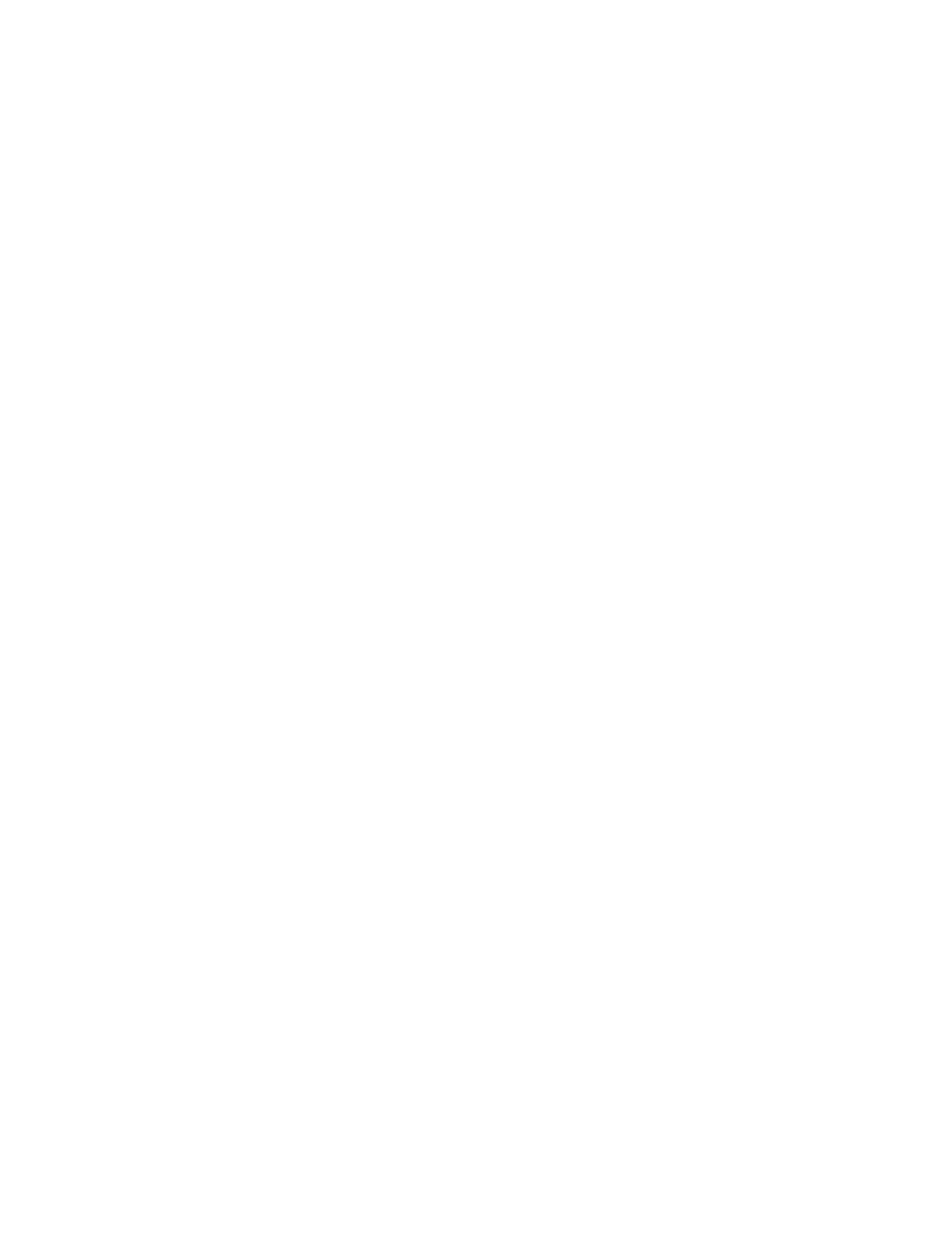
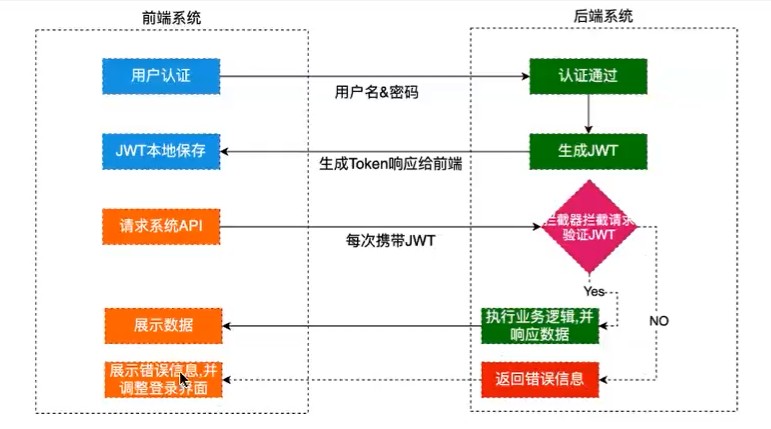
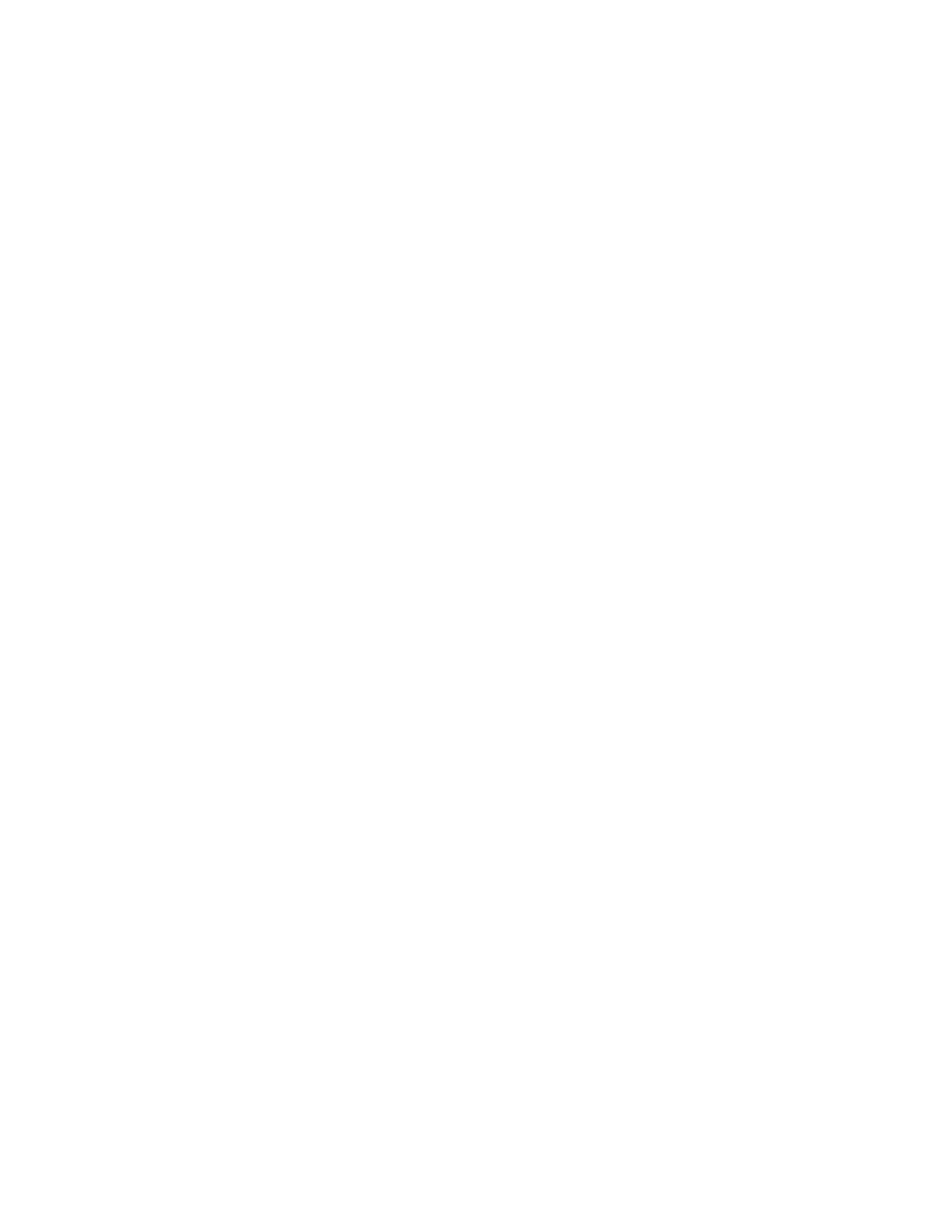
JSON Web Token

