UE4 –Web Services, Social Media, Advanced JPA (25 Punkte)

In der dritten Übung haben Sie eine serverseitige Lösung auf Basis des Play Frameworks implementiert. Ziel dieses Übungsbeispiels ist es nun, diese Implementierung zu erweitern und um eine Anbindung an das BIG Highscore Web Service bzw. an Twitter zu erweitern. Des Weiteren soll die Verwaltung von Quizfragen und –kategorien innerhalb einer Datenbank ermöglicht werden.

Deadline der Abgabe via TUWEL[1]: **Sonntag, 1. Juni 2014 23:55 Uhr**

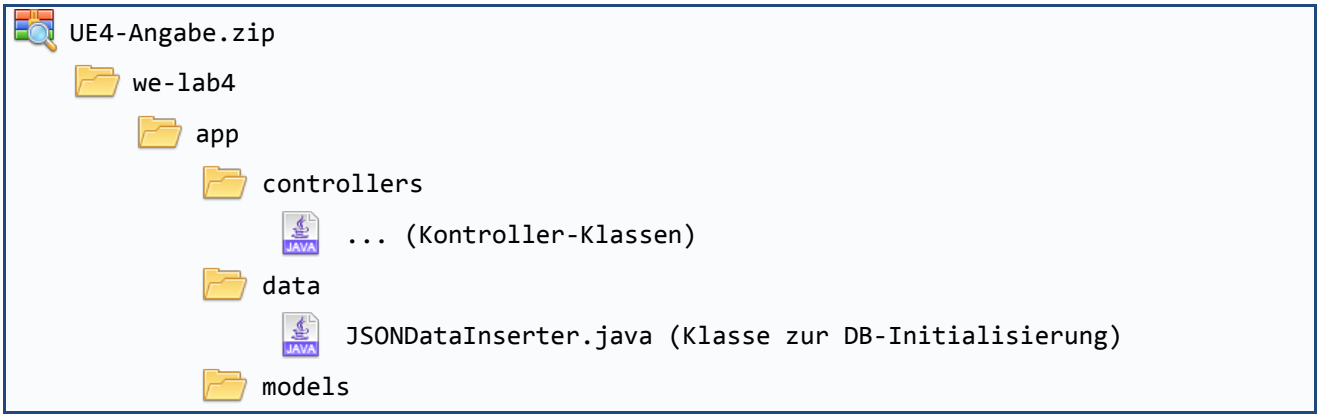
Quiz-Spiel

Beim "Quiz" handelt es sich um ein klassisches Fragespiel mit Mehrfachauswahl ("Multiple Choice"). Registrierte BenutzerInnen müssen während eines Spiels Fragen aus verschiedenen Kategorien beantworten und dabei gegen den Computergegner gewinnen.

Ein Spiel besteht aus fünf Runden und jede Runde aus drei Fragen, die alle einer bestimmten Kategorie zugewiesen sind. Eine Frage kann beliebig viele, aber mindestens zwei, Antwortmöglichkeiten haben, wobei mindestens eine dieser Antwortmöglichkeiten die korrekte Lösung auf die gestellte Frage ist. Eine Frage gilt nur dann als richtig beantwortet, wenn alle korrekten Antwortmöglichkeiten ausgewählt und als Antwort an den Server übermittelt wurden. Zusätzlich gibt es bei jeder Frage ein Zeitlimit, bis zu dem diese Frage beantwortet werden muss. Ist dieses Zeitlimit abgelaufen, wird die aktuelle Auswahl des Spielers als Antwort gewertet und ggf. die nächste Frage gestellt. Am Ende jeder Runde (= nach drei gestellten Fragen aus einer Kategorie), wird der/die RundensiegerIn ermittelt und angezeigt. Der/Die RundensiegerIn ist jene/jener SpielerIn (Mensch oder Computer), welche die meisten Fragen in dieser Runde richtig beantwortet hat. Bei Gleichstand entscheidet die Zeit, die benötigt wurde, um die Fragen zu beantworten (je weniger desto besser). Haben beide SpielerInnen dieselbe Zeit gebraucht um die Fragen zu beantworten, geht die Runde unentschieden aus. Handelt es sich bei der Runde nicht um die letzte Runde, wird eine neue Runde mit einer bisher noch nicht gespielten, zufällig gewählten Kategorie gestartet. Ist die letzte Runde abgeschlossen, wird der/die SpielsiegerIn durch die Anzahl der gewonnenen Runden ermittelt und angezeigt. Herrscht Gleichstand geht das gesamte Spiel unentschieden aus, ansonsten gewinnt der/die SpielerIn mit den meisten Rundensiegen. Nach der Siegesanzeige (= Spielende) kann der/die BenutzerIn ein neues Spiel starten.

