UE3 – JPA, Linked Open Data, Web Services (30 Punkte)

In der zweiten Übung haben Sie mit Hilfe von Java-Technologien eine Web Application mit MVC Architektur implementiert. Ziel dieses Übungsbeispiels ist es nun, die Anwendung um eine Datenbankanbindung zu erweitern und externe Services zu integrieren.

Deadline der Abgabe via TUWEL[1]: **Sonntag, 22. Mai 2016 23:55 Uhr**

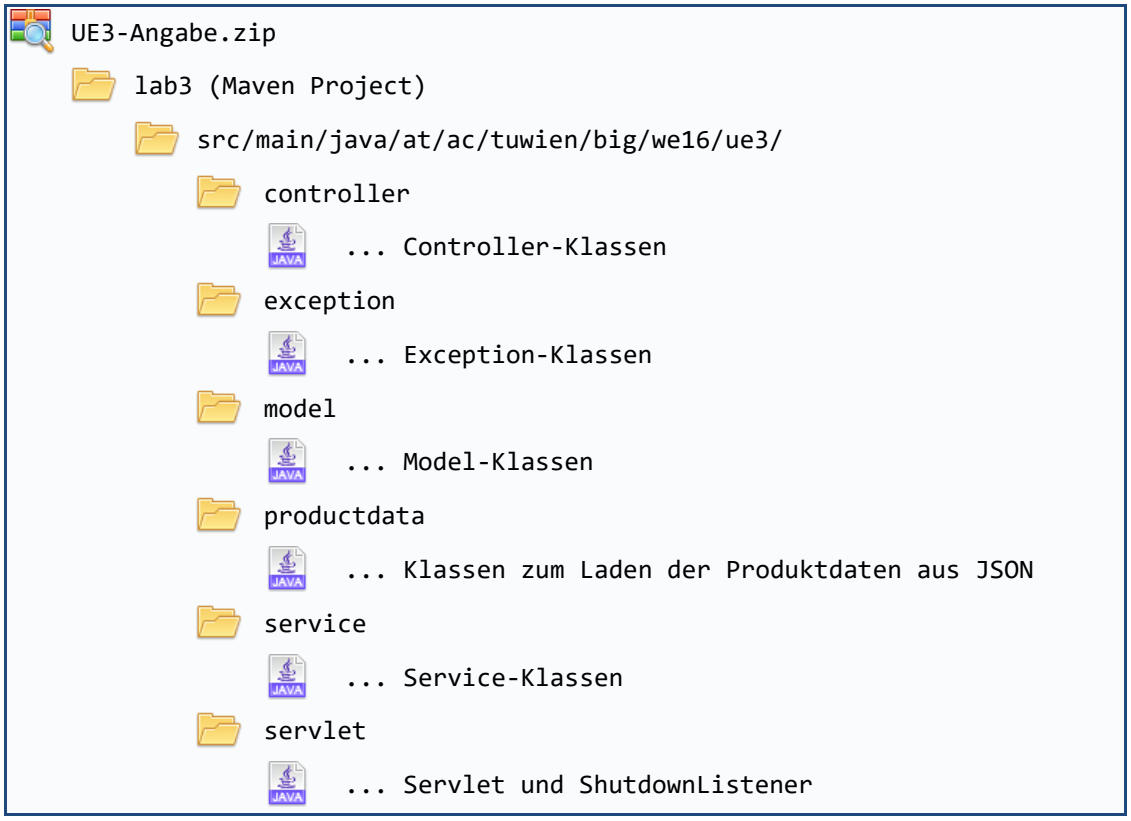
Nur ein Gruppenmitglied muss die Lösung auf TUWEL abgeben.

