

MME - Verwendung von Multimedia

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

MME - Verwendung von Multimedia




Überblick und Lernziele


In den vorangegangenen Lerneinheiten haben Sie verschiedene Aspekte der Handhabung von Instanzen komplexer Datentypen in einer Client-Server Architektur kennengelernt. So haben Sie u. a. Instanzen der von uns exemplarisch betrachteten Datentypen `Topicview` und `Objekt` mittels Formularen erstellt, via HTTP an eine serverseitige NodeJS-Anwendung übertragen und sie von dort aus in einer MongoDB Datenbank gespeichert. Auch das Auslesen der Instanzen aus dieser Speicherschicht wurde durch Ihre Webanwendung initiiert und diente hier insbesondere dem dynamischen Aufbau von Ansichten der Anwendung. Wir haben außerdem gezeigt, wie mittels Multipart Formulardaten Dateiinhalte von einer Client-Anwendung an einen Server übertragen und dort gespeichert werden können - und haben diese Daten mittels der dafür vorgesehenen HTML-Elemente in die betreffenden Ansichten eingebunden.

Als darzustellende Inhalte der Ansichten unserer Anwendung wurden bisher allerdings ausschließlich „statische“ Inhalte verwendet. Neben *Standbildern*, die wir mittels `` referenzieren, waren dies auch Inhalte, die als Fragemente von HTML-Dokumenten bereitgestellt wurden und die dann in die dafür vorgesehenen Elemente der Ansichten, z. B.

`Einfuehrungstext` und `Textauszug`, eingefügt wurden. Als Erweiterung des Repertoires möglicher Inhaltstypen wird die heutige Veranstaltung zeigen, wie unter Verwendung der aktuellen Ausdrucksmittel von HTML „dynamische“ Inhalte als Bestandteil der Ansichten einer Webanwendung verwendet werden können und wie die Wiedergabe dieser Inhalte erfolgt.

Dies illustrieren wir zum einen am Beispiel von Audioinhalten, zum anderen mit Blick auf Videoinhalte, d. h. auf Bewegtbilder. Darüber hinaus werden Sie jedoch auch Ausdrucksmittel kennenlernen, um die Aufzeichnung von Medieninhalten durchzuführen. Die diesbezügliche  JavaScript API für Medistream Recording befand sich zum Zeitpunkt der Erarbeitung des Lehrmaterials im Dezember 2013 noch in Arbeit, daher werden wir hier nur die Aufzeichnung von Standbildern illustrieren, die aber auf Grundlage der APIs zur Handhabung von Videoinhalten erfolgt.

Zusätzlich zur bisherigen Verwendung von formatierten Texten und Standbildern, führt die heutige Lehrveranstaltung damit Ausdrucksmittel ein, die gemeinheitlich mit dem Stichwort „Multimedia“ belegt werden. Mit dem vorliegenden Stoff bewegen wir uns jedoch auch in einem Bereich, hinsichtlich dessen die vorgeschlagenen Standards noch nicht durchgängig durch stationäre und mobile Browser unterstützt werden. So ist z. B. die Verwendung von Untertiteln für Video, die wir anhand eines einfachen Beispiels illustrieren werden, zwar in Chrome aber evtl. noch nicht in Firefox verfügbar.

Um Bilder aufzunehmen brauchen Sie nicht nur eine Kamera auf Ihrem Entwicklungsrechner oder Smartphone, sondern auf letzterem evtl. die aktuelle  Beta-Version von Firefox. Unter Berücksichtigung möglicher verschiedener Systemvoraussetzungen auf Ihrer Seite sieht das Übungsprogramm zur vorliegenden Lerneinheit eine Auswahl aus zwei alternativen Aufgaben vor, sodass die Verfügbarkeit eines Entwicklungsrechners mit Kamera oder eines entsprechenden Smartphones für die Erbringung der Prüfungsleistung keine notwendige Voraussetzung darstellt.



Lernziele

Lernziele

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

- Statische und dynamische Inhalte in Webangeboten zu charakterisieren und im Bezug auf das Nutzungsumfeld mobiler Anwendungen zu reflektieren.
- Zu erklären welchen Stellenwert Multimedia-Elemente in aktuellen Webangeboten einnehmen und was sich hinter dem Begriff „Prosumers“ verbirgt.
- Die architektonischen Möglichkeiten für die Wiedergabe von Audio- und Videoinhalten im WWW und deren Einschränkungen zu beschreiben.
- Untertitel für Videos im Rahmen der Standards für Multimedia-Elemente in HTML umzusetzen.
- Audio- und Videoaufnahmen über JavaScript zu steuern und die Verwendung von Data URLs zu erklären.



Gliederung

Gliederung

- Hintergrund und Motivation
- Wiedergabe von Audio und Video
- Aufnahme von Standbildern
- Übungen
- Prüfungsfragen



Zeitbedarf

Zeitbedarf und Umfang

Für die Bearbeitung der Lerneinheit benötigen Sie etwa 3 Stunden Minuten. Für die Bearbeitung der Übungen der Beispielanwendung etwa 3 Stunden und für die Wissenfragen ca. 1 Stunde.

Gesamtbearbeitungsdauer: 7 Stunden

1 Hintergrund und Motivation

Bevor wir uns mit den technischen Aspekten des hier eröffneten Themengebiets beschäftigen, möchten wir eine kurze Reflexion bezüglich der hier verwendeten Begriffe bzw. des Nutzungsumfelds anstellen, welches die Verwendung der Funktionen kennzeichnet, zu deren implementatorischer Umsetzung Sie durch diese Lerneinheit befähigt werden.


1.1 Statische und dynamische Inhalte

Die Begriffe „statische“ im Gegensatz zu „dynamischen“ Inhalten haben wir oben als selbstverständlich vorausgesetzt, und vermutlich haben Sie deren Verwendung auch nicht als abwegig empfunden. Auf welcher Grundlage aber nehmen wir diese Unterscheidung vor, bezüglich derer Texte und Standbilder als „statische“, Audio und Video hingegen als „dynamische“ Inhalte bezeichnet werden? Betrachten wir dafür eine Definition, die sich in einer Publikation zum Thema Multimedia Retrieval findet:



Definition


Media types

“Media types can be classified according to the relationship with time, which leads to two classes of media types: static and dynamic (or time continuous). (...) Static media do not have a time dimension, dynamic media do. Examples of static types are alphanumeric data, images and graphics. Audio, animation and video are examples of dynamic types.”  [BBFV07]

Zeitdimension

Vermutlich finden Sie in dieser Charakterisierung Ihre Intuition bezüglich der vorzunehmenden Abgrenzung zunächst adäquat wiedergegeben. Bedenken Sie jedoch einmal das entscheidende Differenzierungskriterium der „Zeitdimension“ und die Aussage, derzufolge eine solche einem Text oder einem Standbild im Gegensatz zu einer Video- oder Audioaufzeichnung nicht zukommt. Dieses ist durchaus haltbar, es bezieht sich jedoch ausschließlich auf die Wiedergabe der betreffenden Inhalte, welche sich tatsächlich über ein Zeitintervall hin erstreckt. Texte und Bilder hingegen werden dargestellt und „sind dann da“, ohne einer Wiedergabe bzw. eines technischen „Abspielens“ zu bedürfen.

Im Sinne einer technischeren Begriffsfestlegung könnte man also die Differenzierung „dynamisch vs. statisch“ auch auf das Merkmal der Abspielbarkeit bzw. der Abspielbedürftigkeit zurückführen. Alternativ dazu könnte man unter Bezugnahme auf den Rezipienten eines Inhalts – d. h. den Betrachter eines Videos oder den Hörer einer Audiodatei – diese dynamischen Medientypen anhand eines Merkmals charakterisieren, das aus dieser Abspielbedürftigkeit resultiert. So können dynamische Inhalte aus Sicht des Rezipienten als transitorisch charakterisiert werden. Damit ist gemeint, dass die aufeinanderfolgenden Signale eines solchen Inhalts nur vorübergehend wahrnehmbar sind. Bestimmt wissen Sie den Nutzen technischer Abspielgeräte für solche Inhalte, der durch Funktionen zum Pausieren und Zurückspulen :) gegeben ist, zu schätzen. Solche Funktionen kompensieren nämlich die Defizite, die sich aus der Transitorik der Inhalte ergeben, und sie gewährleisten, dass Sie die Inhalte vollständig rezipieren können, auch wenn Sie zu einem bestimmten Zeitpunkt nicht zugegen oder nicht aufmerksam waren.

Erwähnt sei als Zuspitzung dieser Komfortfunktionen das – in seiner Bezeichnung paradoxe, wenn auch äußerst komfortable – Angebot eines „zeitversetzten Livestreams“, das Ihnen manche Fernsehsender derzeit im Rahmen ihres Internetangebots zur Verfügung stellen (siehe z. B. die Livestream Angebote auf  <http://www.tagesschau.de>).

Rezeption von Inhalten

Im Gegensatz zu dynamischen Inhaltstypen sind statische Typen wie Standbild und Text ohne Zweifel nicht transitorisch. Hier wird die gesamte Inhaltsmenge in einem Schritt wiedergegeben und Sie als Rezipient können zu jedem Zeitpunkt auf diese Inhalte zugreifen – mögen diese Inhalte auch wie z. B. in einem Buch über eine Vielzahl an Seiten verteilt sein. Letzteres Beispiel, aber auch eine Betrachtung der Vorgangs, der sich bei der Rezeption eines Bildes abspielt, wie in folgende Abbildung, zeigen aber auch, dass diesen so selbstverständlich als „statisch“ charakterisierten Inhalten durchaus eine Zeitdimension zukommt.



Abb.: Museumsbesucher als Beispiel für die Zeitdimension, die statischen Inhalten im Hinblick auf ihre Rezeption zukommt

© "... " – Fred R. Conrad für The New York Times

Siehe für Bildquelle und Entstehung der Bilder den Artikel: <http://www.bildwerk3.de>

Diese besteht darin, dass zwar die Wiedergabe der Inhalte „auf einmal“ erfolgen kann, ihre Rezeption aber durchaus als zeitlicher Vorgang aufgefasst werden muss. Eine Differenzierung statischer und dynamischer Inhalte auf Basis des Merkmals einer abwesenden bzw. vorhandenen Zeitdimension ist also nur im Hinblick auf die Wiedergabe der Inhalte gültig und gilt nicht für deren Rezeption.

