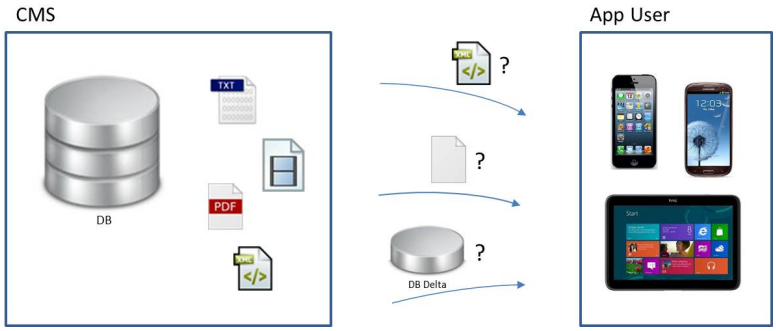
Contentübergabe an eine Mobile App über ein CMS

Alexander Gustafson
Ramon Schilling
13. Februar 2013



Seminararbeit
ausgeführt an der

Zürcher Hochschule für Angewandte Wissenschaften (ZHAW)
Studiengang Informatik

im Seminar "Handheld"

unter der Leitung von
Christian Vils

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

Für content-basierte Mobile Apps müssen immer wieder neue Inhalte publiziert werden. Dies soll jeweils vom Betreiber der App auf einfache Art möglich sein. Dies ist möglich, wenn die Inhalte in einem CMS verwaltet werden. Die App soll automatisch aktuell gehalten werden, ohne dass die ganze App aktualisiert werden muss, und auch ohne dass der Konsument aktiv wird. Wichtig ist zudem, dass die Daten auch offline zur Verfügung stehen und auch grössere Datenmengen unproblematisch übermittelt werden können. Trotzdem muss aber ein Mechanismus gefunden werden, dass nur möglichst kleine Datenmengen übermittelt werden müssen um z.B. auf geringe Bandbreite reagieren zu können, oder Roaming-Gebühren klein zu halten. Zum Beispiel müssen Apps von Museen, Ausstellungen, Konzertbetreibern oder Zeitschriften diese Anforderungen erfüllen

1.2 Aufgabenstellung

- Recherchieren, welche Produkte zur Lösung des Problems bereits bestehen und allfällige Vor- und Nachteile dieser aufzeigen.
- Aufzeigen welche Lösungsvarianten möglich sind.
- Einarbeiten in die Handheld Programmierung.
- Auswahl von Techniken für die eigene Implementation.

1.3 Zielsetzung

Das Ziel der Seminararbeit ist es aufzuzeigen, welche Systeme, Prozesse und Design-Patterns wichtig sind und in Frage kommen um den oben genannten Anforderungen gerecht zu werden. Insbesondere die Schwierigkeit, möglichst effizient aktuelle Daten dem App-Benutzer zur Verfügung zu stellen soll dabei beachtet werden.

Falls möglich können bereits gewisse Teile praktisch umgesetzt werden: z.B. kann ein CMS bereitgestellt werden, über welches der Betreiber der App die Inhalte bereitstellen und zur Aktualisierung für die App freigeben kann. Weiter kann eine App entwickelt werden, welche ihre Inhalte über das CMS bezieht und die definierten Prozesse umsetzt.

1.4 Motivation

Eine eigene Mobile App zu haben, ist heute für eine Firma oder eine Organisation fast schon so selbstverständlich wie eine Website zu besitzen. Durch diese rasante Verbreitung von Mobile Apps wird es immer wichtiger, dass diese auch einfach aktuell gehalten werden können. Uns hat die Idee dieser Semesterarbeit deshalb fasziniert.

Unabhängig von dieser Arbeit haben wir auch Anfragen aus dem Freundes- und Bekanntenkreis erhalten, welche sich für eine App interessierten, die für Promotionszwecke genutzt werden kann. Sie wollten wissen mit wie viel Aufwand eine solche App verbunden wäre. Da wir selbst keine Erfahrung in diesem Bereich hatten konnten wir auch keine Informationen geben. Dies hat uns aber zusätzlich motiviert, nach geeigneten Lösungen zu suchen um die Erstellung von Apps, und vor allem die Erneuerung der Inhalte zu verinfachen.