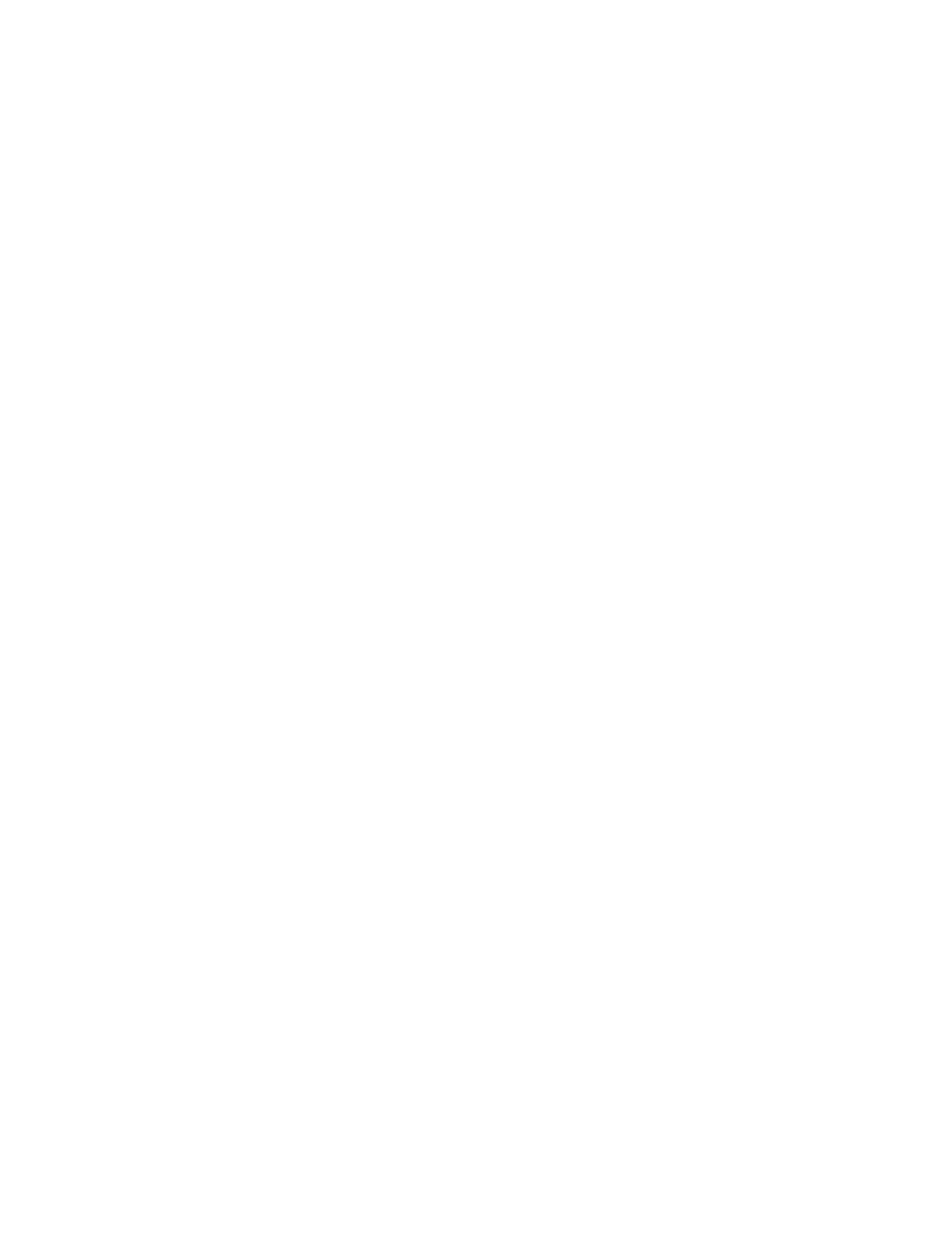
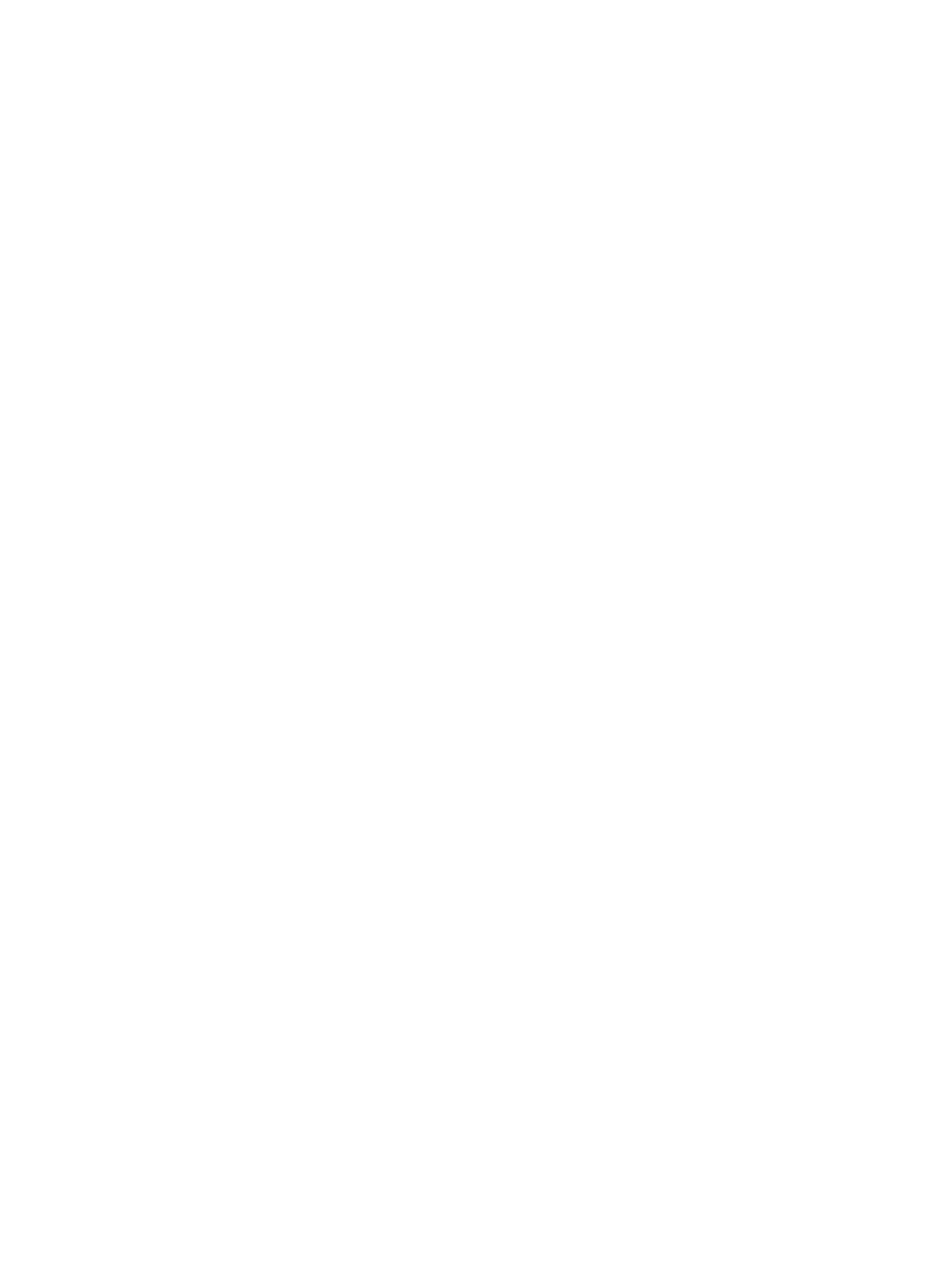


1. 什么是JWT？

* JSON Web Token (JWT)是⼀个开放标准(RFC 7519)，它定义了⼀种紧凑的、⾃包含的⽅式，⽤于作为JSON对象在各⽅之间安全地传输信息。该信息可以被验证和信任，因为它是数字签名的。
  1. 什么时候应该⽤JWT？
* Authorization (授权) : 这是使⽤JWT的最常⻅场景。⼀旦⽤⼾登录，后续每个请求都将包含JWT，允许⽤⼾访问该令牌允许的路由、服务和资源。单点登录是现在⼴泛使⽤的JWT的⼀个特 性，因为它的开销很⼩，并且可以轻松地跨域使⽤。
* Information Exchange (信息交换) : 对于安全的在各⽅之间传输信息⽽⾔，JSON Web Tokens⽆疑是⼀种很好的⽅式。因为JWT可以被签名，例如，⽤公钥/私钥对，你可以确定发送⼈就是它们所说的那个⼈。另外，由于签名是使⽤头和有效负载计算的，您还可以验证内容没有被篡改。

## 认证流程

1. ⾸先，前端通过Web表单将⾃⼰的⽤⼾名和密码发送到后端的接⼝。这⼀过程⼀般是⼀个HTTP POST请求。建议的⽅式是通过SSL加密的传输(https协议) ，从⽽避免敏感信息被嗅探。
2. 后端核对⽤⼾名和密码成功后，将⽤⼾的id等其他信息作为JWT Payload (负载)，将其与头部分别进⾏Base64编码拼接后签名，形成⼀个JWT(Token)。形成的JWT就是⼀个形同11. zzz. xxx的字符串。token head.payload.signature



1. 后端将JWT字符串作为登录成功的返回结果返回给前端。 前端可以将返回的结果保存在localStorage或sessionStorage上， 退出登录时前端删除保存的JWT即可。
2. 前端在每次请求时将JWT放⼊HTTP Header中的Authorization位。 (解决XSS和XSRF问题)

#### 后端检查是否存在，如存在验证JWT的有效性。

* + 检查签名是否正确;
  + 检查Token是否过期;
  + 检查Token的接收⽅是否是⾃⼰(可选)

#### 验证通过后后端使⽤JWT中包含的⽤⼾信息进⾏其他逻辑操作，返回相应结果。

* 1. JWT优势在哪?

1. 简洁(Compact): 可以通过URL，POST参数或者在HTTP header发送，数据量⼩，传输速度快
2. ⾃包含(Self-contained):负载中包含了所有⽤⼾所需要的信息，避免了多次查询数据库
3. 因为Token是 以JSON加密的形式保存在客⼾端的，所以JWT是跨语⾔的，原则上任何web形式都

