HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Bachelorarbeit

Architekturkonzepte moderner Web-Applikationen

Hochschule für Technik Rapperswil
Frühjahressemester 2013

Erstellt: 7. Juni 2013, 01:07

Autoren Manuel Alabor
Alexandre Joly
Michael Weibel

Betreuer Prof. Hans Rudin
Experte Daniel Hiltebrand
Gegenleser Prof. Dr. Ruedi Stoop

Heute ist der Internetbrowser mehr denn je das Zuhause für komplexe Webapplikationen. Durch die Leistungssteigerung und immer ausgefeilteren Möglichkeiten nutzen ihn heutzutage nicht mehr nur Grossunternehmen als Schnittstelle zwischen Angestellten und Anwendungen.

Erwartungsgemäss können traditionelle Paradigmen und Softwarearchitekturkonzepte mit der neuen Technologie nicht mehr immer Schritt halten und Lösungen für wiederkehrende Probleme bieten. Diese Bachelorarbeit analysiert neue Trends auf dem Gebiet der Webapplikationen kritisch und liefert gleichzeitig zukunftsweisende Inputs für das Unterrichtsmodul "Internettechnologien".

In einer Vorstudie wurden zunächst drei grundlegende Fragen geklärt: Welche der insgesamt 28 Softwarearchitekturkonzepte aus der Aufgabenstellung sollen bearbeitet werden? Was für eine Applikation kann die ausgewählten Konzepte optimal veranschaulichen? Und Mit welcher Technologie sollen die ausgewählten Konzepte demonstriert werden?

Nach der ausführlichen Evaluation des Konzeptkatalogs und der Auswahl von 22 Konzepten entschied sich das Projektteam eine vergleichsweise simple Aufgabenverwaltung für Wohngemeinschaften mit dem Namen "Roomies" umzusetzen. Damit wird sichergestellt, dass der Fokus auf die Demonstration der Architekturkonzepte gelegt werden kann. Als wichtigen Schritt Richtung "Bleeding Edge" wurde zudem entschieden, sowohl server- als auch clientseitig auf JavaScript zu setzen.

Die abschliessende Studie bietet einen tiefen Einblick in die Entwicklung einer JavaScript-basierten Client-Server-Webapplikation. Ausführliche Analysen bewerten die untersuchten Architekturkonzepte kritisch. Ergänzend bietet der Quellcode von "Roomies" anschauliche Beispiele zu jedem der untersuchten Konzepte.

Besondere Aufmerksamkeit wurde dem Konzept "Unobtrusive JavaScript" zuteil. Basierend auf "Backbone.js" wurde eine eigene quelloffene Bibliothek namens "barefoot" entwickelt, welche ohne grösseren Mehraufwand den gleichen Quelltext sowohl im Browser des Endbenutzers als auch auf der Serverkomponente lauffähig macht.

Erklärung der Eigenständigkeit

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Manuel Alabor

Alexandre Joly

Michael Weibel

Danksagungen

Inhaltsverzeichnis

Kapitel 1 Management Summary

1.1. Ausgangslage

In der heutigen Zeit ist der Internetbrowser mehr denn je das Zuhause für immer komplexer werdende Applikationen. Durch die Leistungssteigerung und immer ausgefeilteren Möglichkeiten nutzen ihn heutzutage nicht mehr nur Grossunternehmen als Schnittstelle zwischen Angestellten und Anwendungen. So wäre der Google Mail Client oder auch Facebook, um nur zwei prominente Beispiele zu nennen, in ihrer aktuellen, interaktiven Form ohne die grossen Fortschritte im Bereich Browsertechnologie der letzten Jahre nicht umsetzbar.

Erwartungsgemäss können traditionelle Paradigmen und Softwarearchitekturkonzepte mit der neuen Technologie nicht mehr immer Schritt halten und Lösungen für wiederkehrende Probleme bieten. Diese Bachelorarbeit soll neue Trends auf dem Gebiet der Webapplikationen kritisch analysieren und gleichzeitig zukunftsweisende Inputs für das Unterrichtsmodul “Internettechnologien” liefern.

1.2. Vorgehen

In einer Vorstudie wurden zunächst drei grundlegende Fragen geklärt: Welche der insgesamt 28 Softwarearchitekturkonzepte aus der Aufgabenstellung sollen bearbeitet werden? Was für eine Applikation kann die ausgewählten Konzepte optimal veranschaulichen? Und Mit welcher Technologie sollen die ausgewählten Konzepte demonstriert werden?

Nach der ausführlichen Evaluation des Konzeptkatalogs und der Auswahl von 22 Konzepten entschied sich das Projektteam eine vergleichsweise simple Aufgabenverwaltung für Wohngemeinschaften mit dem Namen “Roomies” umzusetzen. Damit wird sichergestellt, dass der Fokus auf die Demonstration der Architekturkonzepte gelegt werden kann. Als wichtigen Schritt Richtung “Bleeding Edge” wurde zudem entschieden, sowohl server- als auch clientseitig auf JavaScript zu setzen.

1.3. Ergebnisse

Die Studie am Ende dieser Bachelorarbeit bietet einen tiefen Einblick in die Entwicklung einer verteilten, JavaScript-basierten Client-Server-Webapplikation. Die ausführlichen Analysen und persönlichen Stellungnahmen des gesamten Projektteams bewer-

ten die untersuchten Softwarearchitekturkonzepte kritisch. Ergänzend dazu bietet der Quellcode von “Roomies” anschauliche Beispiele zu jedem der 22 Konzepte. Besondere Aufmerksamkeit wurde dabei dem Konzept “Unobtrusive JavaScript” zuteil. Basierend auf “Backbone.js” wurde eine eigene quelloffene Bibliothek namens “barefoot” entwickelt, welche ohne grösseren Mehraufwand den gleichen Quelltext sowohl im Browser des Endbenutzers als auch auf der Serverkomponente lauffähig macht.

1.4. Ausblick

Kapitel 2 Analyse der Aufgabenstellung

Diese Arbeit hat zum Ziel (siehe Anhang J, "Aufgabenstellung"), Architekturkonzepte anhand einer Beispielapplikation darzustellen und diese in das Modul Internettechnologien zu transferieren. Sie verzichtet dabei bewusst auf funktionale Anforderungen an die zu erstellende Applikation. Weiter werden auch keine spezifischen Technologien zur Umsetzung vorgegeben.

Diese sehr offene Ausgangssituation wird lediglich durch die folgenden Ansprüche eingegrenzt:

1. Das Produkt soll unter Verwendung einer oder mehreren Internettechnologien konzipiert und umgesetzt werden.

2. Der zu erstellende Quellcode soll *State Of The Art* Architekturprinzipien ([ROC] & [Til]) exemplarisch darstellen und der interessierten Fachperson als auch Studenten der Vorlesung *Internettechnologien* als Anschauungsmaterial dienen können.

Von diesen zwei Leitsätzen ausgehend kann angenommen werden, dass die Demonstration von Architekturprinzipien klar im Vordergrund stehen soll. Da diese Prinzipien aber auch einem lernenden Publikum (Punkt 2) so interessant wie möglich präsentiert werden sollen, ist eine attraktive Verpackung ebenfalls nicht zu vernachlässigen.

Dieses Kapitel beantwortet im weiteren Verlauf dementsprechend folgende Fragen genauer:

1. Wie werden die vorgegebenen Architekturprinzipien am optimalsten auf eine Beispielapplikation abgebildet?

2. Welche Schritte wurden im Bereich der Entwicklung der Applikationsidee (Produktentwicklung) durchlaufen? Auf welche Aspekte wurde besonders eingegangen?

3. Wie wurde die Technologie zur Umsetzung der generierten Produktidee ausgewählt und welche Kriterien waren dabei ausschlaggebend?

4. Kann mit der Evaluierten Technologie ein verwendbarer Prototyp angefertigt und ein "Proof of Concept" erbracht werden?

2.1. Architekturrichtlinien

Die Aufgabenstellung (Anhang J) definiert zwei Dokumente mit Architekturprinzipien welche mittels der Beispielapplikation demonstriert werden sollen:

- *Resource-Oriented Client Architecture* kurz *ROCA Principles* [ROC]
  Ein Satz von insgesamt 18 Richtlinien, sowohl für den Front- als auch Backend-Layer.
- *Building large web-based systems: 10 Recommendations* von Stefan Tilkov [Til]
  Präsentation mit insgesamt 10 Empfehlungen welche teilweise Layerübergreifend genutzt werden können.

Beide Quellen überschneiden sich in vielen Punkten. Dieser Abschnitt befasst sich mit der genaueren Analyse spezifischer Aspekte und definiert Richtlinien, welche während dieses Projektes gültig sein sollen.

Abschliessend wird aufgezeigt, an welchen Stellen der Beispielapplikation welche Richtlinien am einfachsten und effektivsten demonstriert werden können.

2.1.1. Resource-oriented Client Architecture (ROCA)

Backend-Layer

ID	Prinzip	Erläuterung
RP1	REST	Kommunikation mit Serverressourcen folgt dem REST-Prinzip [Fie00]
RP2	Application Logic	Die Businesslogik der Applikation soll im Backend bleiben.
RP3	HTTP	Ergänzend zu *RP1* findet die Kommunikation mit Serverressourcen über wohldefinierte RESTful HTTP Requests [Irv+] statt
RP4	Link	Alle URI's weisen zu einer eindeutigen Ressource.
RP5	Non-Browser	Die Serverkomponente kann ohne Browser resp. Frontendkomponente (z.B. mit *wget* [Freb] oder *curl* [Hax]) verwendet werden.
RP6	Should-Formats	Serverressourcen können ihre Daten in verschiedenen Formaten (JSON, XML) ausliefern.
RP7	Auth	*HTTP Basic Authentication over SSL* [Uni+] wird als grundlegender Sicherheitsmechanismus eingesetzt. Um ältere Clients abzudecken, können formularbasierte Logins in Verbindung mit Cookies eingesetzt werden. Cookies sollen dabei jegliche zustandsbezogene Informationen enthalten.
RP8	Cookies	Cookies werden nur zur Authentifizierung oder zum Tracking des Benutzers verwendet.
RP9	Session	Eine Webapplikation soll im Grossen und Ganzen zustandslos sein. Ausnahme bildet bspw. die Authentifizierung eines Benutzers.

Tabelle 2.1.: Die ROCA Architekturprinzipien: Backend

Frontend-Layer

ID	*Prinzip*	*Erläuterung*
RP10	Browser-Controls	Die Verwendung von Browser-Steuerelementen (Zurück, Aktualisieren usw.) müssen wie erwartet funktionieren und die Applikation nicht unerwartet beeinflussen.
RP11	POSH	Vom Backend generiertes HTML ist semantisch korrekt [Pilb] und ist frei von Darstellungsinformationen
RP12	Accessibility	Alle Ansichten können von Accessibility Tools (z.B. Screen Reader für Sehbehinderte) verarbeitet werden.
RP13	Progressive Enhancement	Die Darstellung des Frontends soll auf aktuellsten Browsern top aussehen, aber auch auf älteren mit weniger Features verwendbar sein.
RP14	Unobtrusive JavaScript	Die grundlegenden Funktionalitäten des Frontends müssen auch ohne JavaScript verwendbar sein.
RP15	No Duplication	Eine Duplizierung von Businesslogik auf dem Frontend-Layer soll vermieden werden (vgl. *RP2*)
RP16	Know Structure	Der Backendlayer soll so wenig wie möglich über die finale Struktur des HTML-Markups auf dem Frontend "kennen".
RP17	Static Assets	Jeglicher JavaScript oder CSS Quellcode soll nicht dynamisch auf dem Backend generiert werden. Die Verwendung von Präprozessoren (SASS [CWE], LESS [Sel] oder CoffeeScript [Ash]) sind erlaubt und sollen sogar genutzt werden.
RP18	History API	Von JavaScript ausgelöste Navigation soll über das HTML 5 History API [Pila] im Browser abgebildet werden.

Tabelle 2.2.: Die ROCA Architekturprinzipien: Frontend

Bewertung & Einschätzung

Die 18 Richtlinien des ROCA Manifests [ROC] propagieren eine verteilte Systemarchitektur.

Dabei wird die eigentliche Applikationslogik klar auf dem Backend-Layer implementiert. Dieser wird über eine wohldefinierte REST Serviceschnittstelle angesprochen und gesteuert.

Im Frontend-Layer werden zwar die neusten Browserfeatures wie CSS 3 oder verschiedenste HTML 5 Features verwendet, es wird aber auch darauf geachtet dass das User Interface zu älteren Browsern kompatibel bleibt.

Das Projektteam kann alle ROCA Richtlinien unterstützen. Einige Bedenken sind jedoch bezüglich der Prinzipien "RP13: Progressive Enhancement" und "RP14: Unobtrusive JavaScript" anzubringen:

- Ein blindes Umsetzen dieser beiden Richtlinien führt unweigerlich zu Trade-Offs in der User Experience und/oder bedeutet einen Mehraufwand in der Umsetzung.

- Situationsabhängig muss entschieden werden, wie wichtig die Unterstützung von alten Browsern wirklich ist
- Gehört die Optimierung für Suchmaschinen zu den als hoch priorisierten Anforderungen, so sind auch diese beiden Richtlinien von grösster Bedeutung
- Sollen lediglich aktuelle Browser auf unterschiedlichen Geräten (Smartphones, Tablets etc.) bedient werden, kann durch Verwendung von Responsive Design Techniken [Wikg] problemlos die Progressive Enhancement Anforderung umgesetzt werden.

2.1.2. Building large web-based systems: 10 Recommendations

ID	*Empfehlung*
TP1	Aim for a web of looseley coupled, autonomous systems.
TP2	Avoid session state wherever possible.
TP3	Eat your own API dog food.
TP4	Separate user identity, sign-up and self-care from product dependencies.
TP5	Pick the low-hanging fruit of frond-end performance optimizations.
TP6	Don't bother readers with write complexity.
TP7	Apply the Web instead of working around it.
TP8	Automate everything or you will be hurt.
TP9	Know, design for & use web components
TP10	You can use new-fangled stuff, but you might not have to.

Tabelle 2.3.: Tilkovs Empfehlungen

Bewertung & Einschätzung

Stefan Tilkovs Empfehlungen entsprechen praktisch durchgehend den Richtlinien des ROCA Manifests. Er ergänzt diese aber um einige interessante eigene Ideen.

Mit "TP3 Eat your own API dog food" bestärkt er die Forderung "RP1 REST", den Backend-Layer über ein Service-Interface ansprechbar zu machen noch einmal. Er hat sogar den Qualitätsanspruch, dass jedes interne API so umgesetzt wird, dass sie problemlos von einem externen Konsumenten verwendet werden könnte.

Die Modularisierung in einzelne Komponenten beschreibt Tilkov mit einem spezifischen Beispiel "TP4 Separate user identity, sign-up and self-care from product dependencies".

Die Nutzung eines externen Identity Providers macht in der heutigen Internetwelt Sinn. Für den Benutzer bedeutet dies, dass er nicht für jede Webapplikation ein eigenes Konto mit eigenem Benutzernamen und Passwort anlegen muss. Die Applikation wiederum kann sich auf ihre Kernfunktionalität fokussieren und hat im optimalen Fall geringere Implementationsaufwände.

Als sehr positiv bewertet das Projektteam zudem die Ergänzung um einige pragmatische Software Engineering Ansätze im Bereich von “Dont repeat yourself” (DRY):

- “TP7 Apply the web instead of working around it” propagiert die Verwendung aktueller Browserfeatures statt die Implementierung eigener Lösungen.
  Beispiel: Validierung von Formularwerten.
- “TP8 Automate everything or you will be hurt” fordert die Automatisierung jeglicher wiederkehrender Aufgaben. Continuous Integration, Unit Testing und automatisierte Deployments sind auch im Webumfeld aktueller den je.

2.1.3. Projektspezifische Richtlinien

Nach eingehender Auseinandersetzung mit dem ROCA Manifest und den Empfehlungen von Stefan Tilkov entscheidet sich das Projektteam dazu, die Beispielapplikation unter Berücksichtigung aller ROCA Richtlinien durchzuführen.

Ergänzend werden folgende Empfehlungen von Tilkov integriert:

- TP3 Eat your own API dog food
- TP4 Separate user identity, sign-up and self-care from product dependencies
- TP7 Apply the Web instead of working around it
- TP8 Automate everything or you will be hurt

Somit ergeben sich insgesamt 22 Richtlinien, welche es in einer Beispielapplikation zu demonstrieren gilt.

2.1.4. Richtliniendemonstration

Die folgende Matrix zeigt auf, an welchen Stellen der Beispielapplikation welche Architekturrichtlinien veranschaulicht werden sollen.

Es ist weiter ersichtlich, dass jedes Richtlinie an mindestens einer Systemkomponenten demonstriert werden kann.

	Backend					***Frontend***				***Tools***
	Models	Businesslogik	Autentifizierung	Rendering Engine	Service Interface	HTML Markup	CSS Styling	JavaScript Code	Struktur	
RP1 REST					✔					
RP2 Application logic		✔								
RP3 HTTP					✔					
RP4 Link					✔				✔	
RP5 Non-Browser					✔					
RP6 Should-Formats					✔					
RP7 Auth			✔							
RP8 Cookies			✔		✔					
RP9 Session			✔						✔	
RP10 Browser-Controls				✔				✔	✔	
RP11 POSH				✔		✔	✔			
RP12 Accessibility				✔		✔	✔			
RP13 Progressive Enhancement						✔	✔	✔		
RP14 Unobtrusive JavaScript						✔		✔		
RP15 No Duplication	✔	✔		✔				✔		
RP16 Know Structure					✔	✔	✔			
RP17 Static Assets							✔	✔		✔
RP18 History API								✔		
TP3 Eat your own API dog food					✔					
TP4 Separate user identity and sign-up (...)			✔							
TP7 Apply the Web instead of working around					✔	✔	✔	✔	✔	
TP8 Automate everything or you will be hurt										✔

Abbildung 2.1.: Mapping Architekturrichtlinien - Systemkomponenten

2.2. Produktentwicklung

Die Findung einer passenden Produktidee gestaltete sich unter den im vorherigen Abschnitt erwähnten Bedingungen nicht unbedingt als einfach:

Zwar soll der grösste Teil des Arbeitsaufwandes in das Entwickeln einer beispielhaften Architektur fliessen, diese soll aber in einem für Studierende möglichst attraktiven Gewand präsentiert werden.

Der interessierte Leser findet im Anhang G “Produktentwicklung” weitere Details zum Prozess der konkreten Produktentwicklung. An dieser Stelle soll jedoch der Fokus auf der Grundlegenden Idee liegen.

2.2.1. Die Produktidee: Roomies

Roomies soll einer WG ermöglichen, anfallende Aufgaben leicht unter den verschiedenen Bewohnern zu organisieren. Damit auch langweilige Ämtchen endlich erledigt werden, schafft Roomies durch ein Ranglisten- und Badgesystem (Gamification) einen Anreiz, um seine Mitbewohner übertrumpfen zu wollen.

Durch das Aufgreifen einer Thematik aus dem Studentenalltag soll Roomies für Lernende aus dem Modul *Internettechnologien* einen leichten Einstieg in die tendenziell trockene Materie der Softwarearchitektur bieten.

2.2.2. Branding

Der namensgebende Ausdruck *Roomie* stammt aus dem US-amerikanischen und bedeutet soviel wie *Mitbewohner* oder *Zimmernachbar* [Dic]. Passend dazu soll neben dem Namen auch das restliche Produktbranding an die US-amerikanische College-Welt angelehnt werden.

Vom Logo über die Farbwahl bis zum späteren User Interface Design sollen folgende Stilelemente als roter Faden verwendet werden:

1. *Gedimmte* Farben, keine grellen Akzente
2. Simple, aber eingängige und klar definierte Formensprache
3. Serifen-betonte Schriftart als Stilmittel

Logo

Abbildung 2.2.: Roomies Logo im College Stil

Im Anhang G sind weitere Beispiele (Logos usw.) zum Thema *Branding* einsehbar.

2.3. Technologieevaluation

Das Thema "Architekturkonzepte moderner Web-Applikationen" legt den Schluss nahe, neben aktuellsten Architekturprinzipien auch auf die neusten Technologien bei der Umsetzung der Beispielapplikation zu setzen. Im Bereich der Technologiewahl soll dementsprechend ganz bewusst auf etablierte Sprachen wie Java oder C# (in Verbindung mit deren Web-Frameworks) verzichtet werden.

Unter Berücksichtigung der persönlichen Erfahrungen und Einschätzungen aller Projektteilnehmer wurde in Vereinbarung mit dem Betreuer im Zuge einer Evaluation eine Shortlist mit folgenden Technologiekandidaten zusammengestellt:

- *Ruby*
  Ruby hat sich in der näheren Vergangenheit zusammen mit Ruby On Rails im Markt etablieren können. Als relativ junge Technologie durfte es aus diesem Grund bei einer Evaluation nicht ignoriert werden.

- *Java*
  Trotz des einführenden Statements, grosse und etablierte Programmiersprachen von einer engeren Auswahl auszuschliessen, war das Projektteam aufgrund der vorangegangenen Studienarbeit davon überzeugt, dass Java, insbesondere als Backendtechnologie, mit den beiden anderen Sprachen mithalten kann.

- *JavaScript*
  Während den letzten zwei Jahren erlebte JavaScript eine Renaissance: Mit node.js schaffte es den Sprung vom Frontend-Layer ins Backend und erfreut sich in der OpenSource als auch der Industrie-Community grösster Beliebtheit.

Im folgenden Abschnitt werden übergreifende Bewertungskriterien definiert, welche anschliessend auf alle drei Technologiekandidaten, resp. deren Frameworks angwendet werden können.

2.3.1. Bewertungskriterien & Gesamtbewertung

Die folgende Tabelle definert sechs Kriterien, welche zur Bewertung einer Technologie oder eines Frameworks jeweils mit 0-3 Sternen bewertet werden können.

Die Spalte *Gewichtung* gibt an, als wie wichtig das betreffende Kriterium im Bezug auf die Aufgabenstellung anzusehen ist.

ID	Kriterium	Erläuterung	Gewichtung
TK1	Eigenkonzepte	Wie viele eigene Konzepte & Ideen bringt eine Technologie resp. ein Framework mit? Viele spezifische Konzepte bedeuten eine steile Lernkurve für Neueinsteiger und sind darum tendenziell ungeeignet für die Verwendung in einem Unterrichtsmodul. *Hohe Bewertung = Wenig Eigenkonzepte*	★★★
TK2	Eignung	Wie gut eignet sich eine Technologie oder ein Framework für die Demonstration der Architekturrichtlinien? Geschieht alles "hinter" dem Vorhang oder sind einzelne Komponenten einsehbar? *Hohe Bewertung = Hohe Eignung*	★★★
TK3	Produktreife	Wie gut hat sich das Framework oder die Technologie bis jetzt in der Realität beweisen können? Wie lange existiert es schon? Gibt es eine aktive Community und wird es aktiv weiterentwickelt? *Hohe Bewertung = Hohe Produktreife*	★★★
TK4	Aktualität	Diese Arbeit befasst sich um "moderne Web-Applikationen". So sollte auch die zu verwendende Technologie gewissermassen nicht von "vorgestern" sein. *Hohe Bewertung = Hohe Aktualität*	★
TK5	"Ease of use"	Wie angenehm ist das initiale Erstellen, die Konfiguration und die Unterhaltung einer Applikation? Führt das Framework irgendwelchen "syntactic sugar" [Ray96] ein um die Arbeit zu erleichtern? *Hohe Bewertung = Hoher "Ease of use"-Faktor*	★★
TK6	Testbarkeit	Wie gut können die mit dem Framework oder der Technologie erstellte Komponenten durch Unit Tests getestet werden? *Hohe Bewertung = Hohe Testbarkeit*	★★

Tabelle 2.4.: Bewertungskriterien für Technologieevaluation

Für jedes Framework wird abschliessend eine Gesamtbewertung in Form einer Zahl errechnet. Dies geschieht mit folgender Formel:

$$\frac{\sum_{n=1}^{6} Bewertung_{TKn} \times Gewichtung_{TKn}}{6}$$

Abbildung 2.3.: Berechnungsformel Gesamtbewertung

2.3.2. Ruby

Insbesondere mit dem Framework *Ruby on Rails* [Hanc] wurde Ruby für die Entwicklung von Umfangreichen Webapplikationen seit Veröffentlichung in den 90ern immer beliebter. Ruby bringt viele Konzepte wie Multiple Inheritance (in Form von Mixins) oder die funktionale Behandlung von jeglichen Werten/Objekten von Haus aus mit.

Für den Einsteiger etwas verwirrend setzt es zudem auf eine für den Menschen "leserlichere" Syntax als beispielsweise von Java oder anderen verwandten Sprachen gewohnt. Folgende Codebeispiele bewirken die selbe Ausgabe auf der Standardausgabe, unterscheiden sich aber deutlich in ihrer Formulierung:

```
1 if(!enabled) {
2   System.out.println("Ich bin deaktiviert!");
3 }
```

Quelltext 2.1: Negierte if-Abfrage in Java

```
1 puts "Ich bin deaktiviert!" unless enabled
```

Quelltext 2.2: Negierte if-Abfrage in Ruby

Während der kurzen Technologieevaluationsphase wurde im Bereich Ruby das Hauptaugenmerk auf *Ruby on Rails* gelegt. Insbesondere die Scaffoldingtools und der daraus generierte Quellcode wurde näher begutachtet. Die Resultate sind wiederum auf die sechs Bewertungskriterien appliziert worden.

Bewertung Ruby on Rails

	TK1 Eigenkonzepte	*TK2 Eignung*	*TK3 Produktreife*	*TK4 Aktualität*	*TK5 "Ease of use"*	*TK6 Testbarkeit*	*Gesamtbewertung*
Ruby on Rails	★	★	★★★	★	★★★	★★	4

Tabelle 2.5.: Bewertung Ruby on Rails

Interpretation

Nach genauerem Befassen mit Ruby on Rails sind die Einschätzung des Projektteams gespalten.

Zum Einen minimiert Ruby on Rails den Aufwand für das Erledigen von Routineaufgaben extrem (Scaffolding). Der generierte Code ist sofort verwendbar und, gute Ruby-Kenntnisse vorausgesetzt, gut erweiterbar.

Zum Anderen ist aber gerade die Einfachheit, wie bspw. Controllers oder Models erzeugt und in den Applikationsablauf eingebunden werden, nicht besonders optimal wenn es darum geht, Architekturrichtlinien eindeutig und klar demonstrieren zu können.
Dies ist vorallem so, weil Frameworks wie Ruby on Rails Standardmässig bereits viele der Richtlinien implementieren und es dadurch etwas schwieriger wird, die genauen Unterschiede zeigen zu können.

Unter dem Vorbehalt, dass Ruby on Rails für die Demonstration der definierten Architekturrichtlinien evtl. nicht die richtige Wahl sein könnte, kann das Projektteam nur eine bedingte Empfehlung für das Ruby Framework abgeben.

2.3.3. Java

Schon vor dieser Bachelorarbeit kann das Projektteam Erfahrungen mit Java vorweisen. Zum Einen aus privaten und beruflichen Projekten, zum Anderen auch ganz themenspezifisch aus der Studienarbeit, welche ein Semester früher durchgeführt wurde.

Als Teil einer grösseren Applikation wurde dort ein Webservice mit REST-Schnittstelle umgesetzt. Zum Einsatz kamen diverse Referenzimplementierungen von Java Standard API's. Die sehr positiven Erfahrungen mit der dort orchestrierten Zusammenstellung von Bibliotheken legen den Schluss nahe, diese auch für eine potentielle Verwendung innerhalb dieser Bachelorarbeit wiederzuverwenden.

Der Studienarbeit-erprobten Kombination sollen jedoch auch andere Alternativen gegenübergestellt werden. Insgesamt ergeben sich so folgende Analysekandidaten im Bereich der Technologie *Java*:

Framework	*Erläuterung*
Studienarbeit-Zusammenstellung	Die Zusammenstellung von *Google Guice*, *Jersey*, *Codehaus Jackson* sowie *EclipseLink* hat sehr gut harmoniert. Die Verwendung von einem Java-fremden Framework für die Implementierung des Frontends wäre jedoch erneut abzuklären.
Spring	Spring hat sich in den letzten Jahren in der Industrie etablieren können. Es bietet eine Vielzahl von Subkomponenten (MVC, Beanmapping etc.).
Plain JEE	Java Enterprise bietet von sich aus viele Features, welche die Frameworks von Dritten unter anderen Ansätzen umsetzen. Es gilt jedoch abzuwägen, wie gross der Aufwand ist, um beispielsweise eine REST-Serviceschnittstelle zu implementieren.
Vaadin	Vaadin baut auf Googles GWT und erlaubt die serverzentrierte Entwicklung von Webapplikationen.
Play! Framework	Seit dem Release der Version 2.0 im Frühjar 2012 erfreut sich das Play! Frameworks grosser Beliebtheit. Insbesondere die integrierten Scaffolding-Funktionalitäten und MVC-Ansätze werden gelobt.

