WBA Modellierung

Wesentliches Merkmal eines verteilten Systems ist die Tatsache, dass Komponenten der Anwendung auf mehrere Systeme verteilt sind. Da sich für eine typische Client-Server Architektur entschieden wurde, sind dementsprechend die Anwendungskomponenten auf den Client und den Server verteilt. Dabei nehmen beide die jeweils gängige und passende Rolle ein.

Der Server (auch **Dienstgeber**) dient durch eine Anbindung an eine zentrale Datenbank unter anderem als Anlaufstelle zum Lesen und Schreiben von Datensätzen. In einem offenen, nutzerbasierten System dient der Server auch zur *Authentifizierung* und *Authorisierung* von Nutzern. Natürlich sollte in einer wohlgeformten Anwendung der Schritt der serverseitigen Authorisierung nur als Sicherheitsmaßnahme gegen unbefugte Zugriffe dienen; letztendlich sollte der Nutzer auf normalem Wege solche unbefugten Zugriffe gar nicht tätigen können.

Der Dienstgeber soll in Node.js programmiert werden, welches ermöglicht, mit den gesendeten und empfangenen JSON-Daten direkt zu arbeiten, anstatt sie serialisieren und deserialisieren zu müssen. Ein Datenbanksystem, welches ebenfalls direkt mit JSON-Daten umgehen kann, schließt sich daran logisch an.

Als Datenbanksystem wird *mongoDB* als NoSQL-DBS genutzt. Als Rahmenwerk um mongoDB ist *mongoose* sehr etabliert, da es eine einfache Schnittstelle zur mongoDB anbietet. mongoDB hat mit mongoose im Vergleich zu anderen Systemen wie Redis umfangreichere Möglichkeiten zur Strukturierung, Schematisierung und Abfrage von bestimmten Datensätzen. Besonders wird die *populate*-Funktion von mongoose benötigt, da mit ihr Documents anhand eines Indexes oder sogar einer Liste von Indizes in ein anderes Document übergeführt werden können, ohne dabei die schematische Struktur zu verletzen. Dies ist vergleichbar mit *Joins* aus herkömmlichen SQL-Datenbanksystemen.

Die asynchrone Kommunikation wird vom Server angeleitet. Eine Interaktion des Clients mit dem Server ist in dieser Anwendung stets synchron, sprich auf eine Anfrage des Clients an den Server folgt immer eine direkte Antwort des Servers. Zur asynchronen Kommunikation wird Google Cloud Messaging für den Versand von Push-Notifications verwendet. Ebenso wird zur Verifizierung von Studenten ein Mailserver als asynchrones Kommunikationsmittel genutzt.

Der Client (auch **Dienstnutzer**) dient vorwiegend als direkte Interaktionsoberfläche für den Nutzer. Durch die Interaktionen des Nutzers wird beispielsweise – möglicherweise schrittweise – ein *Request* an den Server geformt, bis jener Request letztendlich zum Server geschickt wird. Antworten (*Response*) vom Server müssen ebenso verarbeitet werden, um diese empfangenen Daten in eine visuell ansprechende Struktur für den Nutzer zu bringen.

Der Client soll in Java unter der Plattform Android programmiert werden. Eine Android-App stellt in der Nutzungsdomäne einen geeigneten Anwendungstypen dar.

Eine unabdingbare Funktionalität im Zusammenspiel von Client und Server ist die schon genannte Verifizierung von Studenten. Der Client steuert hierbei die Anfrage zur Verifizierung und führt auch im Nachhinein anhand eines Codes die eigentliche Verifizierung durch.

Verifizierung

Bei der Verifizierung fordert der Client zunächst eine Liste aller Hochschul-TLDs vom Server an. Anhand dieser Liste kann der Client entscheiden, ob die vom Nutzer angegebene E-Mail Adresse überhaupt zulässig ist. Bei erfolgreicher Überprüfung wird eine neue Verifikation beantragt, durch die über den Server eine E-Mail mit einem Verifikationslink an die angegebene Mailadresse gesendet. Dieser Verifikationslink lässt sich wiederrum nur mit dem Client öffnen, welcher dann den Link anhand eines Codes im Verifikationslink überprüft und bei finaler erfolgreicher Überprüfung die Benutzer mit einem verifizierten Eintrag aktualisiert.

Beitragsverarbeitung

Die Datenstruktur der Beiträge wird als Baumstruktur wiedergegeben. Diese Baumstruktur wird für die Antwort vom Server mit der oben genannten *populate*-Funktion bis zu einer bestimmten Tiefe erstellt. Genutzt wird dies sowohl bei der Ausgabe eines Studiengangs als auch bei der Ausgabe einer einzelnen Thread-Struktur. Weiterhin werden HATEOAS-typische Hyperlinks in die Antwort eingebettet, um beispielsweise tiefer geschachtelte Beiträge aufrufen zu können, um eine Pagination von Beiträgen zu ermöglichen und um die Anwendungslogik in anderen Bereichen steuern zu können.
Die vom Client empfangenen Daten werden sortiert und in bestimmte Rubriken gefiltert.
Bei der Übertragung von Beiträgen vom Client zum Server (Erstellen von neuen Beiträgen) wird die ElternID des neuen Beitrags mitübertragen, damit der Server diesen neuen Beitrag korrekt einordnen kann. Vom Server muss dann der neue Beitrag im Elternbeitrag referenziert werden und der Elternbeitrag im neuen Beitrag referenziert werden. Nur so ist eine erfolgreiche *population* möglich.
Weiterhin ist es möglich einen Beitrag zu bearbeiten und zu löschen. Beim Löschen werden zwar die Beitragsdaten komplett gelöscht, in der Baumstruktur wird allerdings angezeigt, dass ein Beitrag gelöscht wurde, um die Baumstruktur nicht zu verletzen.

Studiengangssuche

Die Suche nach einem Studiengang kann als Textsuche oder als Umfeldsuche anhand der Geo-Position erfolgen. Die Priorität der Geo-Suche steht allerdings an zweiter Stelle. Das Ergebnis der Suche ist eine Liste von passenden Studiengängen, die aus Datensicht jeweils das Wurzelelement der Beitrags-Baumstruktur darstellen.

Für die Suche wird die “GET /studiengang” Ressource mit Query-Parametern zum Filtern der Ergebnisse genutzt.

Die skizzierte Funktionweise der **Nutzerverwaltung** in einer vereinfachten OAuth-Methodik ist in der Architekturbegründung zu finden.