Angabe

Diese Angabe umfasst folgende Dateien:



[1] https://tuwel.tuwien.ac.at/course/view.php?id=5399

```
        ... (Modell-Klassen)
    twitter
        ... (Twitter-Klassen)
    views
        ... (Scala-HTML-Templates)
    Global.java (Konfiguration von Applikationsstart und -ende)
conf
    application.conf (Hauptkonfigurations-Datei)
    data.json (Fragen in deutscher und englischer Sprache)
    messages.de (Meldungen in deutscher Sprache)
    messages.en (Meldungen in englischer Sprache)
    routes (Definierte Routen)
    META-INF
        persistence.xml (Hibernate-Einstellungen)
lib
    big-dbpedia-api.jar (DBpedia-API)
project
    build.properties (Marker für das sbt-Projekt)
    plugins.sbt (sbt-Plugins inkl. Play)
public
    images
        ... (Bilder für das Quiz)
    javascripts
        ... (JavaScript-Dateien)
    stylesheets
        main.css (CSS Musterlösung aus Übung 1)
build.sbt (Build-Script für die Anwendung)
JPA-Slides.pdf (Zusätzliche Folien für JPA)
UE4-Angabe.pdf (Dieses Dokument)
```

Teil 1 – Highscore Service

Nach erfolgreichem Abschluss eines Spiels sollen die Spielergebnisse automatisch auf einem Highscoreboard gepostet werden. Für jedes Spiel soll dabei jeweils genau einmal das Highscoreergebnis gespeichert werden. Highscoreergebnisse können mittels eines SOAP/WSDL Web Services gepostet werden.

URL des Highscoreboards:
http://playground.big.tuwien.ac.at:8080/highscore/

URL des Highscore Services:
http://playground.big.tuwien.ac.at:8080/highscore/PublishHighScoreService?wsdl

User-Key:
rkf4394dwqp49x

Die Highscoreergebnisse werden mit Hilfe der Datenstruktur übermittelt, welche Sie im Rahmen der VU Semistrukturierte Daten entworfen haben. Dabei müssen vom Schema jedoch nicht alle Attribute und Elemente übermittelt werden, sondern nur jene, die in nachfolgendem Ausschnitt beschrieben sind.

- Ein Servicerequest wird nur dann verarbeitet, wenn Sie im `<UserKey>` Element den korrekten UserKey angeben.
- Es müssen genau zwei `<user>` Elemente angeben werden – ein `<user>` für den Gewinner und ein `<user>` für den Verlierer.
- Im Falle des Verlierers steht im `name` Attribut des `<User>` "loser", im Falle des Gewinners "winner". Andere Werte sind nicht zulässig.
- Das `<password>` Element muss leer bleiben.
- Im `<firstname>` Element wird der Vorname des Spielers übermittelt.
- Im `<lastname>` Element wird der Nachname des Spielers übermittelt.
- Im `<birthdate>` Element wird das Geburtsdatum des Spielers übermittelt.
- Für die einzelnen Werte eines Benutzers übernehmen Sie die Daten, welche bei der Anmeldung des Benutzers erfasst wurden.
- Für den zweiten Benutzer (= Ihr PC) machen Sie sinnvolle Annahmen.