Tabelle 2.6.: Shortlist Analysekandidaten Java

Bewertungsmatrix

	TK1 Eigenkonzepte	*TK2 Eignung*	*TK3 Produktreife*	*TK4 Aktualität*	*TK5 "Ease of use"*	*TK6 Testbarkeit*	*Gesamtbewertung*
Studienarbeit-Zusammenstellung	★★★	★★★			★★★	★★	5
Spring			★★	★★★		★	2
Plain JEE		★★	★★★	★★★		★★★	4
Vaadin	★		★★★	★★		★★★	3
Play! Framework	★		★★	★★		★★★	3

Tabelle 2.7.: Bewertungsmatrix Java Frameworks

Interpretation

Plain JEE, *Vaadin* und *Play! Framework* spielen ihre Stärken klar in der Produktreife und der dadurch hohen Wartbarkeit resp. Testbarkeit aus. Im Bezug auf die Eigenkonzepte benötigen alle Kandidaten einen gewissen initialen Lernaufwand. *Studienarbeit-Zusammenstellung* arbeitet mit einem klar zugänglichen Schichtenmodell und verwendet über dies hinaus ein komplett vom Backend entkoppeltes Frontend. Zwar wäre eine

solche Lösung auch mit *Spring* oder *Plain JEE* möglich, jedoch versagen diese beiden Frameworks wiederum im Bezug auf die Eignung, die aufgestellten Architekturrichtlinien transparent demonstrieren zu können.

Die Produktreife von *Studienarbeit-Zusammenstellung* ist zu vernachlässigen. Die einzelnen Komponenten für sich haben sich bereits länger in der Praxis bewähren können und sind lediglich genau in dieser Kombination evtl. weniger erprobt.

Aufgrund der vorangegangenen Bewertung soll für Java die *Studienarbeit-Zusammenstellung* mit den Frameworks der beiden anderen Technologien verglichen werden.

2.3.4. JavaScript

JavaScript ist oder war hauptsächlich als "Webprogrammiersprache" bekannt. In den vergangenen Jahren wurde es ausschliesslich innerhalb des Internetbrowsers verwendet um starren Internetseiten zu mehr "Interaktivität" zu verhelfen. Insbesondere den Anstrengungen von Mozilla [Nc] und Google [Gooc] verdankt der früher für schlechte Performance und Codequalität verschrieenen Programmiersprache ihren neuen Glanz: Mit JavaScript sind heute innerhalb des Browsers komplexe und umfangreiche Applikationen problemlos umzusetzen und weit verbreitet.

Die quelloffene V8 JavaScript Engine [Gooc] von Google hat 2009 zudem den Sprung weg vom Browser geschafft: node.js [Joyb] bietet die Engine als eigenständigen Skriptinterpreter an und verfügt über eine umfangreiche, leistungsstarke und systemnahe API. Die neu entstandene Plattform erfreut sich in der Opensource Community grösster Beliebtheit. So sind seit der ersten Veröffentlichung eine Vielzahl an Frameworks und Bibliotheken zu den verschiedensten Themengebieten entstanden.

Auch der Bereich der Webframeworks kann so unterdessen eine Menge an interessanten Libraries aufweisen. Für die Evaluation im Bezug auf diese Bachelorarbeit werden folgende Frameworks genauer analysiert:

Framework	*Erläuterung*
Express.js	Express.js [Expb] baut auf dem Basisframework Connect [Senb] auf. Das damit durchgängig umgesetzte Middleware-Konzept ermöglicht die entkoppelte Entwicklung von simplen Webservern bis hin ausgefeilten Webservices. Zu Beginn genügen max. 10 Zeilen Quellcode, um einen GET-Request [Irv+] entgegen zu nehmen und eine Antwort an den Aufrufer zurück zu senden. Dem Ausbau zu komplexeren Applikationen steht dank der erwähnten Middleware-Architektur jedoch nichts im Wege.
Tower.js	Pate für Tower.js [Pol] steht nach eigenen Angaben des Entwicklers Ruby on Rails. Entsprechend umfangreich sind auch hier die Scaffoldingfunktionalitäten. Das Framework selbst ist mit CoffeeScript [Ash] entwickelt worden. Wie das Ruby-Vorbild bietet auch Tower.js ein ausgewachsenes MVC-Pattern zur Entwicklung eigener Applikationen an.
derby	Das Framework Derby [Cod] nimmt sich das oben eingeführte Express.js zur Grundlage und erweitert es um das Model-View-Controller-Pattern. Es bleibt dabei extrem leichtgewichtig, ermöglicht aber leider keine Browserclients welche der Anforderung des *Unobtrusive JavaScripts* genügen mögen: Derby-Applikationen werden komplett im Browser gerendert und bieten keine Möglichkeit, HTML-Markup auf dem Server zu generieren.
Geddy	Geddy [Ged] ist ein weitere node.js Kandidat, welcher sich Ruby on Rails zum Vorbild nimmt. Dementsprechend vergleichbar ist der Funktionsumfang mit Tower.js. Einer der auffälligeren Unterschiede ist jedoch, dass Geddy auf CoffeeScript vollends verzichtet.
Sails	Sails [Balb] ist das jüngste JavaScript-Framework. Die Grundkonzepte von Ruby on Rails werden mit um einen interessanten "Real-Time"-Aspekt mittels Websockets erweitert. So kann jede Ressource über eine einfache REST-Schnittstelle als auch über eine Websocket-Verbindung angesprochen werden. Damit wird das asynchrone resp. servergesteuerte Aktualisieren des Frontends erleichtert. Zusätzlich beinhaltet Sails bereits eine ORM Bibliothek.

Tabelle 2.8.: Shortlist Analysekandidaten JavaScript

Bewertungsmatrix

	TK1 Eigenkonzepte	*TK2 Eignung*	*TK3 Produktreife*	*TK4 Aktualität*	*TK5 "Ease of use"*	*TK6 Testbarkeit*	*Gesamtbewertung*
Express.js	★★★	★★★	★★	★★★	★★	★★★	6
Tower.js	★★	★★	★	★★★	★★★		4
derby	★★	★	★	★★★	★★		3
Geddy	★★★	★	★★	★★	★★★		4
Sails	★★	★★	★	★★★	★★	★	4

Tabelle 2.9.: Bewertungsmatrix JavaScript Frameworks

Interpretation

Das Kandidatenfeld lässt sich grob in zwei Felder aufteilen:

1. *Grundlagenframework*
   Das Grundlagenframework Express.js bietet ein solides und ausbaubares Fundament

2. *MVC-Frameworks*
   Tower.js, derby, Geddy und Sails setzen auf einer höheren Abstraktionsebene an und maskieren zugrundeliegende MVC-Komplexität.
   Im Falle von Tower.js, derby und Sails wird das oben erwähnte Express.js sogar als Basis verwendet.

Eine besondere Erwähnung verdient Sails: Es bringt als einziges Framework Real-Time-Funktionalitäten mit. Zwar zählt dieses Feature nicht zu den bewerteten Kriterien, trotzdem setzt sich Sails damit von den restlichen Kandidaten ab. Weiter bemerkenswert ist, dass Sails im Vergleich zu den restlichen MVC-Frameworks einen relativ tiefen Einblick in die Konzeptinnereien zulässt.

Nach Punkten gewinnt das Basisframework Express.js ganz klar die Evaluation in der Kategorie JavaScript.

Aufgrund der interessanten Ansätze welche Sails wählt, und dabei im Vergleich zu Express.js bereits ein ORM mitbringt, entscheidet sich das Projektteam trotz der schlechteren Bewertung im Bereich JavaScript für *Sails*.

2.3.5. Entscheidung

Nach eingängiger Auseinandersetzung mit den drei Technologien Ruby, Java und JavaScript ergeben sich für die finale Technologieentscheidung folgende drei Frameworkkandidaten:

	TK1 Eigenkonzepte	*TK2 Eignung*	*TK3 Produktreife*	*TK4 Aktualität*	*TK5 "Ease of use"*	*TK6 Testbarkeit*
Ruby: Ruby on Rails	★	★	★★★	★	★★★	★★
Java: Studienarbeit-Zusammenstellung	★★★	★★★			★★★	★★
JavaScript: Sails	★★	★★	★	★★★	★★	★

Tabelle 2.10.: Finale Frameworkkandidaten für Technologieentscheidung

Im Entscheidungsmeeting vom 6. März (siehe Anhang K "Meetingprotokolle") wurde ausgiebig darüber diskutiert, welche Technologie resp. welches Framework für die bevorstehende Implementation der Beispielapplikation die am geeignetste sein könnte.

Vergleicht man die finalen Kandidaten anhand der Auswahlkriterien, sehen die Bewertungen meist ziemlich ausgeglichen aus. Die Betrachtung des Kriteriums *TK4 Aktualität* lässt jedoch im Bezug auf die grundlegende Idee dieser Arbeit, sich mit neuen Technologien auseinanderzusetzen, bereits eine ziemlich gute Vorsortierung zu. So wird klar, dass *Java* keine Option für eine Umsetzung sein kann.

Weiter besitzt das Framework *Sails* im Vergleich zu *Ruby on Rails* einen ziemlich schlechten *TK3 Produktreife*-Wert. Die Neugier auf eine unverbrauchte Technologie und die Herausforderung, etwas frisches auszuprobieren war jedoch beim Betreuer als auch beim Projektteam ein nicht zu vernachlässigender Faktor. So entschieden sich die Anwesenden abschliessend gegen *Ruby on Rails* und *Ruby*.

Als Gewinner aus der gesamten Evaluation geht so schlussendlich *JavaScript* hervor. Mit dem Framework *Sails* soll die Beispielapplikation auf *node.js* implementiert werden.

2.4. Proof of Concept

In der Technologieevaluation zu JavaScript stachen die beiden Frameworks Express.js [Expb] und Sails.js [Balb] heraus.

Obwohl Express.js stabiler ist und weitaus mehr genutzt wird, ist Sails.js aufgrund der neuen Ideen und interessanten Ansätzen für einen ersten Prototyp ausgewählt worden.

2.4.1. Prototyp mit Sails.js

Um sich ein Bild von Sails.js zu machen wurde ein einfacher Prototyp [Weic] erstellt.

Sails.js verwendet Scaffolding um einerseits ein neues Projekt zu erstellen, andererseits auch um verschiedene Komponenten wie Model oder Controller zu erstellen. Wie im Entity-Relationship Diagramm beschrieben, werden u.a. ein Task und ein Resident Model (sowie entsprechende Tabelle) benötigt.

Um das ORM "Waterline" [Balc] zu testen, wurden diese beiden Models implementiert. Das Task-Model mittels Sails.js definiert sieht so aus:

```
1  var Task = {
2    attributes: {
3      name: "string"
4      , description: "string"
5      , points: "int"
6      , userId: "int"
7      , communityId: "int"
8    }
9  };
10 module.exports = Task;
```

Quelltext 2.3: Task Model in Sails.js

Mit einer Definition eines Models wird automatisch eine REST-API für dieses erstellt.

Damit lassen sich einerseits CRUD-Operationen direkt über HTTP ausführen, andererseits existiert auch die Möglichkeit Socket.IO [Rau] zu aktivieren, um Models direkt von einem offenen Websocket zu verwenden.

Dieses Feature macht Sails.js sehr nützlich für Real-Time-Applikationen.

Um eine effektive HTML-Seite darstellen zu können wird ein Controller sowie eine View benötigt. Dies wurde im Prototyp für Aufgaben (Tasks) implementiert.

```
function findTaskAndUser(id, next) {
  Task.find(id).done(function(err, task) {
    User.find(task.userId).done(function(err, user) {
      next(task, user);
    });
  });
}

function renderResponse(req, res, response) {
  if (req.acceptJson) {
    res.json(response);
  } else if(req.isAjax && req.param('partial')) {
    response['layout'] = false;
    res.view(response);
  } else {
    res.view(response);
  }
}

var TaskController = {
  get: function(req, res) {
    var id = req.param('id');
    findTaskAndUser(id, function(task, user) {
      var response = {
        'task': task,
        'user': user,
        'title': task.name
      };
      renderResponse(req, res, response);
    });
  }
};
module.exports = TaskController;
```

Quelltext 2.4: Task Controller in Sails.js

In diesem Controller wird ein Task aufgrund des GET-Parameters “id” (Linie 52) geladen. Das Code-Stück zeigt die grosse Schwäche des ORMs Waterline [Balc]. Statt dass man auf dem “task”-Objekt direkt “.user” aufrufen könnte, muss man den Umweg über “User.find()” gehen.

Bei einem ausgereiften ORM würden solche Methoden wegen der Definition von Relationen direkt zur Verfügung stehen.

Der Controller verwendet das folgende Template, um das HTML zu rendern:

```
<div id="task-display">
  <h1>Task: <%= task.name %></h1>
  <ul>
    <li>Points: <%= task.points %></li>
    <li>Created At: <%= task.createdAt %></li>
  </ul>
```

```
7   <h2>User: <%= user.name %></h2>
8   <ul>
9     <li>Created At: <%= user.createdAt %></li>
10  </ul>
11 </div>
12 <a href="#" id="reload">Reload!</a>
13 <script>
14   $('#reload').on('click', function() {
15     $.ajax('/task/get/?id=<%= task.id %>&partial=true', {
16       success: function(response) {
17         var $response = $('<div class="body-mock">' + response + '</div>');
18         html = $response.find('#task-display');
19         $('#task-display').replaceWith(html);
20       }
21     });
22   });
23 </script>
```

Quelltext 2.5: Task Template

Mit dem einfachen Script am Schluss des Task Templates wird aufgezeigt, wie man ohne Neuladen der Seite direkt HTML ersetzen kann. Dies ist grundsätzlich Framework-unabhängig und es wurde dabei auch nicht auf “RP14” (Unobtrusive JavaScript) Wert gelegt.

Schlussfolgerung

Wie bereits vorher angemerkt, ist das ORM von Sails.js nicht ausgereift. Weder Assoziationen zwischen Models [Bala] noch das setzen von Indizes ist möglich.

Für das Resident-Model ist es u.a. auch nötig, Facebook IDs zu speichern. Diese sind 64 Bit gross und wegen mangelhafter Unterstützung des ORMs wäre das gar nicht möglich.

Nebst dem ORM ist auch das Framework und die zugehörige Dokumentation wenig umfangreich. Auch die Community war zum Zeitpunkt der Evaluation (siehe Anhang H zur Technologieevaluation) sehr klein. Die Fakten deuten sehr darauf hin, Sails.js nicht für die konkrete Beispielapplokation zu verwenden. Aus diesem Grund soll ein zweiter Prototyp unter Verwendung von Express.js erstellt werden.

2.4.2. Prototyp mit Express.js

Express.js [Expb] ist ein leichtgewichtiges Framework, welches mittels Connect-Middlewares [Senb], leicht erweitert werden kann.

Damit die Eignung von Express.js überprüft werden kann, wurde ebenfalls ein Prototyp [Weia] erstellt. Dieser Prototyp wurde dann später für die eigentliche Applikation weiterverwendet.

Der initiale Startpunkt einer Express.js Applikation ist die Datei “app.js” [Weib]. Dort werden alle benutzten Middlewares registriert, die Datenbank aufgesetzt und Controller registriert.

```
exports.index = function(req, res){
  // first Parameter: Template File to use
  // 2nd Parameter: Context to pass to the template
  res.render('index', { title: 'Express' });
};
```

Quelltext 2.6: Beispiel eines Controllers in Express.js

Ein zugehöriges Template kann folgendermassen aussehen:

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
    <%- LRScript %>
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

Quelltext 2.7: Template in Express.js

In den vorangegangenen zwei Quellcode-Listings 2.6 und 2.7 ist ersichtlich, dass der Applikations-Entwickler sehr grosse Kontrolle über Express.js hat.

Automatisch passiert in Express.js eigentlich nichts. Dies ist ein grosser Vorteil im Bezug auf die Veranschaulichung der Architekturrichtlinien.

ORM

Im Gegensatz zu den anderen evaluierten Frameworks ist bei Express.js kein ORM enthalten. Deswegen musste auch bzgl. des ORMs eine kurze Evaluation gemacht werden. Tabelle 2.11 zeigt die Kriterien und Gewichtung für die Evaluation des ORMs.

ID	Kriterium	Erläuterung	Gewichtung
OK1	Unterstützung DBs	Wieviele unterschiedliche Datenbanken unterstützt das ORM? Werden auch NoSQL-Datenbanken unterstützt? *Hohe Bewertung = Grosse Anzahl an Datenbanken*	★
OK2	Relationen	Sind Relationen definierbar zwischen Tabellen? Verwenden diese die Datenbank-spezifischen Foreign Keys dafür (falls möglich)? *Hohe Bewertung = Relationen möglich und verwendet Datenbank-spezifische Datentypen*	★★★
OK3	Produktreife	Wie gut hat sich das ORM bis jetzt in der Realität beweisen können? Wie lange existiert es schon? Gibt es eine aktive Community und wird es aktiv weiterentwickelt? *Hohe Bewertung = Hohe Produktreife*	★★★
OK4	"Ease of use"	Wie angenehm ist das initiale Erstellen, die Konfiguration und die Unterhaltung von Models? Führt das ORM irgendwelchen "syntactic sugar" [Ray96] ein um die Arbeit zu erleichtern? *Hohe Bewertung = Hoher "Ease of use"-Faktor*	★
OK5	Testbarkeit	Wie gut können die mit dem Framework oder der Technologie erstellte Komponenten durch Unit Tests getestet werden? *Hohe Bewertung = Hohe Testbarkeit*	★★

Tabelle 2.11.: Bewertungskriterien für ORM-Evaluation

	OK1 Unterstützung DBs	OK2 Relationen	OK3 Produktreife	OK4 "Ease of use"	OK5 Testbarkeit	Total
JugglingDB	★★★	★	★	★★	★★	3
Node-ORM2	★★	★★	★	★★★	★	3
Sequelize	★	★★	★★	★★	★	3

Tabelle 2.12.: Bewertungsmatrix JavaScript ORMs

Alle verglichenen ORMs haben eine ähnliche Gesamtbewertung. Bei "Sequelize" stechen jedoch die Produktreife und die Unterstützung für Relationen heraus.

Diese zwei Gründe zusammen mit der Roadmap [Depa] von Sequelize haben schliesslich zur Überzeugung geführt, dass Sequelize die richtige Wahl ist.

Kapitel 3 Richtliniendemonstration

Im Kapitel 2 im Abschnitt “Architekturrichtlinien” ging das Projektteam auf die in der Aufgabenstellung vorgestellten Architekturrichtlinien und Prinzipien ein.

Als Ergebnis dieser Analyse entstand die unter Punkt 2.1.4 vorgestellte, konsolidierte Liste mit Architekturrichtlinien.

Dieses Kapitel soll nun aufzeigen, wie und insbesondere wo eine jeweilige Architekturrichtlinie in der entwickelten Beispielapplikation demonstriert werden konnte.

Dabei wird zuerst verifiziert, ob die potentielle Demonstrationsstelle aus Tabelle 2.1 wie erwartet umgesetzt werden konnte. Anschliessend wird das konkret zur Richtlinie entstandene Beispiel näher analysiert und erklärt.

Am Ende dieses Kapitel wird bewertet, wie gut die jeweiligen Richtlinien an der Beispielapplikation demonstriert werden konnten.

3.1. Übersicht

Die Tabelle 3.1 bietet eine Übersicht über die Analyse aller in Abschnitt 2.1.4 “Richtliniendemonstration” definierten Richtlinien.

Die Spalte “Resultat” kann dabei die Ausprägungen “Positiv”, “Neutral” oder “Negativ” haben.

Richtlinie	*Demonstriert*	*Resultat*	*Bemerkung*
RP1 REST	✔	😐	Versionierung / Caching fehlt
RP2 Application logic	✔	☺	
RP3 HTTP	✔	☺	
RP4 Link	✔	☺	
RP5 Non-Browser	✔	😐	Aufgrund Facebook Login ist die Verwendung ohne Browser schwierig.
RP6 Should-Formats	✔	☺	
RP7 Auth			
RP8 Cookies			
RP9 Session			
RP10 Browser-Controls	✔	☺	
RP11 POSH			
RP12 Accessibility			
RP13 Progressive Enhancement			
RP14 Unobtrusive JavaScript	✔	☺	Funktionalität bleibt ohne JavaScript erhalten
RP15 No Duplication			Gemeinsame Codebasis für Client & Server
RP16 Know Structure	!	☹	Keine Anwendung unter Berücksichtigung des Prinzips RP14
RP17 Static Assets	✔	☺	CSS & View Template Preprocessing
RP18 History API	✔	☺	
TP3 Eat your own API dog food	✔	☺	Wiederverwendbare API umgesetzt
TP4 Separate user identity and sign-up (...)	✔	☺	Facebook als Identity-Provider
TP7 Apply the Web instead of working around	✔	😐	Aufgrund fehlender Standardisierung nicht zufriedenstellend
TP8 Automate everything or you will be hurt	✔	☺	CI, Make

Abbildung 3.1.: Übersicht Architekturrichtlinienanalyse

3.2. RP1 REST

REST beschreibt einen Architekturstil welcher grundlegende Auswirkungen auf die Strukturierung einer komplexen Client-Server Software hat und durch mehrere Randbedingungen gegeben ist:

- Kommunikation zwischen Client und Server ist zustandslos
- HTTP wird als Grundlage verwendet
- URIs sind Identifikatoren für Ressourcen auf dem Server
- Daten-Serialisierung (XML, JSON, etc.) ist nicht vorgegeben

Durch die Bedingung der Zustandlosigkeit wird durch Einsatz dieses Stils ermöglicht, skalierbare Schnittstellen zur Verfügung zu stellen.

Dadurch dass REST-APIs meistens versioniert sind, können Client und Server mehrheitlich unabhängig voneinander entwickelt werden. Sobald die API eine neue Version erstellt und Rückwärtskompatibilität mit den alten Versionen garantiert, kann die Client-Software weiterentwickelt werden.

REST liegt HTTP zugrunde. Es werden explizit die sogenannten HTTP-Verben (GET, POST, PUT, DELETE etc.) verwendet um Ressourcen abzufragen oder zu manipulieren. Durch den Einsatz des HTTP-Standards wird Caching auch direkt ermöglicht. Voraussetzung dafür ist, dass die Software die entsprechenden Caching-Headers in den Antworten sendet.

Damit spezifische Instanzen einer Ressourcen ansprechbar sind, ist die Verwendung korrekter und eindeutiger URIs Pflicht. Jeder Objekt-Typ und jedes Objekt sollte eindeutig identifizierbar sein.

Der Datentyp mit welchem Objekte und Collections übertragen werden ist hingegen nicht definiert. Dem Software Entwickler ist die Serialisierung der Daten somit selber überlassen. Vielfach wird heute jedoch JSON eingesetzt, da es eine kompakte und doch gut lesbare Serialisierungsform ist.

Geplante Umsetzung

Die Beispielapplikation *Roomies* soll REST mit JSON als Datentyp für seine API verwenden.

Jede Ressource welche über die Web-Applikation angefragt werden kann, soll auch über die REST-API verfügbar sein. Um den Sourcecode möglichst schlank halten zu können, soll auch intern die API verwendet werden. Somit ist die API zwingend entkoppelt vom sonstigen Quelltext.

Eine Versionierung der API und das Caching der API-Zugriffe ist geplant.

Konkrete Umsetzung

Die API der Beispielapplikation ist als separater Service-Layer implementiert worden. Auf diese wurd client- als auch serverseitig transparent zugegriffen.

Das Beispiel im Quelltext 3.1 zeigt zwei Definitionen einer solchen Route in der API.

```
23 var controller = require('./controller')
24   , basicAuthentication = require('../policy/basicAuthentication')
25   , authorizedForCommunity = require('../policy/authorizedForCommunity')
26   , communityValidators = require('./validators')
27   , utils = require('../utils')
28   , modulePrefix = '/community';
29
30 module.exports = function initCommunityApi(api, apiPrefix) {
31   var prefix = apiPrefix + modulePrefix;
32
33   // GET /api/community/:id
34   api.get(prefix + '/:id(\\d+)', [
35     basicAuthentication
36     , authorizedForCommunity
37     , controller.getCommunityWithId]);
38
39   // GET /api/community/:slug
40   api.get(prefix + '/:slug', [
41     basicAuthentication
42     , authorizedForCommunity
43     , controller.getCommunityWithSlug]);
44
45   // POST /api/community
46   api.post(prefix, [
47     basicAuthentication
48     , communityValidators.createCommunity
49     , controller.createCommunity
50   ]);
51   //...
52 };
```

Quelltext 3.1: Community API Definition [AJWf]

Bei Zeile 34 wird eine “GET” API für das Abfragen einer WG mit der *ID* definiert. Wie man im definierten Array sieht, werden dabei mehrere Callbacks definiert, welche der Reihe nach aufgerufen werden und sicherstellen, dass jede Anfrage authentifiziert und authorisiert ist.

Zeile 46 definiert eine “POST” Route um eine neue Community zu erstellen. Auch hier wird überprüft ob der Benutzer authentifiziert ist. Zudem wird auch ein Daten-Validator definiert, damit sichergestellt werden kann dass die Daten korrekt und ohne unerwünschte Zeichen sind.

Aus Zeitgründen wurde kein Caching und keine Versionierung implementiert.

Eine Versionierung wäre jedoch durch ein zusätzliches Präfix für die API-Route ohne Probleme lösbar.

Das selbe gilt für das Caching. Jedes Objekt in der Datenbank hat eine Spalte mit der Information, wann der Datensatz zuletzt modifiziert wurde. Durch diese Information kann generisch an einem Punkt ein Caching implementiert werden.

Diskussion

REST ist ein immer wichtigerer Architekturstil und wird in dieser Form überall eingesetzt (siehe Twitter [Twib], Stripe [Str], etc.). Es definiert die wichtigsten Grundbedürfnisse und überlässt applikationsspezifische Entscheidungen dem Software Entwickler.

Seit einiger Zeit ist es immer wichtiger eine generische Schnittstelle auch für kleinere Applikationen zu erstellen. Vielfach wird seit dem aufkommen von Smartphones nicht nur eine Webseite erstellt, sondern auch entsprechende Apps für Mobiltelefone.

Dadurch dass REST auch keine Serialisierungsform festlegt, ist es dem Entwickler der API möglich, verschiedene Datenformate zu unterstützen (siehe auch 3.7 “RP6 Should-Formats”). Dies ermöglicht beispielsweise die Nutzung eines kompakten Datenformats für Mobiltelefone, damit die Datenrechnung nicht strapaziert wird.

Durch die relativ flexible REST-Definition mit einigen wenigen Randbedingungen tut sich aber auch ein wichtiger Diskussionspunkt auf: Mit welchem HTTP-Verb (POST oder PUT) werden Updates gehandhabt? Es gibt hier verschiedene Ansichten und die Diskussion ist bei Weitem nicht abgeschlossen. Eine gute Hilfestellung hierfür bietet “Stack Overflow - HTTP PUT vs POST in REST” [Comb].

Abschliessend gibt das Projektteam folgenden Rat: Wenn es nötig ist eine flexible und generische API zu erstellen, ist REST eine sehr gute Lösung. Es erlaubt Entwicklern bestehendes Know-How über das Web wiederzuverwenden und die API wird durch die Zustandslosigkeit skalierbar.

3.3. RP2 Application Logic

Die “Application Logic” ROCA-Richtlinie legt fest, dass jegliche Applikationslogik auf dem Server implementiert sein muss.

Falls der Server bereits REST einsetzt (siehe vorheriger Abschnitt 3.2, “RP1 REST”), ist dieses Prinzip bereits vorgegeben. Die Trennung von Applikations- und Präsentationslogik wird dadurch erfüllt.

Geplante Umsetzung

Die Beispielapplikation “Roomies” soll eine generalisierte REST-API zur Verfügung stellen, welche sowohl vom Client- wie auch vom Server-Code verwendet werden soll.

Diese API implementiert jegliche Businesslogik zum authentifizieren, authorisieren der Benutzer, sowie die Validierung und Speicherung der Daten.

Die Präsentationslogik wird als separates Layer implementiert. Dabei wird insbesondere dieser Quelltext mittels “barefoot” geteilt.

Konkrete Umsetzung

Die geplante API wurde umgesetzt. Client und Serverside Code verwendet die gleichen API-Definitionen um Daten zu manipulieren. Dabei wird immer sichergestellt, dass nicht authentifizierte oder authorisierte Benutzer keinen Zugriff haben. Weiter wird bei Datenmodifikationen sichergestellt, dass die Daten valid sind.

Durch die Trennung von API und Präsentation ist erfolgreich eine Trennung der Aufgaben (Separation of Concerns [Wikh]) umgesetzt worden. Dies erlaubt es zum Beispiel in Zukunft, die API auf separaten Servern laufen zu lassen.

Weitere Informationen zur Softwarearchitektur können im Kapitel 5 Technische Architektur nachgelesen werden.

