     }
90 }

```

## 4.5 会话管理API

### 1. 用户登录：

- 请求类型：POST
- 路径： `/session`
- 参数：用户名、密码
- 功能：调用 `sessionService.loginSession()` 方法进行用户登录操作，并返回登录结果，包含token信息。

### 2. 用户登出：

- 请求类型：DELETE
- 路径： `/session`
- 参数：用户名
- 功能：调用 `sessionService.logoutSession()` 方法进行用户登出操作，并返回登出结果，使得token失效。

详细实现代码节选如下：

```

1 package com.java.warehousemanagementsystem.controller;
2
3 import com.java.warehousemanagementsystem.service.SessionService;
4 import com.java.warehousemanagementsystem.vo.ResponseResult;

```

```

5  import io.swagger.v3.oas.annotations.Operation;
6  import io.swagger.v3.oas.annotations.Parameter;
7  import io.swagger.v3.oas.annotations.responses.ApiResponse;
8  import io.swagger.v3.oas.annotations.tags.Tag;
9  import org.slf4j.Logger;
10 import org.slf4j.LoggerFactory;
11 import org.springframework.web.bind.annotation.*;
12 import reactor.core.publisher.Mono;
13
14 import java.util.Map;
15
16 @Tag(name = "会话管理", description = "会话管理的相关操作")
17 @RestController
18 @RequestMapping("/session")
19 public class SessionController {
20     private static final Logger logger =
21         LoggerFactory.getLogger(UserController.class);
22     private final SessionService sessionService;
23
24     public SessionController(SessionService sessionService) {
25         this.sessionService = sessionService;
26     }
27
28     @Operation(summary = "用户登录")
29     @ApiResponse(responseCode = "200", description = "登录成功")
30     @ApiResponse(responseCode = "400", description = "登录失败")
31     @ResponseBody
32     @PostMapping("")
33     public Mono<ResponseResult<Map<String, String>>> login(
34         @Parameter(name = "username", description = "用户名") @RequestParam
35         String username,
36         @Parameter(name = "password", description = "密码") @RequestParam
37         String password) {
38         logger.info("(SessionController)用户登录, username = {}, password = {}", username, password);
39         return sessionService.loginSession(username, password)
40             .map(ResponseResult::success);
41     }
42
43     @Operation(summary = "用户登出")
44     @ApiResponse(responseCode = "200", description = "登出成功")
45     @ApiResponse(responseCode = "400", description = "登出失败")
46     @ResponseBody
47     @DeleteMapping("")
48     public Mono<ResponseResult<String>> logout(
49         @Parameter(name = "username", description = "用户名") @RequestParam
50         String username) {

```

```
47         logger.info("(SessionController)用户登出, username = {}", username);
48         return sessionService.logoutSession(username)
49             .map(ResponseResult::success);
50     }
51
52     @Operation(summary = "测试admin")
53     @PostMapping("/test")
54     public String test() {
55         return "admin";
56     }
57 }
```

## 五、测试

### 测试用例规则：

- a) 根据需求文档的变更，实时补充；
- b) 以测试类型为基础，包含正常功能和可靠性（异常处理和恢复等）测试。

### 常规方法：

等价类划分、边界值、因果图等。

### 数据验证要求：

- a) 数据一致性：对数据在不同页面、不同系统间流转的一致性的验证；
- b) 数据同步：设计数据更新，数据库同步方面的测试；
- c) 数据有效性：满足和不满足置顶模块的输入数据的要求的测试；
- d) 兼容性测试：对不同系统浏览器进行兼容测试。

## 5.1 单元测试

对于Spring MVC架构下的关键层Controller和服务Service，我们对其进行了单元测试。单元测试将随后续开发进一步跟进。

### 5.1.1 Controller层（下面以用户部分的为例，包含user和session）

```
1 package com.java.warehousemanagementsystem.controller;
2
3 import com.java.warehousemanagementsystem.config.SecurityConfig;
4 import com.java.warehousemanagementsystem.pojo.User;
5 import com.java.warehousemanagementsystem.service.UserService;
6 import com.java.warehousemanagementsystem.vo.ResponseResult;
7 import org.junit.jupiter.api.BeforeEach;
```

```

8  import org.junit.jupiter.api.Test;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.boot.test.autoconfigure.web.reactive.WebFluxTest;
11 import org.springframework.boot.test.mock.mockito.MockBean;
12 import org.springframework.context.annotation.Import;
13 import org.springframework.http.MediaType;
14 import
    org.springframework.security.test.web.reactive.server.SecurityMockServerConfigu
    rers;
15 import org.springframework.test.web.reactive.server.WebTestClient;
16 import reactor.core.publisher.Flux;
17 import reactor.core.publisher.Mono;
18
19 import java.util.Arrays;
20 import java.util.List;
21 import java.util.Map;
22
23 import static org.mockito.BDDMockito.given;
24 import static
    org.springframework.web.reactive.function.BodyInserters.fromValue;
25
26 @WebFluxTest(UserController.class)
27 @Import(SecurityConfig.class)
28 public class UserControllerTest {
29
30     @Autowired
31     private WebTestClient webTestClient;
32
33     @MockBean
34     private UserService userService;
35
36     @BeforeEach
37     public void setUp() {
38         this.webTestClient =
    this.webTestClient.mutateWith(SecurityMockServerConfigurers.mockUser());
39     }
40
41     @Test
42     void testGetAllUsers() {
43         User user1 = new User(1, "User1", "Password1", 1);
44         User user2 = new User(2, "User2", "Password2", 1);
45         List<User> users = Arrays.asList(user1, user2);
46
47         given(userService.findAllUser()).willReturn(Flux.fromIterable(users));
48
49         webTestClient.get().uri("/users")
50             .accept(MediaType.APPLICATION_JSON)

```

```

51         .exchange()
52         .expectStatus().isOk()
53         .expectBodyList(User.class).isEqualTo(users);
54     }
55
56     @Test
57     void testRegisterUser() {
58         given(userService.register("NewUser", "Password1", "Password1"))
59             .willReturn(Mono.just(true));
60
61         webTestClient.post().uri("/users/register")
62             .contentType(MediaType.APPLICATION_JSON)
63             .body(fromValue(Map.of("username", "NewUser", "password",
64 "Password1", "confirmedPassword", "Password1")))
65             .exchange()
66             .expectStatus().isOk()
67             .expectBody(ResponseResult.class)
68             .value(response -> {
69                 assert response.getCode() == 200;
70                 assert response.getMsg().equals("注册成功");
71             });
72
73     @Test
74     void testUpdateUser() {
75         User updatedUser = new User(1, "UpdatedUser", "UpdatedPassword", 1);
76         given(userService.updateUser(updatedUser)).willReturn(Mono.just(true));
77
78         webTestClient.put().uri("/users")
79             .contentType(MediaType.APPLICATION_JSON)
80             .body(fromValue(updatedUser))
81             .exchange()
82             .expectStatus().isOk()
83             .expectBody(ResponseResult.class)
84             .value(response -> {
85                 assert response.getCode() == 200;
86                 assert response.getMsg().equals("更新用户成功");
87             });
88     }
89
90     @Test
91     void testGetUserById() {
92         User user = new User(1, "User1", "Password1", 1);
93
94         given(userService.findUserById(1)).willReturn(Mono.just(user));
95
96         webTestClient.get().uri("/users/1")

```

```

97         .accept(MediaType.APPLICATION_JSON)
98         .exchange()
99         .expectStatus().isOk()
100        .expectBody(ResponseResult.class)
101        .value(response -> {
102            assert response.getCode() == 200;
103            assert response.getData().equals(user);
104        });
105    }
106
107    @Test
108    void testDeleteUser() {
109        given(userService.deleteUser(1)).willReturn(Mono.just(true));
110
111        webTestClient.delete().uri("/users/1")
112            .exchange()
113            .expectStatus().isOk()
114            .expectBody(ResponseResult.class)
115            .value(response -> {
116                assert response.getCode() == 200;
117                assert response.getMsg().equals("成功删除用户");
118            });
119    }
120 }

```

```

1  package com.java.warehousemanagementsystem.controller;
2
3  import com.java.warehousemanagementsystem.config.SecurityConfig;
4  import com.java.warehousemanagementsystem.service.SessionService;
5  import com.java.warehousemanagementsystem.vo.ResponseResult;
6  import org.junit.jupiter.api.BeforeEach;
7  import org.junit.jupiter.api.Test;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.boot.test.autoconfigure.web.reactive.WebFluxTest;
10 import org.springframework.boot.test.mock.mockito.MockBean;
11 import org.springframework.context.annotation.Import;
12 import org.springframework.http.MediaType;
13 import
    org.springframework.security.test.web.reactive.server.SecurityMockServerConfigu
    rers;
14 import org.springframework.test.web.reactive.server.WebTestClient;
15 import reactor.core.publisher.Mono;
16
17 import java.util.Map;
18

```

```
19 import static org.mockito.BDDMockito.given;
20 import static
    org.springframework.web.reactive.function.BodyInserters.fromValue;
21
22 @WebFluxTest(SessionController.class)
23 @Import(SecurityConfig.class)
24 public class SessionControllerTest {
25
26     @Autowired
27     private WebTestClient webTestClient;
28
29     @MockBean
30     private SessionService sessionService;
31
32     @BeforeEach
33     public void setUp() {
34         this.webTestClient =
35         this.webTestClient.mutateWith(SecurityMockServerConfigurers.mockUser());
36     }
37
38     @Test
39     void testLoginSession() {
40         given(sessionService.loginSession("user1", "password1"))
41             .willReturn(Mono.just(Map.of("token", "123456789")));
42
43         webTestClient.post().uri("/sessions/login")
44             .contentType(MediaType.APPLICATION_JSON)
45             .body(fromValue(Map.of("username", "user1", "password",
46 "password1")))
47             .exchange()
48             .expectStatus().isOk()
49             .expectBody(ResponseResult.class)
50             .value(response -> {
51                 assert response.getStatusCode() == 200;
52                 //assert response.getMessage().equals("Login successful");
53             });
54     }
55
56     @Test
57     void testLogoutSession() {
58         given(sessionService.logoutSession("user1"))
59             .willReturn(Mono.just("Logout successful"));
60
61         webTestClient.post().uri("/sessions/logout")
62             .contentType(MediaType.APPLICATION_JSON)
63             .body(fromValue(Map.of("username", "user1")))
64             .exchange()
```

```

63         .expectStatus().isOk()
64         .expectBody(ResponseResult.class)
65         .value(response -> {
66             assert response.getCode() == 200;
67             //assert response.getMsg().equals("Logout successful");
68         });
69     }
70 }

```

### 5.1.2 Service层（下面以用户部分的为例，包含user和warehouse）

```

1  package com.java.warehousemanagementsystem.service.impl;
2
3  import com.java.warehousemanagementsystem.mapper.UserMapper;
4  import com.java.warehousemanagementsystem.pojo.User;
5  import org.junit.jupiter.api.BeforeEach;
6  import org.junit.jupiter.api.Test;
7  import org.mockito.InjectMocks;
8  import org.mockito.Mock;
9  import org.mockito.MockitoAnnotations;
10 import reactor.core.publisher.Flux;
11 import reactor.core.publisher.Mono;
12 import reactor.test.StepVerifier;
13
14 import java.util.ArrayList;
15 import java.util.List;
16
17 import static org.mockito.ArgumentMatchers.any;
18 import static org.mockito.Mockito.*;
19
20 public class UserServiceImplTest {
21
22     @Mock
23     private UserMapper userMapper;
24
25     @InjectMocks
26     private UserServiceImpl userService;
27
28     @BeforeEach
29     void setUp() {
30         MockitoAnnotations.openMocks(this);
31     }
32
33     @Test
34     void testAddUser() {

```



