

Internetanwendungen für mobile Geräte – Übungsprogramm

Virtuelle Fachhochschule und Beuth Hochschule für Technik

Sommersemester 2019, Stand: 12. Februar 2019

Jörn Kreutel

Inhaltsverzeichnis

SETUP	5
HTM+CSS	9
JSL	15
JSR	18
MWF	22
NJM+LDS	30
FRM	35
MFM	39
MME	42
OFF	44
NAV	47

Wichtige Hinweise zur Bearbeitung der Übungsaufgaben

Die Abgabe der Aufgaben erfolgt in den folgenden 2-3 Blöcken zu Terminen, die konkret im Laufe des Semesters unter Berücksichtigung des Lernfortschritts festgelegt werden:

- Abgabe 1: Aufgaben zu HTM+CSS, JSL, JSR
- Abgabe 2a: Aufgaben zu MWF, NJM+LDS
- Abgabe 2b: Aufgaben zu FRM, MFM, MME, OFF

Die Aufgaben von Abgabe 2a und Abgabe 2b können ggf. in einer Abgabe zusammengefasst werden. Die Funktionalität der Aufgaben pro Abgabe baut jeweils in der genannten Reihenfolge aufeinander auf. Pro Abgabe sind daher alle Aufgaben jeweils in *einer* integrierten Anwendung zusammenzufassen, die alle Anforderungen aller Aufgaben der Abgabe erfüllt. Beispielsweise muss die Umsetzung der Aufgaben zu JSR die Anforderungen zu HTM+CSS und JSL erfüllen. Dafür müssen jeweils gemeinsame HTML, CSS oder JavaScript Ressourcen genutzt werden. Selbiges gilt für die Aufgabenblöcke 2a und 2b insgesamt, d.h. die für Abgabe 2a umgesetzte Funktionalität muss in die Umsetzung der Aufgaben für Abgabe 2b unter Verwendung gemeinsamer Ressourcen integriert sein. Der finale Navigationsverlauf der Anwendung, der gemäß den beschriebenen Anforderungen mit Abgabe 2b erreicht werden soll, ist auf der letzten Seite dieses Dokuments im Gliederungspunkt NAV dargestellt.

Bitte beachten Sie, dass die Übungen für Abgabe 1 ausschließlich unter Verwendung der durch aktuelle Browser wie Firefox oder Chrome unterstützten Ausdrucksmittel von HTML, CSS und JavaScript umzusetzen sind. Die Verwendung externer Bibliotheken wie jQuery ist nicht zulässig. Dieser Aufgabenblock dient dazu, anhand des Beispiels einer Listenansicht eine Vorstellung davon zu bekommen, wie typische gestalterische und

funktionale Aspekte mobiler Webapplikationen mit den elementaren Ausdrucksmitteln von Webtechnologien umgesetzt werden können.

Für die Umsetzung der darauf folgenden Aufgaben steht Ihnen ein durch den Modulautor entwickeltes *Mobile Web Framework* (MWF) zur Verfügung. Die Einführung in dessen Ausdrucksmittel erfolgt in Form eines Tutorials, in welchem detailliert und schrittweise die Umsetzung einer Anwendung beschrieben wird, die den in den Aufgaben MWF1-MWF4 formulierten Anforderungen vollständig genügt. Diese Anwendung dient dann als Grundlage für die Umsetzung der weiteren Aufgaben.

Während für die Aufgaben **für Abgabe 1 die Verwendung von Frameworks nicht zulässig** ist, steht es Ihnen für die Aufgaben ab MWF frei, diese in einem Framework Ihrer Wahl umzusetzen. Dies muss nicht notwendigerweise MWF sein. Beachten Sie aber, dass die Punktvergabe für die Aufgaben MWF1-MWF4 die Tatsache berücksichtigt, dass die Umsetzung der Anforderungen allein durch Nachvollziehen des Tutorials zu MWF möglich ist, d.h. es ist nicht auszuschließen, dass Ihnen bei Verwendung eines anderen Frameworks höhere Aufwände entstehen. Bezüglich des Frameworks MWF sollte andererseits erwähnt werden, dass dieses in einer aus Sicht des Autors für die Zwecke einer Lehrveranstaltung nutzbaren, aber bei weitem noch nicht optimalen Version vorliegt. Dies betrifft sowohl die implementatorische Umsetzung, als auch den für die Anwendungsentwicklung unterstützten Funktionsumfang des Frameworks.

Wichtige Hinweise zum Stand von Lehrmaterial und Übungsprogramm

Das vorliegende Übungsprogramm stellt eine grundlegende Überarbeitung der ursprünglich für die Lehrveranstaltung konzipierten Übungen dar, die im Lehrmaterial des Moodle Kurses wie auch in PDF Form vorliegen. Während dort als durchgängiger Anwendungsfall auf eine mobile Anwendung mit einem vergleichsweise individuellen Grafik- und Interaktionsdesign Bezug genommen wird, orientiert sich die vorliegende Fassung der Übungen stärker an mobilen Applikationen mit standardisierten Bedienelementen wie Listenansichten, Detailansichten, Formularen, Aktionsmenüs und Popup-Dialogen, deren Umsetzung durch das o.g. Framework MWF z.T. in hohem Maße unterstützt wird. Die im Moodle-Kurs vorliegenden Lerneinheiten beziehen sich derzeit jedoch noch auf den älteren Anwendungsfall und erläutern die in den einzelnen Lerneinheiten eingeführten Ausdrucksmittel auf Basis von dessen Gestaltungs- und Funktionsmerkmalen. Aus diesem Grund stehen Ihnen auch nach wie vor die zur Illustration bereit gestellten Implementierungsbeispiele für diesen Anwendungsfall zur Verfügung.

Die in den praktischen Kapiteln der Lerneinheiten verwendeten Implementierungsbeispiele sollten Sie jedoch dazu befähigen, über den dort betrachteten Anwendungsfall hinaus auch die nachfolgenden, aktualisierten, Übungsaufgaben umsetzen. Dies gilt insbesondere für die folgenden Aspekte des Übungsprogramms:

- Dokumentenstruktur und Gestaltung (Lerneinheit HTM)
- Umsetzung von Ansichtselementen eines vorgegebenen Designs mit CSS Regeln auf Basis des CSS Box Models (LE CSS)
- Eingabeverarbeitung mit DOM Events und DOM Manipulation (LE JSL)
- Dynamischer Aufbau von Ansichten (LE JSL)
- Abstimmung der Funktionen von HTML, CSS und JavaScript zur Interaktionssteuerung, z.B. zur Umsetzung von animierten Ansichtsübergängen (LE JSL)
- Umsetzung einfacher und komplexer Anforderungen an Formulare (LE FRM und LE MFM)
- Verwendung von Multimedia-Elementen (LE MME)
- Herstellung der Offlinefähigkeit von Webanwendungen (LE OFF)

Nicht erwähnt sind hier die beiden Lerneinheiten NJM und LDS, die sich mit der Umsetzung lesender und schreibender Zugriffe auf die von einer Anwendung genutzten und wahlweise server-seitig oder lokal vorliegenden Datenbestände beschäftigen, d.h. einem wesentlichen Funktionsmerkmal von mobilen Anwendungen

und von Softwaresystemen überhaupt. Diese Funktionen zeichnen sich zumindest bei Anwendungen mit vergleichsweise einfachem Datenmodell wie der hier betrachteten jedoch zugleich durch eine sehr hohe Verallgemeinerungsfähigkeit aus, die durch das MWF Framework weitestgehend ausgeschöpft wird. So unterscheidet sich denn das vorliegende Übungsprogramm abgesehen vom Anwendungsfall nicht zuletzt dadurch deutlich von seinem Vorgänger, dass Sie lesende und schreibende Zugriffe auf lokale und server-seitige Datenbestände nicht mehr selbst durch Verwendung einer jeweils spezifischen Programmierschnittstelle implementieren müssen. Vielmehr stellt Ihnen MWF eine verallgemeinerte Programmierstelle für Datenzugriffe zur Verfügung, die von der lokalen vs. der durch einen Server-Zugriff vermittelten Umsetzung der Zugriffe abstrahiert. Den Lerninheiten NJM und LDS kommt damit in Bezug auf das aktualisierte Übungsprogramm stärker die Rolle der Vermittlung von Hintergrundwissen zu. Für ein tiefer gehendes Verständnis des vollständigen Funktionsumfangs einer Anwendung, zu dem nicht nur die von Ihnen entwickelten Codebestandteile gehören, sondern auch die durch ein Framework bereit gestellte Funktionalität, ist dieses Wissen jedoch unabdinglich.

Für die Bearbeitung der Aufgaben ist es empfehlenswert, dass Sie zunächst die gesamte Aufgabenbeschreibung, inklusive der Bearbeitungshinweise, durchlesen und erst dann mit der Umsetzung beginnen.

SETUP

0. Systemvoraussetzungen

- Stellen Sie sicher, dass die Versionsverwaltungssoftware *Git* auf Ihrem Entwicklungsrechner installiert ist. Installieren Sie andernfalls eine für Ihren Rechner geeignete Version: <https://git-scm.com/downloads>
- Als Browser wird die Verwendung von Chrome (<https://www.google.com/chrome/browser/> oder Firefox <https://www.mozilla.org/de/firefox/new/> empfohlen.
 - Stellen Sie bei Verwendung von Firefox sicher, dass die Option ‘*Chronik anlegen*’ in den Datenschutzeinstellungen von Firefox aktiv ist.