#### ⽀持。

1. 不需要在服务端保存会话信息，特别适⽤于分布式微服务。
   1. JWT具体包含信息
2. header

标头通常由两部分组成: 令牌的类型(即JWT) 和所使⽤的签名算法，例如HMAC、SHA256或RSA。 它会使⽤Base64 编码组成JWT 结构的第⼀部分。

#### 注意:Base64是⼀ 种编码，也就是说，它是可以被翻译回原来的样⼦来的。它并不是⼀种加密过程。

##### Java

1 {

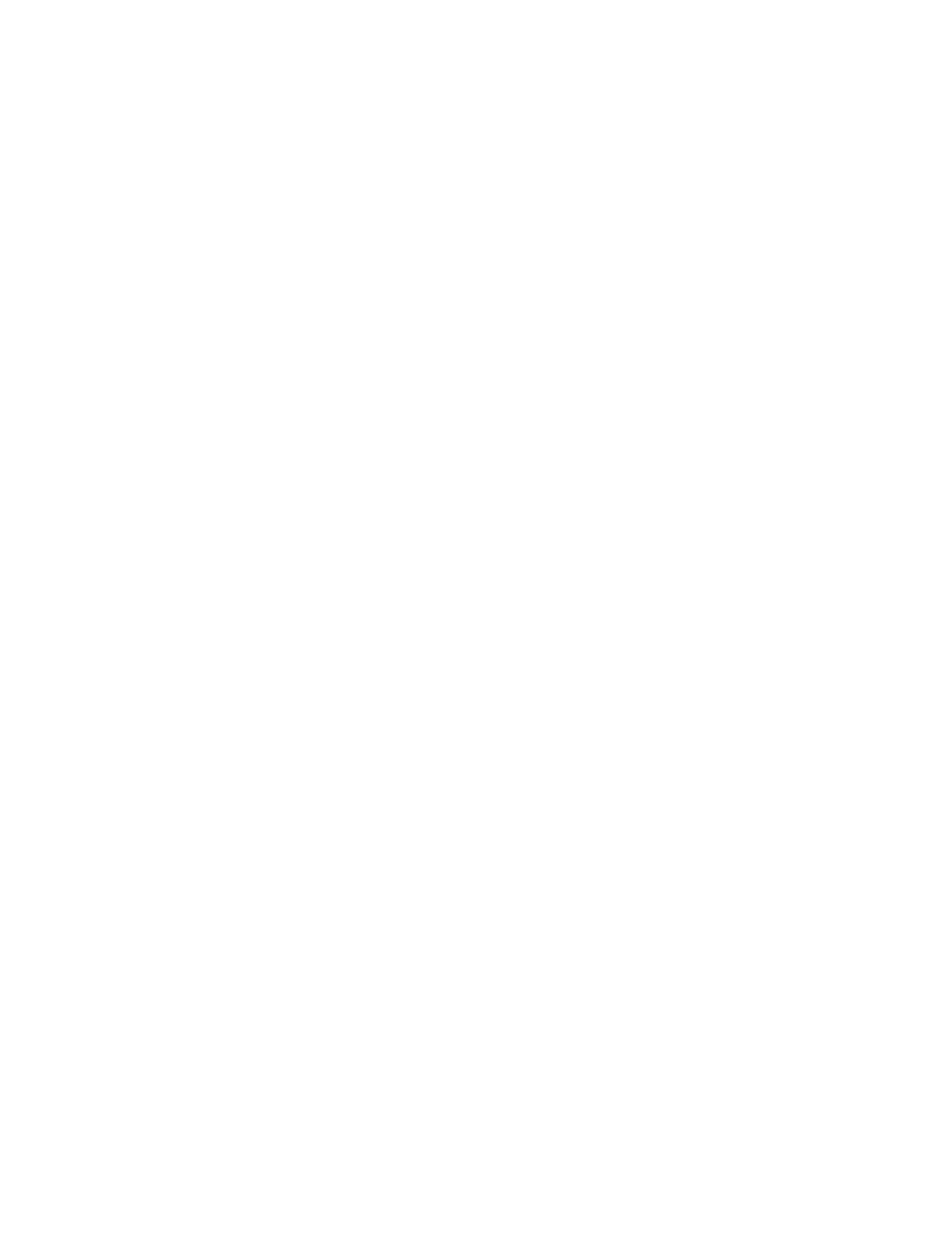
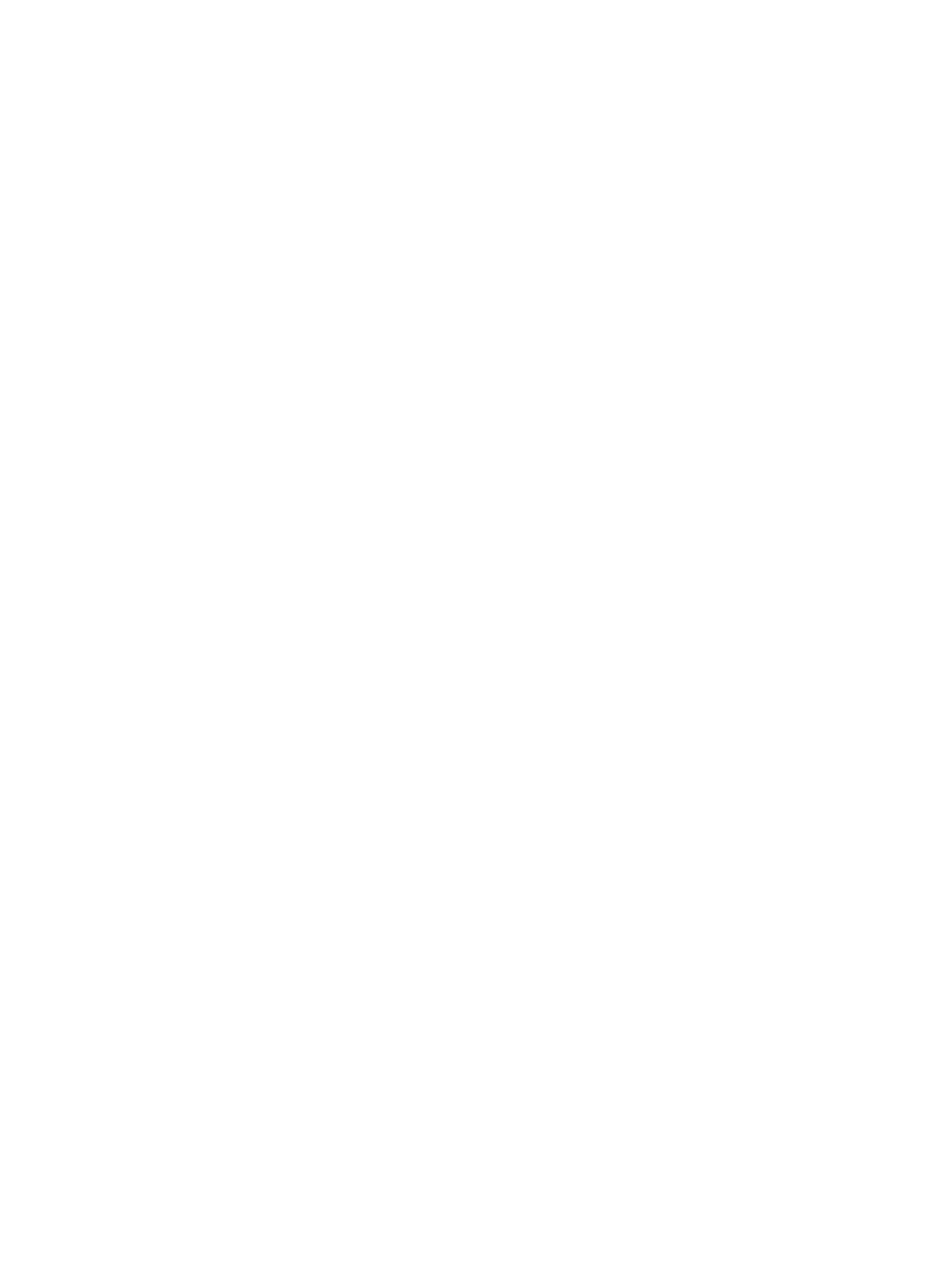
2 "alg":"HS256",

3 "typ":"JWT" 4 }

### Payload

令牌的第⼆部分是有效负载，其中包含声明。声明是有关实体(通常是⽤⼾)和其他数据的声明。同样 的，它会使⽤Base64 编码组成JWT结构的第⼆部分

|  |  |  |
| --- | --- | --- |
| JSON |  | |
| 1 | { |  |
| 2 |  | "sub" : "HS256" |
| 3 |  | "name" : "yjiewei" |
| 4 |  | "admin" : "true" |
| 5 | } |  |



*<!--引⼊mybatis-->*

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.1.0</version>

</dependency>

*<!--引⼊jwt-->*

<dependency>

<groupId>com.auth0</groupId>

<artifactId>java-jwt</artifactId>

<version>3.10.3</version>

</dependency>

### Signature

header和payload都是结果Base64编码过的，中间⽤.隔开，第三部分就是前⾯两部分合起来做签 名，密钥绝对⾃⼰保管好，签名值同样做Base64编码拼接在JWT后⾯。（签名并编码）