```

35         User user = new User();
36         when(userMapper.insert(any(User.class))).thenReturn(1);
37
38         Mono<Boolean> result = userService.register("test", "password",
"password");
39         StepVerifier.create(result)
40             .expectNext(true)
41             .verifyComplete();
42     }
43
44     @Test
45     void testUpdateUser() {
46         User user = new User();
47         when(userMapper.updateById(any(User.class))).thenReturn(1);
48
49         Mono<Boolean> result = userService.updateUser(user);
50         StepVerifier.create(result)
51             .expectNext(true)
52             .verifyComplete();
53     }
54
55     @Test
56     void testFindUserById() {
57         User user = new User();
58         user.setId(1);
59         when(userMapper.selectById(1)).thenReturn(user);
60
61         Mono<User> result = userService.findUserById(1);
62         StepVerifier.create(result)
63             .expectNext(user)
64             .verifyComplete();
65     }
66
67     @Test
68     void testFindAllUsers() {
69         List<User> users = new ArrayList<>();
70         User user = new User();
71         users.add(user);
72
73         when(userMapper.selectList(any())).thenReturn(users);
74
75         Flux<User> result = userService.findAllUser();
76         StepVerifier.create(result)
77             .expectNextSequence(users)
78             .verifyComplete();
79     }
80

```

```

81     @Test
82     void testDeleteUser() {
83         when(userMapper.deleteById(1)).thenReturn(1);
84
85         Mono<Boolean> result = userService.deleteUser(1);
86         StepVerifier.create(result)
87             .expectNext(true)
88             .verifyComplete();
89     }
90 }

```

```

1  package com.java.warehousemanagementsystem.service.impl;
2
3  import com.java.warehousemanagementsystem.mapper.WarehouseMapper;
4  import com.java.warehousemanagementsystem.pojo.Warehouse;
5  import org.junit.jupiter.api.BeforeEach;
6  import org.junit.jupiter.api.Test;
7  import org.mockito.InjectMocks;
8  import org.mockito.Mock;
9  import org.mockito.MockitoAnnotations;
10 import reactor.core.publisher.Flux;
11 import reactor.core.publisher.Mono;
12 import reactor.test.StepVerifier;
13
14 import java.util.ArrayList;
15 import java.util.List;
16
17 import static org.mockito.ArgumentMatchers.any;
18 import static org.mockito.Mockito.*;
19
20 public class WarehouseServiceImplTest {
21
22     @Mock
23     private WarehouseMapper warehouseMapper;
24
25     @InjectMocks
26     private WarehouseServiceImpl warehouseService;
27
28     @BeforeEach
29     void setUp() {
30         MockitoAnnotations.openMocks(this);
31     }
32
33     @Test
34     void testAddWarehouse() {

```

```

35     Warehouse warehouse = new Warehouse();
36     when(warehouseMapper.insert(any(Warehouse.class))).thenReturn(1);
37     Mono<Boolean> result = warehouseService.addWarehouse("name",
38     "location", "manager", "description").hasElement();
39     StepVerifier.create(result)
40         .expectNext(true)
41         .verifyComplete();
42
43     @Test
44     void testUpdateWarehouse() {
45         Warehouse warehouse = new Warehouse();
46         when(warehouseMapper.updateById(any(Warehouse.class))).thenReturn(1);
47         // Mono<Warehouse> updateWarehouse(Integer id, String name, String
48         location, String manager, String description);
49         Mono<Warehouse> result = warehouseService.updateWarehouse(1, "name",
50         "location", "manager", "description");
51         StepVerifier.create(result)
52             .expectNext(warehouse)
53             .verifyComplete();
54
55     @Test
56     void testFindWarehouseById() {
57         Warehouse warehouse = new Warehouse();
58         warehouse.setId(1);
59         when(warehouseMapper.selectById(1)).thenReturn(warehouse);
60
61         Mono<Warehouse> result = warehouseService.selectWarehouseById(1);
62         StepVerifier.create(result)
63             .expectNext(warehouse)
64             .verifyComplete();
65
66     @Test
67     void testFindAllWarehouses() {
68         List<Warehouse> warehouses = new ArrayList<>();
69         Warehouse warehouse = new Warehouse();
70         warehouses.add(warehouse);
71
72         when(warehouseMapper.selectList(any())).thenReturn(warehouses);
73
74         Flux<Warehouse> result = warehouseService.selectWarehouse("", 1L, 10L);
75         StepVerifier.create(result)
76             .expectNextSequence(warehouses)
77             .verifyComplete();
78     }

```