1. WebStorm

- Laden Sie WebStorm in einer für Ihren Entwicklungsrechner geeigneten Version herunter: <https://www.jetbrains.com/webstorm/download/>
- Installieren Sie die Anwendung.
- Nutzen Sie entweder zunächst die kostenlose 30 tägige Evaluierungslizenz oder registrieren Sie sich *unter Ihrer Hochschuladresse* bei JetBrains unter <https://www.jetbrains.com/student/> – für Adressen unter @beuth-hochschule.de und @th-brandenburg.de bekommen Sie freien Zugriff auf WebStorm und andere JetBrains Produkte.

2. Implementierungsbeispiele auf Github

- Auf Github stehen Ihnen die folgenden drei Projekte als Implementierungsbeispiele bzw. Codegerüst für die Umsetzung der Übungsaufgaben zur Verfügung:
 - https://github.com/dieschnittstelle/org.dieschnittstelle.iam.css_jsl_jsr.git (Implementierungsbeispiele und Codegerüst für Aufgaben zu CSS, JSL, JSR)
 - <https://github.com/dieschnittstelle/org.dieschnittstelle.iam.mwf.skeleton.git> (Codegerüst für Aufgaben ab MWF)
 - <https://github.com/dieschnittstelle/org.dieschnittstelle.iam.mwf.sample.git> (Implementierungsbeispiele für Aufgaben ab MWF)
- Um diese Projekte als Projekte in Webstrom zu importieren, wählen Sie nach dem Starten von WebStorm die Option *Checkout from Version Control → Git* und geben dann als *Git Repository URL* jeweils die o.g. URL des Projekts ein.
- Wählen Sie in *Preferences → Languages & Frameworks → JavaScript* als *JavaScript Language Version* ‘*ECMAScript 6*’ aus.

3. NodeJS

Die Installation von NodeJS ist für die Bearbeitung der Übungsaufgaben ab den Übungen zu MWF erforderlich. Die in den folgenden Anweisungen bezüglich WebStorm genannten Punkte beziehen sich auf die Neuerstellung einer *Run*-Konfiguration für NodeJS. Die o.g. Projekte sind jedoch bereits jeweils mit einer Konfiguration ausgestattet, die jeweils unter dem Namen *Run Node* ausgeführt werden

kann. Nach Installation von NodeJS sind u.U. keine weiteren Anpassungen mehr notwendig. Sollte die Konfiguration dann nicht ausführbar sein, muss die unten beschrieben Zuweisung des *node Executable* aktualisiert werden.

- Laden Sie NodeJS in einer für Ihren Entwicklungsrechner geeigneten Version herunter: <https://nodejs.org/en/download/>
- Installieren Sie NodeJS.
- Wählen Sie aus dem *Run* Menü von WebStorm die Option *Edit Configurations...*.
- Wählen Sie nach Ausführung von + (*Hinzufügen*) die Option *Node.js*.
- Weisen Sie als *Node Interpreter* das `node` Executable aus dem `bin` Verzeichnis der NodeJS Installation zu, falls dieses Feld nicht schon vor-ausgefüllt ist.
- Weisen Sie der Konfiguration einen Namen Ihrer Wahl zu.
- Wählen Sie als *Working Directory* Ihr Projektverzeichnis.
- Wählen Sie als *JavaScript file* die Datei `webserver.js` im Projektverzeichnis.
- Beenden Sie die Einrichtung mit *Apply* und schließen Sie das Fenster mit *Close*
- Weitere Hinweise finden Sie hier: <https://www.jetbrains.com/webstorm/help/node-js.html>

4. MongoDB

Die Installation von MongoDB ist für die Bearbeitung der Übungsaufgaben ab NJM+LDS erforderlich.

- Laden Sie eine für Ihren Entwicklungsrechner geeignete Version von MongoDB von der MongoDB Website herunter: <https://www.mongodb.com/download-center#community>
- Installieren Sie MongoDB bzw. entpacken Sie die herunter geladene Archivdatei.
- Richten Sie MongoDB als *External Tool* in WebStorm ein.
 - Öffnen Sie die *Preferences* bzw. *Settings* Ansicht in WebStorm.
 - Wählen Sie unter den Optionen von *Tools* die Option *External Tools*.
 - Führen Sie die Aktion ‘Hinzufügen’ (+) aus.
 - Geben Sie einen Namen Ihrer Wahl ein und wählen Sie als *Program* das `mongod` Executable aus dem `bin` Verzeichnis der MongoDB Installation aus.
 - Erstellen Sie, falls noch nicht vorhanden, in Ihrem Projektverzeichnis ein Verzeichnis `mdb/mdbdata/db`.
 - Geben Sie unter *Parameters* folgendes an: `-dbpath ./mdb/mdbdata/db`
 - Weisen Sie als *Working Directory* Ihr Projektverzeichnis zu.
 - Falls Sie ein anderes Verzeichnis für die Datenbank als das hier angegebene verwenden möchten, dann müssen Sie *Parameters* und *Working Directory* so anpassen, dass der angegebene `-dbpath` von letzterem Verzeichnis aus auflösbar ist.

- Schließen Sie die Einrichtung mit *OK* ab.
- Siehe zur Einrichtung von *External Tools* in Webstorm auch: <https://www.jetbrains.com/webstorm/help/configuring-third-party-tools.html>
- Nach abgeschlossener Einrichtung kann die MongoDB Datenbank durch Öffnen des *Tools* Menüs, Auswahl der Option *External Tools...* und Auswahl des oben vergebenen Namens gestartet werden.

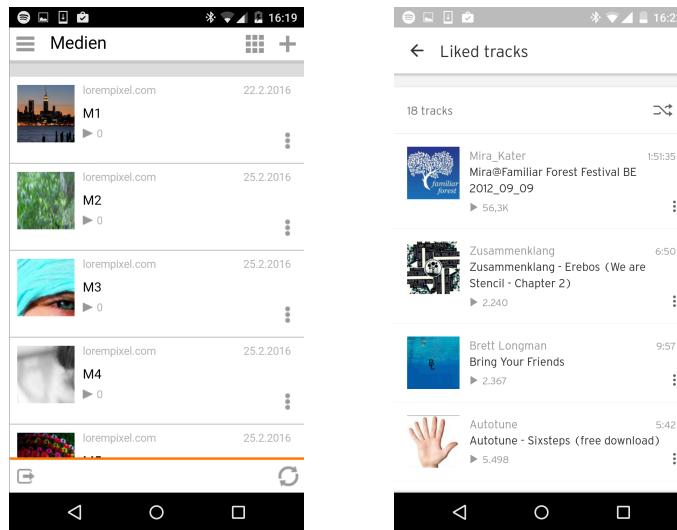
HTM+CSS

Ü CSS1 Umsetzung einer Listenansicht

(28 Punkte)

Aufgabe

Konzipieren Sie eine HTML Dokumentenstruktur für die nachfolgend links dargestellte Listenansicht und erstellen Sie CSS Regeln zur Umsetzung der visuellen Oberfläche. Die Ansicht ist inspiriert durch die rechts daneben dargestellte Listenansicht aus der *Soundcloud* App für Android. Es ist ausreichend, wenn Sie die Ansicht dem Augenschein nach entsprechend der Vorlage links umsetzen, d.h. wenn Farben und Proportionen der Elemente offensichtlich der Vorlage sehr nahe kommen. Die Vorlage selbst wurde auf diese Weise nach dem Vor-Bild der *Soundcloud*-Listenansicht entwickelt.



Anforderungen

1. Gesamtaufbau

- (a) Kopfleiste, Fußleiste und Hauptbereich der Ansicht nehmen immer exakt 100% der verfügbaren Höhe ein, die gesamte Ansicht ist weder in der Höhe abgeschnitten, noch vertikal scrollbar. (3 P.) ✓
- (b) Die Ansicht nimmt insgesamt 100% der verfügbaren Breite ein und ist nicht horizontal scrollbar. (1 P.) ✓
- (c) Der Hauptbereich der Ansicht, in der die Listenelemente dargestellt werden, ist vertikal scrollbar, falls erforderlich. (1 P.) ✓
- (d) Die Default-Hintergrundfarbe der Ansicht ist durchgängig weiß. (1 P.) ✓
- (e) Die Default-Textfarbe ist schwarz. (1 P.) ✓

2. Kopfzeile



- (a) ‘Sandwich’-Icon und Überschrift sind linksbündig. (2 P.) ✓
- (b) ‘Kachel’- und ‘+’-Icon sind rechtsbündig. (1 P.) ✓

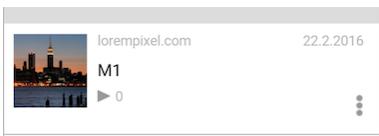
- (c) Der untere Rand der Kopfzeile ist mit einer Schattierung versehen. (1 P.) ✓
 (d) Alle Icons sind vertikal zentriert. (1 P.) Stimmt nur fast ✓ ✗

3. Fußzeile



- (a) Die Fußzeile ist nach oben durch eine orangenen Rand abgegrenzt. (1 P.) ✓
 (b) Das 'Logout'-Icon und das 'Refresh'-Icon sind links bzw. rechtsbündig dargestellt. (1 P.) ✓
 (c) Die Icons sind vertikal zentriert. (1 P.) ✓

