WIP: Webapplication

# Aufbau

* Spring Boot 2 Application mit Swagger (<https://localhost:9000/swagger-ui.html>)
* Komponentenbasierter Aufbau mit zentraler Konfiguration
* Entitäten mit JPA auf H2 Basis – evt. Einsatz von DTO oder Ersatz mit Jooq
* SSL Verschlüsselung mit Self Signed Certificate

# Idee Authentifizierung

Grundgedanke: Wer ist mein Client 🡪 Antwort:Webbrowser

* Webbrowser: Webbrowser sind unsicher und zentral gegen CSRF/XSS Angriffe zu schützen, da sie allen Content laden, denn sie bekommen (So beispielsweise die badbank.com, welche einen CSRF Angriff fährt). Webbrowser können aber Inhalte in Cookies und Local Storage speichern. Mögliche Authentifizierungen: Stetige Basic Authentication, initiale Basic Authentication mit sicherer Session, JWT/Oauth2 in Local Storage oder Cookie
* Backendsysteme (z.B. Java to Java): Sicherer, da sie nicht blindlings wie ein Browser konsumieren und ihm folgen. Mögliche Authentifizierungen: Basic Authentication
* Mobile Systeme/Applikationen: Können schlecht oder gar nicht mit Session/Cookies umgehen. Deshalb weicht man auf stateless Tokentechnologien wie OAuth2/JWT aus und speichert das Token persistent in der Applikation

Wir beziehen uns hier auf Webbrowser und die möglichen Technologien:

* Stetige Basic Authentication
  + Einfach
  + Unsicher, da das Passwort stetig mitgeschickt wird und so auch irgendwo gespeichert werden muss
* Initiale Basic Authentication + mit weiterführender Session
  + Initial wird der Username und Passwort via Basic Authentication an den Server geschickt (<https://de.wikipedia.org/wiki/HTTP-Authentifizierung>). Verschlüsselung des Transportlayers wird wie überall auch vorausgesetzt
  + Der Server nimmt die Credentials entgegen, prüft sie und erstellt dem Client ein Session. Die Session ID alias JSESSIONID speichert er via httpOnly (<https://wiki.selfhtml.org/wiki/Grundlagen/sichere_Cookies#HttpOnly>) im Client (Der Client schickt dieses Cookie immer mit, aber es kann nicht via JavaScript ausgelesen werden 🡪 XSS Schutz)
  + Da wir mit einer Session und Cookies arbeiten, müssen wir uns auch gleich um einen CSRF Schutz kümmern. Spring Boot bietet hier die Möglichkeit an, das CSRF Token nicht im Header dem Client zu liefern, sondern es im JavaScript lesbar im Cookie zu speichern (<https://jaxenter.de/angular-csrf-spring-boot-52223>). Tätigt der Client einen API Zugriff, z.B. via fetch API, nimmt er das CSRF Token aus dem Cookie und hängt es dem fetch Header an – der Server liesst es dann aus dem Header und prüft es
  + Möchte man eine Vebrindung terminieren, loggt man sich via POST auf den Logout-Endpoint aus oder lässt via Admininterface die Session vom Server löschen. Client kann so nicht mehr zugreifen
* JWT mit initialer Authentication + Tokenspeicherung in Local Storage
  + Local Storage bietet keinen Schutz vor XSS Angriffen 🡪 Jeder kann das JWT Token klauen und je nach System auch beliebig oft verlängern 🡪 Keine Revocation möglich ohne als Secret ein automatisch generiertes Secret mit Serverneustart zu verwenden
  + JWT sollte nur für kurze Zeitspannen (Maximal 5 Minuten verwendet werden)
* JWT mit initialer Authentication + Tokenspeicherung in Cookie
  + Cookie ohne httpOnly bietet keinen Schutz vor XSS Angriffen 🡪 Jeder kann das JWT Token klauen und je nach System auch beliebig oft verlängern 🡪 Keine Revocation möglich ohne als Secret ein automatisch generiertes Secret mit Serverneustart zu verwenden
  + JWT sollte nur für kurze Zeitspannen (Maximal 5 Minuten verwendet werden)
* OAuth2/JWT mit initialer Authentication + Tokenspeicherung in sicherem Cookie (Sicher = httpOnly, sprich nicht für die JavaScript Engine zugänglich)
  + Naja, jetzt erfinden wir wieder die Initiale Basic Authentication mit weiterführender Session, warum also nicht gleich das verwenden

# PKCS12 Keystore generieren

* Passwort ist 123456aA
* Grundlage: <https://code-adda.com/2018/06/enable-https-with-self-signed-certificate-in-spring-boot/>
* keytool -genkey -alias tomcatssl -keyalg RSA -keysize 2048 -keystore keystore.jks -validity 3650
* Danach die JKS Datei unter src/main/resources ablegen
* Danach die Datei src/main/resources/application.properties anpassen

# Verbesserungen

* Authentifizierung anhand eines Integration Tests zeigen und testen
* OSIV Pattern durch DTOs lösen (oder gleich auf Jooq wechseln)