```

79
80     @Test
81     void testDeleteWarehouse() {
82         when(warehouseMapper.deleteById(1)).thenReturn(1);
83         Mono<Void> result = warehouseService.deleteWarehouse(1);
84         StepVerifier.create(result)
85             .expectNext()
86             .verifyComplete();
87     }
88 }

```

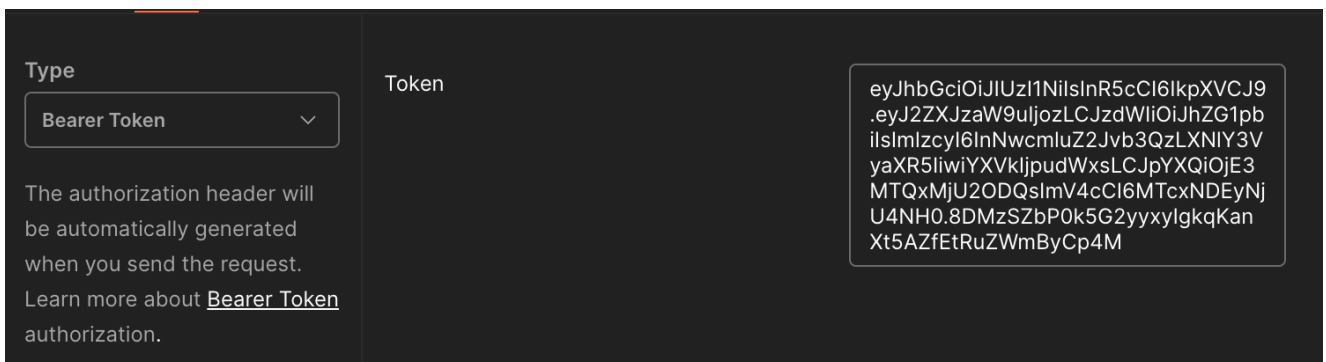
## 5.2 集成测试

集成测试按照下面的逻辑进行：

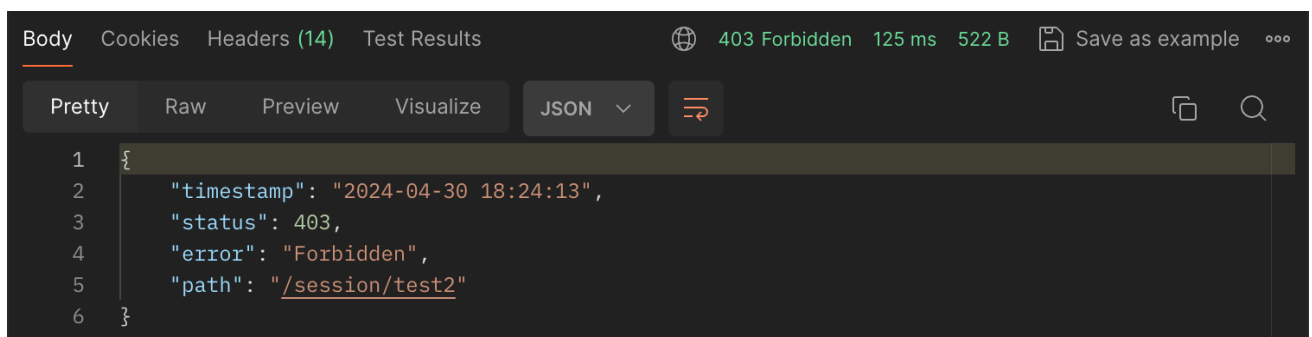
首先是用户注册，然后登入，然后对仓库、物品、订单进行覆盖所有业务逻辑的操作，最后登出。实现并运行后，通过了当前阶段的集成测试。

## 5.3 接口测试（使用Postman）

其中大部分接口需要配置token进行鉴权，否则会因为权限不足无法使用。



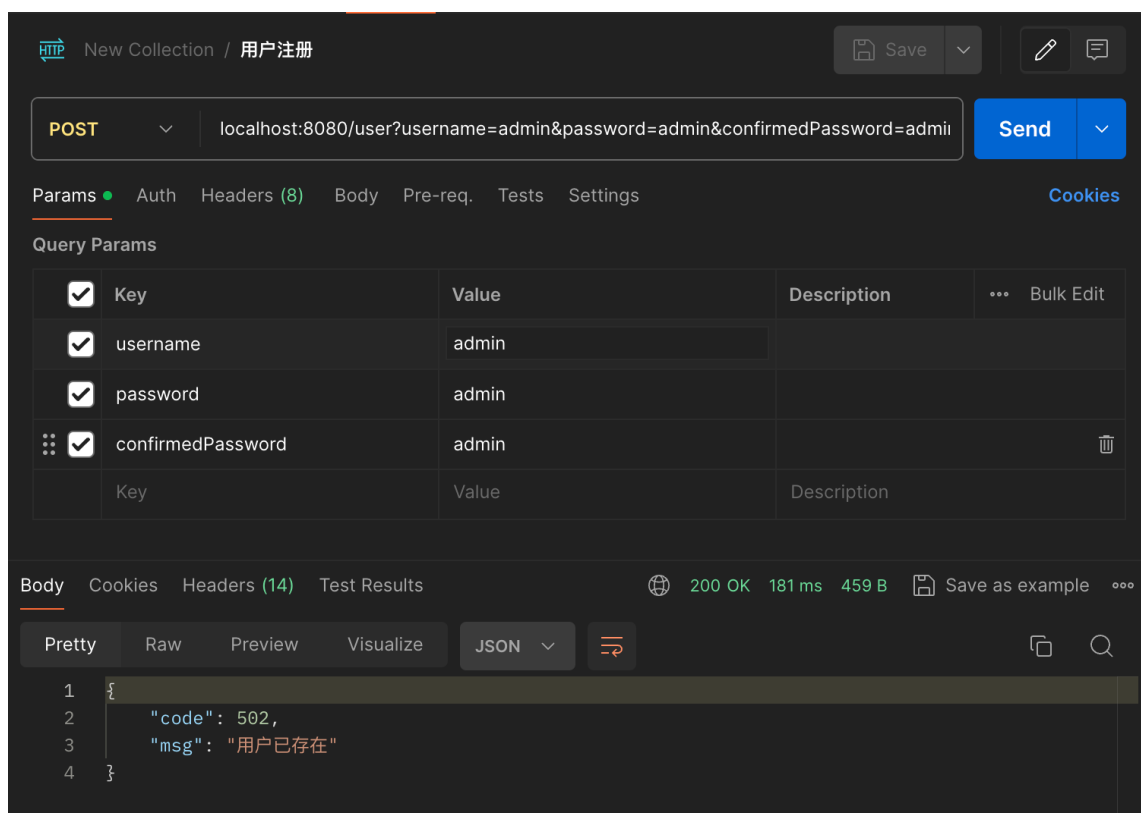
无权限，接口会返回403



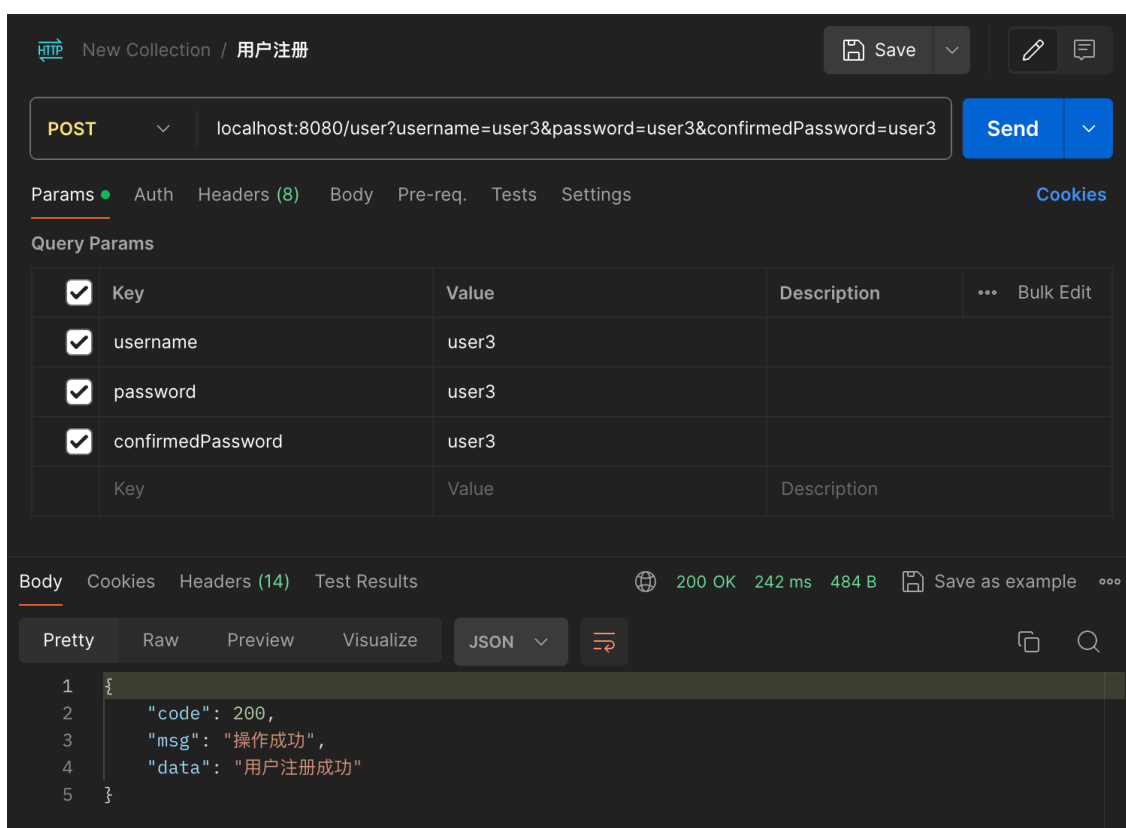
### 5.3.1 用户管理接口测试

#### 1. 用户注册

##### a. 已存在用户

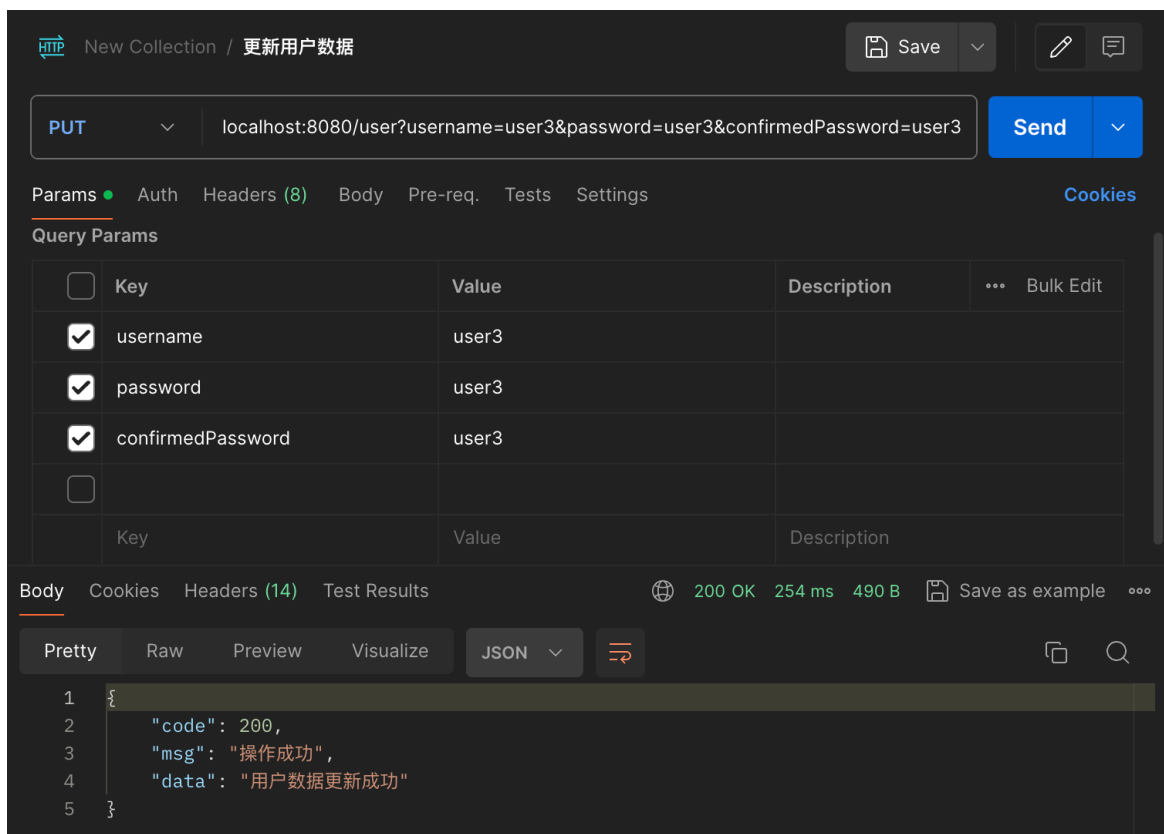