Diskussion

Um zu garantieren dass alle Daten korrekt sind und von den richtigen Benutzern ausgeführt werden, muss die Businesslogik auf dem Server liegen. Falls jedoch insgesamt die User Expierence verbessert werden soll, können *zusätzlich* einzelne Teile der Validierung von Daten auf dem Client ausgeführt werden.

In der Beispielapplikation wird keine Validierung auf Clientseite vorgenommen. Durch den Einsatz von Validatoren, welche auch auf dem Client ausführbar sind (siehe [OHa]) und der Unterstützung von Backbone.Models für Validation (siehe [Docb]) könnte das zusätzlich jedoch auch noch auf dem Client gemacht werden.

Im Projektteam ist man sich einig: Die Applikationslogik muss grösstenteils auf dem Server liegen. Einzelne Validierungen o.ä. kann auch auf dem Client ausgeführt werden, jedoch nur *zusätzlich* zur bereits bestehenden Validierung auf dem Server. Diese Richtlinie ist für Client-Server Applikationen ein Muss. Ist das Programm allerdings keine klassische Client-Server Software (zum Beispiel Peer-to-Peer), kann das nicht so angewendet werden. Mit der Verwendung der WebRTC APIs wird somit dieses Prinzip für Teile der Applikation nicht wirklich einsetzbar. Dementsprechend müssen dafür neue Ideen gefunden werden, beziehungsweise bereits etablierte Peer-to-Peer Software- und Netzwerktechniken auf den Browser adaptiert werden.

3.4. RP3 HTTP

Das HTTP Prinzip von ROCA geht in die gleiche Richtung wie Kapitel 3.2 “RP1 REST”. Eine Applikation mit einer REST-API ist die Grundvoraussetzung dafür, dass die Clientseite der Webapplikation mit dem Server RESTful kommunizieren kann.

Geplante Umsetzung

Die Kommunikation des Clients mit dem Server über REST geschieht unter anderem über Backbone Models welche mittels den beiden Methoden “save()” und “sync()” bzw. “fetch()” (siehe [Docc]) mit dem Server kommunizieren.

Um auch normale HTML-Formulare RESTful zu machen, soll ein Weg gefunden werden, mit dem nebst “POST” auch “PUT” und “DELETE” Abfragen ermöglicht werden.

Konkrete Umsetzung

Die geplante Umsetzung ist so implementiert worden. Als Beispiel zeigt Quelltext 3.2 ein *Backbone.Model*, welches eine URL definiert hat und damit synchronisiert werden kann.

```
1  /** Class: Models.Community
2   * Community model as a subclass of <Barefoot.Model at
3   * http://swissmanu.github.io/barefoot/docs/files/lib/model-js.html>
4   */
5  var Barefoot = require('node-barefoot')()
6    , Model = Barefoot.Model
7    , CommunityModel = Model.extend({
8      urlRoot: '/api/community'
9      , idAttribute: 'id'
10     , toString: function toString() {
11       return 'CommunityModel';
12     }
13   });
14
15 module.exports = CommunityModel;
```

Quelltext 3.2: Community Model [AJWm]

Um dieses Model mit Daten abzufüllen, kann wie in Quelltext 3.3 gezeigt, eine Instanz vom “DataStore” geholt werden (Zeile 14) und die “fetch()”-Methode aufgerufen werden (Zeile 33).

```
10 /** Function: initialize
11  * Initializes the view
12  */
13 , initialize: function initialize() {
14   var community = this.options.dataStore.get('community');
15   this.community = community;
16 }
17
18 /** Function: beforeRender
19  * Before rendering it will fetch the community if not done yet.
20  *
21  * Parameters:
22  *   (Promise.resolve) resolve - After successfully doing work, resolve
23  *                               the promise.
24  */
25 , beforeRender: function beforeRender(resolve) {
26   /* jshint camelcase:false */
27   var _super = this.constructor.__super__.beforeRender.bind(this)
28     , resolver = function resolver() {
29       _super(resolve);
30     };
31
32   if(!this.community.has('name')) {
33     this.community.fetch({
34       success: resolver
35       , error: resolver
36     });
37   } else {
38     resolver();
```

```
39   }
40 }
```

Quelltext 3.3: Community Model Synchronisation [AJWq]

RESTful Forms

HTML-Formulare unterstützen nur zwei Arten von HTTP-Methoden, “POST” und “GET” [Mozb]. Um eine Applikation RESTful zu machen, sollten aber zumindest zumindest zusätzlich “PUT” und “DELETE” unterstützt werden.

Die Unterstützung dieser zusätzlichen Methoden erfordert einen Hilfskonstruktion:

- Jedes Formular das nicht “POST” oder “GET” verwendet, erhält ein zusätzliches verstecktes Feld namens *_method* und dem Wert der gewünschten Methode
- Sobald der Benutzer das Formular abschickt, wird vom Server dieses Feld gelesen
- Der Server ruft danach den entsprechenden Controller mit der entsprechenden Methode auf.

Quelltext 3.4 zeigt die Anbindung der entsprechenden “MethodOverride” Middleware [Sena] an eine Express-Applikation auf Zeile 38.

```
18 /** Function: setupHttp
19  * Adds described middlewares to the passed Express.JS application
20  *
21  * Parameters:
22  *   (Object) app - Express.JS application
23  *   (Object) config - Configuration
24  */
25 function setupHttp(app, config) {
26   var db = app.get('db');
27
28   app.use(express.bodyParser());
29   app.use(express.cookieParser());
30
31   app.use(express.session({
32     store: new SequelizeStore({
33       db: db
34     })
35     , secret: config.sessionSecret
36   }));
37
38   app.use(express.methodOverride());
39 }
40
41 module.exports = setupHttp;
```

Quelltext 3.4: HTTP Middleware [AJWj]

Als Beispiel zeigt Quelltext 3.5 das Formular für das markieren einer Aufgabe als erledigt. Auf Zeile 7 wird das entsprechende Feld definiert.

```
6  <form class="reset-style" action="/community/{{community.slug}}/tasks/{{id}}"
       data-task-id="{{id}}" method="post">
7    <input type="hidden" name="_method" value="put"/>
8
9    <input type="hidden" name="name" value="{{name}}"/>
10   <input type="hidden" name="reward" value="{{reward}}"/>
11   <input type="hidden" name="dueDate" value="{{formatDate dueDate}}"/>
12   <input type="hidden" name="fulfillorId" value="{{resident.id}}"/>
13   <input type="hidden" name="fulfilledAt" value="{{formatDate now}}"/>
14
15   <button type="submit" class="reset-style">
16     <i class="icon-check-empty"></i>
17   </button>
18 </form>
```

Quelltext 3.5: Formular mit verstecktem *_method* Feld [AJWp]

Dieser Workaround kann nicht als wirklich schön bezeichnet werden, jedoch funktioniert er ohne Probleme und in allen Browsern.

Diskussion

"RP3 - HTTP" ist eine Richtlinie, welche nach Auffassung des Projektteams nur hinsichtlich der Formulare bereichernd für den Konzeptkatalog ist. Die restliche REST Thematik ist schon mit RP1 REST abgedeckt und sollte somit hinlänglich Thematisiert worden sein.

Die Verwendung von "PUT", "DELETE" etc. für Formulare hat sowohl gute wie auch schlechte Seiten: Einerseits können die gleichen API-Routen (und somit der gleiche Code) wie für normale API-Aufrufe verwendet werden. Dafür ist allerdings ein relativ unschöner (aber doch relativ eleganter) Workaround zu verwenden.

Falls die eigene Applikation komplett mittels einer REST-API aufgebaut wurde ist diese Hilfskonstruktion sicher ein gehbarer Weg, ohne zu viel Aufwand zu bringen.

3.5. RP4 Link

Das "Link"-Prinzip ist eng verbunden mit Kapitel 3.11, "RP10 Browser-Controls" und stellt die Anforderung, dass jede Seite eindeutig per Link identifizierbar sein muss.

Wenn Web-Applikationen diesem Prinzip nicht folgen (bspw. Digitec [Dig]), dann können Seiten nicht per Link mit Freunden geteilt werden. Dies ist für einen Nutzer häufig nicht nachvollziehbar und die User Expierence leidet damit.

Bevor Browser die History API [Mozc] implementiert haben, konnte das für JavaScript-lastige Webseiten nur schwer eingehalten werden.

Geplante Umsetzung

Die Beispielapplikation soll eindeutige URLs für Ressourcen aufweisen. Dies soll sowohl für die REST-API gelten, wie auch für die Webseite selber.

Konkrete Umsetzung

Weil die Beispielapplikation so oder so den REST [Fie00] Prinzipien entspricht, ist diese Richtlinie ein Muss.

Jede URL entspricht einer eindeutigen Ressource und kann angesprochen werden. Mithilfe der History API [Mozc] wird auch auf dem Browser die URL geändert, obwohl u.U. nur ein AJAX-Request gemacht wird.

Diskussion

Schon Tim Berners-Lee hat vor Jahren geschrieben: “Cool URIs don’t change” [Ber]. Damit eine URL “cool” ist, muss sie zuerst mal vorhanden und funktional sein.

Auch aus einem weiteren Grund sind URLs nur dann “cool”, wenn sie vorhanden sind und niemals ändern: Mit den heuten Möglichkeiten des “Sharing” auf diversen Sozialen Netzwerken muss es möglich sein, direkt auf die momentane Seite verlinken zu können.

3.6. RP5 Non Browser

Das “Non Browser”-Prinzip beschreibt, dass die Applikationslogik auch ohne die üblichen Browser verfügbar sein muss. Dies ist normalerweise der Fall, wenn man die bereits vorangegangenen Prinzipien einhaltet, insbesondere Kapitel 3.2, “RP1 REST”.

Geplante Umsetzung

Eine REST-API wird laut Kapitel 3.2 umgesetzt und somit ist es möglich, diese Ressourcen mit z.B. “cURL” [Hax] oder “wget” [Freb] abzurufen. Durch die Verwendung von Facebook Login wird es aber eher schwierig, eine authentifizierte Session zu erhalten. Ein weiterer Login-Mechanismus ist aber nicht geplant.

Konkrete Umsetzung

Eine REST-API wurde umgesetzt. Durch die Anbindung an den Identity Provider “Facebook” (siehe Kapitel 3.21) ist es jedoch unumgänglich, vorher ein valides Login-Cookie anzufordern.

Um dies per “cURL” auf der Kommandozeile zu erreichen, kann wie folgt vorgegangen werden:

1. Ein “cURL” Cookie-Jar erstellen lassen
2. Mit einem anderen Browser auf “Roomies” einloggen
3. Der Wert des Cookies “connect.sid” kopieren

4. Im Cookie-Jar den Wert einsetzen
5. Einen Request auf eine geschützte API-Ressource machen

Dies wird im Quelltext 3.6 veranschaulicht.

```
1 # Schritt 1: Erstellung eines cURL Cookie-Jars in cookies.txt
2 ~ $> curl -X 'GET' 'http://localhost:9001' --cookie-jar cookies.txt --verbose
      --location
3
4 # ------------------------------------------- #
5 # Jetzt müssten die Schritte 2-4 gemacht werden #
6 # ------------------------------------------- #
7
8 # Schritt 5: Request auf eine geschützte API Ressource
9 ~ $> curl -X GET 'http://localhost:9001/api/community/ba/tasks' --cookie
    cookies.txt  --verbose --location
```

Quelltext 3.6: cURL Request auf Roomies

Es ist auch möglich, ohne den Dritt-Browser ein valides Cookie zu bekommen. Auf diese Variante wird hier aber nicht eingegangen.

Wie geplant wurde kein weiterer Login-Mechanismus implementiert.

Diskussion

Durch den Einsatz einer generalisierten API-Schnittstelle mittels REST kann eine Software auch ohne HTML-Parser die entsprechenden Daten auslesen.

Falls dabei Facebook Login eingesetzt wird, ist eine valide Session auf Facebook unumgänglich. Um solche Bedingungen für eine interne Kommunikation zwischen Komponenten nicht zu haben, kann zum Beispiel OAuth [oau] eingesetzt werden.

Wie auch bei Kapitel 3.2 empfiehlt das Projektteam den Einsatz einer REST Schnittstelle. Es muss für die Authentifizierung aber beachtet werden, dass z.B. auch interne Komponenten auf die API zugreifen müssen. Deswegen empfiehlt sich auf jeden fall einen zusätzlichen Authentisierung-Provider zu implementieren.

3.7. RP6 Should-Formats

Wie es bereits im Name ROCA (Resource-oriented Client Architecture) steht, die Anwendung soll Ressourcen-orientiert sein. Damit diese Ressourcen auch in anderen Anwendungen als in einem Browser verwendet werden können, muss das erzeugte HTML entweder maschinenlesbar sein oder es müssen alternative Formate (wie z.B. JSON und/oder XML) verfügbar gemacht werden.

Geplante Umsetzung

Die “öffentlichen” Daten sollen über eine API als JSON verfügbar sein. Diese können über eine einheitliche URL abgerufen werden.

Konkrete Umsetzung

Der Server bietet eine API für jegliche Ressource, welche in einer View benötigt wird. Diese API entspricht einer RESTful-Architektur. Die Daten können über die bekannte GET HTTP-Request-Methode abgefragt werden, mit POST können neue Datensätze hinzugefügt werden, mit PUT verändert und natürlich mit DELETE gelöscht werden.

Diskussion

3.8. RP7 Auth

Geplante Umsetzung

Konkrete Umsetzung

Diskussion

3.9. RP8 Cookies

Die ROCA Richtlinie RP8 legt fest, dass Cookies lediglich zur Authentifizierung oder zur statistischen Analyse (Tracking) eines Benutzers verwendet werden soll.

Geplante Umsetzung

Die Beispielapplikation soll lediglich im Bereich der Anmeldung des Benutzers auf Cookies zurückgreifen. Wie geplant konnte dies erfolgreich umgesetzt werden.

Konkrete Umsetzung

Um einen User während einer Session authentifizieren zu können verwendet Express.js [Expb] die Middleware "*connect-session*" [Senc].

Konnte ein Benutzer erfolgreich authentifiziert werden, sendet der Server dem Client ein Cookie mit einer eindeutigen Session-ID. Diese ID sollte anschliessend mit allen künftigen Requests mitgeschickt werden.

In der Beispielapplikation wird die Session-Middleware wie folgt eingesetzt:

```
1 app.use(express.cookieParser());
2 app.use(express.session({ secret: config.sessionSecret }));
```

Quelltext 3.7: Connect Session Middleware [AJWk]

Diskussion

Das Einbinden von mehr Informationen in Cookies hat mehrere Nachteile:

- Sicherheit
  Wenn z.B. Username und Passwort in ein Cookie gespeichert werden, kann dies ohne Probleme von Drittpersonen mitgelesen werden. Falls HTTPS eingesetzt wird, zwar nur direkt im Browser, trotzdem ist dies ein nicht-gangbares Risiko.
  Ausserdem ist es einem User möglich, Cookies abzuändern. Ohne Validierung und/oder dem Einsetzen eines Message Authentication Codes ist dies auf dem Server nicht erkennbar.
- Datenmenge
  Cookies werden mit jeder Anfrage und jeder Antwort des Servers mitgeschickt. Zwar ist die Grösse eines Cookies beschränkt, trotzdem wird die Datenmenge unnötig höher.

Aus diesen Gründen sollte unbedingt darauf Verzichtet werden, Cookies für andere Zwecke als für die Wiederkehrende Authentifizierung über eine Session-ID oder für Tracking zu verwenden.

3.10. RP9 Session

Geplante Umsetzung

Konkrete Umsetzung

Diskussion

3.11. RP10 Browser-Controls

> Man befindet sich auf einem Online-Shop und sucht mit Hilfe der eingebauten Suche, Zubehör für seine neu ergatterte Fotokamera. Das Suchresultat ist aber noch zu grob. Deshalb klick man auf dem Zurückknopf des Browsers um auf die Suche zurück zu gelangen. Aber was uns hier erwartet, ist nicht etwa das Suchformular, welches man erwartet hätte, sondern die Startseite.

Dies ist nicht etwa ein fiktives Beispiel, sondern ein reeller Szenario aus dem Online-Shop des Elektronikgrosshändlers Digitec [Dig].

Das ROCA-Prinzip “Browser-Controls” will genau dieses unangenehme Erleben verhindern.

Geplante Umsetzung

“Browser-Controls” ist sehr eng mit dem Prinzip “RP4 Link” verbunden. Indem jede Seite ihre eigene URI hat und diese beim navigieren in die “History” des Browsers eingefügt wird, funktionieren die Standardbedienelemente (Zurück, Vorwärts und Aktualisieren) eines Browsers erwartungsgemäss.

Dies soll auch in der Beispielanwendung “Roomies” der Fall sein. Jeder Seitenwechsel soll über die Browserhistory nachvollziehbar sein und somit dem Benutzer erlauben, die eingebauten Funktionen des Browsers zu benutzen.

Konkrete Umsetzung

Mit der Funktion “Backbone.history.start()” [Docd] bietet Backbone.js eine Anbindung an die History API [Mozc] an und ermöglicht damit JavaScript Applikationen das dynamische navigieren inklusive Aktualisierung der URL bei Seitenwechseln.

```
29 // from modernizr, MIT | BSD
30 // http://modernizr.com/
31 var useHistory = !!(window.history && history.pushState);
32
33 Backbone.history.start({
34   pushState: useHistory
35   , hashChange: useHistory
36   , silent: true
37 });
```

Quelltext 3.8: Activierung der History in Barefoot [Alab]

Diskussion

3.12. RP11 POSH

“Posh”, ausgeschrieben “Plain Old Semantic HTML”, bezeichnet grundsätzlich das Erstellen von HTML-Seiten mithilfe von semantische Elementen. [Pilb]

Semantisches HTML bietet den Vorteil, dass “Search Engine Spiders” die Seite besser verstehen, interpretieren und kategorisieren können. Wird eine Seite gut kategorisiert erscheint sie eher bei einer Suche in einer Suchmaschine wie Google oder Bing.

Geplante Umsetzung

Die HTML-Seiten der Testanwendung sollen eine klare und logische Struktur beinhalten. Alle Seiten haben einen Titel, eine Überschrift und ein Inhalt. Für Überschriften werden die Tags *h1*, für die grösste und wichtigste Übeschrift, bis *h6*, für die schwächste. Tabellen sollen für tabellarische Daten verwendet werden und nicht etwa um die Seite zu strukturieren, wie es in der Vergangenheit oft angewendet wurde.

Konkrete Umsetzung

Um die möglichen semantischen Unkorrektheiten zu reduzieren, wurde es mit Templates gearbeitet. So können beispielsweise Menüs oder Footers definiert und wiederverwendet werden, ohne dass man es bei jeder Benutzung neu schreiben muss. Dies geht sehr stark nach dem DRY (Don’t repeat yourself) Prinzip. So werden Fehler verhindert, indem man sie gar nicht zu schreiben hat.

Im folgenden Beispiel-HTML sieht man einen kleinen Ausschnitt aus dem Haupttemplate von “Roomies”. Man kann klar die Aufteilung der Seite erkennen. Oben den Header mit dem Menü, gefolgt von einem Platzhalter für allfälligen Fehler-, Warnungs-

oder Informationsmeldungen. Die Hauptsektion mit der ID “main”. Darin Befindet sich die ganzen Inhalte. Und zum Schluss der Footer, in welches sich das Abmelden-Knopf befinden wird.

```
27 <body>
28   <header id="menu"></header>
29   <div id="flash-messages" class="row flash-messages"></div>
30   <section id="main"></section>
31   <footer id="footer"></footer>
32 </body>
```

Quelltext 3.9: Layout Definition [AJWh]

Diskussion

3.13. RP12 Accessibility

Alle Seiten sollen semantisch Korrect sein. Damit soll soweit wie möglicht gesichert werden, dass auch andere User Agents, wie zum Beispiel Screen Readers, das strukturierte Markup interpretieren kann.

Geplante Umsetzung

Konkrete Umsetzung

Diskussion

3.14. RP13 Progressive Enhancement

Die Client-Technologien HTML und CSS haben sich über die Jahre weiterentwickelt. Dies aber immer mit dem Hintergedanken “Progressive Enhancement”. Das bedeutet, dass die Entwicklung immer versucht hat, Rücksicht auf die älteren Browserversionen zu nehmen. Deutlich sieht man es bei dem neuen HTML5 Standard. Neue Formulartypen wurden eingeführt wie “tel”, “email”, “color”, “date”, etc. Browser die HTML5 nicht unterstützen, weichen bei einem für sie unbekannten Typ auf einen Default zurück. Im Falle vom Formulartyp wird es als “text” interpretiert.

```
1 <input type="tel" name="telefon">
```

Quelltext 3.10: Inputfeld in HTML5, welches eine Telefonnummer erwartet

Natürlich gibt es keine Regel ohne Ausnahmen. Manche Tags, die im HTML5 Standard eingeführt wurden, werden überhaupt nicht beachtet. Dies kann vor allem bei Versionen kleiner als neun des Internet Explorers beobachtet werden.

Geplante Umsetzung

Die Beispielanwendung soll, wie bereits in 4.2 erwähnt, Internet Explorer 8 und höher, Chrome 25 und höher, Firefox 19 und höher und Safari 6 und höher unterstützen.

Auch Browser auf Smartphones sollten nicht vernachlässigt werden. Hierfür sollen Safari 6 und höher und Android Browser 4.0 und höher unterstützt werden.

Konkrete Umsetzung

Da alle geplanten zu unterstützenden Browser nicht HTML5 interpretieren können, musste ein kleiner Trick angewendet werden, um so die Rückwärtskompatibilität zu erhöhen. Dieser Trick heisst “modernizr” [Ate]. Modernizr ist eine JavaScript-Bibliothek, welche gezielt den verwendeten Browser auf seinen Funktions- und Unterstützungsumfang von HTML5 und CSS3 überprüft. Das Resultat wird dann in einem JavaScript-Objekt gespeichert und zur Verfügung gestellt. Zusätzlich läuft modernizr in eine kleine Schleife, um die neuen Tags (head, section, article, nav, u.w.) zu aktivieren. Das bedeutet, dass diese Tags nicht mit “div” ersetzt werden müssen, wie es sonst der Fall wäre.

Um diese Library einzubinden, hat man nichts weiteres zu tun als das Script im Header nach dem Stylesheet hinzuzufügen.

```
11 <link href="/stylesheets/app.css" rel="stylesheet"/>
12 <script src="/javascripts/lib/custom.modernizr.js"></script>
```

Quelltext 3.11: Einbinden von modernizr [AJWi]

Mit diesem “Trick” konnte erreicht werden, dass in allen geplanten Browser alle Elementen erscheinen, wenn auch nicht immer korrekt. Nicht immer korrekt, weil im Internet Explorer 8, die Darstellung des Bildes im oberen linken Rand nicht dem entspricht, was erwartet wurde.

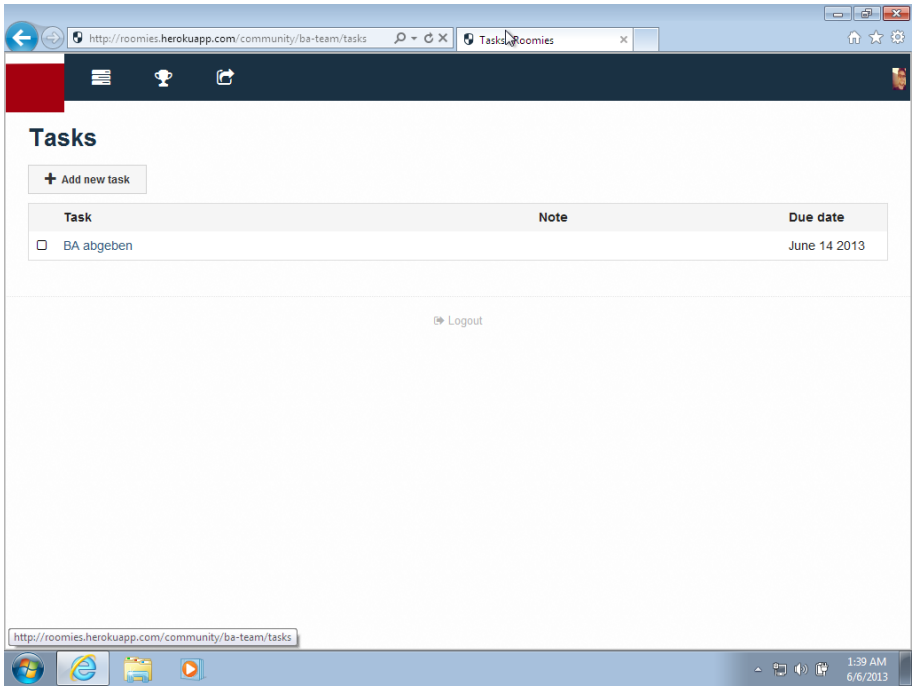


Abbildung 3.2.: Darstellung im Internet Explorer 8

Diskussion

3.15. RP14 Unobtrusive JavaScript

Aktuelle Webapplikationen können grob in zwei Kategorien eingeteilt werden:

Kategorie	***Beispiel***	***Erläuterung***
Statisch	*GitHub* [Git]	User Interface wird auf dem Server gerendert, JavaScript bringt lediglich dynamisch geladene Inhalte, Effekte oder zusätzliche "optionale" Features.
JavaScript Client	*Google Drive* [Gooa]	User Interface wird komplett im Browser mittels JavaScript aufgebaut. Ohne JavaScript keine Funktionalität oder schlechtere User Experience.

Abbildung 3.3.: Kategorisierung aktueller Webapplikationen

Die Kategorie *Statisch* zeichnet sich durch eine hohe Kompatibilität mit allen möglichen Internetbrowsern aus. Durch die Generierung des HTML Markups losgelöst vom schlussendlichen Zielclient, liegt die ganze Verantwortung, vom beschaffen anzuzeigender Daten bis hin zum zusammenstellen des HTML DOM's komplett bei der Serverkompo-

nente.

Zwar kommt auch bei diesem Typus oftmals JavaScript zur Anwendung, meist beschränkt sich dessen Anwendung aber auf die Ergänzung des bereits statisch geladenen Inhaltes. So lädt *Mila* [AGb] beim Seitenwechsel, sofern JavaScript aktiviert, neuen Inhalt über einen AJAX Request. Nach Erhalt des vorgerenderten HTML Markups aus der Antwort ersetzt es entsprechende Inhalte im aktuell angezeigten HTML DOM des Browsers.

Als Programmiersprache auf dem Applikationsserver kommt hier oft Java, Python, PHP, Ruby o.Ä. zum Einsatz.

Beim puren *JavaScript Client* liegt der Programmcode für das Rendern des User Interfaces mit all seinen Inhalten als JavaScript Quelltext vor. Nach erfolgreicher Übertragung zum Internetbrowser initiiert dieser die Erstellung der Applikationsoberfläche im HTML DOM des Clients.

Sollen dynamische Informationen angezeigt werden, müssen diese über eine Serviceschnittstelle beim entsprechenden Anbieter angefragt werden (siehe bspw. Abschnitt 3.2 “RP1 REST”).

Die Verlagerung des User Interface Quelltexts direkt in den Browser hat den Vorteil, dass UI Element effizienter verändert und aktualisiert werden können. Möchten wir bspw. Informationen aus einem Webservice in einer Tabelle anzeigen lassen, können wir diese einmalig laden. Zur effektiven Anzeige muss keine weitere Anfrage an die Backendkomponente getätigt werden, um die bezogenen Informationen in HTML Markup umwandeln zu lassen. Der bereits im Browser verfügbare JavaScript Code übernimmt diese Aufgabe.

Durch diese Vereinfachung entfallen unnötige Wartezeiten zwischen Benutzereingabe und Systemreaktion. Dies resultiert wiederum in einer verbesserten User Experience.

Es ist bereits zu erahnen, dass diese Art von Webapplikation ohne JavaScript-Unterstützung im Browser des Endbenutzers nicht ausgeführt werden kann. Ein Beispiel hierfür liefert der *Google Drive* [Gooa] Webclient. Wie in Abbildung 3.4 ersichtlich verweigert dieser ohne aktiviertes JavaScript die Funktion und zeigt ein leeres Standardlayout mit einer entsprechenden Meldung an.

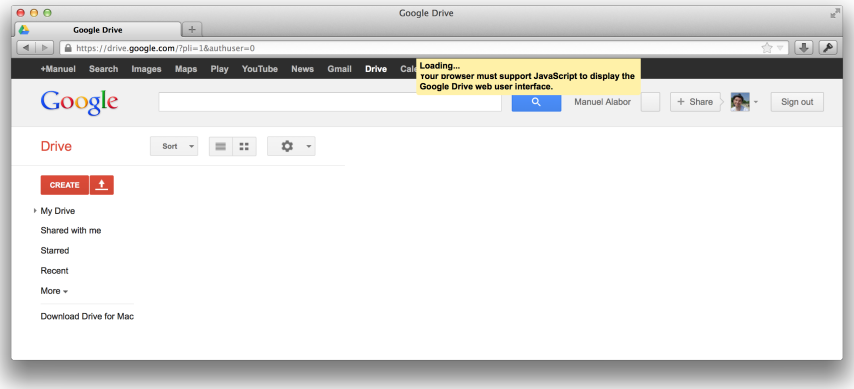


Abbildung 3.4.: *Google Drive* in Firefox 21.0 mit deaktiviertem JavaScript

Die Richtlinie *RP14 Unobtrusive JavaScript* aus dem ROCA Manifest verlangt, dass eine Webapplikation auch bei deaktiviertem JavaScript weiter funktionstüchtig bleibt. Konkret wünscht *RP14* also eine Mischform aus den vorgestellten Applikationstypen.