4. Liste und Listenelemente



- (a) Von der Kopfzeile ist die Listenansicht im nicht gescrollten Zustand durch einen grauen Puffer abgesetzt. Dieser wird beim Scrollen der Ansicht aus dem sichtbaren Bereich verschoben. (2 P.) ✓
 (b) Die Listenelemente sind durch eine graue Randlinie voneinander abgegrenzt. (1 P.) ✓
 (c) Das oberste Element hat keine obere Randlinie, das unterste Element hat eine Linie. (1 P.) ✗
 (d) Das Bildelement ist mit Abstand an der linken oberen Ecke des Listenelements ausgerichtet und linksbündig mit dem Sandwich-Icon der Kopfzeile. (1 P.) ✓
 (e) Die Bildanzeige ist stets quadratisch, verzerrungsfrei und hat eine fixe Größe, unabhängig von der Bildvorlage. (2 P.) ✓
 (f) Die Anzeige des Bildursprungs (*lorempixel.com*), der Titel des Bildes (*M1*) und die Anzeige der verfügbaren Tags (*0*) sind durch einen Abstand vom Bild-Element getrennt und horizontal bündig untereinander angeordnet. (2 P.) ✓
 (g) Die Textfarbe der Elemente ist mit Ausnahme des Bildtitels grau. (1 P.) ✓
 (h) Die Tag-Anzahl ist bezüglich der Pfeilspitze zentriert. (1 P.) ✓
 (i) Das Datum und das 'Optionen'-Icon sind rechtsbündig angeordnet. (1 P.) ✓
 (j) Die Tag-Anzeige ist nach oben hin bündig mit dem 'Optionen'-Icon. (1 P.) ✓

Bearbeitungshinweise

- Das Projekt `org.dieschnittstelle.iam.css_jsl_jsr` können Sie als Anschauungsbeispiel und Ausgangspunkt für diese und die weiteren Aufgaben bis JSR verwenden. Für die Aufgaben zu CSS sind insbesondere die Dateien `htm.html` und `css/css.css` von Interesse (deren Benennung sich an der Benennung der Lerneinheiten orientiert).
- Die Icons können Sie dem dort enthaltenen Verzeichnis `css/img/glyphicons/png` entnehmen. Beachten Sie, dass aufgrund der geltenden Lizenzbedingungen (<http://glyphicons.com/license/>) bei Speicherung in einem öffentlich zugreifbaren Verzeichnis, z.B. auf *Github*, ein Hinweis auf die Lizenz eingefügt werden muss, siehe z.B. die `LICENSE` Dateien aus den hier verwendeten Projekten.

- Zur Anpassung der Farbe der Icons können Sie in CSS die folgenden Filter-Funktion anwenden:
`filter: invert(60%); -webkit-filter: invert(60%);`
- **Anforderung 1** Für die Höhenberechnung des `<main>` Elements können Sie unter Anwendung von `calc()` die Höhen von Footer und Header von 100% subtrahieren. Beachten Sie, dass in den `calc()`-Ausdrücken Leerzeichen zwischen den Rechenoperatoren und den Operanden obligatorisch sind.
- **Anforderung 3** Wenn Sie den orangenen Rand als `border-top` des Footers umsetzen, müssen Sie ebenfalls zusätzlich die Dicke dieses Randes von 100% subtrahieren.
- **Anforderung 4** Den Puffer können Sie z.B. als Geschwister-Element von `` mit passender Höhe und Hintergrundfarbe realisieren.
- **Anforderung 4** Um die quadratischen Bildelemente umzusetzen, können ggf. überschüssige Bildbestandteile abgeschnitten werden. Bilder bekommen Sie z.B. von <http://placeimg.com/> mittels URLs der Form <http://placeimg.com/<width>/<height>>.

Lehrmaterial

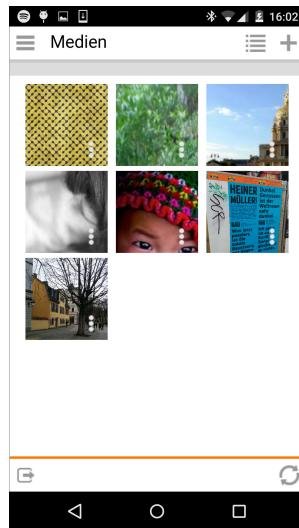
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMHTM/06struktur/index.shtml>
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMCSS/02ausdruck/index.shtml> (*insbesondere Anwendung von CSS*)

Ü CSS2 Umsetzung einer Kachelansicht

(11 Punkte)

Aufgabe

Stellen Sie die Listenansicht aus Ü CSS1 alternativ als Kachelansicht dar, *ohne die Dokumentenstruktur zu ändern oder zu vervielfältigen*, d.h. dieselben Elemente im DOM, die für die Listenansicht verwendet werden, sollen der Kachelansicht zugrunde liegen. Nutzen Sie zur Unterscheidung der beiden Ansichtsvarianten lediglich eine geeignete `class`-Zuweisung, z.B. auf dem Wurzelement der Ansicht.



Anforderungen

1. In der Kopfzeile soll anstelle des ‘Kachel’-Icons aus der Listenansicht ein ‘Liste’-Icon dargestellt werden (**1 P.**)
2. Für das ‘Liste’-Icon soll dasselbe DOM Element verwendet werden, das in der Listenansicht als ‘Kachel’-Icon realisiert wurde. (**1 P.**) **NOT APPROVED**
3. Die Kacheln sind quadratisch, linksbündig angeordnet und innerhalb der Gesamtansicht horizontal zentriert. (**3 P.**)
4. Die Anzahl der Kacheln in einer Zeile ist abhängig von der verfügbaren Bildschirmbreite, d.h. je nach Veränderung der Bildschirmbreite werden mehr oder weniger Kacheln in einer Zeile dargestellt. (**1 P.**)
5. In einer Kachel wird lediglich das Bild des zugrunde liegenden Listenelements dargestellt, alle anderen Elemente werden ausgeblendet. (**3 P.**)
6. Das ‘Optionen’-Icon befindet sich in der rechten unteren Ecke einer Kachel. (**2 P.**)

Bearbeitungshinweise

- **Anforderung 4** Hierfür können Sie eine fixe Breite und Höhe zuweisen, deren Werte Sie an den Dimensionen des von Ihnen präferierten Endgeräts ausrichten.

- **Anforderung 6** Zur Darstellung des Icons können Sie den folgenden Filter anwenden:

```
filter: invert(90%); -webkit-filter: invert(90%);
```

Lehrmaterial

- <http://www.mademyday.de/css-height-equals-width-with-pure-css.html>
(fortgeschrittene Lösung für skalierbare quadratische Kacheln)

JSL

Ü JSL1 Übergang zwischen Listen- und Kachelansicht (8 Punkte)

Aufgabe

Nutzen Sie das ‘Kachel’/‘Liste’-Bedienelement zum Umschalten zwischen den beiden Ansichten.

Anforderungen

1. Beim Zugriff auf die Anwendung soll zunächst die Listenansicht dargestellt werden. Die Betätigung des Bedienelements soll dann einen Übergang zur Kachelansicht initiieren. Nochmalige Betätigung führt Sie wieder zur Listenansicht zurück. (2 P.) ✓
2. Beim Umschalten der Ansichten soll die aktuell dargestellte Ansicht zunächst über einen Zeitraum von 2 Sekunden vollständig ausgeblendet werden. Danach soll die Zielansicht über einen Zeitraum von 1 Sekunde eingeblendet werden. (2 P.) ✓
3. Aus- und Einblenden soll für beide Richtungen des Ansichtsübergangs funktionsfähig sein, d.h. von der Listen- zur Kachelansicht und umgekehrt. (2 P.) ✓
4. Die Darstellung des Umschaltelements in der Kopfzeile soll erst mit dem Beginn des Einblendens der jeweiligen Zielansicht modifiziert werden. (1 P.) ✓
5. Nur der Hauptbereich sollen aus- und eingeblendet werden, Kopf- und Fußzeile sollen dauerhaft sichtbar und hinsichtlich ihrer vertikalen Position stabil sein. (1 P.) ✓

Bearbeitungshinweise

- Im Projekt `org.dieschnittstelle.iam.css_jsl_jsr` dient insbesondere die Skriptdatei `js/jsl.js` als Anschauungsbeispiel für die Umsetzung dieser und der folgenden Aufgabe.
- Für das Aus/Einblenden können Sie die Style-Property `opacity` verwenden.
 - `opacity: 0.0:` ‘vollkommen transparent’, d.h. unsichtbar
 - `opacity: 1.0:` ‘vollkommen intransparent’, d.h. ohne durchschimmernden Hintergrund sichtbar.
- Den Ansichtswechsel nach Ausblenden der Ansicht können Sie z.B. durch Setzen eines `transitionend` Event Listeners veranlassen. Eine Vorlage dafür finden Sie in der Umsetzung des ‘Toasts’ in den Implementierungsbeispielen.