## b. 成功注册用户



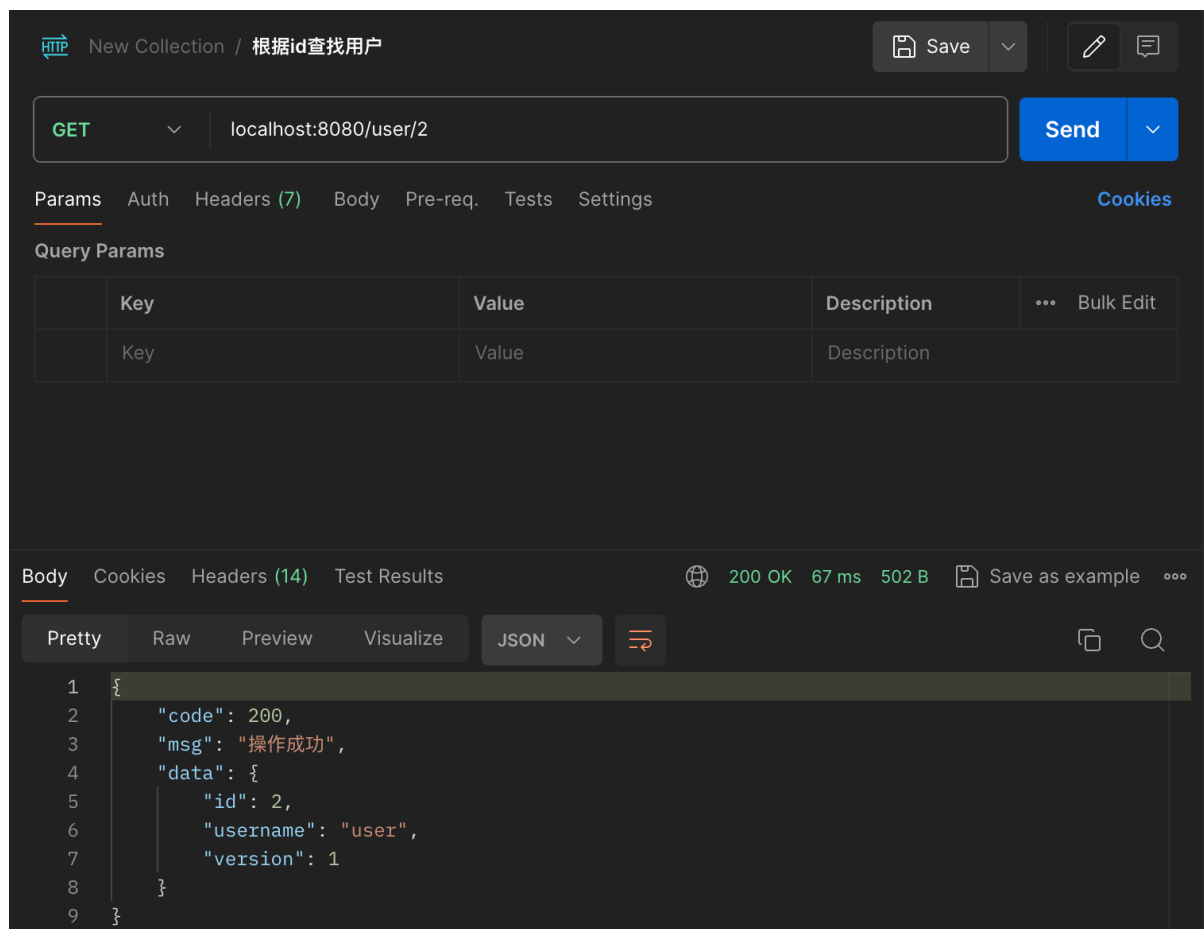
## 2. 更新用户数据

### a. 更新数据成功



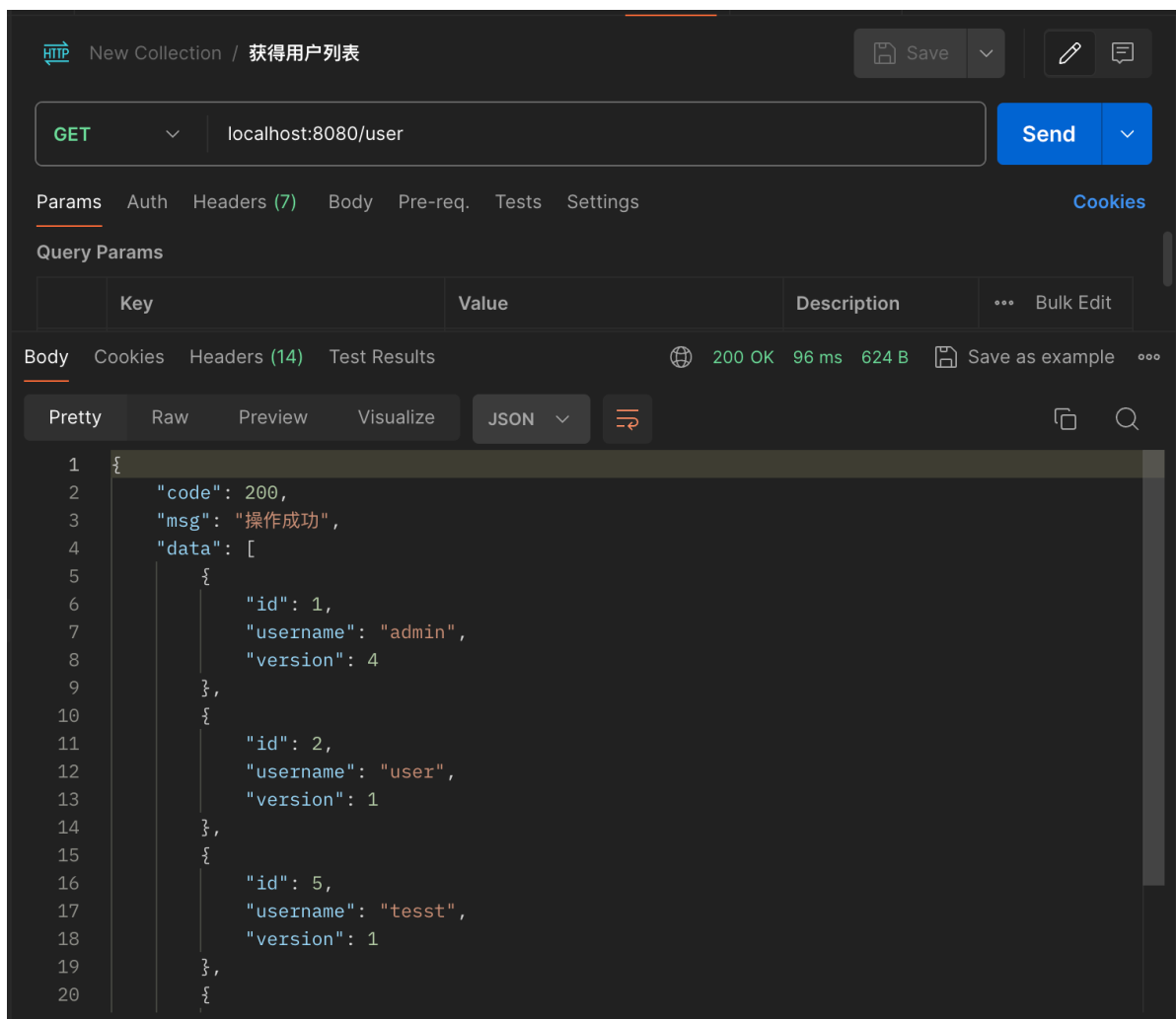
### 3. 根据id查找用户

#### a. 返回用户信息，自动屏蔽密码



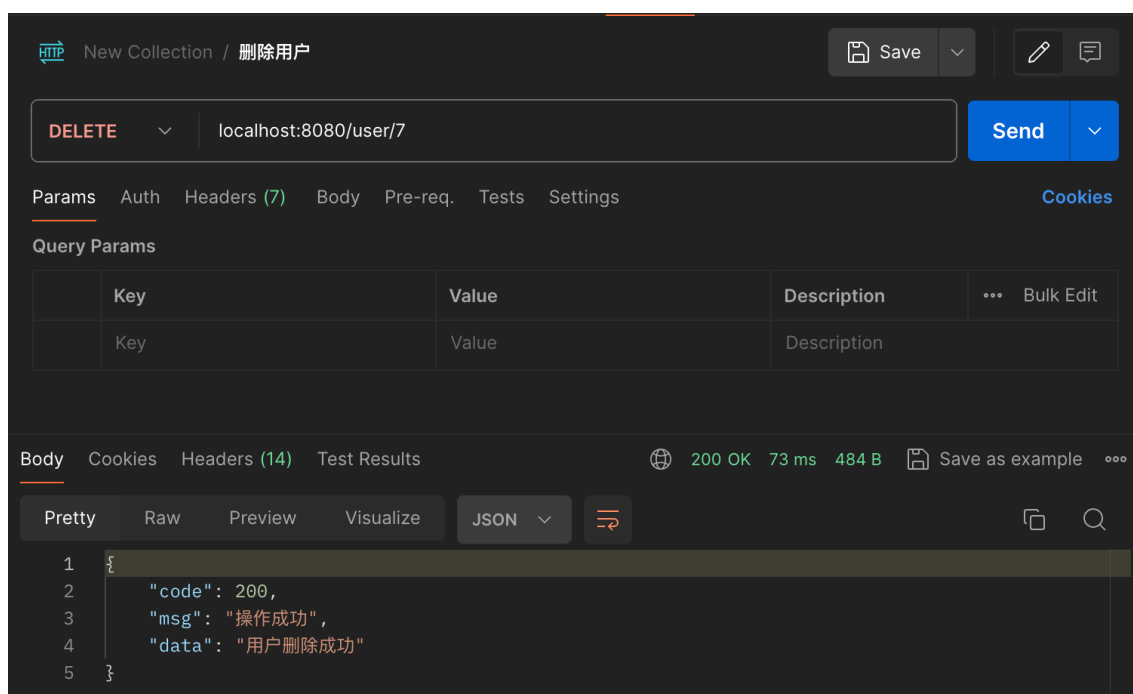
### 4. 获得用户列表

#### a. 获得所有用户的列表，自动屏蔽密码



## 5. 删除用户

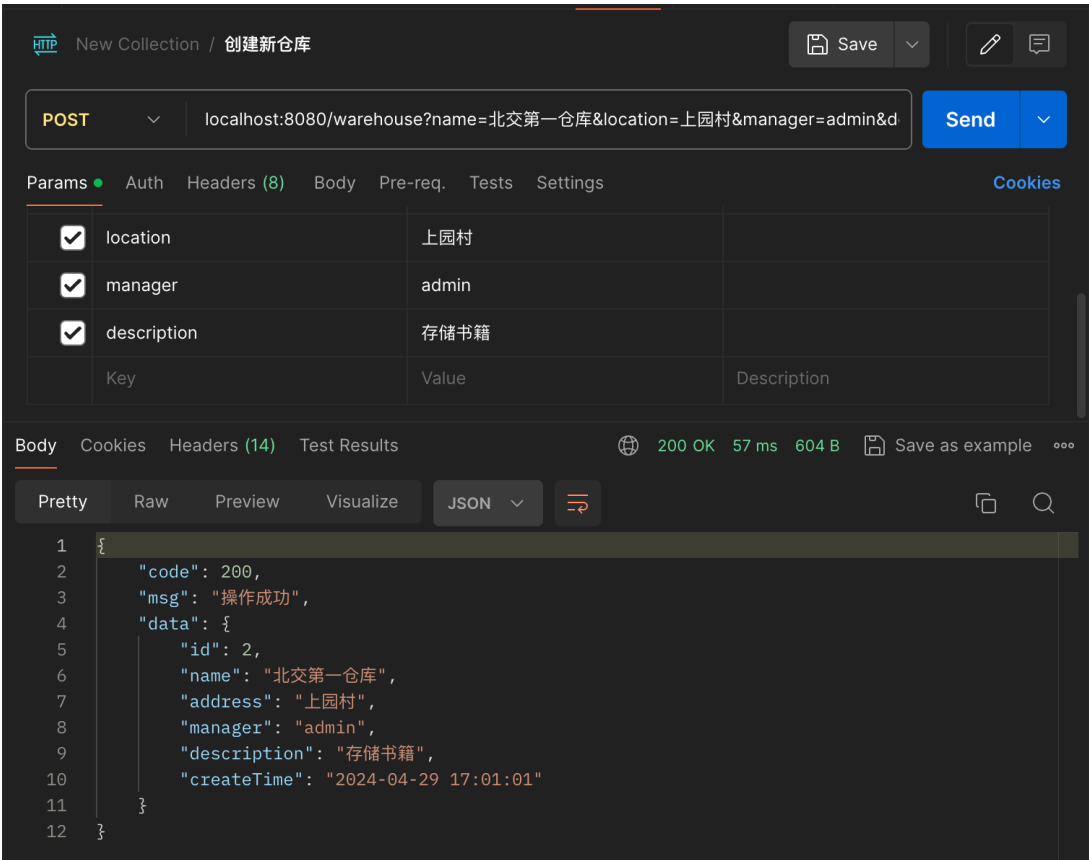
### a. 删除用户成功



## 5.3.2 仓库管理接口测试

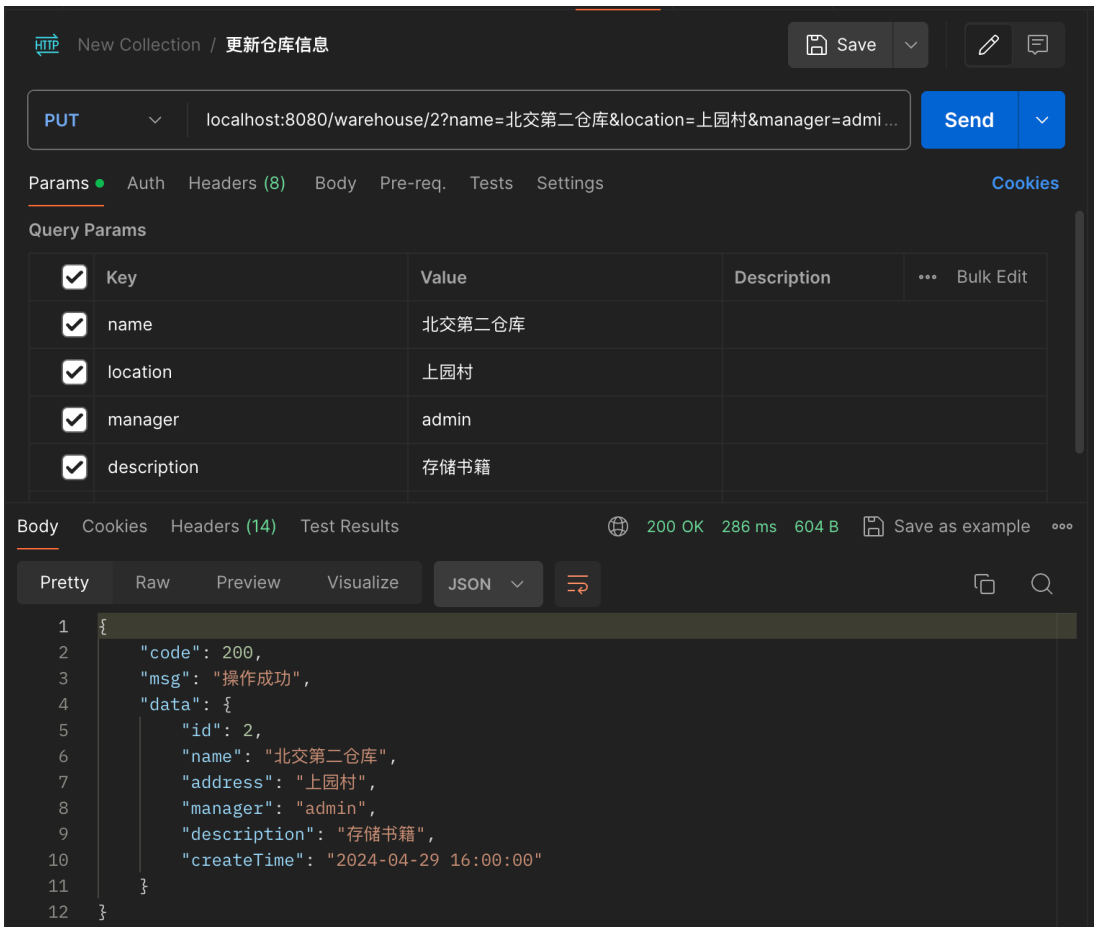
### 1. 创建新仓库

a. 仓库创建成功



2. 更新仓库信息

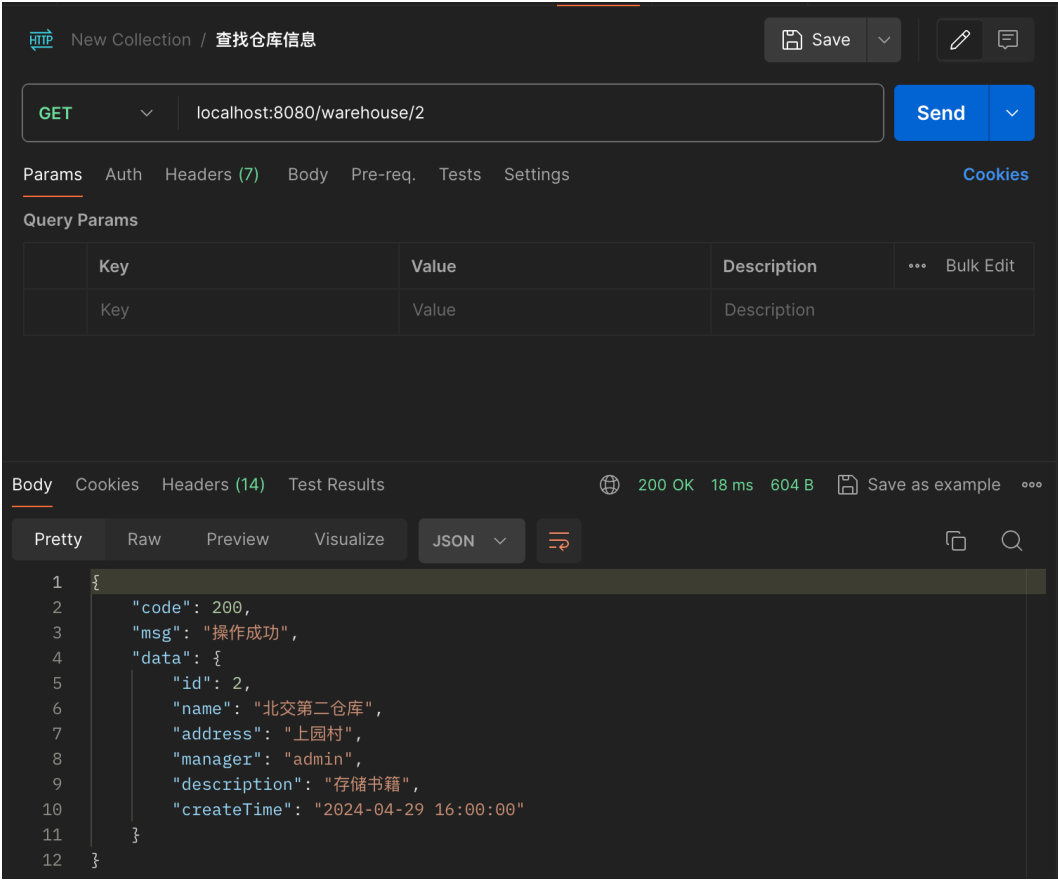
a. 仓库信息更新成功





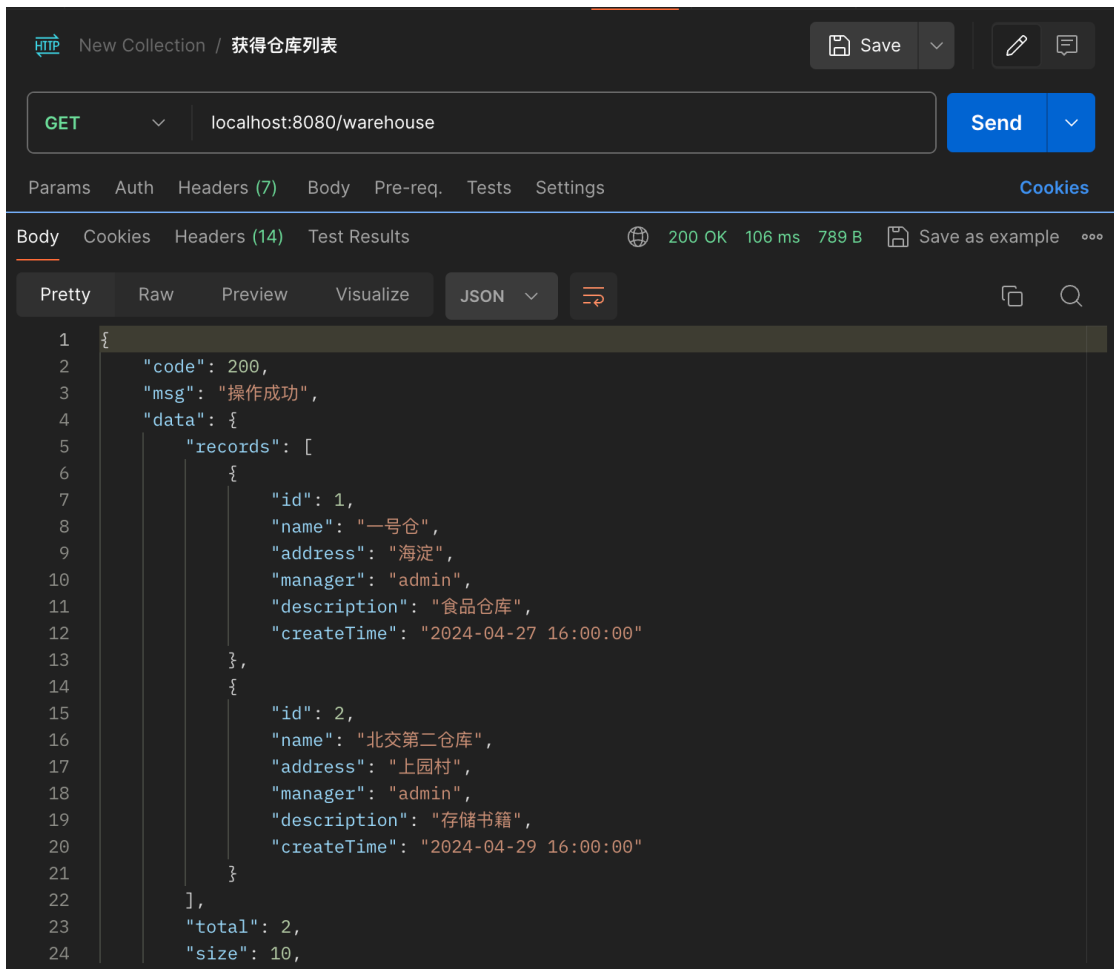
3. 根据id查找仓库信息

a. 查找信息成功



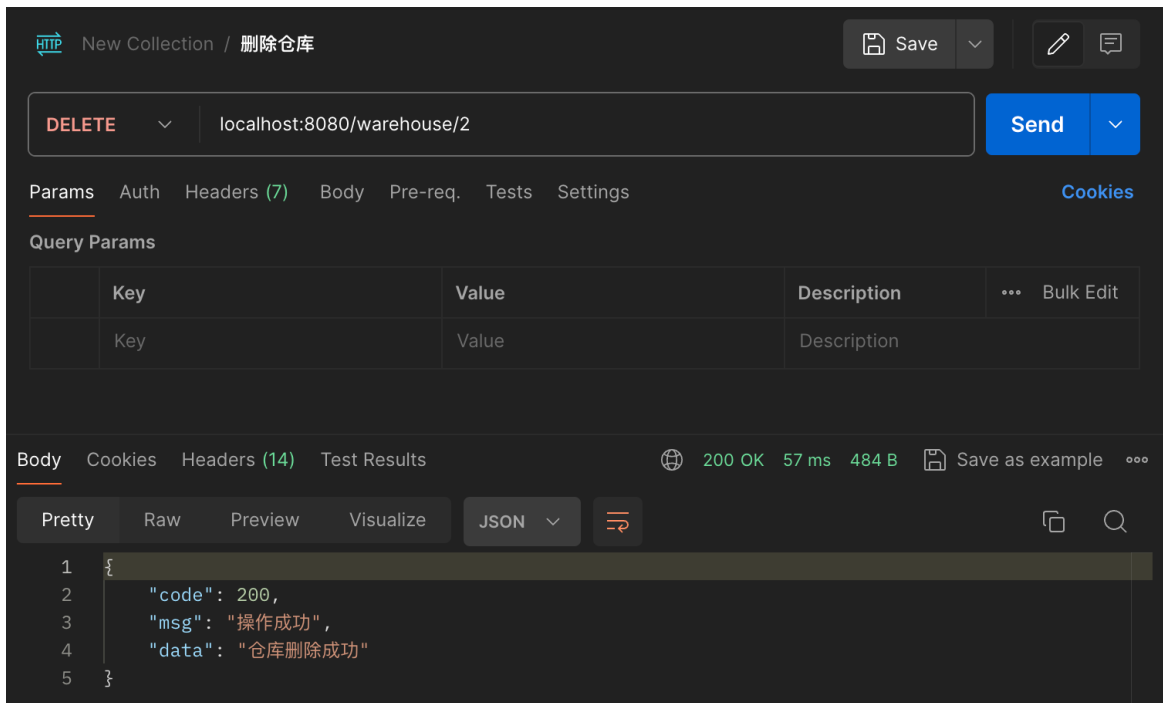
4. 获得仓库列表

a. 查询仓库信息成功



## 5. 删除仓库

### a. 删除仓库成功



### 5.3.3 物品管理接口测试

#### 1. 获得物品列表

##### a. 成功获得物品列表

HTTP New Collection / 获得物品列表

GET localhost:8080/item

Send