BIG Bid

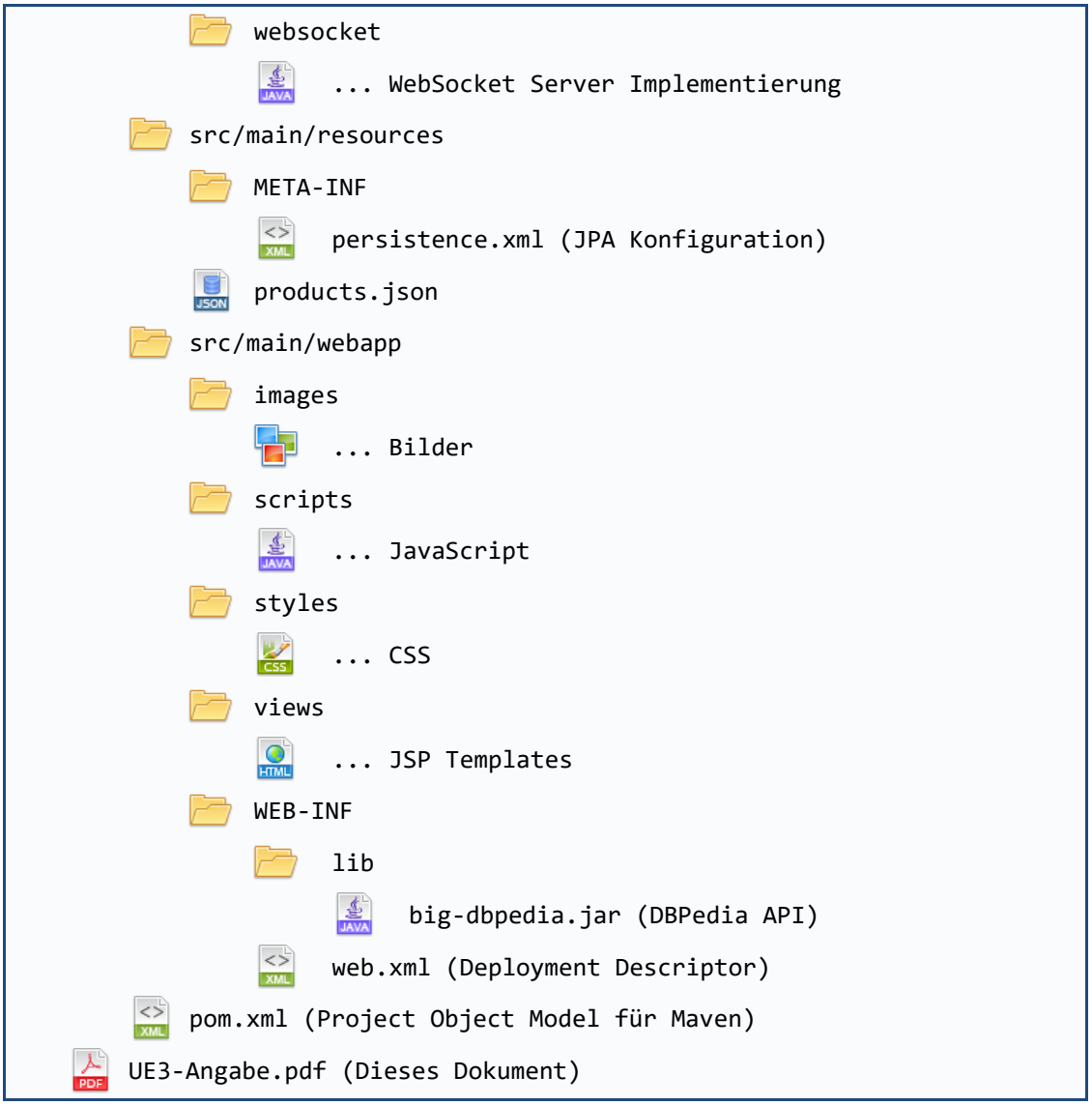
BIG Bid ist eine Online-Plattform, auf der registrierte Benutzerinnen und Benutzer Bücher, CDs und DVDs ersteigern können. Um sicherzustellen, dass die Benutzerinnen und Benutzer nur Gebote abgeben, die sie sich auch tatsächlich leisten können, müssen sie vor der Teilnahme an Auktionen Guthaben bei BIG Bid kaufen. Neue Gebote werden dann sofort von diesem Kontostand abgezogen. Wenn eine Benutzerin oder ein Benutzer überboten wird, dann wird ihr oder ihm der Betrag wieder gutgeschrieben. Nur volljährige Personen können sich bei BIG Bid registrieren.

