Systemarchitektur

Diskussion

Im Folgenden werden die Vor- und Nachteile eines zentralen und eines dezentralen Architektur Stils anhand von Beispielen diskutiert.

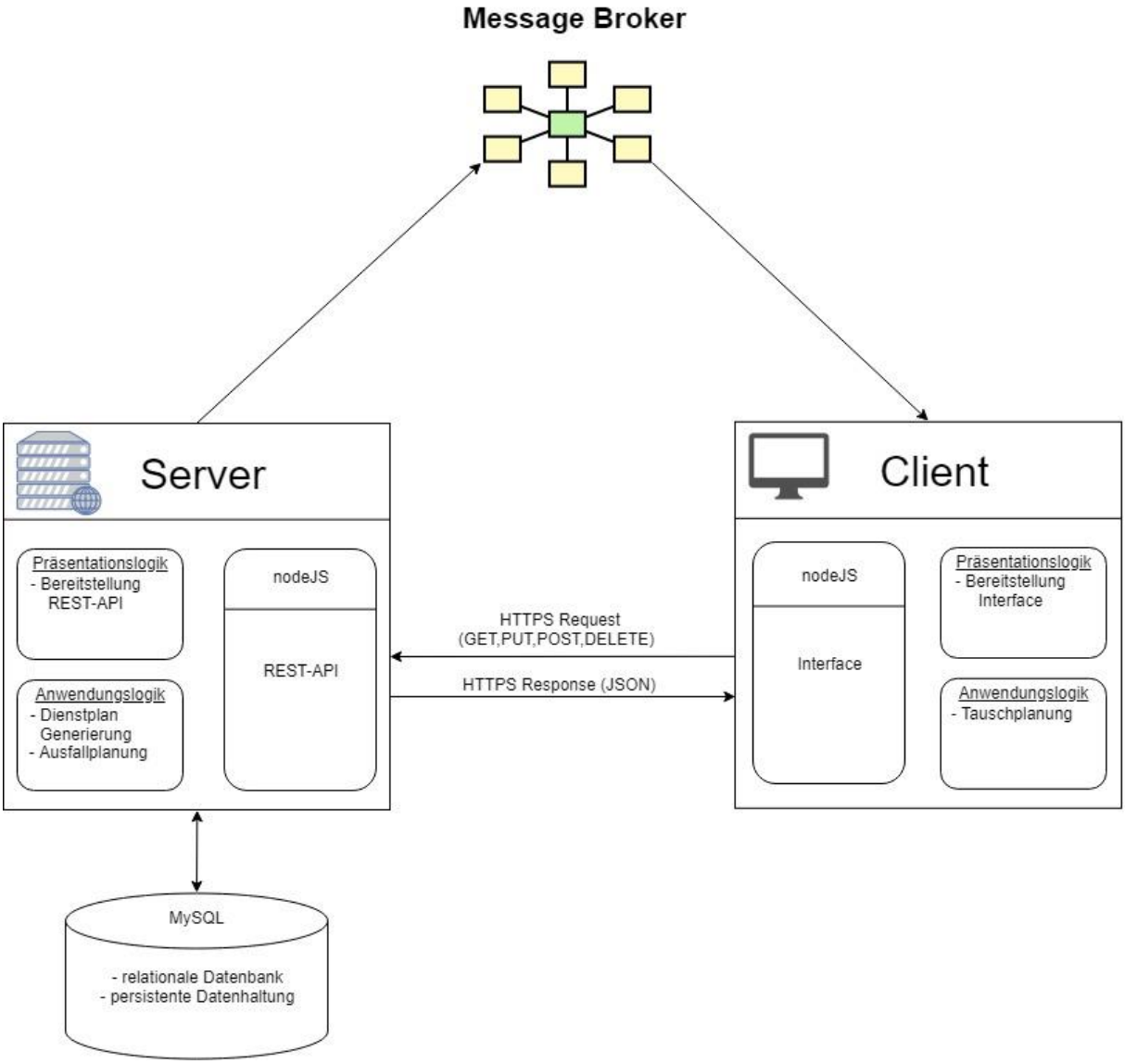
Das Client – Server System basiert auf einem zentralen Stil. Bei dieser Orchestrierung übernimmt der Server die Rolle des Dirigenten. Der Client hingegen steht dem Server als Musiker entgegen. Im genaueren heißt das, dass der Server eine oder mehrere Schnittstellen besitzt über diese Daten vom Client/-s empfangen oder gesendet werden. Der Client kann Informationen anfragen und diese darauf weiterverarbeiten. Hierbei übernimmt der Server die Datenhaltung, wodurch die Redundanz von Ressourcen vermieden werden kann. Der Ausfall eines Clients im System ist leicht zu verkraften, fällt hingegen die Serverkomponente aus, sind alle Clients betroffen. Diese könnten nur noch beschränkt oder gar nicht ausgeführt werden. Deshalb ist es empfehlenswert einen redundant arbeitenden Server zu installieren, um die Ausfallmöglichkeit des Systems zu dezimieren.