Wenn Sie alle Elemente und Attribute korrekt übergeben haben, bekommen Sie im Response eine UUID retourniert, welche Sie dann für die nächste Teilaufgabe benötigen. Wenn Sie fehlerhafte Daten übergeben, werden entsprechende Fehlermeldungen vom Service retourniert.

Beispielrequest:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:data="http://big.tuwien.ac.at/we/highscore/data">
  <soapenv:Header/>
  <soapenv:Body>
    <data:HighScoreRequest>
      <data:UserKey>rkf4394dwqp49x</data:UserKey>
       <quiz>
        <users>
         <user name="loser" gender="male">
            <password></password>
            <firstname>Martin</firstname>
            <lastname>Moser</lastname>
            <birthdate>1982-01-03</birthdate>
         </user>
         <user name="winner" gender="female">
            <password></password>
            <firstname>Barbara</firstname>
            <lastname>Reistendorfer</lastname>
            <birthdate>1985-03-03</birthdate>
         </user>
        </users>
        </quiz>
    </data:HighScoreRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Zusammenfassung der geforderten Funktionalität von Teil 1:

- Highscoreergebnisse müssen auf dem Highscoreboard veröffentlicht werden und die UUID muss korrekt retourniert werden.
- Damit man das korrekte Ansprechen des Highscore Services nachvollziehen kann, muss die retournierte UUID als Logging-Ausgabe ausgegeben werden. Benutzen Sie für das Logging die Klasse `play.Logger`. Loggen Sie sowohl im Erfolgs-, als auch im Fehlerfall.
- Für den Fall, dass das Highscore Service nicht erreichbar ist, muss ein entsprechendes Exception Handling vorhanden sein (der User darf zB keinen HTTP 500 Fehler sehen).

Teil 2 – Twitter Integration

Die in Aufgabe 1 gewonnene UUID soll in diesem Schritt per Aufruf der Twitter API als Tweet zu Twitter geschickt werden. Dabei sind folgende zwei Java Klassen verpflichtend zu verwenden:

- TwitterStatusMessage.java
- ITwitterClient.java

Der Text, welcher als Tweet auf Twitter gepostet werden soll, wird durch Aufruf der Methode

```
public String getTwitterPublicationString()
```

in der Klasse `TwitterStatusMessage` retourniert. Ihr TwitterClient muss das Interface `ITwitterClient` implementieren.

Twitter URL: https://twitter.com/BIGEWA2013 (kein Rechtschreibfehler - es wird der Twitter-Feed vom letzten Jahr verwendet)

Consumer Key:
GZ6tiy1XyB9W0P4xEJudQ

Consumer Secret:
gaJDlW0vf7en46JwHAOkZsTHvtAiZ3QUd2mD1x26J9w

Access Token:
1366513208-MutXEbBMAVOwrbFmZtj1r4Ih2vcoHGHE2207002

Access Token Secret:
RMPWOePlus3xtURWRVnv1TgrjTyK7Zk33evp4KKyA

Zusammenfassung der geforderten Funktionalität von Teil 2:

- Die UUID, welche durch den Aufruf des Highscore Service gewonnen wurde, soll als Tweet auf Twitter veröffentlicht werden
- Nach erfolgreichem Aufruf des Highscore Services und dem erfolgreichem Posten der UUID auf Twitter, soll dem Benutzer eine Statusmeldung „UUID xyz wurde auf Twitter veröffentlicht“ angezeigt werden.
- Damit man das korrekte Ansprechen der Twitter API nachvollziehen kann, muss eine Logging-Ausgabe ausgegeben werden. Benutzen Sie für das Logging die Klasse `play.Logger`. Loggen Sie sowohl im Erfolgs-, als auch im Fehlerfall.
- Für den Fall, dass das Publizieren auf Twitter fehlgeschlagen ist, muss ein entsprechendes Exception Handling vorhanden sein (der User darf zB keinen HTTP 500 Fehler sehen)