Angabe

Diese Angabe umfasst folgende Dateien:



[1] https://tuwel.tuwien.ac.at/course/view.php?id=7423



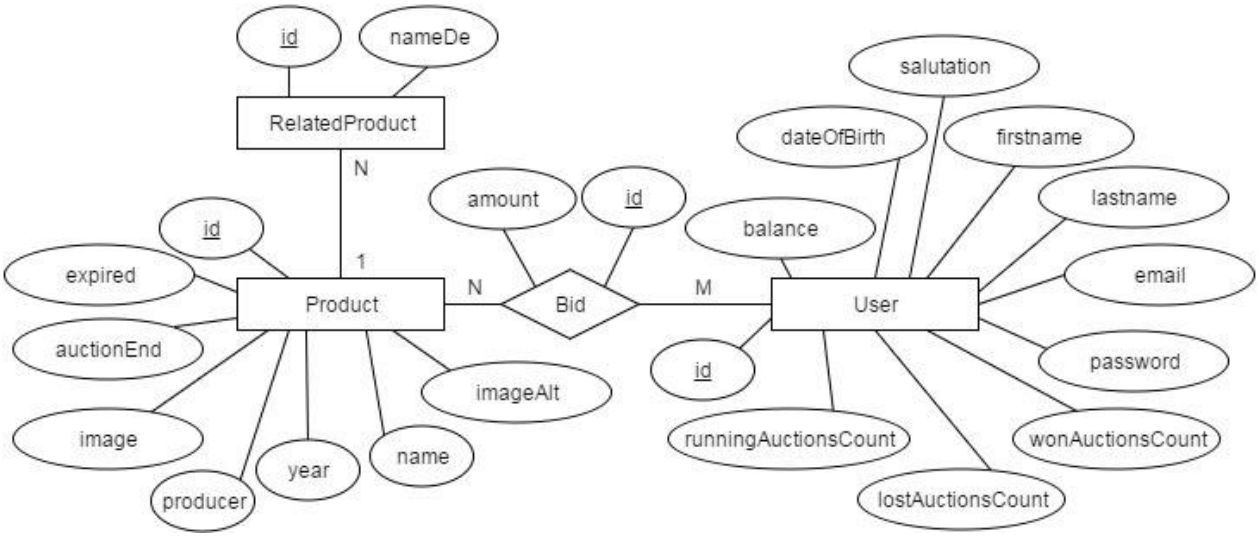
Es steht Ihnen frei, alle von uns zur Verfügung gestellten Klassen und JSP-Dateien zu verändern und zu erweitern. Einige Stellen im Code, an denen Sie auf jeden Fall noch Anpassungen vornehmen müssen, sind mit TODO-Kommentaren markiert. Die Hauptanforderungen an die Funktionsweise von BIG Bid aus der zweiten Übung, die in dieser Angabe bereits implementiert sind, müssen eingehalten werden.

Teil 1 – Registrierung und Login

Neue Benutzerinnen und Benutzer sollen sich bei BIG Bid registrieren können. Das entsprechende Template finden Sie bereits als JSP-Datei in der Angabe. Die Validierung des Registrierungsformulars soll nur serverseitig erfolgen. Eine Validierung mit HTML5 oder JavaScript soll also in dieser Übung nicht umgesetzt werden. Die Validierungsbedingungen entnehmen Sie bitte der Angabe zur ersten Übung. Fehlermeldungen sollen nach dem Abschicken des Formulars direkt beim fehlerhaften Feld angezeigt werden. Jede Benutzerin und jeder Benutzer erhält bei der Registrierung 1.500 € Startguthaben. Die Passwörter der Benutzerinnen und Benutzer werden in dieser Übung unverschlüsselt gespeichert. Anders als in der zweiten Übung sollen die Benutzerdaten nun beim Login überprüft werden.