Die Peer to Peer Architektur ist anders als die Client – Server Architektur ein dezentrales System. Hierbei können die Systemkomponenten Dienste und Ressourcen bereitstellen, aber genauso auch auf diese Zugreifen. Dies ist besonders Organisationsbedürftig, um Redundanz zu vermeiden und ein möglichst effizientes System zu schaffen, müssen die Dienste und Ressourcen genau strukturiert und an die einzelnen Systemkomponenten sinnvoll verteilt werden. Es ist nicht aussenvor zu lassen, dass die Ausfallsicherheit bei diesem Stil sehr hoch ist.

Fazit

Nach Abwägung der Architekturen, kann beim Adaptieren mit dem vorliegenden Projekt, der Schluss gefasst werden das eine Peer to Peer Architektur ungeeignet ist. In unserer Domäne erfüllt die Stationsleitung die zentrale Rolle der Dienstplan Erstellung. Ein gemeinsames Erstellen des Dienstplans, würde sehr wahrscheinlich zu einem Chaos führen. Das Client-Server Modell spiegelt diese Rollenverteilung wieder, zudem verfügt dieser Stil über eine klare Verteilung der Anwendungslogik.

Architekturdiagramm



Systemkomponenten

Server

Der Server stellt die zentrale Schnittstelle der Anwendung dar. Er stellt eine REST-API bereit mit der die Clients kommunizieren. Die REST-API wird mithilfe von nodeJS umgesetzt, Ressourcen sind hiermit leicht zu implementieren. Mithilfe von Client Informationen sollen hier automatisiert Dienstpläne erstellt werden und auf Ausfälle entsprechend reagiert werden.

Client

Der Client stellt ein Interface dar, dass mit dem Server kommuniziert. Implementiert wird der Client in nodeJS oder Java, je nachdem ob es sich um eine Webportal oder eine mobile Applikation handelt. Der Client soll mit Hilfe von Server Informationen geeignete Schicht-Tauschpartner identifizieren und diese an den Server übermitteln.

Message Broker

Der Message Broker übernimmt die Aufgabe des Benachrichtigens der Clients. Ein Eventlistener auf dem Server wartet auf einen Zustand, trifft dieser ein setzt er eine Nachricht inklusive Schlüssel an den Message Broker ab. Dieser stellt mit Hilfe des Schlüssels fest welche Clients eine Benachrichtigung erhalten sollen und übermittelt die Nachricht.

Datenhaltung

Die Daten werden auf dem Server in einer relationalen Datenbank, die auf MySQL basiert persistent gespeichert.

Kommunikation // noch zu diskutieren!

Die Kommunikation verläuft synchron über das verschlüsselte HTTPS Protokoll. Server und Client nutzen zur Datenübertragung das JSON Format. Die Kommunikation zwischen Client und Message Broker verläuft asynchron.