

## UE3 – Play Framework, JPA (25 Punkte)

In der zweiten Übung haben Sie eine serverseitige, MVC2-basierte Lösung auf Basis von Java-Technologien implementiert. Ziel dieses Übungsbeispiels ist es nun, diese Implementierung mit Hilfe des Play-Frameworks<sup>1</sup> umzusetzen und zu erweitern sowie eine Datenbankbindung zu integrieren.

Deadline der Abgabe via TUWEL<sup>2</sup>: **Sonntag, 11. Mai 2014 23:55 Uhr**

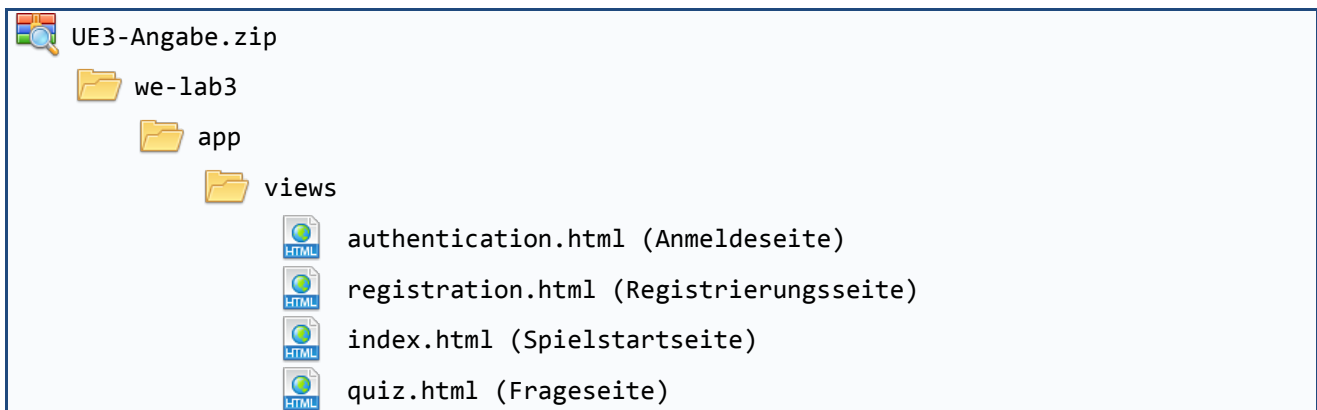
### Quiz-Spiel

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.