Teil 2 – JPA

In Übung 2 wurden Daten zu Benutzerinnen und Benutzern, Auktionen und Geboten in Java-Variablen gespeichert. Nun soll die Applikation um eine Datenbankanbindung erweitert und alle Informationen in einer H2 Datenbank abgelegt werden. Die Datenbank soll wie folgt aufgebaut sein:



Verwenden Sie JPA, um mit der H2 in-memory Datenbank zu kommunizieren, und annotieren Sie die bereits zur Verfügung gestellten Model-Klassen entsprechend dem obigen Schema. Eine passende `persistence.xml` finden Sie in der Angabe. Die Tabellen `Product` und `RelatedProduct` werden beim Start der Applikation mit Daten aus `products.json` beziehungsweise DBPedia (siehe Teil „Linked Open Data“ dieser Angabe) befüllt. In der Spalte `producer` der `Product` Tabelle soll den Namen der/des AutorIn, RegisseurIn oder InterpretIn gespeichert werden. Wenn sich eine neue Benutzerin oder ein neuer Benutzer registriert, dann werden ihre oder seine Daten in der Tabelle `User` abgelegt. Wenn die Benutzerin oder der Benutzer an Auktionen teilnimmt, werden die entsprechenden Spalten der `User` Tabelle aktualisiert und ein Eintrag in der `Bid` Tabelle angelegt. Beim Start der Applikation soll automatisch ein Eintrag in der `User` Tabelle angelegt werden. Dieser Benutzer dient als Computeruser (siehe `DataGenerator`, `BigBidServlet` und `ComputerUserService`). Wenn sie möchten können Sie auch weitere Testbenutzer automatisch anlegen, damit Sie sich nicht nach jedem Deployment neu registrieren müssen.

Hinweis zu Eclipse

Sie müssen JPA in Eclipse installieren. Klicken Sie dazu auf „Help“ → „Install New Software“ und installieren Sie „Hibernate Tools“ und „JBoss Maven Hibernate Configurator“.

Teil 3 – Linked Open Data

Auf den Detailseiten der Produkte sollen die Titel ähnlicher Alben, Bücher und Filme angezeigt werden. Diese Titel sollen automatisch beim Start der Applikation aus DBPedia ausgelesen werden. Zu diesem Zweck soll die Klasse `DataGenerator` aus der Angabe so erweitert werden, dass nach dem Auslesen der Produkte aus `products.json` zu jedem Produkt die Namen von fünf weiteren Alben, Büchern oder Filmen der gleichen InterpretInnen, AutorInnen, oder RegiseurInnen aus DBPedia geladen werden. Die Produktnamen sollen in die Tabelle `RelatedProduct` gespeichert werden.

Hinweise zur Anbindung an DBPedia

Das DBpedia[2] Projekt 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[3] kann man auf diese Daten zugreifen. 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 alle Subjekte retournieren, für welche ein Statement mit dem Prädikat `subject` und dem 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 (`big-dbpedia.jar`), die im Wesentlichen aus den zwei Klassen `DBPediaService` und `SelectQueryBuilder` sowie verschiedenen Vokabulardefinitionen (zum Beispiel Skos oder DBPediaOWL) besteht. Diese API basiert auf dem Apache Jena Framework, welches einerseits noch weitere Vokabulardefinitionen (zum Beispiel FOAF oder RDFS) aber auch das Datenmodell 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`.

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/Properties, 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 (zum Beispiel fehlen Aggregationsfunktionen). Sie können daher die `DBPediaService`-Klasse auch direkt mit SPARQL-Abfrage-Strings verwenden. Für die genaue Verwendung berücksichtigen Sie bitte die JavaDoc-Kommentare in den entsprechenden Klassen und Methoden.

Bitte beachten Sie, dass die Daten von DBPedia automatisch aus Wikipedia verarbeitet werden. Dadurch kann es vorkommen, dass einige Datensätze unvollständig sind, insbesondere was den Ressourcenamen in verschiedenen Sprachen betrifft (abfragbar durch `getResourceName` beziehungsweise `getResourceNames`). Die zur Verfügung gestellte API prüft die Prädikate `foaf:name` und `rdfs:label`.

