Software Architectures for Enterprises – Ergebnisbericht Übungsblatt 2

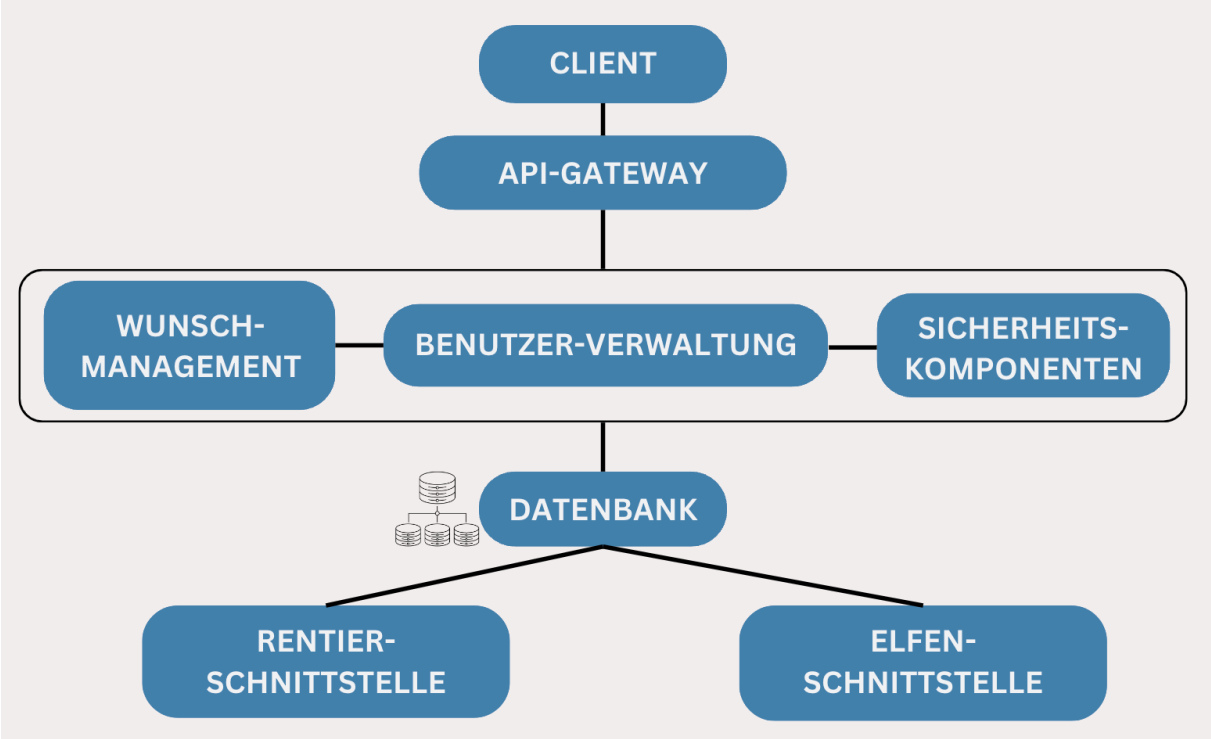
Hakan Bayindir, Matrikelnummer: 1496384

Der zugrunde liegen Code liegt vollständig in einem Git-Repository:

https://github.com/Nooctis/xmaswishes_bayindir

Aufgabe 1

Das Diagramm zur Softwarearchitektur mit den Hauptkomponenten sieht folgendermaßen aus:



Ich möchte im Folgenden auf die einzelnen Komponenten eingehen:

1. Client

   Der Client dient dazu, die Wünsche der Kinder über ein Endgerät, wie zum Beispiel einem Smartphone, aufzunehmen. Die Benutzeroberfläche ermöglicht die Eingabe von Name und Wünschen.

2. API-Gateway

   Das Gateway ist der zentrale Einstiegspunkt für alle Client-Anfragen. Die Anfragen können authentifiziert werden und eventuelle Ratenbegrenzungen können vorgenommen werden. Das erhöht die Sicherheit und liefert eine bessere Kontrolle über den Datenverkehr.

3. Backend-Services

   Hier wird die Geschäftslogik implementiert. Das Wunschmanagement verarbeitet die eingereichten Wünsche und aktualisiert & fragt den Wunschstatus ab. Die Benutzerverwaltung managet die Nutzerdaten. Die Sicherheitskomponenten schützen das Backend vor ungewünschten Zugriffen.

4. Datenbank

   Die Datenbank enthaltet alle relevanten Namen, Wünsche und Statusinformationen. Wir nutzen eine verteilte Datenbank zur Reduzierung der Latenz und zur Verbesserung der Verfügbarkeit. Es wird eine NoSQL-Datenbank (z.B. MongoDB) verwendet. Das stellt ein flexibles Datenmodell dar und ermöglicht horizontale Skalierbarkeit. Die Datenbank wird mit Datenschutzkomponenten ausgestattet, wodurch auch die Einhaltung der DSGVO eingehalten wird. Es wird beispielsweise eine Authentifizierung seitens der Elfen und Rentiere benötigt, um den Zugriff der Daten auf berechtigte Personen zu beschränken. Die Daten werden verschlüsselt und durch den Einsatz von SSL übertragen.

5. Elfen- und Rentierschnittstelle

   Eine API oder andere Kommunikationsschnittstelle, über die die Elfen und Rentiere auf die Daten zugreifen können. Elfen können somit auf die Wünsche zugreifen, um die Geschenke herzustellen und Rentiere können mit den Statusinformationen die Logistikplanung vorantreiben.