Bezieht man *Unobtrusive Javascript* neben dem "Wie und Wo" User Interfaces gerendert werden auf die Thematik *Codequalität*, gehört ein weiterer Aspekt zu diesem Bereich: Die Vermischung von HTML Markup und JavaScript Code soll unterbunden werden. Hierzu zeigt der Quelltext 3.12 ein Beispiel, wie es unter keinen Umständen umgesetzt werden sollte.

```
1 <a href="adresses.html" onClick="$('#progressIndicator').removeClass('hidden
      ');showAddresses();return false;">
2   Display Addresses
3 </a>
```

Quelltext 3.12: Beispiel Vermischung von HTML Markup und JavaScript

In den Quelltexten 3.13 und 3.14 ist ersichtlich, wie eine Separierung von JavaScript Logik und HTML Markup optimal implementiert wird.

```
1 <a href="adresses.html" id="showAddresses">Display Addresses</a>
```

Quelltext 3.13: Beispiel sauberes HTML Markup ohne JavaScript

```
1 $(function() {
2   $('#showAddresses', showAddresses);
3
4   function showAddresses(evt) {
5     $('#progressIndicator').removeClass('hidden');
6     // ... show addresses
```

```
7       return false;
8     }
9   });
```

Quelltext 3.14: Beispiel Event-Handler in ausgelagerter JavaScript Datei

Geplante Umsetzung

Als Herausforderung hat das Projektteam geplant, die Beispielapplikation *Roomies* als Mischung der vorgestellten Applikationstypen umzusetzen.

Grundsätzlich sollen Inhalte statisch auf der Backendkomponente gerendert werden. Hat der Benutzer in seinem Browser JavaScript aktiviert, ermöglicht entsprechender Programmcode die Umsetzung der im einleitenden Abschnitt vorgestellten Funktionalitäten eines vollwertigen *JavaScript Clients*.

Zugriffe auf persistente Applikationsdaten sollen wie in 3.2 “RP1 REST” vorgeschlagen in ein entkoppeltes Serviceinterface gekapselt werden.

Um die Wartbarkeit der Codebasis zu optimieren, soll die Separierung von HTML Markup und JavaScript wie beschrieben strikt eingehalten werden.

Konkrete Umsetzung

Während der Implementation der geplanten Lösung wurde sehr schnell klar, das die entstehende Applikation zwar wie erwartet die gewünschte hybride Form aufweisen wird, aber keinesfalls mit der ROCA Richtlinie *RP15 No Duplication* vereinbar sein wird.

Es hätten viele Codefragmente wie die View-Templates (Beispiel siehe Quelltext 3.15) dank der durchgängigen Verwendung von JavaScript auch im Frontend wiederverwendet werden können. Andere, logikintensivere Komponenten wie die Controller zur Steuerung der eigentlichen User Interface Funktionalitäten (Event-Handling, Datenzugriffe etc.) hätten doppelt implementiert werden müssen.

```
31 {{#if user}}
32 <ul class="right">
33   <li class="account">
34     <a href="/resident/{{user.facebookId}}/profile">
35       <span class="item-label">{{user.name}}</span>
36       <img class="avatar small" src="//graph.facebook.com/{{user.facebookId
             }}/picture" />
37     </a>
38   </li>
39 </ul>
40 {{/if}}
```

Quelltext 3.15: Ausschnitt aus dem *Handlebars* [Kat] Template zur Darstellung von Benutzerinformation in der Menüleiste von *Roomies* [AJWo]

Um diesem Umstand gegensteuern zu können teilte sich das Projektteam nach der ersten Entwicklungsiteration in zwei Gruppen:

- Zwei Mitglieder arbeiteten weiter an der Umsetzung der geplanten Use Cases
- Ein Mitglied fokussierte sich auf die Entwicklung einer Möglichkeit, identischen Applikationscode sowohl in der Backend-Komponente für statisches Rendering als auch direkt im Browser als JavaScript Client verwenden zu können.

Aus diesem Prozess entstand das eigenständige Framework *barefoot* [Alaa]. Es setzt auf der verbreiteten Bibliothek *Backbone.js* [Doca] auf und ermöglicht die Verwendung einer einzigen, einheitlichen Codebasis für JavaScript-basierte Webapplikationen (siehe dazu auch Kapitel "Technische Architektur" Abschnitt 5.5).

Mit der Integration des neuartigen Frameworks kann erstmals komplett auf doppelte Codefragmente verzichtet werden. Gleichzeitig profitiert der Endbenutzer von kurzen Lade- und Reaktionszeiten im User Interface. Sollte auf einem Client kein JavaScript verfügbar sein, greift automatisch das klassische servergestützte Rendering und alle Funktionalitäten bleiben zugänglich.

Diskussion

Ähnlich den zwei Typen von Webapplikationen sind zwei kontroverse Strömungen in der Entwicklergemeinschaft erkennbar [Comc]: Die eine Gruppe drängt zur alleinigen Nutzung der neusten Features und tendiert daher eher zu Lösungen mit reinen *JavaScript Clients*. Andere Gruppierungen geben sich vergleichsweise konservativ. Sie argumentieren damit, dass:

- zum Einen die Kompatibilität zu weniger leistungsstarken Browsern resp. JavaScript Engines (Smartphones, alte Browserversionen etc.) gewährleistet sein muss
- zum anderen die Umsetzung einer eben solchen *unobtrusive* Lösung entsprechend aufwändig sei.

Beiden Lagern kann das Projektteam mit der umgesetzten Beispielapplikation entgegentreten: Mit *barefoot* sind Webapplikationen möglich, welche mit einer einzigen, durchgängigen Codebasis sowohl das statische als auch clientseitige Rendering von User Interfaces resp. deren Ausführung ermöglicht. Daraus resultiert eine im Vergleich zur doppelten Implementierung höhere Effektivität im Entwicklungsprozess. Gleichzeitig kann das Erlebnis für den Endbenutzer optimiert werden.

Ob sich die Investition in die Entwicklung einer Webapplikation, welche *RP14 Unobtrusive JavaScript* genügt, lohnt, kann das Projektteam nicht pauschal beantworten. Je nach Anforderungen kann die Erfüllung dieser Richtlinie aber zu einer höheren Akzeptanz bei den Benutzern führen. Frameworks wie *barefoot* können zudem künftig dazu beitragen, die Hemmschwelle für einen *Opt-In* der Software Entwicklern zu reduzieren.

3.16. RP15 No Duplication

Geplante Umsetzung

Konkrete Umsetzung

Diskussion

3.17. RP16 Know Structure

Gehen wir exemplarisch von einer modernen, entkoppelten Applikationsarchitektur aus, welche eine klare Trennung zwischen Front- und Backend vorsieht, so übernimmt der Frontendteil komplett die Erzeugung des User Interfaces auf dem Clientrechner (Beispiele u.A. bei *TodoMVC* [OS]).

Das Backend liefert beim initialen Request ein HTML Grundgerüst, auf welchem die JavaScript Logik des Frontends das finale UI aufbauen wird.

```
1  <html>
2  <head>
3    <meta charset="utf-8">
4    <title>HTML 5 Example App - Client Side UI Logic</title>
5    <link href="/stylesheets/app.css" rel="stylesheet">
6  </head>
7  <body>
8    <div id="main"></div>
9    <script src="/javascripts/app.js"></script>
10 </body>
11 </html>
```

Quelltext 3.16: Beispiel eines HTML Gerüsts zum Rendering eines User Interfaces

Die Zeile 9 im Quelltext 3.16 zeigt beispielhaft die Einbindung der JavaScript-Datei aus Quelltext 3.17. Nach Beendigung des Ladevorgangs wird unter Verwendung des DOM-Manipulators *jQuery* [Fouc] dynamisch ein Titel-Element in das * *-Element mit der ID *main* eingefügt.

```
1  $(function() {
2    $('div#main').html('<h1>Hello World</h1>');
3  });
```

Quelltext 3.17: JavaScript-Datei *app.js* zu Quelltext 3.16

Das ROCA Prinzip 17 *Know Structure* beschreibt den oben aufgezeigten Aufbau und erachtet es als wichtig, dass die Backendkomponente keine Kenntnis über die vom Frontendteil gerenderten User Interface hat. Das Backend soll lediglich die initiale Struktur des Grundgerüsts aus Quelltext 3.16 kennen und später nur noch als Datenlieferant via einer API dienen.

Geplante Umsetzung

Unter Berücksichtigung des Prinzips *RP14 Unobtrusive JavaScript*, näher beschrieben im Abschnitt 3.15, wird nicht geplant, *RP16 Know Structure* in seiner essentiellen Form innerhalb der Beispielapplikation *Roomies* zur Anwendung zu bringen.

Konkrete Umsetzung

Wie in 3.15 dokumentiert, wurde dem ROCA Prinzip *RP14 Unobtrusive JavaScript* eine grössere Gewichtung zugestanden als *Know Structure*. Aus diesem Grund wurde wie geplant darauf verzichtet User Interface Logik nur in der Frontendkomponente zu verwenden.

Die aus 3.16 und 3.15 bekannte geteilte Codebasis zwischen Front- und Backendkomponente zielt sogar absichtlich darauf ab, dass auch im Backend die komplette Struktur des User Interfaces bekannt ist.

Ein grundlegender Punkt wurde jedoch aus *RP16 Know Structure* adaptiert: *Roomies* verwendet wie vorgeschlagen ein HTML Grundgerüst [AJWh], in welches sowohl im Front- als auch Backend die UI Elemente gerendert werden.

Diskussion

Separation of concerns [Wikh] gehört nicht umsonst zu einem der grundlegendsten Prinzipien im Software Engineering. Darauf bezogen hat die eigentliche Intension von *RP16 Know Structure* durchaus seine Daseinsberechtigung: Eine klare Auftrennung von UI Rendering und eigentlicher Geschäftslogik ist erstrebenswert.

Möchte man die Architekturrichtlinie *RP14 Unobtrusive JavaScript* zwecks bestmöglicher Kompatibilität in die Entwicklung einer Applikation mit einfliessen lassen, kommt es unweigerlich zum Konflikt mit *RP16*. Damit die Applikation auch ohne das User Interface Rendering direkt auf dem Client funktionieren kann, muss die Backendkomponente zwingend über Renderingfunktionalität und damit Wissen über die Struktur des resultierenden HTML Markups verfügen. Damit bricht dieses Vorgehen klar mit den Anforderungen *Know Structures*.

Kann auf *RP14 Unobtrusive JavaScript* verzichtet werden, mag *RP16 Know Structure* seine Stärken ausspielen können. Ist jedoch das clientunabhängige Rendering des User Interfaces eine Anforderung an die zu erstellende Lösung, empfiehlt das Projektteam von *RP16* abzusehen.

3.18. RP17 Static Assets

Die “Static Assets” ROCA Richtlinie will dass jeglicher JavaScript Code und jegliche CSS Stylesheets für den Client von statischer Natur ist. Dies bedeutet, dass der Server keine dynamische Generierung von eben diesem Code vornimmt.

Geplante Umsetzung

CSS Stylesheets

Die verwendeten CSS Stylesheets sollen mit dem SASS Präprozessor [CWE] erstellt werden. Zwar muss der eigentliche Stylesheet Code vorneweg einmalig übersetzt werden, die dadurch entstehenden Vorteile beim Entwickeln der Formatierungsinformationen sind jedoch bei Weitem grösser.

Da zudem statische SASS-Quelldateien übersetzt werden, kann der Anspruch keine dynamischen CSS Formatierungen zu generieren befriedigt werden.

Clientside JavaScript

Die JavaScript Applikation für den Client soll auf verschiedene Dateien aufgeteilt werden. Für den späteren produktiven Betrieb ist das Übertragen vieler kleinen Quellcode-Dateien jedoch nicht effektiv.

Aus diesem Grund sollen auf dem Backend alle Client-Quellcode-Dateien zusammengefasst werden können und mit gängigen Methoden zur schnellstmöglichen Übertragung über das Internet optimiert werden.

Ähnlich wie beim SASS-Präprozessor soll auch hier kein dynamischer Code entstehen. Es wird lediglich eine Optimierung der zu übertragenden Informationen vorgenommen.

Konkrete Umsetzung

Beide geplanten Umsetzungen konnten erfolgreich implementiert werden.

CSS Stylesheets

Zur Entwicklungszeit werden die SASS-Quellcodes bei jedem Start der Beispielapplikation neu umgewandelt. Möchte man die Applikation produktiv verwenden, kann die initial beim Ausführen des *install.sh* Skripts durchgeführt werden, oder gezielt über den Kommandozeilenbefehl *make precompile-sass*.

Die daraus entstehende Datei kann anschliessend ohne weitere Veränderungen vom Webserver an den Browser des Benutzers übertragen werden.

Clientside JavaScript

Mit *barefoot* [Alaa] kann zwar der Server-Quellcode auch für die Client-Applikation verwendet werden. Jedoch stellt sich auch hier die Herausforderung, über verschiedenen Dateien verteilte Programmlogik in eine einzige, grössenoptimierte JavaScript-Date zusammenzufassen.

Zu diesem Zweck verwendet *barefoot* fix die *browserify-middleware* für Express.JS [Forb]. Diese Komponente ermittelt anhand vorhandener *require*-Statements im Quelltext, welche CommonJS Module [Fora] zusammengefasst und bereitgestellt werden müssen.

Zusammen mit einigen Zeilen Boilerplate-Code entsteht so eine eigenständige JavaScript-Datei welche als Ganzes an den Client ausgeliefert werden kann. Falls konfiguriert, wird diese zudem von Kommentaren und unnötigen Füllzeichen befreit und mittels Gzip [Wikd] komprimiert. Dies bringt insbesondere im produktiven Betrieb imense Vorteile.

```
95  /* Browserify: */
96  // Roomies uses browserify to package all neceessary CommonJS modules into
97  // one big JavaScript file when delivering them to the client.
98  // Use these settings to customize how that app.js file is created.
99  //
100 // More information about these settings is available here:
101 // https://github.com/ForbesLindesay/browserify-middleware
102 , clientsideJavaScriptOptimizations: {
103   debug: false
104   , gzip: true
105   , minify: true
106 }
```

Quelltext 3.18: Konfiguration der browserify Middleware [AJWa]

Diskussion

Das Projektteam ist davon überzeugt, dass die umgesetzten und aufgezeigten Methoden und Mechanismen für moderne Webapplikationen ein extrem hilfreiches Werkzeug sind.

Im Bereich der clientseitigen JavaScript Entwicklung ermöglichen das erwähnte *browserify-middleware* oder andere Bibliotheken wie RequireJS [jrb] erst das effiziente verteilen von Quellcode auf verschiedene Dateien.

In die gleiche Richtung strebt SASS: Es ergänzt CSS um viele nützliche Features wie Variablen und Mixins. Daneben ermöglicht es aber, entsprechende Bibliotheken vorausgesetzt, eine extreme Vereinfachung der Entwicklung von CSS-Stylesheets, welche mit verschiedenen Browsern kompatibel ist.

Sollten die vorgestellten Techniken für jede Webapplikation verwendet werden?

Das Projektteam ist der Meinung, dass ab einem gewissen Projektumfang uneingeschränkt auf JavaScript-Modularisierungsmethoden und CSS-Präprozessoren gesetzt werden sollte.

3.19. RP18 History API

Mit dem Aufkommen der JavaScript History API [Mozc] ist es ohne Workarounds möglich, dynamische Seitenwechsel im Browser von JavaScript aus veranzulassen.

Geplante Umsetzung

Mithilfe von “Backbone.History” [Docd] soll jegliche URL-Änderung in modernen Browsern mit JavaScript gemacht werden.

Falls JavaScript oder die History API nicht unterstützt wird, soll es trotzdem möglich sein, die Webseite zu benutzen. Die User Experience soll dadurch nicht eingeschränkt werden. Eventuell soll dabei der Hashbang als Fallback verwendet werden.

Konkrete Umsetzung

Das Geplante konnte wie gewünscht umgesetzt werden. Wie im Abschnitt 3.11 “RP10 Browser-Controls” beschrieben, wurde dafür “Backbone.History” verwendet.

In “Backbone.History” ist es möglich den Hashbang als Fallback zu verwenden. Da es aber im Internet Explorer mehr Probleme verursacht als löst, wurde für “barefoot” entschieden, diese Funktion zu deaktivieren. Dies sieht man im Quelltext 3.19 auf den Zeilen 31 und 35.

```
29 // from modernizr, MIT | BSD
30 // http://modernizr.com/
31 var useHistory = !!(window.history && history.pushState);
32
33 Backbone.history.start({
34   pushState: useHistory
35   , hashChange: useHistory
36   , silent: true
37 });
```

Quelltext 3.19: Aktivierung des *History*-API's in barefoot [Alab]

Diskussion

Vor einigen Jahren war die History API noch nicht verfügbar und mehrere Unternehmen verwendeten als Workaround das sogenannte Hashbang. Diese Technik verwendet das setzen von sogenannten “Anchors” in der URL um von JavaScript aus die URL gemäss aktueller Seite anzupassen. Das grosse Problem dieser Technik ist jedoch, dass der Browser dieses URL-Fragment nicht dem Server mitschickt. Als Folge dessen mussten die Webapplikationen folgenden Workaround verwenden, wenn jemand eine URL mit dem Hashbang eintippte:

1. Startseite ausliefern
2. Mittels JavaScript auslesen, welche Seite eigentlich gefragt war
3. Die eigentlich gewünschte Seite ausliefern

Da es nur noch wenige Browser gibt, welche die History API nicht unterstützen (z.B. Internet Explorer 8), gibt es immer noch einige Seiten welche dies als Fallback einsetzen. Da dieser Fallback jedoch umständlich und nicht förderlich für die User Experience ist, wird immer wie mehr darauf verzichtet (siehe z.B. den Blogpost von Twitter über die Abschaffung des Hashbangs [Twia]).

Die History API wird heutzutage von allen modernen Browsern unterstützt. Dementsprechend gibt es kein Herumkommen um diese API. Das Projektteam empfiehlt zudem, Hashbangs nicht zu verwenden.

3.20. TP3 Eat your own API dog food

Eine Applikation mit einer verteilten, entkoppelten Architektur kommt unweigerlich zu einem Punkt, an welchem die einzelnen Komponenten Schnittstellen definieren müssen.

Die durch diesen Prozess entstehenden API's sind klassischerweise auf spezifische Anwendungsfälle zugeschnitten da diese schnellstmöglich die applikationseigenen Anforderungen umsetzen können.

Als weitere Konsequenz werden "unschöne" Interfacemethoden gerne gar nicht erst für externe Konsumenten verfügbar gemacht.

Auf die lange Dauer gesehen besteht die Gefahr, dass ein Flickwerk aus anwendungsfallspezifischen Interfaces resp. Interfacemethoden entstehen.

Mit *Eat your own API dog food* forciert Tilkov von Beginn an die Konzipierung und Umsetzung guter und generischer Schnittstellen für Applikationskomponenten. Als Motivationsfaktor gehört darum auch der Grundsatz, dass keine privaten Methoden existieren sollen, zu seiner Forderung.

Geplante Umsetzung

Für die Beispielapplikation *Roomies* soll ein Servicelayer auf Basis einer HTTP REST Architektur entwickelt werden. Als Datenformat soll JSON verwendet werden.

Jegliche Interaktion mit den Objekten aus der Problemdomäne soll innerhalb dieses Layers gekapselt werden.

Entsprechend der REST Richtlinien (siehe 3.2 "RP1 REST") soll jedes dieser Objekte gezielt abgefragt und manipuliert werden können.

Es sind keine privaten Methoden geplant. Soll ein Objekt vor Zugriffen unbefugter Konsumenten geschützt werden, sind entsprechende Sicherheitsmechanismen umzusetzen.

Konkrete Umsetzung

Wie bereits im Abschnitt 3.2 des Kapitels "Richtliniendemonstration" erläutert, konnte die generische Serviceschnittstelle für alle Objekte aus der *Roomies* Problemdomäne (siehe 5.2 "Domainmodel") umgesetzt werden.

Es wurde komplett auf private Methoden verzichtet. Zum Schutz sensibler Daten wurde wie in den Abschnitten 3.8, 3.9 sowie 3.10 beschrieben ein Session-basierter Authentifizierungsmechanismus via Facebook (siehe 3.21 "TP4 Separate user identity, sign-up and self-care from product dependencies") implementiert.

Diskussion

Klar strukturiertes und wohldefiniertes Schnittstellendesign ist bereits bei einer kleineren Applikation hilfreich, ab einem Umfang von mehreren Komponenten sogar ein absolutes Muss.

Gerade in einer grösseren Service-Landschaft, wie diese oftmals in grossen Konzernen wie Banken anzutreffen ist (Beispiel *BIAN Service Landscape 2.0* [BIA]), ist es jedoch üblich, auch private Interfacemethoden zu unterhalten.

Bis zu einem gewissen Punkt mag es also durchaus berechtigt sein, die eigene Motivation für korrektes Interfacedesign aus der Zurschaustellung nach Aussen zu beziehen. Je nach Anforderungen kann es aber durchaus Sinn machen, von diesem Grundsatz abzusehen.

Für eine Non-Enterprise-Applikation kann das Projektteam *TP3* anstandslos befürworten. Die ausführliche Auseinandersetzung mit der eigenen API resultiert in einer Schnittstelle, welche ohne Weiteres von einer Smartphone App oder einer beliebigen anderen Applikation konsumiert werden kann.

3.21. TP4 Separate user identity, sign-up and self-care from product dependencies

Viele Dienste im Internet verlangen heute nach der Erstellung eines Benutzerkontos. Ein Forum lässt bspw. das Verfassen von Beiträgen erst zu, nachdem sich der potentielle Autor mit seiner E-Mail-Adresse, einem Benutzernamen und einem Passwort erfolgreich registriert hat. Zur Praxis gehört zudem, dass der Benutzer seine Angaben mittels einem eindeutigen Link, welcher ihm per E-Mail zugestellt wird, bestätigt.

Eine solche Registrierung hat für den Benutzer eines Dienstes als auch für den Betreiber dessen auf den ersten Blick hauptsächlich Vorteile:

- Die User Experience kann von Anfang bis Ende auf den Benutzer zugeschnitten und optimiert werden. (Beispiele: Speichern der eigenen Zeitzone für korrekte Datums- und Zeitangaben, Personalisierung der Frontseite usw.)
- Sicherstellung der Identität: Jedes Benutzerkonto resp. jeder Benutzername wird immer von der selben Person verwendet
- Durch gezielte Auswertungen kann der Betreiber sein Angebot optimieren und ggf. Marketing betreiben resp. Werbung in seinem Dienst schalten

Bei genauerer Analyse ergeben sich aber auch nicht zu vernachlässigende negative Faktoren:

- Der Benutzer muss bei jedem neuen Dienst wiederholt ein Konto erstellen und so erneut persönliche Informationen preisgeben
- Der Betreiber muss sich um die Speicherung und Sicherheit der Benutzerinformationen kümmern

- Der Mechanismus zur Identitätsüberprüfung muss vom Betreiber selber umgesetzt werden
- Nach einer gewissen Zeit hat ein Benutzer tendenziell keine Kontrolle mehr darüber, wo er sich registriert und seine Informationen hinterlegt hat
- Die Umsetzung von Zugriffskontrollen (Wer darf welche Informationen eines Benutzers sehen etc.) ist mit entsprechendem Aufwand verbunden

Tilkovs *TP4* schlägt nun die generelle Separierung von Benutzerinformationen und der eigentlichen Registrierung bei einem Dienst vor.

Der Vorreiter OpenID [Foub] und insbesondere die omnipräsenten sozialen Netzwerke erleichtern resp. forcieren eben diese Auftrennung Heute mehr den je.

Ein Konto bei Facebook oder Twitter ermöglicht so den Zugriff auf Dienste Dritter: Möchte ein Benutzer personalisierte News bei *20 Minuten* lesen, kann er sich mit seinem Facebook Konto anmelden [AGa] ohne genauere Informationen zu seiner Person wiederholt angeben zu müssen. Dabei kann er zudem gezielt steuern, welche Informationen an den Dienstbetreiber durch Facebook weitergegeben werden oder nicht.

Abbildung 3.5 visualisiert das Konzept der getrennten Datenhaltung im Bezug auf applikations- und identitätsspezifsche Informationen.

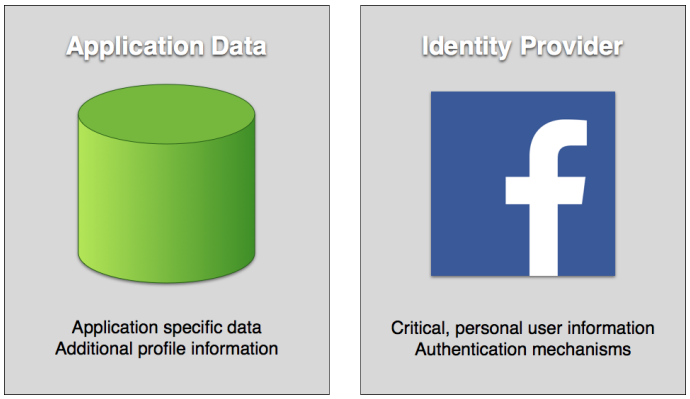


Abbildung 3.5.: Separierung der Applikations- und Identitätsinformationen

In der Abbildung 3.5 ist zudem ersichtlich dass ein *Identity Provider* meist auch den Mechanismus zur effektiven Authentisierung des Benutzers bereitstellt. Viele Anbieter setzen hier auf den de facto Standard *OAuth* [oau] oder verwenden eigene Implementationen.

Geplante Umsetzung

Für die Beispielapplikation *Roomies* hat das Projektteam geplant, Facebook als hauptsächlichen und einzigen *Identity Provider* zu einzusetzen. Dabei soll das offizielle *Facebook*

Login for Web [Faca] SDK eingesetzt werden.

Innerhalb der Beispielapplikation soll lediglich die Facebook ID sowie der Name des Benutzers persistent gespeichert werden. Alle anderen Informationen sollen bei Facebook verbleiben resp. gar nicht erst auf diese zugegriffen werden.

Konkrete Umsetzung

Für die tatsächliche Anbindung des *Facebook Login for Web* [Faca] SDK's wurde wie im Abschnitt 5.4 "Software Layers" des Kapitels "Technische Architektur" beschrieben durch die quelloffene Bibliothek *Passport* [Hanb] umgesetzt.

Die verfügbare *Authentication Strategy* für Facebook [Hana] ermöglicht das einfache Einbinden der Facebook Anmeldemechanismen in jede *Express.js* Applikation.

Zudem werden wie geplant lediglich Facebook ID und Name eines Benutzers in der Applikationsdatenbank abgelegt. Über eine öffentlich zugängliche URL [Facc] wird zusätzlich das aktuelle Profilbild des Benutzers in der Menüleiste von *Roomies* angezeigt, jedoch nicht innerhalb der Applikation zwischengespeichert.

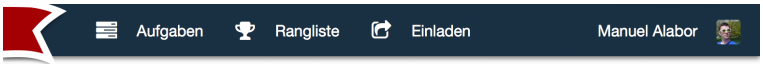


Abbildung 3.6.: Facebook Profilbild des angemeldeten Benutzers der Menüleiste der Beispielapplikation *Roomies*

Diskussion

Lange Zeit hielten sich hartnäckige Vorbehalte gegenüber der Verwendung von Facebook oder ähnlichen Konten für die Anmeldung bei Diensten Dritter. Immer mehr lässt sich jedoch eine gewisse Akzeptanz bei den Benutzern im Internet feststellen. Verbesserte und klarer deklarierte Möglichkeiten zur Anpassung der Zugriffseinstellungen auf hinterlegte Informationen [Facb] haben hier definitiv ihren Teil beigetragen.

Für den Dienstanbieter bringt die Verwendung eines externen *Identity Providers* grosse Vorteile mit sich: Er kann sich auf seine Kernkompetenzen konzentrieren, insbesondere wenn zur Integration umfangreiche Frameworks wie eben *Passport* verwendet werden können.

So ist es, eine entsprechende Architektur vorausgesetzt, möglich, ohne grössere Zusatzaufwände weitere *Provider* anzubinden. Im Falle der vorliegenden Beispielapplikation ist die Integration von *Twitter*, um nur einen *Identity Provider* zu nennen, genau so einfach umsetzbar, wie die Implementation einer komplett eigenständigen Lösung.

