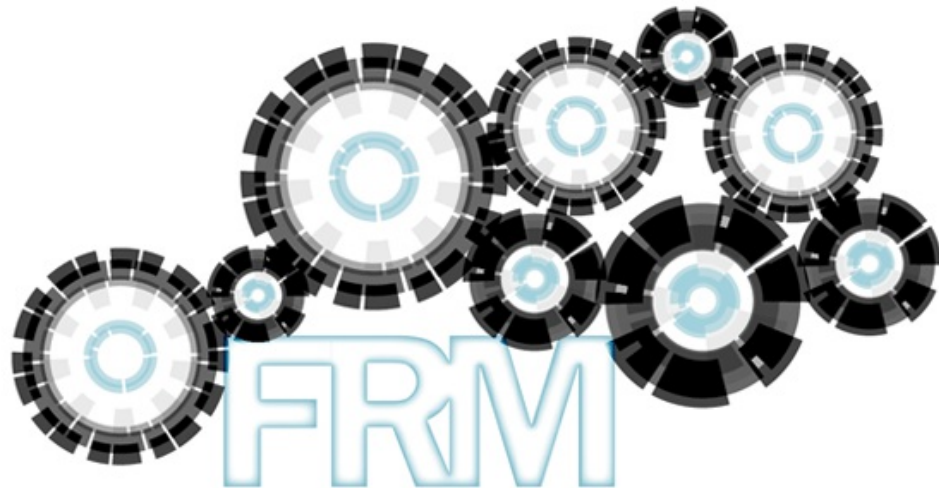


Hinweis:

Diese Druckversion der Lerneinheit stellt aufgrund der Beschaffenheit des Mediums eine im Funktionsumfang stark eingeschränkte Variante des Lernmaterials dar. Um alle Funktionen, insbesondere Animationen und Interaktionen, nutzen zu können, benötigen Sie die On- oder Offlineversion. Die Inhalte sind urheberrechtlich geschützt.
©2018 Beuth Hochschule für Technik Berlin

FRM - CRUD-Datenzugriff mit Formularen



Überblick und Lernziele

Nachdem sich die vorangegangene Lerneinheit ausführlich mit der serverseitigen Ausführung von CRUD-Operationen und deren Initiierung via HTTP-Requests beschäftigt hat, wollen wir nun die Ausdrucksmittel einführen, die wir auf Seiten des User Interfaces einer mobilen Anwendung benötigen, um dem Nutzer die Ausführung von Datenzugriffsoperationen zu ermöglichen und die Daten zu bestimmen, die für die Ausführung einer Operation verwendet werden sollen. Diesem Zweck dienen Formulare, und so stehen diese bzw. die Ausdrucksmittel, die HTML und JavaScript für Formulare zur Verfügung bereitstellen, denn auch im Zentrum der aktuellen Lerneinheit.

Bevor wir uns mit diesen Ausdrucksmitteln beschäftigen, wollen wir aber zunächst die Verwendung von Formularen in Bezug zu den verschiedenen Nutzungszwecken setzen, die die Bereitstellung eines Formulars für die Nutzer einer Anwendung überhaupt erst motivieren. Auf die Anforderungen an die Funktionalität von Formularen, die sich aus ihrem Verwendungszweck ableiten lassen, lassen sich nicht zuletzt auch Richtlinien für die Gestaltung von Formularen zurückführen, für deren Berücksichtigung wir im Rahmen der implementatorischen Umsetzung von Formularen als Anwendungsentwickler verantwortlich sind und für die wir uns Unterstützung durch die von uns verwendete Programmiersprache wünschen.

In welcher Form uns HTML diesbezüglich unterstützt und hinsichtlich welcher Anforderungen Defizite bestehen – die jedoch größtenteils durch existierende Frameworks kompensiert werden – werden Sie im Rahmen dieser Lerneinheit erfahren.

Unter Bezugnahme auf die Funktionalität unserer Beispielanwendung werden wir zunächst nach einer Darstellung der grundlegenden Aspekte von Formularen die wichtigsten Merkmale von HTML-Formularen im Hinblick auf die Ausführung von CRUD-Operationen erläutern. Wir werden uns in der nächsten Lerneinheit dann mit fortgeschrittenen Aspekten von Formularen beschäftigen und die aktuell verfügbaren Ausdrucksmittel zur *Überprüfung der Gültigkeit* von Formularen – der *Formularvalidierung* – zur *Gestaltung von Formularen* sowie zur Verwendung von Formularen zum Zweck des Uploads von Daten beschäftigen.

Wie bei den meisten der in unserer Veranstaltung behandelten Ausdrucksmittel handelt es sich hier um Aspekte von Webtechnologien, die unabhängig davon sind, ob das User Interface einer Webanwendung auf einem mobilen Endgerät, wie einem Smartphone oder Tablet, oder auf einem stationä(re)n Gerät zur Darstellung gebracht wird. Die Adressierung spezifischer Anforderungen mobiler Anwendungen hinsichtlich Funktionalität und User Interface, die für HTML(5) im allgemeinen festgestellt werden kann, lässt sich aber auch mit Blick auf die Ausdrucksmittel für Formulare konstatieren.



Lernziele

Nachdem Sie die Lerneinheit durchgearbeitet haben, sollten Sie in der Lage sein:

- Die Verwendung von Formularen in Bezug zu deren Nutzungszwecken zu setzen.
- Die grundlegenden Begriffe des Formularfelds, des Feldwerts etc. erklären zu können.
- Bei der konkreten Umsetzung eines Formulars Funktions- und Gestaltungsmerkmale zu berücksichtigen.
- Grundlegenden Konzepte zu erläutern, die für HTML-Formulare relevant sind.
- Die Verwendung von Formulardaten auf Ebene von JavaScript umzusetzen.
- Den Begriff Data Binding im Rahmen einer MVC-Architektur zu erklären.



Gliederung

Gliederung

- Überblick und Lernziele
- Funktionalität und Verwendung von Formulare
- HTML Formulare
- Zusammenfassung
- Wissensüberprüfung
- Übungen



Zeitbedarf

Zeitbedarf und Umfang

Für die Bearbeitung der Lerneinheit benötigen Sie etwa 4 Stunden. Für die Bearbeitung der Übungen für die Beispielanwendung etwa 3,5 Stunden und für die Wissensfragen ca. 1,5 Stunden.

1 Funktionalität und Verwendung von Formularen

Vermutlich haben Sie alle bereits im Rahmen von programmierbezogenen Lehrveranstaltungen User Interfaces implementiert, in denen Formulare eingesetzt wurden. Womit haben wir es aber eigentlich zu tun, wenn wir von „Formularen“ reden? Wozu dienen Formulare? Wenn Sie einen Blick auf folgende Abbildung werfen, dann werden sie dort bestimmt Formulare wiederfinden, mit denen Sie als Webnutzer, aber auch unabhängig von der Verwendung maschineller Nutzerschnittstellen bereits einmal in Berührung gekommen sind.

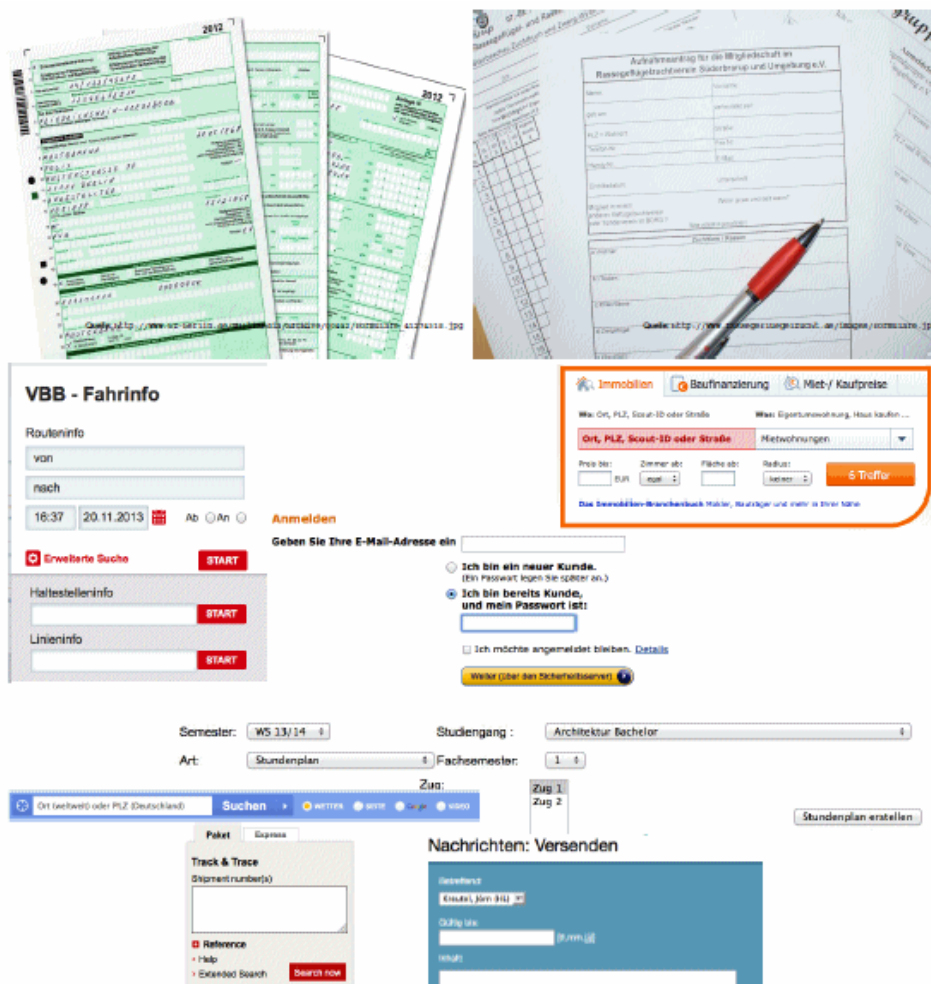


Abb.: Auswahl exemplarischer
papierner und elektronischer
Formulare

Im folgenden wollen wir versuchen, diese verschiedenen Ausprägungen von Formularen auf eine gemeinsame, differenzierte, Basis zurückzuführen, bevor wir uns daran anschließend wieder exklusiv mit Formularen als Bestandteil des User Interfaces von Web-Applikationen beschäftigen.

Formulare und
Anliegensformulierung

Im Gegensatz zu einer Spielanwendung oder einer Nachrichtenseite, auf die Sie vermutlich auch mit dem Ziel der Zerstreuung und Unterhaltung zugreifen, haben Sie vermutlich noch nie ein Formular „einfach so“ ausgefüllt. So kann denn die Nutzung eines Formulars immer auf ein Ziel oder Anliegen ihrerseits zurückgeführt werden, dessen Formulierung das Ausfüllen des Formulars dient.

So können die in eben gezeigter Abbildung dargestellten Formulare Sie z. B. dabei unterstützen, „Mitglied in einem Verein zu werden“, „eine Rückerstattung Ihrer Steuerzahlungen für einen bestimmten Zeitraum zu erhalten“ oder „ein ÖPNV-Abo in einem angegebenen Tarifbereich zu erwerben“. Die Unterstützungsleistung eines Formulars – im Ggs. z. B. zu einem leeren Blatt Papier oder einem großen Freitexteingabefeld – besteht dabei darin, dass ein Formular Ihnen eine strukturierte Formulierung Ihres Anliegens in Form elementarer Sachverhalte erlaubt z. B.:

- „mein Name/Geschlecht/Familienstand/Beruf ist...“
- „ich wohne in...“
- „ich züchte (Kaninchen und Meerschweinchen).“
- „mein Arbeitsweg beträgt...“

Zur Sinnhaftigkeit eines Formulars gehört außerdem immer auch ein *Empfänger*, an den das Formular zur Bearbeitung übermittelt werden soll. Dieser Empfänger kann wahlweise ein Sachbearbeiter im Finanzamt, der Vorstand eines Kleintierzüchtervereins oder eine Softwareanwendung sein, die entweder selbst zur automatischen Bearbeitung Ihres Formulars in der Lage ist oder dieses an einen dazu befähigten menschlichen (Sach)bearbeiter weiterleitet. So dient die strukturierte Beschreibung der Anliegen, die Ihnen ein Formular ermöglicht, nicht zuletzt der erleichterten Bearbeitung durch den betreffenden Empfänger – sei es ein Mensch oder eine Maschine.

Typen von Anliegen

Wenn wir uns die verschiedenen Anliegen, die durch ein Formular zum Ausdruck gebracht werden können, näher anschauen, dann lassen diese sich grob in zwei Gruppen unterteilen. Diese bezeichnen wir als *Informationsanliegen* zum einen und als *Transaktionsanliegen* zum anderen. Zu Informationsanliegen gehören z. B. Suchanfragen, für deren Formulierung Sie ein Formular verwenden. Sie zielen darauf ab, dass Sie Wissen bezüglich existierender Sachverhalte erlangen, z. B. dass „Sie wissen...“

- „... welche Bücher Sie derzeit ausgeliehen haben und was die Rückgabefristen sind.“
- „... welche Thai-Restaurants sich in Ihrer Umgebung befinden.“
- „... wann die nächste U-Bahn nach Hause fährt.“

Im Gegensatz dazu zielen Transaktionsanliegen auf die Herbeiführung noch nicht existierender Sachverhalte ab, z. B. „dass Sie...“

- „... Mitglied im... Verein sind.“
- „... eine Steuerrückerstattung in Höhe von... erhalten.“
- „... Inhaber eines ÖPNV-Abos für den Bereich... sind.“
- „... auf Ihrem Girokonto über... EUR vom Tagesgeldkonto verfügen können.“

Erwähnt sei, dass beide Arten von Anliegen in einem übergeordneten Anliegen begründet sein können, z. B. dem Anliegen „rasch einen Imbiss zu sich zu nehmen“, „so schnell wie möglich nach Hause zu kommen“, oder „zum Zweck eines Produkterwerbs Ihr Girokonto aufzufüllen“.

Anliegensmodifikation

Mitunter kann es auch erforderlich sein, eine durch ein ausgefülltes Formular ausgedrückte Anliegenformulierung zu modifizieren, z. B. in Fällen, in denen die Formulierung eines Informationsanliegens zu vage oder zu spezifisch ist. So ist es mit Blick auf die oben beschriebenen Anliegen z. B. denkbar, dass es zu einem angegebenen Zielbahnhof zahlreiche Verbindungen gibt, u. a. Direktverbindungen und Verbindungen mit Umstieg, dass es in Ihrer Umgebung zwar keine Thai-Restaurants gibt, aber andere Lokale mit asiatischer Küche, oder dass zum gegebenen Zeitpunkt zwar die U-Bahn nicht mehr fährt, aber eine Nachtbusverbindung mit Umsteigen existiert.