1.2 Konsumption und Produktion von Inhalten

Durch Bereitstellung von Formularen haben Sie die Nutzer Ihrer Anwendung im Rahmen des Übungsprogramms dazu in die Lage versetzt, eine ursprünglich unveränderliche Menge von Inhalten, die über eine Anwendung zugreifbar gemacht wurden, durch Hinzufügung neuer Inhalte zu erweitern oder durch Modifikation bestehender Inhalte zu manipulieren. Bisher lagen diese Inhalte bereits vor, bevor sie mittels Formularen in die Anwendung übertragen oder durch externe URLs referenziert wurden.

Die aktuelle Lerneinheit und das dazugehörige Übungsprogramm wird die Bereitstellung von Inhalten für eine Anwendung dahingehend ausbauen, dass nun die Anwendung selbst Ihnen die Erstellung der Inhalte ermöglichen wird, indem sie Sie auf die Kamera des verwendeten Endgeräts zugreifen lässt. Im Hinblick auf die Aktivitäten, die die Nutzer einer Anwendung bezüglich der durch die Anwendung bereitgestellten Inhalte ausüben, soll an dieser Stelle ein Begriff thematisiert werden, den Sie evtl. gar nicht mehr kennen, da er einen Sachverhalt bezeichnet, der mittlerweile insbesondere durch die zunehmende Verbreitung von mobilen Endgeräten zu einer Selbstverständlichkeit geworden ist.

Prosumer



Es handelt sich um den Begriff des „Prosumers“. Dieser bezeichnet einen Nutzer, welcher sich nicht auf die bloße Konsumption von Inhalten beschränkt, die über Webanwendungen einem potentiell globalen Nutzerkreis zugänglich gemacht werden, sondern solche Inhalte aktiv bereitstellt und produziert. Klassische Betätigungsfelder von Prosumern sind soziale Netzwerke jeglicher Art, inklusive produkt- oder markenbezogener Nutzerforen, aber auch Blogs und Wikis als auf die Verbreitung von Inhalten orientierte Anwendungen. Die für solche Anwendungen durch Prosumer bereitgestellten Inhalte werden üblicherweise als *User generated Content* (UGC) bezeichnet.

In diesem Sinne lassen sich die gerade beschriebenen Funktionen, die Sie im Rahmen der Lerneinheiten NJM, FRM und MFM umgesetzt haben, als Funktionen charakterisieren, die – in ihrem jeweils begrenzten Umfang – eine ursprünglich auf statischen Inhalten operierende Anwendung um Use Cases erweitern, die dem Konzept des Prosumers entsprechen, indem sie die Erstellung von User Generated Content erlauben.

Prosumer und Web 2.0

Der Prosumer-Begriff kann als ein Schlüsselbegriff angesehen werden, der den Nutzer des „Web 2.0“ kennzeichnet – auf letzteren Begriff waren wir ja bereits bei der Betrachtung von Formularen und die Abgrenzung eines „Web 2.0“ von einem „Web 1.0“ Umsetzungsstils für Formulare eingegangen. Dort wurde durch „Web 2.0“ vor allem eine Reihe technischer Ausdrucksmittel bezeichnet, mit deren Unterstützung die Bedienung von Webanwendungen näher an die Nutzererfahrung nativer Anwendungen – seien es Anwendungen für stationäre oder mobile Geräte – herangeführt werden kann. Dieser technische Aspekt von „Web 2.0“ wird durchaus adäquat durch das Schlagwort AJAX zusammengefasst, das wir in Lerneinheit JSL Abschnitt 2.7 eingeführt und ausgelegt hatten.

Demgegenüber bezeichnet der Begriff des Prosumers einen Aspekt von „Web 2.0“, welcher weniger auf die technische Realisierung und die User Interface Gestaltung von Webanwendungen abhebt, sondern sich auf die Anwendungsfälle bezieht, die durch Webanwendungen unterstützt werden – etwas weiter gespannt, könnte man mit dem Begriff auch eine grundlegende Haltung bezeichnen, die ein Nutzer gegenüber „dem Web“ einnimmt.

Der Ursprung des Begriffs des Prosumers bzw. der damit bezeichneten Haltung liegt jedoch deutlich weiter zurück als die  programmatische Ausrufung des „Web 2.0“ durch Tim O'Reilly im Jahr 2005. So findet sich die Idee bereits in einem in den 20er Jahren entstandenen Text, welcher aus Sicht des Autors dieses Skripts als „medieninformatische Pflichtlektüre“ angesehen kann. Im Rahmen der Medientheorie hat er bereits kanonischen Status (siehe z. B.  [Bo190]), auch da er exemplarisch für die Vorwegnahme von Ideen ist, die in unserer „Tageswahrnehmung“ vermeintlich untrennbar mit den technischen Möglichkeiten der vergangenen 20 Jahre verbunden sind:



Hinweis

Rundfunk

„... der Rundfunk ist aus einem Distributionsapparat in einen Kommunikationsapparat zu verwandeln. Der Rundfunk wäre der denkbar großartigste Kommunikationsapparat des öffentlichen Lebens, ein ungeheures Kanalsystem, das heißt, er wäre es, wenn er es verstünde, nicht nur auszusenden, sondern auch zu empfangen, also den Zuhörer nicht nur zu hören, sondern auch sprechen zu machen und ihn nicht zu isolieren, sondern ihn in Beziehung zu setzen. Der Rundfunk müsste demnach aus dem Lieferantentum herausgehen und den Hörer als Lieferanten organisieren.“ [Bre67]

Mit der „Organisation des Hörers als Lieferant“ wird exakt das Nebeneinander der Konsumption und Produktion von Inhalten charakterisiert, welches der Begriff des Prosumers auf den Punkt bringt. Auch die Anwendungsfälle, die im weiteren Verlauf des Textes erwähnt werden erscheinen mit Blick auf existierende Videoblogs oder die zeitweilige Diskussionen über exzessive Twitter-Nutzung politischer Entscheidungsträger, aber auch hinsichtlich der unter Konsumenten durchgeführten „Debatten“ vertraut:



Hinweis

Rundfunk

„(...) zu diesen Verpflichtungen des obersten Beamten [Anm. JK: des Reichskanzlers] gehört es, regelmäßig durch den Rundfunk die Nation von seiner Tätigkeit und der Berechtigung seiner Tätigkeit zu unterrichten. (...) Er hat überdies hinaus die Einforderung von Berichten zu organisieren, das heißt die Berichte der Regierenden in Antworten auf die Fragen der Regierenden zu verwandeln. Der Rundfunk muss den Austausch ermöglichen. Er allein kann die großen Gespräche der Branchen und Konsumenten über die Normung der Gebrauchsgegenstände veranstalten, die Debatten über Erhöhung der Brotpreise, die Dispute der Kommunen. Sollten Sie dies für utopisch halten, so bitte ich Sie darüber nachzudenken, warum es utopisch ist.“

WWW als Medium

Auf technischer Ebene – wenn auch nicht im Hinblick auf die durch den Autor gemeinte politisch-gesellschaftliche Dimension – wurde diese vor gut 80 Jahren formulierte „Utopie“ mittlerweile durch das World Wide Web eingelöst, sodass wir als „*Work-in-Progress*“ Definition dessen Charakter als Medium wie folgt fassen können:



Definition

World Wide Web

Das **WWW** ist ein Distributions- und Kommunikationsmedium für digitalisierte Inhalte verschiedener Modalitäten/Inhaltstypen, die auf digitalen Speichermedien vorliegen und mittels HTTP und den darunter liegenden Netzwerkschichten übertragbar sind.

Browser und die darin ausgeführten Anwendungen sind Apparaturen für die Rezeption, Distribution und Produktion digitalisierter Inhalte verschiedener Modalitäten, die über das WWW kommuniziert werden.

Inwiefern ein Browser im Anschluss an diese Definition bereits tatsächlich als „universelles Arbeitsgerät des Prosumers“ angesehen werden kann, ist mit Blick auf den aktuellen durch Browser unterstützten Funktionsumfang bezüglich der für die „Produktion“ von Inhalten erforderlichen Aufnahmefähigkeiten nicht unstrittig. Mobilien Endgeräten und der Menge der auf ihnen laufenden Anwendungen kann die hier formulierte Auszeichnung aber mittlerweile durchaus zugeschrieben werden. Eindrücklich wird dies anhand einer Gegenüberstellung zweier Bilder, die in folgende Abbildung zu sehen sind und die die mittlerweile erreichte Durchdringung des Alltags durch Smartphones deutlich widerspiegeln.



Abb.: Consumer vs. Prosumers

© AP/DPA Quelle: www.spiegel.de

Angemerkt sei allerdings, dass zwar der örtliche Kontext, nicht aber der zeremonielle Rahmen der beiden Aufnahmen übereinstimmt. So nimmt die Menschenmenge in der Aufnahme von 2005 an der Aufbahrung des Leichnams von Papst Johannes Paul II teil. Im Bild von 2013 sind hingegen die Zuschauer zu sehen, die die Bekanntgabe von Person und Namen des gerade durch das Konklave gewählten Papstes erwarten, als welcher sich dann Jose Mario Bergoglio bzw. Papst Franziskus vorstellen wird. So legt denn der Kontext von 2005 eher als der von 2013 den Verzicht auf die aktive Nutzung der damals bereits weit verbreiteten mobilen Endgeräte nahe.

Ungeachtet der unterschiedlichen Rahmenbedingungen, in denen die beiden Bilder entstanden sind – und ungeachtet der Frage, welche der in der Aufnahme von 2013 gezeigten Nutzer tatsächlich ihre Aufnahmen öffentlich verfügbar machen werden, illustriert letztere Aufnahme die Bedeutung, die Smartphones und Apps als Gerätschaften zur Erstellung von User Generated Content zukommt. So erleichtern diese die Bereitstellung von Inhalten verschiedener Inhaltstypen und insbesondere von Multimedia-Inhalten ganz erheblich, da hier z. B. aus einer einzigen Anwendung heraus die Aufnahme von Inhalten, die Nachbearbeitung der aufgenommenen Inhalte und deren Distribution durch Hochladen an eine serverseitige Anwendungskomponente möglich ist.

Auf Grundlage der jeweils verwendeten Smartphone Plattform und deren Programmierschnittstellen können mobile Anwendungen verschiedene Teilfunktionen integrieren, die in den Ursprungsjahren des „Web 2.0“ noch weitgehend separaten Anwendungen und sogar Geräten vorbehalten waren. So mussten bis vor wenigen Jahren für den besagten Anwendungsfall z. B. Fotos aus einer Digitalkamera zunächst auf einen mehr oder weniger stationären Rechner übertragen werden, konnten dort dann, falls gewünscht, mit einer geeigneten Anwendung bearbeitet werden und konnten schließlich unter Verwendung eines Browsers und Zugriff auf die gewünschte Website für den öffentlichen Zugriff verfügbar gemacht werden.

