Titel Titel Titel Titel

Untertitel Unter btitel Untertitel Untertitel

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Maria Mustermann

1234567

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Vorname Nachname

Zweitprüfer: Prof. Vorname Nachname

Hamburg, 2. 2. 2020

Inhaltsverzeichnis

Abstract

Form and layout of this LaTeX-template incorporate the guidelines for theses in the Media Technology Department „Richtlinien zur Erstellung schriftlicher Arbeiten, vorrangig Bachelor-Thesis (BA) und Master-Thesis (MA) im Department Medientechnik in der Fakultät DMI an der HAW Hamburg“ in the version of December 6, 2012 by Prof. Wolfgang Willaschek.

The thesis should be printed single-sided (simplex). The binding correction (loss at the left aper edge due to binding) might be adjusted, according to the type of binding. This template incorporates a binding correction as BCOR=1mm (suitable for adhesive binding) in the LaTeX document header.

This is the english version of the opening abstract (don't forget to set LaTeX's language setting back to ngerman after the english text).

Zusammenfassung

Diese LaTeX-Vorlage berücksichtigt in Form und Layout die Vorgaben für Abschlussarbeiten im Department Medientechnik „Richtlinien zur Erstellung schriftlicher Arbeiten, vorrangig Bachelor-Thesis (BA) und Master-Thesis (MA) im Department Medientechnik in der Fakultät DMI an der HAW Hamburg“, Fassung vom 6. Dezember 2012 von Prof. Wolfgang Willaschek.

Der Ausdruck soll einseitig erfolgen (Simplex). Je nach Bindung ist ggf. die Bindekorrektur (Verlust am linken Seitenrand durch die Bindung) noch anzupassen. In dieser Vorlage ist eine Bindekorrektur im header der LaTeX-Datei mit BCOR=1mm für Klebebindung eingestellt.

Das ist die deutsche Version der vorangestellten Zusammenfassung. Beide Versionen – englisch und deutsch – sind verbindlich!

1 Einleitung

1.1 Motivation

„There is no cloud it's just someone else's Computer“ - eigentlich ein ganz triviales Statement, doch es wurde zu einem Internetphänomen, auch „Meme“ genannt, weil die Internetindustrie es geschafft hat, für einen Benutzer transparent werden zu lassen, dass hinter so manchem Dienst sich in der Realität ein ganzes Rechenzentrum befindet.

Die große Rechenleistung, die von jedem Ort, jeder Zeit verfügbar ist, machte auch eine Breite verschiedener portabler Anzeigegeräte ubiquitär. Die Werbemarketingspezialisten sprechen von einem „Second Screen“, aber in Wirklichkeit ist jedes andere internetfähige Gerät gemeint, welches parallel zum laufenden Fernsehprogramm genutzt wird. Und bei der Auswahl aus Netbook, Tablet, Phablet, Smartphone, Smartwatch ist bei manchem Anwender die Zahl dieser Geräte längst über zwei. Viele kleine Applikationen sollen diese portablen Geräte zu intelligenten persönlichen Assistenten machen. (Techcrunch) spricht sogar von einem neuen Software Goldrausch, der in den letzten 7 Jahren stattgefunden hat.

„Like all gold rushes, they must come to an end. It is clear that everyone close to technology is suffering from what the market is calling “app fatigue.” If it wasn't already hard enough to differentiate your app from the millions of others in the app store, it's now becoming even harder. From a consumer perspective, there are just too many apps. New apps, by in large, are not providing nearly enough value for consumers to come back, and most simply replicate existing experiences with a story of a better design. Apps are not an order of magnitude better than their predecessor; thus, adoption drops off as quickly as it started.“Schippers (2016)

Es stellt sich daher eine Situation dar, in der die Anwender zwar die schnelle Reaktionsfähigkeit nativer Applikationen zu schätzen wissen, jedoch nicht sofort bereit sind weitere Software auf ihre Geräte zu installieren. Und das macht eine bestimmte Applikation wieder populärer den je - den Webrowser.

Auch im Zeitalter der Smartphone hat sich an den Grundprinzipien und Protokollen, die das World Wide Web seit 1991 zu Nutze macht, nicht viel geändert. Eine sog. Rich Internet Applikation wird vom Browser auf die gleiche Art und Weise geladen, wie die allererste HTML Webseite. Lediglich eine grundlegende Neuerung kam im Jahr 1995 hinzu. Netscape ermöglichte es den Entwicklern mehr Interaktivität durch Auslieferung vom Script Code in die statischen Webseiten einzubauen und leitete den Aufstieg von JavaScript - gegenwärtig einer der populärsten Programmiersprachen. Allerdings ist diese Berühmtheit ganz und gar nicht dadurch entstanden, dass die Sprache eine besonders elegante Erfindung war. Es war einfach die einzige von Browsern von Haus aus unterstützte Option. Ganz im Gegenteil, JavaScript basiert auf einigen schlechten Entscheidungen. Ein populäres Fachbuch nennt sich auch daher nicht umsonst - ̈JavaScript - the good parts.̈

„JavaScript is a language with more than its share of bad parts. It went from nonexistence to global adoption in an alarmingly short period of time. It never had an interval in the lab when it could be tried out and polished. It went straight into Netscape Navigator 2 just as it was, and it was very rough. When Java™ applets failed, JavaScript became the “Language of the Web” by default. JavaScript’s popularity is almost completely independent of its qualities as a programming language.