Betreffen die genannten Fälle – wie überhaupt die hier geschilderten Informationsanliegen – insbesondere Formulare, deren Bearbeitung maschinell und idealerweise „in Echtzeit“ erfolgt, so sind Probleme bei der Formulierung von Transaktionsanliegen durchaus auch für manuell bearbeitete Formulare üblich – diese äußern sich z. B. darin, dass Sie vom Bearbeiter Ihres Anliegens ein Schreiben erhalten, in dem Sie auf das Problem hingewiesen werden und nach Möglichkeit auch einen Weg zur Problembehebung aufgezeigt bekommen. So ist es für die Erfüllung von Transaktionsanliegen insbesondere erforderlich, fehlerhafte oder widersprüchliche Anliegenformulierungen zu vermeiden, z. B. kann die Bearbeitung eines Anliegens bestimmte Voraussetzungen erfordern wie die Volljährigkeit des Antragstellers, das Angebot eines Abos oder anderer Ermäßigungsangebote für einen bestimmten Bereich oder eine Reiseroute, oder die Verfügbarkeit eines gewünschten Überweisungsbetrags unter Berücksichtigung von Geschäftsregeln z. B. bezüglich Maximalsummen innerhalb eines gegebenen Zeitraums.

Falls Probleme wie die hier geschilderten auftreten, kann es erforderlich sein, Ihr Anliegen entweder zu modifizieren oder es aufzugeben, oder aber eine alternative Realisierung des ggf. vorhandenen übergeordneten Anliegens zu suchen – in den genannten Fällen z. B. durch Wahl eines alternativen Verkehrsmittels oder einer alternativen Zahlungsform.

Mit Fällen wie den hier genannten wurden Sie bestimmt schon in der einen oder anderen Weise bei der Verwendung von Formularen konfrontiert. Bei der Behandlung von allzu vagen oder restriktiven Anliegensformulierungen sowie bei der Gültigkeitsprüfung von Anliegensformulierungen handelt es sich um Anforderungen, die durch die maschinelle Bearbeitung von Formularen berücksichtigt werden sollten. Auch sollte Ihnen die Anwendung, in deren User Interface ein Formular verwendet wird, in solchen Fällen nach Möglichkeit eine sofortige Rückmeldung geben, und Ihnen ggf. eine Modifikation Ihrer Anliegensformulierung oder des Anliegens selbst ermöglichen.

Ziel jeglichen Formulars sollte es denn auch sein, die Nutzer des Formulars bei der effizienten und korrekten Formulierung ihrer Anliegen zum Zweck der erfolgreichen Anliegenserfüllung zu unterstützen. In unserer Betrachtung von Formularen sind wir damit bei spezifischen Eigenschaften von Formularen als Nutzerschnittstellen zur Anliegensformulierung angelangt, mit denen wir uns im folgenden Teilkapitel beschäftigen werden.

1.1 Formulare in GUIs

Formulare können aufgefasst werden als komplexe Bestandteile der Ansichten einer graphischen Nutzerschnittstelle, die dem Nutzer zur Formulierung von Anliegen zum einen eine Menge dafür geeigneter Eingabeelemente zur Verfügung stellen. Mittels letzterer können jeweils einzelne Teilaussagen, die zur Anliegensformulierung beitragen, ausgedrückt werden, z. B. „mein Name ist...“, „ich wohne in...“, etc. Entsprechend der Form der Teilaussage bzw. der Anzahl der Varianten, die für deren Formulierung möglich sind, können Eingabeelemente u. a. wahlweise als Freitexteingabeelemente, Auswahlelemente oder Binärauswahlelemente realisiert werden. Dass solche Elemente nicht auf GUIs als maschinelle Nutzerschnittstellen beschränkt sind, sondern sich auch in papiernen Formularen finden, zeigt folgende Abbildung.

Freitexteingabefelder für Daten unterschiedlicher Typen:

The image shows a form with several input fields. The first field is labeled 'Name'. Below it is a field labeled 'Ggf. Geburtsname'. Then there is a section for 'Telefon' which is divided into three sub-fields: 'Vorwahl international', 'Vorwahl national', and 'Rufnummer'. At the bottom is a field labeled 'E-Mail'.

Auswahlfelder für Daten unterschiedlicher Typen:

The image shows a form with a dropdown menu labeled 'Religion'. To the left of the dropdown, there is a list of options: 'Religionsschlüssel: Evangelisch= EV', 'Römisch-Katholisch= RK', and 'nicht kirchensteuerpflichtig= VD'.

Binärauswahlfelder:

The image shows a form with a binary choice question: 'Möchten Sie am Lastschriftinzugsverfahren, dem für beide Seiten einfachsten Zahlungsweg, teilnehmen?'. Below the question is a checkbox and the text 'Ja, die ausgefüllte Teilnahmeerklärung ist beigelegt.'

Abhängige Eingabefelder:

Waren Sie (oder ggf. Ihr Ehegatte) in den letzten drei Jahren für Zwecke der Einkommenssteuer steuerlich erfasst?

☐ Nein ☐ Ja

Finanzamt

Steuernummer

Werden in mehreren Gemeinden Betriebsstätten unterhalten? ☐ Nein

☐ Ja lfd. Nr.

Bezeichnung

Anschrift, Straße

Haus-Nr.

Haus-Nr. Zusatz

Postleitzahl

Ort

Steuernummer

Abb.: Beispiele für verschiedene Typen von Eingabeelementen in nicht elektronischen Formularen

Da ein Formular immer im Hinblick auf die Übermittlung an einen Empfänger ausgefüllt wird, müssen Formulare in GUIs notwendigerweise auch eine solche Übermittlung bzw. das „Absenden“ der Formulardaten an eine das Formular bearbeitende Anwendung ermöglichen. Üblicherweise wird hierfür ein Button (dt. „Schaltfläche“) verwendet. Als Analogie zu diesem Eingabeelement im Bereich nicht-elektronischer Formulare kann – wie der Begriff des „Absendens“ sehr schön ausdrückt – der Versand des Formulars per Brief, aber auch die direkte manuelle Übergabe an einen Sachbearbeiter oder ein hierfür vorgesehene Postfach angesehen werden.

Mit Blick auf die Bedeutung der Formularvalidierung für die Anliegenverfolgung eines Nutzers sollten nicht nur manuell bearbeitete Formulare, sondern auch Formulare in GUIs eine möglichst frühe Gültigkeitsprüfung der in ein Formular eingegebenen Daten vornehmen. Dies gilt nicht zuletzt in besonderer Weise für Formulare in mobilen Nutzerschnittstellen, da die diesbezüglich verfolgten Anliegen sich häufig durch eine besondere Dringlichkeit der Anliegenbefriedigung auszeichnen.

1.2 Formulare und Datentypen

„Formularfelder“,
„Feldwerte“,
„Formulardaten“

Zu den grundlegenden Begriffen, die wir im weiteren Verlauf dieser Lerneinheit verwenden werden und bezüglich derer wir eine klare Begriffsfassung für erforderlich halten, gehört der Begriff des Formularfelds. Dieser bezeichnet ein Attribut, bezüglich dessen ein Formular eine elementare Teilaussage ermöglicht, z. B. „Name“, „Wohnort“, „Betrag“. Der Begriff des Feldwerts bzw. des Formularfeldwerts bezeichnet hingegen einen variablen Informationsbestandteil, der in Verbindung mit einem Formularfeld eine elementare Teilaussage vornimmt, z. B. „Mein Name ist J.K.“, „Ich wohne in Berlin.“, „Ich möchte einen Betrag von 50 EUR überweisen“, „Ich möchte zum Leopoldplatz fahren“, etc. Unter „Formulardaten“ verstehen wir schließlich die Menge der durch die Feldwerte eines ausgefüllten Formulars ausgedrückten Teilaussagen. Diese ist üblicherweise als Konjunktion („Verundung“) zu lesen, z. B. wie folgt:



Hinweis

Konjunktion („Verundung“)

„Ich möchte zum Leopoldplatz fahren **und** ich fahre vom vom Alexanderplatz ab **und** ich fahre sofort ab.“

Formulardaten und
Datentypinstanzen

Mit dem Begriff des Attributs haben wir gerade einen Begriff verwendet, der Ihnen aus der Betrachtung komplexer Datentypen und deren Instanzen vertraut sein dürfte – nicht zuletzt aus der Behandlung von Objekten in JavaScript. Auch darüber hinaus besteht ein Zusammenhang zwischen Formularen und den Instanzen komplexer Datentypen, auf den wir Sie an dieser Stelle aufmerksam machen wollen und den Sie in folgender Abbildung illustriert finden.

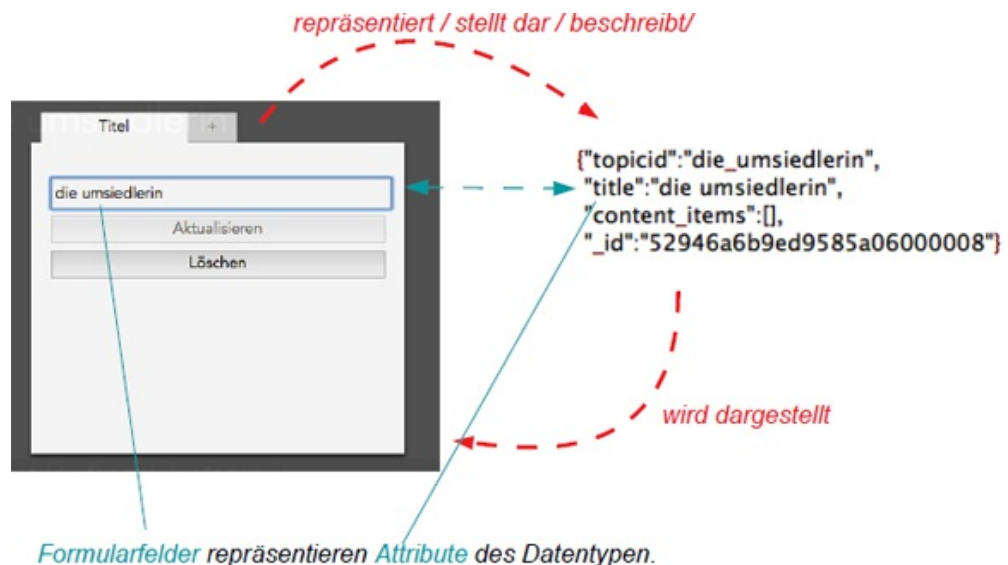


Abb.: Illustration des
Zusammenhangs von
Formulardaten und Instanzen
komplexer Datentypen


So können Formulardaten – im gerade definierten Sinne – als Beschreibung von *Instanzen komplexer Datentypen* angesehen werden. Mit Blick auf die hier verwendeten Beispiele sind dies z. B. „Rutenbeschreibungen“, „Überweisungsdaten“, „Produktmerkmale“, etc. Wenn Sie sich nun die oben vorgenommene Unterscheidung von Informationsanliegen und Transaktionsanliegen noch einmal vergegenwärtigen, fällt Ihnen evtl. auf, dass diese sich auch in ihrem jeweiligen Bezug zu den Instanzen des mit einem Formular assoziierten Datentyps unterscheiden.

Formulare bezüglich Informationsanliegen können nämlich grundsätzlich als Beschreibung einer Menge möglicher Instanzen eines Datentyps aus Sicht des Nutzers aufgefasst werden – und die Bearbeitung des Formulars dient dann dazu, die Menge der tatsächlichen Instanzen des betreffenden Datentyps zu ermitteln. Dies ist unabhängig davon, ob der Nutzer wirklich Information bezüglich einer Menge von Instanzen erlangen möchte wie bei einer Restaurantsuche oder bezüglich einer spezifischen Instanz wie üblicherweise beim Blick in ein „Telefonbuch“ – dieses Beispiel sei hier trotz unserer Fokussierung auf mobile Endgeräte erlaubt.

Im Gegensatz dazu beschreiben Formulare in Transaktionsanliegen genau eine mögliche Instanz eines Datentypen aus Sicht des Nutzers, und die Bearbeitung des Anliegens soll dazu dienen, aus dieser Instanzbeschreibung eine tatsächliche Instanz zu erzeugen – beispielsweise die Daten eines Überweisungsformulars, die immer noch als Ausdruck eines „Wunsches“ des Nutzers angesehen werden können, in eine tatsächliche Überweisung zu überführen. Denkbar ist aber auch die Kombination beider Anliegenstypen zum Zweck der Modifikation oder des Löschens bestehender Instanzen, die z. B. zunächst mittels eines Suchformulars beschrieben werden.

Informationsanliegen und Transaktionsanliegen lassen sich damit auch in Bezug setzen zu unterschiedlichen CRUD-Operationen, wie wir sie in der vorangegangenen Lerneinheit betrachtet haben. Während für die Erfüllung von Informationsanliegen lediglich ein lesender Zugriff auf eine Menge von Instanzen erforderlich, sind Transaktionsanliegen immer mit einem schreibenden Zugriff verbunden – mag er die Erzeugung, die Modifikation oder das Löschen von Instanzen zum Ziel haben. Zugespitzt ausgedrückt könnte man den Unterschied der beiden Anliegenstypen auch darin begreifen, dass bei der Bearbeitung von Informationsanliegen „die Welt bleibt, wie sie ist“ – abgesehen vom damit verbundenen Wissenszuwachs auf Seiten des Nutzers – während die Bearbeitung eines Transaktionsanliegens immer eine Veränderung der Welt herbeiführt.

Begriff „Welt“

Ohne an dieser Stelle zu sehr abzuschweifen, sei noch erwähnt, dass die hier vorgenommene Differenzierung voraussetzt, dass unter Welt die Menge aller durch Instanzen komplexer Datentypen ausgedrückten Sachverhalte verstanden wird. Siehe dazu auch  [\[Wit89\]](#)

Der letztgenannte Aspekt unterstreicht noch einmal die Relevanz der Validitätsprüfung bezüglich der Bearbeitung von Formularen für Transaktionsanliegen. So hat diese nichts weniger zum Ziel als zu verhindern, dass ein „inkonsistenter Weltzustand“ eintritt, d. h. dass etwas der Fall ist, was entsprechend den geltenden Gültigkeitskriterien „nicht sein kann“ oder „nicht sein darf“. Mit dieser Überlegung wollen wir nun aber endgültig die Sphäre der Theorie verlassen und uns endgültig den praktischen Aspekten von Formularen zuwenden.


1.3 Praktische Aspekte von Formularen

Vermutlich fragen Sie sich – verständlicherweise – worin die Relevanz der in den vorangegangenen Abschnitten dargelegten theoretischen Überlegungen zu Formularen liegt. Diese ist dadurch gegeben, dass wir ein Verständnis der Theorie von Formularen benötigen, um „zu wissen, was wir tun“, wenn wir Formulare für mobile und andere Nutzerschnittstellen entwickeln. Im Kern beinhaltet diese Tätigkeit, dass wir mit einem Formular den Nutzern einer Nutzerschnittstelle die Formulierung von Anliegen ermöglichen. Daraus resultieren zum einen Anforderungen bezüglich der Bedienbarkeit von Formularen, welche ihrerseits Anforderungen an die softwarearchitektonische und implementatorische Umsetzung von Formularen zur Folge haben.