2 Projektmanagement

2.1 Projektteam

Das Projektteam besteht aus Alex Gustafson und Ramon Schilling. Die Recherchearbeit haben wir zwischen beiden Projektmitgliedern aufgeteilt und die Resultat jeweils miteinander besprochen. Die Implementation wurde hauptsächlich von Alex Gustafson gemacht, während die Dokumentation von Ramon Schilling erstellt wurde.

2.2 Termine

12. Sept. 2012 - Kick Off Meeting
26. Sept. 2012 - Abgabe der Aufgabe
28. Nov. 2012 - Abgabe Teaser
05. Dez. 2012 - Arbeitstreffen
13. Feb. 2013 - Abgabe Schriftliche Arbeit
20. Feb. 2013 - Präsentation

2.3 Projektplanung

2.4 Aufwand

Der Aufwand für die Seminararbeit sollte pro Person 75 h betragen. Da wir die Arbeit zu zweit machen, haben wir unsere Planung deshalb auf 150 h ausgelegt.

Beschreibung	**Soll**	**Ist**
Recherche von möglichen Synchronisationsabläufen	30 h	30 h
Recherche bereits erhältlicher Lösungen	10 h	9 h
Testen verschiedener Synchronisationsabläufe	8 h	12 h
Einrichten der Entwicklungsumgebung	4 h	5 h
Programmieren / Testen CMS	35 h	40 h
Programmieren iPhone App	42 h	38 h
Beispiel App erstellen	12 h	10 h
Dokumentation	13 h	14 h
Total	**154 h**	**158 h**

2.5 Hilfsmittel

2.5.1 Versionskontrolle

Um eine Versionskontrolle zu haben, und damit wir beide gleichzeitig am Projekt arbeiten konnten, haben wir bei Github [**Github**] ein Repository erstellt und sämtliche für das Projekt nötigen Dateien dort eingecheckt.

2.5.2 Dokumentation

Dokumentation haben wir mit LaTeX erstellt.

2.5.3 Programmierung

Als Entwicklungsumgebung für die Programmierung mit PHP [**php**] hat uns PHP Storm [**PhpStorm**] gedient.

Für die iPhone App Programmierung haben wir Xcode 4 [**Xcode**] genutzt.

Als Datenbank haben wir auf SQLite [**sqlite**] zurückgegriffen.

Das Optimus Dashboard [**optimus**] nutzen wir bei der Entwicklung des CMS.

Für den File Upload nutzten wir elfinder [**elfinder**].

3 Recherche

3.1 Content Deployment

Es gibt diverse Ansätze um einer App den aktuellen Content zu übergeben. Hier sollen verschiedene Möglichkeiten aufgezeigt und deren Vor- bzw. Nachteile kurz erläutert werden.

3.1.1 Ganze App aktualisieren

Eine Möglichkeit ist selbstverständlich bei jeder Änderung des Inhalts die gesamte App zu aktualisieren. Bei Apps mit statischem Inhalt welcher sehr selten wechselt kann das durchaus ein gangbarer Weg sein. Hat die App aber einen grösseren Umfang und werden regelmässig Inhalte geändert ist diese Methode nicht geeignet da viele Anwender keine automatische Aktualisierung ihrer Apps zulassen und jedem Update zustimmen müssen.

3.1.2 Web Application

Bei einer Web Application wird der Inhalt jedes Mal wenn die App genutzt wird neu geladen. Grundsätzlich handelt es sich um eine Website welche für die verschiedenen Geräte angepasst wird. Der Vorteil dieser Methode besteht in der relativ einfachen Umsetzung. Damit die App genutzt werden kann, muss eine Internetverbindung bestehen. Dies ist natürlich ein Nachteil, da dies nicht immer gewährleistet werden kann und insbesondere im Ausland hohe Roaminggebühren anfallen können.

3.1.3 Nur Daten aktualisiern

Um eine App mit den möglichst aktuellen Daten auch offline nutzen zu können, muss sich die App jeweils die neuen Daten herunterladen wenn eine Netzwerkverbindung zur Verfügung steht. Der Vorteil ist natürlich, dass bei Inhaltsaktualisierungen nicht die ganze App aktualisiert werden muss und die App offline genutzt werden kann. Dafür muss in der App zusätzliche Logik eingebaut werden. Es muss jeweils überprüft werden ob aktuellere Daten verfügbar sind und diese müssen dann bei Bedarf und Möglichkeit heruntergeladen und überprüft werden. Besonders bei grossen Datenbeständen lohnt es sich, wenn nur Änderungen und nicht der gesamte Datenbestand übermittelt wird.

