Workshop FHNW wodss Dokumentation

Projektname: Workshop FHNW wodss
Autoren: Thibault Gagnieux, Philip Lüthi und Simon Wächter
Version: 0.0.2

Dokumentenmanagement

Version:	0.0.2
Datum:	03.04.2019
Autoren:	Anastasiia Graftceva, Angelica Marra und Simon Wächter
Dokumentname:	Teamübung.docx
Klassifizierung	Firmenintern + Freigabe für Dozent (Louis-Paul Wicki) samt FHNW Gruppenmitglieder (Anastasiia Graftceva und Angelica Marra)

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

Inhaltsverzeichnis

1 Übersicht

1.1 Gruppe

Unsere Gruppe besteht aus:

- Thibault Gagnieux
- 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
- Backend
  - Spring Boot mit Java zur Realisierung des Webservers
  - 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 (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

2 Inbetriebnahme

2.1 Überlegung

Wir möchten das Deployment unseres Projektes für andere Gruppen so einfach wie möglich gestalten und dabei die auf den jeweiligen Plattformen (Windows, OS, Linux) auftretenden Probleme weitgehend vermeiden. Die einzig vernünftige Lösung dafür ist, von vorhinein auf Docker zu setzen und die Applikation containerbasiert bei einem SaaS Provider laufen zu lassen. Wir haben uns für Heroku entschieden, da man eine Instanz samt PostgreSQL Datenbank kostenlos verwenden kann.

2.2 Vorbereitung

Vor dem eigentlichen Beginn muss deshalb 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 auch Docker installiert werden.

2.3 Erstellen und Deployen einer Instanz

Nach dem Anmelden 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@DATENBANKURL**

Da unser Build diese Credentials benötigt, müsse sie in den Bereicht «**ext**» in der «**build.gradle**» eingetragen werden. Beispiel:

```
buildscript {
    ext {
        databaseUrl = 'jdbc:postgresql://ec2-23-19-136-232.compute-1.amazonaws.com:5432/d9klsnvh3v0cbs'
        databaseUser = 'suthhpumelhsxz'
        databasePassword = '3db09f1a079fjcefad572a3jeecc07a8061be9ebde9faj5589aa585b610e4252'
    }
    // Snipped
}
```

Hinweis: Die einzelnen drei Komponenten müssen getrennt kopiert werden, da der Heroku Link anders als ein JDBC Link aufgebaut ist:

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

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

Bauen und Hochladen der Applikation
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:

Sind die Credentials angepasst, wird die Skalierung der Applikation auf 1 gesetzt:

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