Bedienbarkeit von Formularen

Aus den vorstehend dargelegten konzeptuellen Überlegungen zu Formularen lassen sich die folgenden Anforderungen bezüglich der Umsetzung nutzungstauglicher Formulare ableiten:

- **Einfachheit**
Nutzer sollen ohne Verständnisschwierigkeiten ein Formular ausfüllen können.
- **Effizienz**
Die Anzahl an elementaren Nutzeraktionen (Klicks / Tastaturanschlägen) zum Ausfüllen soll möglichst gering sein.
- **Zielgerichtetheit**
Die Gestaltung des Formulars soll die korrekte Eingabe von in ihrem Zusammenhang korrekten Formulardaten vereinfachen.
- **Frühe Fehlerdetektion**
Fehler bei der Eingabe sollen so früh wie möglich erkannt werden.
- **Kooperative Fehlerbehandlung**
Falls ein Fehler auftritt, soll der Nutzer darauf hingewiesen werden, wie der Fehler behoben werden kann.
- **Flexibilität**
Formulare sollen verschiedene Ausprägungstypen der formulierbaren Anliegen ermöglichen (z. B. Oneway vs. Return Option bei Routenplanungen).

Angemerkt sei, dass diese Merkmale auch alternativ aus den Grundsätzen der Dialoggestaltung von Mensch-Maschine-Schnittstellen gemäß  DIN ISO 9241 - 10 angeleitet werden können. Zu ihrer Einlösung ist es indes erforderlich, dass bei der konkreten Umsetzung die folgenden Funktions- und Gestaltungsmerkmale berücksichtigt werden:

Umsetzung von Formularen in
GUIs


- Kennzeichnung der Formularfelder mittels aussagekräftiger, verständlicher und lesbarer Bezeichnungen
- Freitexteingabefelder mit typspezifischer Eingabevalidierung und ggf. spezifischem Tastaturlayout für mobile GUIs
- Berücksichtigung notwendiger vs. optionaler bzw. alternativ auszufüllender Felder
- Behandlung abhängiger Felder, d. h. Felder, deren notwendige / optionale Eingabe von Eingaben anderer Felder abhängt, z. B. von Binärauswahlfeldern
- Dynamische Bereitstellung von Inhalten für abhängige Auswahlfelder, z. B. um nur gültige Optionen zuzulassen.
- Sofortige Validierung von Eingaben unabhängiger Felder
- Frühst mögliche Validierung von Eingaben bezüglich mehrerer Felder, die zusammenhängend validiert werden müssen.

Eingabedetektion

Daraus resultieren jedoch verschiedene Anforderungen an die Implementierung von Formularen, die wir nachfolgend noch ohne spezifische Bezugnahme auf die Ausdrucksmittel von Web Technologien generalisierend beschreiben. So muss jeweils im Rahmen der für die Entwicklung verwendeten Programmierschnittstelle eine Möglichkeit bereitstehen, um ggf. auf den Vollzug einer Nutzereingabe, in jedem Fall aber auf deren Abschluss zu reagieren. Üblicherweise – und so denn auch in JavaScript bzw. durch die Spezifikation von DOM Events – werden hierfür durch die API verschiedene Eingabeereignisse definiert, bezüglich derer eine Anwendung Event Handler zur Reaktion auf das betreffende Ereignis deklarieren kann.

Data Binding

Aus dem dargelegten Zusammenhang von Formulardaten und Instanzen komplexer Datentypen lässt sich bereits grundsätzlich die Frage ableiten, welche Ausdrucksmittel verwendet werden können, um diesen Zusammenhang in den beiden in vorherigen Abbildung „Illustration des Zusammenhangs von Formulardaten und Instanzen komplexer Datentypen“ dargestellten Richtungen implementatorisch abzubilden. Auch für die Umsetzung von Validierungsoperationen ist es relevant, auf welchem Weg die eingegebenen und zu validierenden Formulardaten dafür zur Verfügung gestellt werden.

So kann es z. B. mangels ausgefeilterer Ausdrucksmittel erforderlich sein, dass die betreffenden Eingabefelder manuell durch den Entwickler ausgelesen werden müssen. In der umgekehrten Richtung ist es hingegen erforderlich, dass die Daten darzustellender und ggf. zu manipulierender Instanzen in die Felder eines Formulars übertragen werden. Auch hierfür sind verschiedene Lösungen denkbar, welche anfangen von der „manuellen“ Übertragung der Objektattribute in die Felder eines Formulars bis hin zu einer deklarativen Assoziation von Eingabefeldern und Attributen reichen, wie sie z. B. im Rahmen von  Java Server Faces konzipiert wird. Als Bezeichnung für den Abgleich von Formulardaten und Instanzen von Datentypen, der in den hier beschriebenen beiden Richtungen erfolgen kann, verwenden wir nachfolgend den hierfür gebräuchlichen Begriff des Data Binding.

Validierung


Hinsichtlich der Durchführung von Validierungsmaßnahmen, deren Relevanz für die Nutzbarkeit von Formularen wir vorstehend hervorgehoben haben, stellt sich für die Umsetzung zum einen die Frage, in welcher Form generische und anwendungsspezifische Validierungsmaßnahmen beschrieben werden können. Auch hier sind verschiedene Lösungen im Bereich zwischen rein imperativen vs. deklarativen Ausdrucksmitteln denkbar. Ferner stellt sich bezüglich der Validierung die Frage, ob diese nur clientseitig, nur serverseitig oder auf beiden Seiten durchgeführt werden soll und wie bei beidseitiger Durchführung redundante Entwicklungsaufwände vermieden werden können.

Formulare und MVC

Bereits die hier dargestellten Aspekte machen deutlich, dass bei der Umsetzung von Formularen alle Aspekte zum Tragen kommen, die wir in der Lerneinheit CSS im Hinblick auf die Umsetzung einer MVC-Architektur als relevant beschrieben hatten. Dies gilt nicht zuletzt auch für die Realisierung von Feedback, mittels dessen dem Nutzer der Vollzug oder der Abschluss einer Eingabeverarbeitung und deren Resultat kommuniziert werden.

HTML

Betrachten wir vorgreifend auf die weiteren Ausführungen die aktuellen Ausdrucksmittel von HTML, dann sehen wir, dass hier deklarative Ausdrucksmittel für zahlreiche der formularbezogenen Anforderungen angeboten werden. Dies betrifft zum einen die Detektion von Nutzereingaben durch DOM Events im allgemeinen, aber auch die spezifischeren Aspekte von Formularen. Es existieren z. B. verschiedene Alternativen zur Realisierung von Freitext-, Auswahl- und Binärauswahlelementen, die Möglichkeit einer typspezifischen Eingabevalidierung, sowie deklarative Ausdrucksmittel zur Behandlung obligatorischer vs. optionaler Formularfelder.

Andere Anforderungen müssen jedoch weitgehend durch imperative Programmierung in JavaScript umgesetzt werden, darunter nicht nur die Realisierung anwendungsspezifischer Validierungsprüfungen und die Behandlung von Abhängigkeiten zwischen Feldern bezüglich deren Erfordernis und Validierung. So müssen insbesondere auch beide Richtungen des Data Binding mittels JavaScript imperativ programmiert werden. Angemerkt sei jedoch, dass gerade das letztere Defizit der standardisierten Ausdrucksmittel für Formulare durch MVC Frameworks wie  AngularJS kompensiert wird, für welche zumindest denkbar ist, dass sie sich in Richtung von „de facto Standards“ entwickeln werden.

2 HTML Formulare

Wir werden nun einen Überblick über die aktuellen durch HTML bereitgestellten Ausdrucksmittel zur Realisierung von Formularen geben. Auch hier verfolgen wir nicht die Absicht, Ihnen eine erschöpfende Übersicht aller Ausdrucksmittel darzustellen. Es geht uns vielmehr darum, Ihnen anhand einer Auswahl zum einen eine Grundlage für die Bearbeitung der Übungsaufgaben bereitzustellen. Anhand dessen wollen wir Ihnen zum anderen einen Einblick in die grundlegenden Konzepte verschaffen, die für HTML-Formulare relevant sind und die auch bei zunehmender Erweiterung der konkreten Ausdrucksmittel „Web 1.0“ Bestand haben dürften.

„Web 1.0“ Formulare

Die Behandlung von Formularen in HTML wirft aber auch Fragen auf, die nur unter Bezugnahme auf den ursprünglichen in den 90er Jahren entwickelten und mittlerweile weitgehend obsoleten Ansatz bezüglich der Architektur von Webanwendungen zu klären sind. Betrachten wir dafür zunächst einmal ein sehr einfaches Formular wie das folgende, in dem bereits die wichtigsten Ausdrucksmittel für Formulare verwendet werden:



Quellcode

Ausdrucksmittel für Formulare

```
001 <form name="titleForm" action="/http2mdb/topicviews" method="post">
002 <input name="title"/>
003 <input type="submit" value="Erzeugen" />
004 </form>
```

Wir sehen hier, dass ein Formular aufgebaut wird aus einem Wurzelement `<form>`, als dessen Kinder `<input>` Elemente verwendet werden. Diese erfüllen zumindest zweierlei Funktion, wie wir bei einem Blick auf die Darstellung des Formulars sehen. Das Styling entnehmen wir hier und im folgenden den Implementierungsbeispielen, und wir verweisen auf Lerneinheit MFM Kapitel 2 „Gestaltung von Formularen“, wo wir uns detaillierter mit der Gestaltung von Formularen beschäftigen:

Abb.: Beispiel für einfaches Formular

Sie sehen hier, dass `<input>` Elemente zum einen verwendet werden, um Eingabemöglichkeiten für Formularfeldwerte bereitzustellen, wobei der Name des Feldes durch das `name` Attribut des Elements bezeichnet wird. Beachten Sie aber, dass dieses Attribut nur intern verwendet wird und nicht auf der Oberfläche des Formulars angezeigt wird. Zum anderen dient `<input>` auch dazu, ein Eingabeelement zum Absenden der Formulardaten darzustellen. Zu diesem Zweck muss ein spezieller Wert – `submit` – als Wert des `type` Attributs von `<input>` angegeben werden. Mittels `<form>` und einer Menge von `<input>` Elementen kann also die Grundfunktionalität von Formularen – die Eingabe von Formulardaten und deren Übermittlung an die Formularbearbeitung – bereits erbracht werden.

Betrachten wir nun die beiden Attribute `<action>` und `<method>`, die im Beispiel oben auf `<form>` gesetzt sind. Diese können wir verwenden, um einen HTTP Request zu beschreiben, welcher bei Betätigung des `submit` Elements durch den Browser aufgebaut und an den Server übermittelt wird, von welchem das dargestellte HTML-Dokument geladen wurde. Die beiden Attribute geben hierfür an, bezüglich welcher URL bzw. mit welcher HTTP-Methode dieser Request ausgeführt werden soll. In Abhängigkeit vom Wert von `method` – möglich sind hier nur `GET` oder `POST` – erfolgt die Übertragung der Formulardaten dann in der folgenden Form als URL-kodierter String entweder als URL Query Parameter eines `GET` Requests oder im Body eines `POST` Requests:

```
title=Die+Umsiedlerin&attr2=...&...
```

Wir sehen hier also bereits eine sehr einfache Möglichkeit, um mittels Formularen CRUD-Operationen via HTTP auf eine serverseitig laufenden Anwendung auszulösen. Vielleicht fragen Sie sich aber angesichts des Formular-Markups, wie wir in diesem Fall auf den uns vom Server übermittelten Rückgabewert reagieren können. So nehmen wir ja z. B. an, dass ein `POST` Request bezüglich einer Ressource wie `/topicviews` zur Erstellung einer Instanz von `Topicview` mit den übermittelten Formulardaten verwendet wird und dass wir als Rückgabewert die erstellte Instanz als JSON-Objekt mit dem durch die verwendete Datenbank gesetzten eindeutigen Identifikator erhalten.

Tatsächlich aber würde uns bei Ausführung der `submit` Aktion im vorliegende Fall durch den Server zwar evtl. ein JSON-Objekt zurück übermittelt. Der Browser würde jedoch versuchen, dieses Objekt anstelle des HTML-Dokuments darzustellen, in welchem das Formular enthalten ist.

Diese Kontrast zwischen unserer Erwartungshaltung und der tatsächlichen Ausführung unseres Formulars bezeichnet den Gegensatz zwischen einer „Web 2.0“ Auffassung von Formularen, wie sie auch der von uns verwendeten Architektur von Webanwendungen zugrunde liegt, und der ursprünglichen Konzeption von Formularen durch den HTML-Standard. Letztere nahm nämlich an, dass als Rückgabewert für die Bearbeitung von Formularen nicht etwa ein „rohes“ JSON-Objekt an den Browser übermittelt wird, sondern ein ggf. dynamisch generiertes HTML-Dokument.

Dieses sollte die auf die erfolgreiche Bearbeitung des Formulars hin darzustellende Folgeansicht enthalten, oder aber im Fall von Fehlern bei der Gültigkeitsprüfung das Formular inklusive der ursprünglich eingegeben Daten mit einer entsprechenden Fehlermeldung versehen. Anstelle einer partiellen clientseitigen Aktualisierung der dargestellten Ansicht auf Grundlage des übermittelten HTTP-Responses hätte hierfür das der darzustellenden Ansicht zugrunde liegende HTML-Dokument serverseitig aufgebaut und vom Browser dargestellt werden müssen.

Dass letztere Lösung nicht vereinbar ist mit einem Architekturansatz, bei dem jegliche Entscheidung bezüglich Übergangs zwischen Ansichten wie auch bezüglich ihres ggf. dynamischen Aufbaus clientseitig erfolgt, dürfte offensichtlich sein. Zu beachten ist jedoch auch, dass im Ggs. zu den clientseitigen Implementierungsmaßnahmen, die wir für die Behandlung von Formularen im weiteren Verlauf der Lerneinheit darstellen werden, Formulare in ihrer ursprünglichen Konzeption ggf. ohne jegliche Implementierung von JavaScript Code realisiert werden konnten.

Einen Hinweis darauf liefern nach wie vor die beiden Attribute `action` und `method`, wie auch die Tatsache, dass die „manuelle“ Ausführung von CRUD-Operationen in JavaScript als Reaktion auf die Betätigung eines `submit` Elements in ihrer konkreten Umsetzung und im Hinblick auf die Verarbeitung des Ereignisses durch den Browser immer noch Züge eines „Workarounds“ trägt.

2.1 Aufbau von Formularen

Wie einführend beschrieben werden Formulare durch das Element `<form>` markiert und können hier mit einer `id` und/oder einem `name` Attribut versehen werden, unter welchem sie in JavaScript identifizierbar sind.

Die möglichen Kind-Elemente von Formularen sind nicht auf `<input>` oder andere auf die Formularfunktionalität bezogene Elemente beschränkt. So kann eine große Anzahl der Dokumentstrukturelemente von HTML genutzt werden, um die visuelle Gestaltung eines Formulars zu realisieren, z. B. könnte die oft gewünschte gleichmäßige Anordnung von Formularfeldbezeichnungen und den betreffenden Eingabeelementen auch mittels eines `table` Elements als Tabelle erfolgen.



Hinweis

Ausgeschlossen ist jedoch die Einbettung von Formularen in Formulare.

Im folgenden werden wir auch mit Blick auf unsere einleitenden Ausführungen zunächst die Umsetzung von Eingabeelementen in Formularen mit einem Fokus auf Auswahl Elemente vorstellen und im Anschluss daran aufzeigen, wie die Übermittlung von Formulardaten auf Ebene von JavaScript initiiert werden kann. Für eine vollständige Darstellung der Ausdrucksmittel für Formulare verweisen wir auf die Ausführungen im HTML Living Standard.