Params Auth Headers (7) Body Pre-req. Tests Settings Cookies

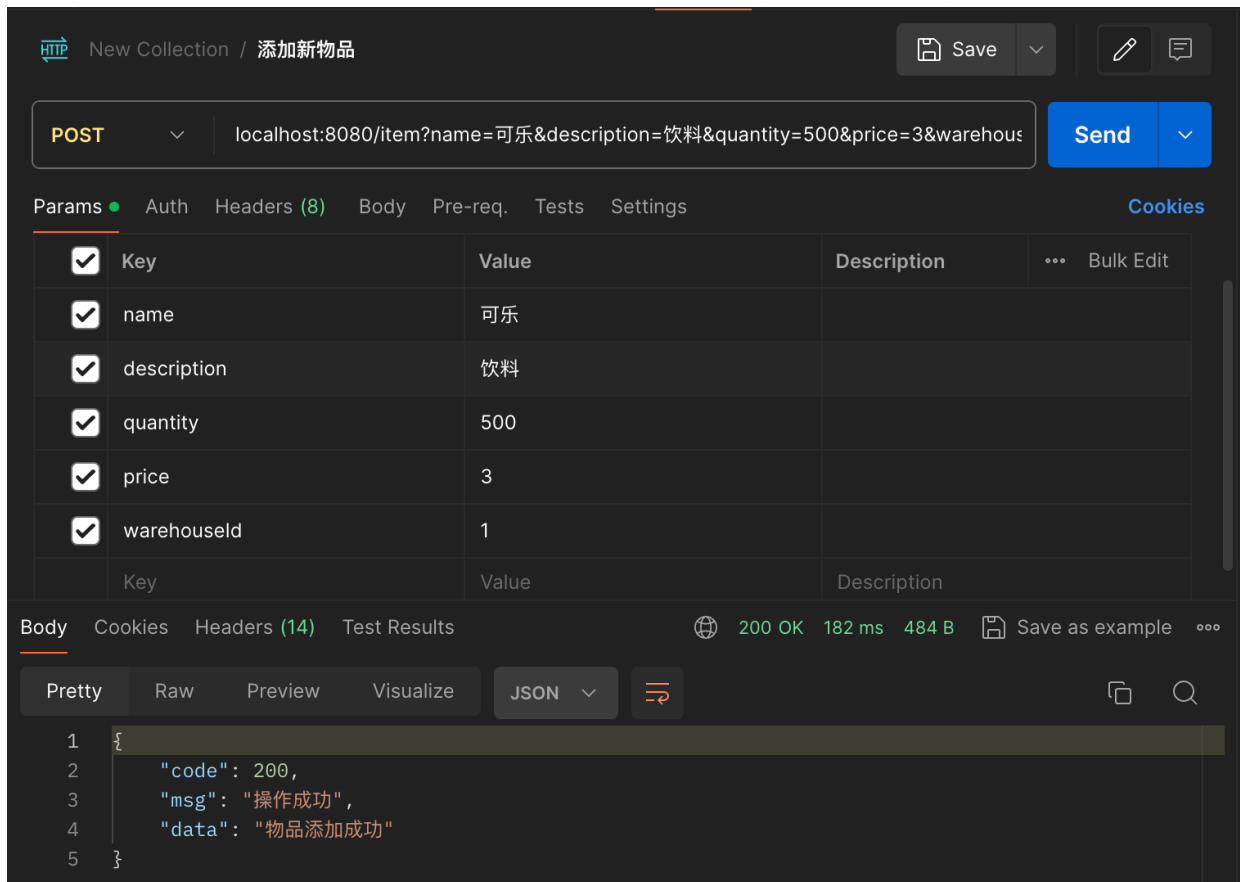
Body Cookies Headers (14) Test Results 200 OK 54 ms 797 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "code": 200,
3   "msg": "操作成功",
4   "data": [
5     {
6       "id": 1,
7       "name": "薯片",
8       "description": "薯片零食",
9       "quantity": 50,
10      "price": 2.0,
11      "warehouseId": 1,
12      "createTime": "2024-04-27 16:00:00",
13      "updateTime": "2024-04-27 16:00:00"
14    },
15    {
16      "id": 2,
17      "name": "薯条",
18      "description": "薯条零食",
19      "quantity": 50,
20      "price": 5.0,
21      "warehouseId": 1,
22      "createTime": "2024-04-25 16:00:00",
23      "updateTime": "2024-04-26 16:00:00"
24    }
25  ]
26 }
```