Lehrmaterial

- <http://moodle.oncampus.de/modules/ir493/onmod/IAMJSL/02interakt/eingabe.shtml>
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMJSL/03abstimm/schritt.shtml>

Ü JSL2 Auswahl von Listenelementen

(8 Punkte)

Aufgabe

Reagieren Sie auf die Interaktion des Nutzers mit den in der Listenansicht dargestellten Elementen

Anforderungen

1. Bei Auftreten eines Click/Tap Ereignisses auf einem Listenelement soll der Titel des betreffenden Elements in einem `alert()` Dialog ausgegeben werden. Das Ereignis soll auf dem *gesamten* Listenelement mit Ausnahme des Optionen-Elements aus **Anforderung 2** berücksichtigt werden. (2 P.)
2. Bei Auftreten eines Click/Tap Ereignisses auf dem ‘Optionen’-Element eines Listenelements sollen der Titel des betreffenden Elements und die URL des dargestellten Bildes in einem `alert()` Dialog ausgegeben werden. (2 P.)
3. Der Dialog aus **Anforderung 1** soll bei Betätigung des ‘Optionen’-Elements nicht dargestellt werden. (1 P.)
4. Zur Darstellung der `alert`-Meldungen in **Anforderung 1** und **Anforderung 2** sollen keine redundanten Attribute auf den Bestandteilen der Listenelemente und keine hardcodierten Werte für spezifische Listenelemente verwendet werden. Die Meldungen sollen in beiden Ansichtsvarianten (Listen- und Kachelansicht) funktionsfähig sein. (2 P.)
5. Die Zuweisung der Funktionen zur Behandlung der Eingabeereignisse soll auf Ebene von JavaScript und nicht im HTML Dokument erfolgen. (1 P.)

Bearbeitungshinweise

- **Anforderung 2** Die Darstellung des `alert()` Dialogs wird später durch JSR2, **Anforderung 2**, ergänzt, wobei der `alert()` Dialog durch einen `confirm()` Dialog ersetzt wird. Als Vorbereitung sei die hier beschriebene Umsetzung jedoch empfohlen.

JSR

Ü JSR1 Dynamischer Aufbau der Listenansicht

(12 Punkte)

Aufgabe

Bauen Sie die in den vorangegangenen Übungen entwickelte Listenansicht inklusive der Bedienung aus den Übungen zu JSL dynamisch auf Grundlage eines vom Server gelieferten Arrays aus JSON Objekten auf. Die statischen Elemente aus den vorangegangenen Aufgaben sollen nicht mehr angezeigt werden.

Anforderungen

1. Greifen Sie unmittelbar nach dem Laden des HTML Dokuments mittels JavaScript auf die durch den Server bereit gestellten Daten zu. (3 P.)
2. Erstellen Sie für jedes Element des vom Server gelieferten Arrays ein Listenelement mit der in den vorangegangenen Aufgaben von Ihnen entwickelten Struktur und fügen Sie dieses der Listenansicht hinzu. (4 P.)
3. Leeren Sie bei Betätigung des ‘Refresh’-Elements die Listenansicht und befüllen Sie sie dann von neuem mit den Daten vom Server, ohne dass das dargestellte Dokument neu geladen wird. (2 P.)
4. Der Übergang zwischen Listen- und Kachelansicht und die Darstellung des `alert()`-Dialogs bei Auswahl eines Listenelements sollen auch für die dynamisch aufgebaute Ansicht voll funktionsfähig sein. Die Reaktion auf die Bedienung des ‘Optionen’-Elements werden wir in der folgenden Aufgabe neu umsetzen. (3 P.)

Bearbeitungshinweise

- Im Projekt `org.dieschnittstelle.iam.css_jsl_jsr` dient insbesondere die Skriptdatei `js/jsr.js` als Anschauungsbeispiel für die Umsetzung dieser und der folgenden Aufgabe. Der Array aus JSON Objekten, der geladen werden soll, wird als Inhalt der Datei `data/listitems.json` geliefert. Der Aufbau der darin enthaltenen JSON Objekte soll nicht modifiziert werden, Sie können aber weitere Objekte zu dieser Datei hinzufügen.
- **Anforderung 1** Hierfür können Sie, wie in `jsr.js` gezeigt, die Funktion `xhr()` aus der Skriptdatei `js/lib/xhr.js` nutzen, die die Verwendung von `XMLHttpRequest` für den Zugriff auf den Server kapselt.
- **Anforderung 3 Punkte werden nur vergeben, wenn kein Neuladen des Dokuments erfolgt.**

Lehrmaterial

- <http://moodle.oncampus.de/modules/ir493/onmod/IAMJSL/02interakt/dom.shtml>
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMJSL/02interakt/client.shtml>
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMJSL/02interakt/xmlhttp.shtml>

- <http://moodle.oncampus.de/modules/ir493/onmod/IAMJSL/02interakt/ajax.shtml>

Ü JSR2 Hinzufügung und Entfernen von Elementen

(7 Punkte)

Aufgabe

Ermöglichen Sie die Modifikation der Listenansicht und nutzen Sie dafür die vorhandenen Bedienelemente.

Anforderungen

1. Fügen Sie bei Betätigung des ‘+’-Elements ein neues Element zur Liste hinzu. Sie können dafür hardcodierte Werte für die Attribute `title`, `src`, `owner` und `numOfTags` verwenden (für die JSR1 die entsprechenden Werte aus der JSON Datei übernimmt). Setzen Sie als Wert des `added` Attributs den jeweils aktuellen Zeitstempel. (3 P.)
2. Ersetzen Sie den `alert()` Dialog aus JSR2, **Anforderung 2**, durch einen `confirm()` Dialog, der eine Rückfrage enthält, in der Sie wiederum den Titel und die URL des ausgewählten Elements nennen. Entfernen Sie bei Bestätigung der Rückfrage das betreffende Element aus der Liste. (2 P.)
3. Das Entfernen eines Elements mit Rückbestätigung und die Reaktion auf die Auswahl des Elements mittels `alert()`-Dialog sollen auch für neu hinzugefügte Elemente funktionsfähig sein. (1 P.)
4. Hinzufügen und Entfernen sollen in beiden Varianten der Ansicht, d.h. Listenansicht und Kachelansicht, funktionsfähig sein. (1 P.)

Bearbeitungshinweise

- **Anforderung 1** Das aktuelle Datum wird Ihnen formatiert durch den Ausdruck `(new Date()).toLocaleDateString()` geliefert.
- **Anforderung 1** Die genannten Attribute `src`, `owner` und `numOfTags` werden auch in der durch den Server bereit gestellten Datei `data/listitems.json` verwendet.

MWF

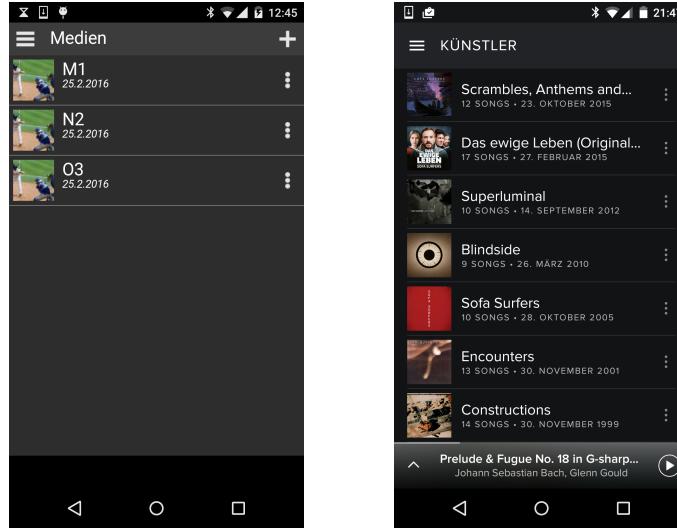
Ü MWF1 Listenansicht

(19 Punkte)

Aufgabe

Setzen Sie die nachfolgend links gezeigte und beschriebene Listenansicht gestalterisch und funktional um. Die Gestaltung dieser Ansicht wie auch der weiter in den folgenden Ansichten umgesetzten Dialoge orientiert sich am Beispiel der Spotify-App für Android.

Die Bepunktung dieser und der folgenden Aufgaben zu MWF berücksichtigt die Tatsache, dass alle genannten Anforderungen durch das angebotene Tutorial zur Lerneinheit MWF abgedeckt werden.



Anforderungen

1. Datenmodell

Die hier und im folgenden zu entwickelnde Anwendung soll es ermöglichen, `MediaItem` Objekte zu verwalten. Für den Zweck der Aufgaben zu MWF verfügen `MediaItem` Objekte über die folgenden Attribute:

- Titel (**1 P.**)
- Bildquelle (**1 P.**)
- Erstellungsdatum des `MediaItem` Objekts (**1 P.**)

2. Gestaltung

- Die Ansicht soll aus Kopfzeile, Hauptbereich und Fußzeile aufgebaut sein. (**1 P.**)
- Die Ansicht soll exakt 100% Höhe und Breite der verfügbaren Anzeigefläche einnehmen und ohne horizontales oder vertikales Scrolling realisiert werden. (**1 P.**)
- In der Kopfzeile sollen linksbündig ein ‘Sandwich’-Icon und die Überschrift ‘Medien’ sowie rechtsbündig ein ‘+’-Icon dargestellt werden. (**2 P.**)
- Listenelemente sollen linksbündig ein Bild-Element einen Titel mit darunter platziertem Datum sowie rechtsbündig ein ‘Optionen’-Icon verwenden. (**3 P.**)

- Im Titel- und Untertitelement sollen Titel bzw. Erstellungsdatum des `MediaItem`, im Bild-Element der Inhalt der Bildquelle des Objekts dargestellt werden (**3 P.**)

