Vorbereitung

Rufen Sie sich die verschiedenen HTTP-Methoden und Arten der Parameter-übergabe ins Gedächtnis, oder noch besser, suchen Sie sich eine schöne Übersichtsseite und schreiben sich die raus, die Sie brauchen und wozu Sie sie brauchen am besten gleich mit!

Informieren Sie sich außerdem über die gebräuchlichsten HTTP-Status-Codes und deren Bedeutung. Schlagen Sie außerdem die Prinzipien von REST nach und machen Sie sich bewusst, wie diese in Restopoly angewendet werden müssen. Vor allem HATEOAS benötigt ein paar Gedanken mehr, um das Konzept zu durchdringen und richtig einsetzen zu können.

Einführung – Client-Service- & Service-Service-Communication

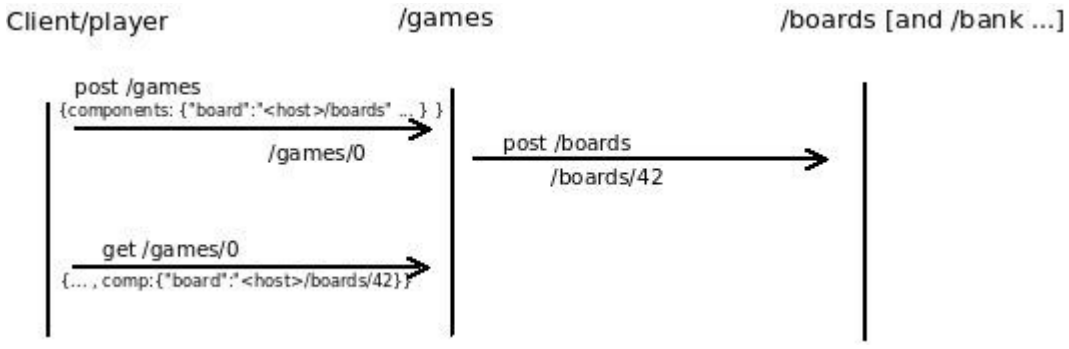
Im ersten Aufgabenblatt wurden nur einfache Anfragen an die Services gestellt, welche leicht über den Browser überprüfbar waren und wenig Interaktion benötigten.

In diesem Aufgabenblatt sollen Sie die zurückgegebenen Werte weiter verwenden und mittels eines Clients verarbeiten und damit neue Anfragen an andere Services stellen, sowie Inter-Service-Kommunikation benutzen. Die API ist als RAML spezifiziert (verfügbar über die OwnCloud).

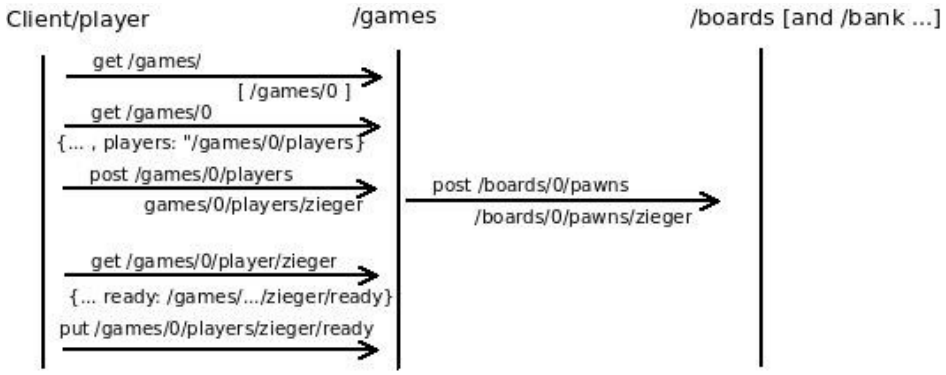
Hierbei werden alle REST-Prinzipien angewendet. Besonders viel Disziplin erfordert HATEOAS, so dass nur solche URIs benutzt werden, die direkt aus vorherigen Abfragen ableitbar sind.

