Service Architektur von „DefinitelyNotTwitter“

Technologien:

Die Web-Anwendung ist über seine sehr breite Service Landschaft ausgebreitet. Im Folgenden werden die einzelnen Technologien aufgelistet, und die Beweggründe für die Wahl erläutert. Ziel war dabei auch, eine möglichst heterogene Servicelandschaft, im besonderen Bezug auf Datenbanken, zu entwickeln.

Frontend – Angular:

Im Frontend wurde auf Angular zurückgegriffen. Angular ist heutzutage eines der bekanntesten und weit verbreitetes Clientseitiges Web Framework, welches für die Erstellung von Single Page Applikationen verwendet wird.

Warum?

Das Framework ist sehr umfangreich und bietet sehr viel Funktionalität. Darüber hinaus war schon Erfahrung über Angular und TypeScript im Team vorhanden. Aus diesen Gründen fiel die Wahl für das Frontend schnell auf Angular.

Backend APIs – Spring Boot

Auf der Backend Seite bauen wir die APIs mit Spring Boot auf. Spring bietet für die Spring Boot „startet“ Abhängigkeiten sehr viele Abhängigkeitspakete für die diverse Datenbankanbindungen. Aus diesem Grund kann Spring sehr leicht für die jeweiligen Anforderungen konfiguriert werden.

Warum?

Der initiale Plan war, die APIs hauptsächlich mit Node.js und Express aufzusetzen. Durch die Empfehlung des LVA-Leiters haben wir uns bei allen APIs für Spring Boot entschieden. Im Nachhinein war die Entscheidung goldrichtig. Spring Boot, mit diversen Abhängigkeiten nimmt viel Arbeit bei der Verbindung mit Datenbanken ab. Darüber hinaus hat man ein uniformes System zur Überprüfung der JWT Token über alle Services hinweg.

Graph Datenbank – Neo4j

In der neo4j Datenbank werden die „folgt“ Beziehungen zwischen Nutzern abgebildet. Eine Graph Datenbank eignet sich sehr gut, um diese Beziehungen abzubilden.

Warum?

Der Hinweis, diese Datenbank für diesen Zweck zu verwenden, war eine eindeutige Empfehlung des LVA-Leiters in der ersten Einheit. Dieser Fakt spiegelt sich in dem Verwenden der anderen Teams für den gleichen Zweck.

Datenmodell

In der Neo4J Datenbank speichern wir lediglich die ID des Benutzers. Mehr ist für das Abbilden der Beziehungen nicht notwendig.

Dokument Store Couchbase

Die Daten der Postings speichern wir in einer Dokument Datenbank.

Warum?

Die meisten Teams setzen beim Speichern der Posting Daten auf einen NoSQL Dokument basierten Speicher. Die meisten anderen Teams wählen dafür MongoDB. Unser Beweggrund für Couchbase war eine andere Technologie in die LVA zu bringen und auch selbst kennen zu lernen.

Datenmodell

Ein Posting wird durch folgende Felder zusammengesetzt:

ID: Wird vom uuid plugin in Angular erstellt.

createdAt: Ein Unix Timestamp der Erstellungszeit

mood: Der kodierte Emojie

authorname: Der Vorname und Nachname des Autors des Posts

authorid: Die ID des Autors des Posts

content: Der Text des Posts

Suchdatenbank – Solr

Wir indexieren die wichtigen Felder von Postings und Benutzern in einer Solr Datenbank. Über diese Solr Datenbank wird einen Volltextsuche in der Applikation umgesetzt.

Warum?

Zuerst haben wir uns in die elasticsearch Technologie eingelesen. Jedoch bestanden für uns bei diesem Service schon Probleme eine kostenlose Version zu finden. Aus diesem Grund haben wir auf eine Alternative aus dem Foliensatz aus dem ersten Termin zurückgegriffen. Die Wahl fiel dabei auf Solr. Solr ist unter der Apache Lizenz gratis und sehr mächtig. Selbst die Internetsuche Duck Duck Go indexiert Websites mit Solr. Die Solr Datenbank wird über eine Queue adressiert. Damit stellen wir sicher, dass kein Posting oder Nutzer nicht in die Such Datenbank kommt – falls es zu häufiger/gleichzeitiger Anlegung von Benutzern/Postings kommt.