Apache

1 HMACSHA256 (base64Ur1Encode(header) + "." + base64Ur1Encode(payload) , secret);

# 光说不练假把式

* 1. 整合pom

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

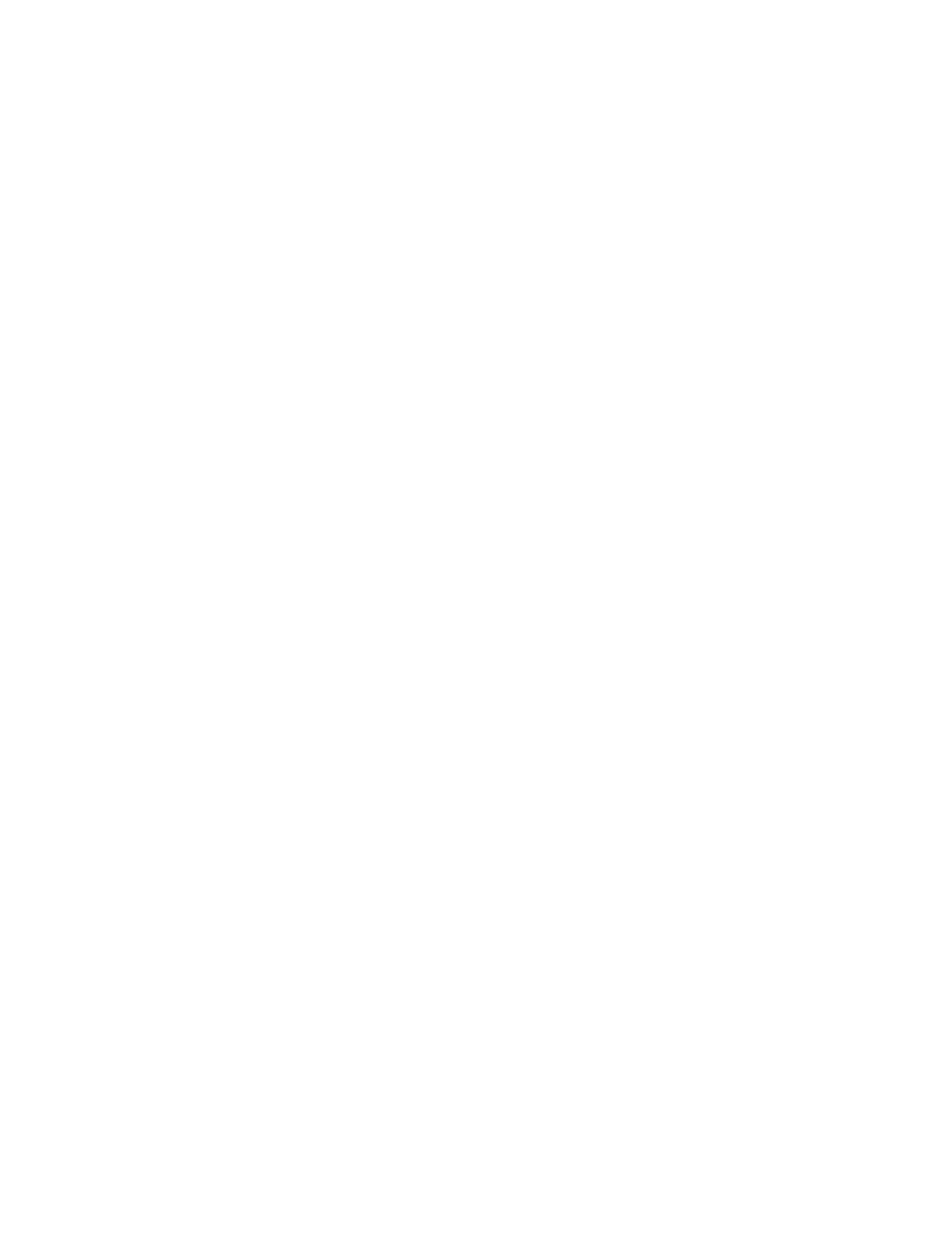
16

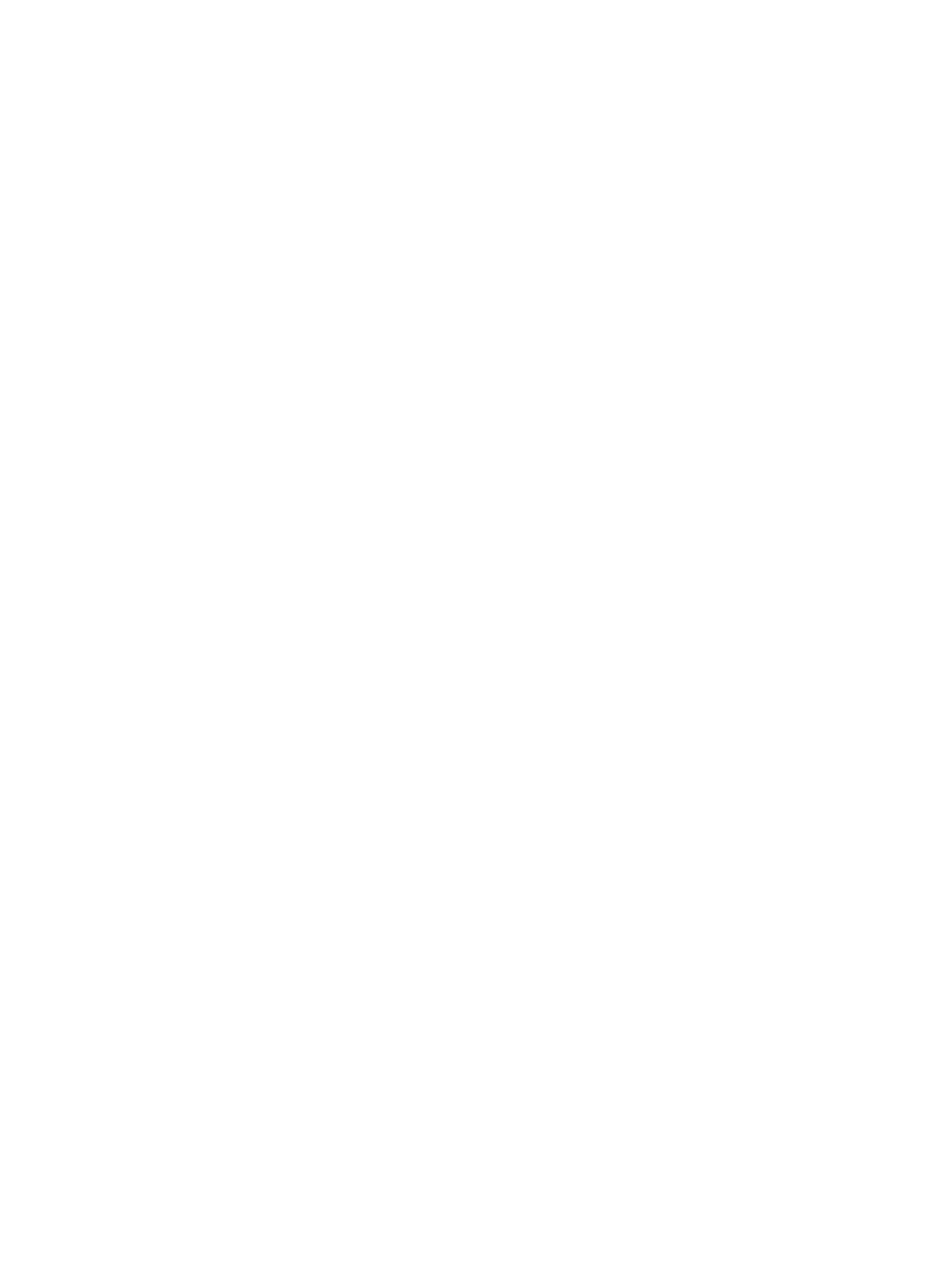
17

18

19

HTML

20



* + 1. *<!--引⼊mysql-->*
    2. <dependency>
    3. <groupId>mysql</groupId>
    4. <artifactId>mysql-connector-java</artifactId>
    5. <version>8.0.16</version>
    6. </dependency> 27

1. *<!--引⼊druid-->*
2. <dependency>
3. <groupId>com.alibaba</groupId>
4. <artifactId>druid</artifactId>
5. <version>1.2.0</version>
6. </dependency> 34
7. <dependency>
8. <groupId>org.projectlombok</groupId>
9. <artifactId>lombok</artifactId>
10. <optional>true</optional>
11. </dependency>
12. <dependency>
13. <groupId>org.springframework.boot</groupId>
14. <artifactId>spring-boot-starter-test</artifactId>
15. <scope>test</scope>
16. </dependency>
17. </dependencies>
    1. 测试JWT加密过程

TypeScript

1 public class Jwt {

|  |  |  |
| --- | --- | --- |
| 2 |  |  |
| 3 | */\*\** |
| 4 | *\* 获取JWT令牌* |
| 5 | *\*/* |
| 6 | @Test |
| 7 | public void getToken() { |
| 8 | Map<String, Object> map | = new HashMap<>(); |
| 9 |  |  |

10

11

12

Calendar instance = Calendar.*getInstance*();

instance.add(Calendar.*SECOND*, 2000);

13

14

15

16

17

18

19

20

21

22

23

24 }

25

26 */\*\**

*/\*\**

* *header可以不写有默认值*
* *payload 通常⽤来存放⽤⼾信息*
* *signature 是前两个合起来的签名值*

*\*/*

String token = JWT.*create*().withHeader(map) *//header*

.withClaim("userId", 21)*//payload*

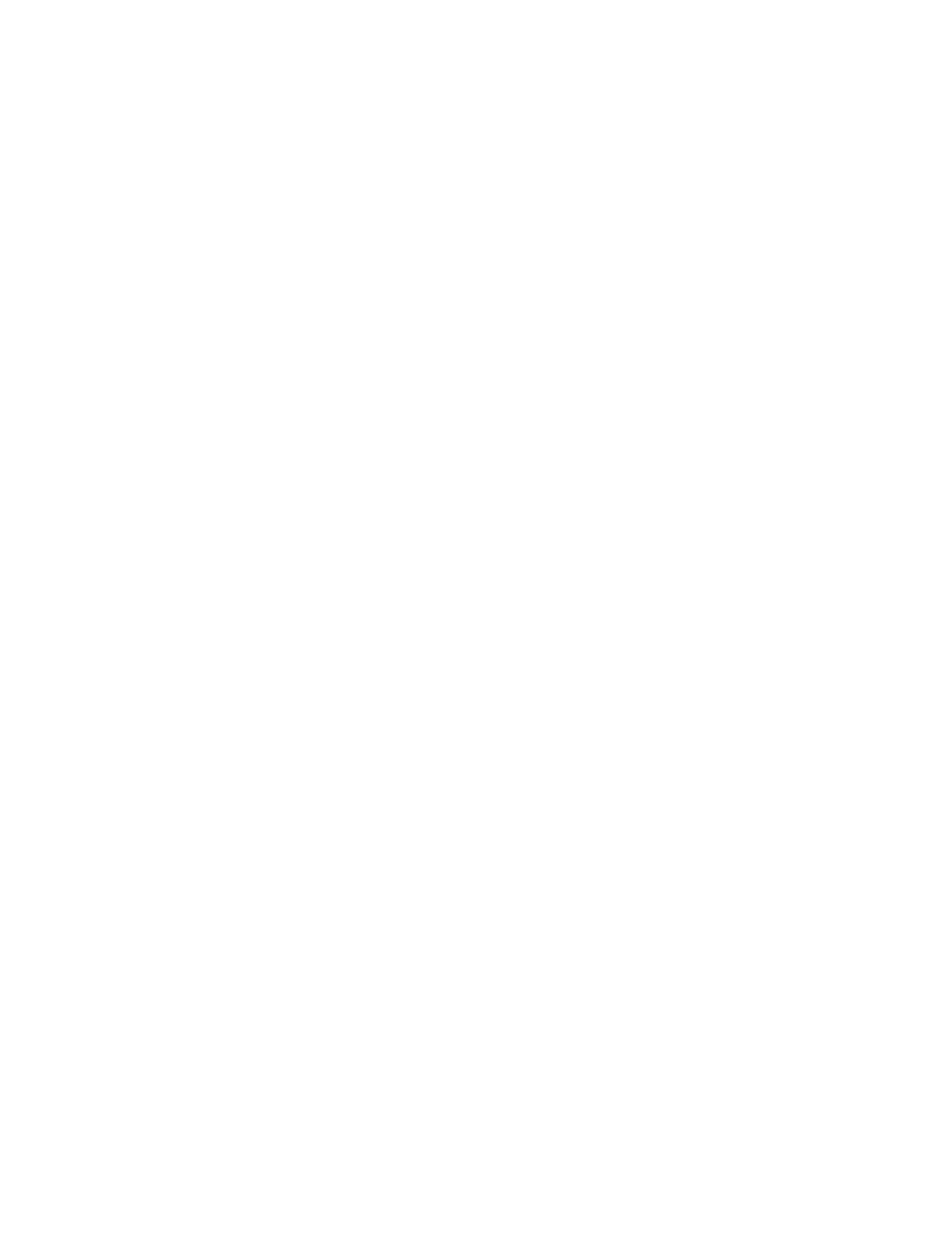
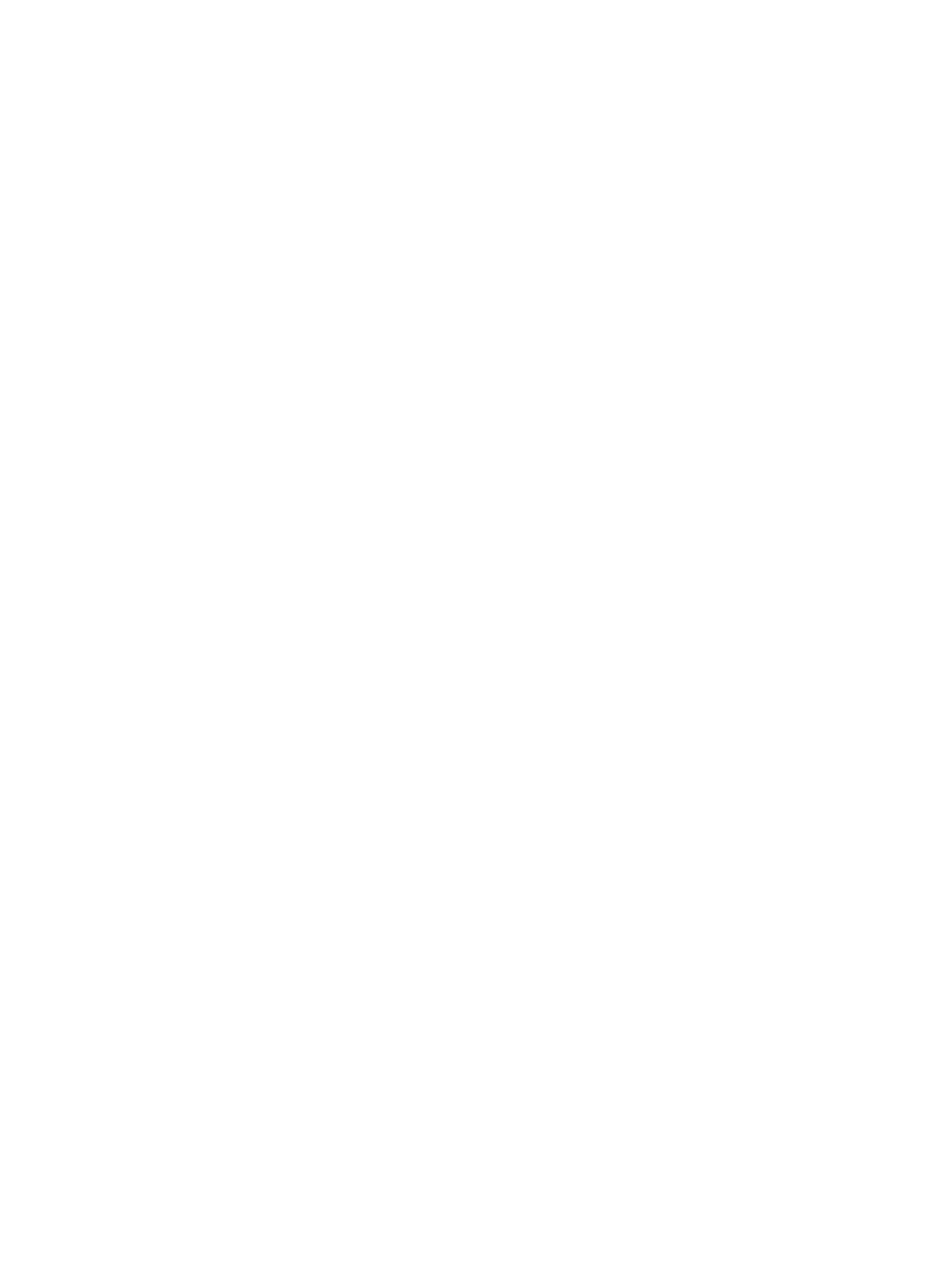
.withClaim("username", "yjiewei")*//payload*