Mobile Anwendungen erlauben aber nicht nur die Zusammenführung dieser Teilfunktionen auf einem einzigen Gerät. Sie ermöglichen bei Verfügbarkeit eines mobilen Datennetzes mit ausreichender Bandbreite darüber hinaus auch die sofortige Verfügbarmachung von Inhalten für andere Nutzer. Sie schließen damit die Produktion und Distribution von Inhalten auf eine Weise kurz, die die zunehmende Verbreitung von User Generated Content erheblich begünstigt.

Webanwendungen für Prosumer

Gegenüber dem hier geschilderten Anwendungsszenario können Anwendungen, die auf Basis standardisierter Webtechnologien entwickelt und im Browser eines mobilen Endgeräts ausgeführt werden, zumindest auf Basis unseres bisherigen Kenntnisstands lediglich auf bereits aufgenommene Inhalte zugreifen und diese distribuieren. Ausgenommen davon sind Textinhalte, die über die Eingabefelder von Formularen erstellt werden können. Die angestrebte Überwindung des Abstands zwischen nativen mobilen Anwendungen und mobilen Webanwendungen, die als roter Faden nahezu aller auf HTML gerichteter Bemühungen erkennbar ist (*als weiterer „roter Faden“ kann die Bereitstellung von Ausdrucksmitteln für semantisches Markup angesehen werden*), erstreckt sich jedoch auch auf jene Funktionalität, die erforderlich ist, um einer Webanwendung direkten Zugriff auf die für die Inhaltsproduktion nutzbaren Aufnahmevorrichtungen wie Mikrofon und Kamera zu geben.

2 Wiedergabe von Audio und Video

Derzeit stehen Ihnen grundsätzlich drei Möglichkeiten zur Verfügung, um Audio- und Video-Inhalte, die durch einen Server bereitgestellt werden, auf einem Endgerät wiederzugeben. Verwenden können Sie dafür wahlweise.

- Eine außerhalb des Browsers auf dem Endgerät laufende Anwendung, die vom Browser aufgerufen wird.
- Ein Browser-Plugin, z. B. Windows MediaPlayer oder Flash, das die Darstellung der Inhalte ermöglicht und die zum Abspielen benötigten Bedienelemente bereitstellt.
- Die HTML5-Elemente `<audio>` und `<video>`, die im Zentrum dieses Abschnitts stehen und nachfolgend als Multimedia-Elemente bezeichnet werden.

Multimedia Elemente

Bereits Anfang 2011 wurden die beiden Multimedia-Elemente durch >50% der in Gebrauch befindlichen Browser unterstützt, dennoch erscheint ihre Verwendung, wie die in folgender Abbildung dargestellte Verbreitung unter den meistgenutzten Websites zeigt, nach wie vor eher exotisch, insbesondere wenn man dem die Verwendung von Flash entgegenhält.


Websites using Shockwave Flash Embed




Websites using HTML 5 Video Audio Tags



Abb.: Gegenüberstellung der Nutzung von Flash vs. HTML

Ein Grund hierfür dürfte sein, dass das Flash-Plugin nahezu flächendeckend in Browsern verschiedener Anbieter und auf unterschiedlichen Generationen dieser Browser installiert ist. Das offensichtlichste Hindernis, das einer Verwendung von `<audio>` und `<video>` entgegensteht, dürfte angesichts dessen die Tatsache sein, dass nicht alle Videoformate bzw. alle darin verwendeten Codecs für Video und Audio von allen Browsern unterstützt werden. So wird insbesondere durch Firefox der weit verbreitete *MPEG-4* Video Codec nicht unmittelbar unterstützt, sondern ist nur dann nutzbar, wenn der Codec auf Ebene des verwendeten Betriebssystems verfügbar ist. Diese nur partielle Unterstützung kommt daher, dass die  Mozilla Foundation, die die Firefox-Entwicklung betreibt, nicht für die Lizenzgebühren für das MPEG-4 Format aufkommen will bzw. aufkommen kann. Dies gilt auch für Opera, aber nicht für Safari, Chrome und den Internet Explorer.

Vollständig unterstützt werden durch Firefox hingegen lizenzfreie Kodierungen, insbesondere die in den Formaten *ogg* bzw. *ogv* verwendeten Codecs, sowie *WebM*, die ihrerseits aber nicht durchgängig in Browsern der anderen Hersteller lauffähig sind. Siehe für die aktuelle Lage eine  Übersicht über unterstützte Medienformate. Will man also bei Verwendung der Multimedia-Elemente sicherstellen, dass eine Audiodatei oder ein Video auf einer größtmöglichen Anzahl der zugreifenden Browser abspielbar ist, müssen die betreffenden Inhalt in mehreren Formaten verfügbar sein. Zwar wird dem durch die Syntax der Multimedia-Elemente Rechnung getragen, für die Akzeptanz und Nutzung der Elemente stellt diese Problematik dennoch ein grundlegendes Hindernis dar. Stellen Sie dem z. B. die problemlose Nutzung verschiedener Bildformate gegenüber, die in `` Elementen verwendet werden können.

<audio> und <video>

Wie nachfolgend gezeigt ist, können analog zu `img` beide Multimedia-Elemente den abzuspielenden Inhalt in einem `src` Attribut auf dem Element selbst deklarieren. Alternativ kann jedoch auch eine Menge von `<source>` Elemente angegeben werden, die jeweils eine Variante des betreffenden Inhalts in einem unterschiedlichen Format referenzieren:



Quellcode

Multimedia Elemente einbinden

```
001 <!-- Verwendung eines src Attributs -->
002 <audio src="media/audio.mp3"/>
003
004 <!-- Verwendung von source Elementen -->
005 <video>
006   <source src="media/video.mp4" type=video/mp4"/></source>
007   <source src="media/video.ogv" type=video/ogg"/></source>
008 </video>
```

Bei Angabe mehrerer `<source>` Elemente wählt der Browser die erste Ressource aus, deren Format abspielbar ist. Ist dies nicht anhand eines `type` Attributs überprüfbar, dann lädt der Browser die Ressource und entscheidet anhand von deren Metadaten, ob es sich um ein abspielbares Format handelt. Diese Verwendung von `<source>` Elementen als Syntaxalternative zu einem einzigen `src` Attribut erlaubt es insbesondere, Multimedia Elemente mit Unterstützung für mehrere alternativen Formate in einem statischen HTML-Dokument zu verwenden. Es ist also nicht erforderlich, in einer Anwendung zunächst einmal zu überprüfen, welche Formate durch den betreffenden Browser unterstützt werden, und abhängig davon das Markup dynamisch zu generieren.

Bedienelemente

Eingangs haben wir erwähnt, dass Video und Audio sich von statischen Inhalten durch ihre Abspielbarkeit unterscheiden. Bei Einbindung von Multimedia-Elementen in ein HTML-Dokument ist es also erforderlich, dass die betreffenden Inhalte abgespielt und ggf. der Abspielvorgang durch den Nutzer kontrolliert werden kann. Die einfachste Lösung hierfür ist die Verwendung der beiden booleschen Attribute `controls` und `autoplay`. Bei Setzung des ersteren werden die für die Abspielkontrolle erforderlichen Elemente in der durch den Browser vorgesehenen Form zur Verfügung gestellt, und mittels `autoplay` kann – recht selbsterklärend – das automatische Abspielen von Inhalten unmittelbar nach erfolgtem Laden veranlasst werden.

2.1 Multimedia API

Die vorgestellten Ausdrucksmittel erlauben die Verwendung von Multimedia allein auf Ebene des statischen oder ggf. auch dynamisch generierten HTML-Markups. Das Verhalten beider Multimedia-Elemente in Bezug auf die darzustellenden Inhalte lässt sich aber auch in vollem Umfang durch eine JavaScript API steuern. Die API erlaubt u. a. die Entwicklung anwendungsspezifischer Bedienelemente, wie Sie es in den Implementierungsbeispielen sehen können. In Bezug auf die Wiedergabe der Multimedia-Inhalte illustrieren diese die Umsetzung der elementaren Funktionen *Starten*, *Pausieren*, *Stoppen* und *Stummschalten* sowie die Reaktion auf den Fortschritt des Abspielens, die z. B. für die Aktualisierung einer Zeitanzeige verwendet werden kann.

Die Implementierung dieser Funktionen im Skript `mediacontrols.js` ist dabei sowohl für `<audio>`, als auch für `<video>` funktionsfähig. Sie stützt sich auf das Interface [HTMLMediaElement](#), das die für beide Elemente nutzbaren Funktionen deklariert. Nachfolgend stellen wir überblicksartig die für beide Elemente definierten Zustände, die zur Zustandsänderung verfügbaren Aktionen sowie die Ereignisse vor, bezüglich derer Event Handler deklariert werden können. Für die spezifisch für `audio` und `video` verfügbaren Funktionen verweisen wir auf die Interface-Definitionen [HTMLAudioElement](#) bzw. [HTMLVideoElement](#).

Zur Verwendung der JavaScript API für Multimedia-Elemente lesen Sie zunächst – wie gewohnt – das gewünschte Element aus dem [DOM](#)-Objekt aus. Auf diesem stehen Ihnen dann u. a. die nachfolgend gezeigten Funktionen zur Verfügung, mittels derer Sie Aktionen bezüglich der wiederzugebenden Inhalte ausführen können. Erwähnt sei, dass die `load()` Funktion nur aufgerufen werden muss, falls die URL des Inhalts dynamisch gesetzt wurde und nicht bereits im initialen HTML-Dokument gesetzt war. Im letzteren Fall werden Multimedia-Inhalte automatisch geladen – entsprechend dem Laden von mittels `img` referenzierten Bildinhalte.