Fortunately, JavaScript has some extraordinarily good parts. In JavaScript, there is a beautiful, elegant, highly expressive language that is buried under a steaming pile of good intentions and blunders. The best nature of JavaScript is so effectively hidden that for many years the prevailing opinion of JavaScript was that it was an unsightly, incompetent toy.“(Crockford 2008: S. 2)

Bei vielen Webanwendungen beschränkt sich daher heutzutage die Hauptinterkation immer noch darauf, beim Betätigen eines Knopfes neuen Markup vom Server zu laden. Alles, was an Benutzerinteraktion darüber hinaus geht, ist ein Nebengedanke. Und so kommt es vor, dass der serverseitige Teil, zuständig für das Rendern der Hauptinhalte, mit allen bekannten Prinzipien des guten Software Designs realisiert ist, der clientseitige, für die weitergehende Interaktivität zuständige JavaScript-Teil, aber eine bloße Ansammlung loser Scripte darstellt. Bei kleinem Anteil solchen clientseitigen Programmcodes wird diese Praxis aus Kostengründen toleriert, ist jedoch bei jeder mittleren Komplexität nicht mehr hinnehmbar. Diese Arbeit betrachtet die Implementierung eines solchen komplexen Webandwendung Clients.

1.2 Zielsetzung

Diese Arbeit beschäftigt sich mit der Realisierung eines potentiell an Benutzerinteraktionen reichen, nativ ähnlichen Clients einer Webanwendung am Beispiel der Implementierung einer Photoverwaltungssoftware.

Dabei wird auf folgende Schwerpunkte eingegangen:

- Adaptierung an verschiedene Endgeräte
- Auslagerung der Darstellungslogik an den Client
- Architektonische Trennung von Verantwortlichkeiten
- Authentifizierung der Benutzersitzung
- Kommunikation mit dem Server

1.3 Aufbau

2 Analyse

2.1 Einleitung

In dem Kapitel Motivation wurde geschildert, dass eine Web-Photoverwaltungssoftware als Exempel für die Zielsetzungen dieser Arbeit dienen wird. Nun soll analysiert werden, welche Realanforderungen notwendig sind, um den zuvor definierten Zielsetzungen gerecht zu werden.

2.2 User Experience

Folgend werden Anforderungen beschrieben, die sich mit dem Nutzungserlebnis und der visuellen Gestaltung der Applikation befassen.

2.2.1 Schnelles Feedback

Die Hauptanforderung an die Webanwendung besteht darin dem Benutzer das an eine native Applikation angelehnte Nutzungserlebnis zu gewährleisten. Die bei einer klassischen Webanwendung entstehenden Ladezeiten, welche nach jeder Benutzerinteraktion durch das erneute Laden und Darstellen des gesamten Inhaltes auftreten, sollen vermieden werden.

2.2.2 Flaches Design

Das Benutzerinterface der Anwendung soll unter Anwendung der Paradigmen vom flachen Design minimalistisch, jedoch mit klar erkennbaren Aktionsaufrufen, gestaltet werden.

2.2.3 Responsive Design

Die Webanwendung soll sich auf potentiell unterschiedliche Displaygrößen anpassen. Das Benutzerinterface soll hier jedoch nicht komplett für jede mögliche Abstufung der Displaygröße neu gestaltet werden. Es sollen Layout Regeln verwendet werden, die es dem gleichen Interface erlauben seine Elemente bei schrumpfender bzw. wachsender Größe neu zu positionieren.

2.2.4 Photo Zentrierung

Ein besonderes Unterproblem der Adaptierung an verschiedene Gerätedisplays ist die Photobetrachtung. Hier sind sowohl die Auflösung des Photos als auch des Displays für die Software nicht zu Implementierungszeit, sondern erst zur Laufzeit bekannt. Um das Photo in der Gesamtheit auf einem beliebigen Display zu betrachten, soll es zur Laufzeit sowohl in der Detailansicht als auch innerhalb seines Platzhalters in der Gesamtübersicht zentriert werden.

2.3 User Stories

In diesem Unterkapitel werden die einzelnen Features der Beispiel-Applikation deffiniert.

2.3.1 Authentifizierung

Die Photosoftware soll den Benutzern nur anhand einer Benutzerkennung und eines Passworts den Zugang gewähren. Unberechtigter Zugriff soll folglich mit entsprechender Fehlermeldung verweigert werden.

2.3.2 Navigation Menü

Der Benutzer soll im Stande sein zwischen den Hauptfunkionen der Anwendung aus jedem beliebigen Unterbereich zu navigieren.

2.3.3 Photo Galerie

Dem Benutzer soll eine Auflistung seiner gespeicherten Photos dargestellt werden.

2.3.4 Paginierung/Nachladen der Photos

Falls sich sehr viele Photos in der Photo Galerie befinden, sollen diese nicht im selben Augenblick geladen werden, damit die Anwendung nicht überlastet wird. Stattdessen soll zuerst eine bestimmte Anzahl der Photos dargestellt werden und es anschließend dem Benutzer ermöglicht werden, weitere Bilder stapelweise nachzuladen.