<http://www.whatwg.org/>

2.1.1 Eingabeelemente

`input` Element

Wie wir bereits gesehen haben ist das wichtigste HTML-Element für die Realisierung von Eingabeelementen für Formularfeldwerte das Element `<input>`. Dessen `name` Attribut bezeichnet – wie das `name` Attribut aller anderen Eingabeelemente – den Namen des Feldes, unter dem dieses auch aus JavaScript zugegriffen werden kann. Der Wert von `name` muss innerhalb eines Formulars eindeutig sein, kann aber in verschiedenen voneinander getrennten Formularen innerhalb eines HTML-Dokuments wiederverwendet werden.

`type` Attribut

Wir haben erwähnt, dass in HTML alle bereits genannten Typen von Formularfeldern in einer oder mehreren Ausprägungen unterstützt werden. Dafür wird vor allem das `type` Attribut von `<input>` verwendet. Dessen besondere Funktion zur Kennzeichnung der Absende-Aktion bezüglich der eingegebenen Formulardaten haben wir bereits kennengelernt. Das Attribut kann aber auch verwendet werden, um den Datentyp - der durch ein Eingabeelement bereitzustellenden Werte - zu kennzeichnen. Beispielsweise können `<input>` Elemente u. a. bezüglich der Typen `text`, `url`, `email`, `datetime`, `number`, `range` oder `file` deklariert werden, aber auch der Typ `color` wird durch den HTML-Standard definiert.

Bezüglich dieser Typen obliegt es dem Browser, einem Nutzer für die Eingabe von Werten spezifische Bedienelemente zur Eingabe bereitzustellen, z. B. eine Kalenderansicht für die Eingabe von Werten bezüglich eines `datetime` Inputs. Mobile Browser verwenden hierfür üblicherweise die existierenden nativen Bedienelemente oder bieten dem Nutzer – z. B. für die Eingabe von Zahlenwerten, Mail-Adressen oder URLs ein spezifisches Tastaturlayout an, das die verwendbaren bzw. die notwendigerweise erforderlichen Zeichen, z. B. „@“ ohne Umschaltnotwendigkeit eingebbar macht.

Bei Angabe des Typs `file` wird dem Nutzer üblicherweise ein Dateiauswahldialog angeboten. Liegt keine spezifische Browser-Unterstützung für einen Wert von `type` vor, dann wird das Element per Default als Freitexteingabefeld realisiert. Der Wert `password` schließlich schränkt zwar die Eingabemöglichkeiten nicht ein, bewirkt aber, dass eingegebene Werte verschleiert, d. h. üblicherweise „ausgepunktet“ werden. Letzteres erfolgt in mobilen Browsern üblicherweise mit einer gewissen Verzögerung pro eingegebenem Zeichen, um dem Nutzer zumindest ein kurzes Feedback bezüglich seiner Eingabe zu geben und so ggf. eine Korrektur zu ermöglichen.

Wie wir später sehen werden, werden die Werte von `type` nicht nur für die ggf. durch spezifische

Elemente erleichtere Eingabe von Werten verwendet, sondern auch für die Validierung eingegebener Werte herangezogen – so ist letzteres ja gerade für den Fall erforderlich, dass keine spezifischen Eingabemöglichkeiten bereitstehen oder der Nutzer sich z. B. über ein angebotenes Tastaturlayout hinwegsetzt. Im folgenden Abschnitt werden wir außerdem zeigen, dass sich mittels entsprechender Werte für `type` auch ggf. typisierte Freitexteingabelemente von Auswahlelementen unterscheiden lassen.

`<textarea>`

Eine Einschränkung von `<input>` bezüglich der Eingabe von Freitext besteht insofern, als die Eingabe von Text mit Zeilenumbrüchen darin nicht möglich ist. Hierfür kann das `<textarea>` Element verwendet werden, das mit Größeneinstellungen für die Höhe, d. h. die Zeilenanzahl (`rows`) und Breite (`columns`), d. h. die Anzahl an Zeichen in einer Zeile versehen werden kann:

```
<textarea name="txt" cols="40" rows="3"></textarea>
```

Diese Angaben stellen jedoch insofern nur eine „Richtgröße“ für den Browser dar, als sie durch Stylezuweisungen – z. B. `width` – modifiziert werden können. Desktopbrowser erlauben es üblicherweise auch, die eingestellte Größe durch Dragging zu verändern.

Mit Blick auf unsere Anforderungen zur Bedienbarkeit von Formularen möchten wir auf weitere Attribute hinweisen, die auf `<input>` und anderen Eingabeelementen wie `<textarea>` sowie auf dem unten beschriebenen `<select>` Element gesetzt werden können:

- Das boolesche Attribut `required` markiert ein Formularfeld als notwendig, d. h. ohne Eingabe eines Werts soll keine Formularverarbeitung durchgeführt werden. Auf diese Weise können bereits im Zuge der Verarbeitung der Formulardaten auf Clientseite unvollständige Formulardaten detektiert werden, ohne dass hierfür ein spezifischer Implementierungsaufwand erforderlich wäre.
- `autofocus` ist ebenfalls ein boolesches Attribut und setzt den Textcursor bei Darstellung des Formulars auf das markierte Feld, d. h. der Nutzer kann sofort die Eingabe tätigen und braucht das Feld nicht auszuwählen.
- `autocomplete= „off“` deaktiviert die Autovervollständigung, die Browser oft per Default aktiviert haben, die aber z. B. für die Eingabe alphanumerischer Codes als störend empfunden werden dürfte.
- `hidden` markiert als boolesches Attribut „unsichtbare“ Formularfelder, deren Werte für die Verarbeitung der Formulardaten zwar verwendet werden sollen, die der Nutzer aber nicht zu Gesicht bekommen soll. Falls eine Shopanwendung beispielsweise einen Identifikator für einen Warenkorb nutzt, könnte dieser beim Hinzufügen von Produkten z. B. mittels eines `hidden` Inputs den vom Nutzer eingegebenen Formulardaten hinzugefügt werden.

Feldnamen

Bisher haben wir uns nur mit den Möglichkeiten zur Bereitstellung von Formularfeldwerten durch den Nutzer beschäftigt. Mit Blick auf die Realisierung von Formularen als Bestandteile eines User Interfaces ist jedoch auch jeweils zu erwägen, wie die Darstellung des Namens eines Formularfelds umgesetzt werden soll. Oben haben wir erwähnt, dass ein ausgefülltes Formular als eine Menge von „verundeten“ elementaren Aussagen aufgefasst werden kann.

Für die maschinelle Bearbeitung des Formulars ist die „Lesbarkeit“ der Formulardaten in diesem Sinne denn auch durch die eindeutige Benennung von Eingabeelementen mittels `name` gewährleistet. Die Lesbarkeit ist jedoch auch aus Sicht des Nutzers relevant, da dieser wissen möchte, welche Aussagen er gerade an eine Anwendung übermittelt oder bezüglich welcher Aussagen im Fehlerfall Änderungen erforderlich sind. Hierfür muss dem Nutzer entweder der Aufbau des Formulars bekannt sein. z. B. zeichnen sich Login-Formulare oder Formulare zur Adresseingabe durch einen weitgehend konventionalisierten Aufbau aus, im Hinblick auf welchen eine Menge eingegebener Werte auch ohne Beschriftung der Eingabefelder selbsterklärend ist. Ist letzteres jedoch nicht gewährleistet, ist es im Hinblick auf die Bedienbarkeit von Formularen erforderlich, zusätzlich zu den Eingabeelementen die Bedeutung der eingegebenen Werte z. B. durch die gleichzeitig sichtbare Darstellung der Namen der Formularfelder transparent zu machen.

Besonders mit Blick auf die Platzeinschränkungen, die mobile Anwendungen zumindest auf Smartphones zu berücksichtigen haben, stellt die Darstellung der Formularfeldnamen eine

Herausforderung für die UI-Gestaltung dar. So erscheint denn hier zunächst die Verwendung eines Ausdrucksmittels naheliegend, das in Anlehnung an native Anwendungen zum aktuellen Funktionsumfang von HTML gehört und das es erlaubt, einen „Placeholder Text“ anzugeben. Wie im unten gezeigten Beispiel kann dieser durchaus auch für die Darstellung des Feldnamens verwendet werden:

```
<textarea name="txt" cols="40" rows="3" placeholder="Einführungstext">
</ textarea>
```

Die Verwendung des Attributs resultiert dann in der nachfolgend links gezeigten Darstellung, die bei tatsächlicher Eingabe eines Texts durch den Nutzer aber durch die rechts gezeigte Ansicht ersetzt wird:

Abb.: placeholder-Attribut

D. h. der Placeholder kann zwar durchaus als „Hinweis“ auf die einzugebenden Werte genutzt werden – so wird hierfür in Android XML Layouts auch das Attribut `hint` angeboten. Sobald ein Wert jedoch eingegeben worden ist, verschwindet der Text. Mit Blick auf die Selbsterklärungsfähigkeit eines *ausgefüllten* Formulars erscheint `placeholder` daher gerade nicht geeignet (Siehe dazu auch www.pardot.com/faqs/best-practices/placeholders-and-labels/).

Die Darstellung der Feldnamen erscheint daher gerade auch für mobile Anwendungen grundsätzlich erforderlich, und entsprechend existieren, wie in folgender Abbildung für Android und iOS jeweils Vorschläge, wie Formulare in dieser Hinsicht gestaltet werden können.

Abb.: Alternativen zur Realisierung von Formularfeldern und Eingabeelementen in Android (links) und iOS (rechts)

`<label>`

Für die explizite Zuordnung von „Feldbeschriftungen“ zu Eingabeelementen auf Ebene des Markups sieht HTML ein Element namens `<label>` vor. In dessen `for` Attribut kann die id eines Eingabeelements angegeben werden, auf welches sich das `<label>` Element bezieht. Damit kann dann z. B. auf Ebene von CSS eine Formatierung entsprechend den gewünschten Gestaltungsrichtlinien vorgenommen werden.

`<fieldset>` und `<legend>`

Eine Alternative stellt die Verwendung der Elemente `<fieldset>` und `<legend>` dar, die ursprünglich – wie der Namen sagt – zur Gruppierung mehrerer Felder in Formularen verwendet werden können – denken Sie z. B. an eine Reisebuchung, wo Angaben zur *Route*, zur *Hinfahrt* sowie zur *Rückfahrt* in einer Formulardarstellung jeweils gruppiert werden können. Wie das folgende Beispiel zeigt, können die Elemente aber auch verwendet werden, um die Beschriftung einzelner Felder durchzuführen:



Quellcode

```

<fieldset>
001 <form>
002 <fieldset>
003   <legend>Bild</legend>
004   <output id="detailview-img">
005   </output>
006   <button type="button">Aufnehmen</button>
007 </fieldset>
008 <fieldset>
009   <legend>Titel</legend>
010   <input type="text" placeholder="..." autofocus="autofocus" required="
011         required"/>
012 </fieldset>
013 </form>

```

Mittels Zuweisung geeigneter Class-Attribute, die wir hier aus Gründen der Übersichtlichkeit entfernt haben, lässt sich das vorstehend gezeigte Formular dann z. B. wie folgt realisieren:

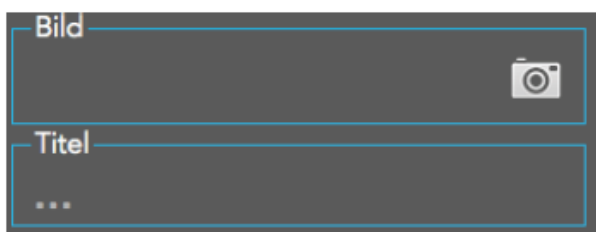


Abb.: Formular mittels Class-Attribute

Das im Beispiel verwendete Element `<output>` dient dazu, im Markup die Orte zu markieren, an dem das Ergebnis einer durch eine Nutzereingabe veranlassten Datenverarbeitung ausgegeben wird – im vorliegenden Fall kann der Nutzer z. B. ein Bild aufnehmen, welches an der markierten Stelle eingefügt wird. Falls der in `output` darzustellende Wert von den Werten anderer Formularfelder abhängt – z. B. wenn er aus diesen automatisch ermittelt werden kann – kann dies durch Angabe der `id` Werte der betreffenden Felder in einem `for` Attribut auf `<output>` markiert werden.

Auf das im Beispiel ebenfalls genutzte `<button>` Element und dessen Abgrenzung zu einem als `<submit>` markierten `<input>` Element werden wir unten im Zusammenhang mit der Verarbeitung von Formulardaten noch einmal eingehen.

2.1.2 Auswahlelemente

Checkbox

Auswahlelemente lassen sich in HTML auf verschiedene Weisen realisieren. Beispielsweise lässt sich eine einfache Binärauswahl mittels einer Checkbox umsetzen, für welche ihrerseits ein entsprechender `type` Wert für `<input>` bereit steht:

```
<input name="fieldname" type="checkbox"/>
```

Für die Darstellung der beiden Auswahlzustände werden dann jeweils browserspezifische Bilddateien verwendet, z. B. wie folgt in Firefox:



Abb.: Checkbox in Firefox

Radiobuttons

Hingewiesen sei darauf, dass auch für Checkboxes eine geeignete Lösung für die Beschriftung des Eingabeelements erforderlich ist. Findet sich für die Benennung eines Binärauswahlfelds allerdings keine Formulierung, die beide durch die Checkbox ausdrückbaren Alternativen gleichermaßen kurz und aussagekräftig zum Ausdruck bringt, dann können alternativ auch *Radiobuttons* verwendet werden. Diese sind zudem nicht auf eine Binärauswahl eingeschränkt, sondern können auch für die Auswahl aus mehr als zwei Alternativen verwendet werden, wie das nachfolgende Beispiel-Markup und dessen Realisierung zeigt:



Quellcode

Radiobuttons

```

001 <fieldset class="radiogroup">
002   <input type="radio" name="contentMode" value="href" id="contentMode_txt"/>
003   <label for="contentMode_txt">Text eingeben</label>
004   <input type="radio" name="contentMode" value="upload" id="
005     contentMode_upload"/>
006   <label for="contentMode_upload">Datei hochladen</label>
007   <input type="radio" name="contentMode" value="formdata" id="
008     contentMode_formdata"/>
009   <label for="contentMode_formdata">Datei mit FormData hochladen</label>
010 </fieldset>

```

Die Einbettung der Buttons in ein `fieldset` Element wird hier mit Blick auf die Gestaltung vorgenommen, ist aber für die Funktionsfähigkeit des Formulars nicht erforderlich:

Abb.: Radiobuttons



Notwendig ist für die gewünschte Funktionsweise die Vergabe eines gemeinsamen `name` Werts – im vorliegenden Beispiel `contentMode` – an die zusammengehörenden `radio` Elemente. Auf dieser Grundlage nämlich ändert der Browser ohne weiteres Zutun unsererseits die dargestellte Auswahl für den Fall dass der Nutzer eine bereits ausgewählte Option ändert. Wir brauchen uns also nicht darum zu kümmern, das nicht mehr ausgewählte Element zurückzusetzen.