Quellcode

Laden und Abspielen von Videos

```
001 // greife auf das zu verwendende Element über die DOM API zu
002 var mmel = document.querySelector("video");
003
004 // lade den abzuspielenden Inhalt (bei dynamischer Setzung von src)
005 mmel.load();
006 // starte das Abspielen bzw. setze es fort
007 mmel.start();
008
009 // unterbrich das Abspielen
010 mmel.pause();
```

Die nachfolgend gezeigten Aktionen werden nicht mittels Funktionsaufrufen, sondern durch Setzen von Attributwerten ausgeführt:

```
001 // gehe zurück zum Anfang des wiederzugebenden Inhalts
002 mmel.currentTime = 0;
003
004 // schalte das Abspielen stumm
005 mmel.muted = true;
006
007 // setze die Lautstärke (Werte zwischen 0.0 und 1.0):
008 mmel.volume = 0.5;
```

Zu den aus der Ausführung von Aktionen oder aus dem Fortschreiten des Abspielvorgangs resultierenden Zuständen eines Multimedia-Elements gehören u. a. die folgenden:

```
001 if (mmel.paused) { /* Abspielen wurde pausiert */ }
002
003 if (mmel.ended) { /* Abspielen wurde abgeschlossen */ }
004
005 if (mmel.muted) { /* Abspielen wurde stummgeschaltet */ }
006
007 if (mmel.currentTime > 0) { /* Abspielen hat begonnen */ }
```

Soll eine anwendungsspezifische Behandlung von Multimedia-Elementen erfolgen, ist dafür u. a. die Reaktion auf die nachfolgend gezeigten Ereignisse mittels Setzen von Event Handlern auf dem betreffenden Element notwendig:

- `canplay`: zeigt an, dass das Abspielen der Inhalte möglich ist und Metadaten zum abzuspielenden Inhalt – z. B. `duration` – verfügbar sind.
- `timeupdate`: wird während des Abspielens fortwährend ausgelöst und ermöglicht z. B. die Aktualisierung einer Fortschrittsanzeige.
- `ended`: das Ende des abzuspielenden Inhalts wurde erreicht. Unter Verwendung dieser Ereignisse kann mittels Setzen des folgenden Event Handlers z. B. eine Endloswiedergabe umgesetzt werden:

```
001 mmel.addEventListener("ended",function(event) {
002     mmel.currentTime = 0.0;
003     mmel.play();
004 });
```

Im Hinblick auf die genannten Ereignisse ist insbesondere zu beachten, dass das `canplay` Ereignis für die Fälle, in denen die abzuspielenden Inhalte statisch referenziert werden, unabhängig vom Ladezustand des umgebenden HTML-Dokuments ausgelöst wird. Dies kann zu Irritationen führen, falls Sie eine Callback-Funktion bezüglich des `load` Ereignisses auf dem `<body>` Ihres Dokuments verwenden, um global verfügbare Variablen zu initialisieren, z. B. Attribute, die die Bedienelemente des Dokuments repräsentieren. Da `canplay` ggf. zuvor ausgelöst wird, sind die betreffenden Variablen bei Auslösung dieses Ereignisses nämlich noch nicht gesetzt.

2.2 Untertitel für Videos

Im Rahmen der Standardisierung der Multimedia-Elemente von HTML ist als weiterführende Funktionalität auch die Verfügbarmachung von Untertiteln für Videos vorgesehen. Diese Teilfunktion lässt sich ihrerseits in zwei Aspekte untergliedern. So sieht der Standardisierungsvorschlag zum einen die Verwendung des spezifischen Textformats *WebVTT* (*Web Video Text Tracks*) vor, in dem eine feingranulare Zuordnung von Texten zu Zeitintervallen in Millisekundenschärfe vorgenommen werden kann. In Texten können außerdem HTML-Elemente verwendet werden, die bei der Darstellung der Untertitel durch den Browser berücksichtigt werden. Nachfolgend sehen Sie einen Ausschnitt aus den Untertiteln der Implementierungsbeispiele, die in Chrome wie in folgender Abbildung gezeigt dargestellt werden.



Beispiel

Ausschnitt aus den Untertiteln

```
WEBVTT
1
00:00:01.000 --> 00:00:03.000
lorem ipsum
2
00:00:05.000 --> 00:00:08.000
dolor <i>sit</i> amet
3
00:00:10.000 --> 00:00:14.000
consectetur <b>adipisicing</b> elit
```



Abb.: Videowiedergabe mit Untertiteln im Chrome Browser auf einem MacBook

Sollen beim Abspielen eines Videos Untertitel verwendet werden, ist dafür die Einbindung einer WebVTT-Ressource in das HTML-Dokument bzw. das betreffende `<video>` Element erforderlich. Dafür wird ein Element `<track>` genutzt, das wie `<source>` als Kind von `<video>` auftreten kann und das anhand der Attributsetzung `kind="subtitles"` als „Untertitel-Spur“ deklariert wird. In ähnlicher Weise, wie `video` die Deklaration mehrerer `<source>` Elemente für unterschiedliche Videoformate erlaubt, können auch unterschiedliche `subtitle` Tracks für verschiedene Sprachen angegeben werden, aus denen der Browser dann die dem `Locale` des Endgeräts entsprechende Sprache aussuchen kann. Um zu kennzeichnen, dass eine Sprache ungeachtet des `Locale` als Default-Sprache betrachtet werden soll, ist ebenfalls die Setzung eines entsprechenden Attributs möglich, z. B.:



Quellcode

Untertitel Default-Sprache

```
001 <video>
002   <source src="media/video.ogv"></source>
003   <track kind="subtitles" src="media/subtitles.vtt" srclang="de"
004     label=" German" default="default"/>
005 </video>
```

Weiterführende Informationen zur [Verwendung von Untertiteln](#) und zum WebVTT Format finden Sie u. a. in der Firefox-Dokumentation. Diese scheint dem Firefox Browser allerdings insofern einen Schritt voraus zu sein, als Untertitel durch den Browser zumindest noch nicht auf allen Plattformen unterstützt werden.

Das in diesem Abschnitt behandelte Video-Element wird uns auch im folgenden Abschnitt begegnen. Dieser wird Ihnen Ausdrucksmittel vorstellen, die Sie zur Aufnahme von Standbildern über die Kamera Ihres Endgeräts anwenden können.

3 Aufnahme von Standbildern

Wir werden nun zwei Alternativen für die Aufnahme von Standbildern vorstellen, die jeweils verschiedene Aspekte der aktuellen HTML-Spezifikation inklusive der in ihrem Rahmen definierten JavaScript APIs illustrieren. Die erstere Alternative ist dabei in vollem Komfortumfang ausschließlich für mobile Endgeräte nutzbar und stützt sich auf die Typisierung von Eingabeelementen, auf die wir im Rahmen unserer Betrachtung von Formularen bereits eingegangen sind. Sie zeigt, dass auf Grundlage des „abstrakten“ `<input>` Elements von HTML und unter Betrachtung insbesondere von dessen `type` Attribut immer mehr typspezifische Eingabeelemente entwickelt werden können und dass diese immer auch für mobile Nutzerschnittstellen eine hohe Relevanz aufweisen. Mittels der zweiten Alternative hingegen können Web APIs für einen partiellen „Nachbau“ von Medieneingabefunktionen eingesetzt werden, die auf einem mobilen Endgerät allerdings üblicherweise in Form nativer Anwendungskomponenten und User Interfaces bereits verfügbar sind.

Data URLs

Eine wichtige Rolle in beiden Lösungen spielen sogenannte Data URLs. Bei diesen handelt es sich um URLs, die eine Ressource nicht nur referenzieren, sondern die Ressourceninhalte selbst enthalten. Dafür werden letztere in einem String-Format kodiert, z. B. mittels Base64 Encoding. Nachfolgend sehen Sie den Ausschnitt einer Data URL, die ein Bild beinhaltet. Hier wird nach dem URL Schema `data:`, das die URL als Data URL kennzeichnet, das Bildformat sowie das verwendete Encoding angegeben. Daran werden die kodierten Daten an die URL hinzugefügt:



Beispiel

Data URL

```
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAZAAAAEsCAYAAADtt+XCAAAGAE
ElEQVR4nES8V1cjWbquWzdnjFqVlYnwHgnkbYRMRMiH8B55BwIJL4UVZJbpPl2219rnZz/n
Ili9L+YQgnDSxfff01/GdbfWwJzVsrykx7mDqNcZGh8mkz1i7Zqw7yzSa2EYXS+9jaTcY+gO
W2UI3m5hmA1OvYdsdzGkD2+xgmR1sq4thD7CMayxjgG3fYVp9LKuBbXWxrR6meYNpDjGsIY
Z9jWV2sYwGU6vK12kDy6rxbl9j6i1su4umt9D0Fvq4wfilxuNDldHoiOHomJf7E55GhzW09
xnclriqy2TLptIFH1I5THlfYPc0RWkvqgjzEpd9hDNbxFIbiJkN4sIGofg6wegio5ElNv0z
bPrn2fCtsrDpZs27w5rPzarPw4p3m7n1FdY3
```

Die Bereitstellung von Ressourceninhalten durch eine URL kann als Grenzfall einer URL als Uniform Resource *Locator*, d. h. als *Referenz* auf eine Ressource angesehen werden. Data URLs können damit aber u. a. überall dort eingesetzt werden, wo entsprechend dem ursprünglichen Konzept von URLs Referenzen auf Ressourcen vorliegen, die durch eine serverseitige Anwendungskomponente bereitgestellt werden. Sie erweisen sich auf diese Weise insbesondere als interessant für die Umsetzung von Webanwendungen, die gar nicht notwendigerweise einen Zugriff auf serverseitige Ressourcen benötigen sollen.

In Verbindung mit den aktuellen Möglichkeiten clientseitiger persistenter Datenspeicherung, die Sie in der folgenden Lerneinheit kennenlernen werden, stellen Data URLs damit eine mögliche Grundlage für Anwendungen dar, die nutzergenerierte Inhalte verschiedener Medientypen im Offline-Modus verwenden sollen.

3.1 Verwendung von <input> mit accept

Eine sehr einfache Möglichkeit, den Zugriff auf die Gerätekamera auf mobilen Browsern zu veranlassen, stellt die Verwendung eines <input> Tags dar, dessen accept Attribut wie folgt auf den Content Type für Bilder gesetzt wird:

```
<input type="file" name="src" accept="image/*"/>
```

In stationären Browsern veranlasst diese Setzung einen hinsichtlich des eingestellten Typs gefilterten Dateiauswahldialog. Wenn Sie allerdings einen aktuellen mobilen Browser verwenden, wird Ihnen hier – wie in folgender Abbildung gezeigt – ein je nach Plattform unterschiedliches Bedienelement angeboten, das Ihnen wahlweise die Auswahl eines Fotos aus Ihrem lokalen Fotoalbum oder die Aufnahme eines neuen Fotos ermöglicht.

