Zeitplanung wodss

# Gruppe

Unsere Gruppe besteht aus:

* Thibault Gagnieux
* Philipp Lüthi
* Simon Wächter

# Verwendete Technologien

* Frontend
  + Preact, um auf eine leichtgewichtige React.js Alternative zu setzen. Zumal diese über keine Lizenzprobleme wie React.js verfügt: <https://medium.freecodecamp.org/facebook-just-changed-the-license-on-react-heres-a-2-minute-explanation-why-5878478913b2>
* Backend
  + Spring Boot mit Java zur Realisierung des Webservers
  + jOOQ als Abstraktionslayer zu SQL (Basis: PostgreSQL Server)
  + Eventuell MapStruct zum Mappen der DTO/Entitäten
  + Springfox zum Dokumentieren und Anbieten eines Swagger Interfaces
* Building
  + Gradle mit Java 11
* Deployment
  + Docker Compose mit drei Containern (Frontend, Backend, Datenbankserver)

# Beschreibung der Schnittstelle

Basierend auf den initialen Arbeiten von David und Christian hat Simon die API erweitert und möchte über diese am 18. März abstimmen lassen. Der Kundenwunsch von Herrn König zum Integrieren von Zeitpensen wurde umgesetzt. Spezifikation: <https://github.com/swaechter/fhnw-wodss-spec>

Momentan noch nicht vorhanden ist ein Testdatenset, welches die Integration vereinfacht

# Beschreibung Authentifizierung

Die Schnittstelle basiert auf dem JWT Mechanismus, welcher wie folgt abläuft:

1. Client besitzt noch keine Authentifizierung
2. Client steuert POST /api/token mit einer Emailadresse samt Passwort als Request Parameter an
3. Der Server verifiziert diese Informationen und stellt ihm ein JWT Token in der Response aus. In diesem JWT Token ist der ganze Mitarbeiter als Claim «employee» integriert (Siehe Grafik unten)
4. Der Client speichert dieses Token ab (Local Storage)

Der Aufruf einer geschützten Schnittstelle läuft wie folgt ab:

1. Der Client liest das Token aus dem Local Storage
2. Der Client schickt das Token als HTTP Header «Authorization» im Format «Bearer TOKEN» in der jeweiligen Anfrage mit
3. Der Server validiert via Filter die Signatur des Tokens und lässt dementsprechend den Zugriff zu

Da Tokens über ein Ablaufdatum verfügen, müssen diese von Zeit zu Zeit erneuert werden:

1. Der Client liest das Token aus dem Local Storage
2. Der Client schickt das Token als HTTP Header «Authorization» im Format «Bearer TOKEN» an POST /api/token/{TOKEN}
3. Der Server verifiziert diese Informationen und stellt ihm ein neues JWT Token in der Response aus (Beim Refreshen wird im Gegensatz zum Austellen die Emailadresse/Passwort nicht mehr benötigt, da wir nur die Signatur prüfen müssen). In diesem JWT Token ist der ganze Mitarbeiter als Claim «emplyee» integriert (Siehe Grafik unten)
4. Der Client speichert dieses neue Token ab (Local Storage) und verwirft das alte

# Beispielaufbau eines Tokens



* Erster Block: Head mit dem Signierungsalgorithmus HS512
* Zweiter Block: Body mit dem Issuer (iss), dem Subject (sub) sowie dem Ausstelldatum (iat) und Ablaufdatum (exp). Zudem wird der ganze Benutzer im Claim «employee» gespeichert
* Dritter Block (Gesnipped): Signatur des ersten und zweiten Blockes zur Validierung

# Verwendung in Swagger

Da das Projekt mit Springfox dokumentiert ist, verfügt das Backend über einen integrierten Swagger Client. Um auf die geschützten Schnittstellen zuzugreifen, muss wie folgt vorgegangen werden:

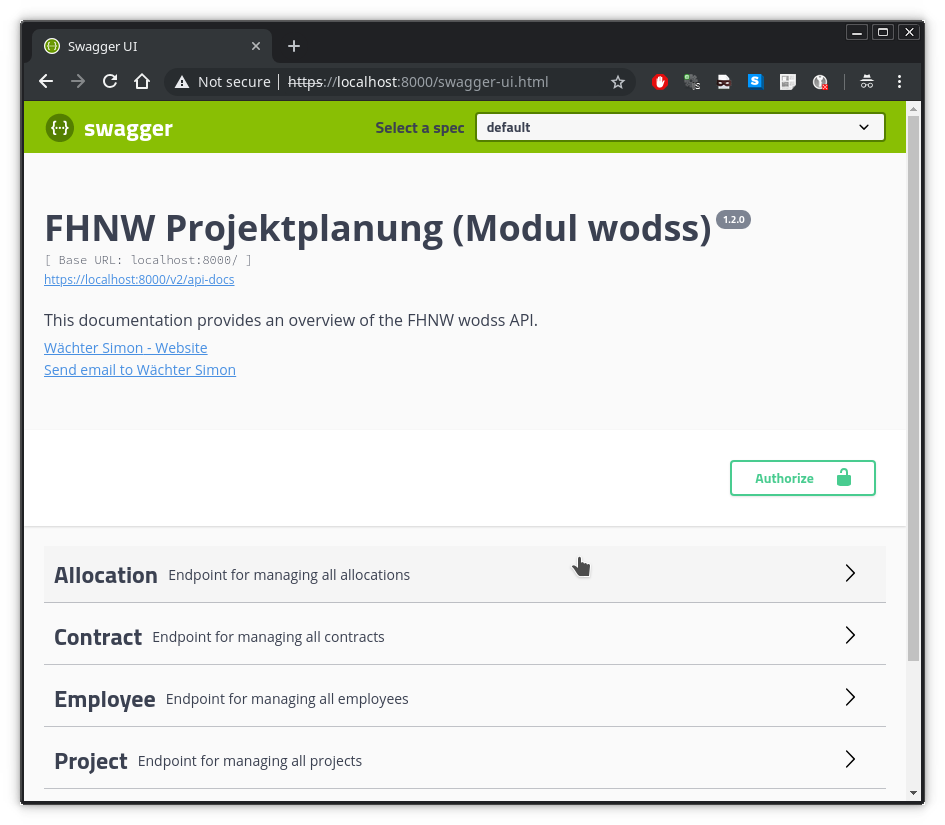
## Schritt 1: Selbstsigniertes Zertifikat neu erstellen