Des Weiteren werden Multi-Tier-Service-Aufrufe eingeführt, so dass die angefragten Services innerhalb eines Requests selbst weitere Services anfragen, um richtige Ergebnisse liefern zu können. Hierbei tritt der zuerst aufgerufene Service selbst in die Rolle eines Client. Hierfür sind folgende Abläufe beispielhaft:

Spiel Initialisierung



Spieler Registrieren



Aufgabe 2.1 A: Meta Game - /games

1.) Implementieren Sie einen Service so, dass dieser über den /games-Service die folgenden Funktionen anstoßen kann:

- Benutzer können ein neues Spiel eröffnen.
- Eine Liste mit aktuellen Spielen kann geholt werden.
- Benutzer können sich als Spieler registrieren.
- Benutzer können melden, dass sie bereit sind und das Spiel losgehen kann bzw. ihr Zug abgeschlossen ist.
- Während das Spiel im Status „running“ ist, muss der aktuelle Spieler abfragbar sein und automatisch zum nächsten Spieler wechseln, wenn ein Spieler „ready“ meldet.

Hierbei soll darauf geachtet werden, dass auch die weiteren benötigten Komponenten erstellt werden (z.B. ein Spielbrett!). Dazu sollen, so weit gegeben, die Komponenten benutzt werden, die in dem Request genannt werden.

2.) Für die Koordination von Komponenten werden häufig Mutexe verwendet. In diesem Fall darf nur ein Spieler zur Zeit an der Reihe sein und aktiv Aktionen ausführen. Hierfür implementieren Sie einen verteilten Mutex unter /games/{gameid}/turn

Dieser wird in der nachfolgenden Aufgabe benutzt.

Aufgabe 2.2 A: Keep on rolling!

Implementieren Sie den /boards-Service, so dass

- der Benutzer zum Würfeln einen Wurf an das Board übergeben kann
- die Position der Spielfigur vom Service verändert wird
- die Position abgefragt werden kann
- der Zustand des Brettes abgefragt werden kann

Synchronisazion: Achten Sie darauf, dass nicht zwei Spieler gleichzeitig Würfeln können! Nutzen Sie hierzu einen verteilten Mutex (siehe oben). Ein Spieler darf nur mit dem /boards-Service agieren, wenn dieser erfolgreich den Mutex erworben hat.

- Der Mutex kann erworben werden mittels
  `put /games/{gameid}/turn`
- Es kann abgefragt werden, ob ein Spieler den Mutext hält mit
  `get /games/{gameid}/turn`
- Freigegeben wird der Mutex durch
  `delete /games/{gameid}/turn`
- Der Mutex sollte frei gegeben werden direkt bevor der Zug beendet wird.
- Der /board-Service sollte Überprüfen, ob der würfelnde Spieler den Mutex hält!

Aufgabe 2.3 B: Der Klient

Implementieren Sie einen Client für Restopoly! Dieser soll von einem menschlichen Spieler bedient werden und derzeit ermöglichen

- Die aktuellen (offenen) Spiele anzuzeigen.
- Einem ausgewählten Spiel beizutreten.
- Auf dem Board zu würfeln, wenn man dran ist.

Aufgabe 2.2 B: Die /Bank

1.) Implementieren Sie den `/banks`-Service, so dass

- ein Konto erstellt werden kann mit
  `post /banks/{gameid}/players`
- der Kontostand abgefragt werden kann mit
  `get /banks/{gameid}/players/{playerid}`
- Geld von der Bank überwiesen werden kann mit
  `post /banks/{gameid}/transfer/to/{to}/{amount}`
- Geld eingezogen werden kann mit
  `post /banks/{gameid}/transfer/from/{from}/{amount}`
- Geld von einem zu anderen Konto übertragen werden kann mit
  `post /banks/{gameid}/transfer/from/{from}/to/{to}/{amount}`

2.) Transaktion: Achten Sie darauf, dass ein Geldtransfer immer als eine Transaktion zu verstehen ist: entweder wird diese ganz – oder gar nicht – ausgeführt.
Implementieren Sie hierfür die API für Transaktionen und stellen Sie sicher, dass die Aktionen nur gemeinsam ausgeführt werden bei Commit der Transaktion.