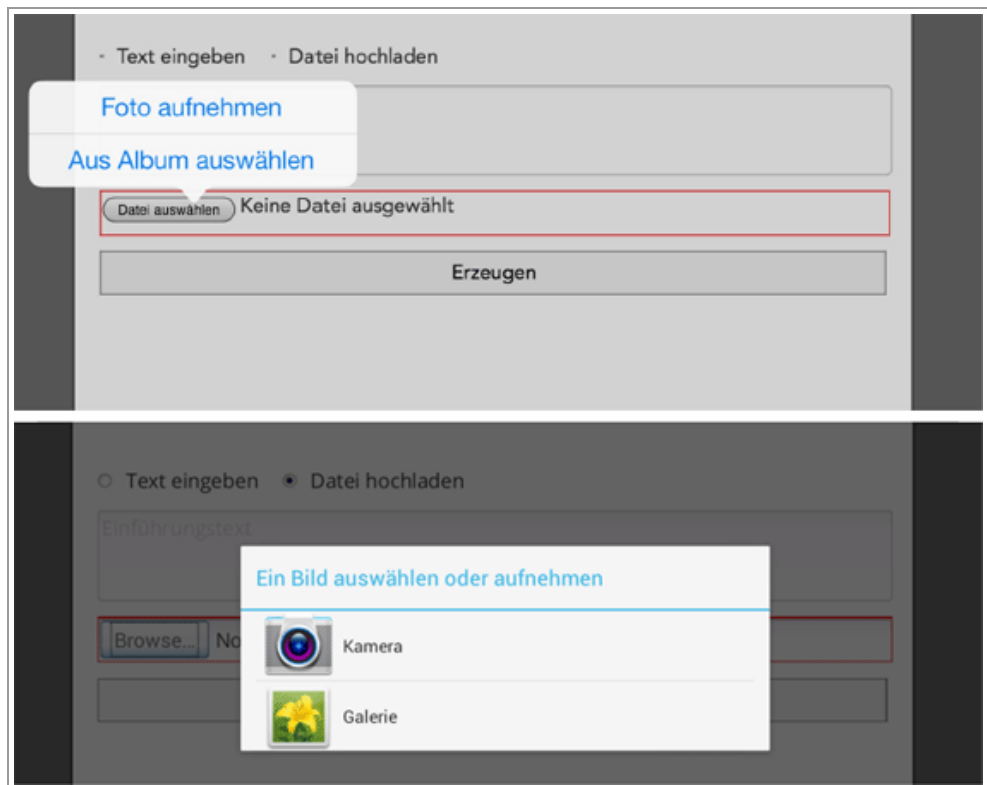


Abb.: Kamerazugriff-Realisierung auf iOS (oben) und Android (unten)

Kamerazugriff mittels Attributzuweisung accept="image/*" an ein <input type="file"> Element. Dargestellt ist die Realisierung auf iOS (oben) und Android (unten)

Falls Sie hier die letztere Option wählen, wird die native Kameraanwendung auf dem Endgerät gestartet, die Sie mit allen ihren Bedienmöglichkeiten nutzen können. Sobald Sie eine Aufnahme akzeptiert haben, gelangen Sie zurück in die Webanwendung und können hier die Befüllung des Formulars fortsetzen. Unterschiede können bezüglich des durch den Browser bereitgestellten Feedbacks bestehen: so blendet Safari (nachfolgend oben) eine Miniaturansicht des Bildes in das input Element ein, während Firefox auf Android (unten) den Dateinamen anzeigt, der die aufgenommenen Bilddaten enthält:

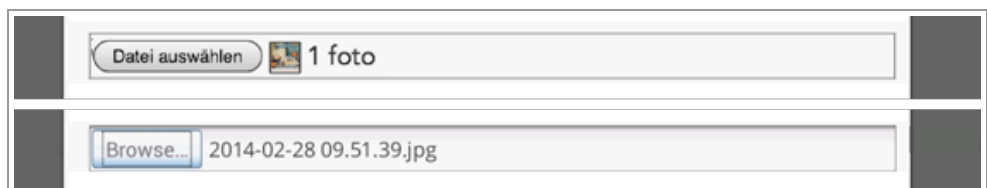


Abb.: Browserfeedback nach dem Hochladen eines Fotos (Safari oben, Firefox unten)

Zugriff auf Daten vor der
Übertragung?

Die Möglichkeiten zur Weiterverarbeitung der Bilddaten unterscheiden sich nicht von denen, die wir im Rahmen unserer Betrachtung von Multipart Formularen bereits aufgezeigt haben. Insbesondere können die Bilder an eine serverseitige Anwendung übertragen, dort persistiert und via URLs referenzierbar gemacht werden. Wäre es aber auch möglich, vor einer etwaigen Übertragung auf diese Daten zuzugreifen, um z. B. auf Ebene unserer Webanwendung das Verhalten von Safari für den Zugriff via Android „nachzubauen“, d. h. die aufgenommenen Bilddaten anstelle des – nicht notwendigerweise aussagekräftigen Dateinamens – innerhalb des Formulars als Miniaturansicht darzustellen?

Bedenken Sie, dass wir dafür einen Zugriff auf lokale Bilddaten benötigen, wie wir ihn bisher noch nicht umgesetzt haben – so konnten wir oben bei unserer Verwendung von Multipart Formularen die Bilddaten ja erst in unsere Ansichten einbinden, **nachdem** sie erfolgreich an den Server übertragen worden waren. Gibt es aber eine Möglichkeit, z. B. via „lokaler URLs“ Bilddaten aus dem Dateisystem eines Endgeräts in eine auf diesem Endgerät dargestellte Browseransicht einzubinden?

An dieser Stelle kommen die eingangs erwähnten Data URLs ins Spiel. Bei diesen handelt es sich zwar nicht um „lokale URLs“ im vorgenannten Sinne, aber um URLs, mit denen Sie zweifellos auf die gewünschten Daten zugreifen können, da diese Daten selbst in der URL enthalten sind. Wie aber können wir auf Grundlage eines `<input>` Elements mit `type="file"`, für das der Nutzer bereits eine Datei ausgewählt hat, auf den Inhalt dieser Datei zugreifen und daraus eine Data URL aufbauen? Die Lösung hierfür war in früheren Versionen von JavaScript denkbar einfach – der nachfolgende Funktionsaufruf dürfte selbsterklärend sein:



Quellcode

Data URL

```
001 // lies das img-Element aus, das für die Darstellung der Dateiinhalte
002 // verwendet werden soll
003 var img = document.getElementById("input-img-thumbnail");
004
005 // wandle den Inhalt der ausgewählten Datei in eine Data URL um
006 var dataurl = form.src.files[0].getAsDataURL();
007
008 // setze die Data URL als Wert des src Attributs auf img
009 img.src = dataurl;
```

FileReader

Die hier verwendete Funktion `getAsDataURL()` wird mittlerweile aber [www](#) nicht mehr unterstützt. Dies ist insofern nicht verwunderlich, als hier im Rahmen eines synchronen Funktionsaufrufs eine I/O Operation ausgeführt wird und damit potentiell eine Blockierung der JavaScript-Ausführung im Browser eintreten kann, falls es zu Verzögerungen bei der Ausführung dieser Operation kommt, z. B. aufgrund der Größe der via `<input>` ausgewählten Datei.

Denken Sie dafür auch zurück an unsere Ausführungen zu NodeJS und die hier beschriebene strikte Verfolgung eines Ansatzes, der insbesondere auf die asynchrone und nicht blockierende Ausführung von I/O Operationen abhebt. In diesem Sinne steht uns nun auch browserseitig eine asynchrone API für die Erzeugung einer Data URL auf Basis einer ausgewählten Datei zur Verfügung, die durch ein Objekt namens [www](#) `FileReader` bereitgestellt wird. So können Sie einer Instanz von `FileReader` eine Callback-Funktion zuweisen, die nach Abschluss des Ladevorgangs bezüglich einer ausgewählten Datei aufgerufen wird und der die ausgelesenen Daten übergeben werden. Wir nehmen nachfolgend wieder an, dass `img` das Element repräsentiert, das zur Darstellung der Dateiinhalte verwendet werden soll:




Quellcode

FileReader

```
001 // erzeuge eine Instanz von FileReader
002 var reader = new FileReader();
003
004 // deklariere eine Callback-Funktion, der die auszulesenden Daten in einem
005 // event Objekt übergeben werden
006 reader.onload = function(event) {
007     img.src = event.target.result;
008 }
009
010 // rufe die Auslesefunktion für Data URLs auf
011 reader.readAsDataURL(form.src.files[0]);
```

Eine Abhängigkeit zwischen dem Aufruf von `readAsDataURL()` und der Implementierung der Callback-Funktion, die auf das `load` Ereignis reagiert, besteht hier insofern, als der Typ der an diese Funktion übergebenen Daten von der auf `FileReader` aufgerufenen Funktion abhängt. So unterstützt die `FileReader` API nicht nur Data URLs, sondern erlaubt u. a. auch den Zugriff auf eine binäre sowie eine textuelle Repräsentation der Inhalte der angegebenen Datei. Gewöhnungsbedürftig erscheint Ihnen evtl. die Form, in der die Daten in der Callback-Funktion zugegriffen werden, nämlich über das Attribut `result` auf dem `target` eines Event-Objekts. Dieser Repräsentationsform werden wir noch weiter unten begegnen, wenn wir uns mit der API für IndexedDB beschäftigen, die ihrerseits die in `FileReader` verfolgte Designmaxime asynchroner I/O Operationen sehr konsequent umsetzt.


3.2 Verwendung von `<video>` und `<canvas>`

Als Alternative zur Verwendung der eingebauten Kamerafunktion werden wir nachfolgend zeigen, wie Sie die Aufnahme eines Kamerabildes auf Basis von Web-Technologien und ohne Rekurs auf verfügbare native Komponenten partiell „nachbauen“ können. Die Darstellung dieser Alternative ist dabei didaktisch motiviert und dient u. a. der Vermittlung von Kenntnissen bezüglich der Verwendung der  MediaCapture API, deren Standardisierung derzeit allerdings noch nicht abgeschlossen ist. Außerdem werden Sie hier weitere Verwendungsmöglichkeiten für das `<video>` Element kennenlernen, mit denen wir uns bereits im ersten praktischen Abschnitt dieser Lerneinheit beschäftigt haben.

Ob Sie im Rahmen eines konkreten Anwendungsprojekts dann tatsächlich den nachfolgend aufgezeigten Weg gehen oder aber auf die für mobile Browser gewissermaßen standardisierte Verwendung inhaltspezifischer `<input>` Elemente mit `accept` Attribut zurückgreifen, wird zum einen von den konkreten Projektanforderungen hinsichtlich der Funktionalität Ihrer Anwendung, zum anderen von den zu unterstützenden Geräteplattformen abhängen.