Welche Möglichkeiten zur Anmeldung bei einem Dienst angeboten werden sollen ist wie so oft abhängig von verschiedensten Faktoren:

- Datenschutz
- Potentielle Akzeptanz bei Benutzern
- Verbereitung spezifischer *Identity Providers* bei Benutzern

- Usability & User Experience
- Neue oder bestehende Applikation?

Gerade für Applikationen aus einem tendenziell kritischen Geschäftsfeld wie eBanking-Suiten mag es Heute lächerlich klingen, sich via Facebook-Konto anzumelden. Die grosse Verbreitung von PayPal [Pay] als zentralen Anbieter von Geldtransaktionsdiensten zeigt aber, dass selbst hier mit entsprechenden Sicherheitsmassnahmen viel Skepsis erfolgreich wett gemacht werden kann.

Lassen es die Anforderungen einer Applikation zu, so empfiehlt das Projektteam zum heutigen Zeitpunkt die von *TP4 Separate user identity, sign-up and self-care from product dependencies* vorgeschlagene Teilung von Applikations- und Identitätsinformationen resp. Authentifizierungsmechanismen.

3.22. TP7 Apply the Web instead of working around

Der moderne Internetbrowser kapselt eine Vielzahl von leistungsfähigen Funktionen des jeweiligen Hostrechner in ein für den Software Entwickler leicht zu verwendendes Interface. Dazu gehört die Integration von systemnahen Komponenten wie GPU's [Netc] genauso wie die Möglichkeit, gleichzeitig mehrere Fenster oder Tabs für die selbe oder verschiedenen Applikationen resp. Internetseite offen zu halten.

Das Hin- und Herspringen zwischen besuchten Seiten mittels Vorwärts- und Zurück-Schaltflächen gehört seit Beginn der Webära zum festen Bestandteil der User Experience im Internet.

In der Vergangenheit gehörten wiederkehrende Umsetzungen von Funktionen wie der Validierung von Formularinhalten zu lästigen, aber nötigen Ärgernissen. Mit der Einführung der neusten Revision 5 des HTML Standards können gerade solche Aufgaben bequem dem Browser [Pilc] überlassen werden

Mit der immer mächtiger werdenden Formatierungssprache CSS und dessen neuster Version 3 sind heute gestalterische Effekte möglich, welche bis vor Kurzem nur mittels umständlicher Einbindung von Grafikdateien (Stichwort Schlagschatten [Moza] oder Farbverlauf [Moze]) möglich waren.

Mit der Richtlinie *TP7* hält Stefan Tilkov Software Entwickler dazu an, die Werkzeuge welche vom Internetbrowser angeboten werden, gewinnbringend zu nutzen.

Geplante Umsetzung

In der Beispielapplikation sollen gezielt HTML 5 Features verwendet werden:

- Semantisch korrekte Tags (*<header>*, *<section>* etc.) [Pilb]
- Formularvalidierung [Pilc]

Das entstehende, semantisch korrekte HTML Markup soll mit CSS 3 gestaltet werden. Neue Möglichkeiten zur grafischen Darstellung sollen ausgenutzt werden. Die Verwendung von *Media Queries* [Netb] soll die Darstellung auf verschiedenen Bildschirmen (Desktops, Tablets, Smartphones usw.) optimieren und vereinfachen.

Bei der Entwicklung der Front- als auch Backend-Komponente muss zwingend darauf geachtet werden, dass die Verwendung der Browser-Funktionen *Vorwärts*, *Zurück* und *Aktualisieren* zu keinem ungewünschten resp. unerwarteten Verhalten führen.

Konkrete Umsetzung

HTML Markup

Das gerenderte HTML Markup verwendet wie geplant vom HTML 5 Standard eingeführte Funktionen. Quellcode 3.20 zeigt die Verwendung des *<header>* sowie *<nav>* Tags zur Beschreibung des Applikationsmenüs der Beispielapplikation.

```
<header id="menu">
  <div class="fixed-navigation">
    <nav class="navigation" role="navigation">
      <div class="title-area">
        <a href="/" class="banner" title="Roomies"><h1>Roomies</h1></a>
      </div>
      <section class="nav-section">
        <ul class="left">
            <li>
              <a href="/community/ba-team/tasks" title="Aufgaben">
                <i class="icon-tasks icon-large"></i>
                <span class="item-label"> Aufgaben</span>
              </a>
            </li>
            <!-- ... more items -->
        </ul>
        <!-- ... displaying the facebook profile picture -->
      </section>
    </nav>
  </div>
</header>
```

Quelltext 3.20: Ausschnitt des gerenderten HTML Markups der Menüleiste *roomies*

Weiter wurde für das *Fällig bis*-Feld auf der Ansicht *Aufgabe bearbeiten* ein Textfeld vom Typ *date* verwendet. Gerade auf einem Mobile Browser wie *Safari für iPhone* kommt diese Implementation voll zum tragen.

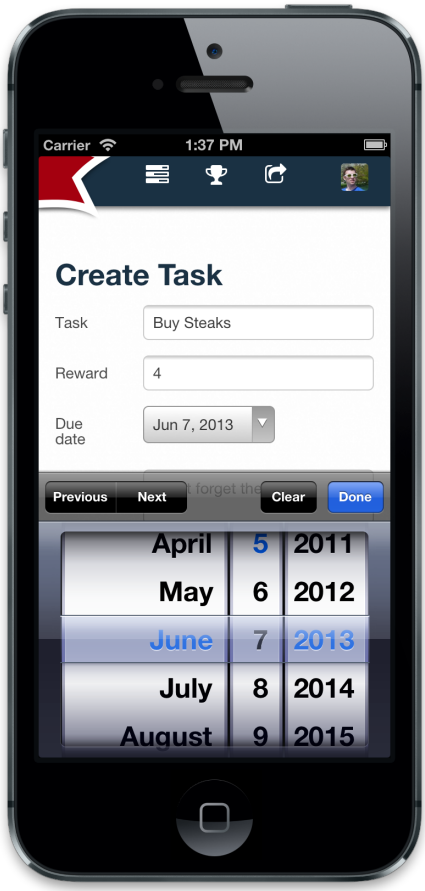


Abbildung 3.7.: Datumsauswahl für ein Textfeld vom Typ *date* in *Safari für iPhone*

CSS

Der erstellte CSS Quellcode macht ausführlichen Gebrauch der neusten CSS 3 Features. So wird für die Darstellung von visuellen Effekten oft auf *box-shadow* oder halbtransparente Farben zurückgegriffen.

Verschiedene Internetbrowser interpretieren CSS Formatierungsbefehle teilweise immer noch sehr unterschiedlich. Um diesem Problem beizukommen wurde die SASS Funktionsbibliothek *Bourbon* [tho] eingesetzt. Verschiedene Mixins ermöglichen mit der Verwendung des SASS Präprozessors [CWE] das Generieren von crossbrowser-kompatiblem CSS Quelltext.

```
4 .button {
5   @include button($button-color-bg);
6   @include border-radius(8px);
7   margin-top: 1px;
```

Quelltext 3.21: Einbindung des *@border-radius* Mixins von *Bourbon* [AJWl]

Im Bereich *Responsive Design* wurde ebenfalls keine Lösung von Grund auf selber entwickelt. Unter Zuhilfenahme der *Foundation* SASS Bibliothek [ZUR] wurde auf einfache Art und Weise ein flexibles User Interface Layout entworfen, welches dynamisch auf die verschiedenen Bildschirmgrössen reagieren kann.

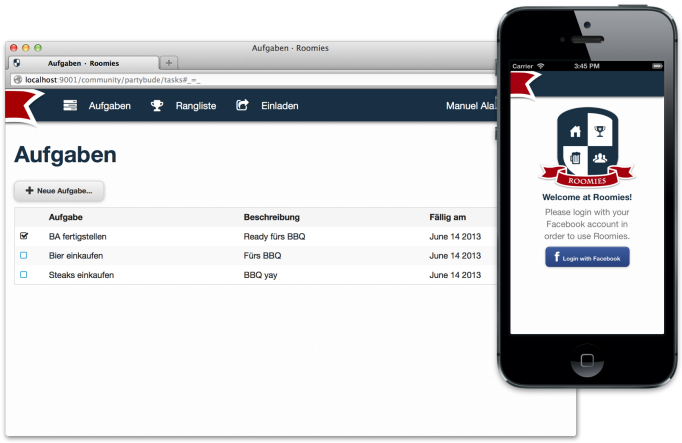


Abbildung 3.8.: Media Queries ermöglichen dynamische Anpassung des Layouts entsprechend der verfügbaren Bildschirmgrösse

Navigation mittels Browserfunktionen

Wie geplant konnte ein einheitliches Navigationsverhalten implementiert werden. Der Benutzer kann sowohl Browsersteuerelemente als auch applikationsinterne Links zur Navigation verwenden, ohne ein unerwartetes Verhalten zu provozieren.

Ausführlichere Informationen zu dieser Thematik sind Abschnitt 3.11 “RP10 Browser-Controls” enthalten.

iOS Webapp Kompatibilität

Zusätzlich zu den geplanten Features wurde die von Apple definierte Spezifikation für *Web Applications* [Inc] in *Roomies* integriert. Entsprechende Metatags im *<head>*-Bereich des HTML Markups ermöglichen eine bessere Integration der Applikation in die iOS Umgebung. Dazu gehört u.A. ein eigenes Bookmark-Symbol für den iOS Homescreen (siehe Abbildung 3.9) als auch angepasste Ladebildschirme während dem *Aufstarten* der Applikation.

Abbildung 3.9.: *Roomies* als Webapp auf dem iPhone Homescreen

Diskussion

Mächtigere Browser ermöglichen die Ausführung immer ausgefeilterer Internetapplikationen. Umfangreiche Browser API's ersparen die Implementierung eigener Überprüfungsmechanismen für Benutzereingaben oder erleichtern die Gestaltung ansprechender User Interfaces.

In Zukunft wird es zudem vermehrt Möglichkeiten geben, wie vermeintlich schwerfällige Webapplikationen in die Betriebssystemumgebung von Smartphones integriert werden können. Diesbezüglich darf man sehr auf Firefox OS gespannt sein, welches den Ansatz von Apples Webapps auf die Spitze treiben wird [Neta].

Leider scheitern die neuen Standards, welche de facto offiziell noch keine sind, momentan oft an den herstellerspezifischen Implementierungen. So erscheint auf dem iPhone ein benutzerfreundlicher Helfer für die Auswahl eines Datums, in *Mozilla Firefox* [Deva] wird das Datumseingabefeld weiterhin als einfaches Textfeld angezeigt. Für eine Webapplikation hat dies zur Folge, dass im Client weiterhin Logik für die Überprüfung von Benutzereingaben implementiert werden muss. Unter dem Aspekt von Sicherheitsmassnahmen mag die grundsätzlich logisch erscheinen, bedeutet dies aber trotzdem eine erhöhten Aufwand im Entwicklungsprozess.

Viele der neuen Browserfeatures können bereits heute angewendet werden. Durch die mangelnde Standardisierung unter den verschiedenen Browserherstellern ergibt sich für das Projektteam jedoch ein eher durchzogener Eindruck der Situation auf diesem Gebiet der Webstandards.

Aus diesem Grund ermutigt das Projektteam zwar zur Verwendung neuester Funktionalitäten, ermahnt jedoch immer eine Fallbacklösung bereitzuhalten, sollte ein Feature auf einem Browser nicht verfügbar sein.

3.23. TP8 Automate everything or you will be hurt

TP8 greift einen allgemein gültigen Vorsatz aus dem Software Engineering auf: Mit "*Don't repeat yourself*" wird zum einen sich wiederholender Quellcode minimiert, als

auch wiederkehrende Routineaufgaben automatisiert.

Entwickelt man den *DRY*-Ansatz weiter, so landet man unweigerlich bei der Verwendung von automatisierten Tests und Deployments mittels Continuous Integration Systemen.

Geplante Umsetzung

Gemäss der Tabelle 2.1 "Mapping Architekturrichtlinien - Systemkomponenten" im Kapitel "Analyse der Aufgabenstellung" soll *TP8* durch die Verwendung verschiedener unterstützender Tools umgesetzt und demonstriert werden.

Die Tabelle 3.1 zeigt eine Auflistung der zu automatisierenden Aufgaben:

ID	Aufgabe
TP8.1	Starten der Beispielapplikation
TP8.2	Ausführung der Unit Tests
TP8.3	Qualitative Überprüfung des Quellcodes (Code Style Guidelines)
TP8.4	Umwandlung von SASS zu CSS Stylesheets
TP8.5	Erstellung von Quellcode Dokumentation

Tabelle 3.1.: Automatisierte Aufgaben (geplant)

Konkrete Umsetzung

Neben den oben erwähten Aufgaben wurde in der konkreten Implementation des Projektes weitere Tasks erfolreich automatisiert:

ID	Aufgabe	CLI Befehl
TP8.1	Starten der Beispielapplikation	*npm start*
TP8.2	Ausführung der Unit Tests	*make test*
TP8.3	Qualitative Überprüfung des Quellcodes (Code Style Guidelines)	*make lint*
TP8.4	Umwandlung von SASS zu CSS Stylesheets	*make precompile-sass*
TP8.5	Erstellung von Quellcode Dokumentation	*make docs*
TP8.6	Vorbereitung von View Templates	*make precompile-templates*
TP8.7	Veröffentlichung von Dokumentation (dieses Dokument, aber auch Quellcode Dokumentation), Testergebnisse und Test Code Coverage Berichten	Travis CI
TP8.8	Umwandlung des LaTeX Quellcodes zur finalen PDF Dokumentation (Dokumentations Repository)	*make*
TP8.9	Erstellen und veröffentlichen von Test Coverage Reports	*make test-coveralls*
TP8.10	Installation der Beispielapplikation	*./install.sh*

Tabelle 3.2.: Automatisierte Aufgaben (umgesetzt)

Als Kernkomponente für die Automatisierung der Aufgaben in Tabelle 3.2 wird ein *Makefile* ([AJWc] und [AJWb]) verwendet. Dieses wird von *GNU Make* [Frea] interpretiert und ermöglicht so das Ausführen von verschiedensten Operationen.

Der Quelltext 3.22 zeigt exemplarisch die Befehlsdefinition für das Ausführen der Unit Tests.

```
99  docs:
100   -mkdir ./docs
101   @NaturalDocs -i ./src -o HTML ./docs -p ./.naturaldocs -xi ./src/server/
          public -s Default style
```

Quelltext 3.22: Ausschnitt Makefile: Code Dokumentation erstellen [AJWc]

Durch den einfache Kommandozeilenbefehl *make docs* wird der in 3.22 definierte Befehl ausgeführt.

Continuous Integration

Die in Tabelle 3.2 vorgestellten Aufgaben *TP8.2* bis *8.9* werden sowohl lokal auf dem Entwicklerrechner als auch auf dem Continuous Integration System ausgeführt.

Entsprechend der Definition im Kaptiel 7 “Qualitätsmanagement” wird hierzu die Open Source Plattform Travis CI [CI] verwendet. Der Quelltext 3.23 zeigt einen Ausschnitt der Datei *.travis.yml* [AJWs]. Diese steuert den Build auf Travis CI.

```
5   before_install:
6       - ./travis/before_install.sh
7
8   after_success:
9       - ./travis/after_success.sh
10
11  language: node_js
12  node_js:
13    - 0.8
14
15  script: "make docs test-coveralls lint"
```

Quelltext 3.23: Ausschnit aus .travis.yml [AJWs]

Diskussion

Muss eine Aufgabe zweimal ausgeführt, sollte dies bereits als Rechtfertigung zur Automatisierung dieser bereits genügen. Dies trifft umso mehr zu, wenn mehrere Entwickler am Entwicklungsprozess beteiligt sind.

Die Praxis zeigt dass die Unterstützung von Projektmitarbeitern durch Automatisierung von Routineaufgaben eine hohe Zeitersparnis und Effektivitätssteigerung mit sich bringt. Wird die Ausführung der Unit Tests erleichtert, lässt sich zudem die Akzeptanz eines Test Driven Development Prozesses erhöhen.

Das Projektteam war positiv davon überrascht, was sich mit frei zugänglichen Continuous Integration Lösungen wie Travis CI [CI] umsetzen lässt. Von der Generierung von Dokumentationen bis hin zur regelmässigen Ausführung von Unit Tests lassen sich ohne grossen Aufwand unbeliebte Aufgaben problemlos Automatisieren und Auslagern.

Das Credo "Don't repeat yourself" ist somit ganz klar Trumpf und so wird auch die Richtlinie *TP8 Automate everything or you will be hurt* vom Projektteam unterstützt.

3.24. Abschliessende Bewertung

todo

Kapitel 4 Anforderungsanalyse

Die Beispielapplikation (siehe 2.2 Produktentwicklung) soll nach einer pragmatischen Software-Entwicklungs-Vorgehensweise implementiert werden.

Das folgende Kapitel beschreibt Anforderungen und Use-Cases, welche die Applikation erfüllen muss.

4.1. Funktionale Anforderungen

ID	*Name*	*Beschreibung*	*Priorität*
F1	WG erstellen	Die Applikation erlaubt es eine WG zu erstellen.	★★★
F2	Einladung	Die Applikation erlaubt es, einen Benutzer in eine WG einzuladen.	★★★
F3	Aufgabe erstellen	Die Applikation erlaubt es, eine Aufgabe zu erstellen.	★★★
F4	Aufgabe erledigen	Die Applikation erlaubt es, eine Aufgabe zu erledigen.	★★★
F5	WG verlassen	Die Applikation erlaubt es, eine WG zu verlassen.	★★
F6	Aufgabe bearbeiten	Die Applikation erlaubt es, eine Aufgabe zu bearbeiten.	★★
F7	Rangliste anzeigen	Die Applikation erlaubt es, eine Rangliste für die Bewohner einer WG anzuzeigen.	★★
F8	Erfolge vergeben	Die Applikation erlaubt es, Erfolge aufgrund von Regeln zu vergeben.	★★
F9	WG auflösen	Die Applikation erlaubt es, eine WG aufzulösen.	★
F10	Bewohnerverwaltung	Die Applikation erlaubt es, die Bewohner einer WG zu verwalten.	★
F11	Inhalte teilen	Die Applikation erlaubt es, Inhalte auf Social Media Kanälen zu teilen.	★

Tabelle 4.1.: Funktionale Anforderungen

4.2. Nichtfunktionale Anforderungen

ID	*Name*	*Beschreibung*
NF1	Antwortzeit	Die Applikation antwortet bei normalen Anfragen innerhalb von 0.2s.
NF2	Desktop Browserkompatibilität	Die Applikation unterstützt Internet Explorer 8 und höher, Chrome 25 und höher, Firefox 19 und höher sowie Safari 6 und höher.
NF3	Mobile Browserkompatibilität	Die Applikation unterstützt Safari 6.0 und Android Browser 4.0.
NF4	Sicherheit	Die Applikation kontrolliert den Zugriff auf geschützte Ressourcen.
NF5	ROCA Prinzipien	Die Applikation entspricht den ROCA [ROC] Prinzipien.

Tabelle 4.2.: Nichtfunktionale Anforderungen

4.3. Use Cases

Das folgende Use Case Diagramm zeigt eine Übersicht der zu implementierenden Use Cases und deren Akteure.

Darauffolgend werden die Use Cases einzeln beschrieben.

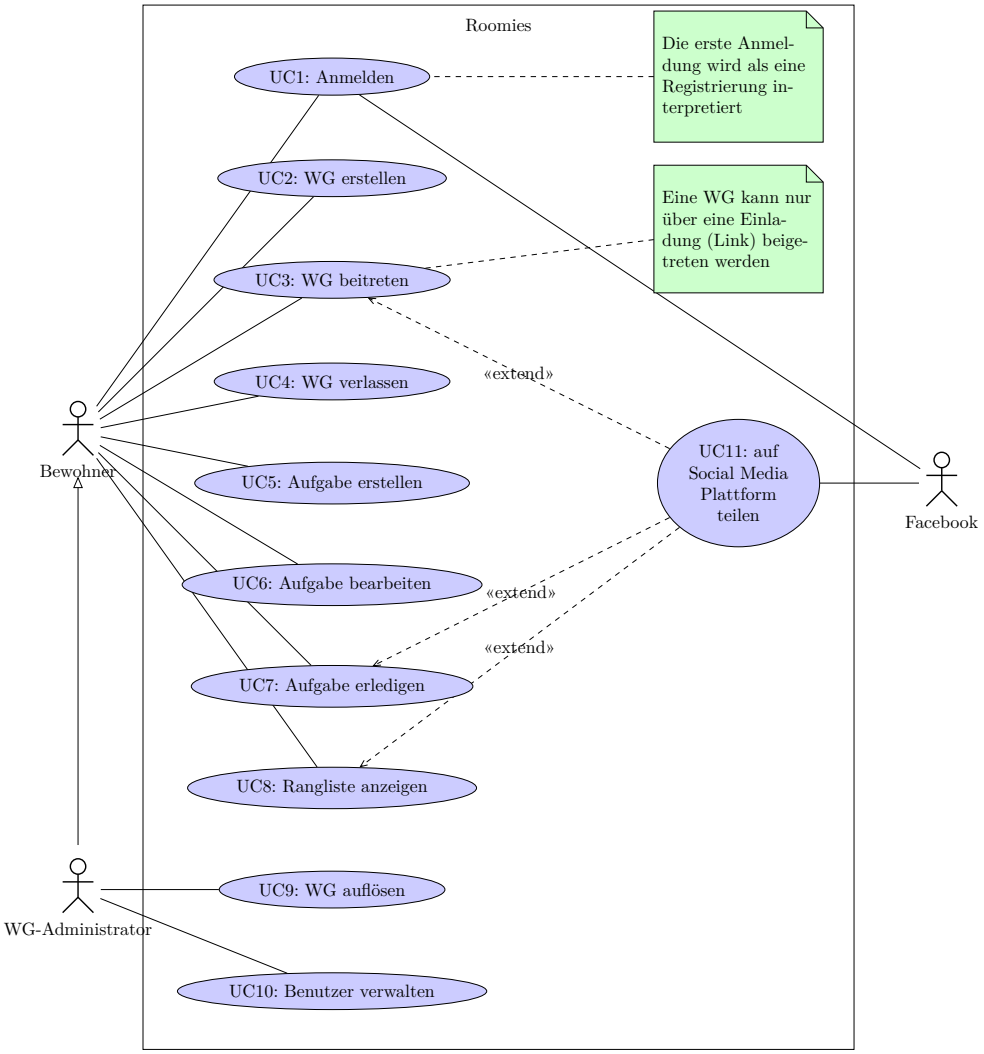


Abbildung 4.1.: Use Case Diagramm

Da die Anforderung F8 (siehe 4.1) durch einen automatischen, sich wiederholenden Auftrag (Cronjob) ausgeführt wird, ist jener keinem Use-Case zugeordnet.

4.3.1. Akteure

Name	***Beschreibung***
Bewohner	Als Bewohner wird ein Anwender der Roomies Anwendung bezeichnet, der zu einer WG gehört. Dieser besitzt die Rechte Aufgaben seiner WG zu verwalten.
WG-Administrator	Der WG-Administrator ist eine Erweiterung des Aktors Bewohner. Der Ersteller einer WG wird automatisch zu deren Administrator. Diese Rolle kann an Bewohner weitergegeben werden (siehe UC4: WG verlassen und UC10: Benutzer verwalten).
Facebook	Facebook ist die Schnittstelle zur Social Media Plattform. Diese ermöglicht dem System auf die Daten des Bewohners, die auf Facebook verfügbar sind, zuzugreifen. So einen einfachen Login anzubieten und Daten zu teilen.

Tabelle 4.3.: Aktoren

4.3.2. UC1: Anmelden

Mapped Requirement	-
Primary Actor	Bewohner
Secondary Actor	Facebook
Story	Der Bewohner startet Roomies. Hier zeigt ihm das System das Anmeldeformular. Der Benutzer meldet sich mittels seines Facebook-Logins an. Das System überprüft die Daten. Sind sie gültig wird der Benutzer auf die WG-Startseite weitergeleitet.

Tabelle 4.4.: UC1: Anmelden

4.3.3. UC2: WG erstellen

Mapped Requirement	F1
Primary Actor	Bewohner
Story	Ein Bewohner hat die Möglichkeit eine WG zu erstellen, falls er noch zu Keiner angehört. Hierfür wählt der Bewohner die Option "WG erstellen". Das System leitet ihn auf das entsprechende Formular und fordert den Bewohner die WG-Daten einzugeben. Nachdem der Bewohner diese eingegeben hat, überprüft das System die Gültigkeit der Daten und leitet den Bewohner auf die Aufgabenseite der WG weiter. Die Rolle WG-Administrator wird dem Bewohner automatisch zugeteilt.

Tabelle 4.5.: UC2: WG erstellen

4.3.4. UC3: WG beitreten

Mapped Requirement	F2
Primary Actor	Bewohner
Story	Um einer WG beizutreten, muss der Link zur Einladung dem "zukünftigen" Bewohner bekannt sein. Ein solcher Link wird vom System erzeugt. Die Verbreitung ist dem WG-Administrator überlassen und ist nicht Teil der Anwendung. Hat ein Bewohner den Link geöffnet, wird er vom System aufgefordert das Beitreten zu bestätigen. Bestätigt der Bewohner diese, wird er als Bewohner der WG "registriert" und zur Aufgabenliste weitergeleitet. Infolge dieses Use Cases kann "UC11: auf Social Media Plattform teilen" angewendet werden.

Tabelle 4.6.: UC3: WG beitreten

4.3.5. UC4: WG verlassen

Mapped Requirement	F5
Primary Actor	Bewohner
Story	Wählt ein Bewohner die Option "WG verlassen", wird er vom System aufgefordert dies zu Bestätigen. Nach der Bestätigung setzt das System den Bewohner als inaktiv und leitet den "Ex"-Bewohner auf eine "Aufwiedersehen-Seite" weiter. Hat der Bewohner als einziger die Rolle "WG-Administrator", so muss er vor dem inaktiv Setzen seine Rolle an einen anderen Bewohner übetragen.

Tabelle 4.7.: UC4: WG verlassen

4.3.6. UC5: Aufgabe erstellen

Mapped Requirement	F3
Primary Actor	Bewohner
Story	Ein Bewohner wählt die Option "Aufgabe erstellen". Das System zeigt das zugehörige Formular. Nachdem der Bewohner es ausgefüllt hat, überprüft das System die Daten, speichert es und leitet den Bewohner zurück auf die Aufgabenseite.

Tabelle 4.8.: UC5: Aufgabe erstellen

4.3.7. UC6: Aufgabe bearbeiten

Mapped Requirement	F6
Primary Actor	Bewohner
Story	Der Bewohner wählt die Aufgabe welche bearbeitet werden soll. Parallel zum "UC5: Aufgabe erstellen" erstellen wird ein Formular mit den bereits vorhandenen Daten der Aufgabe dargestellt. Der Bewohner ändert die Daten. Das System überprüft diese und leitet dann den Bewohner auf die Aufgabenliste weiter.

Tabelle 4.9.: UC6: Aufgabe bearbeiten

4.3.8. UC7: Aufgabe erledigen

Mapped Requirement	F4
Primary Actor	Bewohner
Story	In der Aufgabenliste kann ein Bewohner eine Aufgabe als erledigt setzen. Hierfür wählt er für die entsprechende Aufgabe die Option "erledigt". Das System setzt für den Bewohner die Eigenschaft "Erledigt durch". Infolge dieses Use Case kann "UC11: auf Social Media Plattform teilen" angewendet werden.

Tabelle 4.10.: UC7: Aufgabe erledigen

4.3.9. UC8: Rangliste anzeigen

Mapped Requirement	F7
Primary Actor	Bewohner
Story	Der Bewohner wählt die Option "Rangliste anzeigen". Das System zeigt eine Rangliste aller Bewohner der WG. Infolge dieses Use Case kann "UC11: auf Social Media Plattform teilen" angewendet werden.

Tabelle 4.11.: UC8: Rangliste anzeigen

4.3.10. UC9: WG auflösen

Mapped Requirement	F9
Primary Actor	WG-Administrator
Story	Der WG-Administrator hat die Möglichkeit eine WG aufzulösen. Wird diese Option gewählt, so wird vom WG-Administrator verlangt, dies zu bestätigen. Danach setzt das System alle Bewohner der WG inaktiv und die WG selbst als inaktiv. Der WG-Administrator wird auf die WG-Erstellen Seite weitergeleitet.

Tabelle 4.12.: UC9: WG auflösen

4.3.11. UC10: Benutzer verwalten

Mapped Requirement	F10
Primary Actor	WG-Administrator
Story	Der WG-Administrator besitzt das Recht die Bewohner der WG zu verwalten. Unter Verwalten sind zwei verschiedene Fälle zu unterscheiden. Fall eins besteht darin, einem Bewohner WG-Administratorrechte zu vergeben. Fall zwei ist die Möglichkeit einen Bewohner aus der WG auszuschliessen.

