Vorbereitung

Die Sprache: Java! Oder etwas anderes…

Java-RMI benötigt offensichtlich Java! Das bedeutet jedoch nicht, dass Sie auch für den REST Java nutzen müssten. Eine Stärke von REST ist es ja gerade unabhängig von Sprachen zu sein.
Für Java gibt es durch uns grundsätzlich mehr Support, aber Ihre anderen (Lieblings-) Sprachen werden auch akzeptiert. Doch es liegt an Ihnen, uns davon zu überzeugen, dass es funktioniert!
Außerdem müssen Sie dann einen lauffähigen Docker-Container vorlegen (siehe Allgemeine Hinweise), welcher Ihre Anwendung ausführt.

Für Java können Sie z.B. folgendes nutzen:

- **Maven2** Projekt aufsetzen
  Maven ist ein sogenannter „build and dependency manager“, der automatisch die angegebenen Dependencies aus den Repositories zieht.
  https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

- **Spark Rest Framework** besorgen und Dependency eintragen
  Spark ist ein simples Rest-Framework auf Basis von Jetty und mit Maven einfach zu integrieren. Vorteil für Sie: Sehr flache Lernkurve!
  http://sparkjava.com

- **Gson** besorgen und Dependency eintragen
  Gson ist eine Library zum Handhaben von Json

Alle angegebenen Libraries und Frameworks sind nur Empfehlungen und damit Vorschläge. Wie schon bei der Sprache an sich gilt: Überzeugen Sie uns, dass alles klappt!

Aber unbedingt besorgen! ... eine REST-Test-Erweiterung für Ihren Browser, z.B. für Firefox **REST Easy**

Viel Erfolg!

Allgemeine Hinweise

Docker
Die von Ihnen entwickelten Services werden in Docker-Containern laufen. Wer in Java entwickelt und ein Jar (+ Dependencies) produziert, muss sich mit den Details nicht beschäftigen. Es wird ein Upload angeboten, der das Jar in einen Docker-Container lädt und den Service startet. Der Prozess ist ausführlich beschrieben in der Anleitung zur Abgabe.

Pro Container wird nur ein Port nach außen freigegeben. Dieser wird auf einen Port der Host-Maschine abgebildet, so dass auch von außen mit dem Container kommuniziert werden kann. **Untereinander können die Container frei kommunizieren, d.h. alle Ports sind erreichbar!**

Wer sich etwas spezielles wünscht, wie z.B. eine andere Programmiersprache, bestimmte Pakete als Voraussetzung, muss einen eigenen Docker-Container bauen. Dieser Container wird dann eingereicht und gestartet. Somit ist darauf zu achten, dass der Container ohne weitere Parameterübergabe verlässlich startbar ist.

Folgende Links sollten helfen, um eigene Container zu bauen:
- https://docs.docker.com/userguide/dockerimages/
- https://docs.docker.com/articles/dockerfile_best-practices/

Gruppen
Die Aufgaben sind in 4er Gruppen zu bearbeiten aus jeweils zwei 2er Teams. Aufgaben gekennzeichnet mit A und B werden nur von jeweils einem der beiden Teams gefordert. Aufgaben ohne A oder B sind von allen Teams zu bearbeiten. Es ist erforderlich, dass sich die Teams einer Gruppe stark untereinander austauschen, da nur gemeinsam ein Gesamtsystem entsteht.

Optional
Es wird immer wieder optionale Aufgaben geben mit denen Sie extra glänzen können. Diese sind nicht verbindlich und müssen nicht bearbeitet werden. Für die wirklich schnellen ist es jedoch eine Option die Anwendung weiter voran zu treiben und am Ende evtl. ein komplettes Spiel vorlegen zu können!