`<canvas>`

`<canvas>`

Ein wichtiges aktuell verfügbares Ausdrucksmittel von HTML, das wir neben dem `<video>` Element, der MediaCapture API und Data URLs zur Umsetzung der gewünschten Funktionalität benötigen, ist das  `<canvas>` Element. Dieses stellt eine 2D „Zeichenoberfläche“ bereit, vergleichbar den APIs für 2D Grafiken, die Sie auf den jeweiligen Endgeräteplattformen in Form der – gleichnamigen – `Canvas` API für Android bzw. dem `Core Graphics Framework` für iOS verwenden können. Das `<canvas>` Element kann neben den Multimedia Elementen auch als ein weiterer Baustein angesehen werden, der zur Unabhängigkeit von Webanwendungen von externen Browser-Plugins, insbesondere von Flash, beitragen soll.



Für unsere Zwecke ist insbesondere die Funktion `drawImage()` von Interesse, die uns erlaubt, ein als Bitmap repräsentiertes Bild auf die Zeichenoberfläche zu übertragen. Angegeben werden kann dafür mittels Ursprung sowie Dimensionen für Höhe und Breite der Bereich auf der `<canvas>` Oberfläche, in die die Bilddaten übertragen werden sollen – d. h. damit lässt sich eine Vergrößerung oder Verkleinerung des dazustellenden Bildes erzielen:



Quellcode

canvas Element

```
001 // lies das canvas Element aus dem DOM Objekt aus
002 var canvas = document.getElementsByTagName("canvas")[0];
003
004 // verwende ein 2D Zeichenwerkzeug
005 var drawingContext = canvas.getContext("2d");
006
007 // zeichne ausgehend vom Ursprung (10,10) die Bilddaten und verkleinere die
008 // Bildansicht auf 1/4:
009 drawingContext.drawImage(img,10,10,img.width/2,img.height/2);
```

Wie alle Zeichenoperationen bezüglich `<canvas>` wird die `drawImage()` Funktion nicht auf dem Element selbst aufgerufen, sondern auf einem `RenderingContext` Objekt. Dieses stellt gewissermaßen das „Zeichenwerkzeug“ dar, das Sie für den Zugriff auf Canvas nutzen. Bezüglich `<canvas>` stehen Ihnen zum einen einfache  2D Zeichenwerkzeuge zur Verfügung, zum anderen aber die alternative und mächtigere API  WebGL, die Sie für die Darstellung von 3D Grafiken in `<canvas>` nutzen können.

Auf die Oberfläche eines `<canvas>` können Sie jedoch nicht nur, wie hier gezeigt, schreibend zugreifen. Ihnen stehen vielmehr auch Funktionen zum Auslesen der Oberfläche zur Verfügung, so können Sie z. B. mittels `toBlob()` eine Binärrepräsentation der Inhalte erhalten. Für uns von Interesse ist die Funktion `toDataURL()`, die (bisher noch...) synchron aufgerufen wird und Ihnen als Rückgabewert eine Repräsentation der Inhalte als Data URL liefert. Das Repräsentationsformat dafür können Sie bei Aufruf der Funktion als Argument übergeben, z. B.:

```
// lies den Inhalt eines canvas als Data URL aus
var dataurl = canvas.toDataURL("image/png");
```

Nachfolgend werden wir zeigen, wie diese durch `<canvas>` bereitgestellten Grundfunktionen in Verbindung mit der Media Capture API zur Aufzeichnung von Kamerabildern eingesetzt werden können.

Media Capture API

Media Capture API

Die Media Capture API legt fest, wie aus JavaScript heraus der Zugriff auf die Aufnahmeverrichtungen für Video und Audio erfolgt, über die ein Endgerät verfügen kann. Wichtigste Funktion hierfür ist die Funktion `navigator.getUserMedia()`, die ggf. noch browserspezifisch zugegriffen werden muss, d. h. via `mozgetUserMedia()`, `webkitgetUserMedia()` etc. Mittels eines Funktionsarguments kann beim Aufruf angegeben werden, ob Video und/ oder Audio aufgenommen werden soll:



Quellcode

Aufnahme von Video und/oder Audio

```
001 // instantiiere die getMedia Funktion browserübergreifend
002 navigator.getMedia = (navigator.getUserMedia || navigator.
003   webkitgetUserMedia || navigator.mozgetUserMedia || navigator.
004   msgetUserMedia);
005
006 // überprüfe, ob die Funktion überhaupt zur Verfügung steht.
007 if (!navigator.getMedia) {
008   alert("media capture is not supported by your browser!");
009 } else {
010   // rufe getMedia() auf und übergib die gewünschten Aufnahmemodalitäten
011   // sowie Callbacks für Erfolgs- und Fehlerfall
012   navigator.getMedia({audio: true, video: true}, onUserMediaSuccess,
013     onUserMediaFailure);
014 }
```

Dass der Zugriff auf die Eingabeverrichtungen des Endgeräts als I/O Zugriff asynchron ausgeführt wird und entsprechend ein Callback angegeben werden muss, dürfte hier nicht verwundern. Eine nicht-blockierende Ausführung der Funktion ist im vorliegenden Fall aber umso wichtiger, da die Dauer bis zum Aufruf der Callbacks nicht allein durch die zugrunde liegende Hardware bestimmt wird, sondern der Nutzer in den Zugriff mit einbezogen wird.

So löst der Aufruf von `getUserMedia()` (wir verwenden hier und nachfolgend den als Standard vorgesehenen Namen) einen nicht modalen, d. h. nicht blockierenden Popup-Dialog aus, in dem der Nutzer den Zugriff ggf. mit einer Auswahl der zu nutzenden Eingabealternativen bestätigen oder ihn verweigern kann:

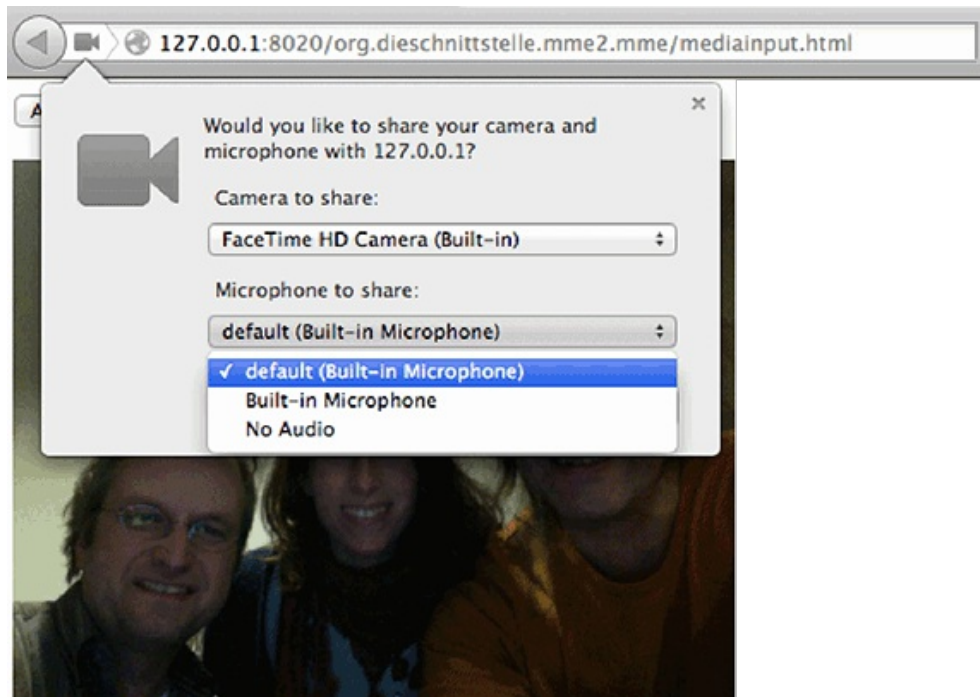


Abb.: Popup-DialogEingabealternativen

Bestätigt der Nutzer den Zugriff, wird die beim Aufruf übergebene Callback-Funktion aufgerufen und bekommt ein `MediaStream` Objekt übergeben. Dieses umfasst, je nach Angebot bzw. Auswahl durch den Nutzer ggf. mehrere Tracks für Audio und Video – zur Vertiefung hierfür verweisen wir auf die Dokumentation der [MediaStream API](#). Für die hier umzusetzende Funktion ist zunächst einmal nur die Tatsache relevant, dass dieses `MediaStream` Objekt einer Funktion übergeben werden kann, die uns eine URL bezüglich des betreffenden Streams erstellt. Diese URL kann dann überall dort verwendet werden, wo HTML den Einsatz von URLs bezüglich dynamischer Content-Typen vorsieht, z. B. als Wert des `src` Attributs von `<video>` oder `<audio>` Elementen. Nachfolgend ist die Attributsetzung für ein `<video>` Element gezeigt:



Quellcode

Attributsetzung für ein `<video>` Element

```
001 // success callback für getUserMedia()
002 function onUserMediaSuccess(localMediaStream) {
003     // lies das video Element aus
004     var videoel = document.getElementsByTagName("video")[0];
005
006     // ermittle eine URL für den Stream
007     var videosrc = window.URL.createObjectURL(localMediaStream);
008
009     // weise die URL dem video Element zu
010     videoel.src = videosrc;
011 }
```

Ab dem Zeitpunkt der Attributsetzung bezüglich `src` steuert dann das `<video>` Element die Wiedergabe des Inputstreams von Kamera und ggf. Mikrofon. Falls das Element auf `autoplay` gesetzt ist, erfolgt die Wiedergabe ohne weitere Nutzerinteraktion. Insbesondere sind über das `<video>` Element – sei es mittels der Default-Kontrollleiste oder eigener Bedienelement – aber auch diejenigen Funktionen verfügbar, die wir eingangs in Bezug auf diese Elemente vorgestellt haben und die nicht an die Speicherbarkeit des Streams gebunden sind. Beispielsweise kann mittels `pause()` die Livebilddarstellung angehalten werden oder durch Setzen des `muted` Attributs eine Stummschaltung erfolgen.

Um die gewünschte Kamerafunktion umzusetzen, wissen wir nun also, wie wir mittels `getUserMedia()` auf das Kamerabild zugreifen können. Wir wissen auch bereits, wie wir die Inhalte eines `<canvas>` Element als Data URL erhalten können. Uns fehlt also nur noch ein Zwischenschritt, nämlich die Übertragung des Videobilds auf das `<canvas>` Element. Dafür können wir die oben bereits gezeigte Funktion `drawImage()` auf dem Rendering Context von `<canvas>` nutzen und dieser das `video` Element übergeben. Dargestellt wird auf dem `<canvas>` dann das zum Zeitpunkt der Übergabe aufgenommene Bild.