3.2 Marktsituation

Wir haben uns mit der Frage beschäftigt, welche Produkte es bereits gibt um einer App die Daten über ein CMS zur Verfügung zu stellen.

Uns sind einige interessante Produkte aufgefallen, welche die Plattformunabhängige Entwicklung von Apps erleichtern sollen und die Verteilung auf den verschiedenen Systemen vereinfachen. Dies geht zum Teil so weit, dass die aktualisierten Apps automatisch in den verschiedenen App Stores eingespielt werden und dass die Aktualisierung der bereits installieretn Apps automatisch überwacht wird.

Weiter gibt es eine Menge an Produkten welche die Entwicklung von Web Applications erleichtern sollen.

Wir sind aber auf kein käufliches oder freies Produkt gestossen, welches sich auf die effizeinte Verteilung von Inhalten auf eine App spezialisiert. Es gibt zwar diverse Firmen welche Apps entwickeln und dem App-Betreiber ermöglichen Content über ein CMS zu verwalten. Da diese Firmen aber inhouse Lösungen benutzen, hatten wir keinen Einblick in die Art und Weise wie sie die Daten übermitteln.

3.3 Datenspeicherung

Die Form der Datenspeicherung hat Einfluss auf die Möglichkeiten wie die Daten verwaltet und übermittelt werden können.

XML Files wären eine Möglichkeit, Daten textbasiert zu speichern. Da wir die Möglichkeit in Betracht ziehen die Daten über github zu verwalten, könnte diese Form der Datenspeicherung sehr nützlich sein. Ein Vorteil von dieser Variante wäre ausserdem, dass einzelne Dateien übermittelt werden können.

Die Speicherung der Daten in einer relationalen Datenbank hätte den Vorteil, dass durch die binäre Speicherung weniger Speicherplatz benötigt wird und der Zugriff grundsätzlich etwas schneller ist. Da dies aber vor allem bei grossen Datenmengen zum tragen kommt, ist dies kein ausschlaggebendes Kriterium. Mit SQLite existiert ausserdem eine Datenbank welche von allen gängigen Smartphone Systemen gut unterstützt wird.

3.4 Verfahren für die Datenübermittlung

Unser Ziel ist es eine Umgebung zu entwicklen, in welcher eine Appe mit aktuellen Inhalten versorgt wird, die Inhalte aber auch offline Nutzen kann.

3.4.1 Änderungen Übermittlung

Um auch bei häufigen Änderungen das zu übertragende Datenvolumen möglichst klein zu halten ist es wichtig, dass nicht bei jeder Änderung sämtliche Daten übertragen werden. Es sollen jeweils nur die geänderten Daten übermittelt werden.

3.4.2 git

Versionierungssysteme wie z.B. git [**git**] setzen diese Technik um. Wir haben deshalb die Möglichkeit geprüft, die Datenübermittlung vom CMS zur Mobile App über github zu lösen. Ein Vorteil bei diesem Vorgehen wäre, dass die Logik zur Ermittlung der Änderungen nicht neu entwickelt werden müsste. Da git zwar Änderungen in Binärdateien erkennt, aber nur bei Text-Dateien die Differenz markieren kann, kommt der Vorteil von git nur bei Text-Dateien zum Tragen. Dieses Verfahren würde deshalb die Auswahl des Datenformats beeinflussen. Ausserdem gibt es zwar für alle gängigen Smartphone Systeme git-Clients, diese haben aber nicht alle Funktionen von git implmentiert.

Der entscheidende Punkt welcher gegen den Einsatz von github sprach, liegt aber bei der Art und Weise wie github funktioniert. Änderungen welche auf github gepusht (heraufgeladen) werden, werden dort nur akzeptiert, wenn der Client zuvor die aktuellste Version gepullt (heruntergeladen) hatte und einen Merge (eine Art Synchronisation der Datenbestände) gemacht hatte. Das gibt dem Client die Möglichkeit nur die Änderungen zu pushen, und nicht alle Daten. Werden im CMS Daten geändert und dann auf github gepusht, werden also nur die Änderungen übermittelt.

