

# LU10a - Umgang mit Exceptions

## Spring ResponseStatusException

Allgemein gilt, dass Exceptions in Bezug auf die Performanz schlechter abschneiden als die entsprechenden Checks. Daher sollte man Exceptions nie in „Schönwetter“-Fällen benutzen, sondern höchstens in unerwarteten Situationen. Ein ungültiger API-Aufruf ist in unserem Fall aber unerwartet und kann mittels einer Exception gelöst werden. Werden aber viele ungültige API-Aufrufe erwartet, sollte man eine Exception-freie Alternative in Erwägung ziehen.

Spring bietet eine sehr einfache Version, um Fehlerfälle mittels Exceptions abzuhandeln. Dabei wird direkt automatisch eine Antwort mit dem entsprechenden HTTP-Statuscode erstellt und zurückgeschickt.

Im nachfolgenden Beispiel wird jeweils ein Project-Objekt geladen und falls kein Objekt in der Datenbank vorhanden ist, wird der Statuscode 404 zurückgegeben.

Lange Variante	Verkürzte Variante	Variante mit Exception
Optional<Project> optionalProject = projectRepository.findById(projectName); if(optionalProject.isEmpty()){ return ResponseEntity.notFound().build(); } Project project = optionalProject.get();	Project project = projectRepository.findById(projectName).orElse(null); if(null == project){ return ResponseEntity.notFound().build(); }	Project project = projectRepository.findById(projectName).orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "Project not found"));

Ein sehr grosser Vorteil der dritten Variante ist die Wiederverwendbarkeit: Ein ResponseEntity-Objekt kann nur in der API-Methode zurückgegeben werden. Eine ResponseStatusException kann aber auch in einer verschachtelten Methode geworfen werden. Das erlaubt, dass man Methoden, die man zum Beispiel bei mehreren API-Methoden benötigt, bequem auslagern kann.

## Validierung des JWT

In der vorgeschlagenen Klasse JwtUtil.java gibt es bereits eine Möglichkeit, um ein JWT zu prüfen (Siehe auch LU08.A05 Register/Login).

```
public boolean validateToken(String token) {  
    try {  
        Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);  
        return true;  
    } catch (Exception e) {  
        return false;  
    }  
}
```

### Kleine Wiederholung



Setzt man Exceptions ein, sollte man diese auch loggen oder weiterverschicken (gegebenenfalls sogar ans Frontend). Ein „catch“ ohne Verwendung der gefangenenen Exception, wie im



Beispiel, ist zu vermeiden.

Wenn nun zudem Informationen wie das „Subject“ aus dem JWT herausgelesen werden müssen, ergibt es keinen Sinn, eine separate „Exception“-basierte Validierung zu machen. Man kann die Validierung und das Auslesen direkt kombinieren. Nutzt man zudem die ResponseStatusException, hat man eine Methode, die man in mehreren API-Methoden einsetzen kann.

```

public String verifyTokenAndExtractSubject() {
    try{
        String token = extractTokenFromHeader();
        return extractSubject(token);
    } catch (Exception e) {
        // Exception mitschicken
        // throw new ResponseStatusException(HttpStatus.UNAUTHORIZED,
        "Invalid token", e);

        // Exception loggen
        log.warn("Invalid token", e);
        throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "Invalid
token");
    }
}

private String extractSubject(String token) {
    return Jwts.parserBuilder()
        .setSigningKey(key)
        .build()
        .parseClaimsJws(token)
        .getBody()
        .getSubject();
}

private String extractTokenFromHeader() {
    ServletRequestAttributes attributes = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
    if (attributes == null){
        return null;
    }
    String authHeader = attributes.getRequest().getHeader("Authorization");
    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
        return null;
    }
    return authHeader.substring(7);
}

```

From:  
<https://wiki.bzz.ch/> - **BZZ - Modulwiki**



Permanent link:  
<https://wiki.bzz.ch/de/modul/ffit/3-jahr/java/learningunits/lu10/a>

Last update: **2025/11/10 00:02**