Datenmodell

Ein Benutzer in der Solr Datenbank wird durch folgende Felder zusammengesetzt:

ID: Die ID des Benutzers

firstname: Der Vorname des Nutzers

lastname: Der Nachname des Nutzers

email: Die E-Mail des Benutzers

pokemonid: Das zufällig zugewiesene Pokemon

Der Benutzer hat in der Solr Datenbank kein Passwort und keine Telefonnummer abgespeichert.

Ein Post in der Solr Datenbank wird nach dem gleichen Modell wie in der Couchbase Datenbank gespeichert.

Relationale Datenbank - Oracle XE 18

Die Nutzerdaten werden in einer relationalen Datenbank gespeichert. Oracle XE ist eine Container Datenbank. Für dieses Projekt wurde in diesem Container eine Pluggable Database Users erstellt. Wir benutzen SQL Developer, um mit der Datenbank zu verbinden und die Daten einzusehen/zu manipulieren.

Warum?

Wir wussten, dass wir, im Sinne der heterogenen Landschaft, die User in einer relationalen Datenbank speichern wollen. Es hatte keinen tieferen Grund, ob wir Oracle oder mySQL oder eine andere relationale Datenbank verwenden.

Datenmodell

Ein Nutzer in der Oracle XE wird durch folgende Felder zusammengestellt:

ID: Wird durch eine Sequence in der relationalen Datenbank erstellt (Diese ID wird danach auch in anderen Datenbanken verwendet)

firstname: Der Vorname des Nutzers

lastname: Der Nachname des Nutzers

email: Die E-Mail des Nutzers

password: Das hashed Passwort (BCrypt Hash)

phonenumber: Die Telefonnummer des Nutzers

pokemonid: Das zufällig zugewiesene Pokemon

Message Broker – Active MQ

Um bei diversen Kommunikationen einen Puffer dazwischen zu schalten, verwenden wir einen Message Broker. Dies soll sicherstellen, dass eingefügte Daten auch wirklich in jeder involvierten Datenbank ankommen. Als Beispiel können wir hier den Indexing Service nennen, welcher Daten in die Solr Datenbank speichert. Sollten mehrere Postings und User gleichzeitig gespeichert werden, soll der Indexing Service nicht überlasten werden, und die Daten im Broker gepuffert werden können.

Warum?

Während die meisten Teams hier auf die RabbitMQ setzen, haben wir uns hier dazu entschlossen, den ActiveMQ Broker aus der Apache Lizenz einzusetzen.

Key-Value Store – Redis

Benachrichtigungen, wenn ein Nutzer einen neuen Abonnenten hat, werden in einem Redis Key-Value Store angelegt. Dabei ist die User-ID der Key und eine Liste an Notification Objekten der Value für jeden Eintrag. Die Wahl fiel auf einen Key-Value Store aufgrund der heterogenen Datenbanklandschaft.

Warum?

Wir haben uns für Redis aufgrund der Empfehlung auf den Folien der ersten LVA entschieden.

Datenmodell

Der Key Value Store ist nach folgendem Muster aufgebaut:

Key: Die Nutzer ID

Value eine Liste von Notification-Objekten

Ein Notification Objekt setzt sich aus diesen Feldern zusammen:

ID: Die ID der Notification (im Frontend uuid generiert)

createdAt: Der Unix Timestamp zum Zeitpunkt der Erstellung

text: Der Text der Benachrichtigung

read: Boolean der aussagt, ob die Benachrichtigung gelesen wurde

Authentifizierung

Für die Authentifizierung arbeitet unsere Anwendung mit JWT Tokens. Bei erfolgreichem Login sendet das Backend einen JWT Token mit einer Gültigkeitsdauer von einer Stunde zurück. Dieser Token wird auf den Local Storage des Browsers gelegt. Damit kann man auf diesen zugreifen, auch wenn man den Browser geschlossen hatte. Wenn der Token noch Gültigkeit hat, wird man automatisch in der Anwendung angemeldet. Der Token wird bei API-Calls auf anderen Services auf Gültigkeit überprüft.