Tabelle 4.13.: UC10: Benutzer verwalten

4.3.12. UC11: auf Social Media Plattform teilen

Mapped Requirement	F11
Primary Actor	Bewohner
Story	Gewisse Interaktionen, wie UC3: WG beitreten, UC7: Aufgabe erledigen oder UC8: Rangliste anzeigen, sollen über die Social Media Plattform Facebook geteilt werden können. Möchte dies der Bewohner tun, so wird ein entsprechender Text erzeugt und die Möglichkeit geboten, diesen auf Facebook zu teilen.

Tabelle 4.14.: UC11: auf Social Media Plattform teilen

Kapitel 5 **Technische Architektur**

5.1. Einleitung

Die Architektur besteht aus den folgenden zwei Haupt-Komponenten:

- Eine Shared Codebase mit *barefoot* [Alaa]
- Eine API-Komponente mit *Express.js* [Expb]

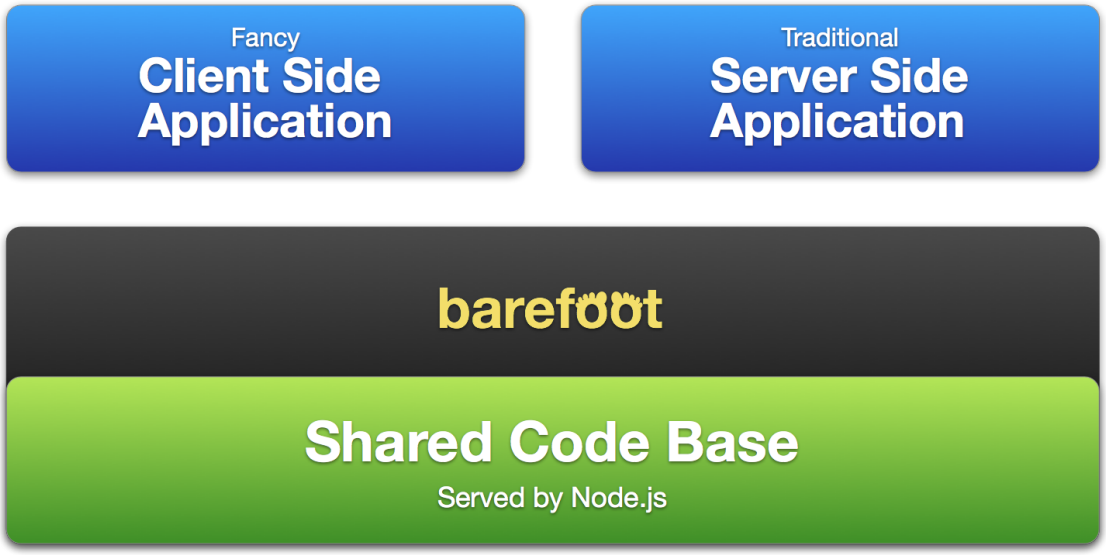


Abbildung 5.1.: Architektur

5.1.1. Shared Codebase mit barefoot

Barefoot [Alaa] ist ein Framework welches während der Bachelorarbeit entwickelt wurde. Dieses Framework ermöglicht Client und Server den gleichen Code für Controller und Templates zu benutzen und setzt auf Backbone.js [Doca] für die Code-Strukturierung auf.

Als Beispiel sei hier der Router genannt:

```
25 module.exports = Router.extend({
26   routes: {
27     '': 'home'
28     , 'community': 'createCommunity'
29     //...
30   }
31
32   /** Function: home
33    * Home page which renders the login button if not authorized.
34    * Otherwise it redirects to community/create or community/:slug/tasks.
35    */
36   , home: function home() {
37     if(!this.isAuthorized()) {
38       this.render(this.createView(HomeView));
39     } else {
40       var community = this.dataStore.get('community');
41       if(!community) {
42         this.navigate('community/create', { trigger: true });
43       } else {
44         this.navigate('community/' + community.get('slug') +
45           '/tasks', { trigger: true });
46       }
47     }
48   }
49
50   /** Function: createCommunity
51    * Create community view
52    */
53   , createCommunity: function createCommunity() {
54     debug('create community');
55     if(!this.redirectIfNotAuthorized()) {
56       this.render(this.createView(CreateCommunityView));
57     }
58   }
59   //...
60 });
```

Quelltext 5.1: Router der Beispielapplikation [AJWn]

Quelltext 5.1 zeigt ein Auszug des Routers aus der Beispielapplikation. Dieser Router wird sowohl vom Client wie auch vom Server direkt verwendet und registriert die gezeigten URLs mit den Funktionen.

Wie man sieht, wird für die “home”-Route, falls der Besucher nicht eingeloggt ist, die “HomeView” dargestellt. Diese ist im Quellcode 5.2, “HomeView der Beispielapplikation [AJWr]” zu sehen.

```
7  module.exports = View.extend({
8    el: '#main'
9    , template: templates.login
10
```

```
11    /** Function: renderView
12     * Renders the home view.
13     */
14    , renderView: function renderView() {
15      this.$el.html(this.template({}));
16    }
17
18    /** Function: afterRender
19     * Sets the document title.
20     */
21    , afterRender: function afterRender(resolve) {
22      var _super = this.constructor.__super__.afterRender.bind(this);
23
24      this.setDocumentTitle(this.translate('Welcome'));
25      _super(resolve);
26    }
27    //...
28  });
```

Quelltext 5.2: HomeView der Beispielapplikation [AJWr]

5.1.2. API-Komponente

Damit sichergestellt werden kann, dass alle Daten valid sind, ist auf Server-Seite eine REST-API [Fie00] erstellt worden. Die Shared Codebase greift auf diese zu, um Daten zu speichern oder zu laden. Dabei unterscheidet "barefoot" intelligent zwischen API-Aufrufen vom Server oder vom Client. Wird vom Client aus aufgerufen, wird ein normaler AJAX-Aufruf gemacht. Wird die API hingegen vom Server aus aufgerufen, werden direkt die entsprechend definierten Callbacks für die gewünschte Route aufgerufen.

Der Quellcode 5.3 zeigt ein Beispiel für einen solchen Callback, während Quelltext 5.4 ein Beispiel für eine definierte API-Route aufzeigt.

```
294 /** Function: getCommunityWithId
295  * Looks up a community with a specific ID.
296  *
297  * Parameters:
298  *   (Function) success - Callback on success. Will pass the community data
299  *                        as first argument.
300  *   (Function) error - Callback in case of an error
301  *   (String) id - The id of the community to look for.
302  */
303 function getCommunityWithId(success, error, id) {
304   debug('get community with id');
305
306   var communityDao = getCommunityDao.call(this)
307
308     /* AnonymousFunction: forwardError
309      * Forwards an error object using the error callback argument
310      */
311     , forwardError = function forwardError(err) {
```

```
312       return error(err);
313     }
314
315     /* AnonymousFunction: afterCommunitySearch
316      * Calls the success or error callback after searching for a community.
317      */
318     , afterCommunitySearch = function afterCommunitySearch(community) {
319       if(!_.isNull(community)) {
320         success(community.dataValues);
321       } else {
322         forwardError(new errors.NotFoundError(
323           'Community with id ' + id + 'does not exist.')
324         );
325       }
326     };
327
328   communityDao.find({ where: { id: id, enabled: true }})
329     .success(afterCommunitySearch)
330     .error(forwardError);
331 }
```

Quelltext 5.3: API-Controller Beispiel [AJWe]

```
31 var prefix = apiPrefix + modulePrefix;
32
33 // GET /api/community/:id
34 api.get(prefix + '/:id(\\d+)', [
35   basicAuthentication
36   , authorizedForCommunity
37   , controller.getCommunityWithId]);
```

Quelltext 5.4: API-Route Beispiel [AJWf]

Die Beispielapplikation ist somit nach einem MVC-Pattern aufgebaut.
Die einzelnen Komponenten haben folgende Aufgabe:

- **M**odel ist das traditionelle Model welches zwischen Server & Client geshared wird
- **V**iew ist eine “Barefoot.View” [Alac] und rendert Templates
- **C**ontroller ist einerseits ein “Route”-Controller, welcher aufgrund von URLs die richtige View aufruft und andererseits ein API-Controller, welcher die REST-API-Logik kapselt

5.2. Domainmodel

Das folgende Domainmodel in Abbildung 5.2 zeigt eine Übersicht über alle in der Problemdomäne enthaltenen Objekte. Zu jedem dieser ist im Anschluss eine genauere Erklärung zu finden.

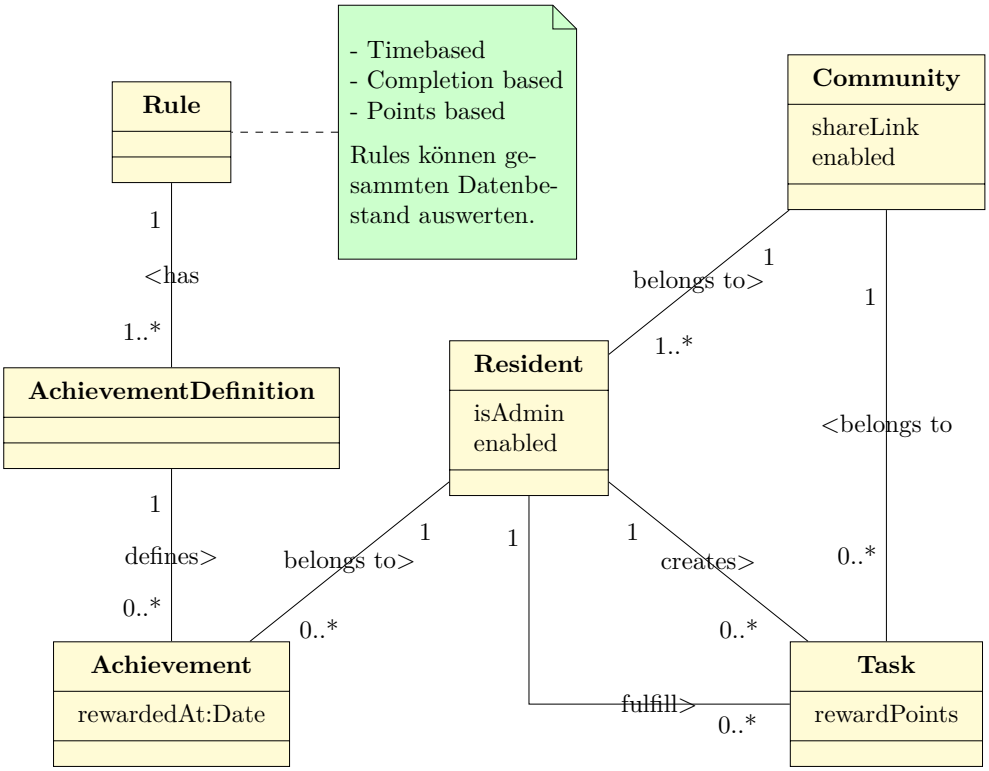


Abbildung 5.2.: Domainmodel

Achievement (Erfolg)

Achievements werden an einen Resident vergeben, sobald diese bestimmte Regeln ausreichend befriedigt haben.

Rule (Regel)

Rules beschreiben, welche Aktionen oder Verhalten notwendig sind, damit ein Resident ein bestimmtes Achievment erhalten kann.

Eine Rule kann verschiedene Ausprägungen haben:

- Zeitbasiert
  Beispiel: Ein Resident ist bereits zwei Monate Teil einer Community.

- Punktebasiert
  Beispiel: Ein Resident hat durch das erledigen von Tasks 50 Punkte gesammelt.
- Spezifische Aktionen
  Beispiel: Ein Resident hat 10 Tasks erledigt.

AchievementDefinition (Erfolgsdefinition)

Die AchievementDefinition verknüpft ein Achievment mit den umzusetzenden Rules.

Community (WG)

Eine Community ist eine Wohngemeinschaft in welcher mehrere Residents wohnen können. Zudem gehören Tasks immer zu einer spezifischen Community.

Resident (Bewohner)

Residents sind Bewohner einer Community und die eigentlichen Benutzer des Systems. Ein Resident kann über die *isAdmin*-Eigenschaft erweiterte Berechtigungen zum Verwalten von Communities sowie deren Tasks und Residents erhalten.

Task (Aufgabe)

Eine Community führt eine Liste von zu erledigenden Aufgaben, den Tasks. Ein Resident einer Community kann Tasks erstellen, bearbeiten, löschen und erledigen. Über das Erledigen von Tasks erhält der entsprechende Resident Punkte, welche ihn auf der Community-internen Rangliste emporsteigen lassen.

5.3. Entity-Relationship Diagramm

Folgendes ER-Diagram repräsentiert die Abbildung des Domainmodels auf Datenbankebene.

Für beide Modelle ist die selbe Terminologie gültig.

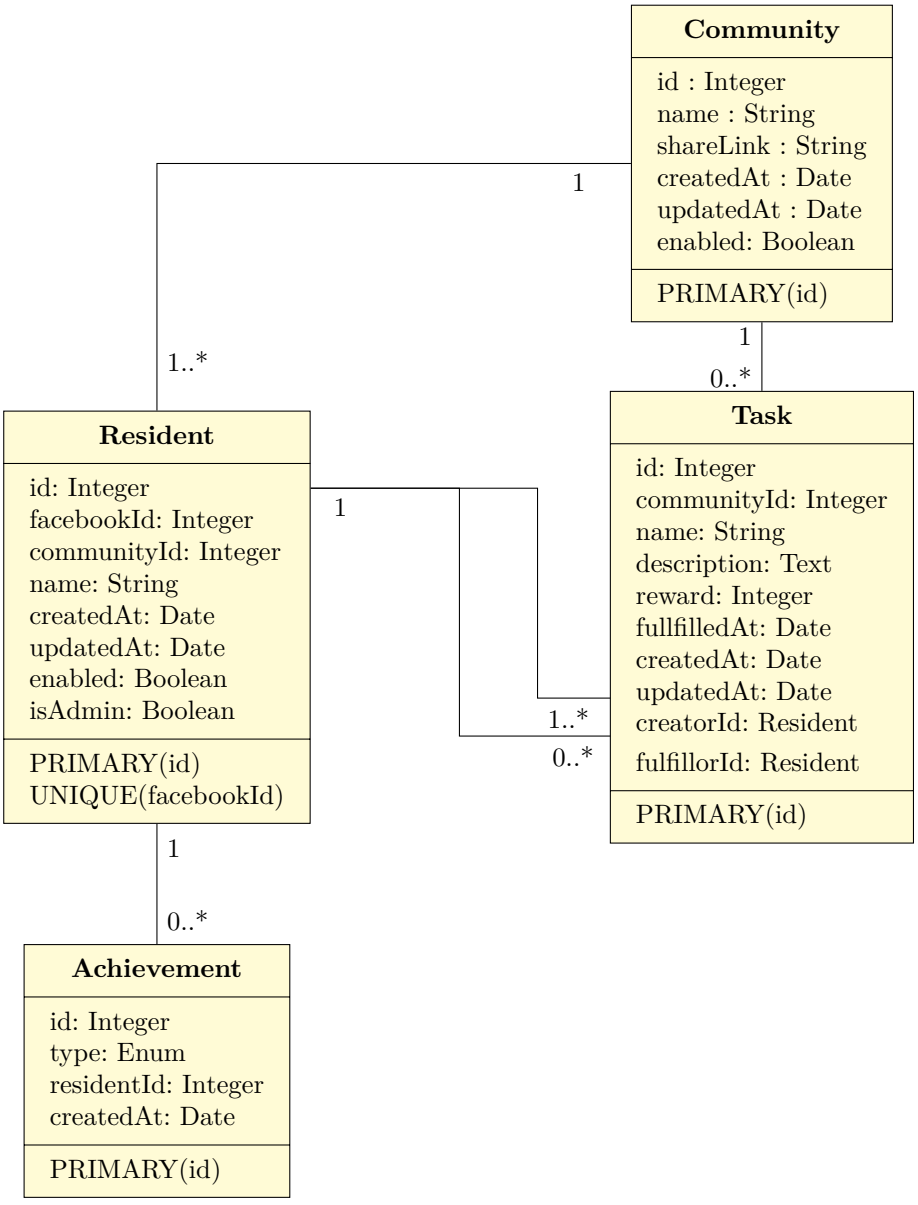


Abbildung 5.3.: Entity-Relationship Diagramm

5.4. Software Layers

In diesem Abschnitt werden die verschiedenen Ebenen der Architektur der Beispielapplikation beschrieben.

Um einen grundsätzlichen Einblick zu erhalten wird jede Ebene zuerst komplett unabhängig von jeglicher konkreter Technologie vorgestellt. In einem weiteren Schritt werden die verwendeten Technologien eingeführt und kurz erläutert.

In der Diagrammen stehen durchgezogene Pfeile jeweils für einen synchronen, gestrichelte Pfeile für einen asynchrone Kommunikationsvorgang.

Technologieneutral

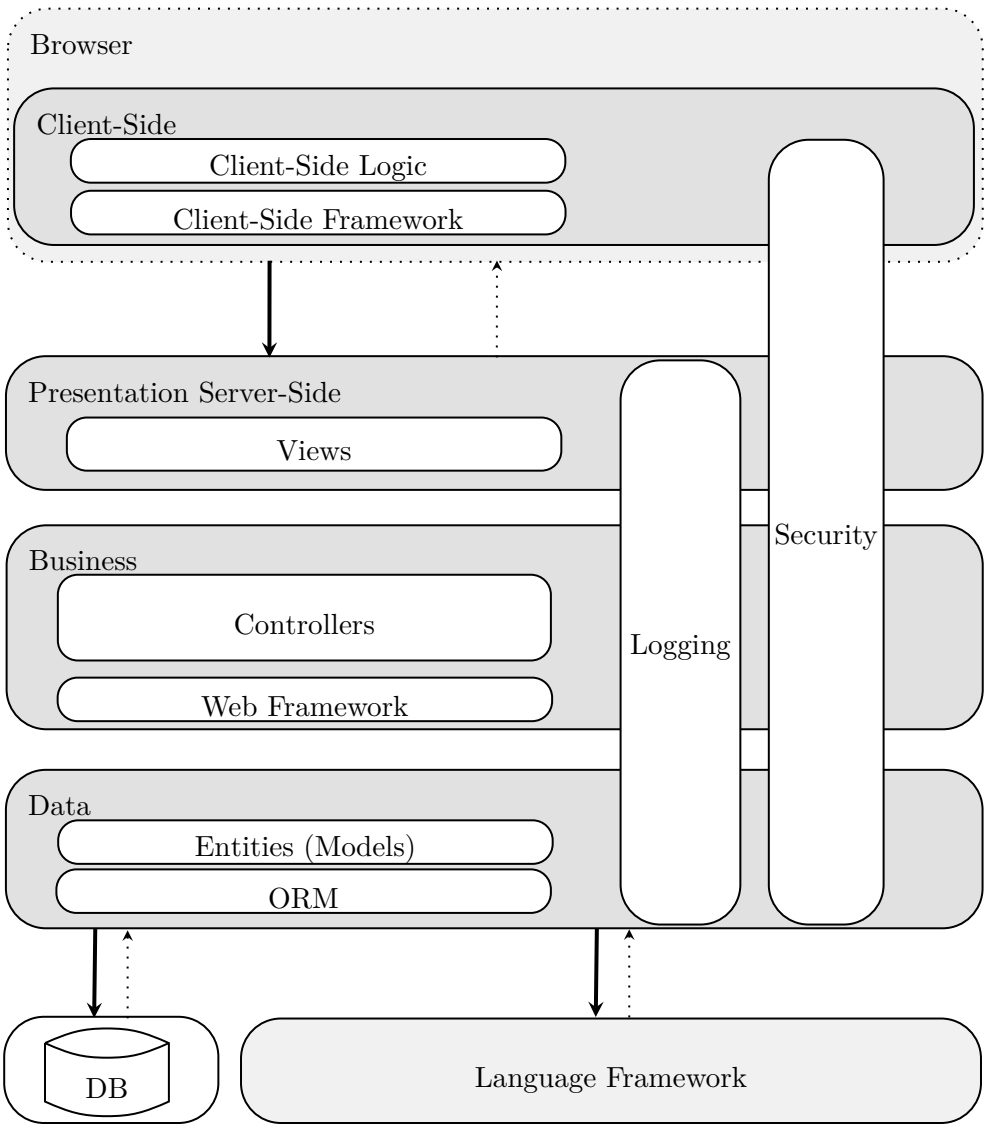


Abbildung 5.4.: Software Layers – Technologieneutral

Client-Side

Diese Schicht wird im Browser selber als JavaScript ausgeführt und synchronisiert sich mit dem Server über eine REST Schnittstelle.

Business

In der Business-Schicht wird die eigentliche Business-Logik definiert und mit der Datenbank über Models kommuniziert.

Language Framework

Das Language Framework repräsentiert die von der jeweiligen konkreten Plattform zur Verfügung gestellten API.

Security

Sowohl Client wie auch Server beinhalten Sicherheitsmassnahmen um zu gewährleisten, dass ein Benutzer immer authentisiert ist und keine gefährlichen Daten übermitteln kann.

Konkrete Implementation

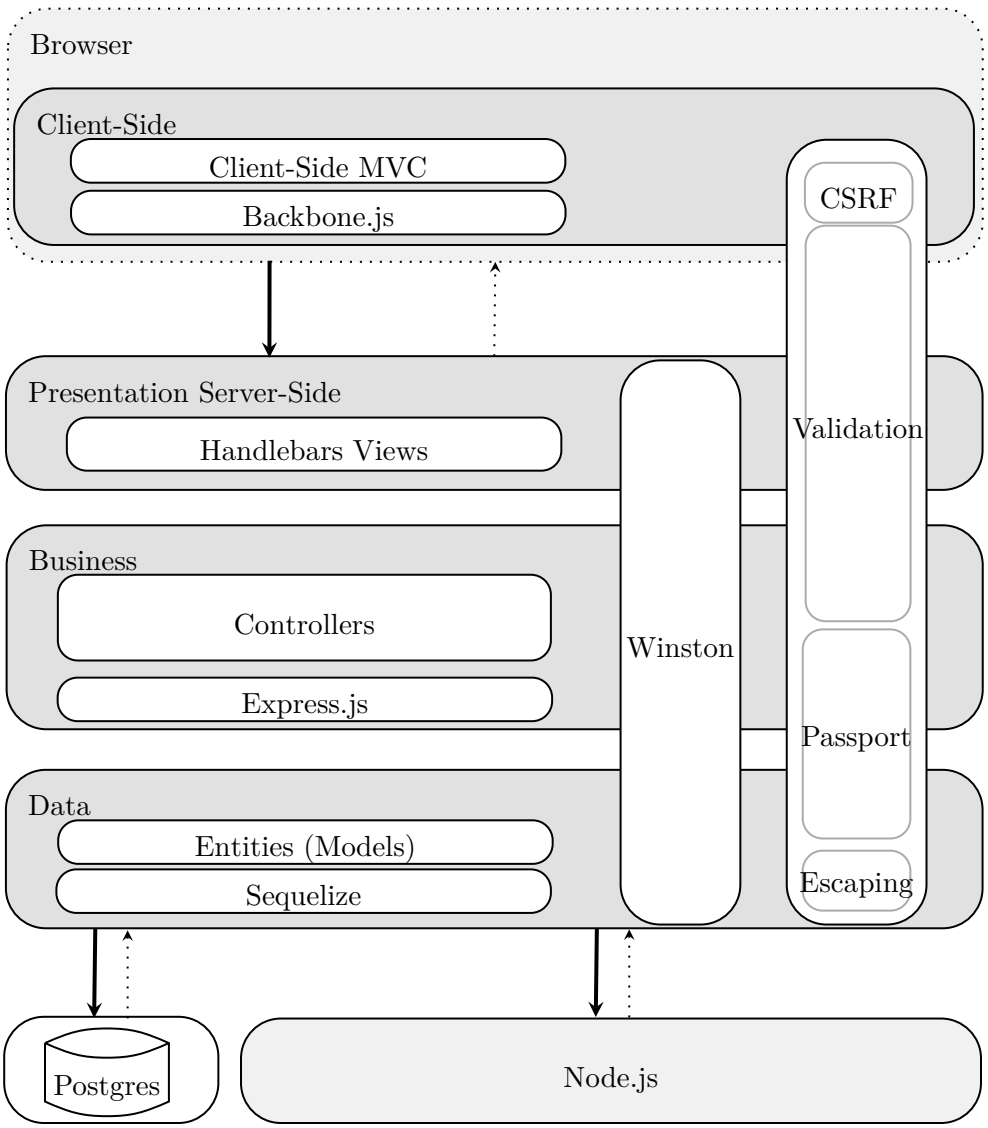


Abbildung 5.5.: Software Layers – Konkrete Implementation

Client-Side MVC

Auf der Client-Seite im Browser wird ebenfalls das MVC-Pattern [Wikl]verwendet. Dies ermöglicht eine entkoppelte Implementation.

Das Client-Side MVC ist auf Backbone.js [Doca] aufgebaut.

Handlebars Views

Die Präsentations-Schicht verwendet Handlebars [Kat]. Die praxiserprobte Library ermöglicht die Erstellung von Templates mit minimaler enthaltener Programmlogik.

Winston

Winston [Fla] ist ein Logging-Framework für Node.js. Einer der grössten Vorteile von Winston ist die umfangreiche Konfigurierbarkeit, wie die unterschiedlichen Logs behandelt werden sollen.

CSRF

Cross-Site Request Forgery-Schicht um CSRF-Attacken zu verhindern. Dies wird von Express.js zur Verfügung gestellt [Expa] und muss lediglich in der entsprechenden Applikation aktiviert werden.

Validation

Formular-Inhalte werden durch “node-validator” [OHa] auf vorher definierte Bedingungen überprüft und bereinigt.

Passport

Passport [Hanb] ist ein Authentisierungs-Framework für Node.js. Es bietet eine Vielzahl von Strategien (Strategy-Pattern [Gam+94]) für das Einloggen über verschiedenste externe Authentisierungs-Provider (z.B. Facebook Login for Web [Faca]) an.

Escaping

Um SQL-Injection [Wiki] Attacken u.Ä. zu unterbinden, bietet das ORM Sequelize [Depb] umfangreiche Funktionalitäten für das Bereinigen von tendenziell gefährlichen SQL-Abfragen an.

5.5. Implementations-Sicht

5.5.1. Rendering des User Interfaces und Event-Behandlung

Durch die Verwendung von *barefoot* [Alaa] kann die komplette Beispielapplikation sowohl eigenständig auf dem Server gerendert werden, als auch als moderne JavaScript Applikation im Browser des Endbenutzers ausgeführt werden. Aus diesem Grund zeigen die folgenden zwei Abschnitte jeweils ein separates Sequenzdiagramm: Eines für das Event-Handling auf dem Client und eines für den Rendering-Vorgang auf dem Server.

Grundlegende Unterschiede gibt es hierbei lediglich beim Zugriff auf den *APIAdapter*. Auf dem Server werden diese direkt lokal behandelt, im Client-Browser wird die entsprechende API-Abfrage über einen HTTP REST Request übertragen.

Server

Beim initialen Aufruf der Beispielapplikation werden die kompletten Inhalte des User Interfaces auf dem Server gem. dem Diagramm 5.6 gerendert. Sollte der Client-Browser zudem JavaScript deaktiviert haben oder nicht unterstützen, so werden auch nachfolgende Requests nach dem gleichen Schema verarbeitet und gerendert.

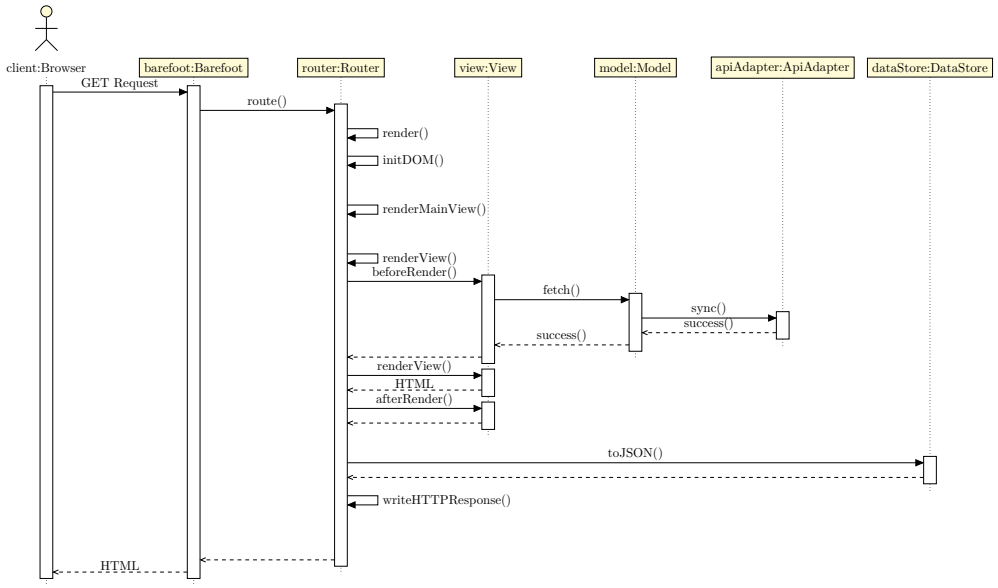


Abbildung 5.6.: Sequenzdiagramm: Rendering und Event-Verarbeitung auf dem Server

Client

Das Sequenzdiagramm 5.7 zeigt den Kontrollfluss nachdem der Benutzer im Browser auf einen Eintrag im Navigationsmenü geklickt hat.