2.3.5 Photo Details

Der Benutzer soll in der Lage sein, Photos mit Metadaten wie Name und Beschreibung zu annotieren. Bei der Auswahl eines einzelnen Photos in der Galerie sollen diese annotierten Photoinformationen dargestellt werden.

2.3.6 Photo Großansicht

Dem Benutzer soll ermöglicht werden, ein bestimmtes Photo in der vollständiger Größe zu betrachten.

2.3.7 Photo Slider

Wenn der Benutzer die Detailansicht eines Photos aus der Galerie auswählt, soll es ihm ferner möglich sein aus der Detailansicht zum nächsten oder dem vorherigen Photo zu navigieren.

2.3.8 Photo Freitext Suche

Der Benutzer soll in Beschreibungen und Namen nach seinen Photos durch Eingabe von Freitext suchen können. Das Resultat der Suche soll ebenfalls wie die Photogalerie paginiert werden.

2.3.9 Autovervollständigung der Freitextsuche

Die Freitext-Suche aus 2.3.8 soll aus den in der gesamten Photosammlung befindenden Benennungen und Beschreibungen während der Sucheingabe vervollständigt werden.

2.3.10 Photos nach Erstellungsdatum Filtern

Der Benutzer soll in der Lage sein, nur Photos aus einem von ihm gewählten Zeitraum zu betrachten.

3 Grundlagen

3.1 Client/Server Model

Webanwendungen sind eine erweiterte Form von normalen Webseiten und funktionieren nach den selben Prinzipien des World Wide Webs. Diesen liegt wiederum das Client-/Server-Model zu Grunde.

Der Client ist ein Programm des Benutzers und ist dafür zuständig, den Inhalt der Applikation oder Webseite auf dem Bildschirm in benutzerfreundlicher Art und Weise zu verarbeiten. Ein solcher typischer Client ist der Web Browser.

Der Inhalt selbst befindet sich auf einem entfernten Rechner, genannt der Server. Server verarbeiten eingehende Anfragen der Clients nach Inhalten und liefern eine Kopie dieser Inhalte aus. Der heruntergeladene Inhalt kann schließlich vom Client angezeigt werden.

Die Fachbezeichnung für den remote Inhalt ist Ressource. Ressourcen können aus Bildern, Videos, Webseiten und andere Dateien bestehen. Aber wie am Anfang angedeutet, sind Ressourcen nicht nur auf Dateien und Webseiten beschränkt. Sie können auch in Form vom Software vorkommen, welche es z.B. erlauben, Aktien zu handeln oder Videospiele zu spielen. Ressourcen werden dabei durch einen eindeutigen Bezeichner - die URL - identifiziert.

Ein simples Diagramm, wie Client und Server interagieren können, sieht wie folgt aus:

Historisch ergeben, nutzen Client und Server das Kommunikationsprotokoll HTTP für die Kommunikation untereinander. Diese Übertragung ist zustandslos. Diese Eigenschaft wurde absichtlich konzipiert, um die Protokoll-Implementierung einfach zu halten und um Serverressourcen zu sparen. Der Server muss dabei keine Benutzerinformation zwischen den Anfragen merken. Im Fehlerfall muss ebenfalls nichts aufgeräumt werden. Die beiden Gründe machen HTTP zu einem sehr belastbaren Protokoll, aber auch gleichzeitig zu einem schwierigen Protokoll, um zustandsbehaftete Webanwendungen zu implementieren.

(Parikh u. a. 2015: Background) beschreibt das Problem wie folgt:

"When you go to Facebook, for example, and log in, you expect to see the internal Facebook page. That was one complete request/response cycle. You then click on the picture – another request/response cycle – but you do not expect to be logged out after that action. If HTTP is stateless, how did the application maintain state and remember that you already input your username and password? In fact, if HTTP is stateless, how does Facebook even know this request came from you, and how does it differentiate data from you vs. any other user? There are tricks web developers and frameworks employ to make it seem like the application is stateful..."

D.h, dass es eine Reihe von verschiedenen Techniken gibt, welche auf Anwendungsebene realisiert werden müssen, um Zustandshaftigkeit in einem zustandslosen Protokoll zu gewährleisten.

Vgl. (Parikh u. a. 2015: Background), Federico u. a. (2016)

3.2 HTTP

In 3.1 wurde erwähnt, dass im World Wide Web das Kommunikationsprotokoll HTTP verwendet wird. Ein Protokoll zeichnet sich zunächst durch 3 grundlegende Eigenschaften aus:

- Syntax - Datenformat und Kodierung
- Semantik - Steuerungsinformation und Fehlerbehandlung
- Zeitablauf - Geschwindigkeitsanpassung und Reihenfolge

Dubost (2012) zeigt, dass Kommunikationsprotokolle nicht nur ein künstliches Konstrukt sind, sondern auch aus der realen Welt stammen:

"When two people meet, they engage using a communication protocol: for example, in Japan, a person will make a specific gesture with the body. One such gesture is a bow, which is the syntax used for the interaction. In Japanese customs, the gesture of the bow (among others) is associated with the semantics of greeting someone. Finally, when one person bows to another person, a sequence of events has been established between the two in a specific timing."