[2] http://dbpedia.org

[3] http://wiki.dbpedia.org/OnlineAccess

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. Die Überprüfung der Namen ist im Beispiel nicht enthalten.

```
// Check if DBpedia is available
If (!DBPediaService.isAvailable()) {
	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);
```

Teil 4 - Web Services

Wenn ein Produkt auf BIG Bid verkauft wurde, dann soll eine entsprechende Nachricht im BIG Bid Board veröffentlicht werden. Das BIG Bid Board ist unter https://lectures.ecosio.com/ verfügbar. Neue Verkäufe sollen über eine REST API im JSON Format an das Board übermittelt werden. Wenn die API erreichbar ist, dann soll jeder Verkauf genau einmal veröffentlicht werden. Wenn eine Auktion ohne Gebot abläuft, dann sollen keine Informationen an das Board gesendet werden.

Der API Endpoint https://lectures.ecosio.com/b3a/api/v1/bids akzeptiert POST Requests mit `Content-Type application/json` im folgenden Format:

```
{
    "name": "John Doe",
    "product": "The Catcher in the Rye",
    "price": "22.30",
    "date": "2016-03-06T11:09:31.701Z"
}
```

Falls alle Felder vorhanden und korrekt befüllt sind, antwortet das Web Service mit HTTP Status Code 200, den gespeicherten Daten und einer UUID, zum Beispiel:

```
{
  "name": "John Doe",
  "product": "The Catcher in the Rye",
  "price": "22.30",
  "date": "2016-03-06T11:09:31.701Z",
  "id": "36cd00a7-e041-4925-aff5-bb21b69cf03f"
}
```

Die UUID soll ausgelesen[4] und im Teil „Twitter API“ dieser Übung an Twitter gesendet werden. Falls der Request ungültig ist, wird mit einer Fehlermeldung und einem entsprechenden Status Code geantwortet. Sollte das BIG Bid Board nicht verfügbar sein, dann werden die Auktionen nicht veröffentlicht. Stellen Sie in diesem Fall sicher, dass Sie alle Exceptions korrekt behandeln. Damit die Anbindung nachvollzogen werden kann, muss im Erfolgsfall die erhaltene UUID und im Fehlerfall eine sinnvolle Fehlermeldung auf `stdout` bzw. `stderr` geloggt werden. Sie können dafür eine Library wie log4j[5] benutzen, wenn Sie möchten.

Teil 5 - Twitter API

Die im Teil „Web Services“ dieser Übung erhaltene UUID soll per Aufruf der Twitter API auf Twitter veröffentlicht werden. Den Inhalt des Tweets erhalten Sie durch Aufruf der Methode `getTwitterPublicationString()` in der Klasse `TwitterStatusMessage`. Veröffentlichen Sie den Rückgabewert dieser Methode auf Twitter. Benutzen Sie für die Kommunikation mit Twitter die twitter4j[6] Library mit den folgenden Zugangsdaten:

Twitter URL[7]:	https://twitter.com/BIGEWA2013
Consumer Key:	GZ6tiy1XyB9W0P4xEJudQ
Consumer Secret:	gaJDlW0vf7en46JwHAOkZsTHvtAiZ3QUd2mD1x26J9w
Access Token:	1366513208-MutXEbBMAVOwrbFmZtj1r4Ih2vcoHGHE2207002
Access Token Secret:	RMPWOePlus3xtURWRVnv1TgrjTyK7Zk33evp4KKyA

Damit die Anbindung nachvollzogen werden kann, muss im Erfolgsfall der Inhalt des abgeschickten Tweets und im Fehlerfall eine sinnvolle Fehlermeldung auf `stdout` bzw. `stderr` geloggt werden. Sie können dafür eine Library wie log4j benutzen, wenn Sie möchten.

---

[4] Sie können dafür die gson Library benutzen, die für das Einlesen der Produkte bereits in das Projekt eingebunden wird.