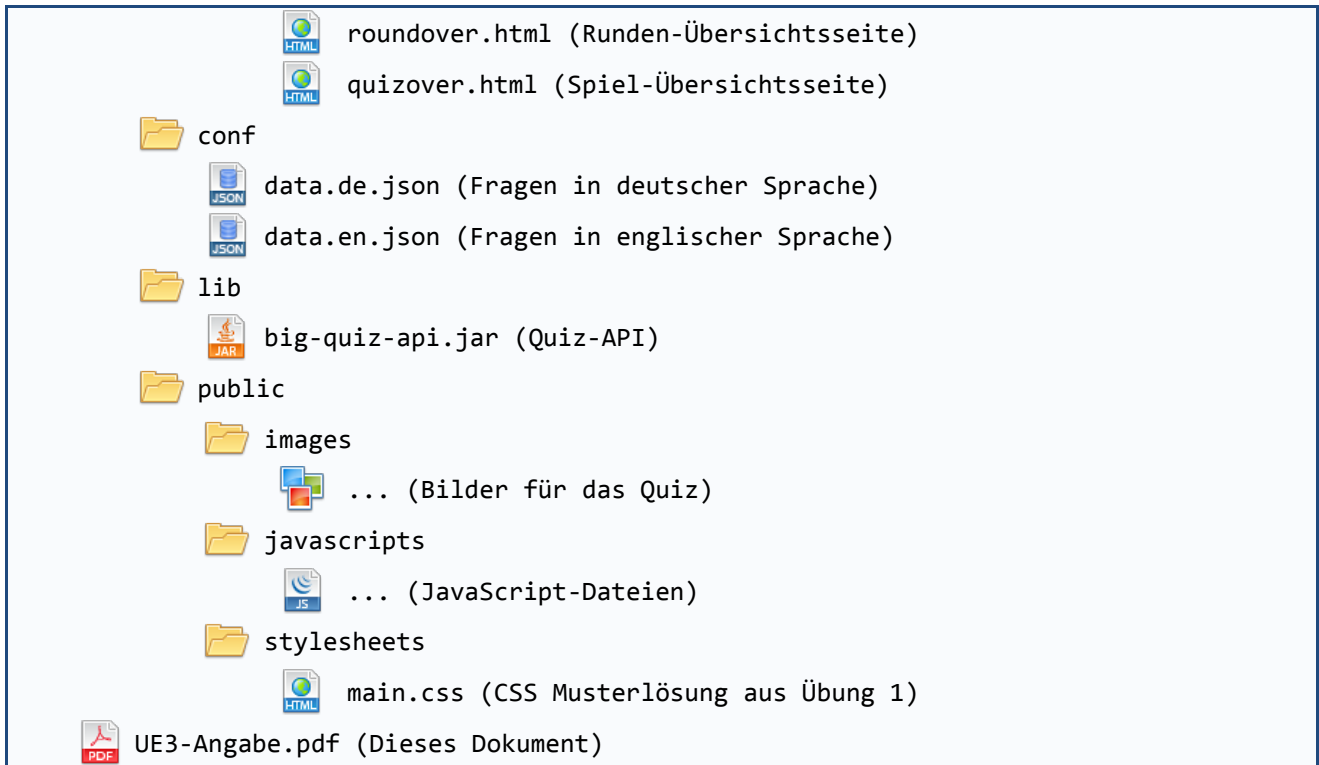
### Angabe

Diese Angabe umfasst folgende Dateien:



<sup>1</sup> <http://www.playframework.com>

<sup>2</sup> <https://tuwel.tuwien.ac.at/course/view.php?id=5399>



Implementieren Sie mit Hilfe des Play Frameworks eine MVC2-Web-Applikation und achten Sie, wie auch mit bei Übung 2, auf eine Trennung von Model, View (nun Scala-HTML-Templates) und Controller. Legen Sie dazu erst eine Play-Anwendung an (siehe "Hinweise") und kopieren Sie die Dateien aus den Angaberessourcen in die gleichnamigen Verzeichnisse. Stellen Sie sicher, dass Ihre Anwendung mit der per Default erstellten Ressourcen läuft.

Wandeln Sie nun die von uns zur Verfügung gestellten HTML5-Dateien (siehe Angaberessourcen) in entsprechende Scala-HTML-Templates<sup>3</sup> um. Achten Sie darauf, dass dabei alle Element-IDs übernommen werden, da dies ist für die automatisierte Validierung Ihrer Lösung erforderlich ist. Das Aussehen der Seiten soll so gut es geht erhalten bleiben. Diese Templates (Endung muss `.scala.html` sein!) dienen als View. Erstellen Sie außerdem die Controller-Klassen.

Hauptanforderungen an Ihre Implementierung:

- **Seitenfluss:** Wird die Anwendung das erste Mal aufgerufen, soll der/die BenutzerIn auf die Anmeldeseite (`authentication.html`) weiter geleitet werden. Von dort aus, kann er sich entweder anmelden oder auf die Registrierungsseite (`registration.html`) navigieren. Nach erfolgreicher Registrierung wird der Benutzer wieder zurück auf die Anmeldeseite geleitet. Bei Fehlern im Registrierungsformular werden diese Fehler direkt beim Formular angezeigt. Ebenso werden Fehler beim Anmelden direkt bei der Anmeldeseite angezeigt. Hat sich der/die BenutzerIn erfolgreich angemeldet, so wird er/sie auf die Spielstartseite (`index.html`) geleitet, von welcher aus das Spiel gestartet werden kann. Ein gestartetes Spiel zeigt die Frageseite (`quiz.html`) für jede Frage an. Nach jedem Rundenende wird eine entsprechende Übersichtsseite (`roundover.html`) und nach dem Spielende der entsprechende Spielsieger angezeigt (`quizover.html`). Auf der Spielende-Seite hat man zusätzlich die Möglichkeit ein neues Spiel zu beginnen. In diesem Fall wird man direkt auf die Frage-Seite weitergeleitet und nicht auf die Spielstartseite. Meldet sich der/die

<sup>3</sup> <http://www.playframework.com/documentation/2.2.2/JavaTemplates>

SpielerIn während dem Spielen ab (Klick auf "Abmelden") so wird das aktuell laufende Spiel abgebrochen und die Anmeldeseite angezeigt.

Verwenden Sie *Controller*-Klassen<sup>4</sup> und *routes*<sup>5</sup> um den Seitenfluss entsprechend dem Spielzustand zu steuern. Achten Sie auf eine korrekte Verwendung der HTTP-Methoden (GET, POST, etc.).

- *Spielablauf*: Das Spiel muss den am Anfang dieses Dokuments beschriebenen Regeln folgen (siehe Quiz-Spiel). Dabei können Sie davon ausgehen, dass die von uns zur Verfügung gestellten Fragen (data.\*.json) bereits den Kriterien entsprechen (ausreichend Fragenkategorien, mindestens eine korrekte Antwortmöglichkeit etc.).

Implementieren Sie hierfür Controller-Klassen, welche die zur Verfügung gestellte API (big-quiz-api.jar, siehe auch Hinweise) verwenden.

- *Registrierung*: SpielerInnen sollen die Möglichkeit haben sich zu registrieren (register.html) und sich anzumelden (login.html). Die Validierung der Registrierungsfelder soll im Gegensatz zu den bisherigen Übungen nun serverseitig erfolgen, d.h., eine Überprüfung mit Hilfe von JavaScript ist in dieser Übung nicht mehr notwendig. Die Kriterien der Felder können Sie der Angabe aus Übung 1 entnehmen. Zusätzlich gilt natürlich, dass es nicht zwei Benutzer mit dem selben Benutzernamen geben darf.

Hierfür können Sie beispielsweise die Formular-Helper in Play<sup>6</sup> verwenden.

- *Authentication*: SpielerInnen, die nicht eingeloggt sind, sollen auf die anderen Seiten der Anwendung keinen Zugriff haben.

Verwenden Sie hierfür den Security-Mechanismus von Play, d.h., implementieren Sie einen eigenen Security.Authenticator<sup>7</sup> und verwenden Sie entsprechende Annotationen.

- *Datenbankanbindung*: Die Spielerdaten aus der Registrierung sollen in einer In-Memory-Datenbank (H2) gespeichert werden und im Falle eines Logins wieder aus dieser ausgelesen werden.

Implementieren Sie daher Ihre eigene User-Modell-Klasse (welche das Interface User implementiert) und verwenden Sie die Annotationen der Java Persistence API (siehe auch Hinweise).

- *Internationalisierung (i18n)*<sup>8</sup>: Ihre Web-Anwendung soll sowohl für deutschsprachige als auch für englischsprachige BenutzerInnen zugänglich sein. Integrieren Sie diese beiden Sprachen in Ihre Anwendung, in dem Sie sprachspezifische Meldungen extern verwalten und aufgrund der ausgewählten Sprache des Benutzers die korrekte Frage-Datei laden. Es ist für diese Übung ausreichend, die Fragen einmal am Anfang des Spiels korrekt zu laden, eine laufende Überprüfung der Sprache ist nicht notwendig.
- *Standards und Barrierefreiheit*: Das User Interface muss den Anforderungen von XHTML5 sowie WCAG-AA gerecht werden.