Bei Rest Calls aus dem Angular Frontend wird über einen Interceptor automatisch der Token im Header gesetzt. Ein anderer Interceptor überprüft die Antworten der Backend Services. Wenn der Status 401 „Unauthorized" zurückkommt, wird der Nutzer automatisch abgemeldet. Bei einer „No Response" Antwort wird eine Benachrichtigung gegeben, dass die Server nicht erreichbar sind.

Technische Details von Benutzerinteraktionen

Registrierung/Benutzer löschen:

Bei einer Registrierung wird der neue Benutzer dem User Service Backend übergeben. Das Formular im Frontend ist stark validiert, um fehlerhaften Input dort schon zu vermeiden. Der User Service fügt den Benutzer in die relationale Datenbank ein. Dort wird die ID erstellt. Der neue User mit der erstellten ID wird zurückgegeben. Danach wird per Rest der neue Benutzer and die Neo4j und an den Redis Key-Value Store gesendet. Dort wird der User mit der generierten ID angelegt. Dadurch kann der Benutzer anderen folgen und gefolgt werden. Außerdem wir das Notification Wrapper Objekt angelegt, um für diesen Nutzer im weiteren

Verlauf Notifications anlegen zu können. Im letzten Schritt wird der neue User über die Queue an die Solr Such Datenbank übergeben. Falls in irgendeinem Schritt ein Fehler auftritt, wird der Benutzer aus bereits bearbeiteten Datenbanken wieder entfernt und eine Fehler Meldung im Frontend angezeigt.

Den Benutzer Löschen funktioniert auf den gleichen Kommunikationswegen. Das Error-Handling findet auch gleich statt, mit dem Unterschied, dass der Benutzer in bereits gelöschten Datenbanken wieder eingefügt wird.

Posting abgeben/bearbeiten/löschen/lesen:

Postings werden über einen Rest Service in die Queue gegeben. Über die Queue gelangt der neue Post zur Suchdatenbank Solr und zur Postingdatenbank Couchbase. Das Posting Löschen erfolgt ebenfalls über die Queue. Die Queue ermöglicht das Posten, auch wenn die Backend Services offline sind.

Der lesende Zugriff auf die Postings geht direkt mit Rest auf den Posting Service. Hier wird keine Queue benutzt.

Eigenen Benutzer bearbeiten:

Der eigene Benutzer kann durch einfache Rest Aufrufe auf den User Service bearbeitet werden. Beim Ändern des Passworts wird das Passwort noch einmal überprüft. Außerdem wird überprüft, ob der übergebene Token zum geänderten User passt.

Anderen Benutzern folgen/entfolgen:

Mit Rest aufrufen auf den Relation Service wird eine Verbindung zwischen Usern erstellt oder gelöscht.

Volltextsuche abgeben:

Mit Rest Aufrufen wird im Such Service eine Abfrage auf Solr mit dem Suchbegriff durchgeführt. Dabei werden mit separaten Repositories die User und die Postings durchsucht und die Ergebnisse zurückgegeben.

Notifications erstellen/lesen:

Im aktuellen Stand werden Notifications nur beim Folgen eines anderen Users ausgelöst. Dieser Rest Aufruf geschieht direkt aus dem Frontend, nicht aus dem Backend, wie beim User anlegen. Nachdem die Notification angelegt wurde, wird sie dem Empfänger in der Nachrichten Inbox angezeigt. Der Aufruf wird aus dem Frontend gemacht, um sicherzustellen, dass der Follow auf

jeden Fall ausgeführt wird. Es soll der Relation Service keinen Fehler weil die Rest Kommunikation mit dem Notification Service nicht funktioniert hat.