3. CRUD Operationen

- Beim Zugriff auf die Listenansicht sollen `MediaItem` Objekte aus einer lokalen IndexedDB Datenbank ausgelesen und in der Listenansicht dargestellt werden. (**2 P.**)
- Bei Betätigen des ‘+’-Bedienelements soll die Erstellung eines neuen Elements unter Verwendung des in MWF3 umgesetzten Dialogs erfolgen. (**1 P.**)
- Alle nachfolgend umgesetzten CRUD Operationen sollen bezüglich der lokalen Datenbank durchgeführt werden. (**3 P.**)

Lehrmaterial

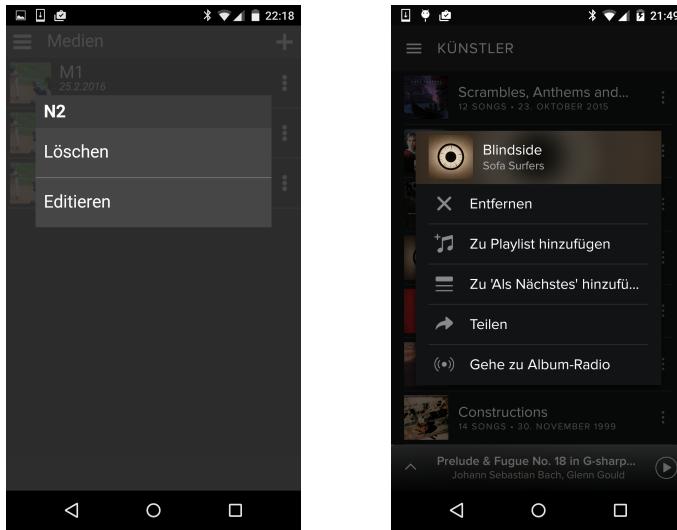
- MWF Tutorial, Kap. 4.1-4.3
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMLDS/03aktuell/indexed.shtml> (Hintergrundstoff zu IndexedDB)

Ü MWF2 Aktionsmenü für Listenelemente

(8 Punkte)

Aufgabe

Erstellen Sie ein als Dialog realisiertes Aktionsmenü für Listenelemente



Anforderungen

1. Das Aktionsmenü soll in einer Titelleiste den Titel des ausgewählten Listenelements anzeigen und darunter die beiden Aktion ‘Löschen’ und ‘Editieren’ anbieten. (2 P.)
2. Das Aktionsmenü soll bei Betätigung des ‘Optionen’-Icons eines Listenelements geöffnet werden. (1 P.)
3. Bei Click/Tap außerhalb des Menüs wird dieses wieder geschlossen. (1 P.)
4. Wird das Aktionsmenü geöffnet, dann soll die Listenansicht über einen kurzen Zeitraum hinweg partiell ausgeblendet werden. (1 P.)
5. Bei Auswahl der ‘Löschen’-Aktion soll des betreffende `MediaItem` aus der lokalen Datenbank gelöscht und aus der Listenansicht entfernt werden. (2 P.)
6. Bei Auswahl der ‘Editieren’-Aktion soll der in MWF3 entwickelte Dialog zum Ändern eines `MediaItem` dargestellt werden. (1 P.)

Bearbeitungshinweise

- **Anforderung 5** In NJM2 werden Sie hier vor dem Löschen einen Rückbestätigungsdialog umsetzen.

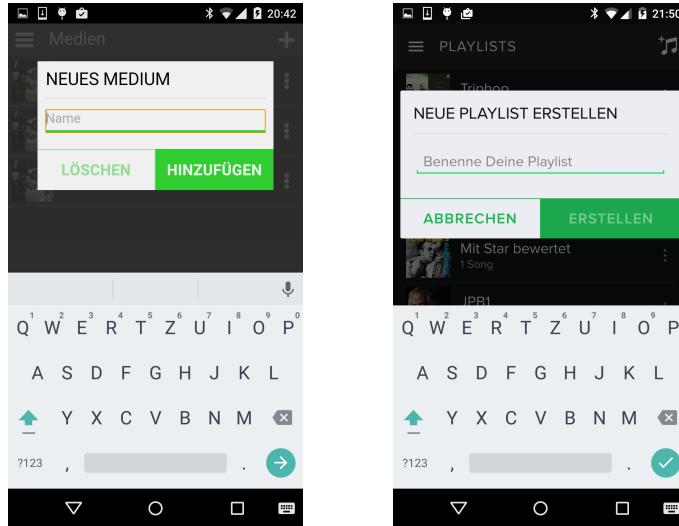
Lehrmaterial

- MWF Tutorial, Kap. 4.4.2

Ü MWF3 Dialog zur Erstellung und Modifikation von Elementen (13 Punkte)

Aufgabe

Ermöglichen Sie die Erstellung und Modifikation von neuen Elementen durch einen Dialog mit Texteingabefeld.



Anforderungen

1. Der Dialog soll über eine Titelzeile verfügen, in der je nach Anwendungsfall wahlweise der Text ‘Neues Medium’ oder ‘Medium editieren’ angezeigt wird. (**1 P.**)
2. Der Dialog soll ein Texteingabefeld bereit stellen, das je nach Anwendungsfall entweder leer ist oder den Titel des zu editierenden `MediaItem` Objekts darstellt. (**1 P.**)
3. Beim Öffnen des Dialogs soll das Texteingabefeld fokussiert werden, u.a. um ohne weitere Nutzerinteraktion die Anzeige der Tastatur auf mobilen Geräten zu veranlassen. (**1 P.**)
4. Der Dialog soll außerdem über zwei Bedienelemente verfügen, die das Löschen bzw. Erstellen oder Modifizieren von `MediaItem` Objekten veranlassen. (**2 P.**)
5. Das ‘Löschen’-Element soll nur bei Editieren eines bestehenden Elements aktiv sein. (**1 P.**)
6. Die Betätigung des ‘Löschen’-Elements soll das dargestellte `MediaItem` aus der verwendeten Datenbank entfernen und die Listenansicht aktualisieren. (**2 P.**)
7. Das Bedienelement zum Erstellen bzw. Modifizieren soll je nach Anwendungsfall unterschiedlich beschriftet sein. (**1 P.**)
8. Je nach Anwendungsfall soll bei Betätigung des Elements **Anforderung 7** entweder ein neues `MediaItem` erstellt bzw. ein bestehendes modifiziert und die Listenansicht entsprechend aktualisiert werden. (**3 P.**)
9. Bei Öffnen des Dialogs soll die Listenansicht über einen kurzen Zeitraum hinweg ausgeblendet werden, Click/Tap außerhalb des Dialogs schließt diesen. (**1 P.**)

Lehrmaterial

- MWF Tutorial, Kap. 4.4.4

Ü MWF4 Leseansicht**(11 Punkte)****Aufgabe**

Ergänzen Sie die Anwendung um eine Ansicht, in der ein ausgewähltes `MediaItem` dargestellt wird.

Anforderungen

1. In der Kopfzeile der Leseansicht sollen neben dem ‘Sandwich’-Icon der Titel des `MediaItem` Objekts und rechtsbündig ein ‘Papierkorb’-Icon dargestellt werden. (**2 P.**)
2. In der Fußzeile soll linksbündig ein ‘Zurück’-Icon angezeigt werden. (**1 P.**)
3. Der Hauptteil der Ansicht zeigt den Inhalt der Bildquelle des `MediaItem` über die gesamte Bildschirmbreite ohne Abschneiden des Bildes und ohne horizontales Scrolling, ggf. mit vertikalem Scrolling. (**2 P.**)
4. Geöffnet wird die Leseansicht bei Auswahl eines `MediaItem` in der Listenansicht. (**1 P.**)
5. Die Rückkehr zur Listenansicht wird durch Betätigung des ‘Zurück’-Icons veranlasst. (**1 P.**)
6. Der Übergang von der Listenansicht zur Leseansicht und zurück erfolgt unter kurzem Aus- und Einblenden der Ansichten. (**1 P.**)
7. Die Betätigung des ‘Papierkorb’-Icons veranlasst das Löschen des Icons aus der verwendeten Datenbank und die Rückkehr zur Listenansicht. **Hier und in allen folgenden Anforderungen, die eine ‘Rückkehr’ in eine Vorgängeransicht beschreiben, werden Punkte nur dann vergeben, falls die Rückkehr tatsächlich als solche umgesetzt wird. Die Verwendung von `nextView()` ist hierfür nicht zulässig..(2 P.)**
8. Wird ein `MediaItem` in der Leseansicht gelöscht, dann ist dieses nach Rückkehr in die Listenansicht nicht mehr in der Liste enthalten. (**1 P.**)

Lehrmaterial

- MWF Tutorial, Kap. 5

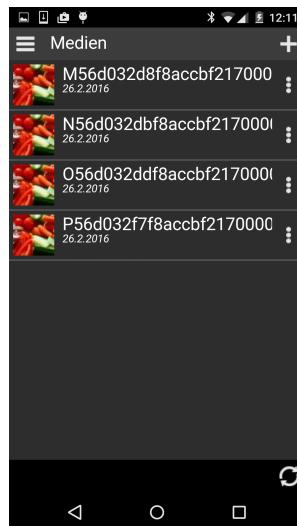
NJM+LDS

Ü NJM1 Lokale und server-seitige CRUD Operationen (8 Punkte)

Aufgabe

Greifen Sie als Alternative zu der in den MWF Übungen verwendeten lokalen IndexedDB Datenbank auf eine server-seitige MongoDB Datenbank zu und nutzen Sie dafür die durch den NodeJS Webserver bereit gestellte REST Schnittstelle, deren Dokumentation Sie man Ende dieser Aufgaben finden. Erlauben Sie das Umschalten zwischen lokalen und server-seitigen CRUD Operationen.