Fordert aber ein Client die aktuellen Daten an, macht er einen Pull und es wir ihm alles übermittelt. Die Mobile App würde sich also immer alle Daten herunterladen.

Es gibt die Möglichkeit, Patches zu erstellen, welche nur gewisse Änderungen enthalten. Diese Patches könnten dann zu den mobilen Endgeräten übermittelt werden. Da nicht auf allen Endgeräten der selbe Stand der Daten vorhanden ist, müssten dies Patches verwaltet werden. Ausserdem unterstützen nicht alle freien git-Clients für Smartphone die Verwendung von Patches.

Um mit github arbeiten zu können, müsste also sowohl Serverseitig die Patchverwaltung, wie auch in der Mobile App gewisse git-Funktionalität und die Patchverarbeitung aufwendig implementiert werden. Da dies den Rahmen dieser Arbeit sprengen würde und nicht dem Fokus dieser Arbeit entspricht haben wir diese Idee verworfen.

3.4.3 Database Change Log Files

Sämtliche Änderungen in einer Datenbank kann man in einem Log File speichern. Dazu werden alle ausgeführten SQL Statements welche Änderungen in der Datenbank zur Folge haben in dieses Log File geschrieben. Erstellt man für jede Version der Datenbank ein eigenes solches Log File, kann man jede ältere Version der Datenbank in eine neuere Version überführen, indem man die entsprechenden SQL Statements sequentiell abarbeitet.

Im Einsatz mit einer Mobile App werden viele zusätzliche Resourcen wie Bilder, Videos, usw. genutzt. Es müsste also ebenfalls festgehalten werden, welche Resourcen

hinzugefügt wurden. Diese Resourcen können in einem seperaten Log File aufgelistet werden, welches ebenfall für jede Version der Datenbank erstellt wird.

Anstatt die gesamte Datenbank und sämtliche Resourcen zu übermitteln, müsste der Mobile App anhand dieser beiden Log Files nur die neuen Resourcen und die nötigen SQL Statements mit den Datenbankänderungen übermittelt werden.

Um auf dem mobilen Endgerät Platz zu sparen, empfiehlt es sich nicht mehr gebrauchte Resourcen zu löschen. Um dies zu bewerkstelligen kann die mobile App ihren Datenbestand nach nicht mehr gebrauchten Resourcen durchsuchen, oder es wird mit dem Update eine Liste mitgegeben welche die zu löschenden Resourcen enthält.

4 Implementation

4.1 Aufbau

Die Implementation besteht aus einem CMS in welchem der Content von einer oder mehreren Apps verwaltet werden kann. Die Inhalte werden in einer Datenbank gespeichert. Eine iPhone App kann sich dann die Inhalte aus der Datenbank herunterladen und darstellen, bzw. verarbeiten.

4.2 Datenstruktur

Um eine grösstmögliche Flexibilität zu gewährleisten wird die Datenstruktur nicht vorgeschrieben, sondern kann den Bedürfnissen angepasst erstellt werden.

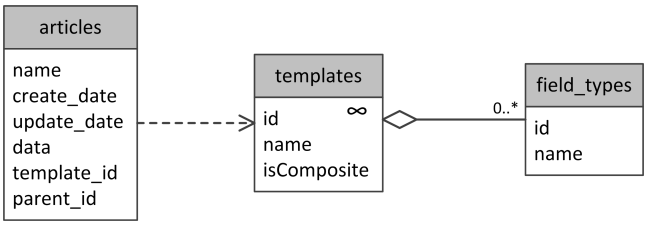


Abbildung 4.1: **Entity Model**

In Abbildung 4.1 ist das Entitätenmodell zu sehen auf welchem die Datenbank beruht.

4.2.1 Field_types

field_types können beliebig defniert werden. Es kann sich dabei um ein Attribut handeln, welches einen Status beschreiben soll, oder um eine Beschreibung eines Elements. Typische *field_types* sind z.B: text, url, color. Wichtig ist, dass ein *field_type* nicht einem Datentyp entsprechen muss.

4.2.2 Templates

Die *Templates* sind das Herzstück der Datenstruktur. Es handelt sich dabei um eine Art Schablone, welche die Struktur für ein beliebiges Element vorgibt. Ein *Template* kann entweder aus beliebig vielen *Field_types* bestehen (einfaches Template) oder aus

beliebig vielen anderen *Templates* und *Field_types* (Composite Template).