.withExpiresAt(instance.getTime())*//指定令牌的过期时间*

.sign(Algorithm.*HMAC256*("!RHO4$%\*^fi$R")); *//签名，密钥⾃⼰记住*

System.*out*.println(token);

1. *\* 令牌验证:根据令牌和签名解析数据*



1. *\* 常⻅异常：*
2. *\* SignatureVerificationException 签名不⼀致异常*
3. *\* TokenExpiredException 令牌过期异常*
4. *\* AlgorithmMismatchException 算法不匹配异常*
5. *\* InvalidClaimException 失效的payload异常*

33 *\*/*

1. @Test
2. public void tokenVerify() {
3. *// token值传⼊做验证*
4. String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2MjkyMDg0NjgsInVzZXJJZCI6MjEsIn VzZXJuYW1lIjoieWppZXdlaSJ9.e4auZWkykZ2Hu8Q20toaks-4e62gerPlDEPHvhunCnQ";

38 */\*\**

1. *\* ⽤⼾Id：21*
2. *\* ⽤⼾名：yjiewei*

41 *\* 过期时间：Tue Aug 17 21:54:28 CST 2021*

42 *\*/*

1. JWTVerifier jwtVerifier = JWT.*require*(Algorithm.*HMAC256*("!RHO4$%\*^fi$R")).build();
2. DecodedJWT decodedJWT = jwtVerifier.verify(token); *// 验证并获取解码后的*

*token*

1. System.*out*.println("⽤⼾Id：" + decodedJWT.getClaim("userId").asInt());
2. System.*out*.println("⽤⼾名：" +

decodedJWT.getClaim("username").asString());

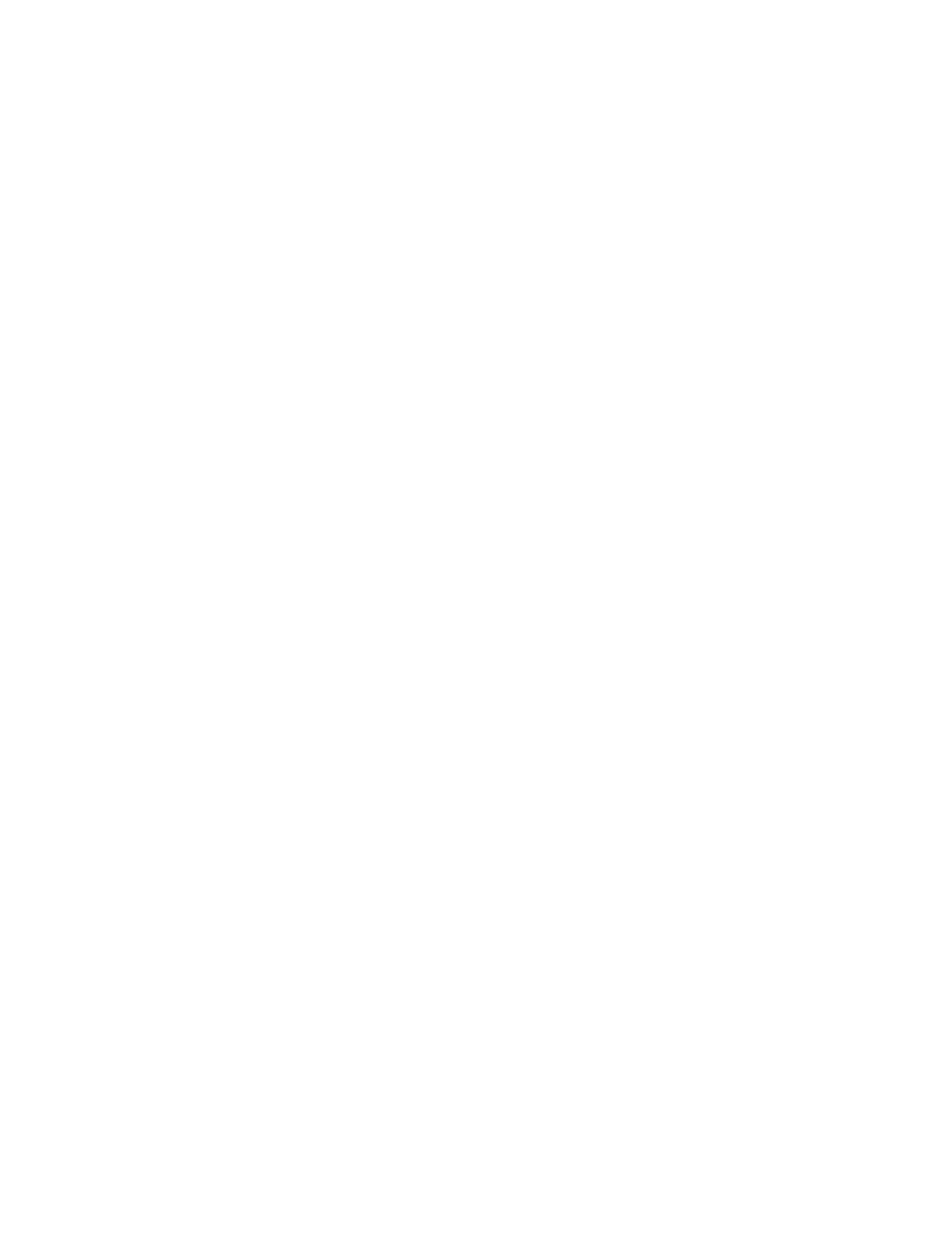
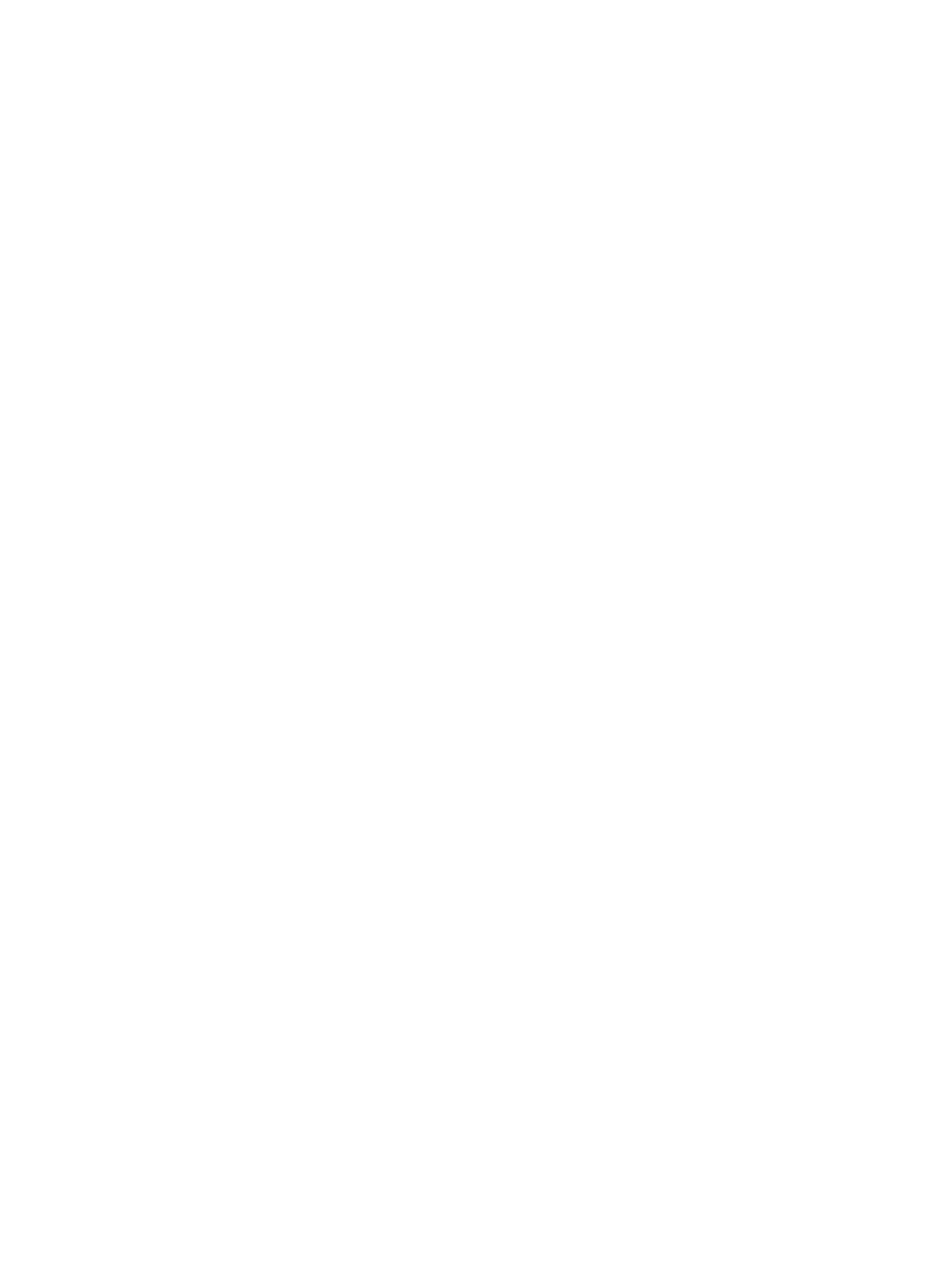
1. System.*out*.println("过期时间：" + decodedJWT.getExpiresAt()); 48 }