Beachten Sie bezüglich der Verwendung des `<input>` Elements außerdem das Attribut `value`. Dieses gibt den Wert an, auf welchen das Formularfeld für die gegebene Auswahl gesetzt wird und der ggf. bei Absenden des Formulars an den Server übermittelt wird. Aus diesem Verhalten ergibt sich aber auch die Konsequenz, dass Radiobuttons für die Umsetzung einer Mehrfachauswahl nicht geeignet sind. Hierfür könnte entweder eine Menge von Checkboxes – eine Checkbox pro Option – oder aber, wie nachfolgend beschrieben, ein `<select>` Element mit Mehrfachauswahl verwendet werden.

<select>

Das für HTML-Formulare verwendbare `<select>` Element ist insbesondere auch dann geeignet, wenn eine Auswahl eine variable Anzahl an Alternativen umfassen soll oder wenn die Menge der Alternativen zu groß ist, um durch Checkboxes oder Radiobuttons abgebildet zu werden. Das Element wird zunächst – wie alle anderen Eingabebelemente – mit dem Namen des Formularfelds markiert, dessen Werte es auswählbar machen soll. Die Menge der auszuwählenden Optionen kann dann mittels `<option>` Tochterelementen angegeben werden. Diese enthalten jeweils den zur Darstellung zu verwendenden Text als Kindelement. Zusätzlich kann ihnen durch Vergabe eines `value` Attributs ein Wert zugewiesen werden, der wie für Radiobuttons bei Auswahl der betreffenden Option als Wert des Feldes gesetzt wird. Mittels der booleschen Attribute `disabled` und `selected` lässt sich eine Option wahlweise als „darzustellen, aber nicht auswählbar“ bzw. als vor-ausgewählt markieren.

Es sei angemerkt, dass insbesondere das `disabled` Attribut üblicherweise nicht wie im folgenden Beispiel im statischen Markup, sondern dynamisch gesetzt werden dürfte – im folgenden Beispiel könnte dies z. B. auf Grundlage dessen erfolgen, dass für einen `Topicview` ein Element des ausgewählten Typs bereits existiert:



Quellcode

elementType

```

001 <select name="elementType">
002   <option value="zeitdokumente">Zeitdokumente</option>
003   <option value="einfuehrungstext">Einführungstext</option>
004   <option value="textauszug" disabled="disabled">Textauszug</option>
005   <option value="objekt">Objekt</option>
006 </select>

```

Zur Mehrfachauswahl mittels `select` kann das boolesche Attribut `multiple` verwendet werden. Was indes die Darstellung des `<select>` Elements angeht, so wird auf Mobilgeräten bei Zugriff auf das Element üblicherweise ein geeignetes natives Dialogelement angezeigt, das für die Einfach- und Mehrfachauswahl gleichermaßen verwendet werden kann, während Desktop-Browser die Auswahlliste an Ort und Stelle expandieren. Folgende Abbildung zeigt z. B. die Realisierung des oben beschriebenen Elements auf einem Android Tabletgerät.

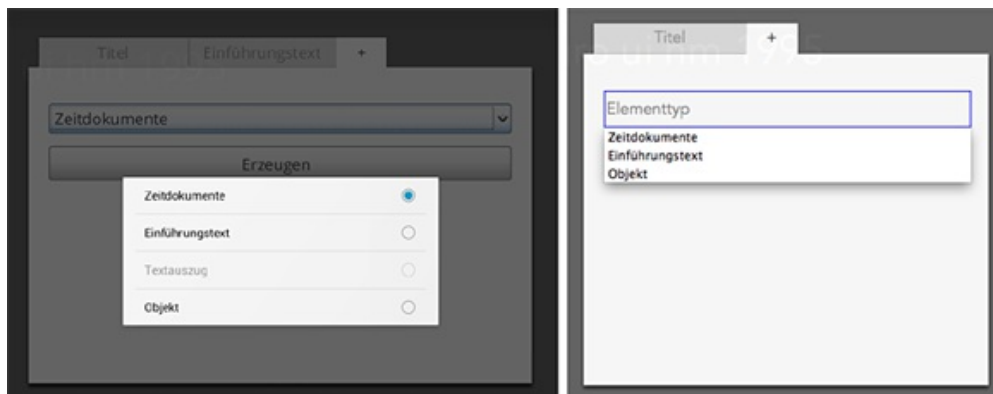


Abb.: Darstellung eines aktiven select Elements auf einem mobilen Endgerät (links) und eines input Elements mit datalist auf einem Desktop-Browser (rechts)

<datalist>

Normalerweise wird bei Nichtvorhandensein einer selected Option die erste Option eines `<select>` Elements vor-ausgewählt, und das placeholder Attribut ist für `<select>` nicht verfügbar. Durch Setzen des Attributs `required` auf `<select>` und Verwendung eines ersten option Elements mit leeren value Attribut kann jedoch die Funktionsweise von Platzhaltertexten nachgebaut werden. In diesem Fall ist für das erfolgreiche Absenden des Formulars die Auswahl eine anderen, nicht -leeren, `<option>`-Elements erforderlich.



Quellcode

Liste von Optionen mit Platzhalter

```
001 <!-- Liste von Optionen mit Platzhalter -->
002 <select name="elementType" required="required">
003   <option value="">select...</option>
004   <option value="zeitdokumente">Zeitdokumente</option>
005   <option value="einfuehrungstext">Einführungstext</option>
006   <option value="textauszug" disabled="disabled">Textauszug</option>
007   <option value="objekt">Objekt</option>
008 </select>
```

Nur eine vermeintliche Lösung für die Platzhalter-Problematik in Verbindung mit einer Wertauswahl stellt hingegen die Assoziation eines `<input>` Elements, für das Platzhaltertext erlaubt ist, mit einem `datalist` Element dar, für welches wie für `select` eine Menge von Optionen angegeben werden kann – die Realisierung ist in der vorherigen Abbildung zu sehen:



Quellcode

datalist

```
001 <!-- deklariere eine Menge von Optionen als datalist -->
002 <datalist id="elementTypes">
003   <option value="zeitdokumente">Zeitdokumente</option>
004   <option value="einfuehrungstext">Einführungstext</option>
005   <option value="textauszug" disabled="disabled">Textauszug</option>
006   <option value="objekt">Objekt</option>
007 </datalist>
008 <form id="form_addElement">
009   <!-- referenziere die Optionen in input -->
010   <input name="elementType" placeholder="Elementtyp" list="elementTypes"
011     autocomplete="off">
012   <!-- (...) -->
013 </form>
```

Im Gegensatz zu `select` kann eine Auswahl von einer der angebotenen Optionen jedoch nicht erzwungen werden, sondern ist die Eingabe beliebiger Werte in das `input` Element möglich. Außerdem werden als `disabled` markierte Optionen im Gegensatz zu `select` aus der dargestellten Liste entfernt. Was schließlich die Unterstützung durch mobile Browser angeht, so scheint sich auch diese noch in einem eher rudimentären Stadium zu befinden.

2.2 Aktionselemente

Nachdem sich die vorangegangenen Abschnitte mit Eingabeelementen im allgemeinen und Auswahlelementen im besonderen beschäftigt haben, werden wir uns hier mit HTML-Bedienelementen beschäftigen, die in besonderer Weise dafür geeignet sind, Aktionen zu initiieren, die dann beispielsweise in der Ausführung einer serverseitig implementierten CRUD-Operation resultieren. Die Tatsache, dass prinzipiell jedes Bedienelement in HTML durch Setzen geeigneter Event Handler als Aktionselement in diesem Sinne verwendet werden kann, haben Sie sich in der vorangegangenen Lerneinheit bereits selbst zu Nutze gemacht, als Sie aus den Click/Tap Event Handlern für die `<a>`-Elemente der Fußleiste die CRUD-Operationen für `Objekt` aufgerufen haben:



Quellcode

CRUD Operationen für Objekt

```
001 <ul id="actionbar_object">
002   <!-- (...) -->
003   <li><a id="createObjectAction" href="javascript:createObject()">Neues
004     Objekt</a></li>
005   <li><a href="javascript:updateObject()">Objekt ändern</a></li>
006   <li><a href="javascript:deleteObject()">Objekt löschen</a></li>
007 </ul>
```

Im Rahmen unserer Beschäftigung mit Formularen wollen wir uns auf die Verwendung von `submit`-Elementen sowie anderen Aktionselementen konzentrieren, die in einem Formular – oder einer Ansicht, die Formulare verwendet – eingesetzt werden können. Betrachten Sie dafür die folgende Ansicht:

Abb.: Ansicht

`<button>`

Hier werden zwei gleichartig wirkende Bedienelemente verwendet, die Sie vor dem Hintergrund Ihrer Erfahrung als *digital natives* unweigerlich als Aktionselemente identifizieren dürften. Nur eines davon – nämlich das Element mit der Beschriftung „Aktualisieren“ – ist aber ein `submit` Element, wie wir es oben beschrieben haben, mittels dessen Betätigung die eingegebenen Formulardaten an die serverseitigen Anwendungskomponenten übermittelt werden können. Bei dem zweiten, bis auf die Beschriftung identischen, Element handelt es sich hingegen um ein `<button>` Element, das visuell zwar üblicherweise genauso wie ein `submit` Element realisiert wird, abgesehen davon jedoch keine dem letzteren vergleichbare Logik des „Einsammelns und Absendens“ von Formulardaten verwendet.

So unterscheidet sich ein `button` hinsichtlich der mit ihm realisierbaren Funktionalität wie auch hinsichtlich der Unterstützung, die uns HTML für deren Umsetzung bereitstellt, zunächst nicht von den `<a>`-Elementen im obigen Beispiel. Lediglich die Assoziation des Elements mit der bei Betätigung auszuführenden Aktion wird hier auf der Ebene von JavaScript realisiert und nicht im HTML-Markup zugewiesen. Der Identifikator des zu löschenden Topicview wird im folgenden Beispiel durch eine Variable zur Verfügung gestellt, auf welche die Funktion `deleteTopicview()` Zugriff hat:



Quellcode

deleteTopicview() Funktion

```

001 // rufe bei Betätigung des Buttons und nach Rückbestätigung die
002 // deleteTopicview() Funktion auf
003 document.getElementById("deleteTitleButton").addEventListener("click",
004     function(event) {
005         if (confirm("Möchten Sie die Ansichtsbeschreibung wirklich löschen?")) {
006             deleteTopicview(function(deleted) {
007                 if (deleted) {
008                     showToast("Die Ansichtsbeschreibung wurde gelöscht.");
009                     /* (...) */
010                 }
011             });
012         }
013     });

```

Wenn wir uns allerdings das der Ansicht zugrunde liegende Markup ansehen, dann wird offensichtlich, dass das `<button>` Element durchaus ein Bestandteil des Formulars ist – auch wenn das Formular selbst bzw. die eingegebenen Formulardaten in der vorliegenden Implementierung für die Behandlung der Betätigung des Buttons nicht erforderlich sind:

```

001 <form name="titleForm">
002     <input name="title" placeholder="Titel" required="required" autofocus="
003         autofocus" autocomplete="off"/>
004     <input type="submit" value="Erzeugen" name="submit"/>
005     <button id="deleteTitleButton" type="button">Löschen</button>
006 </form>

```

Gerade bei einer Gestaltung wie der vorliegenden kann die Platzierung des `button` Elements innerhalb des Formulars aber vorteilhaft sein, um das Element hinsichtlich seiner Dimensionierung relativ zum umgebenden `<form>` Element bzw. analog zu seinen Geschwisterelementen, d. h. den „echten“ Formularbestandteilen, zu realisieren. Vielleicht fragen Sie sich aber, weshalb auf dem Button-Element das scheinbar redundante Attribut `button` gesetzt ist. Dieses ist erforderlich, da andernfalls der Button analog zu einem `<input>` Element mit Typ `submit` behandelt würde und damit die Übermittlung der Formulardaten auslösen würde. Die Attributzuweisung `type="button"` hingegen kennzeichnet den Button als „einfaches“ Aktionselement, das unabhängig vom Formularkontext verwendet werden kann. Sie ist allerdings nur im vorliegenden Fall erforderlich, in welchem der Button innerhalb des Formulars platziert ist, und wäre außerhalb eines Formulars nicht notwendig. Alternativ könnte die hier mittels `<button>` realisierte Funktionalität auch wie folgt umgesetzt werden:

```

001 <form name="titleForm">
002     <!-- (...) -->
003     <input type="button" value="Löschen" id="deleteTitleButton"/>
004 </form>

```

Bei Verwendung mehrerer Aktionselemente innerhalb eines Formulars ist also vor allem zu beachten, dass nur eines der Elemente die „Submit-Funktion“, die im Einsammeln und Übermitteln der eingegebenen Formulardaten besteht, übernimmt. Ob dieses Element als `button`, dessen Default-Typ `submit` ist, oder als `input` Element mit Typ `submit` umgesetzt wird, und welche HTML-Elemente für die anderen Aktionselemente verwendet werden, ist mit Blick auf deren Austauschbarkeit zweitrangig.

<button> vs. <input>

Worin besteht dann aber der Unterschied `button` und `input` und wann soll welches der beiden Elemente bevorzugt verwendet werden? Das Kriterium hierfür können Sie evtl. bereits anhand des obigen Markupbeispiels mit `<button>` ermitteln. So ist `<input>` als leeres Element spezifiziert, dem keine Tochterelemente zugeordnet werden können, während `<button>` Tochterelemente erlaubt. Als solche können nicht nur Textinhalte, wie im vorliegenden Beispiel, verwendet werden, sondern auch Elemente wie `` oder `div`, mittels derer eine differenzierte Formatierung der Beschriftung eines Aktionselements möglich ist, inklusive der Einbindung von Grafiken.

Siehe hierfür auch die Erörterungen zur Verwendung von `<button>` in <http://particletree.com/features/rediscovering-the-button-element/>.

Bevor wir uns mit der Ausführung von Submit-Aktionen beschäftigen – bezüglich welcher wir Ihnen immer noch eine Alternative zur eingangs geschilderten „Web 1.0“ Verarbeitung schuldig sind – möchten wir abschließend noch auf die Umsetzung eines Gestaltungsmusters eingehen, das vor allem in mobilen Anwendungen zur Auslösung von formularbezogenen Aktionen gebräuchlich ist. Die Bedienelemente hierfür werden z. B. in Android seit Version 4.* nicht mehr unterhalb der Eingabefelder eines Formulars realisiert, sondern üblicherweise in einem als Aktionsleiste bezeichneten Bereich am Kopf einer Bildschirmansicht, wie in der Abbildung unten zu sehen ist.

Um eine solche Gestaltungsvorlage aber mit HTML umzusetzen, ist es erforderlich, das betreffende Aktionselement außerhalb des `<form>` Elements zu platzieren, auf das sich die Aktion bezieht – so ließe sich die im Browser dargestellte Ansicht, die in folgender Abbildung rechts dargestellt ist, nur dann durch ein Tochterelement der darunter dargestellten form umsetzen, wenn diesem eine absolute Positionierung zugewiesen würde.