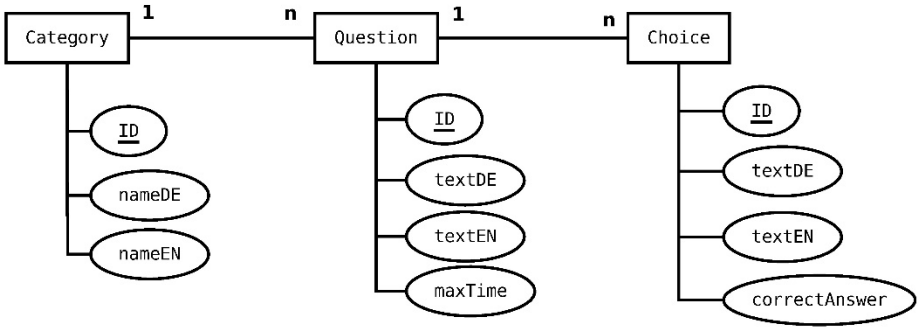
Teil 3 – Advanced JPA

Bisher haben Sie die einzelnen Kategorien, Fragen und Antworten aus JSON Dateien bezogen. In dieser Teilaufgabe sollen Sie die bestehende Datenbank erweitern und die Möglichkeit schaffen, dass auch Kategorien, Fragen und Antworten gespeichert werden können. Im Unterordner /app/models finden Sie die folgenden vier Entityklassen:

- BaseEntity
- Category
- Question
- Choice

Annotation der Entityklassen und Realisierung des DAO Interfaces

Annotieren Sie die oben genannten vier Entityklassen mit den korrekten JPA Annotationen, sodass die folgende DB-Struktur erfüllt ist.



Im Anschluss implementieren Sie das Interface IQuizDAO in der Klasse QuizDAO und realisieren die konkreten Implementierungen der Methoden. Es wird empfohlen, dass Sie die erzeugten Entityklassen und die implementierten Methoden zuerst mit Hilfe von Unittests testen, bevor Sie diese in Ihre Anwendung einbauen.

In den Unterlagen zu dieser Übungsangabe finden Sie auch die Datei JPA-Slides.pdf, in welcher unter anderem das Konzept von 1:n Beziehungen in JPA genau erläutert ist. Die dazugehörigen Beispiele in Java finden Sie unter https://github.com/pliegl/we2014.

Teil 4 – Linked Open Data

In der Ihnen zur Verfügung gestellten Applikation werden bei Applikationsstart die Fragen aus der JSON-Datei (`data.json`) gelesen und in die konfigurierte Datenbank gespeichert (siehe `Global.java` und `JSONDataInserter.java`). Hierfür wird die von Ihnen zu implementierende `persist`-Methode des DAO-Objekts (QuizDAO) verwendet.
Vor Beginn jedes Spiels werden alle Kategorien samt Fragen und Antwortmöglichkeiten aus der Datenbank gelesen und in einem Spiel-Objekt (`QuizGame`) gespeichert, welches den Status des aktuellen Spiels beinhaltet.
Erweitern Sie nun den Datenbestand in der Datenbank, in dem Sie nachdem die JSON-Datei eingelesen wurde, selbständig Fragen auf Basis von DBPedia-Daten erzeugen und in die Datenbank speichern. Ziel ist es, dass Sie 5 unterschiedliche Fragen samt falscher und richtiger Antwortmöglichkeiten zu einer bestimmten Kategorie erzeugen. Die Auswahl der Kategorie ist Ihnen dabei frei gestellt und das Thema muss auch nichts mit Informatik zu tun haben. Achten Sie jedoch darauf, dieselben Fragen sowohl auf Deutsch als auch auf Englisch zu erstellen.

Hinweise

Einbinden von DBpedia

Das DBpedia Projekt[2] stellt Daten aus Wikipedia als Linked Open Data (LOD) zur Verfügung. Die Daten werden dabei aus Wikipedia extrahiert, klassifiziert und entsprechend den LOD-Prinzipien aufbereitet und miteinander verlinkt. Über einen SPARQL Endpoint kann man auf diese Daten zugreifen[3]. Grundsätzlich werden diese Daten als RDF-Graphen retourniert. Ein RDF-Graph besteht aus einer Menge von Statements, die aus den Tripeln Subject-Prädikat-Object bestehen.

