UE4 – AngularJS (10 Punkte)

In den vorhergehenden Übungen haben Sie mit Hilfe von Java-Technologien eine Web-Anwendung mit serverseitiger MVC-Architektur implementiert. Ziel dieses Übungsbeispiels ist es nun, aus der serverseitigen Webanwendung eine Webanwendung mit clientseitiger MVC-Architektur mit Hilfe von AngularJS zu implementieren.

Deadline der Abgabe via TUWEL[1]: **Sonntag, 12.Juni 2016 23:55 Uhr**

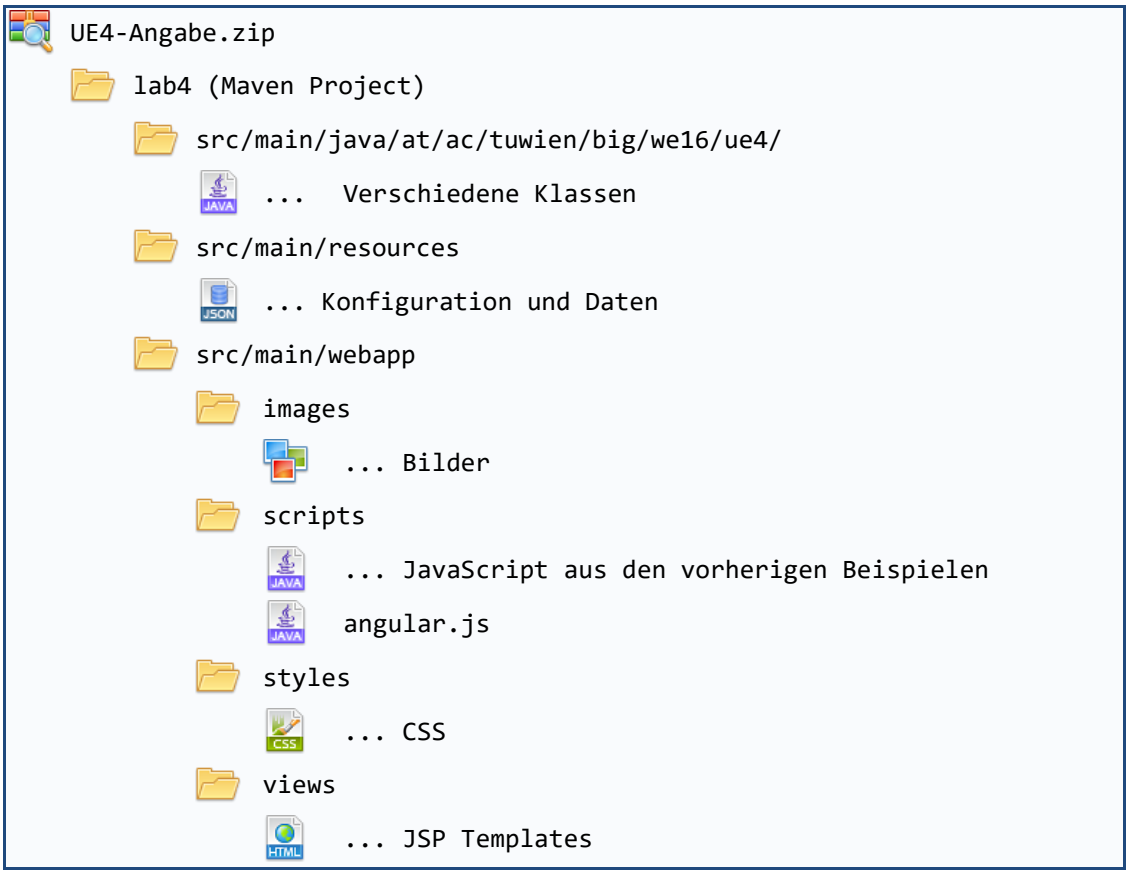
Nur ein Gruppenmitglied muss die Lösung auf TUWEL abgeben.