[5] http://logging.apache.org/log4j

[6] http://twitter4j.org

[7] Es wird der Account von 2013 verwendet

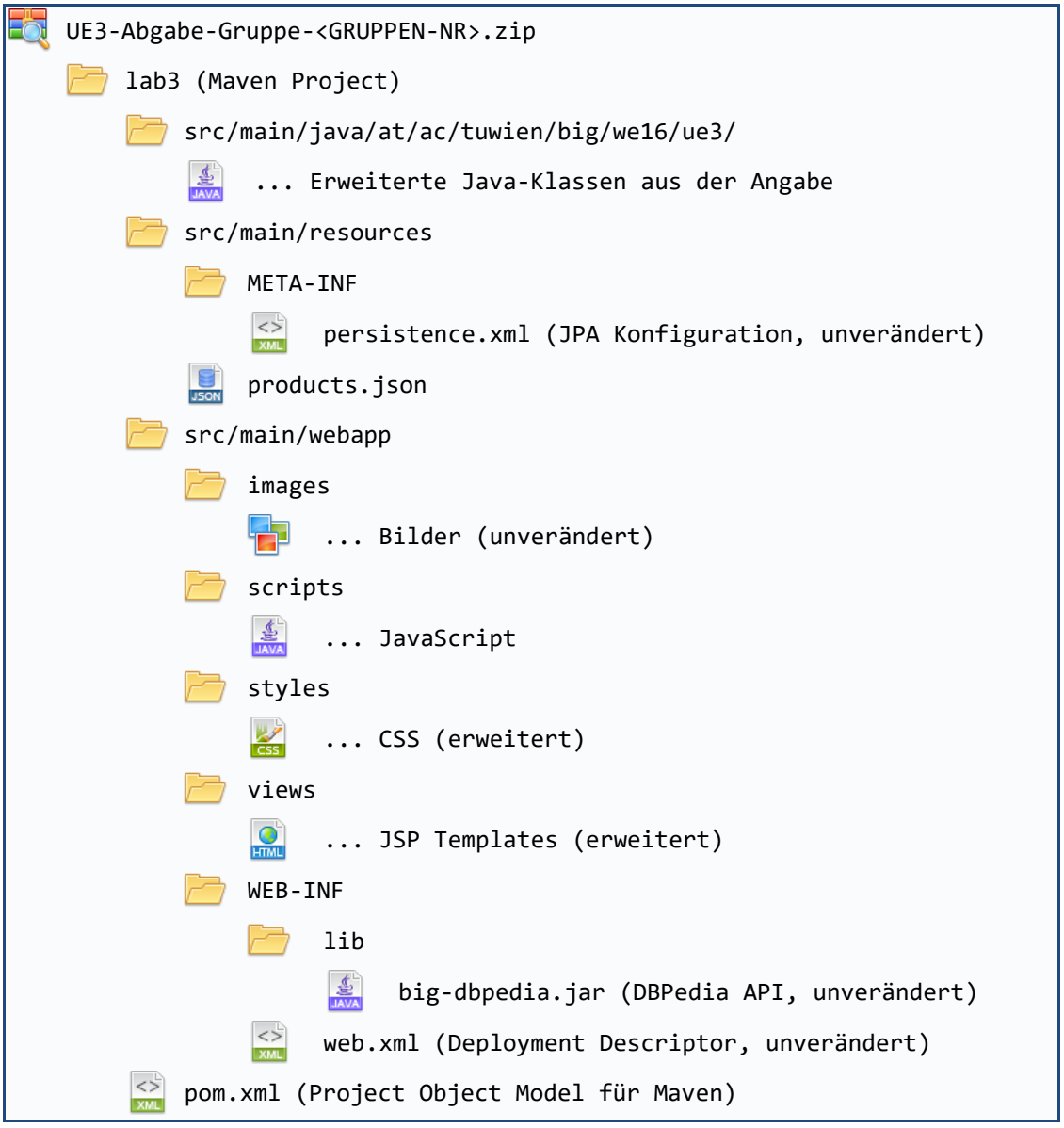
Hinweise

Validierung

Verwenden Sie zur Validierung Ihrer HTML 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 (zum Beispiel http://achecker.ca/). Nähere Infos dazu finden Sie in den Folien beziehungsweise in TUWEL.

Abgabemodalität

Beachten Sie die allgemeinen Abgabemodalitäten des TUWEL-Kurses. 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!