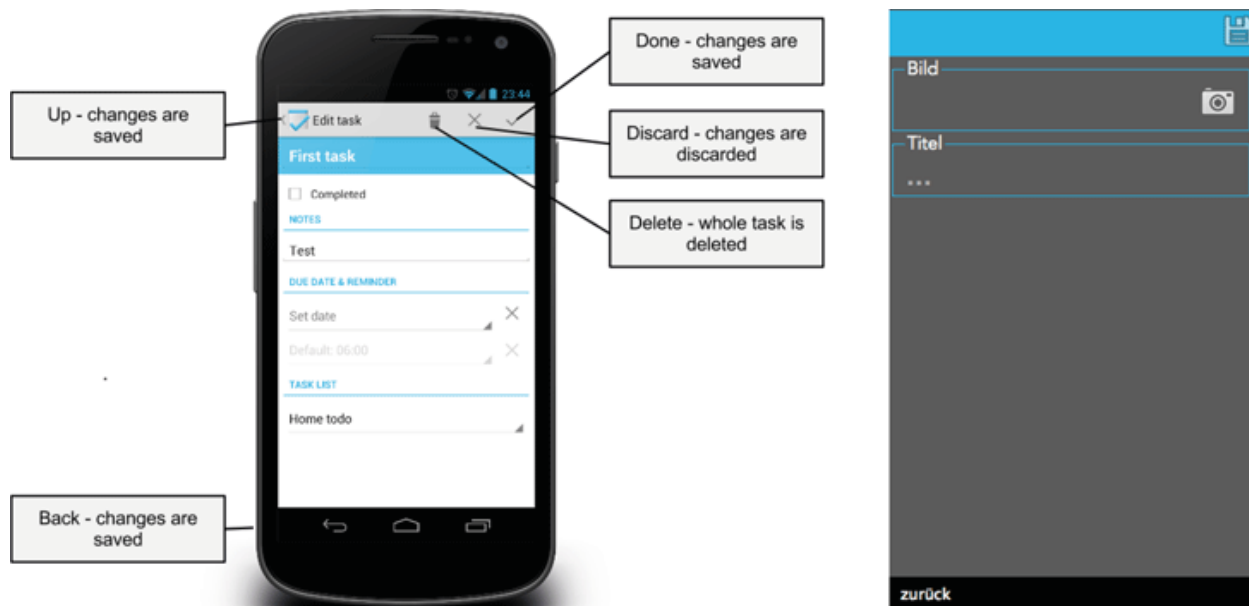


Abb.: Gestaltungshinweise zur Verwendung der Aktionsleiste in Android (links) und Umsetzung einer Aktionsleiste mit HTML und CSS (rechts)

Für die gezeigte Ansicht erscheint es jedoch viel sinnvoller, die Positionierung des Elements relativ zur umgebenden Kopfzeile durchzuführen. Für solche Fälle, in denen ein Submit-Element im Hinblick auf den Aufbau der darzustellenden Ansicht nicht sinnvoll innerhalb eines Formulars platziert werden kann, bietet HTML uns aber die Möglichkeit, mittels eines `form` Attributs auf dem Element kenntlich zu machen, auf welches Formular sich das Element bezieht. Wie hier zu sehen ist, wird als dessen Wert die `id` des betreffenden Formulars gesetzt. Auch hier könnte alternativ ein `button` Element mit entsprechend gesetztem `form` Attribut verwendet werden:



Quellcode

Submit-Element für das Formular

```
001 <header>
002 <!-- Submit-Element für das Formular im Hauptbereich der Ansicht -->
003 <input type="submit" value="Speichern" form="detailview-form"/>
004 </header>
005 <div>
006 <form id="detailview-form">
007 <fieldset>
008 <legend>Bild</legend>
009 <output id="detailview-img">
010 </output>
011 <button>Aufnehmen</button>
012 </fieldset>
013 <fieldset>
014 <legend>Titel</legend>
015 <input type="text" placeholder="..." autofocus="autofocus" required="
016 required"/>
017 </fieldset>
018 </form>
019 </div>
```


Weitere Aspekte der Gestaltung von Formularen werden wir im Rahmen der vertiefenden Ausführungen in der Lerneinheit MFM thematisieren.

2.3 Verarbeitung von Formulardaten


Als vorläufigen Abschluss unserer einführenden Bemerkungen zu Formularen werden wir uns hier zunächst damit beschäftigen, wie alternativ zur Ausführung einer Formularaktion auf Grundlage der im `<form>` Element angegebenen Attribute für `action` und ggf. `method` die Übermittlung von Formulardaten mittels `XMLHttpRequest` initiiert werden kann. Daran anschließend werden wir auf den wichtigen Aspekt des Data Bindings bezüglich Formularen eingehen, d. h. wir werden beschreiben, wie die zu übermittelnden Formulardaten aus den Eingabeelementen aufgebaut werden können bzw. wie in der umgekehrten Richtung die Eingabeelemente eines Formulars mit den Attributwerten eines existierenden Objekts befüllt werden können.

Als alternativen Fall von Data Binding werden wir hier auch auf den Aufbau von Tabellen eingehen. Auf Basis des hier vermittelten Wissens können Sie dann das im Übungsprogramm vorgesehene Formular zur Ausführung von CRUD Operationen bezüglich der Ansichtselemente für `Objekt` umsetzen sowie eine Ansicht aufbauen, die Ihnen eine Übersicht über alle existierenden Instanzen von `Objekt` liefert. Der zweite Teil unserer Behandlung von Formularen wird dann die Aspekte der Validierung von Formularen beleuchten und die spezifischen Merkmale von Formularen, die zum *Datenupload* verwendet werden, darlegen.

2.3.1 Anwendungsspezifische Submit-Ausführung

In unserer einleitenden Betrachtung eines einfachen Formulars hatten wir gezeigt, dass bei Ausführung der Submit-Aktion bezüglich eines Formulars die eingegebenen Formulardaten in einem durch `action`-URL und `method` beschriebenen HTTP-Request an den Server übermittelt werden. Um alternativ dazu die Formulardaten selbst mittels `XMLHttpRequest` zu übermitteln, müssen wir also zunächst einen geeigneten Weg finden, um auf die Submit- Aktion reagieren zu können.

JavaScript URLs

Hierfür existieren verschiedene Möglichkeiten. Der einfachste Weg ist die Angabe einer JavaScript Funktion zur Formulareausführung als Wert des `action` Attributs – bedenken Sie, dass an jeder Stelle, an der in HTML-Elementen eine URL bezüglich einer serverseitigen Ressource verwendet werden kann, auch die Angabe einer JavaScript URL möglich ist, bei der wir den auszuführenden JavaScript Code mittels des URL-Schemas `javascript:`  [präfigieren](#) . Wir könnten im hier betrachteten Formular für `Topicview` Elemente die Ausführung der gewünschten CRUD-Operation also wie folgt initiieren:

```
001 <form name="titleForm" action="javascript:createTopicview();">
002   <!-- (...) -->
003 </form>
```

Im Verlauf der Historie von Webanwendungen und der für deren Zugriff verwendeten Browser war es jedoch lange Zeit erforderlich, Fälle vorzusehen, in denen ein Nutzer entweder einen nur bedingt für JavaScript befähigten Browser verwendete oder die JavaScript-Ausführung im Browser aus Sicherheitsgründen deaktiviert hatte. Für solche Fälle war es wünschenswert, auch solchen Nutzern eine minimale Anwendungsfunktionalität bereit zu stellen und für Formulare idealerweise sowohl eine konventionelle Formulareausführung in „Web 1.0“ Manier als auch ggf. eine Behandlung mittels `XMLHttpRequest` zu ermöglichen. Mit expliziten Aufrufen von JavaScript Funktionen auf Ebene des HTML-Markups wäre eine solche „graceful degradation“ bezüglich widriger Rahmenbedingungen für die Formulareausführung nicht mehr möglich, da hier die Nutzung von JavaScript als alleinige Alternative direkt in das HTML-Markup implantiert wird.

Nun mögen Sie durchaus einwenden, dass das beschriebene Szenario im Rahmen des in unserer Veranstaltung verfolgten „Lab-Ansatzes“, der auf die Verwendung der aktuellsten Ausdrucksmittel und Browser setzt, keine Relevanz mehr besitzt – zumal wir uns mit unserer Betrachtung von Webanwendungen auf mobilen Geräten noch weiter von den Ursprüngen des „Web 1.0“ entfernen. Dennoch gilt es nach wie vor als besserer Programmierstil, Callbacks bezüglich JavaScript-Funktionen, die auf das Auftreten von DOM Events reagieren sollen, in JavaScript selbst zu deklarieren und nicht auf der Ebene des HTML-Markups.

Dagegen könnte man allerdings einwenden, dass eine solche Anbindung an Controller-Funktionalität auf der Ebene eines View in anderen Ansätzen zur Entwicklung von GUIs durchaus üblich ist, sei es in iOS, (ansatzweise) in *Android* oder aber in einem MVC Framework wie Java Server Faces.

View-Controller-Funktionalität bei Android

Hier eingeschränkt auf die Deklaration von Callbacks bezüglich Click-Events.

Event Handler für `submit`

Um die Reaktion auf Submit-Aktionen im Rahmen etablierter Richtlinien umzusetzen, stehen uns zwei Möglichkeiten zur Verfügung. So löst die Aktion ein `submit` Ereignis aus, auf welches wir mit den hierfür generell verfügbaren Mitteln reagieren können, nämlich wahlweise durch Setzen einer Callback-Funktion als alleiniger `onsubmit` Event Handler oder durch Hinzufügung einer Callback-Funktion zur Liste etwaiger existierender Event Handler bezüglich des `submit` Ereignisses. Wir nehmen nachfolgend an, dass uns in der Variable `titleForm` das Formularelement zur Verfügung steht:



Quellcode

Event Handler für `submit`

```
001 // Variante 1: Setzen eines onsubmit Event Handlers
002 titleForm.onsubmit = function(event) {
003     submitTitleForm();
004     return false;
005 };
006
007 // Variante 2: Hinzufügung eines Event Handlers
008 titleForm.addEventListener("submit", function(event) {
009     event.preventDefault();
010     submitTitleForm();
011 });
```

Diese Implementierungsvorschläge dürften weitgehend selbsterklärend sein – mit Ausnahme der Anweisung `return false;` in Variante 1 und dem Aufruf von `preventDefault()` auf dem Submit-Ereignis in Variante 2. Der Zweck beider Anweisungen besteht darin, das Default-Verhalten des Formulars bezüglich der Submit-Aktion zu unterbinden. Bei Vorliegen eines `action` Attributs besteht dieses Ziel nämlich auch im fortgeschrittenen Stadium des „Web 2.0“ Zeitalters in der Ausführung der Aktion via Übermittlung eines HTTP-Requests und im Neuaufbau der dargestellten Ansicht auf Grundlage des HTTP-Response. Die beiden Event Handler Setzungen „unterwandern“ diese Absicht des Browsers und implementieren in `submitTitleForm` eine alternative Übermittlung von Formulardaten an den Server.

Sollte bei der Ausführung der Event Handler jedoch ein Fehler auftreten, wird der Browser darüber – ebenfalls im Sinne einer „graceful degradation“... – hinwegsehen und ungeachtet des Fehlers die Formularaktion entsprechend dem `action` Attribut ausführen. Lediglich bei Nichtvorhandensein dieses Attributs können wir sicher sein, dass auch im Fehlerfall kein Neuladen des dargestellten Dokuments erfolgen wird. Zwar brechen wir damit die Mehrgenerationenfähigkeit unserer Anwendung; insbesondere mit Blick auf die Vermittlung von Entwicklungskompetenzen bezüglich mobiler Webanwendungen scheint uns dies jedoch vertretbar. Dies gilt nicht zuletzt da die hier zugrunde gelegte MVC-Architektur mit ihrer rein clientseitig umgesetzten Controller-Funktionalität ohne aktuelle Ausdrucksmittel von JavaScript gar nicht lauffähig ist.

Es sei angemerkt, dass der oben gezeigte Variante 1 bereits in der Frühzeit von Formularen eine Funktionalität zugeordnet war, aus der heraus die Rückgabe von `false` motiviert werden kann. So konnte, falls erwünscht und möglich, als Reaktion auf das `submit` Ereignis eine clientseitige Validierungsprüfung der eingegebenen Formulardaten durchgeführt werden oder aber eine Rückbestätigung mittels eines `confirm()` Dialogs mit sehr einfachen Mitteln durchgeführt werden. In beiden Fällen sollte der Rückgabewert `false` anzeigen, dass die Voraussetzungen für die tatsächliche Ausführung der Submit-Aktion nicht gegeben sind.


Bezüglich Variante 2 ist andererseits zu beachten, dass für jede Ausführung von `addEventListener()` auf dem Formularelement ein weiterer Callback für das betreffende Ereignis zusätzlich zu ggf. bereits vorhandenen hinzugefügt wird. Falls der hier gezeigte Funktionsaufruf bezüglich `addEventListener()` also mehrfach ausgeführt wird – was mit Blick auf Erfahrungswerte bezüglich der Bearbeitung der Übungsaufgaben nicht auszuschließen ist – resultiert dies in einem mehrfachen Aufruf an `submitTitleForm()` und entsprechend in einer zumeist mit Irritationen verbundenen mehrfachen Übermittlung der Formulardaten.

2.3.2 Data Binding für Formulardaten

Wie bereits an verschiedenen Stellen erwähnt bezeichnet der Begriff des Data Binding zum einen die Übertragung von Daten aus den von einer Anwendung verwendeten Model- Objekten auf die Ebene der Ansichten der Anwendung und zum anderen den umgekehrten Weg des Aufbaus von Objekten auf Basis der durch einen Nutzer über die Bedienelemente der Ansichten eingegebenen Werte. Diese beiden Richtungen können als Outbound bzw. Inbound Data Binding bezeichnet werden. Es wurde auch bereits erwähnt, dass die Basisfunktionalität von HTML und JavaScript für keine der beiden Richtungen des Data Binding spezifische Ausdrucksmittel zur Verfügung stellt.

Bezüglich Formularen könnte man diesbezüglich die Einschränkung vornehmen, dass das Inbound Data Binding für Formulare, deren Daten bei Angabe einer `action` URL durch den Browser ausgelesen und an den Server übertragen werden, durchaus automatisch und ohne Notwendigkeit manueller Entwicklungsmaßnahmen erfolgt – einwenden könnte man dann allerdings, dass dann auf Ebene der serverseitigen Anwendung Data Binding als Aufbau von Objekten auf Grundlage der übertragenen HTTP-Request-Parameter implementiert werden muss. Wie wir gleich sehen werden, steht jedoch auch bei Behandlung von Submit-Ereignissen in JavaScript das durch den Browser implementierte „Einsammeln“ der Formulardaten nicht als wieder verwendbare Komfortfunktion zur Verfügung. Es muss vielmehr durch den Anwendungsentwickler in den Callback-Funktionen für `submit` händisch „nachgebaut“ werden.

Auslesen von Formularen

Ausgangspunkt für das Auslesen eingegebener Formulardaten ist die Repräsentation der Formulare als  Formular-Objekte, die Teil des DOM Baumes bilden. Auf diese können wir selbstverständlich mit allen uns bekannten generischen Mitteln der DOM API zugreifen und z. B. Formular-Objekte mittels `getElementById()` aus dem DOM zur weiteren Verwendung auslesen. Wir möchten jedoch erwähnen, dass uns aufgrund der Prominenz von Formularen für HTML-basierte GUIs eine alternative Zugriffsmöglichkeit zur Verfügung steht, bei der wir ein Formular des Namens `titleForm` – d. h. dessen `<form>` Element die Attributsetzung `name="titleForm"` enthält – wie folgt zugreifen können:

```
// Zugriff auf ein Formular
var titleForm = document.forms["titleForm"];
```

Als weitere Alternative würde einem Event Handler bezüglich `submit` natürlich auch das Formular-Element als Wert von `event.target` übergeben.