Wenn Sie das Tutorial zu MWF Schritt für Schritt nachvollzogen haben, dann sollte Ihre Anwendung den Datentyp `MediaItem` im Abschnitt *Verwendung des MWF Entity Managers* bereits für Remote CRUD Operationen registriert haben. Aus diesem Grund werden auch für diese Aufgabe nur vergleichsweise wenige Punkte vergeben. Beachten Sie, dass die Verwendung server-seitiger CRUD Operationen für die Umsetzung der nachfolgenden Aufgaben zu Formularen erforderlich ist.



Anforderungen

1. Das Umschalten zwischen client-seitigen und server-seitigen CRUD Operationen soll durch ein Bedienelement in der Fußleiste der Listenansicht ermöglicht werden. (3 P.)
2. Nach Durchführung des Umschaltens soll die Listenansicht mit den `MediaItem` Objekten der nun zugewiesenen Datenquelle von neuem initialisiert werden. (3 P.)
3. Die aktive Variante der CRUD Operationen soll in einem Textelement mit Beschriftung 'local' bzw. 'remote' in der Fußleiste linksbündig angezeigt werden. (2 P.)

Bearbeitungshinweise

- Das MWF Framework verfügt über eine generische Komponente – `GenericCRUDImplRemote` – für den Zugriff auf server-seitige REST Schnittstellen, die der nachfolgenden Dokumentation entsprechen. Falls Sie das Tutorial durchlaufen haben, wurde diese Komponente bereits im Zuge der Verwendung des MWF Entity Managers für den Aufruf von remote CRUD Operationen auf `MediaItem` Instanzen initialisiert.

- **Anforderung 1:** Das Umschalten können Sie durch Aufruf der Funktion `switchCRUD()` auf dem `Application`-Objekt veranlassen, das als Wert des öffentlichen Attributs `application` auf allen View Controllern gesetzt wird. Wenn Sie das Tutorial durchlaufen haben, ist dieses Objekt eine Instanz des anwendungsspezifischen Typs `MyApplication`, der den `Application` Typ von MWF beerbt. Übergeben wird der Funktion `"local"` für die Aktivierung lokaler CRUD Operationen oder `"remote"` für den Zugriff auf server-seitige Operationen.
- **Anforderung 1:** Welche CRUD Operationen aktuell verwendet werden, kann durch Auslesen des öffentlichen Attributs `currentCRUDScope` auf dem `Application`-Objekt ermittelt werden. Desse Wert ist entweder `"local"` oder `"remote"`.
- **Anforderung 2** Die Neuinitialisierung der Listenansicht kann durch Aufruf der öffentlichen Funktion `initialiseListView()` auf dem View Controller veranlasst werden. Dieser müssen Sie die mittels `readAll()` neu ausgelesenen `MediaItem` Objekte übergeben.

Lehrmaterial

- <http://moodle.oncampus.de/modules/ir493/onmod/IAMNJM/02nodejs/index.shtml> (Hintergrundstoff zu NodeJS)
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMNJM/03mongodb/index.shtml> (Hintergrundstoff zu MongoDB)
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMNJM/04daten/index.shtml> (Zugriff auf MongoDB via NodeJS)

Dokumentation der REST Schnittstelle

Operation	Methode	URI	Request Body	Response Body im Erfolgsfall
<code>create</code>	POST	<code>/api/mediaitems</code>	<code>MediaItem</code> Objekt ohne <code>_id</code>	<code>{data:obj}</code> , wobei <code>obj</code> das <code>MediaItem</code> Objekt mit <code>_id</code> Attribut
<code>read all</code>	GET	<code>/api/mediaitems</code>	–	<code>{data:objs}</code> , wobei <code>objs</code> ein Array von <code>MediaItem</code> Objekten
<code>read</code>	GET	<code>/api/mediaitems/{_id}</code>	–	<code>{data:obj}</code> , wobei <code>obj</code> das <code>MediaItem</code> Objekt mit der angegebenen <code>_id</code>
<code>update</code>	PUT	<code>/api/mediaitems/{_id}</code>	<code>MediaItem</code> Objekt ohne <code>_id</code>	<code>{data:1}</code> , d.h. die übermittelten Attribute des <code>MediaItem</code> Objekts mit der angegebenen <code>_id</code> wurden aktualisiert.
<code>delete</code>	DELETE	<code>/api/mediaitems/{_id}</code>	–	<code>{data:1}</code> , d.h. das <code>MediaItem</code> Objekt mit der angegebenen <code>_id</code> wurde gelöscht.
<code>upload</code>	POST	<code>/api/upload/</code>	Multipart-Formular mit n Textfeldern und 1 Feld mit Dateiinhalt	<code>{data:obj}</code> , wobei <code>obj</code> ein Objekt mit n Attributen für die übermittelten Textfelder und 1 Attribut, dessen Name der Name des Felds mit Dateiinhalt und dessen Wert die URI des hochgeladenen Inhalts ist.

Weitere Hinweise:

- Mit ‘Objekt’ ist stets ein Objekt im JSON Format gemeint.

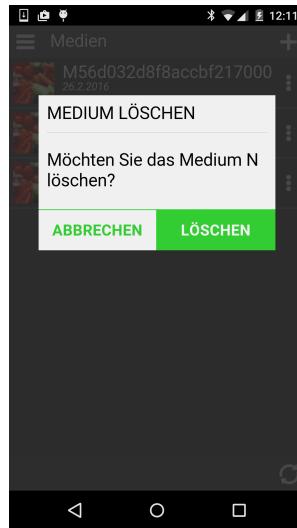
- Im Erfolgsfall wird in allen Fällen der Statuscode 200 gesetzt.
- Bei Ausführung von NodeJS `webserver.js` kann auf alle Ressourcen-URIs jeweils unter der URL `http://{ip}:{port}` zugegriffen werden, wobei `ip` und `port` die IP-Adresse und der Port des NodeJS Webservers sind.

Ü NJM2 Rückbestätigung der Lösch-Operation

(7 Punkte)

Aufgabe

Wählt der Nutzer in der Listenansicht aus dem Aktionsmenü für Listenelemente die Aktion ‘Löschen’, dann soll die Durchführung der betreffenden CRUD Operation durch Darstellung eines Dialogs rückbestätigt werden.



Hier müssen wir noch ran

Anforderungen

1. Der Dialog soll als Popup-Dialog konsistent mit der visuellen Gestaltung der Anwendung realisiert werden – ein Beispiel sehen Sie oben. Bei Verwendung eines `confirm()`-Dialogs werden pauschal nur 2 Punkte für die gesamte Aufgabe vergeben.
2. Der Dialog soll den Titel des zu löschen `MediaItem` Objekts nennen.(**1 P.**)
3. Der Dialog soll über zwei Buttons verfügen, mit denen der Löschvorgang bestätigt bzw. abgebrochen werden kann.(**2 P.**)
4. Im Fall eines Abbruchs soll der Dialog geschlossen werden.(**1 P.**)
5. Bei Bestätigung soll das ausgewählte Element gelöscht, der Dialog geschlossen und die Listenansicht aktualisiert werden.(**3 P.**)

Bearbeitungshinweise

- Bei der visuellen und funktionalen Umsetzung des Dialogs können Sie sich an der im Tutorial beschriebenen Umsetzung des Editier-Dialogs für `MediaItem` orientieren.
- Es reicht aus, wenn der Rückbestätigungsdialog an der beschriebenen Stelle – bei Auswahl von ‘Löschen’ aus dem in MWF2 entwickelten Aktionsmenü für Listenelemente – angezeigt wird. Alls anderen Fälle, in denen das Löschen von `MediaItem` Instanzen ausgeführt werden kann, brauchen nicht rückbestätigt zu werden.

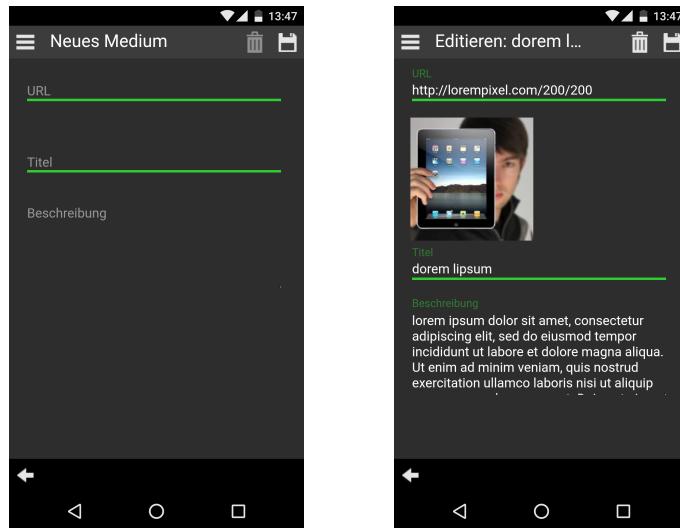
FRM

Ü FRM1 Editieransicht

(20 Punkte)

Aufgabe

Erstellen Sie eine Editieransicht für die Erstellung und Modifikation von `MediaItem` Objekten und verwenden Sie diese Ansicht anstelle des Editier-Dialogs aus MWF. Lediglich im Aktionsmenü für Listenelemente soll für die Aktion ‘Editieren’ nach wie vor der Dialog angezeigt werden und funktionsfähig sein. Das unten abgebildete Beispiel beinhaltet bereits Funktionen, die in den Aufgaben zur Lerneinheit MFM umgesetzt werden.



