**UserService**

Ein Mikroservice für "The Weather App"

Von Lars Linden, Karmen Voncina und Saad Belkhou

Unter der Betreuung von Dr. Thomas Soddemann

# I- EINLEITUNG

Wir waren im Rahmen der Entwicklung einer Wetter-App für den User-Service zuständig.

Unsere Aufgabe war es, eine Datenbank zu erstellen, die Application-Logic zu implementieren und einen Authentication/Authorization Service mittels Keycloak zu integrieren.

# II- SPRINGBOOT

Da das Leitthema des Seminars »Microservices mit Springboot« ist haben wir für unseren User-Service Spring Boot genutzt.

Für die Kopplung mehrerer Services machte dies Sinn, da es eine lose Kopplung der verschiedenen Komponenten durch Dependency Injection und Inversion of Control bietet.

Da Spring Boot die Entwicklung mit dem Spring Framework stark vereinfacht, findet es generell seinen Nutzen in der Entwicklung von Web Applikationen und Microservices.

Spring Boot bietet eine Autokonfiguration und ermöglicht die Erstellung von Standalone Spring-basierten Anwendungen, (Ionos Digital Guide, 2020).

Aufgrunde des Spring Initializers ist die Erstellung einer neuen Applikation trivial und zeiteffizient.

Die Selektion sogenannter 'Starter'-POMS (Project Object Models), vereinfachen die Maven- oder Gradle-Konfiguration maßgeblich.

Für unseren User-Service haben wir uns für Maven als Build-Tool entschieden.

Man kann sowohl die Wahl der Versionen von SpringBoot und Java beliebg einstellen, als auch das Packaging (Jar/War) bestimmen. (Bayer, 2020)

# III- KEYCLOAK

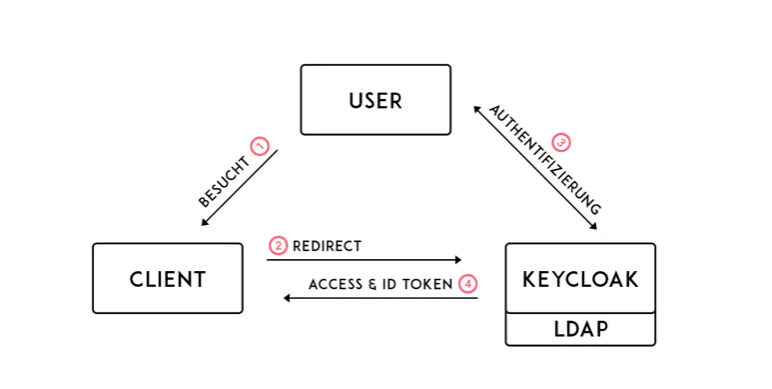
Mit Keycloak kann man eine bestehende Anwendungen oder Services um eine Authentication/Authorization Service erweitern (Klassen, 2017)

Abbildung 1 Authentifizierung mit dem Access & ID Token Quelle: (Klassen, 2017)

Abbildung 1 stellt dar, wie die Kommunikation zwischen dem User, Keycloak und dem Client stattfindet – Nachdem der User durch Keycloak authentifiziert wurde, wird die erfolgreiche Authentifizierung bestätigt und ein Access/ID-Token an den Client zurückgegeben.

Keycloak ist im Wesentlichen ein vorkonfigurierter Wildfly-Application Server, der unabhängig von der abzusichernden Anwendungen ausgeführt wird.

Durch die Isolation dieses Prozesses von der eigentlich Anwendung, müssen die Credentials beziehungsweise Nutzerinformationen nicht preisgegeben werden und der User wird so besser geschützt.

Dies reduziert nicht nur potenzielle Sicherheitslücken bei der Handhabung und Übertragung sensibler Zugangsdaten, sondern bietet auch die Möglichkeit, Ihre eigenen User über eine zentrale Keycloak-Instanz gegenüber externen Anwendungen zu authentifizieren.

User können sich unter anderem mit Keycloak gegenüber einem Active Directory- oder LDAP-Server authentifizieren.

Nach erfolgreicher Authentifizierung des Users stellt Keycloak zusätzlich zum Access Token ein Identity Token in Form eines JSON Web Token (JWT) aus.

Das Access Token ermöglicht es der Anwendung, Dienste im Namen des Users aufzurufen. Das Identity Token hingegen enthält grundsätzlich Informationen über die Identität des Users. Dadurch stellt es den wesentlichen Unterschied zwischen OAuth 2.0 und seiner OpenID Connect-Erweiterung dar, da es im Gegensatz zum klassische OAuth-2.0-Protokoll nicht nur die Berechtigungen in Form eines Zugriffstokens berücksichtigt wird, sondern auch die Identitätsnutzung eines Users.

Anwendungen können ID-Token verwenden, um Informationen wie den Namen oder die E-Mail-Adresse eines Users direkt zu lesen.

Sie können dem Identitätstoken bei Bedarf auch Ihre eigenen Attribute über Keycloak hinzufügen.

Da mehrere Anwendungen (Clients) mit der Keycloak-Instanz verbunden sind, werden User nach dem Einloggen automatisch bei allen verbundenen Clients angemeldet und müssen sich somit nicht mehr für jede Anwendung separat zusätzlich anmelden.