Es gilt zu beachten dass die Aufrufe auf den *APIAdapter* nicht lokal verarbeitet werden sondern direkt auf dem Server.

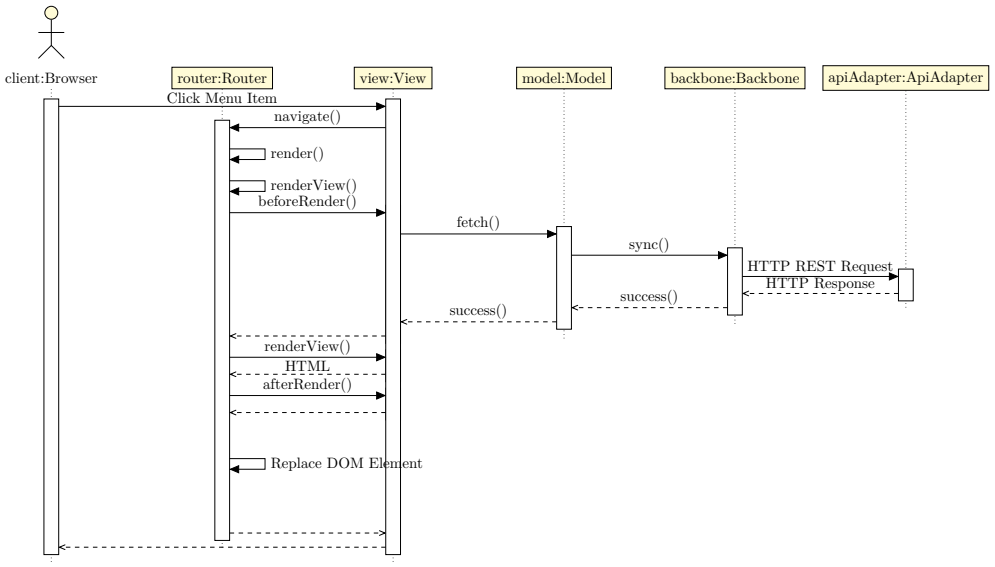


Abbildung 5.7.: Sequenzdiagramm: Rendering und Event-Verarbeitung auf dem Client

5.5.2. APIAdapter

Wie bereits erwähnt kann der *APIAdapter* auf dem Server sowohl mit server-lokalen Requests als auch mit HTTP REST Anfragen umgehen.

Das Diagramm 5.8 verdeutlicht die Abläufe im innern des Adapters für den jeweiligen Anfragemodus.

Hinter dem Element *Controllers* stehen neben dem eigentlichen Controller zusätzlich diverse Sicherheitspolicies und Validatoren, welche eingehende Requests prüfen und ggf. abweisen können, bevor sie zur eigentlichen Geschäftslogik gelangen.

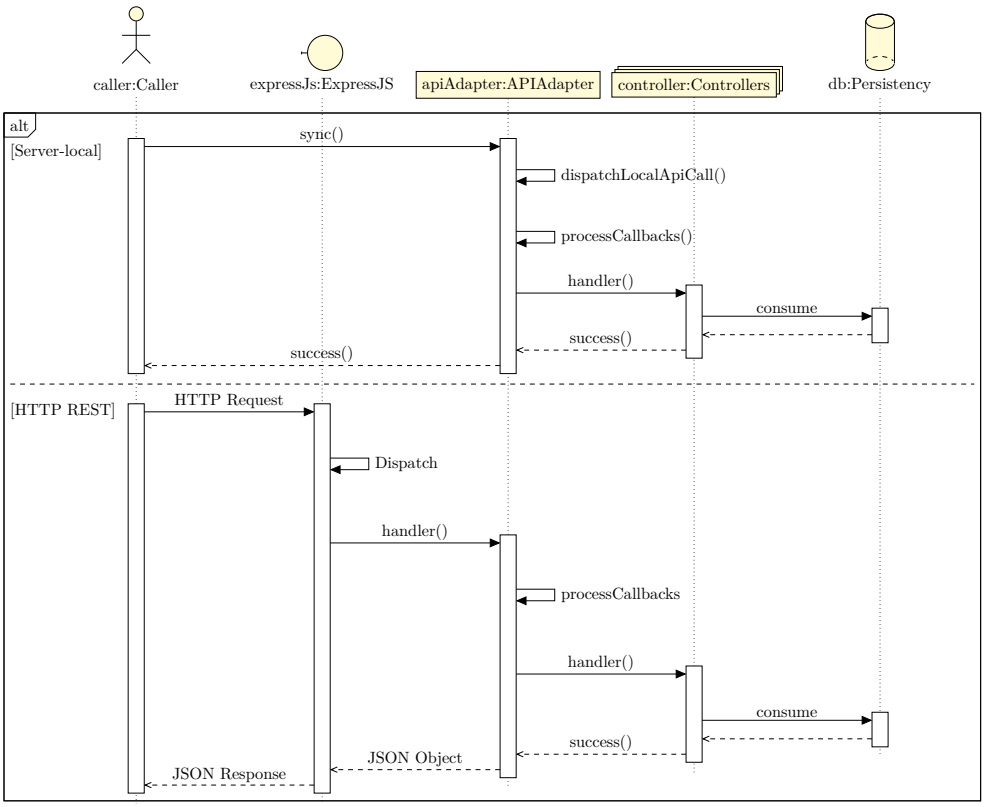


Abbildung 5.8.: Sequenzdiagramm: APIAdapter im server-lokalen sowie HTTP REST Modus

5.5.3. Quellcode Organisation

Auch für Applikationen für Node.JS ist üblich ([Hold], [Sch]), den Quellcode soweit wie möglich in mehr oder weniger komplett eigenständige Komponenten zu unterteilen.

Ein gutes Beispiel hierfür sind die Vielzahl an verfügbaren Modulen welche über den Komponentenmanager NPM [Joya] installierbar sind. Selbst Bibliotheken mit minimalem Umfang werden und sollten als eigene Module gekappselt werden.

Die Beispielapplikation “Roomies” verwendet selber auch mehrere Komponenten aus NPM [Joya] und ist selber auch in Komponenten unterteilt. Dies veranschaulicht die Ordnerstruktur in Abbildung 5.9.

Ordnerstruktur

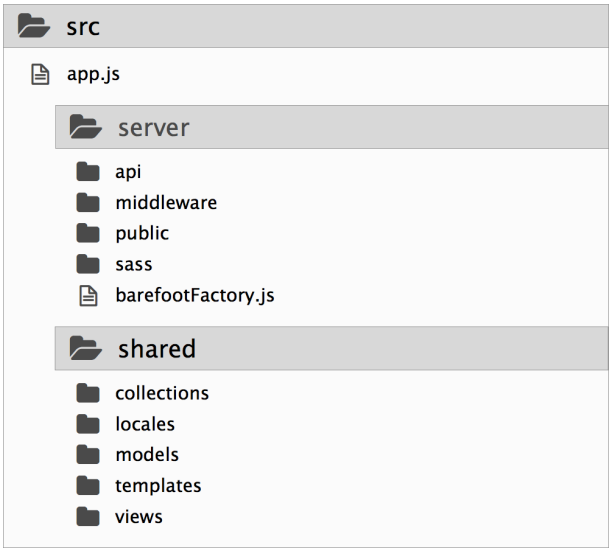


Abbildung 5.9.: Ordnerstruktur des *src*-Unterordners der Beispielapplikation *Roomies*

Jede Komponente mit JavaScript-Code kann mittels einem "require()"-Befehl eingebunden werden. Zu diesem Zweck wird jeweils nur der Ordnername angegeben. Der Befehl wird dann automatisch die "index.js"-Datei laden.

```
1 // Requires actually src/server/api/community/index.js:
2 // Found in src/server/api/index.js
3 var setupCommunityApi = require('./community');
```

Quelltext 5.5: Einbindung der Community-Komponente

Im folgenden werden die beiden wichtigsten Startpunkte der Applikation erklärt.

barefootFactory.js

Die "barefoot"-Factory [AJWg] ist verantwortlich für das Setup der Server-Seite der Applikation. Unter anderem werden folgende wichtige Konfigurationen vorgenommen:

- JavaScript Dateien, die am Client geschickt werden sollen
- API-Adapter eingerichtet
- Express.js [Expb] Middleware geladen
- Express.js [Expb] HTTP Server gestartet

- Server Request Context (u.a. das Model für den eingeloggten Benutzer) eingerichtet

Diese Informationen werden “barefoot” [Alaa] mitgegeben, damit das Framework weiss, was zu starten ist.

app.js

Die Datei “app.js” [AJWd] ist der Startpunkt der Applikation. Darin wird u.a. die “barefootFactory” instanziert und der “DataStore” initialisiert. Der “DataStore” ist verantwortlich, dass sowohl Client und Server die gleichen Models zur Verfügung haben und darüber kommunizieren können.

Kapitel 6 Projektplanung

6.1. Iterationsplanung

Die Iterationsplanung orientiert sich grob am RUP und ist unterteilt in eine *Inception*-, *Elaboration*-, fünf *Construction*- sowie jeweils eine *Transition*- und *Abschlussphase*.

Iteration	***Dauer***	***Beschreibung***
Inception	3 Wochen	Projektsetup, genauere Definition der Aufgabe, Vorbereitungen & Planungen
Elaboration	3 Wochen	Anforderungsanalysen, Entwicklung eines Architekturprototypen und genauere technische Evaluationen. Guidelines für Quellcode und Testing wurden erstellt.
Construction 1	2 Wochen	Umsetzung des Applikationsfundaments, UI Design, Umsetzung erster als *Hoch* priorisierter Use Cases. Qualitätssicherung in Form von Reviews & Unit Testing.
Construction 2	2 Wochen	Fertigstellung der restlichen als *Hoch* priorisierten Use Cases. Qualitätssicherung in Form von Reviews & Unit Testing.
Construction 3	2 Wochen	Umsetzung des Gamification-Teils der Applikation. Qualitätssicherung in Form von Reviews & Unit Testing.
Construction 4	2 Wochen	Implementation der restlichen als *Mittel* priorisierten Use Cases. Qualitätssicherung in Form von Reviews & Unit Testing.
Construction 5	2 Wochen	Umsetzung aller restlichen als *Tief* priorisierten Use Cases sowie erstes Bugfixing gem. geführter Issueliste. Qualitätssicherung in Form von Reviews & Unit Testing.
Transition	1 Wochen	Abschliessende Bugfixing-Arbeiten. Code-Freeze und Erstellung von Deployment-Pakete. SAD ist in einer finalen Version verfügbar.
Abschluss	2 Wochen	Finalisierung der Dokumentation sowie Erstellung der HSR Artefakte *A100* sowie *A101*.

Tabelle 6.1.: Projektierationsbeschreibung

Im Projektverwaltungstool (siehe Tools) ist ergänzend eine detailierte Arbeitspaketplanung mit aktuellem Arbeitsstatus verfügbar.

Jede Iteration wird jeweils von einem Meilenstein abgeschlossen, was in folgendem Gantt-Diagramm ersichtlich ist:

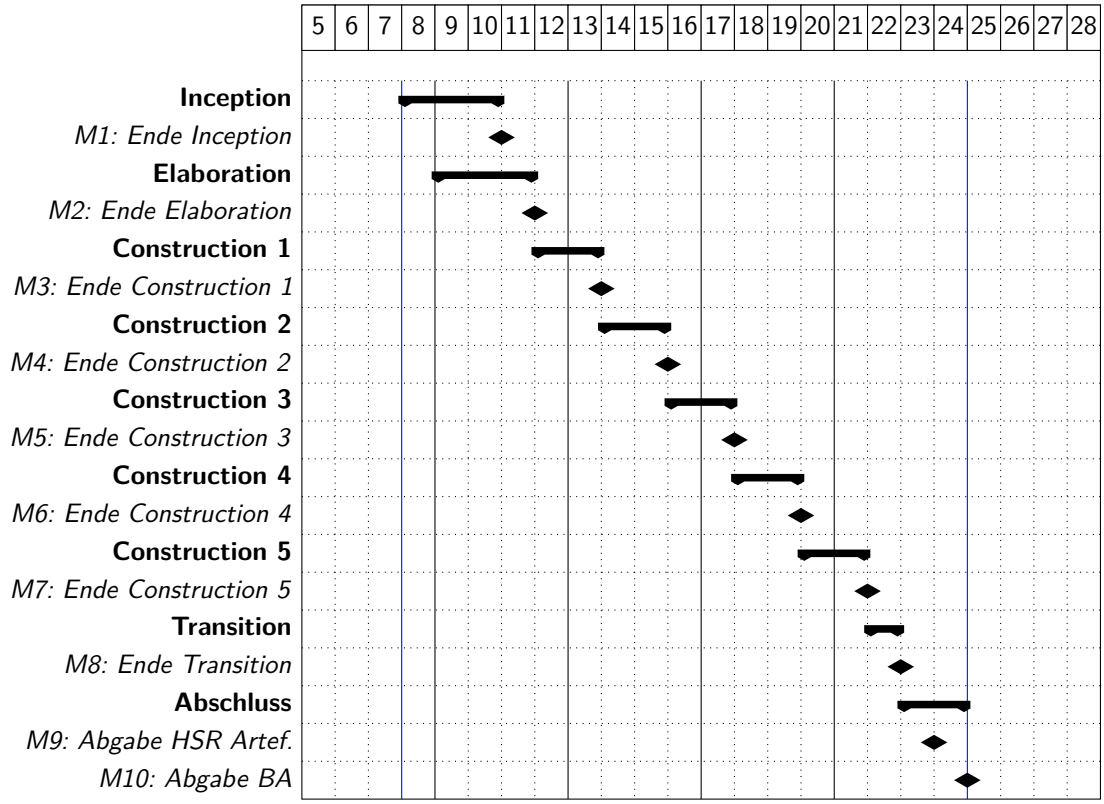


Abbildung 6.1.: Iterationsübersicht mit Meilensteinen, Kalenderwochen Februar bis Juli 2013

Die beiden Phasen *Inception* und *Elaboration* sind überlappend geplant, da zu Beginn des Projekts die Aufgabenstellung noch nicht abschliessend definiert war. Die Überlappung ermöglicht das vorbereitende Erledigen von *Elaboration* Artefakten.

6.2. Meilensteine

ID	*Meilenstein*	*Termin*	*Beschreibung*
M1	Ende Inception	10.03.2013	Die Aufgabenstellung wurde gem. Auftrag klar definiert und die Projektinfrastruktur ist aufgesetzt. Eine initiale Projektplanung besteht.
M2	Ende Elaboration	17.03.2013	Konkrete Technologie und Guidelines sind definiert. Anforderungsdokumente sind erstellt und abgenommen. Initiale SAD und Architekturprototyp bereit.
M3	Ende Construction 1	31.03.2013	Das Fundament der Applikation wurde implementiert. Weiter wurden die ersten Use Cases der Priorität *Hoch* umgesetzt.
M4	Ende Construction 2	14.04.2013	Alle Use Cases der Priorität *Hoch* sind umgesetzt.
M5	Ende Construction 3	28.04.2013	
M6	Ende Construction 4	12.05.2013	
M7	Ende Construction 5	26.05.2013	SAD fertiggestellt.
M8	Ende Transition	02.06.2013	Deployment-Pakete und zugehörige Anleitungen sind bereit. Bugfixing abgeschlossen resp. ausstehende Bugs dokumentiert.
M9	Abgabe HSR Artefakte	07.06.2013	Das A0-Poster sowie die Kurzfassung der Bachelorarbeit sind dem Betreuer zugestellt.
M10	Abgabe Bachelorarbeit	14.06.2013	Alle abzugebenden Artefakte sind dem Betreuer zugestellt worden.

Tabelle 6.2.: Meilensteine

6.3. Artefakte

Dieser Abschnitt beschreibt alle Arbeitsprodukte (Artefakte), welche zwingend erstellt und abgegeben werden müssen.

Falls nicht anders vermerkt sind alle Artefakte Teil der Dokumentation.

ID	*Meilenstein*	*Artefakt*	*Beschreibung*
A20	M2	Projektplanung	Projektablauf, Infrastrukturbeschreibung & Iterationsplanung
A21	M2	Analyse der Aufgabenstellung	Produktentwicklung, Technologieevaluation & Analyse Architekturprinzipien
A22	M2	Guidelines	Quellcode- und Testing-Guidelines
A23	M2	Anforderungsanalyse	Funktionale & nichtfunktionale Anforderungen, Use Cases
A24	M2	Domainmodel	Analyse der Problemdomäne
A25	M2 & M8	SAD	Beschreibung der Architektur für die Beispielapplikation.
A26	M2	Architekturprototyp	Exemplarische Implementierung der angestrebten Technologie/Architektur *Typ: Quellcode/Applikation*
A80	M8	Quellcode Paket	Quellcode der Beispielapplikation zum eigenen, spezifischen Deployment. Bereits zur Weiterentwicklung. *Typ: Quellcode*
A81	M8	Vagrant Paket	VM-Image mit lauffähiger Version der Beispielapplikation. *Typ: Vagrant Image*
A82	M8	Heroku Paket	Beispielapplikation ist so vorbereitet, dass ein Deployment auf Heroku problemlos möglich ist. *Typ: Quellcode*
A83	M8	Installationsanleitung	Anleitung wie die verschiedenen Deployment-Pakete (Artefakte *A80-82*) eingesetzt/installiert werden können.
A100	M10	A0-Poster	Gem. HSR Vorgaben zu erstellendes Poster mit Übersucht zu dieser Bachelorarbeit.
A101	M10	Kurzfassung	Gem. HSR Vorgaben zu erstellende Kurzfassung dieser Bachelorarbeit.
A102	M10	Dokumentation	Alle bisherigen Dokumentationsartefakte zusammengefasst in einem Bericht. Wo nötig, sind entsprechende Kapitel dem Projektablauf entsprechend nachgeführt (bspw. *A25 SAD* etc.)

Tabelle 6.3.: Abzugebende Artefakte

6.4. Meetings

6.4.1. Regelmässiges Statusmeeting

Während der gesamten Projektdauer findet jeweils am Mittwoch um 10 Uhr ein wöchentliches Statusmeeting statt. Die Sitzung wird abwechslungsweise jeweils von einer Person aus dem Projektteam geführt sowie von einer anderen protokolliert.

Das Projektteam stellt die Agenda der aktuellen Sitzung bis spätestens am vorangehenden Dienstag Abend bereit.

6.5. Tools

Ergänzend zu diesem Abschnitt ist der Anhang D “Projektrelevante URL’s” zu erwähnen. Er enthält alle wichtigen Internetadressen zu den spezifischen Tools und Code Repositories.

6.5.1. Projektverwaltung

Für die komplette Projektplanung, die Zeitrapportierung sowie das Issue-Management wird Redmine eingesetzt.

Der aktuelle Stand der Arbeiten am Projekt kann hier jederzeit eingesehen werden und wird vom Projektteam aktiv aktualisiert.

6.5.2. Entwicklungsumgebung

Zur Entwicklung von Quellcode-Artefakten steht eine mit Vagrant [Has] paketierte Virtual Machine bereit. Sie enthält alle notwendigen Abhängigkeiten und Einstellungen:

- node.js 0.10.0
- PostgreSQL 9.1
- Ruby 2.0.0 (installiert via rvm)
- ZSH (inkl. oh-my-zsh)

Das Code Repository enthält ein *Vagrantfile* welches durch den Befehl *vagrant up* in der Kommandozeile automatisch das Image der vorbereiteten VM lokal verfügbar macht und startet.

6.5.3. Git Repositories

Sowohl Quellcodeartefakte als auch die in LaTeX formulierte Thesis (dieses Dokument) wird in auf GitHub abgelegten Git Repositories versioniert bzw. zentral gespeichert.

6.5.4. Continious Integration

Für das Projekt wird Travis CI zur Continious Integration Lösung verwendet. Nähere Informationen sind im Kaptiel 7 “Qualitätsmanagement” unter “Continuous Integration” zu finden.

Kapitel 7 Qualitätsmanagement

7.1. Baselining

Beim erreichen der Meilensteine wird auf dem Quelltext- als auch Dokumentations-Git-Repository ein Tag mit dem entsprechenden Meilenstein-ID erstellt. Auf diese Weise bleibt der Projektverlauf klar verfolgbar und die abzugebenden Artefakte können dem Review des Betreuers resp. Gegenlesers unterzogen werden.

Nach dem Erstellen einer Baseline informiert das Projektteam alle Beteiligten via E-Mail.

7.2. Testing

Die Beispielapplikation wird nach der TDD-Methodik entwickelt.

7.2.1. Unit Testing

Mocha [Hole] wird als Framework für das Erstellen von Unit Tests verwendet. Es ermöglicht das einfache Entwickeln von asynchronen Unit Tests und bietet eine vielzahl verschiedener Testreports.

Um die Unit Tests mit einer möglichst angenehmen Syntax verfassen zu können wird Chai.JS [Lc] eingesetzt. Chai bittet drei verschiedene Assert-Varianten an. Innerhalb dieses Projekts wird wenn möglich die “Should”-Formulierung verwendet:

```
1  var chai = require('chai')
2    ,config = require('../../config_test')
3    ,db = require('../../lib/db');
4
5    chai.should();
6
7    var sequelize = db(config);
8    describe('Community DAO', function(){
9      it('should not have any communities', function(done){
10       (function() {
11         sequelize.daoFactoryManager.getDAO('Community').all().success(
               function(communities) {
12           communities.should.have.length(0);
13           done();
14         }).error(function(error) {
```

```
15            throw(error);
16          })
17        }).should.not.throw();
18      });
19    });
```

Quelltext 7.1: Beispiel eines Unit Tests mit Mocha und Chai.js

7.2.2. Test Coverage Analysis

Zur Überprüfung der Test Coverage wird JSCover [tnt] verwendet. Es ist komplett in den CI Prozess integriert und ermöglicht das einfache Überprüfen der aktuellsten Metriken.

7.3. Continuous Integration

Mit Travis CI [CI] werden fortlaufende Builds der Beispielapplikation erstellt. Diese stellen deren Funktionalität entsprechend der vorhandenen Unit Tests sicher und führen die Test Coverage Analyse durch. Weiter wird jeweils eine aktuelle Version der Quellcode-Dokumentation mittels NaturalDocs erstellt und bereitgestellt.

Als Besonderheit wird auch diese Dokumentation fortlaufend mit Travis CI generiert. Dazu wird LaTeX Quelltext mittels TexLive in ein PDF umgewandelt und auf dem Internet zur Verfügung gestellt.

7.4. Coding Style Guideline

Die Coding Style Guideline ist im Anhang I “Coding Style Guideline” einsehbar.

Zur Unterstützung während des Entwicklungsprozesses wird JSHint [Coma] zur statischen Überprüfung des Quellcodes verwendet. JSHint ist in den Continuous Integration Prozess integriert.

7.5. Code Reviews

Code Reviews sind während der Projektplanungsphase bereits fix in die fünf verschiedenen Construction-Iterationen eingeplant.

Es werden jeweils gezielt Artefakte ausgewählt und überprüft. Dabei soll sowohl ein funktioneller Review stattfinden, als auch die Einhalteung der aufgestellten Code Guideline sichergestellt werden.

Anhang A **Abbildungen, Tabellen & Quellcodes**

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang B Literatur

[AGa] 20 Minuten AG. *20 Minuten Online - community*. URL: http://www.20min.ch/community/login/ (besucht am 04. 06. 2013).

[AGb] Mila AG. *Mila - gemeinsam mehr erledigen*. URL: https://www.mila.com/ (besucht am 06. 06. 2013).

[Aira] Airbnb. *Airbnb JavaScript Style Guide*. URL: https://github.com/airbnb/javascript (besucht am 10. 03. 2013).

[Airb] Airbnb. *Airbnb JavaScript Style Guide (Updated)*. URL: https://github.com/mekanics/javascript-style-guide (besucht am 10. 03. 2013).

[AJWa] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/config.example.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/config.example.js (besucht am 04. 06. 2013).

[AJWb] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA-Dokumentation/Makefile at master*. URL: https://github.com/hsr-ba-ajw-2013/BA-Dokumentation/blob/master/Makefile (besucht am 30. 03. 2013).

[AJWc] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/Makefile at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/Makefile (besucht am 30. 03. 2013).

[AJWd] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/app.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/app.js (besucht am 04. 06. 2013).

[AJWe] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/server/api/-community/controller.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/server/api/community/index.js (besucht am 05. 06. 2013).

[AJWf] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/server/api/-community/index.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/server/api/community/index.js (besucht am 05. 06. 2013).

[AJWg] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/server/barefootFactory.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/server/barefootFactory.js (besucht am 04. 06. 2013).

[AJWh] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/server/layout.html at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/server/layout.html (besucht am 05. 06. 2013).

[AJWi] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/server/layout.html at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/server/layout.html (besucht am 05. 06. 2013).

[AJWj] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/server/middleware/http.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/server/middleware/http.js (besucht am 06. 06. 2013).

[AJWk] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/server/middleware/http.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/server/middleware/http.js (besucht am 30. 03. 2013).

[AJWl] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/server/sass/_buttons.scss at master · hsr-ba-ajw-2013/BA · GitHub*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/server/sass/_buttons.scss#L4 (besucht am 05. 06. 2013).

[AJWm] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/shared/models/community.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/shared/models/community.js (besucht am 05. 06. 2013).

[AJWn] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/shared/router.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/shared/router.js (besucht am 05. 06. 2013).

[AJWo] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/shared/templates/menu.hbs at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/shared/templates/menu.hbs (besucht am 06. 06. 2013).

[AJWp] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/shared/templates/task/listItem.hbs at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/shared/templates/task/listItem.hbs (besucht am 06. 06. 2013).

[AJWq] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/shared/views/community/join.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/shared/views/community/join.js (besucht am 06. 06. 2013).

[AJWr] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/shared/views/home.js at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/src/shared/views/home.js (besucht am 05. 06. 2013).

[AJWs] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/.travis.yml at master*. URL: https://github.com/hsr-ba-ajw-2013/BA/blob/master/.travis.yml (besucht am 30. 03. 2013).

[Alaa] Manuel Alabor. *Barefoot makes code sharing between browser and server reality. Write your application once and run it on both ends of the wire.* URL: https://github.com/swissmanu/barefoot (besucht am 04. 06. 2013).

[Alab] Manuel Alabor. *Barefoot.Start.Client Source.* URL: https://github.com/swissmanu/barefoot/blob/master/lib/client/start.js (besucht am 05. 06. 2013).

[Alac] Manuel Alabor. *Barefoot.View Source Documentation.* URL: http://swissmanu.github.io/barefoot/docs/files/lib/view-js.html (besucht am 04. 06. 2013).

[Ash] Jeremy Ashkenas. *CoffeeScript.* URL: http://coffeescript.org/ (besucht am 13. 03. 2013).

[Ate] Faruk Ates. *Modernizr: the feature detection library for HTML5/CSS3.* URL: http://modernizr.com (besucht am 05. 06. 2013).

[Bala] Balderash. *Diskussion über Beziehungen zwischen Models in Sails.js.* URL: https://github.com/balderdashy/sails/issues/124 (besucht am 16. 03. 2013).

[Balb] Balderash. *Sails | The future of API development.* URL: http://sails.org (besucht am 01. 03. 2013).

[Balc] Balderash. *Waterline ORM for Sails.js.* URL: https://github.com/balderdashy/sails/tree/master/lib/waterline (besucht am 15. 03. 2013).

[Ber] Sir Tim Berners-Lee. *Cool URIs don't change.* URL: http://www.w3.org/Provider/Style/URI.html (besucht am 15. 04. 2013).

[BIA] BIAN. *Service Landscape & Metamodel.* URL: http://bian.org/assets/bian-standards/bian-service-landscape-1-5/ (besucht am 05. 06. 2013).

[Cas] Tim Caswell. *Node Version Manager – Simple bash script to manage multiple active node.js Versions.* URL: https://github.com/creationix/nvm (besucht am 30. 03. 2013).

[Che] Ben Cherry. *JavaScript Module Pattern.* URL: http://www.adequatelygood.com/JavaScript-Module-Pattern-In-Depth.html (besucht am 14. 04. 2013).

[CI] Travis CI. *Travis CI - Free Hosted Continuous Integration Platform for the Open Source Community.* URL: https://travis-ci.org/ (besucht am 30. 03. 2013).

[Cod] CodeParty. *Derby.* URL: http://derbyjs.com/ (besucht am 16. 03. 2013).