Beispielsweise würde die folgende SPARQL-Abfrage[4] alle Subjekte retournieren, für welches ein Statement mit dem Prädikat `subject` und dem den Wert `Category:French_films` für dieses Prädkat existiert. Das Prädikat subject ist in dem Vokabular von PURL definiert, die Kategorie französischer Filme ist durch ein Vokabular in DBPedia definiert.

```
SELECT ?film
WHERE { ?film
        <http://purl.org/dc/terms/subject>
        <http://dbpedia.org/resource/Category:French_films> }
```

Sie können Ihre Abfragen unter http://dbpedia.org/sparql testen.

Um Ihnen den Zugriff auf die DBpedia-Daten zu erleichtern, stellen wir Ihnen eine API zur Verfügung, die im Wesentlichen aus zwei Klassen (`DBPediaService`, `SelectQueryBuilder`) sowie verschiedenen Vokabulardefinitionen (zB `Skos` oder `DBPediaOWL`) besteht. Diese API basiert auf dem Apache Jena Framework, welches einerseits noch weitere Vokabulardefinitionen (zB `FOAF` oder `RDFS`) aber auch das Datenmodel für RDF-Graphen zur Verfügung stellt. RDF-Graphen sind in Jena als Model definiert, Statements als Statement, Subjekte als `Resource`, Prädikate als `Property` und Objekte als `RDFNode` (Super-Klasse von `Resource`, aber auch `Literal`-Werten, zB Zahlen oder Strings).

Die `DBPediaService`-Klasse bietet unter anderem Möglichkeiten um die Verfügbarkeit von DBPedia zu überprüfen (`isAvailable`), SPARQL-Abfragen auszuführen (`loadStatements`), den Namen von Resourcen zu ermitteln (`getResourceName`) und Modelle auszugeben (`writeModel`).

Die `SelectQueryBuilder`-Klasse kann verwendet werden um SPARQL-Abfragen aufzubauen. Dabei kann man folgende Kriterien definieren:

`WHERE`: Statements, die für die gesuchten Subjekte/Resourcen vorhanden sein müssen.

`MINUS`: Statements, die für die gesuchten Subjekte/Resourcen nicht vorhanden sein dürfen.

`FILTER`: Prädikate/Property, die bestimmte Werte (Objekt/RDFNode) aufweisen müssen.

`PredicateExists`: Ein Prädikat muss vorhanden sein, egal welchen Wert es hat.

`PredicateNotExists`: Ein Prädikat darf nicht vorhanden sein.

Der SelectQueryBuilder stellt jedoch nicht die gesamte Funktionalität von SPARQL zur Verfügung (zB fehlende Aggregationsfunktionen). Sie können daher die DBPediaService-Klasse auch direkt mit SPARQL-Abfrage-Strings verwenden.

Für die genauere Verwendung berücksichtigen Sie bitte die JavaDoc-Kommentare in den entsprechenden Klassen und Methoden.

[2] http://dbpedia.org
[3] http://wiki.dbpedia.org/OnlineAccess
[4] http://www.w3.org/TR/sparql11-query/