Als Beispiel dafür kann man sich ein Buch vorstellen, welches aus einem Titel und mehreren Kapiteln besteht. Titel und Kapitel wären *Templates* welche aus dem *Template* text bestehen und das *Template* Buch würde aus den *Templates* Titel und Kapitel bestehen.

4.2.3 Articles

Jeder *Article* beruht auf einem *Template*. Ein *Article* ist die Umsetzung eines *Templates* und entspricht einer dargestellten Seite in der App. Das *Template* gibt die Struktur vor und die Daten werden entsprechend dieser Vorgabe als json-String abgelegt.

Die Mobile App interpretiert dann diesen json-String anhand der durch das Template vorgegebenen Struktur und stellt die Seite dar.

4.3 Datenbank

Als Datenbank haben wir uns für SQLite entschieden, da diese von allen verbreiteten Smartphone Systemen sehr gut unterstützt wird.

Um die oben beschriebene Struktur in einer relationalen Datenbank abzubilden wird die Datenbank wie folgt aufgebaut:

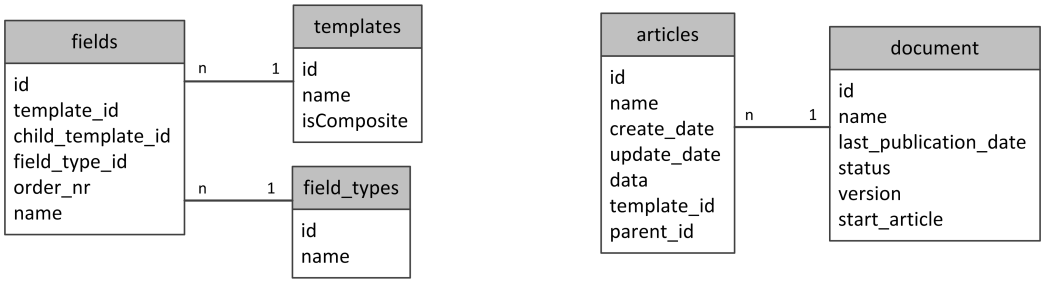


Abbildung 4.2: **Database Model**

Die Zwischentabelle *fields* wird benötigt um die Verschachtelung der *Templates* und *Field_Types* zu ermöglichen. Jeder *Article* ist einem *document* zugeordnet. Im *document* wird der Start Artikel, Status und die Version der Daten festgehalten. Werden mehrere Apps in einer Datenbank verwaltet werden mehrere *documents* erstellt.

4.4 CMS

Das CMS haben wir mit PHP entwickelt. Um die Benutzeroberfläche schneller und benutzerfreundlicher gestalten zu können haben wir das Framework Optimus Dashboard

genutzt. Das CMS ist in die Bereiche Dashboard, Documents, Articles, Templates und Assets unterteilt.

4.4.1 Dashboard

Dieser Bereich wird noch nicht gebraucht. Das Ziel ist es hier generelle Informationen anzuzeigen

4.4.2 Documents

Hier werden einzelne Documents (Apps) erstellt und verwaltet. Es kann der Startartikel für die App angegeben werden sowie die Version der Daten und der Status der App angepasst werden.

4.4.3 Articles

Dies ist der wichtigste Teil für die Bereitstellung von Inhalten. Hier werden die Artikel (Seiten der App) erstellt und bearbeitet.

Um die Daten möglichst einfach einzugeben, muss das CMS pro *field_type* eine entsprechende Eingabemaske anbieten (für *text* ein Textfeld, für *color* ein Color Picker, etc.).

4.4.4 Templates

Die für die Einrichtung wichtigen Templates können hier zusammengestellt werden. Dazu erlaubt das CMS die benötigten Felder zu verknüpfen und zu gruppieren.

4.4.5 Assets

Sämtliche Resourcen wie Bilder, Videos, etc. können hier verwaltet werden.

4.5 App

Das Ziel ist, dass die App generisch ist, und grundsätzlich keine Styles oder Bilder enthält. Jedes im CMS vorhandene Template sollte verarbeitet werden können. Dazu muss für iOS ein View Controller und für Android eine Activity implementiert sein. Bei anderen Systemen muss ebenfalls die Logik implementiert werden dass die möglichen Templates verarbeitet und dargestellt werden. Je mehr Templates definiert sind und von der App unterstützt werden, desto mehr Möglichkeiten bieten sich an.

