

# Workshop FHNW wodss Dokumentation

**Projektname:** Workshop FHNW wodss  
**Autoren:** Thibault Gagnaux, Philipp Lüthi und Simon Wächter  
**Version:** 0.0.4

## Dokumentenmanagement

Version: 0.0.4  
Datum: 04.05.2019  
Autoren: Thibault Gagnaux, Philipp Lüthi und Simon Wächter  
Dokumentname: Dokumentation.docx  
Klassifizierung: Keine Klassifizierung

## Änderungen

Version	Datum	Beschreibung	Autor
0.0.1	17.03.2019	Initiale Dokumentation mit der Beschreibung des Technologiestacks	Simon Wächter
0.0.2	04.04.2019	Wechsel auf Docker und Heroku Deployment	Simon Wächter
0.0.3	17.0.2019	Wechsel auf eine zentrale Konfigurationsmöglichkeit	Simon Wächter
0.0.4	04.05.2019	Hinzufügen des Projektlayouts und der Frontend-dokumentation	Simon Wächter

## Inhaltsverzeichnis

<b>1</b>	<b>Übersicht.....</b>	<b>4</b>
1.1	Gruppe .....	4
1.2	Verwendete Technologien .....	4
1.3	Beschreibung der Schnittstelle .....	4
1.4	Beschreibung Authentifizierung .....	4
<b>2</b>	<b>Inbetriebnahme des Back- und Frontend .....</b>	<b>5</b>
2.1	Überlegungen und Möglichkeiten .....	5
2.2	Variante 1: Verwendung der durch uns gehosteten Heroku Lösung .....	5
2.3	Variante 2: Eigenes Heroku Deployment .....	5
<b>3</b>	<b>Inbetriebnahme des Frontends .....</b>	<b>8</b>
3.1	Klonen des Projektes.....	8
3.2	Anpassen der Backend URL .....	8
3.3	Bauen des Frontends .....	8
3.4	Aufbau.....	8
3.5	Hinweise.....	9

## 1 Übersicht

### 1.1 Gruppe

Unsere Gruppe besteht aus:

- Thibault Gagnaux
- Philipp Lüthi
- Simon Wächter

### 1.2 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>
  - Redux mit Thunk Middleware
- Backend
  - Spring Boot mit Java zur Realisierung des Webserver
  - jOOQ als Abstraktionslayer zu SQL (Basis: PostgreSQL Server)
  - MapStruct zum Mappen der DTO/Entitäten
  - Springfox zum Dokumentieren und Anbieten eines Swagger Interfaces
- Building
  - Gradle mit Java 11
- Deployment
  - Docker Container mit Deployment auf Heroku

### 1.3 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

### 1.4 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
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

## 2 Inbetriebnahme des Back- und Frontend

### 2.1 Überlegungen und Möglichkeiten

Wir haben uns entschlossen, das Projekt via Heroku als Docker Applikation zu betreiben und für andere zugänglich zu machen. Dies resultiert in 2 möglichen Verwendungszwecken:

1. Verwendung unseres Backends unter <https://fhnw-wodss.herokuapp.com> (**Hinweis:** Die Applikation wird im Free-Tier Modell deployt und bei Nichtverwendung schlafen gelegt. Bei einem späteren Aufruf dauert das Wecken bis zu einer Minute)
2. Eigenes Deployment eines Docker Container via Heroku (Nachfolgend genauer beschrieben)

### 2.2 Variante 1: Verwendung der durch uns gehosteten Heroku Lösung

- Link zur Webapplikation: <https://fhnw-wodss.herokuapp.com>
- Link zum Swagger Interface: <https://fhnw-wodss.herokuapp.com/swagger-ui.html>

### 2.3 Variante 2: Eigenes Heroku Deployment

Vor dem eigentlichen Beginn muss ein Heroku Account erstellt und die Heroku CLI heruntergeladen werden: <https://devcenter.heroku.com/articles/heroku-cli>

Nach der Installation meldet man sich lokal an:

Anmelden an der Heroku CLI
heroku login -i

Für das Bauen und Hochladen der containerisierten Applikation muss ferner Docker installiert werden.

Nach den Vorbereitungen und einem Git Clone (<http://github.com/swaechter/fhnw-wodss>) wechselt man in das Projektverzeichnis und erstellt eine neue Heroku Applikation. Der Name „fhnw-wodss“ muss dabei durch einen anderen Namen ersetzt werden, da dieser schon belegt ist (z.B. fhnw-wodss-john):

Erstellen einer neuen Heroku Applikation
heroku create fhnw-wodss

Der erstellten Applikation soll jetzt auch eine PostgreSQL Instanz angehängt werden:

Anhängen einer PostgreSQL Instanz
heroku addons:create heroku-postgresql:hobby-dev

Nach dem Erstellen der PostgreSQL Instanz müssen wir deren Credentials anzeigen und aufsplitten:

Anzeigen der PostgreSQL Credentials
heroku config

Der Aufbau der URL ist wie folgt:

postgres://**DATENBANKBENUTZER:DATENBANKPASSWORT@DATENBANKHOST:5432/DATENBANKNAME**

Da unser Build und die Applikation diese Credentials benötigen, kopieren wir das Template «config/application.properties.template» nach «config/application.properties» und tragen diese dort ein. Unser Buildsystem als auch die Applikation selber verwenden untereinander unterschiedliche & inkompatible URL Formate, weshalb die URL zwei Mal zu konfigurieren ist (aber in einem unterschiedlichen Format):

- postgresql statt jdbc:postgresql
- Integration von Benutzername und Passwort in den Link

Eine fiktive Beispielkonfiguration sieht wie folgt aus:

Fiktive Konfiguration config/application.properties
<pre># Runtime settings for our Spring application (Required for runtime and development) PORT=8000 DATABASE_URL=postgres://uvyrwlvnktglsa:44ce2bd0901d80f3be93968c0552e59ea29ddc3a284cbd7edaa20ed224ad1eb4@ec2-54-221-113-7.compute-2.amazonaws.com:5432/d6hg874i8q0qau  # Build time settings for the Jooq SQL table generation + Flyway data migration (Required for development only) JDBC_DATABASE_URL=jdbc:postgresql://ec2-54-221-113-7.compute-2.amazonaws.com:5432/d6hg874i8q0qau JDBC_DATABASE_USER=uvyrwlvnktglsa JDBC_DATABASE_PASSWORD=44ce2bd0901d80f3be93968c0552e59ea29ddc3a284cbd7edaa20ed224ad1eb4</pre>

Nach dem Setzen der Credentials kann die Applikation in Docker gebaut und hochgeladen werden (Dieser Prozess dauert beim ersten Mal circa 5 Minuten und danach circa 1 Minute):

Bauen und Hochladen der Applikation
heroku container:login heroku container:push web heroku container:release web

Nach dem Hochladen muss auf eine Applikationsinstanz hochskaliert werden. Das Starten der ersten Instanz kann dabei gleich beobachtet werden:

## Setzen der Skalierung

```
heroku ps:scale web=1  
heroku logs --tail
```

Die Applikation kann nach dem Start auch direkt im Browser geöffnet werden:

## Öffnen der Applikation in einem Browser

```
heroku open
```

### 3 Inbetriebnahme des Frontends

Diese Kapitel setzt sich mit der Inbetriebnahme des Frontends auseinander und ist für die integrierende Gruppe wichtig.

#### 3.1 Klonen des Projektes

Das Frontend befindet sich im Verzeichnis **src/main/javascript** unseres Projektes, welche zuerst einmal geklont werden muss:

Klonen des Projektes
git clone https://github.com/swaechter/fhnw-wodss.git cd src/main/javascript

#### 3.2 Anpassen der Backend URL

Nach dem Klonen muss die URL zum Backend in der Datei **src/main/javascript/src/services/config.js** angepasst werden (**Wichtig:** Die URL muss mit einem Slash enden)

#### 3.3 Bauen des Frontends

Das Frontend kann wie folgt gebaut werden:

Bauen des Projektes
npm install npm run build

Der Output des Builds landet dabei im Verzeichnis **src/main/resources/public** und würde bei einem regulären Build gleich in die Applikation integriert werden (Spring Boot Applikation mit SPA Applikation und REST Backend). Das Verzeichnis kann in **src/main/javascript/package.json** im Skriptbefehl **build** angepasst werden

Möchte man das Frontend im Development Modus mit Live Reload starten, kann ein **npm run watch** ausgeführt werden.

#### 3.4 Aufbau

Die Applikation im Verzeichnis **src/main/javascript** ist wie folgt aufgebaut:

Verzeichnis	Beschreibung
src/actions src/reducers	Enthält alle Redux Actions & Reducers. Ursprünglich wurde das Adminfrontend separat entwickelt, weshalb es noch gewisse Admin Action & Reducers übrig sind
src/components	Enthält wiederverwendbare Komponenten wie Fehlerdialoge, Navigation und Locks (Locks enforcen eine Aktion, z.B. dass jemand angemeldet sein muss oder andernfalls eine Fehlermeldung/Redirect erhält)
src/pages	Enthält alle einzelnen gerouteten Seiten
src/services	Enthält alle Services mit ihren jeweiligen Funktionen, z.B. zum Anmelden
src/utlis	Enthält reine Utilityfunktionen, z.B. für Umrechnungen von Daten



## 3.5 Hinweise

Folgende Punkte sind noch erwähnenswert

- Aufgrund der Verwendung von `async & await` muss unser Projekt die experimentelle Preact CLI in Version 3 verwenden. Die stabile Version 2 ist inkompatibel
- Das Token muss den Claim „employee“ besitzen, ansonsten kann die Authentifizierung nicht erfolgen → Das Token wird im Local Storage gespeichert
- Wir haben in unserem Backend CORS Preflights ignoriert, sprich alles akzeptiert → Das Frontend wurde nicht mit striktem CORS getestet
- Benutzer können sich an unserem Backend registrieren und dann freischalten lassen → POST `/api/employee` sehen wir als ungeschützten Endpoint an