Beispiel: Man möchte alle Filme, in denen Johnny Depp gespielt und Tim Burton Regie geführt hat, finden. Danach alle Filme, in denen Johnny Depp gespielt hat, Tim Burton aber nicht Regie geführt hat. Die Namen aller Filme werden sowohl in Deutsch als auch in Englisch benötigt.

```
// Check if DBpedia is available
if(!DBPediaService.isAvailable()) {
   Logger.info("DBpedia is currently not available.");
   return;
}

// Resource Tim Burton is available at http://dbpedia.org/resource/Tim_Burton
// Load all statements as we need to get the name later
Resource director = DBPediaService.loadStatements(DBPedia.createResource("Tim_Burton"));

// Resource Johnny Depp is available at http://dbpedia.org/resource/Johnny_Depp
// Load all statements as we need to get the name later
Resource actor = DBPediaService.loadStatements(DBPedia.createResource("Johnny_Depp"));

// retrieve english and german names, might be used for question text
String englishDirectorName = DBPediaService.getResourceName(director, Locale.ENGLISH);
String germanDirectorName = DBPediaService.getResourceName(director, Locale.GERMAN);

String englishActorName = DBPediaService.getResourceName(actor, Locale.ENGLISH);
String germanActorName = DBPediaService.getResourceName(actor, Locale.GERMAN);

// build SPARQL-query
SelectQueryBuilder movieQuery = DBPediaService.createQueryBuilder()
       .setLimit(5) // at most five statements
       .addWhereClause(RDF.type, DBPediaOWL.Film)
       .addPredicateExistsClause(FOAF.name)
       .addWhereClause(DBPediaOWL.director, director)
       .addFilterClause(RDFS.label, Locale.GERMAN)
       .addFilterClause(RDFS.label, Locale.ENGLISH);

// retrieve data from dbpedia
Model timBurtonMovies = DBPediaService.loadStatements(movieQuery.toQueryString());

// get english and german movie names, e.g., for right choices
List<String> englishTimBurtonMovieNames =
   DBPediaService.getResourceNames(timBurtonMovies, Locale.ENGLISH);
List<String> germanTimBurtonMovieNames =
   DBPediaService.getResourceNames(timBurtonMovies, Locale.GERMAN);

// alter query to get movies without tim burton
movieQuery.removeWhereClause(DBPediaOWL.director, director);
movieQuery.addMinusClause(DBPediaOWL.director, director);

// retrieve data from dbpedia
Model noTimBurtonMovies = DBPediaService.loadStatements(movieQuery.toQueryString());

// get english and german movie names, e.g., for wrong choices
List<String> englishNoTimBurtonMovieNames =
   DBPediaService.getResourceNames(noTimBurtonMovies, Locale.ENGLISH);
List<String> germanNoTimBurtonMovieNames =
   DBPediaService.getResourceNames(noTimBurtonMovies, Locale.GERMAN);
```

Abgabemodalität

Beachten Sie die allgemeinen Abgabemodalitäten des TUWEL-Kurses[5]. Zippen Sie Ihre Abgabe, sodass sie die folgende Struktur aufweist. Alle Dateien müssen UTF-8 codiert sein!
ACHTUNG: Wird das Abgabeschema nicht eingehalten, so kann es zu Punkteabzügen kommen!

[5] https://tuwel.tuwien.ac.at/course/view.php?id=5399

- UE4-Angabe.zip
  - we-lab4
    - app
      - controllers
        - ... (Kontroller-Klassen, erweitert)
      - data
        - JSONDataInserter.java (Klasse zur DB-Initialisierung, unverändert)
      - highscore
        - ... (HighScore-Client und generierte Klassen)
      - models
        - ... (Modell-Klassen, mit JPA erweitert)
      - twitter
        - ... (Twitter-Klassen, erweitert)
      - views
        - ... (Scala-HTML-Templates, unverändert)
      - Global.java (Konfiguration von Applikationsstart und -ende, unverändert)
    - conf
      - application.conf (Hauptkonfigurations-Datei, unverändert)
      - data.json (Fragen in deutscher und englischer Sprache, unverändert)
      - messages.de (Meldungen in deutscher Sprache, unverändert)
      - messages.en (Meldungen in englischer Sprache, unverändert)
      - routes (Definierte Routen, unverändert)
      - META-INF
        - persistence.xml (Hibernate-Einstellungen, unverändert)
    - lib
      - big-dbpedia-api.jar (DBpedia-API, unverändert)
    - project
      - build.properties (Marker für das sbt-Projekt, unverändert)
      - plugins.sbt (sbt-Plugins inkl. Play, unverändert)
    - public
      - images
        - ... (Bilder für das Quiz)
      - javascripts
        - ... (JavaScript-Dateien)
      - stylesheets
        - main.css (CSS Musterlösung aus Übung 1)
    - build.sbt (Build-Script für die Anwendung)