4.5.1 Auf neue Inhalte überprüfen

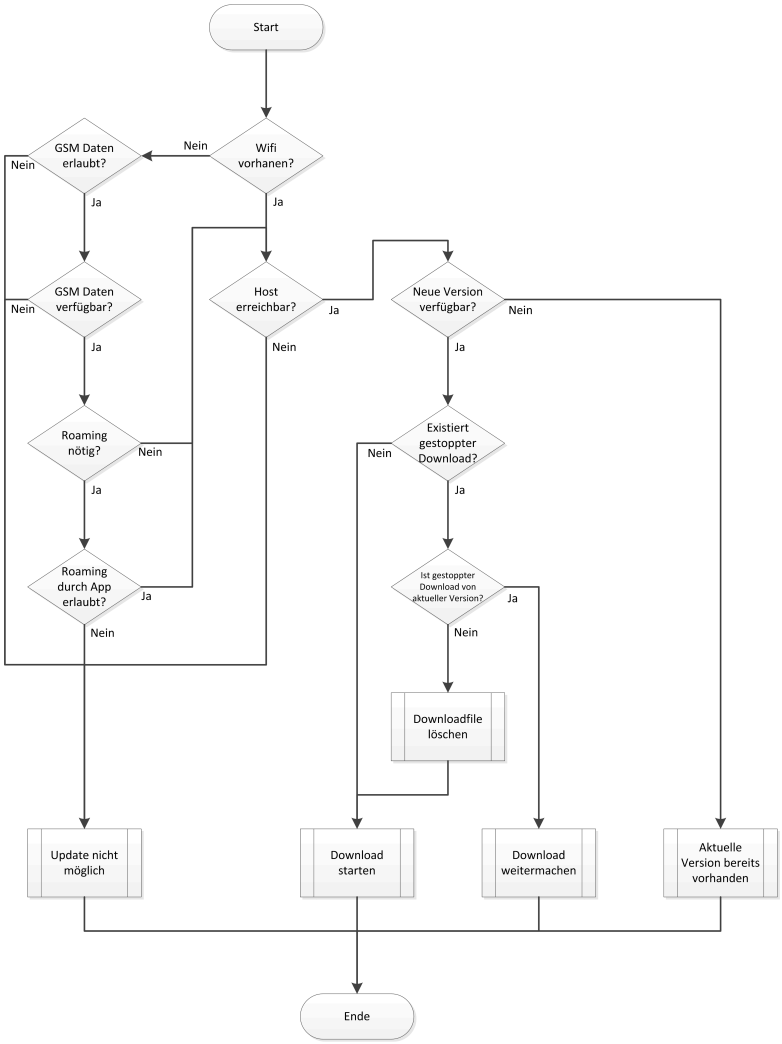


Abbildung 4.3: **Ablaufdiagramm - Daten Update**

Abbildung 4.3 zeigt das Vorgehen der App zur Prüfung ob neue Daten vorhanden sind. Wichtig ist die Überprüfung ob Wifi vorhanden ist, bzw ob GSM Daten erlaubt sind. GSM steht hier stellvertretend für alle Mobiltelefonnetze. Da eine Datenverbindung über ein Mobiltelefonnetz mit Kosten verbunden sein kann (speziell Roaming-Gebühren), fragt die App in diesem Fall nach, ob eine Verbindung gemacht werden darf.

4.5.2 Interpretation von Artikeln

Um ein Artikel darzustellen, analysiert die App den json-String, welcher im Datenfeld des Artikels hinterlegt ist. Anhand des Templates weiss die App wie der json-String zerlegt werden muss und die entsprechenden Elemente darzustellen sind.

Abbildung 4.4: **Interpretation des json-Sting**

In Abbildung 4.4 sehen wir die schematische Darstellung des Templates im CMS, den json-String welcher das Template umsetzt und die Darstellung in der App.

5 Beispielapp - Computer Museum Guide

5.1 App Beschreibung

Als Beispielapp haben wir einen Guide für ein Computer Museum entworfen. Der Guide zeigt Informationen zu den ausgestellten Computern und beinhaltet eine Karte des Museums über welche ebenfalls auf die Informationen über die ausgestellten Computer zugegriffen werden kann. Zusätzlich können Informationen zum Museum abgerufen werden.