Auslesen von Feldwerten

Nachdem wir ein Formular-Objekt auf einem der genannten Wege ausgelesen haben können wir auf die Werte der meisten Felder recht bequem zugreifen, indem wir jeweils ein entsprechend dem Feldnamen benanntes Attribut aus dem Formular auslesen und dessen `value` Attribut verwenden, z. B. könnte wie folgt auf den Wert des Feldes `title` des oben verwendeten Formulars zugegriffen werden:

```
var title = titleForm.title.value;
```

Checkboxes und Radiobuttons

Dieser einfache Zugriff ist insbesondere für `input`, `textarea` und `select` Elemente möglich. Für den Zugriff auf Checkboxes, für deren interne Repräsentation die Verwendung eines `value` Attributs mit Booleschen Werten zumindest denkbar wäre, muss alternativ auf ein Attribut namens `checked` zugegriffen werden. Etwas mehr Umstände bereitet die Ermittlung des jeweils ausgewählten Werts aus einer Menge von Radiobuttons. Werden diese z. B. zur Eingabe bezüglich eines Feldes namens `contentMode` auf einem Formular `einfuehrungstextForm` verwendet – auch dieses Beispiel ist der Beispielimplementierung entnommen – dann ist der Wert von `einfuehrungstextForm.contentMode` nicht wie in den anderen Fällen eine Repräsentation des betreffenden Eingabeelements, sondern ein Array, dessen Elemente die einzelnen `<input type="radio">` Elemente bezüglich `contentMode` sind. Auch deren Zustand wird durch ein Attribut namens `checked` repräsentiert. So könnte denn mittels einer Iteration über den Array und Abgleich von `checked` das Element ermittelt werden, dessen `value` als Wert des Formularfelds `contentMode` verwendet werden soll.

Das folgende Codebeispiel wählt jedoch einen anderen – durchaus diskutablen – Weg. Es nutzt die Tatsache aus, dass bezüglich des betreffenden Radiobuttons die Pseudoklasse `:checked` zutrifft und liest das Element mittels `querySelector()` aus dem Formular aus – dieser Zugriff wäre grundsätzlich auch für Checkboxes möglich, erscheint jedoch deutlich aufwendiger als die unten gezeigte Alternative:



Quellcode


Auslesen von Werten

```
001 // Auslesen des ausgewählten Werts eines select Elements für das Feld
002 // elementType
003 var elementType = newElementForm.elementType.value;
004
005 // Auslesen des Werts einer Checkbox für ein Feld namens translate
006 var doTranslate = einfuehrungstextForm.translate.checked;
007
008 // Auslesen des Werts für ein Formularfeld contentMode, das Radiobuttons
009 // als Eingabeelemente verwendet
010 var contentMode = document.querySelector("input[name=contentMode]:checked
011 ").value;
```

Aus einer architektonischen Perspektive erscheinen diese Unterschiede bei der Behandlung von Radiobuttons vs. Checkboxes vs. Select Elementen insofern problematisch, als es sich hier – wie oben dargelegt – um Auswahllemente handelt, die bis zu einem gewissen Grade auch untereinander austauschbar sind, so sind Radiobuttons grundsätzlich sowohl als Alternative zu Checkboxes, als auch zu Select-Elementen denkbar. Die Notwendigkeit eines jeweils unterschiedlichen Zugriffs auf Ebene der JavaScript API bringt daher eine Abhängigkeit zwischen View und der für das Data Binding zuständigen Komponente mit sich. Diese Abhängigkeit hätte ggf. zur Folge, dass bei einer Änderung der View auch die Implementierung des Data Binding modifiziert werden müsste.

Fragwürdig ist die Ungleichbehandlung der drei Fälle aber auch aus dem Grund, dass uns bei Übermittlung von Formulardaten auf „traditionellem“ Wege das Auslesen der eingegebenen Werte durch den Browser abgenommen wird. Es stellt sich daher die Frage, weshalb auf Ebene der JavaScript API offensichtlich noch keine vereinheitlichte Lösung entsprechend der Behandlung von `input` angeboten wird. So könnte ungeachtet des Elementtyps der eingegebene Wert grundsätzlich über ein `value` Attribut auf einer Repräsentation des Feldes verfügbar sein.

Outbound Data Binding

Unter Berücksichtigung der genannten Sonderfälle, insbesondere bezüglich Radiobuttons, kann auf den Wert von Formularfeldern nicht nur lesend, sondern auch schreibend zugegriffen werden. Die vorstehend verwendeten Attributreferenzen bezüglich `value` bzw. `checked` können daher auch im Rahmen der Implementierung eines *Outbound Data Binding* verwendet werden, bei der wir die Attribute existierender Instanzen komplexer Datentypen, wie in der Abbildung „Illustration des Zusammenhangs von Formulardaten und Instanzen komplexer Datentypen“ (Siehe  Kapitel 1.2) illustriert, in die Felder übertragen, die auf Ebene des Formulars die betreffenden Attribute repräsentieren.

Aber nicht nur der Wert eines Formularfeldes, sondern auch die Werte aller anderen im HTML-Markup verwendbaren Attribute von Formularfeldern sind über die Objektrepräsentation des Feldes *lesend und schreibend* zugreifbar, so lässt sich z. B. die Aktivierung und die Erfordernis eines Feldes wie folgt aus JavaScript heraus manipulieren und z. B. im Rahmen der Behandlung abhängiger Felder als einer weiteren für nutzerfreundliche Formulare relevanten Funktionalität einsetzen:

```
001 var srcField = document.forms["einfuehrungstextForm"].src;
002 srcField.disabled = false;
003 srcField.required = true;
```

Einschränkungen

Bezüglich der Abbildbarkeit von Instanzen komplexer Datentypen auf Formulare in HTML sei schließlich noch auf eine Einschränkung hingewiesen. So erlaubt HTML innerhalb von `form` Elementen zwar neben genuin formularbezogenen Elementen wie `<input>`, `<button>`, `field` etc. zahlreiche weitere HTML-Elemente, die das Formular insbesondere im Hinblick auf die visuelle Darstellung strukturieren. Die Einbettung von Formularen in Formulare ist jedoch in HTML nicht möglich. Entsprechend müssen Instanzen komplexer Datentypen, die Attribute mit ihrerseits komplexen Typen verwenden, beim Data Binding auf die „flachere“ Struktur eines Formulars projiziert bzw. daraus aufgebaut werden. Eine Berücksichtigung eingebetteter Strukturen ist aber zumindest visuell möglich – so haben wir oben ja bereits auf die Verwendung des `<fieldset>` Elements hingewiesen, das z. B. in Verbindung mit `<legend>` zu ebendiesem Zweck verwendet werden kann.

Tabellen

Zuletzt möchten wir unsere Betrachtung des Data Binding in GUIs über Formulare hinaus auf einen weiteren Fall komplexer Ansichtselemente ausdehnen, für den Data Binding relevant ist – wenn auch nur als *Outbound* Data Binding. Dienen Formulare im Hinblick auf letzteres der Darstellung einer einzelnen Instanz eines Datentypen, so können *Tabellen* verstanden werden als Repräsentationen einer Menge von Instanzen, wobei jede Reihe in der Tabelle eine Instanz und jede Spalte ein Attribut des Datentypen darstellt. Mit Blick auf unsere Betrachtung der Persistierung von Datentypen in relationalen Datenbanken in der Lerneinheit NJM erscheint diese Beschreibung zwar redundant, unter Bezugnahme auf das Thema der vorliegenden Lerneinheit soll sie aber verdeutlichen, dass auch Tabellen ein Fall von Data Binding sind, für deren Aufbau wie für Formulare die Verfügbarkeit deklarativer Ausdrucksmittel wünschenswert ist – und durch die Data Binding-Funktionalität von MVC-Frameworks auch gewöhnlicherweise gewährleistet wird.

Auch hier sind deutliche Übereinstimmungen zwischen Frameworks für verschiedene Plattformen – z. B. AngularJS und JSF – zu beobachten. Für den Aufbau einer Tabelle mit den „Bordmitteln“ von HTML und JavaScript sei hier erwähnt, dass hierfür die Elemente `<table>` zur Kennzeichnung einer Tabelle, `<tr>` zur Markierung einer Tabellenzeile sowie `<td>` zur Umsetzung einer Zelle in einer Zeile verwendet werden können und dass die Spaltenüberschriften in einer initialen Zeile mittels des Elements `<th>` anstelle von `<td>` markiert werden können. Weitere Hinweise zu Tabellen finden Sie in einschlägigen [www Online-Nachschlagewerken](#).

Für den dynamischen Aufbau von Tabellen erscheinen die Ausdrucksmittel der DOM API sehr geeignet. Dank der Möglichkeit, Elemente mittels `cloneNode()` zu reproduzieren, lassen sich Tabellen z. B. auch auf Basis eines „Templates“, das das Markup für eine Tabellenzeile bereitstellt, recht einfach ohne Verwendung zusätzlicher Bibliotheken oder Frameworks aufbauen, wie wir es in Lerneinheit JSL im Kapitel 2.4 gezeigt haben.

Listenansichten

In mobilen Anwendungen können Listenansichten, wie in der Abbildung unten gezeigt, als analoge Darstellungsform zu Tabellen angesehen werden. Im Gegensatz zur Visualisierung in einer Tabelle, in der Instanzen vertikal untereinander und die Attribute einer Instanz horizontal nebeneinander dargestellt werden, behalten Listenansichten zwar die Anordnung von Instanzen untereinander bei, kombinieren aber je nach Art und Anzahl der darzustellenden Attribute horizontale und vertikale Anordnungen. So werden in folgender Abbildung in der Ansicht der „Titel“ und „Untertitel“ einer Instanz untereinander, diese aber als Block neben einem dritten Attribut angeordnet, für dessen Darstellung ein Bild verwendet wird.

Hinsichtlich des dynamischen Aufbaus im Rahmen eines Outbound Data Binding unterscheiden sich Listenansichten jedoch weder hinsichtlich der Komplexität der Anforderung, noch hinsichtlich der zur Umsetzung anwendbaren Ausdrucksmittel von Tabellen.

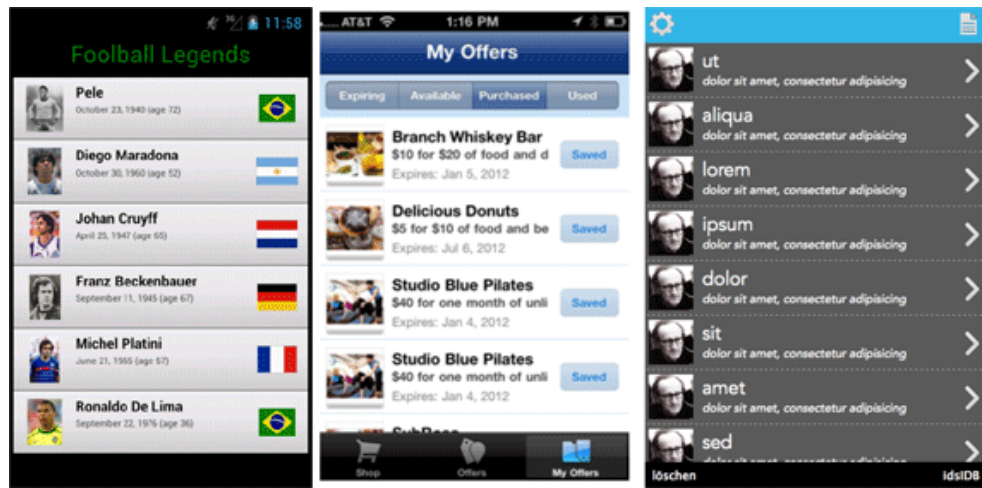


Abb.: Beispiele für die Verwendung von Listenansichten in mobilen Anwendungen (links, mitte) und „Nachbau“ einer Listenansicht mit HTML und CSS

Zusammenfassung

- Die Anliegen, die durch ein Formular zum Ausdruck gebracht werden können, lassen sich grob in zwei Gruppen unterteilen - *Informationsanliegen* und *Transaktionsanliegen*.
- Ziel jeglichen Formulars sollte es sein, die Nutzer bei der effizienten und korrekten Formulierung ihrer Anliegen zum Zweck der erfolgreichen Anliegenbefriedigung zu unterstützen.
- Formulare können aufgefasst werden als komplexe Bestandteile der Ansichten einer graphischen Nutzerschnittstelle, die dem Nutzer zur Formulierung von Anliegen zum einen eine Menge dafür geeigneter Eingabeelemente zur Verfügung stellen.
- Mit Blick auf die Bedeutung der Formularvalidierung für die Anliegenverfolgung eines Nutzers sollten nicht nur manuell bearbeitete Formulare, sondern auch Formulare in GUIs eine möglichst frühe Gültigkeitsprüfung der in ein Formular eingegebenen Daten vornehmen.
- Zu den grundlegenden Begriffen gehören Formularfeld, Feldwert bzw. Formularfeldwert und Formulardaten.
- Die Anforderungen bezüglich der Umsetzung nutzungstauglicher Formulare sind Einfachheit, Effizienz, Zielgerichtetheit, Frühe Fehlerdetektion, Kooperative Fehlerbehandlung und Flexibilität.
- Das wichtigste HTML-Element für die Realisierung von Eingabeelementen für Formularfeldwerte ist das Element `<input>`. Dessen `name` Attribut muss innerhalb eines Formulars eindeutig sein, kann aber in verschiedenen voneinander getrennten Formularen innerhalb eines HTML-Dokuments wiederverwendet werden.
- Data Binding bezeichnet zum einen die Übertragung von Daten aus den von einer Anwendung verwendeten Model- Objekten auf die Ebene der Ansichten der Anwendung und zum anderen den umgekehrten Weg des Aufbaus von Objekten auf Basis der durch einen Nutzer über die Bedienelemente der Ansichten eingegebenen Werte.

Sie sind am Ende dieser Lerneinheit angelangt. Auf den folgenden Seiten finden Sie noch Übungen.