Anforderungen

1. Die Ansicht soll über ein Bedienelement zum *Erzeugen* bzw. *Modifizieren* von `MediaItem` Objekten verfügen, das als `submit`-Input eines Formulars realisiert wird und in der Kopfzeile der Ansicht dargestellt wird. (2 P.)
2. Die Ansicht soll über ein weiteres Bedienelement zum *Löschen* von `MediaItem` Objekten verfügen, das ebenfalls in der Kopfzeile dargestellt wird. (1 P.)
3. Das Formular zum Erzeugen/Modifizieren soll über geeignete Bedienelemente zur Erstellung und Darstellung der Attribute `src`, `title` und `description` von `MediaItem` verfügen. (3 P.)
4. Falls bei Darstellung der Ansicht noch kein `MediaItem` existiert, dann soll bei Betätigung des Bedienelements aus **Anforderung 1** die `create()` Funktion für `MediaItem` aufgerufen werden und die Vorgängeransicht aktualisiert angezeigt werden. (2 P.)
5. Wenn kein `MediaItem` existiert, dann soll das Bedienelement aus **Anforderung 2** nicht bedienbar sein. (2 P.)
6. Existiert bereits ein `MediaItem`, dann sollen dessen Attribute in den Bedienelementen des Formulars aus **Anforderung 3** dargestellt werden. (2 P.)

7. Die Inhalte der Bildquelle des `src` Attributs sollen für existierende `MediaItem` Objekte in einem ‘Vorschaubild’ dargestellt werden. Auch bei Erzeugen eines neuen `MediaItem` soll das Vorschaubild dargestellt werden. Im letzteren Fall soll keine Reactive Data Binding Expression verwendet, sondern die Bildquelle ‘manuell’ im View Controller zugewiesen werden.(2 P.)
8. Wenn ein `MediaItem` existiert, dann soll bei Betätigung des Bedienelements aus **Anforderung 1** die `update()` Funktion für `MediaItem` ausgeführt werden und die Vorgängeransicht aktualisiert angezeigt werden. (2 P.)
9. Wenn ein `MediaItem` existiert, dann soll bei Betätigung des Bedienelements aus **Anforderung 2** die `delete()` Funktion ausgeführt werden und die Vorgängeransicht angezeigt werden. (1 P.)
10. Existiert noch kein `MediaItem`, dann soll in der Kopfzeile der Text ‘Neues Medium’ angezeigt werden, andernfalls der Titel des Objekts. (1 P.)
11. In der Fußzeile soll die Ansicht über ein Bedienelement verfügen, das die Rückkehr in die Vorgängeransicht erlaubt. (1 P.)
12. Stellen Sie die Editieransicht anstelle des Editier-Dialogs aus MWF3 dar, wenn in der Listenansicht das Bedienelement zum Hinzufügen eines neuen `MediaItem` betätigt wird. Aus dem Aktionsmenü der Listenansicht soll für das Editieren eines bestehenden `MediaItem` nach wie vor der Editier-Dialog angezeigt werden. Um die Editieransicht für das Editieren eines bestehenden Objekts zu verwenden, wird in der folgenden Aufgabe die Leseansicht erweitert. (1 P.)

Bearbeitungshinweise

- Zur Umsetzung des oben dargestellten Layouts können Sie die drei Eingabeelemente jeweils als Kindelement eines `<fieldset>` Elements umsetzen und die Feldnamen darin als `<legend>` angeben. Das `<fieldset>` markieren Sie dann mit der Klasse `mwf-material`. Für die ‘Beschreibung’ können Sie eine `<textarea>` nutzen.
- **Anforderung 1** Um ein `submit`-Element außerhalb des Formulars zu platzieren, zu dem es gehört, können Sie darauf ein `form` Attribut setzen, dessen Wert die `id` des Formulars ist.
- **Anforderung 7** Siehe zur Umsetzung eines Vorschaubildes für neu zu erstellende `MediaItem` Objekte die Aufgabe MME2

Lehrmaterial

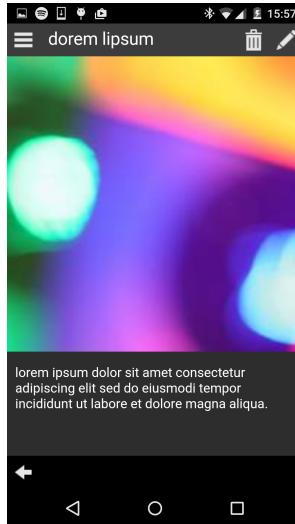
- <http://moodle.oncampus.de/modules/ir493/onmod/IAMFRM/02html/index.shtml>, v.a.: <http://moodle.oncampus.de/modules/ir493/onmod/IAMFRM/02html/verarbei.shtml>

Ü FRM2 Anpassung der Leseansicht

(7 Punkte)

Aufgabe

Nehmen Sie Änderungen an der Leseansicht für `MediaItem` aus MWF4 vor und ermöglichen Sie den Übergang von der Leseansicht zur Editieransicht aus FRM1.



Anforderungen

1. Stellen Sie den Wert des `description` Attributs von `MediaItem` unterhalb der Anzeige des Bildes dar.(1 P.)
2. Fügen Sie zur Kopfzeile außen rechts ein Bedienelement zum Editieren des dargestellten `MediaItem` hinzu. Wird das Bedienelement betätigt, dann soll die Editieransicht für das dargestellte `MediaItem` geöffnet werden.(1 P.)
3. Wird die Editieransicht aus der Leseansicht geöffnet und wird dort eine Änderung des `MediaItem` Objekts vorgenommen, dann soll diese Änderung bei der Rückkehr in die Leseansicht und daran anschließend auch in der Listenansicht dargestellt werden. (3 P.)
4. Wird die Editieransicht aus der Leseansicht geöffnet und wird das `MediaItem` Objekt dort gelöscht, dann soll die Leseansicht nicht mehr angezeigt werden.(2 P.)

Bearbeitungshinweise

- **Anforderung 3** Wenn Sie *Ractive Templates* entsprechend dem Tutorial verwenden, dann übergeben Sie das darzustellende `MediaItem` Objekt in einem Wrapper-Objekt, z.B. `{item: mediaItem}`. Zur Aktualisierung der Ansicht können Sie die Funktion `update()` auf `viewProxy` aufrufen und dieser das darzustellende aktualisierte `MediaItem` Objekt gewrappt übergeben.
- **Anforderung 4** Hinweise zur Umsetzung können Sie dem Abschnitt zu Event Notifikationen im Tutorial entnehmen, insbesondere dem Hinweis auf die Markierung von View Controllern als obsolet.

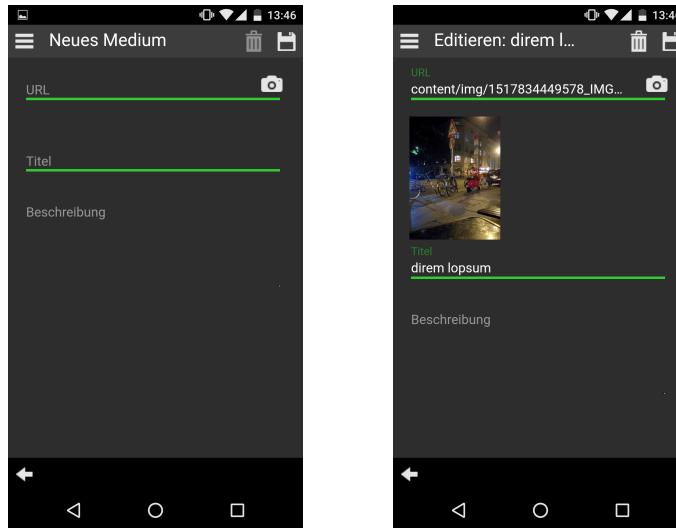
MFM

Ü MFM1 Multipart Formular für Editieransicht

(17 Punkte)

Aufgabe

Erweitern Sie das Formular aus FRM1 um die Möglichkeit, alternativ zur textuellen Eingabe einer URL ein Bild durch Dateiauswahl hochzuladen und dieses mit einem `MediaItem` Objekt zu assoziieren.



Anforderungen

1. Die alternative Eingabemöglichkeit soll nur verfügbar sein, wenn remote CRUD Operationen verwendet werden. Bei Verwendung lokaler CRUD Operationen soll nur die Eingabe einer URL möglich sein. (2 P.)
2. Die Befüllung des 'URL' Feldes soll obligatorisch sein. (1 P.)
3. Nur wohlgeformte URLs sollen verarbeitet werden. (1 P.)
4. Wird die alternative Eingabemöglichkeit ausgewählt, dann soll nach Auswahl eines Bildes vor Absenden der Formulardaten eine geeigneter Wert im URL Feld dargestellt werden. (2 P.)
5. Nach Auswahl der Bilddatei sollen deren Inhalte außerdem als Vorschaubild dargestellt werden. (2 P.)
6. Die Angabe eines Titels soll obligatorisch sein. (1 P.)
7. Die Angabe einer Description soll optional sein. (1 P.)
8. Falls die URL als Text eingegeben wurde, soll bei Absenden des Formulars die Erstellung bzw. die Aktualisierung des `MediaItem` Elements wie in Ü FRM1 erfolgen. (1 P.)
9. Falls eine Bilddatei ausgewählt wurde, soll diese bei Absenden des Formulars auf den Server hochgeladen werden. (4 P.)
10. Der Rückgabewert des Uploads aus **Anforderung 9** soll dann zur Erstellung bzw. Aktualisierung des `MediaItem` Objekts verwendet werden. (2 P.)