5.2 Einrichtung

Im CMS haben wir ein neues Dokument erstellt und als Start-Artikel die Einstiegsseite hinterlegt. Die nötigenField Types und Templates haben wir in der Datenbank hinterlegt.

5.2.1 Field Types

Wir haben folgende *Field_types* definiert:

Name	**Beschreibung**
text	normaler Text
html_text	HTML Text welcher entsprechend verarbeitet wird
resource_path	Pfadangabe für verlinkte Dateien (Bilder, Videos, etc.)
link_to_article	Link zu einem Anderen Artikel
url	URL
number	Zahl
color	Farbe

Hinweis: Mit *html_text* haben wir uns eine Art Joker-Feld geschaffen. Wir können damit beliebigen HTML Code durch die App verarbeiten lassen und können so auch Javascript nutzen. Dies ermöglicht uns, in der App gewisse Logik auszuführen oder Sachen darzustellen welche nicht durch ein implementiertes Template abgedeckt werden.

5.2.2 Templates

Folgende *einfachen Templates* und emphComposite Templates haben wir definiert.

Einfache Templates

- **Main Menu Item** (Hauptmenüpunkt):
  Felder:

Name	**Beschreibung**	**Field_Type**
Label	Beschriftung	text
Article Link	Link zum Artikel	link_to_article
Background Color	Hintergrundfarbe	color
Button Image Normal	Link zum "Button Bild"	resource_path
Button Image Pressed	Link zum "Button gedrückt Bild"	resource_path

- **Tabbar Item** (Tabbar Eintrag):
  Felder:

Name	**Beschreibung**	**Field_Type**
Icon	Link zum Bild	resource_path
Label	Beschriftung	text
Link to Article	Verlinkter Artikel	link_to_article

- **Image** (Bild):
  Felder:

Name	**Beschreibung**	**Field_Type**
Title	Überschrift	text
Image	Link zum Bild	resource_path

- **Location** (Ort):
  Felder:

Name	**Beschreibung**	**Field_Type**
Label	Beschriftung	text
Article Link	Link zum Artikel	link_to_article
X Coordinate	X Koordinaten im Bild	number
Y Coordinate	Y Koordinaten im Bild	number
Pin Icon	Link zum Icon	resource_path

- **Computer Info Page** (Informationen über einen Computer):
  Felder:

Name	**Beschreibung**	**Field_Type**
Title	Überschrift	text
Main Image	Bild des Computers	resource_path
Image Source	Quellenangabe	text
Computer Info Text	Beschreibung	text

- **HTML Page** (HTML Seite):
  Felder:

Name	**Beschreibung**	**Field_Type**
HTML Text	HTML Code der angezeigt wird	html_text

Composite Templates

- **Main Menu** (Hauptmenü):
  Felder:

Name	**Beschreibung**	**Field_Type / Template**
Title	Überschrift	text
Menu Item 1	1. Menüpunkt (ZZ Spectrum)	Main Menu Item
Menu Item 1	2. Menüpunkt (Commodore 64)	Main Menu Item
Menu Item 1	3. Menüpunkt (Lilith Computer)	Main Menu Item
Menu Item 1	4. Menüpunkt (Commodore Amiga 500)	Main Menu Item

- **Tabbar** (Tabbar, welche ausser auf dem Startbildschirm immer angezeigt wird):
  Felder:

Name	**Beschreibung**	**Field_Type / Template**
Tabbar Item 1	1. Tabbar Item (Computers)	Main Menu Item
Tabbar Item 1	2. Menüpunkt (Karte)	Main Menu Item
Tabbar Item 1	3. Menüpunkt (Info)	Main Menu Item

- **Computer Museum Map View** (Museumskarte):
  Felder:

Name	**Beschreibung**	**Field_Type / Template**
Title	Überschrift	text
Map Image	Karte	resource_path
ZX Spectrum Location	Koord. zum ZX Spectrum	Location
C64 Location	Koord. zum C64	Location
Lilith Computer Location	Koord. zum Lilith	Location

5.3 Look and Feel

In der Beispielapp sind einige wenige Views implementiert welche die Nutzung der *Templates* veranschaulichen.