49 }

# 封装⼯具类

Java就是封装抽象封装抽象...所以这⾥也封装⼀个⼯具类。

##### TypeScript



1 public class JWTUtil { 2

3 */\*\**

4 *\* 密钥要⾃⼰保管好*

5 *\*/*

6 private static String *SECRET* = "privatekey#^&^%!save"; 7

8 */\*\**

1. *\* 传⼊payload信息获取token*
2. *\* @param map payload*
3. *\* @return token*

12 *\*/*

1. public static String getToken(Map<String, String> map) {
2. JWTCreator.Builder builder = JWT.*create*(); 15
3. *//payload*
4. map.forEach(builder::withClaim); 18
5. Calendar instance = Calendar.*getInstance*();
6. instance.add(Calendar.*DATE*, 3); *//默认3天过期*

21

1. builder.withExpiresAt(instance.getTime());*//指定令牌的过期时间*
2. return builder.sign(Algorithm.*HMAC256*(*SECRET*)); 24 }

25

26 */\*\**

27 *\* 验证token*

28 *\*/*

1. public static DecodedJWT verify(String token) {
2. *//如果有任何验证异常，此处都会抛出异常*
3. return JWT.*require*(Algorithm.*HMAC256*(*SECRET*)).build().verify(token); 32 }

33

34 */\*\**

35 *\* 获取token中的payload*

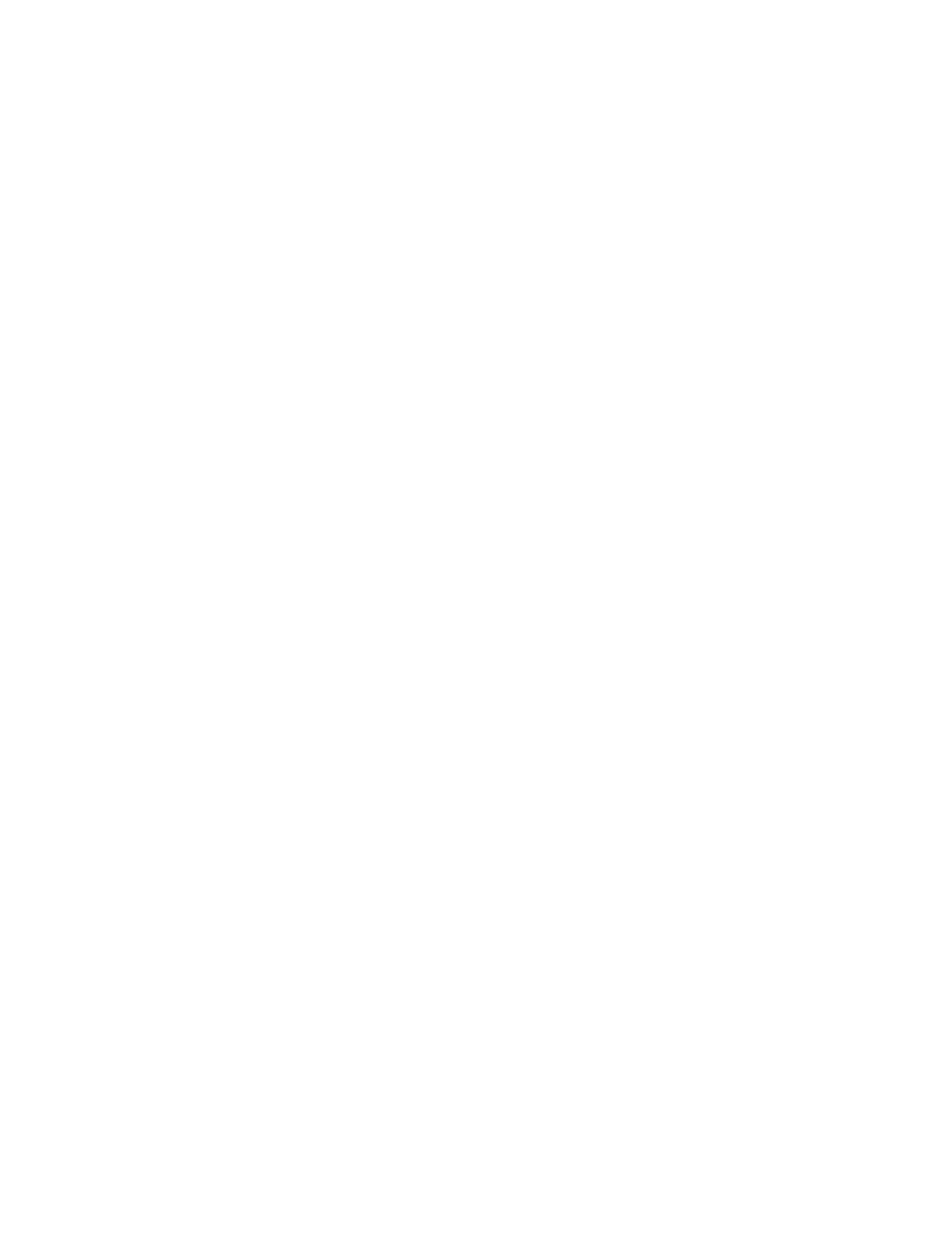
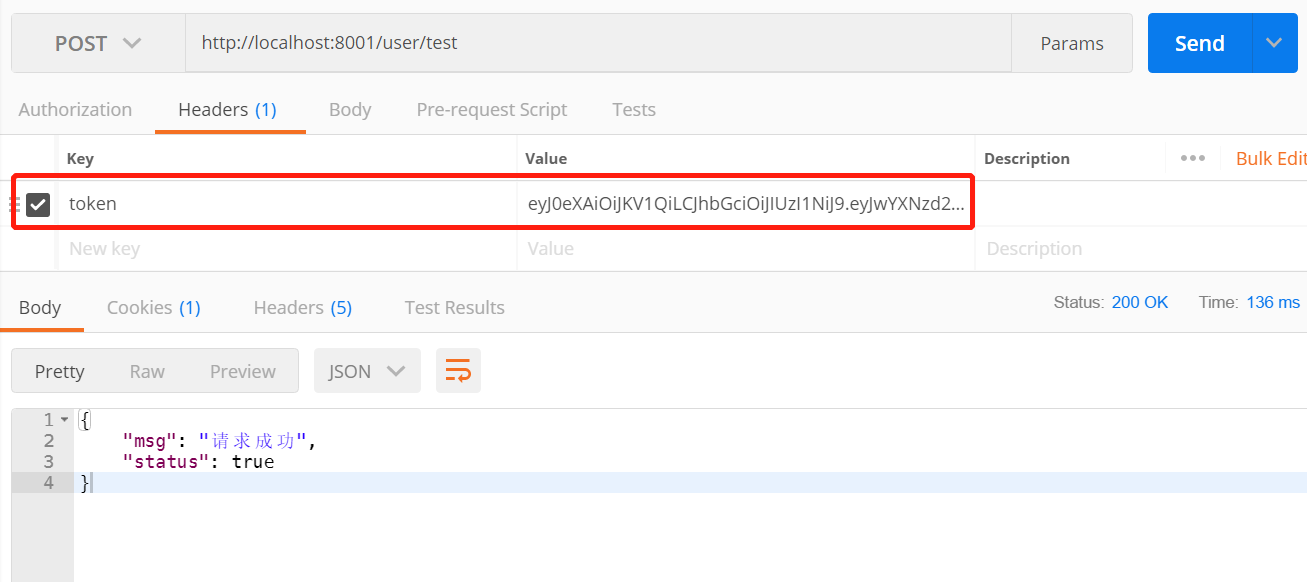
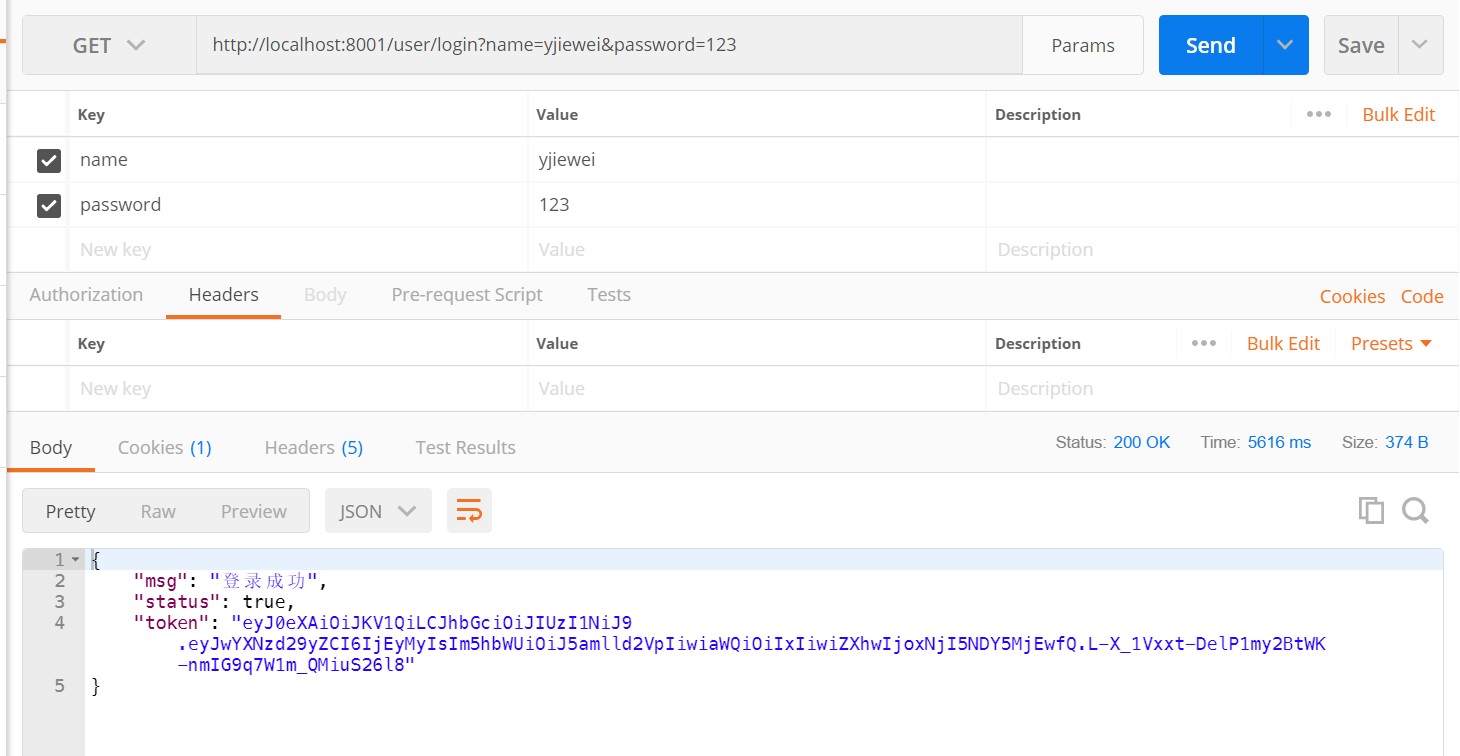
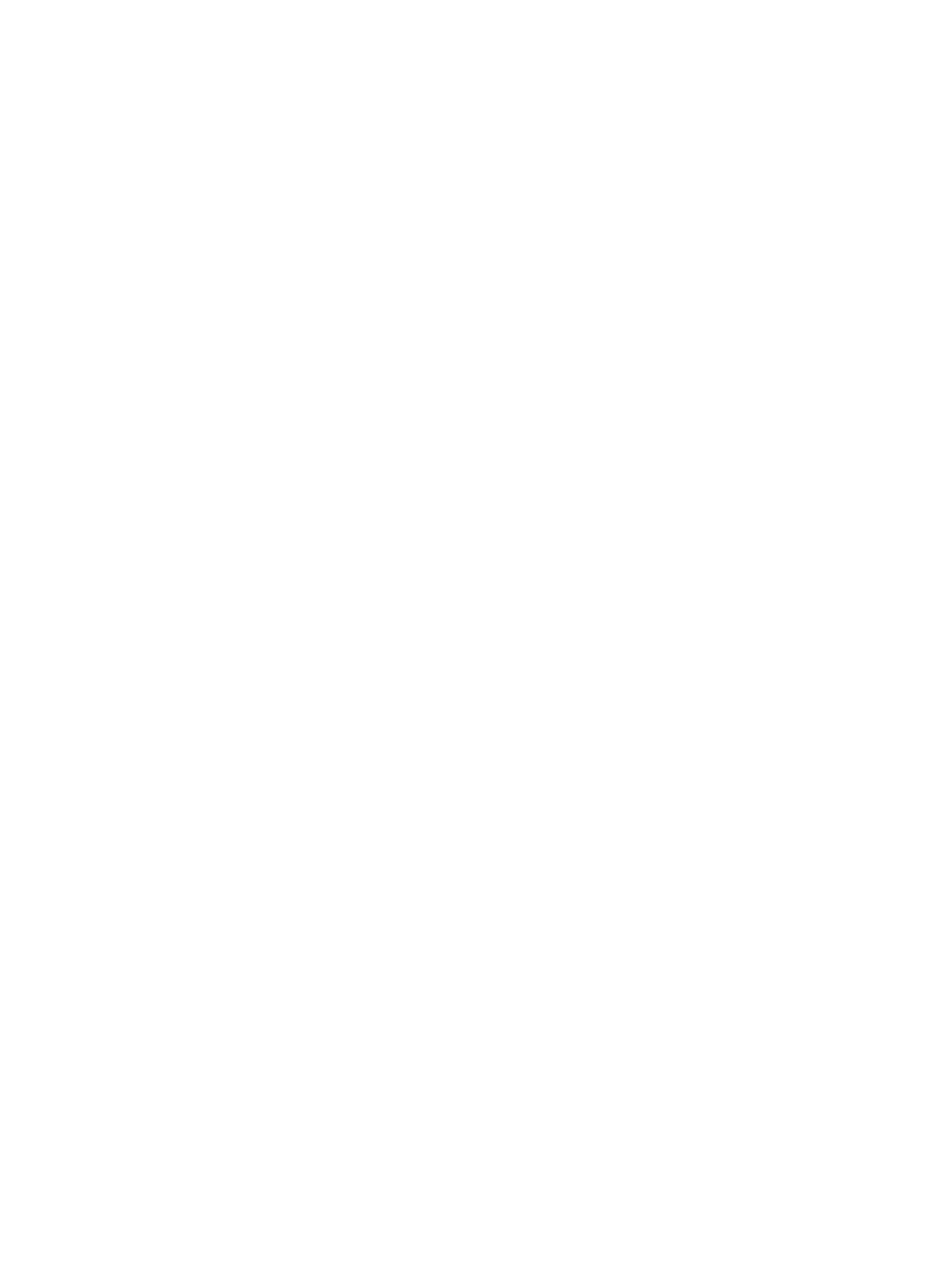
36 *\*/*