Weiterhin beschreibt Dubost (2012), dass in einem Online-Kommunikationsprotokoll die gleichen Elemente vorkommen: die Syntax - die Abfolge von Zeichen, etwa Bezeichnern, die für das Schreiben des Protokolls verwendet werden; die Semantik - die Bedeutung, die mit diesen Bezeichnern assoziiert wird; und schließlich der Zeitablauf - eine vorgegebene Reihenfolge, in der Client und Server diese Bezeichner austauschen.

HTTP ist dabei ein sog. Application Level Protocol, d.h, dass es in der Abstraktionsebene höher angesiedelt ist. Es setzt wiederum auf eine Reihe weiterer Protokolle auf, welche sich etwa um die Übertragung der eigentlichen Datenpakete oder die physikalische Übertragung der elektrischen Signale kümmern. HTTP selbst beschreibt hingegen die Bedeutung und das Format der gesamten übertragenen Nachricht.

Folgend ist eine HTTP GET Anfrage aufgelistet:

```
GET / HTTP/1.1
Host: www.opera.com
User-Agent: Opera
```

Listing 1: HTTP GET Request

Diese Nachricht spezifiziert, dass der Client eine Ressource erhalten möchte. Die Ressource, die der Client erhalten möchte, befindet sich im root-Verzeichnis. Die Übertragung soll mittels HTTP version 1.1 stattfinden. Der Client versucht eine spezifische Webseite zu erreichen, die sich unter der URL *www.opera.com*befindet. Ferner teilt der Client einen sog. HTTP Header, namens *User-Agent* Informationen über das Programm, welches für die Kommunikation verwendet wurde.

Eine Antwort vom Server könnte dabei wie folgt aussehen:

```
HTTP/1.1 200 OK
Date: Wed, 23 Nov 2011 19:41:37 GMT
Server: Apache
Content-Type: text/html; charset=utf-8
Set-Cookie: language=none; path=/; domain=www.opera.com;
  expires=Thu, 25-Aug-2030 19:41:38 GMT
Set-Cookie: language=en; path=/; domain=.opera.com;
  expires=Sat, 20-Nov-2030 19:41:38 GMT
Vary: Accept-Encoding
Transfer-Encoding: chunked

<!DOCTYPE html>
<html lang="en">
...
```

Listing 2: HTTP GET Response

Der Server antwortet, dass er das Protokoll HTTP Version 1.1 versteht. Die Anfrage war erfolgreich und wurde daher mit dem Response Code 200 sowie einer verständlichen Annotation *OK* versehen. Anschließend wird eine Reihe weiterer HTTP Header gesendet, welche beschreiben, wie die Nachricht verstanden werden soll. Und letzendlich wird der Inhalt der Ressource - hier ein HTML Dokument - in den Rumpf (body) der Nachricht eingefügt.

Weitere HTTP-Methoden sind: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT. Jede von denen hat eine unterschiedliche Rolle. Siehe (Fielding und Reschke 2014: Kap. 4).

Vgl. Dubost (2012)

3.3 Serverseitige Webanwendungen

3.3.1 Grundprinzip

Als das World Wide Web geboren wurde, existierte nur ein Webserver und ein Webclient. Dieser Webserver namens httpd war nur in der Lage, statische Ressourcen wie Bilder und Dokumente auszuliefern. Schon bald jedoch machte der Überfluss an Online-Ressourcen Suchmaschinen notwendig. Das bedeutete, dass Benutzer in der Lage sein mussten, Daten, wie den Suchbegriff, an den Server abzuschicken und der Server seinerseits im Stande sein sein musste, diese Daten zu verarbeiten und dynamisch entsprechende Inhalte zu liefern.

Hierfür wurde das Common Gateway Interface (CGI) spezifiziert. Es entwickelte sich zum Standard, um externe Applikationen mit Webservern zu verbinden und um dynamische Information zu generieren. Ein CGI Programm kann beinahe in jeder Programmiersprache implementiert werden. Es muss nur die Fähigkeit besitzen, sein vom STDIN zu lesen und auf STDOUT zu schreiben.

Folgend ist ein exemplarisches CGI "Hello World"Programm dargestellt. Ein Benutzer namens Doug gibt seinen Namen ein, welcher vom Webserver ausgegeben werden soll. Dabei generiert sein Webclient einen HTTP GET Request an folgende URL:

```
http://example.com/cgi-bin/hello.pl?username=Doug
```

Wenn der Webserver diese Anfrage bekommt, weiß er, wie er die URL in zwei Teile trennt: den Pfad zu dem CGI Perl Programm *hello.pl* und den Teil mit der Benutzereingabe (username=Doug, genannt QUERY_STRING). Er leitet also diese Anfrage über STDIN an das *hello.pl* Script weiter. Die Aufagbe des Scriptes ist nun, den QUERY_STRING nach dem Schlüssel

username zu parsen und dessen Wert über SDTOUT auszugeben. Der Webserver wird wiederum diese Ausgabe an den Client weiterleiten. Das Beispiel "Hello user"Programm ist in 3 abgebildet.

```
#!/usr/bin/perl

use CGI qw(:standard);
my $username = param('username') || "unknown";

print "Content-type: text/plain\n\n";
print "Hello $username!\n";
```

Listing 3: "Hello user"CGI script

Ein solches serverseitiges Programm generiert für gewöhnlich dynamische Inhalte, indem es die Information dafür aus einer Datenbank bezieht. Heutige serverseitige Webanwendungen nutzen weiterhin entweder eine Weiterentwicklung der CGI Schnittstelle oder ein ähnliches Prinzip.