Um die Seitenverhältnisse des Kamerabildes zu bewahren, sollten Höhe und Breite des `<canvas>` Elements zuvor in Abhängigkeit von den Dimensionen des `<video>` Elements gesetzt werden. Nachfolgend zeigen wir, wie die Übertragung des `<video>` Bildes auf `<canvas>` entsprechend diesen Überlegungen im Rahmen eines Event Handler auf einem „Auslöse“-Bedienelement veranlasst werden kann:



Quellcode

Event Handler auf einem Auslöse-Bedienelement

```
001 // lies canvas, video sowie ein Auslöse-Element aus dem DOM aus
002 var videoel = document.getElementsByTagName("video")[0];
003 var canvasel = document.getElementsByTagName("canvas")[0];
004 var trigger = document.getElementById("camera-trigger");
005
006 // weise dem Element einen Event Handler für click zu
007 trigger.onclick = function() {
008     // übertrage die Dimensionen von video auf canvas
009     canvasel.height = videoel.height;
010     canvasel.length = videoel.length;
011
012     // übertrage das Videobild in voller Größe auf canvas
013     canvasel.getContext(2d).drawImage(videoel, 0, 0, canvasel.width,
014     canvasel.height);
015 }
```

Beachten Sie, dass die Darstellung der beiden Elemente nicht, wie in den Implementierungsbeispielen und folgender Abbildung links gezeigt, nebeneinander erfolgen muss.



Abb.: Alternative Realisierung der Bildaufnahme mittels `<video>` und `<canvas>`

Links sind `<video>` Element (rechts oben) und `<canvas>` (links unten) getrennt voneinander dargestellt und werden um zwei `` Elemente ergänzt, in die die aus dem `<canvas>` ermittelte Data URL übertragen wird. Rechts überlagert das `<canvas>` Element nach erfolgter Aufnahme das bis zum Aufnahmezeitpunkt im Vordergrund dargestellte `<video>` Element.

Sie können zur Umsetzung einer „Standbildaufnahme“-Funktion auch ein auf dem vorstehenden Weg befülltes `<canvas>` Element mittels CSS über das `<video>` Element platzieren, wie in der Abbildung rechts zu sehen ist, oder aber vor Befüllen von `<canvas>` die Bildwiedergabe mittels `<pause()>` anhalten und das `<canvas>` Element dahinter anordnen. Zur Umsetzung einer Kamera-Funktion können Sie dann z. B. zwei Aktionen anbieten, die dem Nutzer das „Speichern“ oder „Verwerfen“ des aufgenommenen Bildes ermöglichen und im ersten Fall mittels der Funktion `toDataURL()` auf `<canvas>` die Bilddaten als Data URL auslesen, wie wir es oben gezeigt haben. Wie Sie diese Daten weiter behandeln, obliegt Ihren Vorstellungen.

Möglich ist in jedem Fall auch die Darstellung einer Miniaturansicht in einem Formular, z. B.:

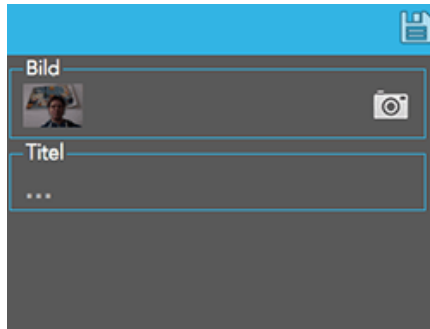


Abb.: Formular-Miniaturansicht

Die Schritte, die bei der Aufnahme von Standbildern auszuführen sind, lassen sich wie folgt noch einmal zusammenfassen:

1. Erstellung des Kamera-Displays

- (a) Zugriff auf den *Media Stream* der *Gerätekamera*
- (b) Ermittlung einer *URL* für den *MediaStream*
- (c) Setzen der URL als *src* Attribut eines *<video> Elements*

! das *<video>*-Element übernimmt dann die Aufgabe des *Displays*

2. Aufnahme des Bildes

- (a) Übertragung der Inhalte des *<video>*-Elements in ein *<canvas>*-Element
- (b) Umwandlung der Inhalte des Canvas-Elements in eine *Data URL*

! Die Data URL enthält die *Bilddaten*.

Wie hier gezeigt, können Sie mittels der *Media Capture API* aus einem Browser heraus also auf die Eingabevorrichtungen eines Endgeräts zugreifen. Dem Nutzer einer Webanwendung ermöglichen Sie damit den unmittelbaren Zugriff auf die visuelle Umgebung seines Nutzungskontexts, ohne das User Interface der verwendeten Anwendung verlassen und ggf. dahin zurückkehren zu müssen.

Wie bereits im Fall des Zugriffs auf die native Nutzeroberfläche der Kamera mittels geeigneter *<input>* Elemente gewährleisten Sie damit eine weitgehend bruchlose Nutzererfahrung, wie sie Nutzern aus der Verwendung nativer Anwendungen auf mobilen Endgeräten vertraut ist – und wie wir sie eingangs als eine erhebliche Vereinfachung für die Erstellung von nutzergenerierten Inhalten beschrieben haben.

Erwähnt sei, dass in unserer Verwendung der Media Capture API die Aufnahme der Media Streams zum Zweck der Speicherung der Inhalte jedoch nur vermittelt erfolgt, indem wir den Inhalt eines *<video>* Elements in ein *<canvas>* Element übertragen und daraus ein Standbild als Data URL extrahieren. Dieses Standbild kann dann z. B. mittels eines ** Elements in ein HTML Dokument eingebunden werden. Eine Speicherung der Streams selbst in Form von abspielbaren Inhalten, die ihrerseits mittels *<audio>* oder *<video>* Elementen wieder gegeben werden können, ist mit den uns bisher bekannten Mitteln jedoch nicht möglich. Eine [Media Stream Recording API](#), die letzteres ermöglichen soll, befindet sich zum Zeitpunkt der Erstellung des vorliegende Lehrmaterials noch in Arbeit.

Mit Data URLs haben wir hier bereits ein Ausdrucksmittel kennengelernt, das wir verwenden können, um die Ansichten einer auf einem Browser ausgeführten Webanwendung unter Verwendung von nutzergenerierten Inhalten aufzubauen, ohne auf den Einsatz einer serverseitigen Anwendung zur Bereitstellung der darzustellenden Inhalte angewiesen zu sein. Welche Mittel uns zur Verfügung stehen, um nutzergenerierte Inhalte über einen einzelnen Nutzungsfall einer Anwendung hinaus dauerhaft auf einem Endgerät zu speichern, werden wir in der folgenden Lerneinheit aufzeigen.

Zusammenfassung

- Die verfügbaren Standards für die Verwendung von Multimedia werden noch nicht durchgängig durch stationäre und mobile Browser unterstützt.
- Der Begriff des „Prosumers“ bezeichnet einen Nutzer, welcher sich nicht auf die bloße Konsumtion von Inhalten beschränkt sondern solche Inhalte aktiv bereitstellt und produziert.
- Derzeit stehen grundsätzlich drei Möglichkeiten zur Verfügung, um Audio- und Video-Inhalte, die durch einen Server bereitgestellt werden, auf einem Endgerät wiederzugeben: Eine außerhalb des Browsers auf dem Endgerät laufende Anwendung, ein Browser-Plugin und die HTML5-Elemente `<audio>` und `<video>`.
- Ein Hindernis, das einer Verwendung von `<audio>` und `<video>` entgegensteht ist, dass nicht alle Videoformate bzw. alle darin verwendeten Codecs für Video und Audio von allen Browsern unterstützt werden.
- Das Verhalten beider Multimedia-Elemente in Bezug auf die darzustellenden Inhalte lässt sich in vollem Umfang durch eine JavaScript API steuern.
- Eine einfache Möglichkeit, den Zugriff auf die Gerätekamera auf mobilen Browsern zu veranlassen, stellt die Verwendung eines `<input>` Tags dar.
- Mittels der Media Capture API kann aus einem Browser heraus auf die Eingabevorrichtungen eines Endgeräts zugegriffen werden. Dem Nutzer einer Webanwendung wird damit der unmittelbare Zugriff auf die visuelle Umgebung seines Nutzungskontexts ermöglicht, ohne das User Interface der verwendeten Anwendung verlassen und ggf. dahin zurückkehren zu müssen.

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

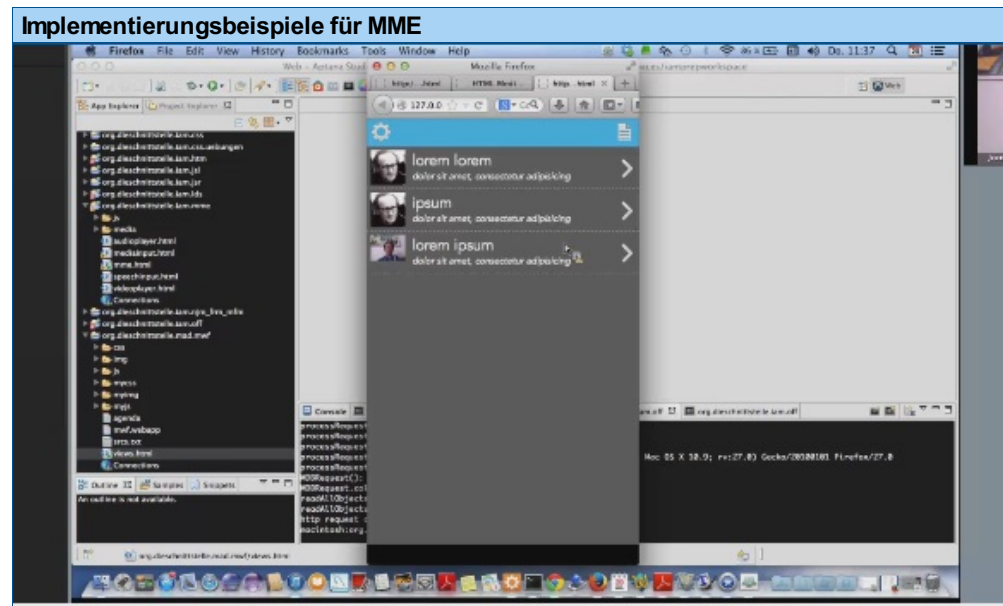
Übungen

Im Projekt `org.dieschnittstelle.iam.mme` finden Sie die Implementierungsbeispiele für die vorliegende Lerneinheit. Neben den hier besprochenen Ausdrucksmitteln für die Wiedergabe und Aufnahme von Multimedia finden Sie dort in `speechinput.html` zusätzlich ein Beispiel für die Verwendung des Google Spracherkenners, für das Sie eine aktuelle Version des Chrome Browsers benötigen.

