BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Übungsblatt 6

Veröffentlicht am	23.06.2016
Anzahl der Seiten	5
Anzahl Punkte im Pflichtteil (entspricht maximal erreichbaren Punkten)	**10**
Anzahl Punkte im Bonusteil	5
Abgabetermin und Demonstration in der Übung	**Do, 07.07.2016**

Anmerkungen

Bitte beachten Sie die folgenden Hinweise zur Bearbeitung der Übungsaufgaben und dem Ablauf im aktuellen Semester.

- **Lesen Sie bei einem Übungsblatt stets alle Aufgaben durch, bevor Sie beginnen.**
- Nach dem Unterricht wird vor der Übung das ggf. neue Übungsblatt in Moodle veröffentlicht.
  https://lms.beuth-hochschule.de/moodle/course/view.php?id=8190
- Sofern zum Aufgabenblatt Code-Bausteine (Vorgaben) dazugehören, werden diese ebenfalls auf Moodle zum Download angeboten und sind als Ausgangsbasis bei der Bearbeitung zu verwenden.
- Ihre Lösung der Aufgaben laden Sie ~*vor*~ Ihrer persönlichen Demonstration in Moodle hoch. Dateiname: Ü[Nr]__Nachnamen__Matrikelnummern.ZIP
  Beispiel: Ü2__Mueller_Meier__12345678__87654321.ZIP
- Erfordern die Teil-Aufgaben eines Übungsblattes, dass Sie mehrere Anwendungen, HTML-Seiten oder Code-Pakete erstellen, dann legen Sie bitte Unterordner in Ihrem ZIP mit den Nummern der Aufgaben an.
- Persönliche Demonstration und Erklärung in der Übung durch alle Gruppenmitglieder. Jedes Gruppenmitglied kann die Abgabe erläutern, sonst keine Punkte.
- Bei den Aufgaben ist jeweils angegeben, ob diese Pflicht- oder Bonus-Aufgaben sind, sowie die maximal erreichbaren Punkte der Teil-Aufgabe.
- Eine Übung gilt als bestanden, wenn mind. 50% der Pflichtpunkte erreicht wurden, sonst gibt es 0 (Null) Punkte.
- Bei verspätete Abgabe von bis zu maximal 2 Wochen können nur noch 50% der möglichen Punkte des Übungsblattes erreicht werden (bei einer Woche verspäteter Abgabe 75% der möglichen Punkte).

Referenzen:

- Foliensatz SU8 zu NoSQL, mongoDB und `mongoose`
- Foliensatz SU9 zu `backbone.js` mit Wiederholung zu `require.js`
- *Tweets in Backbone.js* - Code zum Unterricht SU9
- Require.js Doku http://requirejs.org/docs/start.html
- Backbone.js Doku http://backbonejs.org/#Getting-started
- Referenzvideos (mind. drei Stück) zum Eintragen in Ihre Datenbank stehen zum Download in Moodle als JSON-Format. Sollten die URLs irgendwann kein Video mehr liefern (bspw. in Chrome geht es nicht), dann ändern Sie die URL einfach auf einen Dummy, wie http://www.html5videoplayer.net/videos/toystory.mp4

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

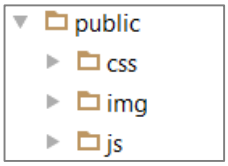
Ziel und Zweck der Übung:

Sie haben serverseitig in den letzten Wochen einen robusten Node.js Server aufgesetzt, welcher statische Dateien ausliefert und über eine REST-Schnittstelle Daten bereitstellt und speichert.
Diese Übung ist die letzte des Kurses ME2 und soll Ihren Blick nun auf client-seitiges, modulares Entwickeln lenken. **Sie werden mit backbone.js Ihre Video-REST-Schnittstelle nutzen, aktuell gespeicherte Videos seitenweise auflisten und beim Abspielen die Anzahl an Wiederholungen (`playcount`) erhöhen**. Am Ende umfassen Ihre Fähigkeiten das Erstellen einer mit require.js modularisierten Single-Page-Application (SPA) auf Backbone.js-Basis.