Aufgabe 2

Wenn die Architektur aus der ersten Teilaufgabe konkret umgesetzt werden müsste, würde ich die einzelnen Komponenten folgendermaßen implementieren:

1. Client
   Für das Front-End würde ich das Framework Vue.JS nutzen. Mit Hilfe von Ionic und Capacitor kann die Vue-Applikation dann einfach als IOS und Android App umgesetzt werden.

2. API-Gateway
   Als Gateway wird Kong genutzt. Kong ist ein leistungsfähiges Open-Source-API-Gateway, das Funktionen wie Authentifizierung, Ratenbegrenzung und Logging unterstützt. Es lässt sich leicht skalieren, was für unseren Anwendungsfall wichtig ist.

3. Backend-Services

   Für die Backend Programmierung nutzen wir Node.js. Es bietet eine hohe Performance und eignet sich gut für I/O-intensive Anwendungen.

4. Datenbank

   Als Datenbank nutzen wir eine NoSQL-Datenbank wie MongoDB, da diese eine hohe Skalierbarkeit und Flexibilität zur Speicherung von unstrukturierten Daten wie Wünschen und Namen bietet. Da wir eine verteilte Datenbank nutzen möchten, greifen wir speziell auf MongoDB Atlas zurück. Atlas bietet geographisch verteilte Replikate zur Reduzierung der Latenz und Verbesserung der Verfügbarkeit.

5. Sicherheits- und Datenschutzkomponenten

   Zur Authentifizierung nutzen wir OAuth und OpenID Connect. Das sind die Standards für sichere Authentifizierung und Autorisierung. Zur Datenverschlüsselung wird AES-256 genutzt.

6. Elfen- und Rentierschnittstelle
   Für diese Schnittstelle müssen wir einfach die Daten abfragen können. Dazu nutzen wir GraphQL. Es bietet eine flexible und effiziente Datenabfrage. Als Kommunikationsprotokoll für die API nutzen wir gRPC. gRPC ist geeignet für hochperformante Kommunikation.

Aufgabe 3

Das Hauptziel dieser Aufgabe was es, zu überprüfen, wie sich der Mehreinsatz von Hardware auf das zusätzliche Nachrichtenvolumen der XmasWishes-API auswirkt. Zudem sollte die maximal Anzahl an API-Calls pro Sekunde ermittelt werden, um die Leistungsfähigkeit und Skalierbarkeit des Servers zu demonstrieren.

Folgende Technologien wurden verwendet:

- Node.js: für die Server-Implementierung
- Mongoose: Object Data Modeling für MongoDB
- PM2: Prozessmanager zur Verwaltung und Skalierung der Node.js-Anwendung
- Axios & Bluebird: Durchführung von Load-Tests
- Winston & Morgan: Logging-Tools zur Überwachung und Fehleranalyse
- Shell-Skripte: zum gleichzeitigen Ausführen mehrerer Load-Test-Clients

Im ersten Schritt wurde der Server erstellt („server.js“). Die Verbindung zu MongoDB gelang mittels Mongoose. In der Server-Datei wurden die API-Endpunkte (/api/wishes für GET und POST) implementiert. Um mehrere Server gleichzeitig zu starten, wurde der Server mit Hilfe von PM2 im Cluster-Modus gestartet. Dadurch werden so viele Instanzen des Servers gestartet, wie CPU-Kerne verfügbar. Der Command lautet:
pm2 start server.js -i max --name xmaswishes.

Ein Skript wurde entwickelt, das sowohl GET- als auch POST-Anfragen an die API sendet, um die Serverleistung unter hoher Last zu testen („client.js“). Als Output werden die Anzahl der erfolgreichen und fehlgeschlagenen Anfragen aufgelistet, sowie die gesamten API Calls pro Sekunde.

Um den Einfluss der Anzahl verschiedener Serverinstanzen zu messen, wurden Load-Tests durchgeführt. Dabei hat ein Client 500 requests an die API gestellt und mit einem Shell-Skript 5 mal gleichzeitig aufgerufen. Das Ergebnis sieht folgendermaßen aus:

Anzahl Serverinstanzen	Erfolgreiche Anfragen	Fehlgeschlag. Anfragen	Max. Dauer [Sekunden]	Anfragen/Sekunde
1	2500	0	174,02	2,87
2	2500	0	100,86	5,19
3	2500	0	80,72	6,33

Aus den Ergebnissen kann abgeleitet werden, dass die Dauer, bis alle Requests eines Clients bearbeitet wurden, mit steigender Serverinstanz gesunken ist.

Aufgabe 4

In diesem Aufgabenteil wurde ein System entwickelt, das eingescannte Wünsche aus einer Datei in das bestehende XmasWishes-System überträgt. Dabei sollte die Integration möglichst effizient und robust gestaltet werden, um die Verarbeitung von Wünschen zu automatisieren.

Der Ansatz war dabei, Apache Camel als Integrationsplattform zu nutzen. Es wurde eine Dateiüberwachung implementiert, die erkennt, sobald Wünsche (.txt-Dateien) in ein bestimmtes Verzeichnis (scanned-wishes) eingehen und die enthaltenen Wünsche an die API des XmasWishes-Systems zu übermitteln. Jede Datei enthält dabei zwei Zeilen, Name und Wunsch.

Das bestehende XmasWishes-System (Node.js) war bereits mit einer REST-API ausgestattet, die das Speichern von Wünschen ermöglichte. Die Route in Apache Camel wurde so gestaltet, dass sie mit dieser API kompatibel ist.

Während also die generierte JAR-Datei läuft, wird das Verzeichnis „scanned-wishes“ überwacht, und sobald eine txt-Datei mit Name und Wunsch hinzugefügt wird, wird der Inhalt an die API gesendet.