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



Film



© Beuth Hochschule Berlin - Dauer: 05:31 Min. - Streaming Media 12 MB



Übung MME-01

Beispielanwendung

audioplayer.html/videoplayer.html

- Fügen Sie die Booleschen Attribute `autoplay` und `controls` zu den Multimedia-Elementen hinzu und überprüfen Sie deren Funktion durch Neuladen der jeweiligen HTML-Dokumente.

mediacontrols.js.togglePlay()

- Was bewirkt die Setzung von `media.currentTime = 0;?`

mediacontrols.js.mediaLoaded()

- Wo wird festgelegt, dass die Funktion `mediaLoaded()` aufgerufen werden soll?
- Warum wird in `mediaLoaded()` evtl. `initialiseView()` aufgerufen?
- Was passiert, wenn dieser Aufruf auskommentiert wird?

mediainput.js

- Welche Aufgabe hat die erste Codezeile, in der `navigator.getMedia` instantiiert wird?
- Was ist der Wert von `navigator.getMedia` für den Fall, dass der Browser Media Capture unterstützt?

mediainput.js.storeMedia()

- Wie wird der Inhalt des `<video>` Elements auf das `<canvas>` Element übertragen?
- Wie könnte hier eine vergrößerte Darstellung des Videobilds im `<canvas>` mit vierfacher Bildfläche herbeigeführt werden?

mediainput.js.captureAndPlaybackMedia()

- Weshalb wird bei Auswahl von *Video* immer eine Bildaufzeichnung mit Ton angeboten?
- Was müssen Sie tun, um nur eine Bildaufzeichnung zu initiieren?
- Welche Anweisung führt dazu, dass der Kamerastream im `<video>`-Element dargestellt wird?

Bearbeitungszeit: 30 Minuten

Die Übungen MME-02a und MME-02b sind alternativ – da für MME-02a die Verfügbarkeit einer Webcam vorausgesetzt wird.



Übung MME-02a

Aufnahme eines Bildes für Imgbox

Aufgabe

Erweitern Sie das CRUD Formular für Imgbox-Elemente um eine Option, die dem Nutzer die Aufnahme eines Bildes ermöglicht, und erstellen Sie das Imgbox-Elemente unter Verwendung des aufgenommenen Bildes.

Anforderungen

1. Zur Bildaufnahme soll die Media Capture API verwendet werden.
2. Die Bildaufnahme soll als „Kamera-Ansicht“ umgesetzt werden, die die Editier-Ansicht mit Tab, Formular etc. überlagert.
3. Es soll dafür kein Neuladen eines HTML-Dokuments erfolgen.
4. Nach Aufnahme des Bildes soll dieses dem Nutzer angezeigt werden.
5. Der Nutzer soll dann auswählen können, ob er das Bild übernehmen oder verwerfen möchte.
6. Übernimmt der Nutzer das Bild, dann wird die Kamera-Ansicht ausgeblendet und das Formular wieder sichtbar. Die Bilddaten werden dem Formular in geeigneter Form übergeben.
7. Übernimmt der Nutzer das Bild nicht, dann wird die Kamera-Ansicht ausgeblendet und das Formular wird ohne weitere Datenübergabe sichtbar.
8. Im Gegensatz zu den Implementierungsbeispielen soll das `<canvas>` Element, in dem das aufgenommene Bild angezeigt wird, deckungsgleich mit dem `<video>` Element sein und dieses verdecken.
9. Bei Erstellung oder Aktualisierung des Imgbox-Elements werden die neu aufgenommenen Bilddaten übertragen und serverseitig in der MongoDB persistiert.
10. Alle implementierten Funktionen bezüglich Imgbox sind in vollem Umfang auch für aufgenommene Bilder funktionsfähig.

Bearbeitungshinweise

- **Anforderung zu 8:** Dafür können `<canvas>` und `<video>` mit `position:absolute;` und identischem Ursprung positioniert werden. Der sichtbare „Vordergrund“ ist u. a. mittels Verwendung der Style-Property `z-index` kontrollierbar.
- **Anforderung zu 9:** Sie können das Imgbox-Element mit einer Data URL als `src` übertragen und persistieren. Dafür ist kein Multipart-Request erforderlich.
- Sie brauchen nur den Fall der Imgbox-Erstellung mit Kamerabild zu berücksichtigen. Es ist nicht erforderlich, dass einem bereits mit einem „normalen“ Bild erstellten Objekt im Rahmen eines Updates ein Kamerabild zugewiesen werden kann.

Bearbeitungszeit: 60 Minuten

**Übung MME-02b****Verwendung eines Videos für Imgbox****Aufgabe**

Erlauben Sie das Hochladen von Videos anstelle von Bildern für Imgbox-Elemente und erstellen Sie das Imgbox-Elemente mit dem Startbild des Videos.

Anforderungen

1. Zur Umsetzung der Funktionalität soll auf Ebene des Formulars kein neues Bedienelement eingeführt werden.
2. Die „Extraktion“ des Startbilds aus dem Video soll clientseitig nach erfolgreichem Hochladen des Videos erfolgen.
3. Nach der Extraktion soll das Imgbox-Element mit dem extrahierten Startbild erstellt werden.
4. Der Extraktionsvorgang darf keine Eingabe des Nutzers erfordern, darf für den Nutzer aber sichtbar sein.
5. Im Gegensatz zu den Implementierungsbeispielen soll das für die Extraktion verwendete `<canvas>` Element, in dem das aufgenommene Bild angezeigt wird, deckungsgleich mit dem zu verwendenden `<video>` Element sein und dieses verdecken.
6. Alle implementierten Funktionen bezüglich Imgbox sind in vollem Umfang auch für Elemente mit Video funktionsfähig.

Bearbeitungshinweise

Nutzen Sie für die „Extraktion“ des Startbilds aus dem Video dasselbe Verfahren, das wir für die Aufnahme eines Standbilds mit der Kamera verwenden.

Anforderung 3: Wenn Sie ein Video hochladen, dann wird Ihnen im Rückgabeobjekt des Servers der Medientyp als Wert des `mediatype` Attributs angegeben. Davon können Sie abhängig machen, ob Sie eine Extraktion vornehmen oder nicht.

Anforderung 4 und Anforderung 3: Zur Umsetzung dieser Anforderungen können Sie für die Imgbox-Erstellung einen `XMLHttpRequest` mit `FormData` verwenden.

Anforderung 5: Dafür können `<canvas>` und `<video>` mit `position:absolute;` und identischem Ursprung positioniert werden. Der sichtbare „Vordergrund“ ist u. a. mittels Verwendung der Style-Property `z-index` kontrollierbar.

Sie brauchen nur den Fall der *Imgbox-Erstellung* mit Video zu berücksichtigen. Es ist nicht erforderlich, dass einem bereits mit einem „normalen“ Bild erstellten Imgbox-Element im Rahmen eines Updates ein Video zugewiesen werden kann.

Bearbeitungszeit: 60 Minuten

**Übung MME-03****Abspielen von Videos****Aufgabe**

Bei Klick/Touch auf die Bilddarstellung von Imgbox-Elementen soll die bestehende Topicview- Ansicht durch eine Videoplayer-Ansicht ersetzt werden.

Anforderungen

1. Das Abspielen des Videos soll keine weitere Nutzereingabe erfordern.
2. Der Videoplayer soll die Standard-Kontrollleiste des Browsers verwenden.
3. Die Hintergrundfarbe des Videoplayers soll Schwarz sein.
4. Die Videoplayer-Ansicht soll nur den Hauptbereich der Topicview-Ansicht ersetzen. Header und Footer sollen unverändert bleiben.
5. Bei Betätigung des „Zurück“-Buttons soll das Abspielen des Videos gestoppt und wieder die Topicview-Ansicht angezeigt werden.
6. Der Übergang zum Videoplayer soll mit Aus- und Einblenden umgesetzt werden.
7. Der Übergang zwischen den Ansichten soll ohne Neuladen von HTML Dokumenten umgesetzt werden.

Bearbeitungshinweise

- Sie können ein geeignetes Video Ihrer Wahl verwenden.

Bearbeitungszeit: 30 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.

**Übung MME-04****Hintergrund**

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

1. Inwiefern trifft die Charakterisierung von Bildern und Texten als „statischen“ Inhalten nur bedingt zu?
2. Was ist mit dem Begriff des „Prosumers“ in Bezug auf die Nutzung des WWW gemeint?
3. In welchem Zusammenhang stehen die beiden Begriffe „Prosumer“ und „User Generated Content“?
4. Was zeichnet native mobile Anwendungen hinsichtlich ihrer Eignung für Prosumer in der Regel gegenüber Webanwendungen aus?

Bearbeitungszeit: 15 Minuten



Formulieren

Übung MME-05

Wiedergabe von Audio und Video in Webanwendungen

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

1. Welche drei „architektonischen“ Möglichkeiten bestehen derzeit für die Wiedergabe von Audio- und Videoinhalten aus dem WWW?
2. Was ist ein Hindernis bezüglich der breiteren Nutzung der HTML5 Multimedia-Elemente?
3. Welche Einschränkung weist Firefox bezüglich der Multimedia-Elemente auf und welche Mehraufwände entstehen dadurch ggf.?
4. Wie können alternative Medienformate, die ggf. für verschiedene Browser erforderlich sind, bei der Verwendung der Multimedia-Elemente behandelt werden? Was ist damit nicht erforderlich?
5. Was ist `HTMLMediaElement`?
6. Was müssen Sie tun, um bei Verwendung der Multimedia-Elemente auf den Vorgang des Abspielens eines Videos reagieren zu können, z. B. um eine Fortschrittsanzeige zu aktualisieren?
7. Was ist WebVTT?

Bearbeitungszeit: 20 Minuten



Formulieren

Übung MME-06

Aufnahme von Standbildern

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

1. Welche Voraussetzungen müssen zur Aufnahme von Audio und Video mittels JavaScript gegeben sein?
2. Welche Schritte sind erforderlich, um unter Verwendung von `video` und `canvas` ein Standbild aufzunehmen?
3. Welche Rolle spielt das `<canvas>` Element bei der Erstellung eines Standbilds unter Verwendung der Kamera?
4. Was sind Data URLs?
5. Welches HTML-Element stellt Ihnen seine Inhalte als Data URLs zur Verfügung?

Bearbeitungszeit: 15 Minuten