Das Projekt läuft standardmässig auf Port 8000 via HTTPS-only. Das selbsterstellte Zertifikat in src/main/resources/keystore.pkcs12 verfügt über ein schwaches Passwort und ist in eine Versionsverwaltung mit mehreren Teammitgliedern eingecheckt – heisst: **Es muss als unsicher angesehen und neu generiert werden:**

* Grundlage: <https://www.baeldung.com/spring-boot-https-self-signed-certificate>
* <Path to Java>/bin/keytool -genkeypair -alias keystore -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore keystore.p12 -validity 3650
* Den Keystore unter config/ ablegen
* Die Datei config/application.properties.template zu application.properties umbenennen.
* Das Keystorepasswort in der Datei config/application.properties anpassen

## Schritt 2: Öffnen des Swagger Clients

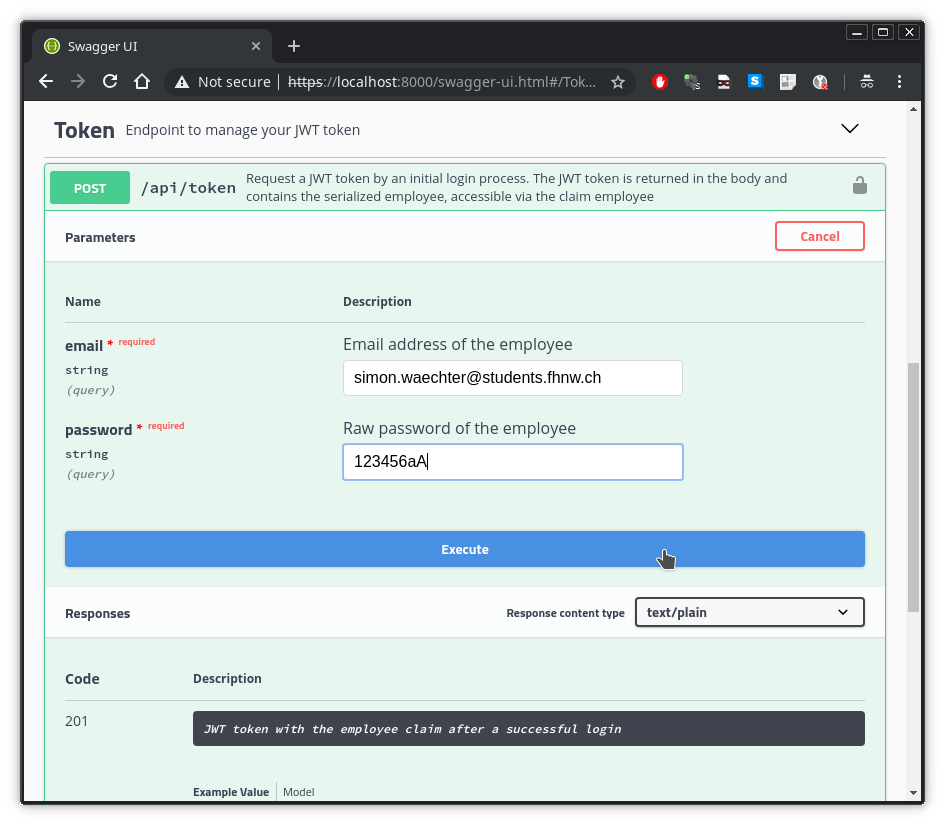
Via Browser muss der Endpoint <https://localhost:8000/swager-ui.html> angesprochen werden:



## Schritt 3: Anmelden und JWT Token generieren lassen

**Hinweis**: Für das Anmelden wird am besten der integrierte Administratorenaccount verwendet, da ansonsten der neu erstellte Benutzer zuerst aktiviert werden muss

Um sich am System anzumelden und ein JWT Token zu erhalten, muss der Endpoint /api/token via POST und den Credentials eines Mitarbeiters aufgerufen werden (Zuerst auf «Try it out» klicken, dann die Daten eingeben und schlussendlich auf «Execute» klicken:



Nach dem Ausführen der Anfrage wird das JWT Token in der Response zurückgesendet. Es muss wegkopiert werden:

## Schritt 4: JWT Token in Swagger setzen

Nach dem Kopieren des Tokens muss dieses in Swagger integriert werden, sodass geschützte Endpoints aufgerufen werden können. Dazu klickt man oben rechts auf «Authorize» und gibt im Feld «Bearer TOKEN» ein (TOKEN mit dem tatsächlichen Token ersetzten) und bestätigt die Aktion. Swagger schickt ab jetzt dieses Token mit.