Bearbeitungshinweise

- Zur Umsetzung des oben dargestellten Layouts können Sie bei Verwendung der bezüglich Ü FRM1 beschriebenen `mwf-material` Klasse dem `<fieldset>` Element für die URL ein Dateiauswahl-Element sowie ein auf letzteres bezogenes `<label>` Element hinzufügen das als `mwf-imgbutton` realisiert wird. Letztere Elemente müssen *nach* dem Texteingabe-Element platziert und mit der Klasse `mwf-material-altinput` versehen werden. Dem Texteingabeelement fügen Sie die Klasse `mwf-material-altinput-target` hinzu.
- **Anforderung 3:** Sie können dafür die in HTML vorgesehene Gültigkeitsprüfung nutzen.
- **Anforderung 4:** Sie können dafür z.B. eine ObjectURL verwenden (siehe *Lehrmaterial* unten).

Lehrmaterial

- <http://moodle.oncampus.de/modules/ir493/onmod/IAMMF03multi/index.shtml>, v.a.: <http://moodle.oncampus.de/modules/ir493/onmod/IAMMF03multi/init.shtml>
- <https://developer.mozilla.org/en-US/docs/Web/API/URL/createObjectURL> (Object URLs) vs. <http://moodle.oncampus.de/modules/ir493/onmod/IAMMME/03aufnahme/input.shtml> (Data URLs)

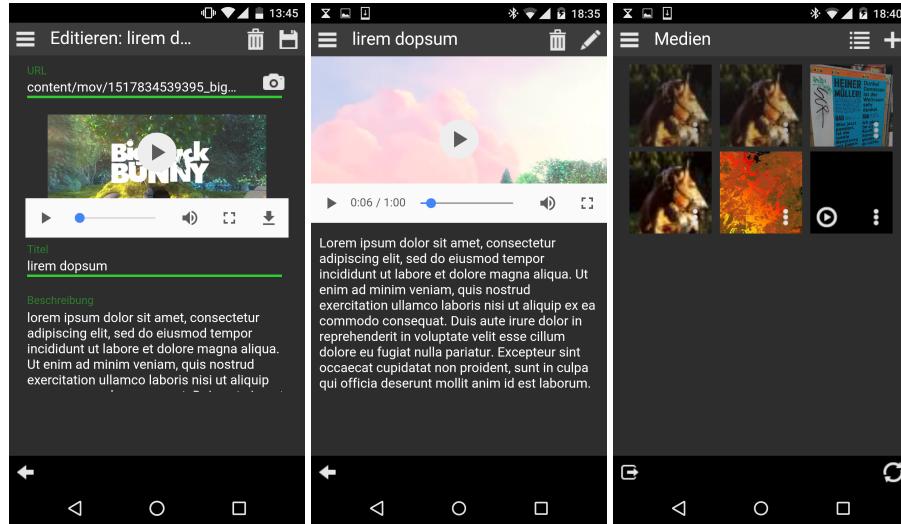
MME

Ü MME1 Verwendung von Videos als Alternative zu Bildern

(10 Punkte)

Aufgabe

Ermöglichen Sie in der Editieransicht das Hochladen von Videos und passen Sie die Ansichten, die Bilder anzeigen, für die Anzeige bzw. das Abspielen von Videos an.



Anforderungen

1. Verweist ein `MediaItem` auf ein Video, dann sollen Editieransicht und Leseansicht das Video anzeigen und das Abspielen ermöglichen. (5 P.)
2. In der Leseansicht soll das Video ohne horizontales Scrollen die gesamte Breite der Anzeigefläche ausfüllen. (1 P.)
3. In der Listenansicht sollen `MediaItem` Objekte mit Video durch ein zusätzliches Icon in der linken unteren Bildecke markiert werden (oben am Beispiel der Kachelansicht illustriert – diese brauchen Sie nicht umzusetzen). (2 P.)
4. In den Ansichten, in denen das Video abgespielt werden kann, soll das Abspielen ggf. beim Verlassen der Ansicht gestoppt werden. (2 P.)

Bearbeitungshinweise

- **Anforderung 3** Sie können davon ausgehen, dass Videos im Landscape-Format vorliegen, und die Darstellung des quadratischen Bildausschnitts dem entsprechend anpassen.
- **Anforderung 4** Dafür kann auf dem Video-Element die Funktion `pause()` aufgerufen werden.

Lehrmaterial

- <http://moodle.oncampus.de/modules/ir493/onmod/IAMMME/02wieder/index.shtml>

OFF

Ü OFF1 Offline-Anwendung

(6 Punkte)

Aufgabe

Machen Sie die in MWF bis MME entwickelte Anwendung als Offline-Webanwendung mit lokalen CRUD Operationen verfügbar. Die Anwendung soll auch ohne laufenden oder zugreifbaren NodeJS Server im Browser ausgeführt werden können, ggf. aber über eine bestehende Internetverbindung auf externe Bildquellen, z.B. auf placeimg.com, zugreifen können.

Anforderungen

1. Alle statischen Ressourcen der Anwendung sollen durch Deklaration eines Cache-Manifests als Offline-Ressourcen geladen werden. (3 P.)
2. Geben Sie als Online-Ressourcen im Manifest eine Auswahl von URLs für Bilder an, z.B. <http://placeimg.com/>, sowie die folgenden partiellen relativen URLs, die für den Zugriff auf die server-seitige REST Schnittstelle und hochgeladene Ressourcen verwendet werden: `api`, `available`, `content` (1 P.)
3. Falls beim Starten der Anwendung kein Zugriff auf den NodeJS Server möglich ist, sollen die lokalen CRUD Operationen ggf. automatisch ausgewählt und die Möglichkeit des Umschaltens der CRUD Operationen unterbunden werden. (2 P.)

Bearbeitungshinweise

- **Anforderung 3** Zum Testen des Zugriffs auf den Server können Sie die Funktion `isWebserverAvailable()` des `mwfUtils` Moduls verwenden. Das Modul, in welchem `mwfUtils` verwendet werden soll muss dafür in seiner initialen `define()` Anweisung den Modulnamen '`mwfUtils`' angeben und die Modulimplementierung in der ebendort deklarierten Funktion als Argument entgegennehmen.

Lehrmaterial

- <http://moodle.oncampus.de/modules/ir493/onmod/IAMOFF/01offline/index.shtml>

Weiterführende Hinweise

- Die im Lehrmaterial beschriebene Nutzung des Offline Manifest ist zum gegenwärtigen Zeitpunkt als browserunterstützter Standard bereits abgekündigt, wird aber nach wie vor u.a. von Chrome und Firefox unterstützt. Als Alternative werden *Service Worker* als bei weitem mächtigeres Ausdrucksmittel vorgeschlagen, die die Kontrolle über das Offline-Verhalten einer Anwendung und über die dafür ggf. zu cachenden Ressourcen vollständig in die Hand der Anwendungsentwicklung geben: https://developer.mozilla.org/en/docs/Web/API/Service_Worker_API
- Eine exemplarische Implementierung eines Service Workers finden Sie im Projekt <https://github.com/dieschnittstelle/org.dieschnittstelle.iam.mwf.sample>. Dieser versucht, ein Cache Manifest entsprechend dem in OFF beschriebenen Standard einzuleSEN und nimmt das Caching der dort aufgelisteten Ressourcen vor. Falls Sie diese Funktionalität

nutzen möchten, können Sie das für die oben beschriebene Aufgabe entwickelte Cache Manifest an die Beispielanwendung anpassen.

- Die Anwendung beinhaltet außerdem ein *Webapp Manifest*, wie es im Rahmen der aus dem Umfeld von Google getriebenen *Progressive Webapps* Initiative (<https://developers.google.com/web/progressive-web-apps/>) vorgeschlagen wird. Siehe dafür die folgenden Quellen:

- <https://developers.google.com/web/fundamentals/web-app-manifest/>
- <https://developer.mozilla.org/en-US/docs/Web/Manifest>
- <https://www.w3.org/TR/appmanifest/>

Mit Blick auf das Manifest handelt es sich bei Progressive Webapps aus Sicht des Autors um eine recht geradlinige Adaption der im Lehrmaterial zu OFF beschriebenen, von Mozilla vorgeschlagenen, *Open Web Apps*.

- Um das Manifest zu nutzen und die Anwendung als Webapp zu ‘installieren’ können Sie in Chrome die Entwicklertools öffnen und dort im Tab *Application* die Aktion *Add to homescreen* ausführen. Die Anwendung wird Ihnen dann unter *Apps* zur Auswahl angeboten.
- Die beschriebenen Funktionen können Sie bei Bereitstellung der Anwendung unter `localhost` bzw. `127.0.0.1` im Browser Ihres Entwicklungsrechners nutzen. Für den Zugriff von einem mobilen Endgerät muss der NodeJS Webserver im HTTPS Modus gestartet werden, wofür die Verwendung von mit einer *passphrase* geschützten SSL Zertifikaten erforderlich ist, die in `webserver.js` eingelesen werden. Nehmen Sie dafür ggf. Änderungen in `webserver.js` vor.

NAV