---

<sup>4</sup> <http://www.playframework.com/documentation/2.2.2/JavaActions>

<sup>5</sup> <http://www.playframework.com/documentation/2.2.2/JavaRouting>

<sup>6</sup> <http://www.playframework.com/documentation/2.2.2/JavaForms>

<sup>7</sup> <http://www.playframework.com/documentation/2.2.2/JavaGuide4>

<sup>8</sup> <http://www.playframework.com/documentation/2.2.2/JavaI18N>

## Hinweise

### Validierung

Verwenden Sie zur Validierung Ihrer XHTML Dateien den Validator <http://validator.nu/> und für Ihre CSS Dateien den vom W3C zur Verfügung gestellten Validation-Service <http://jigsaw.w3.org/css-validator/>. Beachten Sie, dass der Typ "date" für Eingabefelder derzeit nicht in allen Browsern unterstützt wird. Eine entsprechende Warnung bei der Validierung dürfen Sie in diesem Fall ignorieren. Zur Überprüfung der WAI-Tauglichkeit stehen Ihnen eine Vielzahl von Services im Internet zur Verfügung (z.B. AChecker, <http://achecker.ca/checker/index.php>). Nähere Infos dazu finden Sie in den Folien bzw. im TUWEL.

### Anlegen einer Play-Anwendung













Die folgende Anleitung gibt nur einen kurzen Überblick über den typischen Ablauf der Anwendungserstellung in Play. Für detailliertere Informationen, lesen Sie bitte die Vorlesungsfolien (siehe TUWEL), die darin verlinkten Code-Beispiele, sowie die Dokumentation auf der Play-Webseite.

1. Laden Sie die aktuelle Version des Play-Frameworks (= Version 2.2.2) herunter und befolgen Sie die Anleitung zur Installation des Play-Frameworks<sup>9</sup>.
2. Führen Sie zum Anlegen einer neuen Anwendung in der Konsole/Kommandozeile folgenden Befehl aus<sup>10</sup>:

```
play new we-lab3-group<GruppenNr>
```

Geben Sie beim Applikationsnamen denselben Namen (we-lab3-group<X>) noch einmal ein oder klicken Sie <enter>. Wählen Sie als Template die zweite Option (Java application).




Folgende Dateien und Verzeichnisse sollten für Sie erstellt worden sein<sup>11</sup>:

 we-lab3-group<X>	Anwendungsverzeichnis
 app	Ausführbare Artefakte
 controllers	Java-Controller-
 views	Scala-HTML-Templates
 conf	
 application.conf	Haupt-Konfigurations-Datei
 routes	Definierte Routen ("Redirects")
 project	
 build.properties	Marker für das sbt-Projekt
 plugins.sbt	sbt Plugins inkl. Play
 public	
 images	Bilder
 javascript	JavaScript-Dateien

<sup>9</sup> <http://www.playframework.com/documentation/2.2.x/Installing>

<sup>10</sup> <http://www.playframework.com/documentation/2.2.2/NewApplication>

<sup>11</sup> <http://www.playframework.com/documentation/2.2.2/Anatomy>

 stylesheet	CSS-Dateien
 test	Auto-generierte JUnit-Tests
 build.sbt	Build-Script für die Anwendung

3. Wechseln Sie in das neu erstellte Verzeichnis und starten Sie die Applikation<sup>12</sup>:

```
cd we-lab3-group<GruppenNr>
play start // "play run" for developer mode
```

Während die Applikation kompiliert und gestartet wird, werden im Anwendungsverzeichnis neue Verzeichnisse angelegt: logs für Log-Dateien und target, project/project, und project/target für kompilierte und generierte Ressourcen.