1. public static Map<String, Claim> getPayloadFromToken(String token) {
2. return

JWT.*require*(Algorithm.*HMAC256*(*SECRET*)).build().verify(token).getClaims(); 39 }

40 }

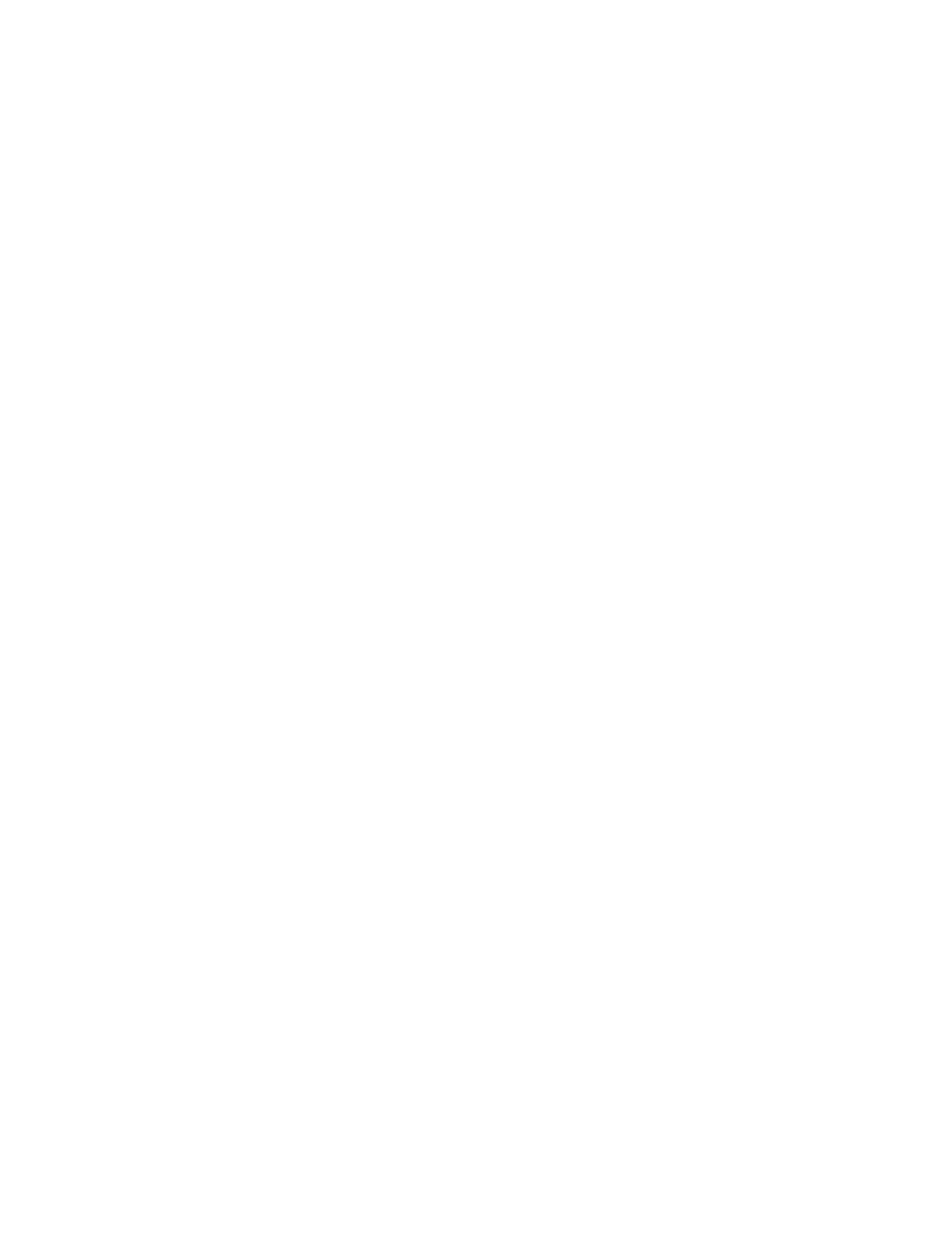
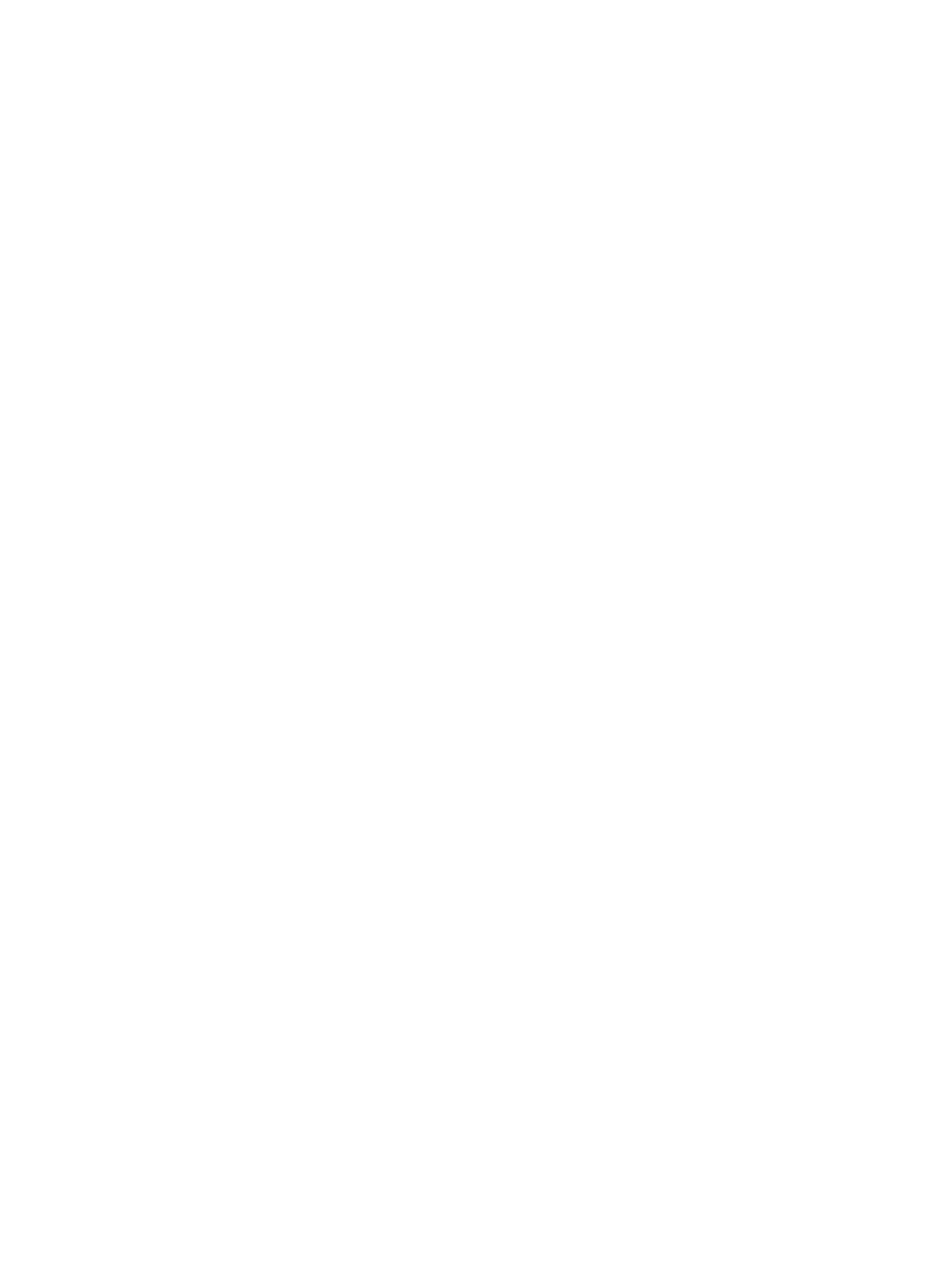
1. JWT 整合SpringBoot