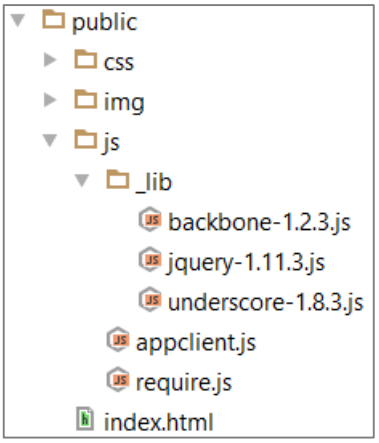
(Die Aufgaben-Anweisungen sind detailliert wie ein Tutorial, daher 5 Seiten)

Vorbereitung (keine Punkte)

1. Nehmen Sie als Code-Basis den node.js-Code für eine funktionierende REST-Schnittstelle für Videos. Entweder Ihre Lösung zu Übung 4, die Musterlösung für Übung 4 oder (besser) Ihre Lösung zu Übungsblatt 5.
2. Legen Sie folgende Ordnerstruktur im Unterordner `.\public` an (siehe Bild rechts)



3. Kopien Sie Ihre eigene oder die Musterlösung zu Übungsblatt 1 in den Ordner .\public und seine Unterordner.
4. Sofern nicht bereits geschehen, aktualisieren Sie Ihren app.js Code, damit mittels express.static Middleware die Dateien aus .\public ausgeliefert werden und starten Sie die Server-Anwendung mit npm start bzw. node app.js[1]. Vergessen Sie nicht zuvor Ihre MongoDB zu starten, sofern Sie in Schritt 1. die REST-Schnittstelle für /videos mit mongoDB gewählt haben.
5. Sie sollten nun Ihre Video-Seite bzw. die Musterlösung zu Übungsblatt 1 unter `http://localhost:3000/index.html` im Browser abrufen können.
6. Kopieren Sie die Dateien `.\public\js\require.js`, `.\public\js\appclient.js` sowie das Verzeichnis `.\public\js\_lib` aus dem Code zum SU9 (backbone.js) in die entsprechenden Ordner Ihres `.\public`-Verzeichnisses. Das Ergebnis sollte circa so aussehen wie auf dem Bild rechts.



Beachten Sie:

- Im Folgenden werden die client-seitigen Dateien im .\public-Verzeichnis geändert. Ein Neustart des node-Servers ist daher i.d.R. nicht erforderlich, wenn Sie Dateien geändert haben.
- Die Aufgaben bauen auf der REST-Schnittstelle und den Attributen von Videos auf, wie Sie in Übung 4 bzw. 5 verwendet wurden.

[1] Sofern Sie in Schritt 1 die Musterlösung zu Blatt 5 verwendet haben, passen Sie noch den Pfad zur faviconbeuth.ico-Datei in der app.js an (oder kommentieren die Middleware aus)

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Aufgabe 1 (Pflicht, 5 Punkte insgesamt)

Sie werden einen ersten Backbone.js Router und View erstellen, um Videos aufzulisten.

Aufgabe 1.a (Pflicht, 1 Punkt)

- Ändern Sie Ihre `index.html` Datei so ab, dass nur die `require.js` Datei als einzige JavaScript-Datei geladen wird. Geben Sie im Script-Tag als Einstiegspunkt für `require.js` die `js/appclient` an. Damit wird `require.js` diese Javascript-Datei automatisch als erstes laden und ausführen.
- Löschen Sie in der `appclient.js` die fünf require-Modulimporte für `models/..` und `views/..` und alle Code-Zeilen innerhalb der Funktion `main` (Zeilen ca. 34-57), denn das gehört alles zum alten tweet-Beispiel des Unterrichts. Schreiben Sie stattdessen ein jQuery wie `$('body').prepend('<h1>Video App</h1>');` in die main-Methode hinein. Wenn Sie diese h1 auf Ihrer SPA-Seite im Browser sehen, funktioniert das Laden der `appclient.js`. Sie haben Ihren ersten funktionstüchtigen Backbone.js Router fertiggestellt.

Aufgabe 1.b (Pflicht, 2 Punkte)

Erstellen Sie einen Ordner `.\models` in `.\public\js` für Ihre Client-seitigen Model-Repräsentationen. Darin legen Sie eine `video.js`-Datei an, in welcher Sie ein neues Modul für VideoModel definieren werden.