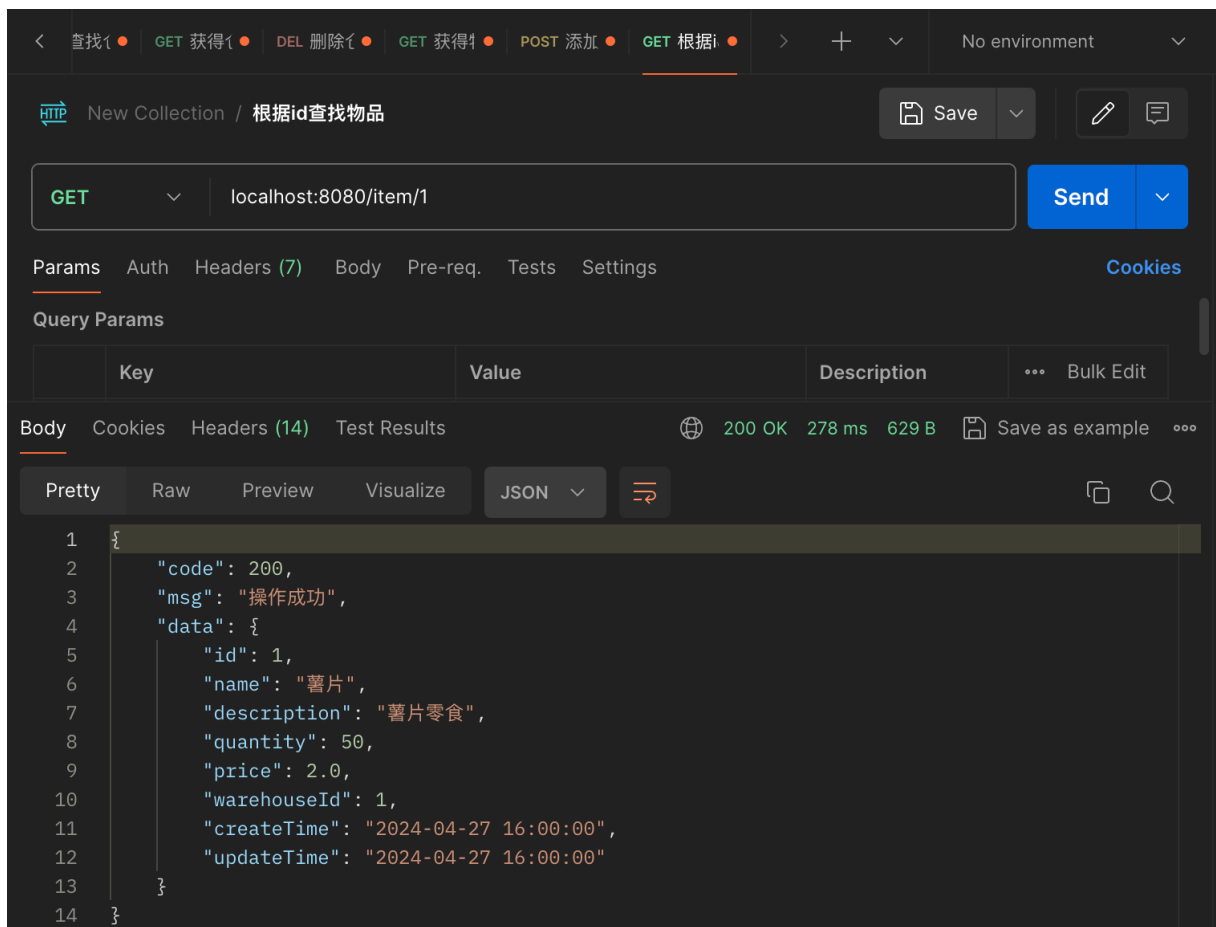
## 2. 添加新物品

### a. 成功添加新物品



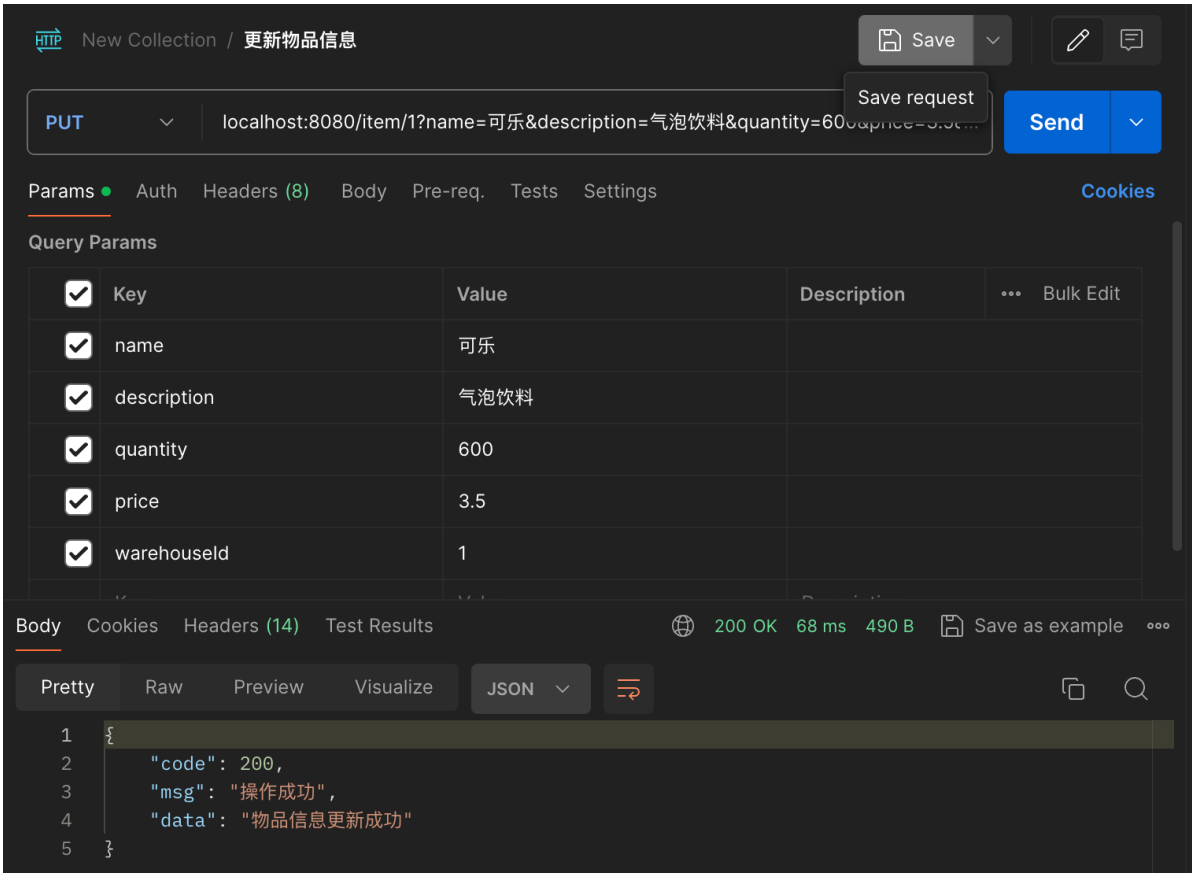
### 3. 根据id获得物品信息

#### a. 成功获得物品



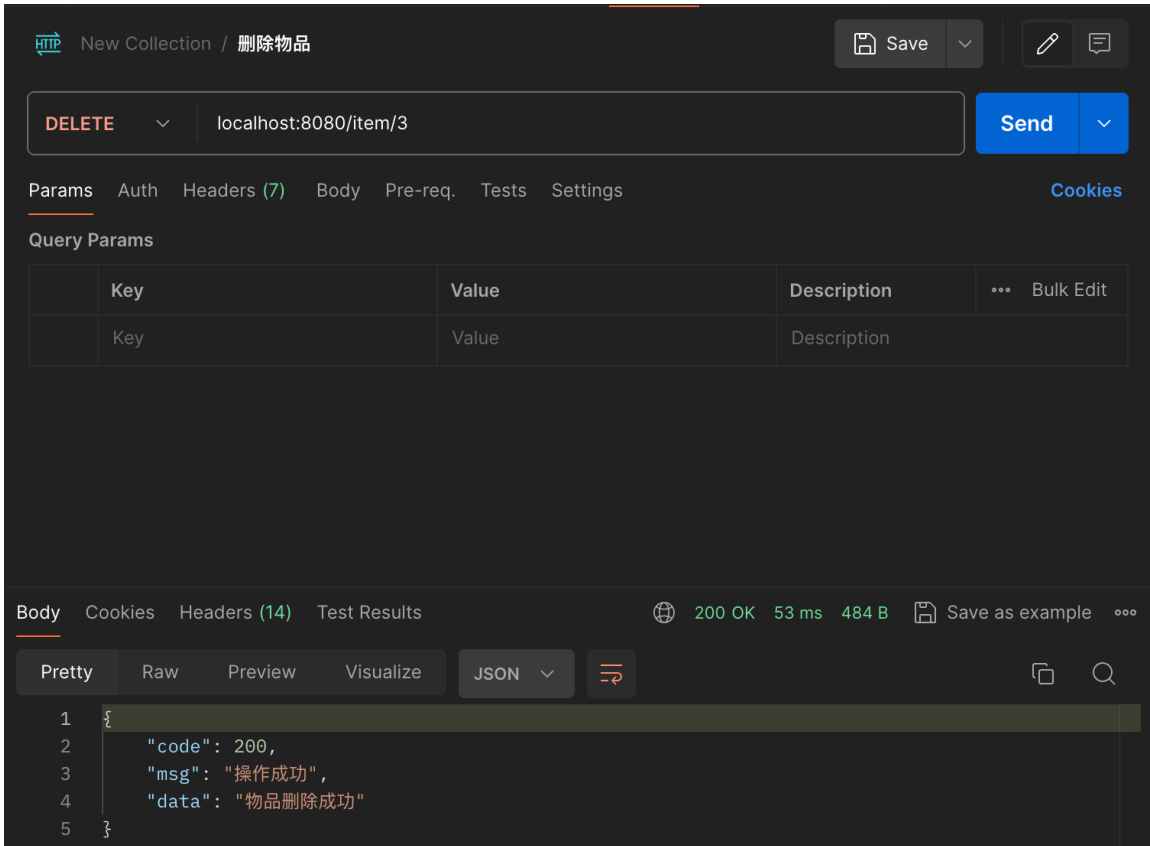
### 4. 更新物品信息

a. 成功更新物品信息



5. 删除物品

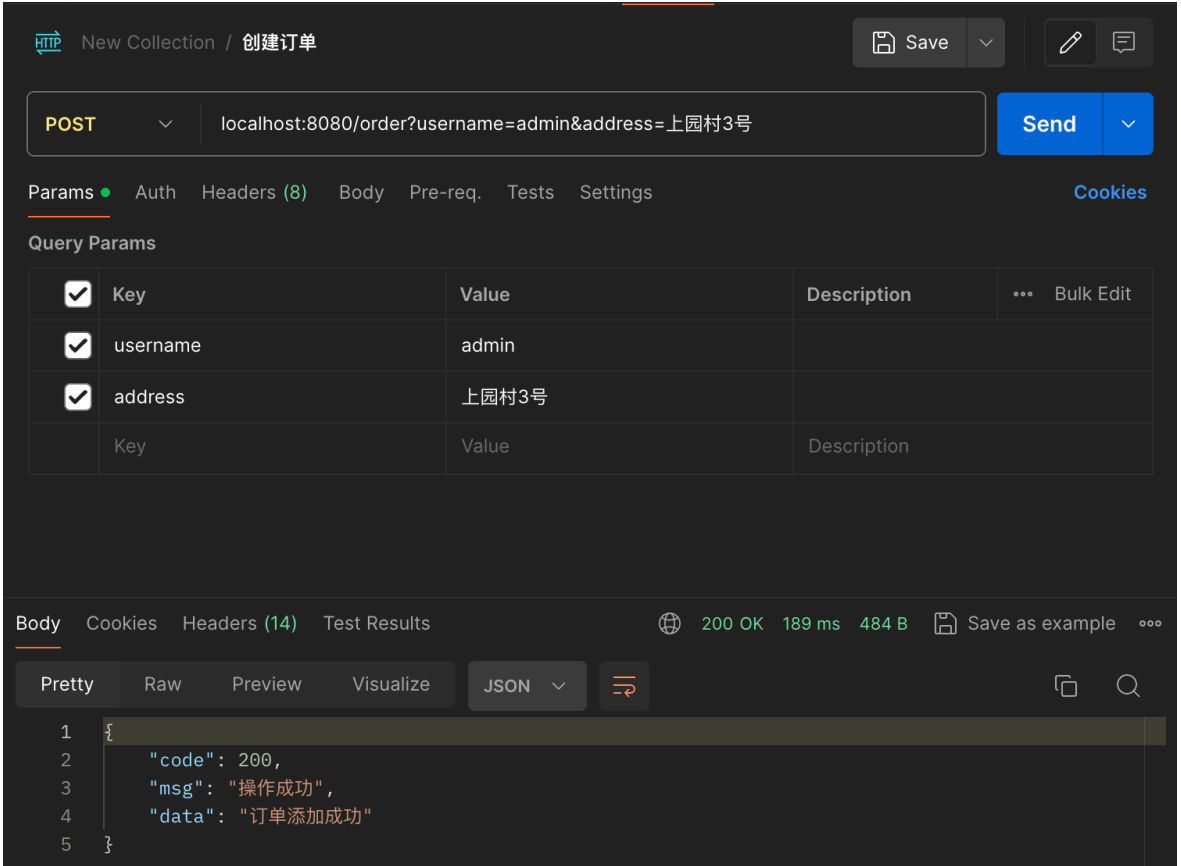
a. 成功删除物品



5.3.4 订单管理接口测试

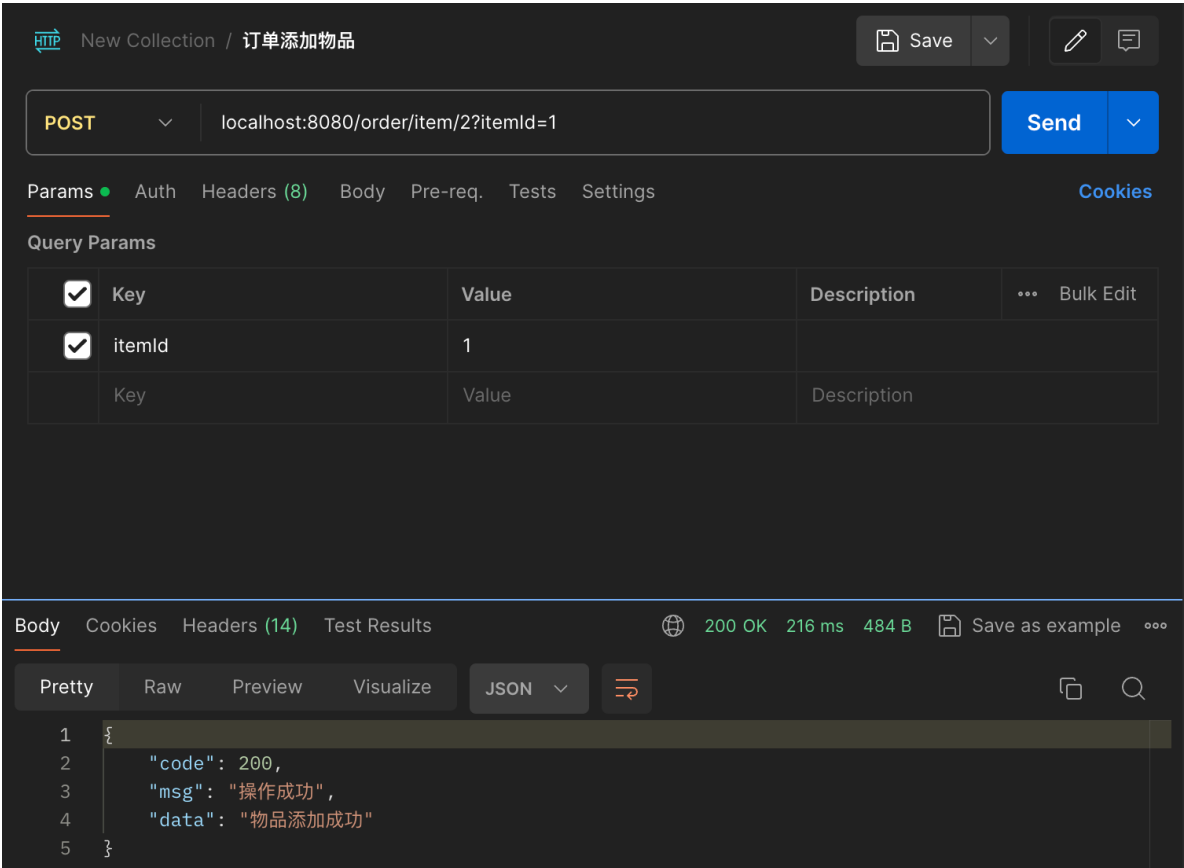
1. 创建订单

a. 成功创建订单



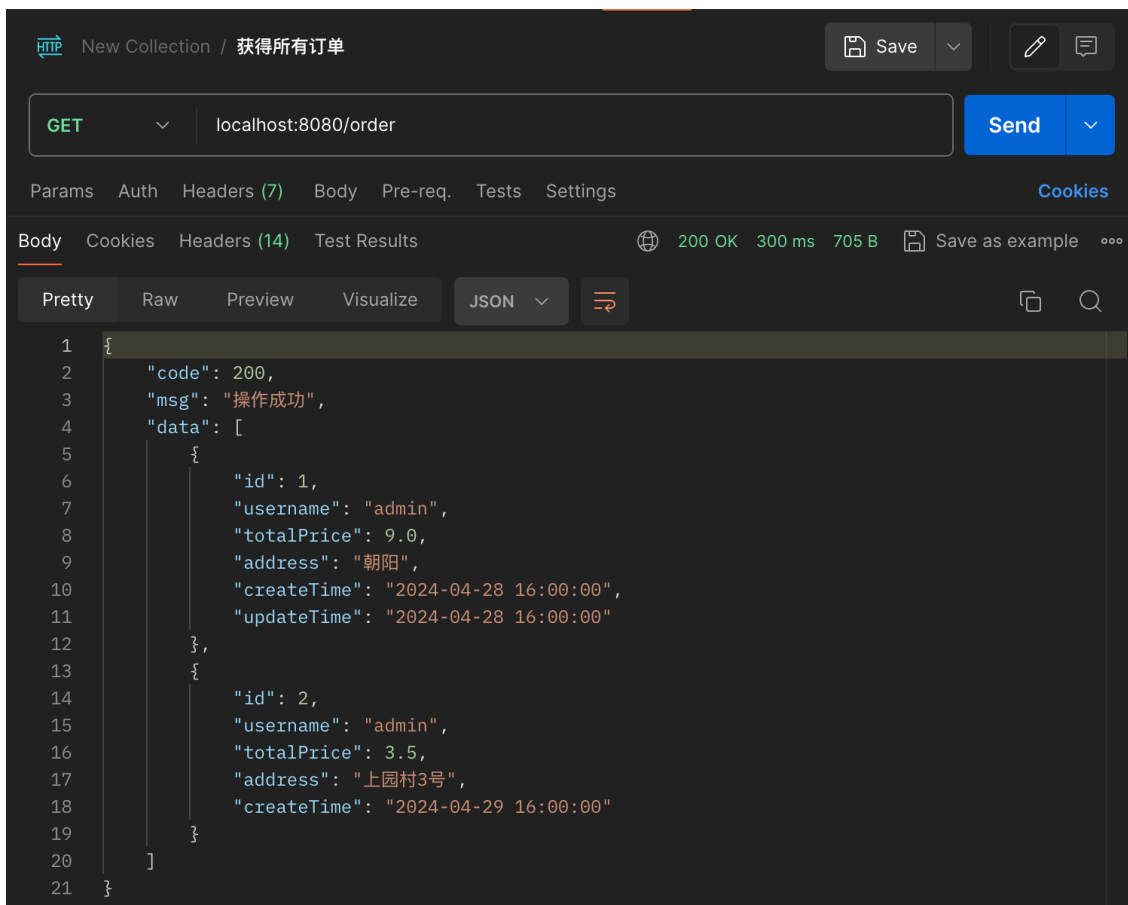
2. 向订单中添加物品

a. 添加物品成功



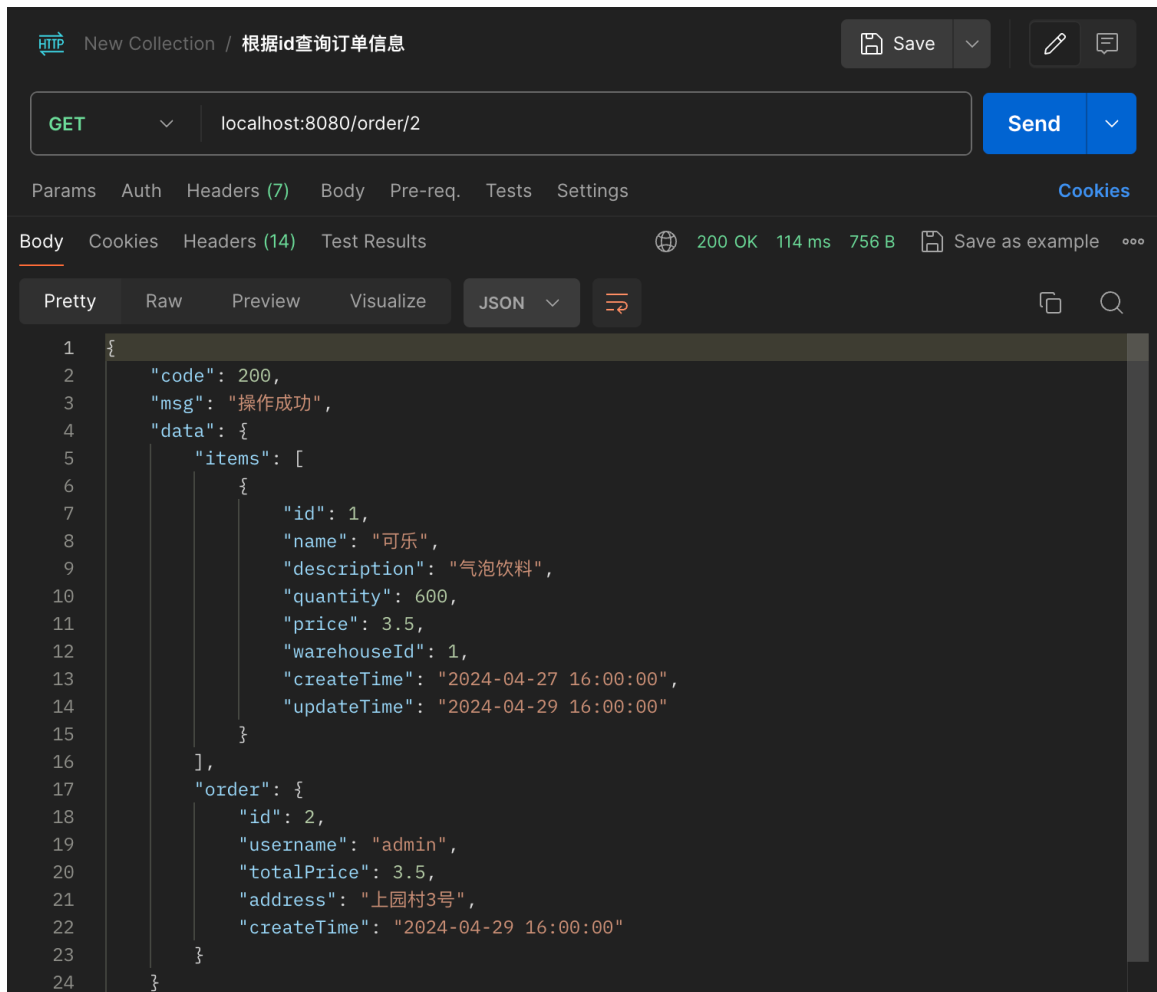