Übungen

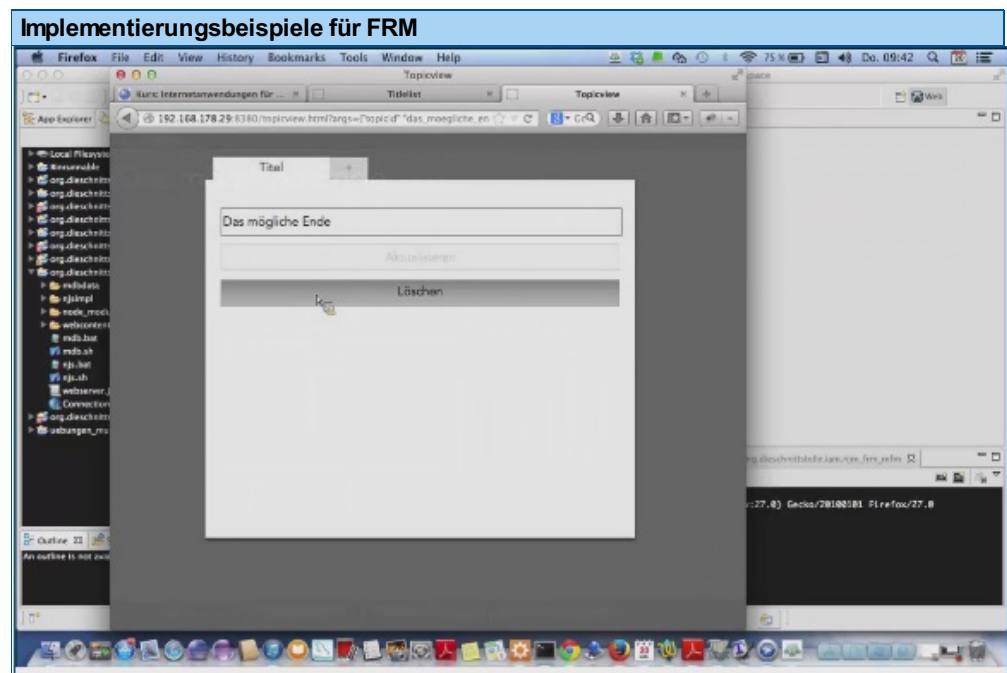
Die Implementierungsbeispiele zum vorangegangenen Abschnitt finden Sie wiederum im Projekt `org.dieschnittstelle.iam.njm_frm_mfm`. Dort sind nun vor allem das Skript `EditviewController.js` bis einschließlich der Funktion `keepEditview()` sowie die auf das `<footer>` Element folgenden Inhalte in `topicview.html` von Interesse. Mit dem darin ebenfalls enthaltenen Formular `form_einfuehrungstext` beschäftigen wir uns im Übungsprogramm zum folgenden Abschnitt.

Die Implementierungsbeispiele für die vorliegende Lerneinheit finden Sie im Projekt `org-.dieschnittstelle.iam.htm`.

Die prüfungsverbindlichen Übungen und deren Bepunktung werden durch die jeweiligen Lehrenden festgelegt.



Film



© Beuth Hochschule Berlin - Dauer: 04:56 Min. - Streaming Media 9 MB



Programmieren

Übung FRM-00

Vorbereitende Maßnahmen

- Startansicht der Anwendung ist die Ihnen bereits bekannte Titelliste. Bei Auswahl eines Elements gelangen Sie in die Topic-Ansicht für den ausgewählten Titel.
- Mittels „Long Press“ auf der Topic-Ansicht oder eines Tap/Click auf die Kopfleiste können Sie das Formular zum Erstellen/ Editieren/Löschen des `topicview` Elements für die ausgewählte Ansicht öffnen.

Bearbeitungszeit: 30 Minuten



Übung FRM-01

Beispielanwendung

Umsetzung des Formulars

Umsetzung von *Create/Update* (`EditviewController.js`)

- Wie wird für die Create/Update Aktion die Auslösung der CRUD-Datenzugriffe via `XMLHttpRequest` umgesetzt?
- Welches DOM-Ereignis löst den Datenzugriff aus?
- Wie könnte man dasselbe Ereignis auf anderem Wege mit der auszuführenden CRUD-Funktion assoziieren?
- Wo werden die Formulardaten ausgelesen, die für die Ausführung der Funktion verwendet und an den Server übermittelt werden?

Umsetzung von *Delete* (`EditviewController.js`)

- Welche Alternative wird für die Umsetzung des Delete Datenzugriffs gewählt?
- Weshalb ist für den Aufruf dieser Funktion kein Formular erforderlich?
- Woher kennt die Umsetzung der Funktion den Identifikator des zu löschenden Objekts?

`njsimpl/http2mdb.js`

- Weshalb bekommen wir einen Fehler, wenn wir aus dem `titleForm` Formular unter der in `<form>` angegebenen `action` via `POST` auf die Webanwendung zugreifen?
- Weshalb tritt dieser Fehler bei Verwendung von `XMLHttpRequest` nicht auf?

Andere Aspekte der Implementierung

Öffnen und Schließen der Editieransicht (`UIUtils.js`, `EditviewController.js`)

- Wie wird das Öffnen der Editieransicht nach `long press` veranlasst?
- Weshalb wird die Editieransicht geschlossen, wenn man außerhalb des Tab-Bereichs klickt und anderweitig nicht?
- Weshalb erfolgt das Schließen im Gegensatz zum Einblenden abrupt?

Bedienung der Tabs (`uiutils.css`, `EditviewController.js`)

- Beachten Sie, dass die Tab-Ansicht ohne Verwendung von JavaScript und allein mittels HTML und CSS umgesetzt ist. Dies erfolgt auf Basis der Vorschläge in www.sitepoint.com/css3-tabs-using-target-selector/.
- Wie wird dafür gesorgt, dass bei Öffnen des Editview sowohl der `Titel`-Tab geöffnet ist, als auch der Focus auf ein Eingabefeld der aktuellen Ansicht gesetzt wird (siehe `openEditview()`)?

Verwendung von Tabs und Umsetzung der Aktion „Zurück“ (`EditviewController.js`, `lib/navigation.js`)

- Wie wird die Vorauswahl des Titel-Elements bei erstmaligem Öffnen der Ansicht umgesetzt?
- Beachten Sie, dass bei Wechsel der Tab-Auswahl jeweils der ausgewählte Tab zum Wert von `windows.location.url` hinzugefügt wird und damit in die Browser History aufgenommen wird.

Demoversion

Bearbeitungszeit: 60 Minuten



Übung FRM-02

CRUD-Formular für Imgbox

Aufgabe

Erstellen Sie eine Formularansicht, die Ihnen das *Erstellen*, das *Aktualisieren* und das *Löschen* eines `Imgbox`-Elements bezüglich der dargestellten Topic-Ansicht ermöglicht.

Anforderungen

1. Die Ansicht soll in der mit `ImgboxTab` markierten `<section>` der Tab-Ansicht dargestellt werden, die in den Implementierungsbeispielen enthalten ist.
2. Die Ansicht soll über ein Bedienelement zum Erzeugen/Modifizieren von `Imgbox`-Elementen verfügen, das als `submit`-Input eines Formulars realisiert wird.
3. Die Ansicht soll über ein weiteres Bedienelement zum Löschen von `Imgbox`-Elementen verfügen.
4. Das Formular zum Erzeugen/Modifizieren soll über geeignete Bedienelemente zur Erstellung und Darstellung der Attribute `src`, `title` und `description` von `Imgbox` verfügen.
5. Falls noch kein `Imgbox`-Element für die dargestellte Topic-Ansicht existiert, dann soll bei Betätigung des Bedienelements aus *Anforderung 2* die `createImgbox()` Aktion aus NJM2/3 aufgerufen werden, d. h. das `Imgbox`-Element soll erstellt und eine Referenz darauf dem `topicview` hinzugefügt werden.
6. Wenn kein `Imgbox`-Element für die Ansicht existiert, dann soll das Bedienelements aus *Anforderung 3* nicht bedienbar sein.
7. Wenn ein `Imgbox`-Element für die Ansicht existiert, dann sollen bei Öffnen der Tab-Ansicht die Tabs für `imgboxTab` und `imgboxListTab` bereits vorhanden sein. Existiert kein Objekt, sind die Tabs nicht sichtbar.
8. Existiert bereits ein `Imgbox`-Element für die dargestellte Topic-Ansicht, dann sollen dessen Attribute in den Bedienelementen des Formulars aus *Anforderung 4* dargestellt werden.
9. Existiert für eine Topic-Ansicht noch kein `Imgbox`-Element, dann soll bei Öffnen der Editieransicht der Tab für das `Imgbox`-Formular nicht dargestellt werden.
10. Wenn ein `Imgbox`-Element für die Ansicht existiert, dann soll bei Betätigung des Bedienelements aus *Anforderung 2* die `updateImgbox()` Aktion aus NJM2/3 ausgeführt werden.
11. Wenn ein `Imgbox`-Element für die Ansicht existiert, dann soll bei Betätigung des Bedienelements aus *Anforderung 3* die `deleteImgbox()` Aktion aus NJM2/3 ausgeführt werden.

Bearbeitungshinweise

- Für die Umsetzung der Controller-Funktionalität des `Imgbox`-Formulars können Sie das Codegerüst `ImgboxFormViewController.js` verwenden.
- Sie können sich bei der Umsetzung an der Handhabung der CRUD-Operationen für `topicview` in den Implementierungsbeispielen orientieren.
- Alle HTML-Bestandteile, CSS-Regeln und JavaScript-Funktionen können Sie aus den Implementierungsbeispielen übernehmen.
- Zur Erleichterung der Implementierung können Sie das `editview <div>` Element zunächst in Ihrem HTML-Dokument mit `class="overlay"` markieren und die Tabs für `Objekt` darin einfügen.

- **Anforderung 6:** Dafür können Sie das Attribut `disabled` auf dem Bedienelement auf `disabled=true` setzen, wie dies auch in den Implementierungsbeispielen gemacht wird.

Demovideo

Bearbeitungszeit: 60 Minuten



Programmieren

Übung FRM-03

Imgbox-Liste

Aufgabe

Erstellen Sie eine Ansicht, die alle existierenden `Imgbox`-Elements der `imgboxs` Collection in einer Tabellenansicht darstellt.

Anforderungen

1. Die Tabelle soll in der mit `imgboxlistTab` markierten `<section>` der Tab-Ansicht dargestellt werden, die in den Implementierungsbeispielen verwendet wird.
2. Die Tabelle soll die Menge aller `Imgbox`-Elemente der `imgboxs`-Collection anzeigen, wobei für jedes `Imgbox`-Element eine Zeile verwendet wird und die Werte der Attribute `src`, `title`, `description` in jeweils einer Spalte dargestellt werden.
3. Zur Darstellung des `src` Attributs soll ein `` Element verwendet werden, in dem das durch das `src` Attribut referenzierte Bild dargestellt wird.
4. Die Breite der Tab-Ansicht soll durch die Tabelle nicht überschritten werden. Falls die Höhe der Tabelle die Höhe der Tab-Ansicht überschreitet, soll vertikales Scrolling möglich sein.
5. Wird über das Formular für `Imgbox` Elemente ein `Imgbox`-Element gelöscht, geändert oder neu hinzugefügt, soll auch die Tabelle entsprechend modifiziert werden, ohne dass die gesamte `Imgbox`-Liste neu geladen wird.

Bearbeitungshinweise

- Für die Umsetzung der Controller-Funktionalität der `Imgbox`-Liste können Sie das Codegerüst `ImgboxlistViewController.js` verwenden.
- Hinweise zur Erstellung von Tabellen in HTML finden Sie im Skript und in den von dort referenzierten Quellen.
- **Anforderung 2:** Spaltenüberschriften sind nicht erforderlich.
- Falls Sie ein stärker an Gestaltungsmerkmalen von Smartphones orientiertes Design verwenden, können Sie anstelle einer Tabelle auch eine Listenansicht umsetzen, bei der `title` und `description` in einer Spalte untereinander angeordnet werden.

Bearbeitungszeit: 60 Minuten

Wissensüberprüfung

Versuchen die hier aufgeführten Fragen zu den Inhalten der Lerneinheit selbständig kurz zu beantworten, bzw. zu skizzieren. Wenn Sie eine Frage noch nicht beantworten können kehren Sie noch einmal auf die entsprechende Seite in der Lerneinheit zurück und versuchen sich die Lösung zu erarbeiten.



Formulieren

Übung FRM-04

Formulare

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Welche beiden Typen von Anliegen können durch Formular zum Ausdruck gebracht werden
2. Was ist der grundlegende Unterschied zwischen Informations- im Ggs. zu Transaktionsanliegen?
3. In welchem Bezug stehen die beiden Anliegenstypen zu den vier CRUD Operationen?
4. Wann kann es erforderlich sein, die durch ein Formular ausgedrückten Sachverhalte zu modifizieren?
5. Was kennzeichnet Formulare als Bestandteile maschineller graphischer Nutzerschnittstellen?
6. Was sind „Formulardaten“?
7. Was wird durch Formulardaten beschrieben und worin unterscheiden sich diesbezüglich die beiden Typen von Anliegen, die durch ein Formular ausgedrückt werden können?
8. Nennen Sie fünf Usability-Anforderungen bezüglich Formularen.
9. Nennen Sie fünf funktionale Anforderungen bzw. Gestaltungsanforderungen, die für die Bereitstellung nutzerfreundlich bedienbarer Formulare umgesetzt werden müssen.
10. Hinsichtlich welcher implementierungsbezogener Merkmale unterscheiden sich Ausdrucksmittel zur Implementierung von Formularen im Rahmen einer MVC Architektur?

Bearbeitungszeit: 30 Minuten



Formulieren

Übung FRM-05

HTML Formulare und Bedienelemente

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Für welche der allgemeinen implementierungsbezogenen Merkmale zur Behandlung von Formularen stellt HTML deklarative Ausdrucksmittel zur Verfügung?
2. Nennen Sie drei Typen von Eingabeelementen, die Sie in HTML für die Umsetzung einer Auswahl aus einer eingeschränkten Menge von Alternativen verwenden können.
3. Nennen Sie drei Möglichkeiten, um in einer mittels HTML realisierten Nutzeroberfläche ohne weitere Formatierungsinstruktionen Ihrerseits einen Button/Schaltfläche zu realisieren. Was ist der Unterschied zwischen `<input>` Elementen mit `type="submit"` vs. `type="button"` innerhalb eines Formulars?

Bearbeitungszeit: 15 Minuten



Formulieren

Übung FRM-06

Formularaktionen

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Nennen Sie drei Möglichkeiten, um in JavaScript auf ein Formular des dargestellten HTML Dokuments zuzugreifen – inklusive der Möglichkeit, die Sie im Rahmen der Behandlung z. B. eines `submit` Ereignisses bezüglich des Formulars haben.
2. Welche beiden Möglichkeiten haben Sie als Entwickler, um nach Betätigung eines `submit`-Elements in einem Formular die Ausführung der Formularaktion zu unterbinden?
3. Worin resultiert die Ausführung einer Formularaktion, falls als Wert von `action` *kein* `javascript:` angegeben wird?
4. Auf welche beiden Weisen können Sie Formularaktionen asynchron und ohne Neuladen des dargestellten Dokuments ausführen?
5. Welche HTTP Methoden können Sie in Formularen selbst verwenden und welche in JavaScript Code, der aus Formularen aufgerufen wird?
6. Worin unterscheiden sich `GET` und `POST` als Werte des `method` Attributs von Formularen im Hinblick auf die Übermittlung der Formulardaten via HTTP?

Bearbeitungszeit: 30 Minuten



Formulieren

Übung FRM-07

Data Binding

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Was wird im Rahmen einer MVC Architektur als „Data Binding“ bezeichnet?
2. Unterstützt HTML5 deklaratives Data Binding?
3. Für welche beiden Typen komplexer GUI Elemente ist Data Binding besonders wichtig?
4. Können Formulare ineinander eingebettet werden? Was resultiert daraus bezüglich des Data Bindings?

Bearbeitungszeit: 15 Minuten