Start View
Die Start View wird beim öffnen der App angezeigt. Das einzige genutzte Template ist ein *Image* mit dem Logo der App und dem Link auf die Main Menu View.

Main Menu View
Das Composite Template *Main Menu* wird genutzt um auf die einzelnen Info Views zu referenzieren und das Template *Tabbar* wird für das schnelle Wechseln zwischen den Kategorien genutzt.

Map View
Die Karte des Museums wird als Bild mit dem Composite Template *Computer Museum Map View* umgesetzt und die Links zu den Computerinformationen sind *Locations*.

Info View Die Info View zeigt die genauen Informationen eines Computers an. Wir haben dafür das Template *Computer Info Page* genutzt.

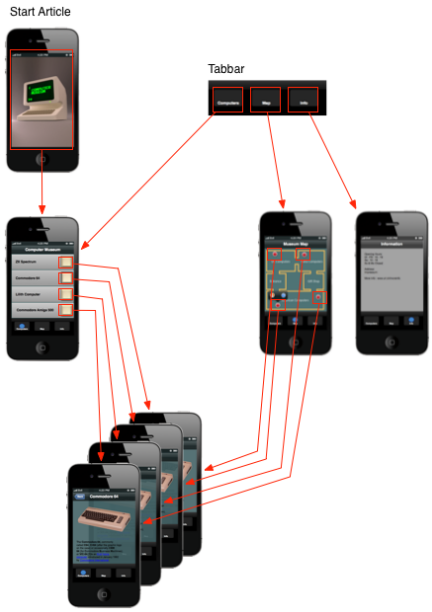


Abbildung 5.1: **Look and Feel**

In Abbildung 5.1 sieht man wie einzelenen Views aussehen und wie die Artikel untereinander referenziert sind um die Navigation zu ermöglichen.

6 Fazit

In dieser Arbeit haben wir uns mit verschiedenen Fragestellungen befasst und nach neuen Lösungsansätzen gesucht um auf einfache und effiziente Weise Inhalte für eine Mobiel App bereitzustellen und zu übermitteln.

Wir hatten grosse Schwierigkeiten herauszufinden, welche Technologien und Verfahren bereits eingesetzt werden, da wir kein Produkt gefunden haben, welches sich primär mit der effizienten Übermittlung von Inhalten auf eine Mobile App befasst. Zusätzlich sind fast keine technischen Informationen verfügbar von Dienstleistern welche sich auf Mobile Apps spezialisiert haben. Das hat die Recherchearbeit erschwert und stellenweise auch etwas langatmig gemacht.

Trotzdem war die Aufgabe sehr spannend. Neben der Informationsbeschaffung und der Auseinandersetzung mit verschiedenen Technologien, haben wir ein eigenes CMS aufbauen können und eine App entwickelt, welche sich über dieses CMS aktualisieren lässt. Das CMS und die App sind so aufgebaut, dass sich leicht neue Inhaltstypen und weitere Funktionalität implementieren lässt.

Selbstverständlich gibt es für die Zukunft noch viel Erweiterungspotential um welches wir uns im Rahmen dieser Arbeit nicht kümmern konnten. Das System ist zwar so aufgebaut, dass man das CMS für mehrere Mandanten einsetzen könnte, diese Funktionalität ist aber noch nicht implementiert. Ebenso fehlt noch eine Userverwaltung und andere Sicherheitsaspekte wurden auch noch nicht berücksichtigt. ausserdem ist das Löschen von nicht mehr benötigten Bildern und anderen Assets auf dem Smartphone ein weiterer wichtiger Punkt.

Abbildungsverzeichnis

Literatur

elfinder, Last Checked 20130213, URL http://elfinder.org/

git, Last Checked 20130213, URL http://git-scm.com/

Github, Last Checked 20130213, URL https://github.com/

optimus, Last Checked 20130213, URL https://wrapbootstrap.com/theme/optimus-dashboard-admin-template-WB0016FX5/

php, Last Checked 20130213, URL http://php.net/

PhpStorm, Last Checked 20130213, URL http://www.jetbrains.com/phpstorm/

sqlite, Last Checked 20130213, URL http://www.sqlite.org/

Xcode, Last Checked 20130213, URL https://developer.apple.com/technologies/tools/