Vgl. (Bekman und Cholet 2003: Kap. 1.1)

Im Kapitel 3.1 wurde auf die Diskrepanz hingewiesen, dass HTTP ein zustandsloses Protokoll ist, eine Webanwendung jedoch Zustandshaftigkeit benötigt, um Benutzersitzungen, etwa Besuch und Einkauf in einem Webshop, auseinanderzuhalten. Eine solche Session kann von dem Client und Server Programm künstlich aufrecht erhalten werden.

Dabei generiert das serverseitige Programm einen Session-Identifier beim ersten Besuch des Benutzers und sendet es an den Client. Der Client wiederum sendet diese ID bei jeder weiteren Anfrage an den Server mit. Ein Mechanismus für die Benutzersessions im Web ist das Setzen von HTTP-Cookies, welche dann automatisch bei den nachfolgenden Anfragen angehängt werden. Weitere manuelle Möglichkeiten sind Übertragung mittels custom HTTP Header oder Umwandeln der URLs durch das Anhängen des zusätzlichen *session_id* Parameters.

Siehe (Parikh u. a. 2015: Stateful Web Applications)

3.3.2 Architektur

In objektorientierter Software ist für die Erstellung von graphischen Anwendungen das Model-View-Controller Pattern vorherrschend.

Galilio schreibt: "Mit Model-View-Controller (MVC) wird ein Interaktionsmuster in der Präsentationsschicht von Software beschrieben. MVC ist wohl einer der schillerndsten Begriffe im Bereich der objektorientierten Programmierung. Viele Varianten haben sich herausgebildet, teilweise einfach aufgrund eines falschen Verständnisses des ursprünglichen MVC-Musters, teilweise als Weiterentwicklung oder Anpassung an neue Anwendungsfälle."

Unabhängig von der jeweiligen Abwandlung des MVC Pattern gilt, dass der Controller für Benutzereingaben, das Model für den Zustand und der View für das Darstellen dieses Zustands verantwortlich ist. Vgl. Galilio 8.2.3

Serverseitige Webanwendungen interpretieren MVC wie folgt. Benutzerinteraktionen führen weitgehend zu Anfragen einer komplett neuen Ressource, e.g:

```
GET http://example.com/articles
GET http://example.com/articles/1
GET http://example.com/articles/1/comments
```

Jeder ankommende HTTP Request wird von einem bestimmten Controller verarbeitet. Dieser liest HTTP Header sowie Request Parameter aus und verwendet Model Objekte, um die notwendige Daten für eine Benutzeranfrage zu liefern. Models laden diese Daten üblicherweise aus einer Datenbank. Und schließlich generiert der Controller die gesamte Seite neu, welche sich nur um den neuen Inhalt von der vorherigen unterscheidet, dessen Layout, Menüs, Header etc. aber gleich bleiben. Hierfür wird eine HTML-Template-Engine und ein passendes Template - das View - verwendet. Dieses stellt im Grunde ein HTML-Markup mit Platzhaltern für dynamische Daten dar, welche vom Controller durch die Model-Daten ersetzt werden. Dieses stellt im Grunde HTML Markup mit Platzhaltern für dynamische Daten dar, welche vom Controller durch die Model Daten ersetzt werden. Siehe 3.1

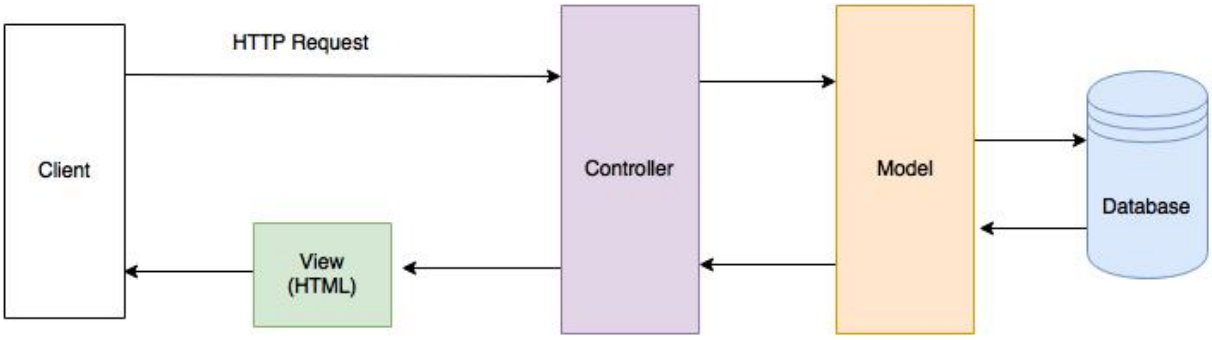

Abbildung 3.1: Serverseitiges MVC

3.4 Clientseitige Webanwendungen

3.4.1 Grundprinzip

In dem Aufsatz, der den Namen *Ajax* geprägt hat, schrieb Garrett (2005): "The classic web application model works like this: Most user actions in the interface trigger an HTTP request back to a web server. The server does some processing — retrieving data, crunching numbers, talking to various legacy systems — and then returns an HTML page to the client. It's a model adapted from the Web's original use as a hypertext medium, but as fans of The Elements of User Experience know, what makes the Web good for hypertext doesn't necessarily make it good for software applications."

