Leistungsbeurteilung 151 Datenbanken in Web-Applikationen einbinden

Basierend auf LBV 151-1 seitens Ecole des Métiers Fribourg (EMF Informatique)

Zusatzaufgabe React (12 Punkte, geschätzter Aufwand 12 Stunden)

Vorliegende Aufgabe sieht die Implementation eines Frontends mithilfe der Bibliothek React vor. Dies als Gegenstück zu dem in der Kernaufgabe erstellten Spring Boot Backend. Nebst der Authentifizierung sowie Autorisierung eines Users, gilt es eine Produktübersicht für die angebotenen Teesorten und deren Exemplare zu erstellen.

Als Basis für die Implementation nachfolgender Anforderungen wird das Github Repository https://github.com/yagan93/React_SpringBoot zur Verfügung gestellt.

Anforderungen

- A1 Anpassung CORS Konfiguration Spring Boot
  In einem ersten Schritt gilt es in der Klasse core/security/WebSecurityConfig den 'Authorization' Header freizugeben. Dies stellt sicher, dass wir aus dem Frontend Zugriff auf das bereitgestellte JWT haben:

```
configuration.setExposedHeaders(List.of("Authorization"));
```

- A2 Authentifizierung eines Users (4 Punkte)
  Als Vorbereitung gilt es manuell per Postman einen neuen User Max Mustermann zu registrieren. Dessen Attributwerte sind der Klasse AuthenticationContext.tsx zu entnehmen. Das Passwort kann frei gewählt werden, muss im api.post call jedoch entsprechend angepasst werden. Bei korrekter Implementation der Kernaufgabe Anforderung A1 sollte Max die Rolle 'CLIENT' automatisch zugewiesen werden.

  Nun sollte die Typescript Funktion authenticate() in der Lage sein Max Mustermann erfolgreich anzumelden und das retournierte JWT im localstorage unter dem Key 'token' zu hinterlegen. Nebst dem JWT wird jedoch auch der User Datensatz von Max Mustermann benötigt. Um diesen in Erfahrung zu bringen, muss in der Spring Boot Applikation ein neuer Endpoint /users/profile erstellt und von React per api.get abgefragt werden. Der neue Endpoint /users/profile ist selbstverständlich nur für authentifizierte User erreichbar und retourniert lediglich das Profil des momentan eingeloggten Users (Principal).

  Hilfestellung: Sobald das JWT von Max Mustermann im localstorage abgespeichert wurde, wird es über einen Axios interceptor den nachfolgenden API requests im Header mitgegeben (siehe AxiosUtility.ts).

  Nebst der Umsetzung von A2 wird erwartet, dass folgende Frage beantwortet wird: Was spricht dagegen, das JWT eines Users im localstorage abzuspeichern? Was wäre eine bessere Alternative?

- A3 Protected Route (1 Punkt)
  Trotz erfolgreicher Authentifizierung von Max Mustermann scheint React immer auf /login (Not Authenticated) weiterzuleiten. Studiere zur Lösung dieses Problems die Klasse Protected.tsx und implementiere den fehlenden Body der Funktion hasAnyAuthority in der Klasse AuthenticationContext.tsx.

```
//TODO: implement hasAnyAuthority() method. Check if principal has any of the authorities passed as parameter
const hasAnyAuthority = (authorities: Authority["name"][]): boolean => {
  return false;
}
```

- A4 Produkteübersicht (4 Punkte)
  In einem ersten Schritt gilt es in der Klasse Product.tsx eine Funktion zu erstellen, welche über den Endpoint /products für authentifizierte User mit der Autorität 'CAN_RETRIEVE_PRODUCTS' das Teeangebot lädt. Anschliessend gilt es das geladene Teeangebot mithilfe Bootstrap Cards https://getbootstrap.com/docs/4.2/components/card/ dem User zu visualisieren. Als Teeangebot werden die 5x2 Produkte aus der Kernaufgabe Anforderung A2 erwartet.

Bereitstellung

Bis anhin nutzten wir Docker lediglich für die Spring Boot Applikation sowie PostgreSQL Datenbank. Nun sollte jedoch auch für die React Applikation ein entsprechendes Image erstellt und per Container bereitgestellt werden.

- B1 Image erstellen (1 Punkt)
  Man erstelle ein File 'Dockerfile' im Root der React Applikation:

```
FROM node:21-alpine
WORKDIR /app
COPY package.json ./
COPY yarn.lock ./
RUN yarn install --frozen-lockfile
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

  Anschliessend kann das Image der React Applikation über den Befehl *docker build -t react-app .* erstellt und per *docker images* aufgelistet werden.

- B2 Compose erstellen (2 Punkte)
  Anstatt das Image getrennt von Spring Boot und PostgreSQL zu starten, ist das File 'docker-compose.yml' von B2 aus der Kernaufgabe zu ergänzen. So kann das Frontend, Backend wie auch die Datenbank in einem einzigen Befehl *docker-compose up --build --force-recreate* gestartet werden.

  Hilfestellung: Es wird empfohlen die React Applikation zusammen mit der Spring Boot Applikation in einem Monorepo unterzubringen.