### 3. 获得所有订单

#### a. 成功获得所有订单



### 4. 根据id查询订单信息

#### a. 成功查询到订单信息



## 5. 根据用户名查询订单信息

### a. 成功查询到订单信息



HTTP New Collection / 根据用户id查询订单

GET localhost:8080/order/user?username=admin Send

Params Auth Headers (7) Body Pre-req. Tests Settings Cookies

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	username	admin			

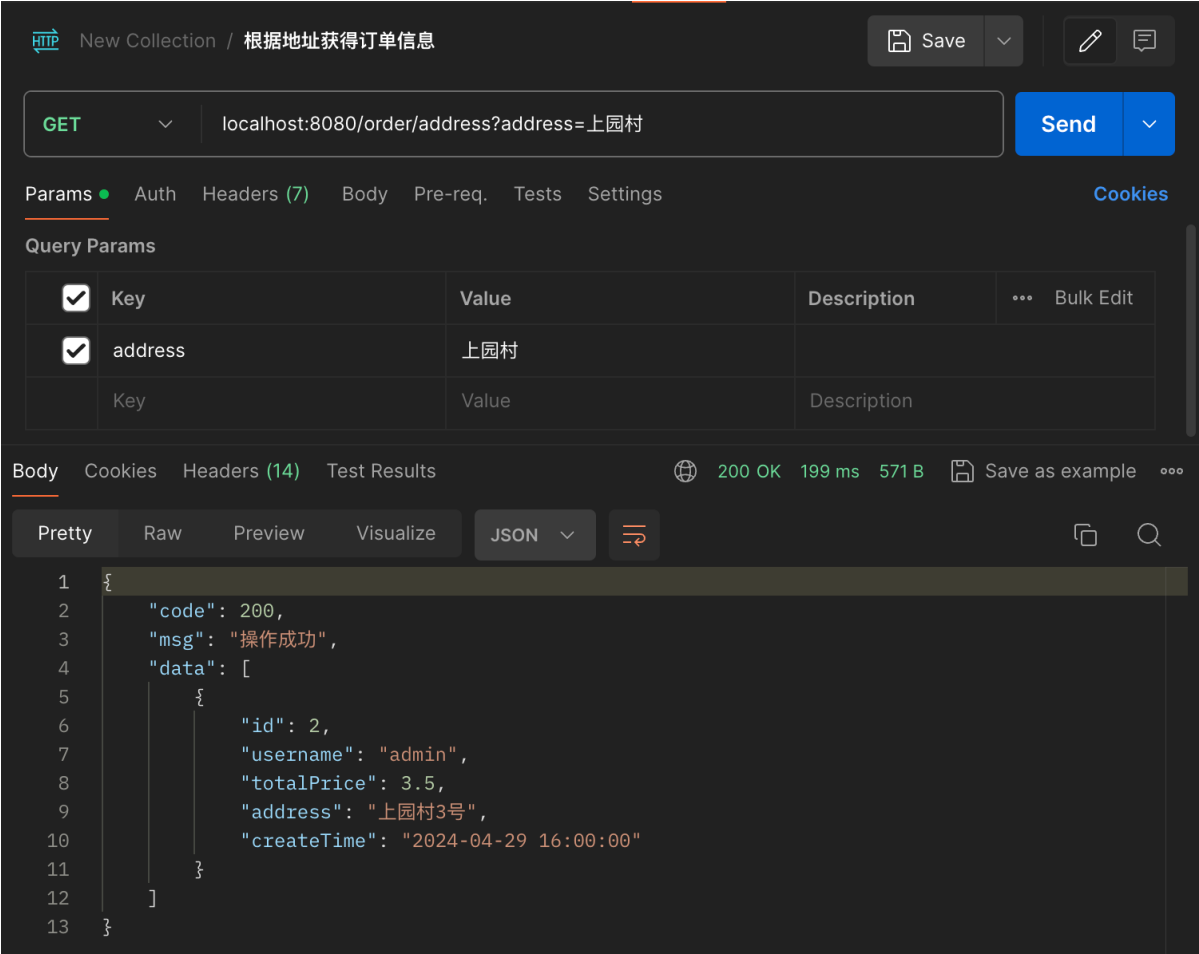
Body Cookies Headers (14) Test Results 200 OK 285 ms 706 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "code": 200,
3   "msg": "操作成功",
4   "data": [
5     {
6       "id": 1,
7       "username": "admin",
8       "totalPrice": 12.0,
