Dokumentation des Service

HTTP Verben und Semantik

Zu beginn unseres Projektes haben wir uns Gedanken über die für das Projekt benötigten Ressourcen und deren HTTP Verben gemacht. Zu diesem Zweck wurde eine Ressourcentabelle erstellt, welche die benötigten Ressourcen und zugehörigen Verben enthält.
Zu beginn enthielt die Tabelle nur die Ressourcen „/user“, „/fach“ und „/question“, jedoch wurde im laufe des Projektes klar, noch mehr Ressourcen nötig waren, um eine adequate Umsetzung des Service zu gewährleisten. Zu diesem zweck wurden die benötigten Ressourcen erweitert, so dass nun die Ressourcen „user“, „/alluser“, „/question“, „/quiz“, „/authenticate“ und „/statistic“ genutzt wurden. Die ursprünglich geplant Ressource „fach“ wurde hierbei gestrichen und durch einen Key ersetzt der in der Ressource „/question“ mit gespeichert wird. Dies ermöglicht nun ein Suchen nach Fragen eines jeweiligen Faches, indem das Fach als Key verwendet wird und aus der Datenbank die gespeicherten JSON strings abgeglichen werden.
Auf die Ressource „/user“ und „/question“ könne die HTTP Verben GET, PUT, POST und DELETE angewendet werden um zusammen mit der ID neue Daten einzuspeichern (POST), zu ändern (PUT), Daten zu löschen (DELETE) oder ganze Datensätze zu löschen (DELETE).

Auf die Ressourcen „/alluser“ und „/quiz/:fach“ kann nur ein GET vorgenommen werden. Ein GET auf „/alluser“ schickt hierbei eine Liste aller User zurück während ein GET auf „/quiz:fach“ Fragen aus der Datenbank zieht welche zu dem angefragten Fach passen und schickt diese weiter. Hierzu später mehr.
„/autenticate“ dient, wie der Name schon sagt, zur Authentifizierung eines Nutzers, weshalb auf diese Ressource nur ein POST ausgeführt werden kann, welcher dem Dienstnutzer zurückliefet, ob der Nutzer gültig ist oder nicht.
Auf die letzte Ressource „/statistic“ können sowohl „POST“ als auch „GET“ ausgeführt werden um bei einer neu beantworteten Frage die Statistiken zu aktualisieren oder die Statistiken abzufragen.

Datenhaltung und Anwendungslogik

Für die Persistente Datenhaltung werden die Daten als JSON strings in der Datenbank, in unserem Fall Redis, gespeichert.
Für die Fragen werden diese mit dem Key „/question:id“ in der Datenbank gespeichert. Jede Frage bekommt hierbei eine eindeutige ID, welche bei jedem speichern um 1 inkrementiert wird. Die Daten der Frage werden hierbei als Value des Keys gespeichert und beinhalten 4 Antwortmöglichkeiten, welche der Antworten die richtige ist und die ID der Frage, genauso wie die Frage selber.
Bei einem GET auf die Ressource „/question“ kann nun über die ID der Frage einfach der Passende Inhalt gefunden und an den Sender der Anfrage gesendet werden.
Genauso kann auch eine GET anfrage auf die Ressource „/user“ zusammen mit einem Fach gestellt werden. Hierbei werden alle Fragen in der Datenbank nach dem passenden Fach durchsucht und es werden nur die IDs der passenden Fragen an den Dienstnutzer weitergeleitet. Dieser kann dann mit einem GET und der ID den kompletten Inhalt der Frage herausfinden.
Zudem kann eine GET Anfrage auf „/question“ gestellt werden ohne ID oder Fach zu spezifizieren. In diesem Fall werden alle Fragen mit ihrem kompletten Datensatz gespeichert und an den Dienstnutzer weitergeleitet.
Mit PUT und DELETE können somit auch einzelne Elemente der Fragen nach ihrer ID geändert, bzw die komplette Frage gelöscht werden.