Aus Sicht des Users verbessert dies die Benutzerfreundlichkeit der Anwendung.

# IV- USERSERVICE

Die User-Klasse repräsentiert einen Benutzer der Applikation und enthält Informationen wie die Keycloak-Id des Users, Nachnamen, Vornamen, E-Mail-Adresse, Stadt und Land.

Es enthält auch eine Liste von Locations, die die bevorzugten Standorte des Users darstellen. Die Location-Klasse repräsentiert einen bestimmten Ort und enthält Informationen wie die Id des Ortes, den Namen, die Längen- und Breitengrad. Es hat auch eine Beziehung zur User-Klasse, die darstellt, dass ein Ort zu einem Benutzer gehört.

Die *SecurityConfig-Klasse* wird verwendet, um die Sicherheitseinstellungen der Applikation zu konfigurieren.

Sie konfiguriert den *KeycloakWebSecurityConfigurerAdapter*, welcher eine Klasse aus dem Keycloak-Framework ist und bietet eine Methode *configureGlobal(AuthenticationManagerBuilder auth)*, die verwendet wird, um den *KeycloakAuthenticationProvider* mit dem Authentifizierungs-Manager zu registrieren.

Es bietet ebenfalls eine Methode *sessionAuthenticationStrategy()*, die verwendet wird, um die Sitzungsauthentifizierungsstrategie zu definieren und Methoden zum Erstellen von Beans für *httpSessionEventPublisher* und *sessionRegistry*.

Die UserController-Klasse enthält Methoden, die die HTTP-Requests verarbeiten, wie zum Beispiel die Erstellung eines Users und nutzt die *UserRepository* und KeycloakService-Klassen, um ihre Aufgaben auszuführen.

Die Methode createUser wird verwendet, um einen neuen Benutzer zu erstellen und nutzt die Annotationen @Valid und @RequestBody, um die Eingabe des Benutzers zu validieren. Sie ruft die Methode registerUserInKeycloak der *KeycloakService-Klasse* auf, um den User im Keycloak zu registrieren und speichert anschließend die Benutzerinformationen in der Datenbank mithilfe des *UserRepository ab*.

Das *UserRepository und LocationRepository* sind für die Interaktion mit der Datenbank zuständig, um CRUD-Operationen auf den User- und Location-Klassen durchzuführen.

Sie sind Schnittstellen, die das JpaRepository erweitern, welches die CRUD-Funktionalitäten bereitstellt.

Die *KeycloakService-Klasse* wird verwendet, um Operationen im Zusammenhang mit dem Keycloak-Benutzerverwaltungssystem durchzuführen.

Das Verhalten von unserem UserService ist unten graphisch dargestellt.

Diagram

Description automatically generatedP.S.  Das Service Keycloak läuft in einem Docker Container.

Abbildung 2 Unsere Anwendung (Quelle: Linden, Lars)

# V- FAZIT

Im Rahmen des Projektseminars durften wir in einer Gruppe zusammenarbeiten und haben somit neben den erlernten neuen Technnologien auch unsere Soft Skills und Teamarbeit verbessert.

Wir haben viel Neues über SpringBoot generell, REST-APIs, Swagger.ui und KeyCloak mitgenommen und werden diese Kenntnisse auch in der Zukunft in anderen Projekten einsetzen können.

Spring Boot war aufgrund der Trivialität eine gute Entscheidung und im Rahmen der Wetter-App eine gute Wahl, da wir auch durch die Bereitstellung der REST-APIs eine lose Kopplung erreichen konnten und somit wenig Koordinierung mit den anderen Gruppen nötig war. Außerdem konnten wir am Ende mit Ngrok das Zusammenspiel aller Mikroservices testen.

Keycloak war eine „Identity and Access Management“ -Komponente, die wir vorher nicht kannten und erstmal viel Einarbeitung von uns verlangte.

Da durch die Isolierung der Anwenderdaten vom Client eine zufriedenstellende Sicherheit erreicht wird, war Keycloak eine gute Entscheidung für unser Projekt.

Da Keycloak in einem Docker Container läuft, hatten wir auch die Möglichkeit, uns grundlegend in Docker einzuarbeiten, was uns sicherlich für zukünftige Projekte innerhalb und außerhalb der Uni gelegen kommen wird.

Das Projektseminar war ein guter Einstieg in Mikroservices mit Springboot und hat uns viele nützliche Kenntnisse und Fähigkeiten für die Zukunft vermittelt.

# VI-LITERATURVERZEICHNIS

Bayer, T. (18. November 2020). *Predic8*. Von Predic8-Website: https://www.predic8.de/microservices-spring-boot-spring-cloud.htm abgerufen

Ionos Digital Guide. (16. März 2020). *Ionos*. Von Ionos-Website: https://www.ionos.de/digitalguide/websites/web-entwicklung/spring-boot-tutorial/ abgerufen

Klassen, D. (9. September 2017). *Heise*. Von Heise-Website: https://www.heise.de/ratgeber/Eine-Identitaet-fuer-alles-mit-Keycloak-3834525.html abgerufen