- Fügen Sie einen `define(…)` Aufruf nach require.js Konvention ein und importieren Sie `backbone`.
- Definieren Sie ein Schema für Videos, welches zu Ihrer REST-Schnittstelle passt und die korrekte `urlRoot` als Attribut hat. Orientieren Sie sich am Code aus SU9 für `tweets`. Erstellen Sie damit ein `VideoModel` als `Backbone.Model`.
- Definieren Sie nun auch eine `VideoCollection` als `Backbone.Collection`, welche als Model das `VideoModel` nutzt. Vergessen Sie nicht, das Attribut `url` Ihrer Collection zu setzen.
- Geben Sie Ihr `VideoModel` und Ihre `VideoCollection` als Modulexport zurück. (Achtung: bei AMD-Style geht dies nicht über `module.exports` sondern über einen passenden `return`-Wert.

Aufgabe 1.c (Pflicht, 2 Punkte)

Passen Sie ihren Code des `Backbone.Router` in der `appclient.js` so an, dass die `main`-Methode eine neue `VideoCollection` anlegt. Damit das funktioniert, müssen Sie Ihr neues Modul aus `models/video` erst beim require-Import hinzufügen.
Führen Sie nach dem Anlegen der neuen `VideoCollection` ein Abrufen vom Server mittels `fetch()` durch. Um das Ergebnis „mitzubekommen“ geben Sie `fetch()` als Parameter ein Objekt mit, welches unter dem Attribut `success:` eine Funktion enthält, die im Erfolgsfall ausgeführt wird. Unter `error:` im Fehlerfall.
Codebeispiel siehe hier:

```
myVideoCollection.fetch({
    success: function() {
    },
    error: function() {
    }
});
```

Geben Sie im Erfolgsfall zunächst beispielsweise mittels `alert`-Funktion oder `console.log` die Anzahl der empfangen Videos aus (die Größe Ihrer VideoCollection).
Achtung: Stellen Sie sicher, dass ihre Server-seitige Datenbank auch Video-Einträge enthält (und nicht leer ist).

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Aufgabe 2 (Pflicht, 5 Punkte insgesamt)

Ihr Backbone-Router und das `VideoModel`, sowie die `VideoCollection` funktionieren nun. In dieser Aufgabe legen Sie passende `Backbone.View`-Komponenten an, um die Liste der Videos dynamisch anzuzeigen.

Aufgabe 2.a (Pflicht, 3 Punkte)

Legen Sie ein neues Verzeichnis `.\public\js\views` an und darin die Datei `video.js` . Definieren Sie in dieser Datei ein Modul und importieren Sie beim `define(..)` Aufruf `backbone`, `jquery` und `underscore` [2].

- Definieren Sie als Modulrückgabe einen `VideoView` auf Basis von `Backbone.View`. Orientieren Sie sich am sog. Creator-View aus dem Unterricht und vergeben Sie einen `tagName`, den Ihr View als sein Root-Element erstellt.
- In Ihrer `index.html` existieren aus Übung 1 bereits HTML-Bausteine für ein Video mit seinen Buttons. Schließen Sie diesen HTML-Baustein in die folgenden Tags ein (und löschen Sie die anderen Video-HTML-Abschnitte in der index.html):

```
<script type="text/template" id="video-template">
    ...
</script>
```

- Nutzen Sie den Inhalt dieses Tags mit der `id video-template` in Ihrem VideoView als Basis für Ihr Template. (siehe Unterrichtsfolien für Beispiel). Damit auch Teile des Videos später im Template eingesetzt werden, editieren Sie ihren Template-Baustein der `index.html` und nutzen die Variablen `title`, `src`, `length`, `description`, `playcount` und `ranking` entsprechend.
- Implementieren Sie die `render`-Methode Ihres VideoViews, welche das Template aufruft und den gerenderten Ergebnis-String am eigenen Rot-Element (`el`) einfügt.

Sie haben nun einen `VideoView` auf Backbone-Basis erstellt, der automatisch passendes HTML rendert, wenn sich das dazu gehörende `VideoModel` ändert. Damit die neuen Fähigkeiten sichtbar werden, ergänzen Sie Ihre `appclient.js` um den Import des neuen Moduls aus `views/video`. Dann nutzen Sie nach erfolgreichem Abrufen der `VideoCollection` den neuen `VideoView`, übergeben im Konstruktor des `VideoView` als `model` einfach *nur das erste* der via REST-Schnittstelle geladenen Videos und rufen die render-Methode Ihres neuen `VideoViews` auf. Anschließend können Sie das Ergebnis direkt auf der Konsole oder im DOM ausgeben, bspw. mittels

```
$('body').append(myVideoView.render().el);
```