Schon in den 90er Jahren implementierten Browserhersteller Skripting Möglichkeiten für Webseiten. Es enstand die Möglichkeit, Programmcode an den Client Rechner zusammen mit dem Markup auszuliefern, um interaktive Animationen auf der Webseite auszulösen. Die Programmiersprache Javascript war geboren. Sie wurde später unter dem Begriff ECMAScript standardisiert.

Im weiteren Verlauf ermöglichte die Implementierung der XMLHttpRequest Schnittstelle, HTTP Anfragen aus Javascript unabhängig von dem Browser Client auszuführen. Dies legte den Grundstein für die von Garret unter dem Begriff *Ajax* zusammengefasste Sammlung von Techniken und somit die Enstehung echter clientseitiger Webanwendungen.

So schreibt Garrett (2005) weiter: „Ajax isn‘t a technology. It‘s really several technologies, each flourishing in its own right, coming together in powerful new ways.“ Und er definiert die Komponenten, welche Ajax ausmachen.

- standards-basierte Darstellung, unter Verwendung von XHTML und CSS
- dynamische Darstellung und Interkation, unter Verwendung des Document Object Model
- Datenaustausch und Manipulation mit Hilfe von XML und XSLT
- Asynchrone Datenabfragen, unter Verwendung des XMLHttpRequest
- Javascript, welches das Ganze zusammenbindet

Anstatt eine Webseite am Anfang der Benutzersitzung zu laden, lädt der Browser eine Ajax Engine - implementiert in Javascript. Diese Engine ist sowohl verantwortlich für das Rendern des Benutzerinterfaces als auch für die Kommunikation zwischen dem Server seitens des Benutzers.

Die von Garret formulierten Ajax Bestandteile gelten noch heute. Allerdings spielt das Datenformat XML keine primäre Rolle. Es ist kein bestimmtes Datenaustausch Format vorgeschrieben, wobei überwiegend das kompakte JSON Format zur Übertragung benutzt wird. Der Schnittstellen Name - XMLHttpRequest blieb aus Kompatibilitätsgründen erhalten. Ab ECMAScript6 existiert eine neue HTTP Api, namens *fetch*.

Vgl. Garrett (2005)

3.4.2 Datenübetragung

Die meist konventionelle Art und Weise, in der Ajax Applikationen mit dem Server kommunizieren, ist eine sog. REST Api. REST steht für representational state transfer.

- *representational* bezieht sich darauf wie eine Representation eine Ressource übertragen wird.
- *state transfer* bezieht sich darauf dass HTTP ein zustandsloses Protokol ist und dass alles was der Server braucht, um eine Anfrage zu verarbeiten, sich in der Anfrage selbst befindet.

Die Grundideen hinter REST wurden auf den Beobachtungen dessen basiert, wie das Web bereits funktionierte. Das Laden von Webseiten, das Absenden von Formularen und das Benutzen von Links, um verwandte Inhalte zu finden sind Faktoren, welche definieren, was REST ist und wie es auf das Web und den Schnittstellen Design zutrifft.

Alle Aktionen innerhalb REST konzentrieren sich also um Ressourcen und somit stellen das Erzeugen (Create), Lesen (Read), Aktualisieren (Update) und Löschen (Delete) die einzigen Aktionen da, welche auf diese Ressourcen angewendet werden. Das Akronym welches diese vier Aktionen beschreibt ist demnach *CRUD*. Beispielweise würde innerhalb des REST Paradigma, um einen Benutzer einzuloggen, nicht etwa eine Remote Procedure *User.login(username, password)* aufgerufen, sondern eine Ressource *UserSession* mit den Attributen *username, password* auf dem Server erstellt.

Eine anschauliche Art REST Schnittstellen zu Implementieren ist alle Aktion auf zwei Kriterien zu brechen

- *Was* - Auf welche Ressource wird eingewirkt ?
- *Wie* - Was passiert mit der Ressource ?

Folgend ist tabelarisch die Akzentuirung des "Wasünd des "Wieäuf den Design einer REST API für den Benutzerlogin dargestellt.

Zielsetzung	Wie		Was	
	Operation	HTTP Methode	Ressource	Pfad
Information über die Session bekommen	Read	GET	Session	/sessions/:id
Eine neue Session erzeugen (Benutzer einloggen)	Create	POST	Sessions Liste	/sessions
Neue Daten zur Session hinzufügen	Update	PUT	Session	/sessions/:id
Session löschen (Benutzer ausloggen)	Delete	DELETE	Session	/sessions/:id

Vgl. (LaunchSchool 2016: Kap. REST and CRUD)

3.4.3 Architektur

Clientseitige Webanwendungen verzichten durchgehend auf komplette Page Loads bei Benutzerinteraktionen. Sie reagieren auf Eingaben, indem sie das Document Object Model (DOM) der Website im Speicher direkt verändern und somit ein Neurendern der betroffenen Teilbereiche auslösen. Anstelle des kompletten Markups werden lediglich die notwendigen neuen Daten in einem serialisierten Format vom einem Webservice angefragt über eine definierte API abgefragt.