4. Per Default bindet Play die neu erstellte Applikation an den Port 9000. Testen Sie die Applikation, indem Sie die folgende Adresse im Browser aufrufen.

<http://localhost:9000/>

### Integration der H2-In-Memory-Datenbank der JPA in Play<sup>13</sup>

1. Fügen Sie die Abhängigkeiten in Ihrem Build-Script hinzu:

```
build.sbt

[...]
```

```
libraryDependencies += Seq(
  javaJdbc,
  javaCore,
  javaJpa,
  "org.hibernate" % "hibernate-entitymanager" % "4.3.1.Final"
)
```

2. Passen Sie die application.conf an, sodass H2 als Datenquelle verwendet wird und binden Sie die Datenquelle mit einem Namen an JNDI. Kommentieren Sie dafür folgende Absätze wieder ein (= Entfernung von #) oder fügen Sie sie hinzu:

```
conf/application.conf

db.default.driver=org.h2.Driver
db.default.url="jdbc:h2:mem:play"

db.default.jndiName=DefaultDS
jpa.name=defaultPersistenceUnit
jpa.default=defaultPersistenceUnit
```

3. Konfigurieren Sie JPA mit Hilfe einer persistence.xml im conf/META-INF-Verzeichnis:

```
conf/META-INF/persistence.xml

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
```

<sup>12</sup> <http://www.playframework.com/documentation/2.2.2/PlayConsole>

<sup>13</sup> <http://www.playframework.com/documentation/2.2.2/JavaJPA>

```
<persistence-unit name="defaultPersistenceUnit" transaction-type="RESOURCE_LOCAL">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>
  <non-jta-data-source>DefaultDS</non-jta-data-source>
  <properties>
    <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
    <property name="hibernate.hbm2ddl.auto" value="create" />
  </properties>
</persistence-unit>
</persistence>
```

4. Verwenden Sie die JPA-Annotation über den Play-Actions, die auf JPA-Funktionen zugreifen.

```
@play.db.jpa.Transactional
public static Result index() {
    ...
}
```