Anders funktioniert ein POST auf „/user“. Hier wird ein Komplettes Nutzerverzeichnis geladen, in dem Alle Nutzer mit Namen und Passwort gespeichert werden. In diesem Nutzerverzeichnis geht der Server dann alle IDs durch und sucht nach der letzten ID. Daraufhin fügt er den neuen User nun mit der Neuen ID hinzu und speichert das komplette Nutzerverzeichnis neu.
Bei einem GET auf den „user“ wird der Nutzername vorausgesetzt. Hier lädt der Server wieder das Nutzerverzeichnis aus der Datenbank und geht dieses wieder durch. Wenn er den Passenden Nutzer gefunden hat filtert er diesen heraus und schickt ihn an den Clienten zurück. Falls kein User mit diesem Namen vorhanden sein sollte wird ein Fehlercode gesendet.
PUT und DELETE arbeiten bei „/user“ nach der ID. Hier wird der User nach der ID, wie bei einem GET, gesucht und es können Daten geändert oder im Falle von DELETE einzelne User aus der Liste aller User gelöscht werden.
Zudem besteht die Möglichkeit, sich über eine GET Anfrage auf die Ressource „/alluser“ ohne ID oder Namen eine Liste aller User ausgeben zu lassen.

Bei einem POST auf die Ressource „/authenticate“ schickt der Dienstnutzer Name und Passwort des Nutzers an den Dienstgeber. Dieser holt sich wieder die Liste aller Nutzer aus der Datenbank und sucht nach dem passenden Namen. Wenn er diesen gefunden hat, prüft er noch ob das Passwort stimmt. Wenn beide Bedingungen erfüllt sind, schickt er einen Token an den Dienstnutzer zurück, welcher den User authentifiziert.
Falls der User nicht gefunden wird wird ein Status 404 an den Dienstnutzer zurück geschickt mit der Meldung, dass der Nutzer nicht gefunden werden konnte. Falls der Nutzer

gefunden werden konnte, aber das Passwort nicht stimmt, schickt er einen Status 403 mit entsprechender Meldung an den Dienstnutzer.

Bei einer GET Anfrage auf die Ressource „/quiz/:fach“ zusammen mit einem Fach durchsucht der Server alle Fragen mit dem passenden Fach und leitet diese, nur mit ihren IDs an den Dienstnutzer weiter. Im Dienstnutzer werden dann die Fragen gefiltert und dieser Fragt dann mit den IDs die Inhalte der Fragen in zufälliger Reihenfolge an um sie für das Quiz aufzubereiten.

Die letzte Ressource des Dienstgebers sind „/statistics“. Diese Speichern Statistiken für jeden Nutzer je nachdem, wie viele Richtig und falsch beantwortete Fragen pro Fach existieren.
Wenn ein Nutzer eine Frage richtig oder Falsch beantwortet wird der Nutzername, zusammen mit dem Fach and den Dienstgeber gesendet. Dieser prüft nun, ob für den jeweiligen Nutzer bereits Statistiken für das entsprechende Fach existieren. Wenn nicht, legt er diese an. Darauf hin inkrementiert er für jede Richtige oder Falsche Antwort den Wert des Entsprechenden Keys in der Datenbank um eins.
Bei einem GET auf „/statistic“ verbunden mit den Nutzernamen ließt der Dienstgeber dann alle Statistiken dieses Nutzers aus der Datenbank aus und speichert diese in einem Array. Dieses schickt er dann an den Dienstnutzer zurück, welcher diese Daten in im HTML speichert wo sie mit javascript zur grafischen Darstellung aufbereitet werden.

Aus Zeitmangel nicht umgesetzte Funktionalitäten

Auf der Seite des Dienstgebers konnten wir nahezu alles umsetzen, was wir uns vorgenommen hatten.
Die Einzigen Funktion, welche wir aus Zeitgründen leider nicht mehr implementieren konnten, war ein Ranking, welches den Rang der Nutzer untereinander anzeigen sollte. Hier sollte den Nutzer die Möglichkeit gegeben werden, sich selber und seinen Wissensstand untern den Nutzern einzuordnen. Dies sollte sowohl für alle Fragen, als auch per Fach möglich sein um dem Nutzer zu zeigen, ich welchem Fach er noch Schwächen hätte.