Auch hier wird bei einer objektorientierten Umsetzung das MVC Pattern bzw. eine Abwandlung davon verwendet. Dabei werden Benutzereingaben direkt von einem dafür zuständigen Controller verarbeitet. Der Controller leitet die Anfrage an das Model weiter, welches mit einem Webservice interagieren kann. Anschließend benachrichtigt der Controller das View über die Änderung des Zustands der Applikation. Schließlich sorgt das View für die direkte DOM Manipulation eines bestimmten Teilbereich der Seite. Siehe 3.2.

Die meisten clientseitigen Frameworks führen eine Technik namens “Databinding” ein. Dabei wird, einst aus einem Markup Template erstelltes, View an bestimmte Datenfelder in einem Model angebunden. Das View aktualisiert sich mit der Zustandsänderung des Models automatisch. Somit bleibt kaum noch Eventverarbeitungs Aufwand für einen typischen Controller notwendig, weswegen dieser eher als Presenter (siehe MSDN (2016a)) oder sog. ViewModel (siehe MSDN (2016b)) verstanden werden kann.

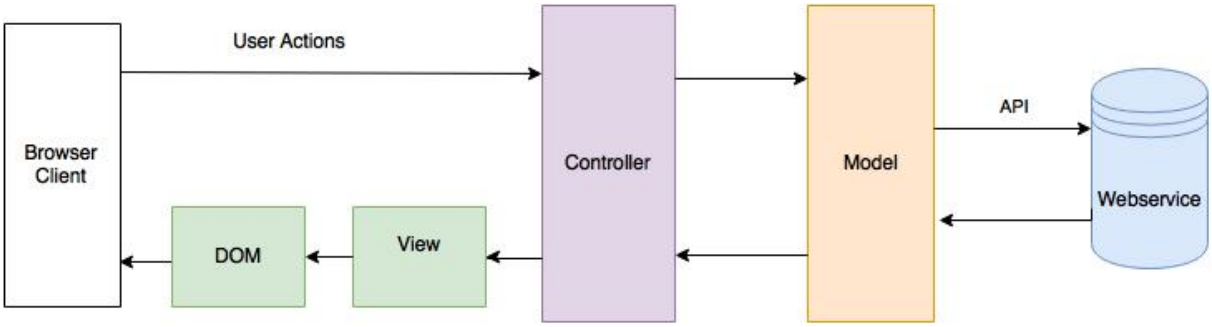

Abbildung 3.2: Clientseitiges MVC

3.5 Responsive Webdesign

(Marcotte 2011: S.8) führt den Beggriff Responsive Webdesign mit folgenden Worten ein:

„But web designers, facing a changing landscape of new devices and contexts, are now forced to overcome the constraints we've imposed on the web's innate exibility. We need to let go. Rather than creating disconnected designs, each tailored to a particular device or browser, we should instead treat them as facets of the same experience. In other words, we can craft sites that are not only more exible, but that can adapt to the media that renders them. In short, we need to practice responsive web design“

Hierfür legte Marcotte 3 Grundelemente fest:

- Flexibles, rasterbasiertes Layout
- Flexible Bilder und Medien
- Media Queries, ein Modul aus der CSS3 Spezifikation

Üblicherweise platzieren Webdesigner die Webseitenelemente auf ein imaginäres Raster. Somit sind bestimmte Komponenten aneinander anhand dieser virtuellen Linien ausgerichtet. In Abb. 3.3 sieht man ein solches Raster. Das Design besteht hier aus zwei Textspalten *main* - 566px breit und *other* - 331 px breit. Diese befinden sich wiederum in einem umschließendem *blog* 900 px Container, welcher wiederum in den *page* Container eingebunden ist. Die Elemente definieren entsprechende Abstände (*margin*), so dass aus Breite der Elemente und der Abstände sich eine Textpositionierung anhand der virtuellen Grid Linien ergibt, wie in der Abb dargestellt.

Das obige Layout kann durch folgenden CSS code erzeugt werden:

Abbildung 3.3: Webdesign Raster

```
#page {
  margin: 36px auto;
  width: 960px;
}
.blog {
  margin: 0 auto 53px;
  width: 900px;
}
.blog .main {
  float: left;
  width: 566px;
}
.blog .other {
  float: right;
  width: 331px;
}
```

Listing 4: Raster CSS

Um nun von einem starren Raster auf ein flexibles Raster zu kommen, welches sich an Bildschirmgrößen anpasst, werden feste Pixel Angaben durch relative Prozentangaben ersetzt. Hierfür wird eine einfach Umrechnungsformel verwendet - $target/context = result$

Wenn also sich das Target Element *main* mit 566px Breite sich bei einem festen Raster inner-

halb des Kontextelementes *blog* mit 900px Breite befindet, ergibt sich dafür laut 566/900 eine 62.8888889% Breite:

```
.blog .main {
  float: left;
  width: 62.8888889%;
}
```

Listing 5: Flexibles Raster CSS

Im Prinzip wird die obige Umrechnung auf alle festen Breiten-, Abstand- und Fontgrößen Angaben angewendet. Daraus ergibt sich ein anpassbares Raster(siehe Abb. 3.4).