### Verwendung der beigefügten Spiel-API

Kopieren Sie die beigefügte API (big-quiz-api.jar) in ein (neu erstelltes) Verzeichnis "lib" und fügen Sie die notwendigen Abhängigkeiten in Ihrem Build-Script hinzu:

```
build.sbt

libraryDependencies += Seq(
  [...],
  "com.google.code.gson" % "gson" % "2.2"
)

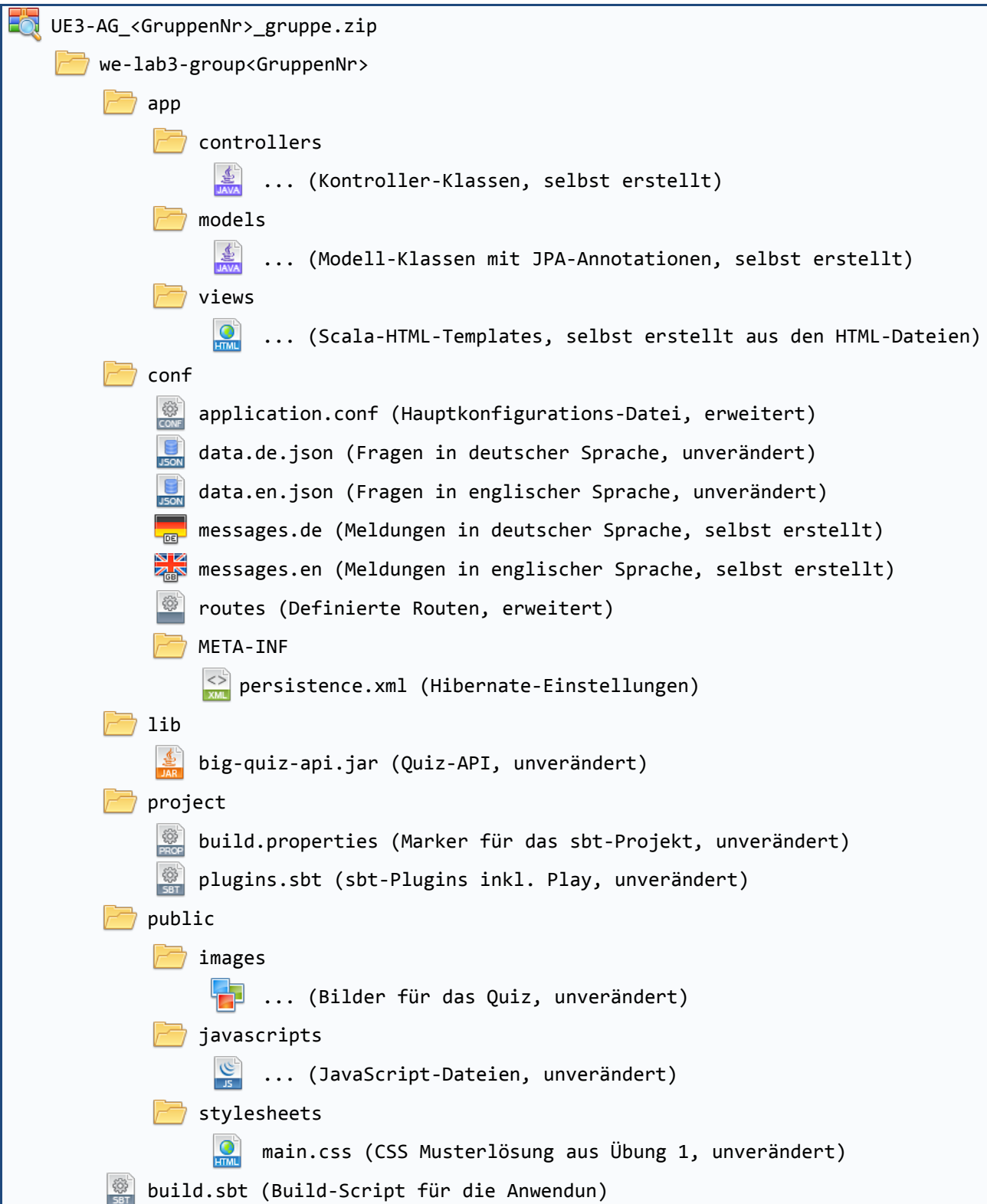
templatesImport += "scala.collection._"
templatesImport += "at.ac.tuwien.big.we14.lab2.api._"
```

Die API selbst können Sie nun wie folgt in Java verwenden. Beachten Sie außerdem die Java-Doc-Kommentare zu den einzelnen Methoden.

```
import at.ac.tuwien.big.we14.lab2.api.*;
...
// user must implement Interface User
QuizFactory factory = new PlayQuizFactory(jsonPath, user);
QuizGame game = factory.createQuizGame();
game.startNewRound(); // start new game/round
game.getPlayers().get(0); // get human player
Round round = game.getCurrentRound(); // current round
round.getAnswer(questionNr, user); // answer of question nr <questionNr> of <user>
Question question = round.getCurrentQuestion(user); // current question of user
question.getAllChoices(); // all possible choices for a question
round.answerCurrentQuestion(user, choices, time); // user with chosen answers and
time left (also automatically answers the question for the computer opponent)
round.getRoundWinner(); // winner of round or null if no winner exists yet
game.isRoundOver(); // check if round is over
game.getWonRounds(user); // number of rounds won by the given user
game.isGameOver(); // check if game is over
game.getWinner(); // winner of round or null if no winner exists yet
```

## Abgabemodalität

Beachten Sie die allgemeinen Abgabemodalitäten des TUWEL-Kurses<sup>14</sup>. 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!

<sup>14</sup> <https://tuwel.tuwien.ac.at/course/view.php?id=5399>