LU10a - Umgang mit Exceptions

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 weiterschicken (gegebenenfalls sogar ans Frontend). Ein „catch“ ohne Verwendung der gefangenen Exception, wie im Beispiel, ist zu vermeiden.

Hierbei ist anzumerken, dass Exceptions in aller Regel 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 invalides Token, bzw. allgemein ein ungültiger API-Aufruf, ist in unserem Fall unerwartet und kann mittels Exceptions gelöst werden. Werden aber viele ungültige API-Aufrufe erwartet, sollte man eine Exception-freie Alternative in Erwägung ziehen.

Wenn zudem

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class DemoController {
    private static final Logger log =
LoggerFactory.getLogger(DemoController.class);
    ...
        log.error("Error during execution of...:", e);
```

Neu ist, dass wir die dazugehörigen Einstellungen einfach in unser `application.properties` schreiben können.

```
logging.level.root=INFO
logging.level.ch.bzz=DEBUG
logging.file.name=app.log
```

Project Lombok hilft uns auch hier die Erstellung des Logger-Objektes zu vereinfachen.

```
import lombok.extern.slf4j.Slf4j;

@Slf4j
public class DemoController {
    ...
        log.error("Error during execution of...:", e);
```