BIG Bid

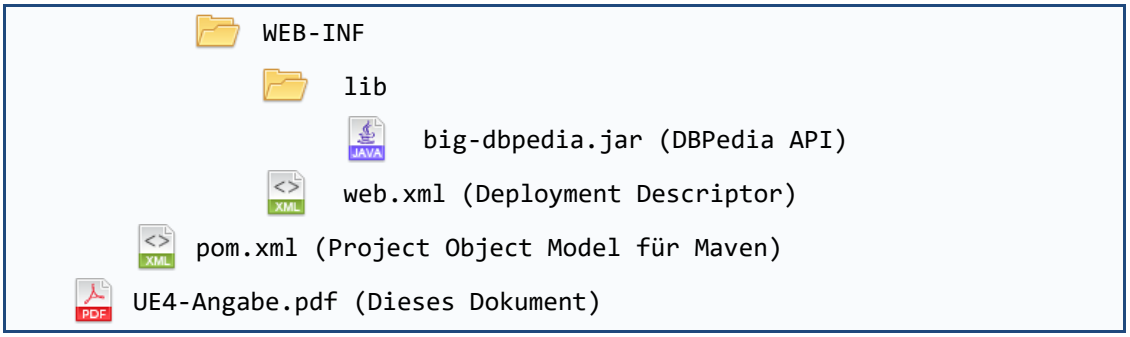
BIG Bid ist eine Online-Plattform, auf der registrierte Benutzerinnen und Benutzer Bücher, CDs und DVDs ersteigern können. Um sicherzustellen, dass die Benutzerinnen und Benutzer nur Gebote abgeben, die sie sich auch tatsächlich leisten können, müssen sie vor der Teilnahme an Auktionen Guthaben bei BIG 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 den vorhergehenden Angaben muss eingehalten werden, d.h. Nutzer(innen) müssen sich registrieren und einloggen können, Produkte in der Produktübersicht und –detailansicht ansehen können, Gebote abgeben können und abgelaufene Auktionen und neue Gebote müssen per WebSockets aktualisiert werden. Die Integration von WebServices und Linked Open Data ist nicht notwendig.

Servlets sollen auf Anfragen nur (JSON-)Daten, aber keinen vorgerenderten DOM-Tree zurückliefern. Die Darstellung ist in dieser Übung ausschließlich Aufgabe des Clients!

Das Ziel dieser Übung ist, eine Single-Page-Webanwendung zu erstellen, d.h. die Nutzer(innen) sollen mit nur einer einzigen .html-Seite interagieren. Diese Seite kann und soll aber selbstverständlich andere Seiten dynamisch, z.B. mit `ng-if` und `ng-include`, einbinden.

Teil 1 – User-Handling

Der aktuell eingeloggte Benutzer wird, wie bisher auch, in der HttpSession gespeichert, aber auch zusätzlich am Client. Um die Userdaten über mehrere Tabs hinweg synchron zu halten, soll das ngStorage-Framework (scripts/ngStorage.js) verwendet werden. Dieses Framework muss als Modul-Abhängigkeit hinzugefügt werden und stellt ein $localStorage-Service bereit, in dem Variablen gespeichert werden können

```
var App = angular.module("App", ['ngStorage']);
App.controller('General', function($scope, $http, $locale, $localStorage) {
            $scope.storage = $localStorage;

            $scope.storage.user = ...

      });
```

Teil 2 – Produkte

Die Produktinformation wird von der Hauptanwendung geladen. Wird die Produktübersichtsseite geladen, werden alle vorhandene Produkte als JSON geladen.

Produktinformation kann nur abgefragt werden, wenn die Nutzerin bzw. der Nutzer eingeloggt ist. Bei jedem Versuch, auf solche Daten zuzugreifen, wird zusätzlich zu den Daten der/die aktuell eingeloggte Nutzer(in) zurückgegeben. Wird eine Produktseite aufgerufen, es gibt aber kein gültiges Login, erfolgt eine Weiterleitung auf die Login-Seite.

Gebote sollen weiterhin per AJAX abgegeben werden können. Die Aktualisierung der Information über Sockets muss serverseitig nicht geändert werden. Clientseitig soll bei neuen Daten aber nicht der DOM-Tree direkt geändert werden, sondern das Datenmodell, aus dem sich dann ein geänderter DOM-Tree ergibt.

Um Sockets einzubinden, kann das Service $socket benutzt werden. Werden Standard-Javascript-Sockets verwendet, muss die Scope-Aktualisierung mit $scope.$apply() nach dem Erhalt neuer Nachrichten angestoßen werden.

Die Countdowns in der Übersicht und in der Detailansicht darf weiterhin mit JQuery realisiert werden, es kann aber auch ein passendes AngularJS-Modul[2] verwendet werden.

Teil 3 – Formularvalidierung

Zur Formularvalidierung sollen die Möglichkeiten von AngularJS genutzt werden, insbesondere sollen Fehlermeldungen ausgegeben werden, wenn die entsprechenden $error-Werte wahr sind.

Zur Überprüfung, ob ein Benutzer bzw. eine Benutzerin volljährig ist, soll eine AngularJS-Direktive geschrieben werden, die auf das entsprechende input-Feld angewendet wird.

Teil 4 – Lokalisierung

Zur Lokalisierung soll das $locale-Service verwendet werden. Das Service verfügt über eine id-Eigenschaft, über die die Clientsprache festgestellt werden kann. Es ist ausreichend, die beim ersten Aufruf der Seite verwendete Locale zu beachten. Die $locale.id in AngularJS enthält standardmäßig immer `en-US`, wenn keine weiteren Internationalisierungsfeatures geladen werden, was in dieser Übung nicht verlangt ist. Um zu testen, ob die Lokalisierung auch dann funktionieren würde, wäre Deutsch eingestellt, kann `$locale.id` durch `navigator.language` ersetzt werden. In Google Chrome liefert das immer die Browser-Sprache zurück, während es in Firefox die aktuell bevorzugte Sprache zurückliefert.

```
if ($locale.id.startsWith('de') || navigator.language.startsWith('de')) {
//Code, wenn der Browser auf Deutsch eingestellt ist
} else {
//Code, wenn der Browser nicht auf Deutsch eingestellt ist (Englischer Text)
}
```