我们要对每个请求都去验证实现单点登录，通过拦截器拦截请求对JWT做验证。

* 1. 登录并获取token
  2. 在请求头中添加token
  3. 重要的代码⽚段

##### TypeScript

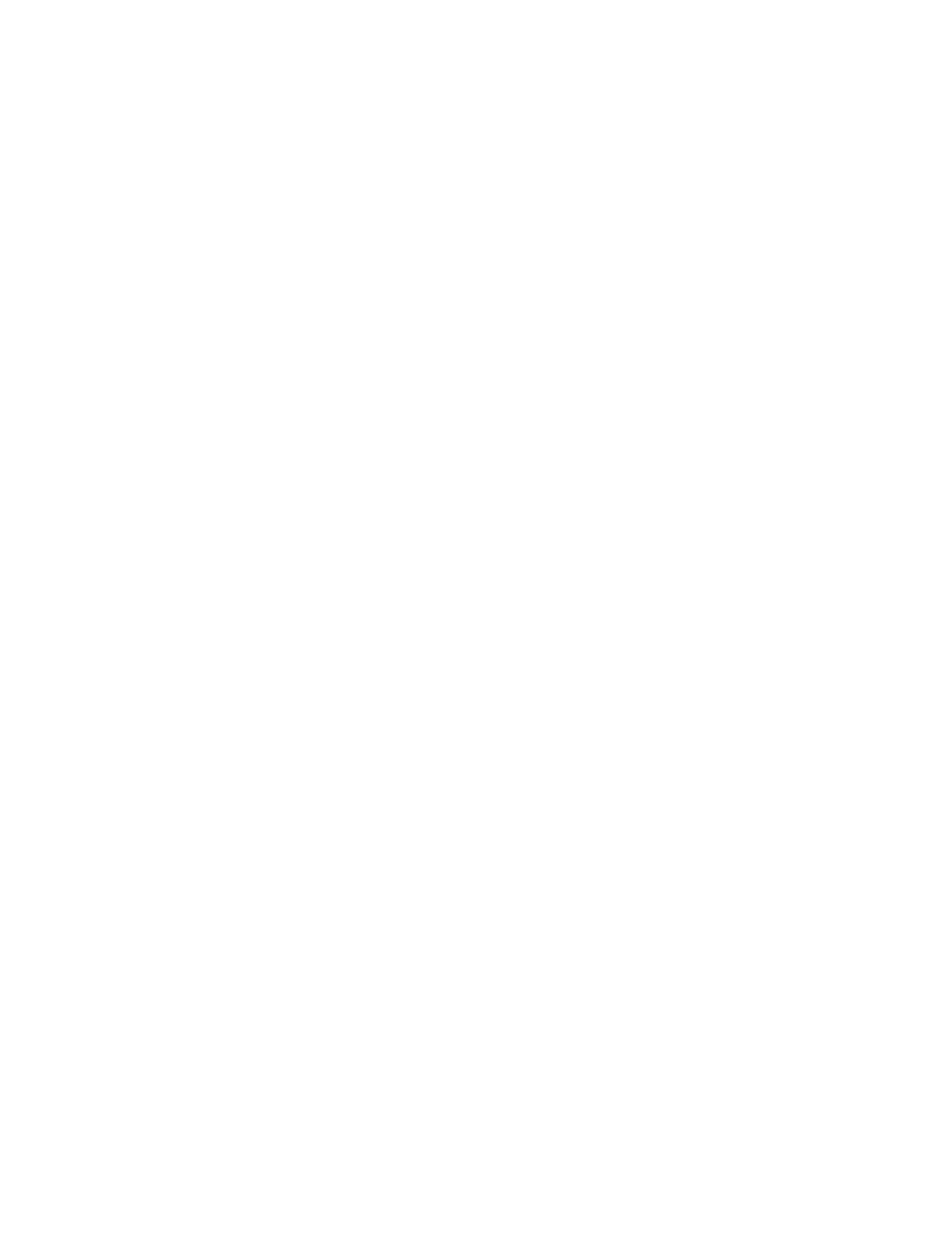


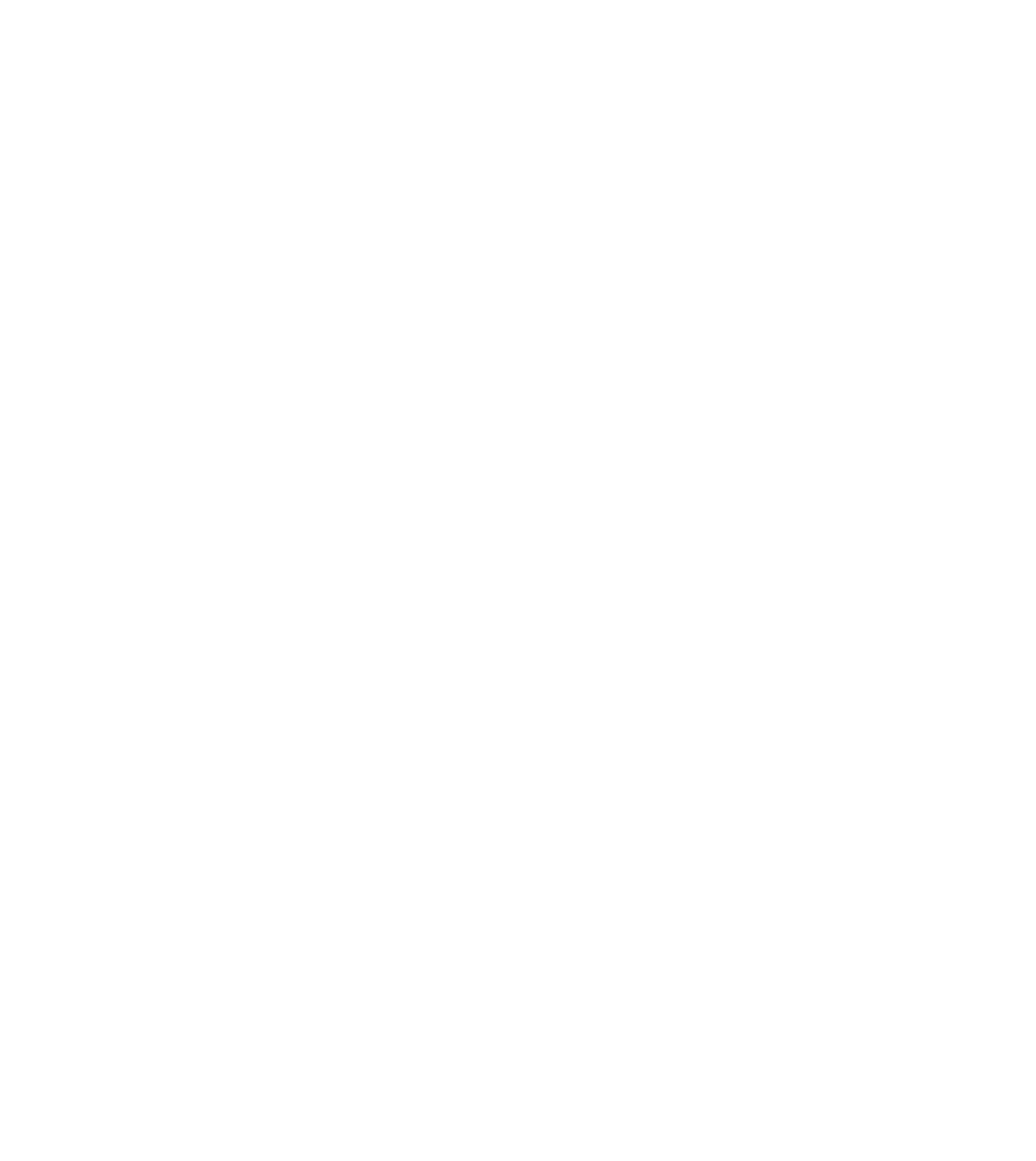
1. *// 拦截器*
2. @Slf4j
3. public class JWTInterceptor implements HandlerInterceptor { 4
4. @Override
5. public boolean preHandle(HttpServletRequest request,
6. HttpServletResponse response,
7. Object handler) throws Exception { 9
8. *//获取请求头中的令牌*
9. String token = request.getHeader("token");
10. [*log*.info](http://log.info/)("当前token为：{}", token); 13
11. Map<String, Object> map = new HashMap<>();
12. try {
13. JWTUtil.*verify*(token);
14. return true;
15. } catch (SignatureVerificationException e) {
16. e.printStackTrace();
17. map.put("msg", "签名不⼀致");
18. } catch (TokenExpiredException e) {
19. e.printStackTrace();
20. map.put("msg", "令牌过期");
21. } catch (AlgorithmMismatchException e) {
22. e.printStackTrace();
23. map.put("msg", "算法不匹配");
24. } catch (InvalidClaimException e) {
25. e.printStackTrace();
26. map.put("msg", "失效的payload");
27. } catch (Exception e) {
28. e.printStackTrace();
29. map.put("msg", "token⽆效"); 33 }

34

35 map.put("status", false); 36

1. *//响应到前台: 将map转为json*
2. String json = new ObjectMapper().writeValueAsString(map);
3. response.setContentType("application/json;charset=UTF-8");
4. response.getWriter().println(json);
5. return false; 42 }

43 }



##### Java

* 1. *// 指定拦截路径*
  2. @Configuration
  3. public class InterceptorConfig implements WebMvcConfigurer { 4

1. @Override
2. public void addInterceptors(InterceptorRegistry registry) {
3. registry.addInterceptor(new JWTInterceptor())
4. .addPathPatterns("/user/test")
5. .excludePathPatterns("/user/login"); 10 }

11 }

##### TypeScript

1. *// 拦截请求并验证请求头中的token*
2. @PostMapping("/user/test")
3. public Map<String, Object> test(HttpServletRequest request) {
4. String token = request.getHeader("token");
5. DecodedJWT verify = JWTUtil.*verify*(token);
6. String id = verify.getClaim("id").asString(); *// 我前⾯存的时候转字符串了*
7. String name = verify.getClaim("name").asString();
8. [*log*.info](http://log.info/)("⽤⼾id：{}", id);
9. [*log*.info](http://log.info/)("⽤⼾名: {}", name);

10

1. *//TODO:业务逻辑*
2. Map<String, Object> map = new HashMap<>();
3. map.put("status", true);
4. map.put("msg", "请求成功");
5. return map; 16 }