Aufgabe 2.b (Pflicht, 2 Punkte)

- Legen Sie nun noch eine `.\public\js\views\video-list.js` Datei an und definieren Sie darin einen `VideoListView` als `Backbone.View` . Dieser View ist ein sog. Delegator-View, welcher selbst kein Template benutzt. Der View wird eine `collection` haben und in seiner `render`-Methode für jedes Element der Collection einen `VideoView` anlegen, rendern und dem eigenen (`VideoListView`)-DOM-Element hinzufügen. Damit diese Änderungen direkt im DOM sichtbar sind, verwenden Sie als Ankerpunkt (`el`) des `VideoListView` ein existierendes DOM-Element. (siehe Unterrichtsfolien zum Delegator View).
- Legen Sie die `initialize`-Methode des `VideoListView` an und registrieren Sie darin die eigene `render`-Methode für das Event „`add`“ auf der `collection` des Views.

[2] Oder eine andere JS Template Engine, wenn Sie diese statt underscore verwenden wollen

- Jetzt können Sie in Ihrer `appclient.js` alle Parameter des `.fetch()` Ihrer `VideoCollection` löschen. Es reicht nun, wenn Sie ~vor~ dem `.fetch()` einen neuen `VideoListView` anlegen und der Konstruktorfunktion als `collection` Ihre Instanz von `VideoCollection` mitgeben. *(Der folgende `.fetch()` löst das `add`-Event der Collection aus, woraufhin der `VideoListView` sich selbst neu rendert. Ihre gesamte `main`-Funktion des Routers besteht also nur noch aus drei Zeilen Code.)*

Aufgabe 3 (Bonus, 1.5 Punkte insgesamt)

Ihre eigenen Play/Pause/Stopp-Buttons funktionieren derzeit nicht. Da Sie bereits das Template für `VideoView` in der `index.html` haben, brauchen Sie nur noch DOM-Events in Ihrem `VideoView` zu registrieren und entsprechende Aktionen auszulösen. Registrieren Sie also `events` in Ihrem `VideoView` für die einzelnen Buttons innerhalb ihres Templates (Achtung: keine IDs verwenden, da das Template mehrfach gerendert wird; pro Video einmal). Beispielcode zu Backbone.js und DOM-Events ist auch in den Unterrichtsfolien.

Aufgabe 4 (Bonus, 1.5 Punkte insgesamt)

Erhöhen Sie beim Abspielen des Videos den `playcount` um Eins (+1) und führen Sie anschließend ein `.save()` der dazugehörigen `VideoModel`-Instanz durch. Damit ist die Änderung auf dem Server gespeichert[3]. Wie aktualisieren Sie nun die Anzeige Ihres Video-Playcounters im DOM automatisch?

Aufgabe 5 (Bonus, 2 Punkte insgesamt)

Fügen Sie eine Blättern-Funktion hinzu. Die `Backbone.Collection` erlaubt es, direkt beim `.fetch()` die `limit` und `offset`-Angaben als Parameter mitzugeben. Ein vorgeschlagenes Vorgehen könnte sein: Einen sog. Enhancer View als `Backbone.View` anzulegen, welcher auf die Klicks im DOM für Vor/Zurück-Blättern reagiert, die Collection zurücksetzt und neue Elemente mit `.fetch(…)` vom Server läd. Die DOM-Anzeige der Videos aktualisiert sich dann (siehe Aufgaben 1+2) automatisch.

PS: Es ist nun auch leicht, mittels eines Backbone.View auf Eingaben in einem Formular zu reagieren und neue Videos in die VideoCollection einzufügen (und auf dem Server zu speichern). Diese Aufgabe ist jedoch Ihrem Selbststudium überlassen…
oder vielleicht möchten Sie auch eine ganz andere Anwendung kreieren?

Viel Erfolg mit Backbone.

[3] Dieses Vorgehen ist nicht robust. Wenn in der Zwischenzeit andere Personen das Video abgespielt haben, gehen diese Playcounts verloren. Daher ist eine nicht idempotente PATCH-Implementierung für `playcount` besser geeignet…oder zumindest ein Vergleich der Versionsnummer oder updatedAt-Zeitstempel beim Speichern (server-seitig).