9       "address": "朝阳",
10      "createTime": "2024-04-28 16:00:00",
11      "updateTime": "2024-04-28 16:00:00"
12    },
13    {
14      "id": 2,
15      "username": "admin",
16      "totalPrice": 3.5,
17      "address": "上园村3号",
18      "createTime": "2024-04-29 16:00:00"
19    }
20  ]
21 }
```

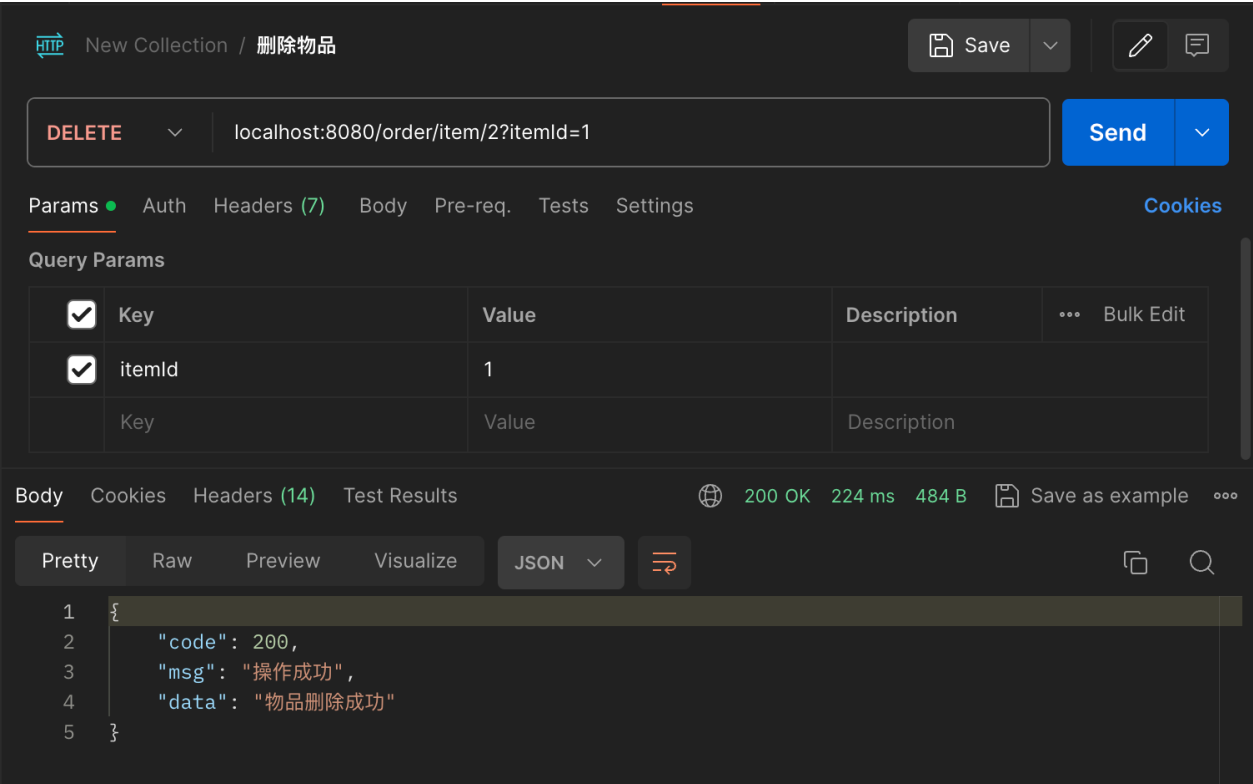
## 6. 根据地址获取订单信息

### a. 成功获得订单信息



7. 删除订单物品

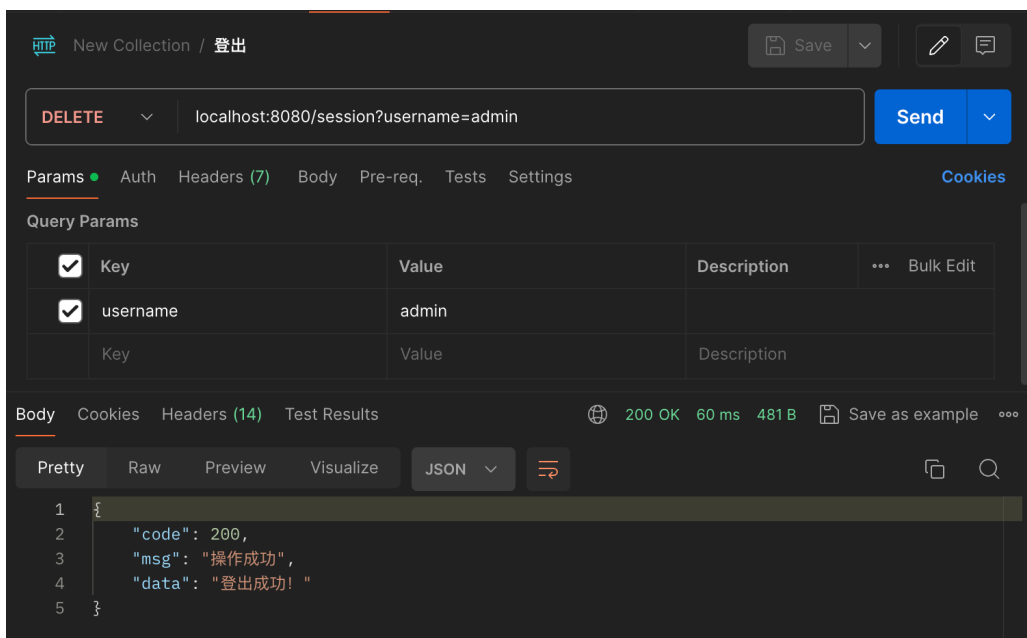
a. 成功删除物品



8. 删除订单信息

a. 成功删除订单

a. 成功登出



## 六、阶段总结

我们完成了当前阶段中的所有basic和credit要求（详见要求的pdf文档）。