Abbildung 3.4: Flexibles Webdesign Raster

Bezüglich flexibler Bilder und Medien verhält es sich ähnlich, wie mit dem Grundansatz des flexiblen Rasters. Werden Bilder in Markup Container eingebettet, so sollte dieser Container sine Abstände sowie Breite widerum relativ zu seinem Contexcontainer gesetzt haben. Siehe Listing.

```
<div class="figure">
  <p>
    <img src="robot.jpg" alt="" />
    <b class="figcaption">Lo, the robot walks</b>
  </p>
</div>
```

Listing 6: Container mit Bild Markup

```
.figure {
    float: right;
    margin-bottom: 0.5em;
    margin-left: 2.53164557;
    width: 48.7341772%;
}
```

Listing 7: Container mit Bild CSS

Diese Einstellung allein wird aber nicht ausreichen. Sollte das Bild in Wirklichkeit größer sein, als sich Platz dafür resultierend aus der dynamischen Breitenangabe und der jeweiligen Gerätauflösung ergibt, wird es standardmäßig mit Scrollbalken innerhalb seines Containers in original Größe gerendert.

Die Lösung dieser Problematik hängt von dem gewünschten Effekt ab. In vielen Fällen wird einfach das Gleiche Skalierverhlaten für das Bild gewünscht, wie für den Rest des Inhalts. Hierfür sorgt eine *max-width: 100%* Einstellung auf das Bildelement selbst.

```
.figure {
    float: right;
    margin-bottom: 0.5em;
    margin-left: 2.53164557;
    width: 48.7341772%;
}

.figure img {
  max-width: 100%;
}
```

Listing 8: Bildskalierung

Es kann aber auch ausreichen sein, dass einfach ein bestimmter Bildausschnitt gerendert wird, so im Falle von Hintergrundbildern. Dieses wird mit Hilfe einer *overflow: hidden%* css Regel erreicht auf dem Context Container des Bildes erreicht.

```
.figure {
  overflow: hidden;
}
.feature img {
  display: block;
  max-width: auto;
}
```

Listing 9: Bildausschnitt

Denkbar wäre auch die Absicht ein komplett anderes, eventuell ähnliches aber unterschiedlich arrangiertes Bild für kleinere Auflösungen zu laden, im Falle von Grafiken beispielsweise. Dieses erfordert jedoch eine zusätzliche Implementierung auf der Serverseite.

In dieser Arbeit wird eine weitere Technik für den Fotoslider geschildert, wobei das Foto nicht nur mit dem Inhlat skallieren, sondern ebenfalls zentriert sein muss.

A Material

A.1 Fragebögen, Messprotokolle etc.

In den Anhängen landen ggf. Listings, Fragebögen, Datenblätter, Messprotokolle, Skizzen zu Versuchsaufbauten und ähnliches Material zur Arbeit. Im LaTeX-Dokument leitet der Befehl appendix die Anhänge ein.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Bekman und Cholet 2003] BEKMAN, Stas ; CHOLET, Eric: *Practical Mod_PERL*. Sebastopol, CA, USA : O'Reilly & Associates, Inc., 2003. – ISBN 0596002270

[Crockford 2008] CROCKFORD, Douglas: *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008. – ISBN 0596517742

[Dubost 2012] DUBOST, Karl: *HTTP - an Application-Level Protocol*. 2012. – URL https://dev.opera.com/articles/http-basic-introduction/. – letzter Zugriff: 06. 11. 2016

[Federico u. a. 2016] FEDERICO ; CULLOCA u. a.: *How the Web woks*. 2016. – URL https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works. – letzter Zugriff: 20. 10. 2016

[Fielding und Reschke 2014] FIELDING, Roy ; RESCHKE, Julian: *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. 5177 Brandin Court Fremont, CA 94538: IETF (Veranst.), 2014. – URL https://tools.ietf.org/html/draft-ietf-httpbis-p2-semantics-26#section-4. – letzter Zugriff: 06. 11. 2016

[Garrett 2005] GARRETT, Jesse J.: *Ajax: A New Approach to Web Applications*. 2005. – URL http://adaptivepath.org/ideas/ajax-new-approach-web-applications/. – letzter Zugriff: 27. 11. 2016

[LaunchSchool 2016] LAUNCHSCHOOL: *Working with Web APIs*. Launch School, 2016. – URL https://launchschool.com/books/working_with_apis. – letzter Zugriff: 27. 11. 2016

[Marcotte 2011] MARCOTTE, Ethan: *Responsive Web Design*. A Book Apart, 2011. – ISBN 9780984442577

[MSDN 2016a] MSDN: *The Model-View-Presenter (MVP) Pattern*. 2016. – URL https://msdn.microsoft.com/en-us/library/ff649571.aspx. – letzter Zugriff: 18. 11. 2016

[MSDN 2016b] MSDN: *The MVVM Pattern*. 2016. – URL https://msdn.microsoft.com/en-us/library/ff649571.aspx. – letzter Zugriff: 18. 11. 2016

[Parikh u. a. 2015] PARIKH, Aarti ; AGRAM, Albert u. a.: *Introduction to HTTP*. Launch School, 2015. – URL https://launchschool.com/books/http. – letzter Zugriff: 20. 10. 2016

[Schippers 2016] SCHIPPERS, Ben: App Fatigue. In: *TechCrunch* (2016). – URL https://techcrunch.com/2016/02/03/app-fatigue/. – letzter Zugriff: 17. 10. 2016

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Maria Mustermann