Aufgabe 1.1: Java RMI Dice

Implementieren Sie zunächst einen Würfel als RMI-Service mit Java. Der RMI-Service muss folgendes Interface erfüllen:

```
public interface DiceRMI extends Remote {
     Roll roll() throws RemoteException;
}
```

wobei gilt:

```
public class Roll implements Serializable{
     private static final long serialVersionUID = 1337L;
     private int number;
     public Roll(int number) {
                                this.number = number; }
     public int getNumber() {
                                return this.number; }
}
```

Aufgabe 1.2: Project RESTopoly

Das große Ziel ist es, Monopoly als verteilte Anwendung zu implementieren. Endgültig wird es eine Anwendung aus REST-Services werden. Sie werden dabei versuchen, so viele Spielelemente wie möglich umzusetzen, nicht alles ist verpflichtend, aber mit den anderen Elementen macht es mehr Spaß.

Bei RESTopoly werden die Entitäten als eigenständige Services mit REST-Interface implementiert.

Aufgabe 1.2.1: Die Würfel – REST /dice

Implementieren Sie einen Würfel als REST-Service. Dieser Service muss das folgende Interface erfüllen (in RAML)

```
/dice:
  get:
    description: Gives you a single dice roll
    responses:
      200:
        body:
          application/json:
            schema: |
                      { "type": "object",
                          "properties": {
                            "number": {"type": "int"},
                            "required": true
                          },
                        }
            example: '{ "number": 4 }'
```

Aufgabe 1.2.2 A: API Spezifikation

RAML ist eine weit verbreitete Spezifikationssprache für REST-Interfaces. Bevor etwas implementiert werden kann, braucht es eine hinreichende Spezifikation, gerade wenn verteilte Teams verteilte Anwendungen entwickeln!

Erstellen Sie eine sinnvolle API-Spezifikation in RAML (http://raml.org/) für einen frei wählbaren Service aus der Liste im Sinne von RESTopoly:

- **/games** – Meta-Ebene des Spieles, verwaltet Spieler und Runden
- /**boards** – Spielbrett, verwaltet Platzierungen
- **/banks** – Bank, die alle Geldangelegenheiten handhabt

/brokers – Verwaltung von Grundstücken und Mieten

/player – Der Spieler selbst

/decks – Die Kartenstapel für Ereignis- und Gesellschaftskarten

/jail – das Gefängnis

Dabei sollen mindestens zwei Methoden verwendet werden, mindestens eine Antwort und eine Parameterübergabe spezifiziert werden.

Aufgabe 1.2.2 B: Replication

Der /dice-REST-Service ist überlastet! Durch die „rechenintensive“ Berechnung der Zufallszahlen, kann schon mal ein einziger Server in Bedrängnis kommen.

Um dennoch ein flüssig lauffähiges System zu gewährleisten, müssen Sie den /dice-Service replizieren und mittels eines Load-Balancers die Last verteilen.

Nutzen Sie hierzu HAProxy (http://www.haproxy.org/)

HAProxy ist eine Anwendung, die sowohl als Reverse-Proxy, als auch als Load-Balancer Verwendung findet. Informieren Sie sich darüber, wie man HAProxy als Load-Balancer konfiguriert und Erstellen Sie eine Konfiguration für mindestens zwei Server, über die die Last verteilt wird.

Erstellen Sie dann zwei /dice-Services als Docker-Container und richten Sie für diese zwei Server einen Load-Balancer ein.

Achten Sie darauf, dass die Konfiguration explizite IP-Adressen benötigt.

Optional 1.3: /decks

Implementieren Sie einen /decks-Service, welcher die Ereignis- bzw. Gesellschaftskarten aushändigt.

```
/decks/{gameid}:
  /chance:
    get:
      description: Gets a chance card
      responses:
        200:
          body:
            application/json:
              schema: card
              example: |
                { "name": "Go to Jail",
                  "text": "Go to jail, do not
                              travel across 'go' and don't
                              receive $200 " }
```

```
/community:
  get:
    description: Gets a community card
    responses:
      200:
        body:
          application/json:
            schema: card
            example: |
              { "name": "Go to Jail",
                "text": "Go to jail, do not
                            travel across 'go' and don't
                            receive $200 " }
```