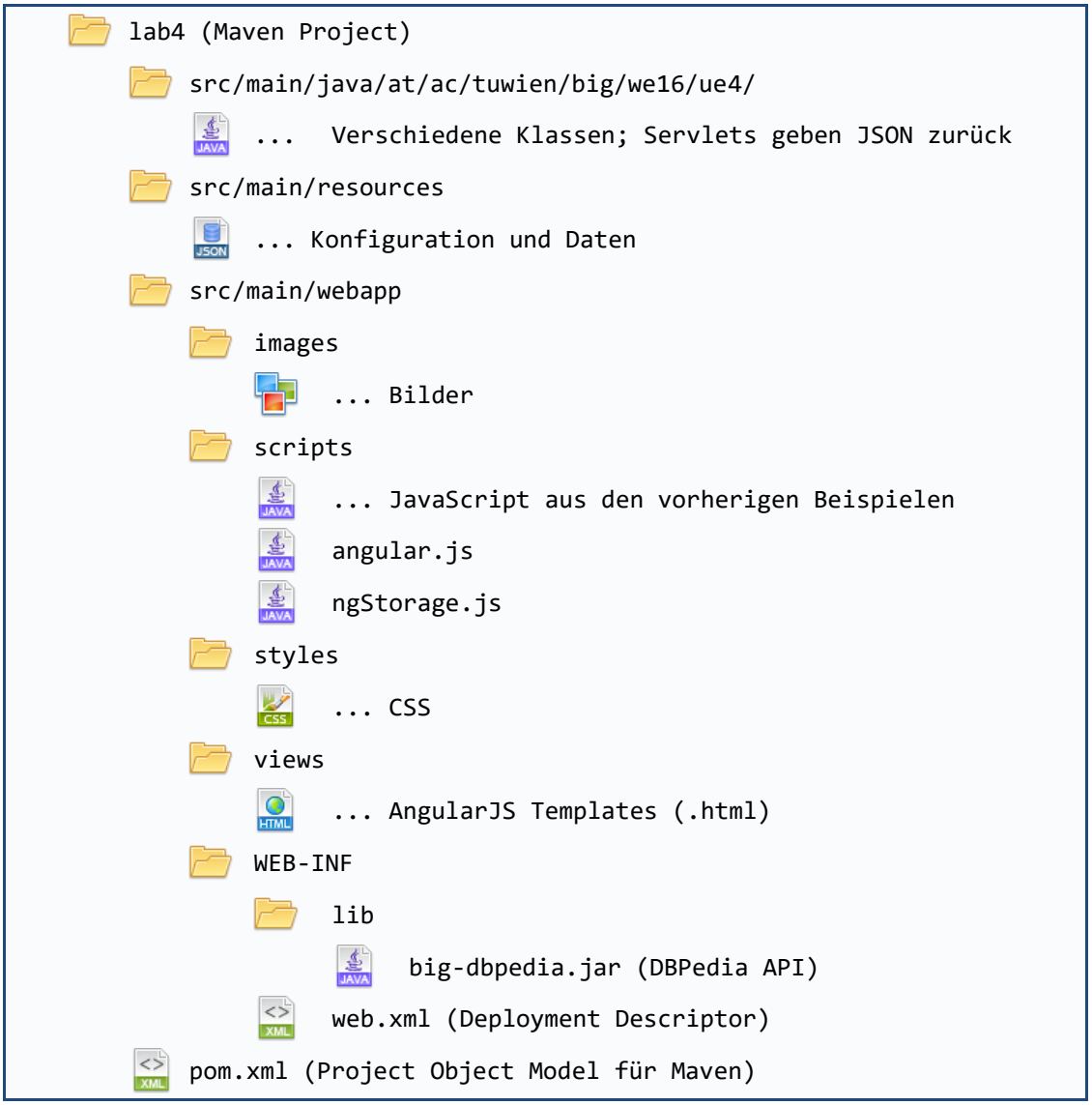
Hinweise

Bei dieser Übung wird die Validität des HTML-Quellcodes nicht überprüft, Validierungsfehler dürfen die Funktionsfähigkeit der Seite aber nicht beeinträchtigen bzw. sollten nur auf die Verwendung von AngularJS-Direktiven zurückzuführen sein.

Abgabemodalität

Beachten Sie die allgemeinen Abgabemodalitäten des TUWEL-Kurses. Zippen Sie Ihre Abgabe, sodass sie die folgende Struktur aufweist:

[2] z.B. https://siddii.github.io/angular-timer/



Alle Dateien müssen UTF-8 codiert sein!

ACHTUNG: Wird das Abgabeschema nicht eingehalten, so kann es zu Punkteabzügen kommen!