[Coma] JSHint Community. *JSHint, a JavaScript Code Quality Tool.* URL: http://www.jshint.com/ (besucht am 26. 03. 2013).

[Comb] Stack Overflow Community. *http - PUT vs POST in REST.* URL: http://stackoverflow.com/questions/630453/put-vs-post-in-rest (besucht am 05. 06. 2013).

[Comc] Stack Overflow Community. *web - Is unobtrusive JavaScript outdated? - Stack Overflow.* URL: http://stackoverflow.com/questions/10428882/is-unobtrusive-javascript-outdated (besucht am 06. 06. 2013).

[Cro08] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008. ISBN: 0596517742.

[CWE] Hampton Catlin, Nathan Weizenbaum und Chris Eppstein. *SASS - Syntactically Awesome Stylesheets*. URL: http://sass-lang.com/ (besucht am 13. 03. 2013).

[Depa] Sascha Depold. *Roadmap für Sequelize.js*. URL: https://github.com/sequelize/sequelize#roadmap (besucht am 17. 03. 2013).

[Depb] Sascha Depold. *Sequelize – A multi-dialect Object-Relational-Mapper for Node.JS*. URL: http://www.sequelizejs.com (besucht am 16. 03. 2013).

[Det+] Sebastian Deterding u. a. *Gamification: Toward a Definition*. URL: http://hci.usask.ca/uploads/219-02-Deterding,-Khaled,-Nacke,-Dixon.pdf (besucht am 11. 03. 2013).

[Deva] Alexis Deveria. *Can I use Date/time input types*. URL: http://caniuse.com/input-datetime (besucht am 05. 06. 2013).

[Devb] Alexis Deveria. *Can I use... Support tables for HTML5, CSS3, etc*. URL: http://caniuse.com/#feat=websockets (besucht am 16. 03. 2013).

[Dic] Urban Dictionary. *Urban Dictionary: roomie*. URL: http://roomie.urbanup.com/1433981 (besucht am 11. 03. 2013).

[Dig] Digitec. *Digitec Online Shop*. URL: https://www.digitec.ch (besucht am 05. 06. 2013).

[Doca] DocumentCloud. *Backbone.js*. URL: http://backbonejs.org/ (besucht am 08. 04. 2013).

[Docb] DocumentCloud. *Backbonejs - Backbone.Model#validate*. URL: http://backbonejs.org/#Model-validate (besucht am 05. 06. 2013).

[Docc] DocumentCloud. *Backbonejs - Backbone#sync*. URL: http://backbonejs.org/#Sync (besucht am 05. 06. 2013).

[Docd] DocumentCloud. *Backbone.js - History*. URL: http://backbonejs.org/#History-start (besucht am 05. 06. 2013).

[Expa] Express.js. *CSRF Middleware for Express.js*. URL: http://expressjs.com/api.html#csrf (besucht am 10. 04. 2013).

[Expb] Express.js. *Express - node.js web application framework*. URL: http://expressjs.com/ (besucht am 27. 02. 2013).

[Faca] Facebook. *Facebook Login - Facebook-Entwickler*. URL: https://developers.facebook.com/docs/facebook-login/ (besucht am 04. 06. 2013).

[Facb] Facebook. *Launching the Improved Auth Dialog- Facebook Developers*. URL: https://developers.facebook.com/blog/post/633/ (besucht am 04. 06. 2013).

[Facc] Facebook. *Pictures - Facebook-Entwickler*. URL: https://developers.facebook.com/docs/reference/api/using-pictures/ (besucht am 04. 06. 2013).

[Fie00] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures. Representational State Transfer“. AAI9980887. Doctoral dissertation. University of California, Irvine, 2000. Kap. 5, S. 76–106. ISBN: 0-599-87118-0. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[Fla] Flatiron. *Winston - a multi-transport async logging library for node.js*. URL: https://github.com/flatiron/winston (besucht am 08. 04. 2013).

[Fora] ForbesLindesay. *CommonJS Modules*. URL: http://www.commonjs.org/specs/modules/1.0/ (besucht am 14. 04. 2013).

[Forb] ForbesLindesay. *SQL-ForbesLindesay/browserify-middleware*. URL: https://github.com/ForbesLindesay/browserify-middleware (besucht am 14. 04. 2013).

[Foua] Free Software Foundation. *GNU Make*. URL: http://www.gnu.org/software/make/ (besucht am 30. 03. 2013).

[Foub] OpenID Foundation. *OpenID Foundation website*. URL: http://openid.net/ (besucht am 07. 04. 2013).

[Fouc] The jQuery Foundation. *jQuery*. URL: http://jquery.com/ (besucht am 06. 06. 2013).

[Frea] Inc. Free Software Foundation. *GNU Make*. URL: http://www.gnu.org/software/make/ (besucht am 30. 03. 2013).

[Freb] Inc. Free Software Foundation. *GNU Wget*. URL: http://www.gnu.org/software/wget/ (besucht am 13. 03. 2013).

[Gam+94] Erich Gamma u. a. „Strategy Pattern“. In: *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. Aufl. Addison-Wesley, 1994. Kap. 2. ISBN: 0-201-63361-2.

[Ged] Geddy. *Geddy | The original MVC Web framework for Node - a simple, structured way to create full stack javascript applications*. URL: http://geddyjs.org/ (besucht am 27. 02. 2013).

[Gei] Felix Geisendörfer. *Felix's Node.js Guide*. URL: http://nodeguide.com/ (besucht am 14. 04. 2013).

[Git] Inc. GitHub. *GitHub*. URL: https://github.com/ (besucht am 06. 06. 2013).

[Gooa] Google. *Google Drive*. URL: https://drive.google.com (besucht am 06. 06. 2013).

[Goob] Google. *Making AJAX Applications Crawlable*. URL: https://developers.google.com/webmasters/ajax-crawling/ (besucht am 06. 06. 2013).

[Gooc] Google. *V8 JavaScript Engine*. URL: https://code.google.com/p/v8/ (besucht am 16. 03. 2013).

[Good] Google. *WebRTC*. URL: http://www.webrtc.org/ (besucht am 06. 06. 2013).

[Gro] PostgreSQL Global Development Group. *PostgreSQL: The world's most advanced open source database*. URL: http://www.postgresql.org/ (besucht am 30. 03. 2013).

[Hana] Jared Hanson. *jaredhanson/passport-facebook*. URL: https://github.com/jaredhanson/passport-facebook (besucht am 04. 06. 2013).

[Hanb] Jared Hanson. *Passport - Simple, unobtrusive authentication for Node.js*. URL: http://passportjs.org/ (besucht am 08. 04. 2013).

[Hanc] David Heinemeier Hansson. *Ruby on Rails*. URL: http://rubyonrails.org/ (besucht am 16. 03. 2013).

[Has] HashiCorp. *Vagrant*. URL: http://www.vagrantup.com/ (besucht am 06. 03. 2013).

[Hax] Haxx. *curl and libcurl*. URL: http://curl.haxx.se/ (besucht am 13. 03. 2013).

[Hola] TJ Holowaychuck. *Express Examples*. URL: https://github.com/visionmedia/express/tree/master/examples (besucht am 14. 04. 2013).

[Holb] TJ Holowaychuck. *Express – Guide*. URL: http://expressjs.com/guide.html (besucht am 14. 04. 2013).

[Holc] TJ Holowaychuck. *Mastering Node - Open Source Nodejs eBook*. URL: http://visionmedia.github.com/masteringnode/ (besucht am 14. 04. 2013).

[Hold] TJ Holowaychuk. *Components*. URL: http://tjholowaychuk.com/post/27984551477/components (besucht am 20. 03. 2013).

[Hole] TJ Holowaychuk. *Mocha*. URL: http://visionmedia.github.com/mocha/ (besucht am 17. 03. 2013).

[Holf] TJ Holowaychuk. *TJ Holowaychuck*. URL: http://tjholowaychuk.com/ (besucht am 14. 04. 2013).

[IBM] IBM. *IBM Rational Unified Process (RUP)*. URL: http://www-01.ibm.com/software/awdtools/rup/.

[Inc] Apple Inc. *Safari Web Content Guide: Configuring Web Applications*. URL: http://developer.apple.com/library/ios/#documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html (besucht am 05. 06. 2013).

[Irv+] R. Fielding UC Irvine u. a. *Hypertext Transfer Protocol – HTTP/1.1*. URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5 (besucht am 13. 03. 2013).

[Joya] Joyent. *Node Package Manager*. URL: https://npmjs.org/ (besucht am 21. 03. 2013).

[Joyb] Inc. Joyent. *node.js*. URL: http://nodejs.org/ (besucht am 16. 03. 2013).

[jrb] jrburke. *RequireJS*. URL: http://requirejs.org/ (besucht am 14. 04. 2013).

[Kat] Yehuda Katz. *Handlebars.js: Minimal Templating on Steroids*. URL: http://handlebarsjs.com/ (besucht am 08. 04. 2013).

[Lc] Jake Luer und individual contributors. *Chai*. URL: http://chaijs.com.

[Moza] Mozilla. *box-shadow | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/box-shadow (besucht am 05. 06. 2013).

[Mozb] Mozilla. *<form> - Method Attribute - HTML | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form#attr-method (besucht am 06. 06. 2013).

[Mozc] Mozilla. *Manipulating the browser history*. URL: https://developer.mozilla.org/en-US/docs/DOM/Manipulating_the_browser_history (besucht am 15. 04. 2013).

[Mozd] Mozilla. *Mozilla Developer Network*. URL: https://developer.mozilla.org/ (besucht am 14. 03. 2013).

[Moze] Mozilla. *Using CSS gradients - Web developer guide | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_gradients (besucht am 05. 06. 2013).

[Nc] Mozilla Developer Network und individual contributors. *SpiderMonkey*. URL: https://developer.mozilla.org/en-US/docs/SpiderMonkey (besucht am 16. 03. 2013).

[Neta] Mozilla Developer Network. *Apps | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/Apps (besucht am 05. 06. 2013).

[Netb] Mozilla Developer Network. *CSS media queries - Web developer guide | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries (besucht am 05. 06. 2013).

[Netc] Mozilla Developer Network. *WebGL | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/WebGL (besucht am 05. 06. 2013).

[Nod] Node.js. *Node.js API Documentation*. URL: http://nodejs.org/api/ (besucht am 14. 04. 2013).

[oau] oauth.net. *OAuth Community Site*. URL: http://oauth.net/ (besucht am 04. 06. 2013).

[OHa] Chris O'Hara. *Node-Validator - string validation and sanitization in JavaScript*. URL: https://github.com/chriso/node-validator (besucht am 08. 04. 2013).

[Ora] Oracle. *Oracle VM Virtualbox*. URL: http://virtualbox.org/ (besucht am 30. 03. 2013).

[OS] Addy Osmani und Sindre Sorhus. *TodoMVC*. URL: http://addyosmani.github.com/todomvc/ (besucht am 11. 03. 2013).

[Osm] Addy Osmani. *Developing Backbone.js Applications*. URL: http://addyosmani.github.io/backbone-fundamentals/ (besucht am 14. 04. 2013).

[Pay] PayPal. *PayPal - online kaufen und verkaufen*. URL: https://www.paypal.com (besucht am 04. 06. 2013).

[Pila] Mark Pilgrim. *Dive Into HTML 5 - History API*. URL: http://diveintohtml5.info/history.html (besucht am 13. 03. 2013).

[Pilb] Mark Pilgrim. *Dive Into HTML 5 - Semantics*. URL: http://diveintohtml5.info/semantics.html (besucht am 13. 03. 2013).

[Pilc] Mark Pilgrim. *Web Forms - Dive Into HTML 5*. URL: http://diveintohtml5.info/forms.html (besucht am 05. 06. 2013).

[Pol] Lance Pollard. *Tower.js - Full Stack Web Framework for Node.js and the Browser*. URL: http://towerjs.org/ (besucht am 16. 03. 2013).

[Rau] Guillermo Rauch. *Socket.IO: the cross-browser WebSocket for realtime apps.* URL: http://socket.io/ (besucht am 15. 03. 2013).

[Ray96] Eric S. Raymond. „Syntactic Sugar“. In: *The New Hackers Dictionary*. 1. Aufl. Eric S. Raymond, 1996, S. 432. ISBN: 978-0262680929.

[RB12] J. Resig und B. Bibeault. *Secrets of the Javascript Ninja*. Manning Pubs Co Series. Manning Publications Company, 2012. ISBN: 9781933988696. URL: http://books.google.ch/books?id=ab8CPgAACAAJ.

[ROC] ROCA. *Resource-oriented Client Architecture*. URL: http://roca-style.org (besucht am 06. 03. 2013).

[Sch] Isaac Z. Schlueter. *TJ Holowaychuk: Components*. URL: http://blog.izs.me/post/27987129912/tj-holowaychuk-components (besucht am 20. 03. 2013).

[Sel] Alexis Sellier. *LESS - The Dynamic Stylesheet Language*. URL: http://lesscss.org/ (besucht am 13. 03. 2013).

[Sena] SenchaLabs. *Connect - methodOverride*. URL: http://www.senchalabs.org/connect/middleware-methodOverride.html (besucht am 06. 06. 2013).

[Senb] Senchalabs. *Connect - High-quality middleware for node.js*. URL: http://www.senchalabs.com/connect/ (besucht am 01. 03. 2013).

[Senc] Senchalabs. *Connect - session*. URL: http://www.senchalabs.org/connect/middleware-session.html (besucht am 30. 03. 2013).

[Str] Stripe. *Stripe API Reference*. URL: https://stripe.com/docs/api (besucht am 05. 06. 2013).

[Teaa] Git Development Team. *Git*. URL: http://git-scm.com/ (besucht am 30. 03. 2013).

[Teab] SQLite Development Team. *SQLite Home Page*. URL: http://www.sqlite.org (besucht am 30. 03. 2013).

[Tei] Pedro Teixeira. *Node Tuts – Node.js Video Tutorials*. URL: http://nodetuts.com/ (besucht am 14. 04. 2013).

[tho] thoughtbot. *Bourbon - A Sass Mixin Library*. URL: http://bourbon.io/ (besucht am 05. 06. 2013).

[Til] Stefan Tilkov. *Building large web-based systems: 10 Recommendations*. URL: http://www.innoq.com/blog/st/presentations/2013/2013-01-22-WebArchitectureRecommendations.pdf (besucht am 12. 03. 2013).

[tnt] tntim96. *JSCover*. URL: http://tntim96.github.com/JSCover/ (besucht am 17. 03. 2013).

[Twia] Twitter. *Improving Performance on Twitter.com*. URL: http://engineering.twitter.com/2012/05/improving-performance-on-twittercom.html (besucht am 06. 06. 2013).

[Twib] Twitter. *The Twitter REST API*. URL: https://dev.twitter.com/docs/api (besucht am 05. 06. 2013).

[Uni+] J. Franks Northwestern University u. a. *HTTP Authentication: Basic and Digest Access Authentication*. URL: http://tools.ietf.org/html/rfc2617 (besucht am 13. 03. 2013).

[Weia] Michael Weibel. *Express.js Prototyp*. URL: https://github.com/mweibel/BA/tree/prototype-express (besucht am 14. 03. 2013).

[Weib] Michael Weibel. *Express.js Prototyp - app.js*. URL: https://github.com/mweibel/BA/blob/prototype-express/app.js (besucht am 16. 03. 2013).

[Weic] Michael Weibel. *Sails.js Prototyp*. URL: https://github.com/mweibel/BA/tree/prototype (besucht am 14. 03. 2013).

[Wika] Wikipedia. *AJAX (Programmierung)*. URL: http://de.wikipedia.org/w/index.php?title=Ajax_(Programmierung)&oldid=118744583 (besucht am 06. 06. 2013).

[Wikb] Wikipedia. *Bidirectional Streams Over HTTP*. URL: http://en.wikipedia.org/w/index.php?title=BOSH&oldid=555755444 (besucht am 15. 03. 2013).

[Wikc] Wikipedia. *Cross-Site Request Forgery*. URL: http://en.wikipedia.org/w/index.php?title=Cross-site_request_forgery&oldid=557162317 (besucht am 08. 04. 2013).

[Wikd] Wikipedia. *gzip*. URL: http://de.wikipedia.org/w/index.php?title=Gzip&oldid=116452018 (besucht am 14. 04. 2013).

[Wike] Wikipedia. *Hash-based message authentication code*. URL: http://en.wikipedia.org/w/index.php?title=Hash-based_message_authentication_code&oldid=557186427 (besucht am 26. 04. 2013).

[Wikf] Wikipedia. *Proof Of Concept*. URL: https://de.wikipedia.org/w/index.php?title=Proof_of_Concept&oldid=116970837 (besucht am 24. 03. 2013).

[Wikg] Wikipedia. *Responsive Webdesign*. URL: http://de.wikipedia.org/w/index.php?title=Responsive_Webdesign&oldid=119168819 (besucht am 13. 03. 2013).

[Wikh] Wikipedia. *Separation of Concerns*. URL: http://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=543036001 (besucht am 09. 04. 2013).

[Wiki] Wikipedia. *SQL-Injection*. URL: http://de.wikipedia.org/w/index.php?title=SQL-Injection&oldid=118022262 (besucht am 14. 04. 2013).

[Wikj] Wikipedia. *Uniform Resource Identifier*. URL: http://de.wikipedia.org/w/index.php?title=Uniform_Resource_Identifier&oldid=118628990 (besucht am 05. 06. 2013).

[Wikk] Wikipedia. *Uniform Resource Locator*. URL: http://de.wikipedia.org/w/index.php?title=Uniform_Resource_Locator&oldid=119183264 (besucht am 05. 06. 2013).

[Wikl] Wikpedia. *Model-View-Controller*. URL: http://en.wikipedia.org/w/index.php?title=Model%C3%A2%C2%80%C2%93view%C3%A2%C2%80%C2%93controller&oldid=558414142 (besucht am 08. 04. 2013).

[ZUR] inc. ZURB. *Foundation: The Most Advanced Responsive Front-end Framework from ZURB*. URL: http://foundation.zurb.com/ (besucht am 05. 06. 2013).

Anhang C Glossar

AJAX

Ajax ist ein Apronym für die Wortfolge “Asynchronous JavaScript and XML”. Es bezeichnet ein Konzept der asynchronen Datenübertragung zwischen einem Internetbrowser und einem Webserver. [Wika]. 51

Benutzer

Ein Benutzer verwendet “Roomies” und ist in den meisten Fällen ein Bewohner.. 72, 88

Bewohner

Ein Bewohner ist einer WG zugeordnet.. 72, 118

Boilerplate

Codefragemente, welche in unveränderter Form immer wieder verwendet werden können. 58

CI

Continuous Integration. 35, 103, 123

CLI

Kurzform für *Command Line Interface*, Kommandozeile oder Terminal. 69

Cronjob

Ein Cronjob ist in der UNIX-Welt ein sich wiederholender Auftrag. Üblicherweise kann spezifiziert werden ob dieser Auftrag alle paar Minuten, Stunden oder ein- bis mehrere Male am Tag gestartet werden soll.. 74

Cross-Site Request Forgery

CSRF [Wikc] ist eine Attacke, in dem der Angreifer Transaktionen im Namen eines angemeldeten Benutzers durchführt. Um diese Attacke zu verhindern, muss für jede Transaktion eine geheime Information mitgegeben und validiert werden.. 90

CRUD

Abkürzung für “Create, Read, Update and Delete”; Die Zusammenfassung der Datenmanipulationsoperationen. 120

DOM

Document Object Model, Repräsentation eines, HTML Dokuments im Speicher. 50, 51, 55

Gamification

Als Gamification oder Gamifizierung (seltener auch Spielifizierung) bezeichnet man die Anwendung spieltypischer Elemente und Prozesse in spielfremdem Kontext [Det+]. 17, 96, 129

GPU

Graphical Processing Unit, Prozessor auf einer Grafikkarte. 64

Hashbang

Mithilfe des URL Fragment-Identifiers verwendeten viele Webseiten den sogenannten Hashbang “#¨” als Ersatz für richtige URLs. Damit dies richtig funktioniert, muss das JavaScript entsprechend umleiten, wenn eine solche URL direkt im Browser eingegeben bzw. reinkopiert wird. Google hat aufgrunddessen eine Möglichkeit gefunden, um Hashbang-URLs zu crawlen (siehe [Goob]). Nach der Entwicklung der History API haben viele Webseiten dies aber wieder komplett abgeschafft (u.a. Twitter [Twia]).. 59

JSON

JavaScript Object Notation. 36, 60

Message Authentication Code

Ein MAC wird verwendet um Daten zu signieren und somit gegen Änderungen jeglicher Art zu schützen. Das gängigste Beispiel ist dazu HMAC [Wike].. 46

Middleware

Middlewares in Express.js [Expb] werden verwendet, um Funktionalitäten wie Cookies, Sessions etc. einer Applikation hinzuzufügen. Dies erlaubt es, jede Applikation genau auf die benötigten Funktionalitäten zu beschränken.. 45

Multiple Inheritance

Multiple Inheritance bezeichnet ein Konzept, welches es erlaubt, eine Klassen von mehr als einer Oberklasse erben zu lassen (vgl. Single Inheritance). 21

Node.js

Node.js ist ein Framework um Javascript auf dem Server laufen zu lassen. Es ist auf der V8 Engine von Google Chrome aufgebaut. 90

NoSQL

NoSQL Datenbanken ist eine relativ neue Art um Datenbanken ohne die SQL-Sprache zu entwickeln. Vielfach haben solche Datenbanken keine Relationen im Sinne der Relationalen Datenbanken implementiert.. 32

ORM

Ein "Object Relational Mapper" wird verwendet um Entitäten auf einer Relationalen Datenbank abzubilden und verwenden zu können. 25, 26, 28, 31, 90

Proof of Concept

Im Projektmanagement ist ein Proof of Concept, auch als Proof of Principle bezeichnet (zu Deutsch: Machbarkeitsnachweis), ein Meilenstein, an dem die prinzipielle Durchführbarkeit eines Vorhabens belegt ist. [Wikf]. 11

Real-Time

Mit "Real-Time" ist in Web-Applikationen meistens "Soft-Real-Time" gemeint. Antwortzeiten sind somit nicht garantiert, sind aber möglichst klein gehalten (im Milisekunden-Bereich). Mittels Websockets oder BOSH [Wikb] sind solche Applikationen im Web realisierbar. 25, 26, 28

REST

Representational State Transfer, definiert von Roy Fielding in seiner Dissertation [Fie00]. 28, 36, 38, 39, 42, 60

RESTful

Eine API kann als RESTful bezeichnet werden, wenn sie den Prinzipien von REST entspricht. Ein Konsument einer REST-API kann die Kommunikation als RESTful bezeichnen, wenn diese ebenfalls den Prinzipien von REST entspricht.. 39, 41

RUP

Rational Unified Process; Iteratives Projektvorgehen [IBM]. 96

SAD

Software Architektur Dokument. 98, 99

Scaffolding

Scaffolding bezeichnet das toolunterstützte Generieren von Quellcodefragmenten. Beispiel: Erstellung einer Model-Klasse inkl. zugehörigen CRUD-Controller. 21, 22, 25, 28

SDK

Software Development Kit, eine Code-Bibliothek eines Anbieters um dessen Dienste in einer eigenen Applikation anzubinden. 63

Search Engine Spider

Computerprogramme, welche vor allem von Suchmachinen verwendet werden, um Webseiten zu durchsuchen und zu analysieren. 47

SQL

Structured Query Language. Sprache zur Abfrage von Daten in Datenbanksystemen. 90

TDD

Abkürzung für test-driven development (testgetriebene Entwicklung). TDD ist eine Methode in der Softwareentwickler, bei welcher erst die Tests geschrieben werden und dann die dazugehörige Funktion.. 102

URI

Uniform Resource Identifier [Wikj], identifiziert eine abstrakte oder physische Ressource.. 36, 46, 121

URL

Uniform Resource Locator [Wikk], identifiziert z.B. eine Webseite in einem Netzwerk. Oft ein Synonym für *Internetadresse*. Es ist eine Unterart von URI.. 59, 63

URL Fragment

In einer URL gibt es den sogenannten “Fragment-Identifier”, ausgedrückt durch ein “#” (English: “Hash”) am Schluss der URL. Nach diesem Fragment-Identifier können beliebige eindeutige Bezeichnungen verwendet werden, um z.B. auf einen Anker einer Überschrift zu verlinken. Der Nachteil an Fragment-Identifiers ist aber, dass es nur von JavaScript aus genutzt werden kann. Es wird nicht an den Server gesendet.. 119

VM

Virtual Machine. 99

WebRTC

Web Real-Time-Communication [Good] beschreibt eine Reihe von JavaScript-APIs und dessen Implementationen in den Browsern. Diese erlaubt es Webapplikationen Peer-to-Peer Verbindungen zwischen mehreren Besuchern der Webseite aufzubauen. Solche Verbindungen können zur Übertragung von Video, Audio oder Daten

verwendet werden und sind somit prädestiniert für eine grosse Anzahl von Anwedungen (inbesondere aber Audio/Video-Konferenz-Software und Spiele).. 39

Websocket

Websockets ermöglichen das Herstellen von Verbindungen zwischen einem Browser und dem Webserver ausserhalb des eigentlichen HTTP Kontextes. Auf diese Weise werden "Serverside-Pushes" auf einfache Art und weise möglich: Über die persistente Verbindung kann der Server jederzeit ohne erneuten Request des Clients Daten an diesen senden. Websockets werden heute von praktisch allen modernen Browsern unterstützt [Devb]. 25, 28, 120

WG

Eine WG ist eine Wohngemeinschaft und kann auf "Roomies" eröffnet werden. . 17, 72, 84, 118, 128, 129

Anhang D Projektrelevante URL's

Ressource	*URL*
Projektverwaltung	`http://redmine.alabor.me/projects/ba2013`
Code: Git Repository	`https://github.com/hsr-ba-ajw-2013/BA`
Code: CI	`https://travis-ci.org/hsr-ba-ajw-2013/BA`
Code: Dokumentation	`http://hsr-ba-ajw-2013.github.com/BA/docs`
Code: Unit Test Coverage Report	`http://hsr-ba-ajw-2013.github.com/BA/unit-coverage.html`
Code: Functional Test Coverage Report	`http://hsr-ba-ajw-2013.github.com/BA/functional-coverage.html`
Thesis: Git Repository	`https://github.com/hsr-ba-ajw-2013/BA-Dokumentation`
Thesis: PDF	`http://hsr-ba-ajw-2013.github.com/BA-Dokumentation/thesis.pdf`
Thesis: CI	`https://travis-ci.org/hsr-ba-ajw-2013/BA-Dokumentation`
Meeting Protokollierung	`https://github.com/hsr-ba-ajw-2013/BA-Dokumentation/wiki/Meetings`

Tabelle D.1.: Projektrelevante URL's

Anhang E Installationsanleitung

E.1. Mit Vagrant

Grundvoraussetzungen

Folgende Software muss im Voraus installiert sein. Es wird hier nicht genauer darauf eingegangen wie diese installiert wird.

- **git** [Teaa]
- **Vagrant** [Has]
- **Virtualbox** [Ora]

Installation der Roomies-Virtualbox

```
1  ~ $> git clone git://github.com/hsr-ba-ajw-2013/BA.git && cd BA
2
3  # Virtualbox starten
4  ~BA/ $> vagrant up
5
6  # Zur Virtualbox per SSH verbinden
7  ~BA/ $> vagrant ssh
8
9  # Initiales setup
10 ~BA/ $> ./install.sh
11
12 # Applikation starten
13 ~BA/ $> npm start
```

Quelltext E.1: Installationsanleitung Vagrant

E.2. Ohne Vagrant

Grundvoraussetzungen

Folgende Software muss im Voraus installiert sein. Es wird hier nicht genauer darauf eingegangen wie diese installiert wird.

- **Node.js** Version 0.8.x [Joyb]
  Eventuell mittels **NVM** [Cas] falls nicht anders möglich.
- **PostgreSQL** [Gro] oder SQLite [Teab]
- **GNU Make** [Foua]
- **git** [Teaa]

Installation der Roomies-Applikation

```
1 ~ $> git clone git://github.com/hsr-ba-ajw-2013/BA.git && cd BA
2
3 # Initiales setup
4 ~BA/ $> ./install.sh
5
6 # Applikation starten
7 ~BA/ $> npm start
```

Quelltext E.2: Installationsanleitung ohne Vagrant

Konfiguration

Für die Konfiguration müssen die Datenbank-Informationen angepasst werden.

Anhang F Technologie Einführung

Der folgende Anhang beschreibt einige Ressourcen und Tutoriale um sich selber auf den aktuellen Stand in Sachen JavaScript und Node.js zu bringen. Weiter zeigt F.3 einige Websites für Libraries die in der Beispielapplikation benutzt wurden, auf.

F.1. JavaScript

Fortgeschrittenes

- JavaScript: The Good Parts [Cro08]
  Sobald ein Basiswissen über JavaScript erreicht wurde, ist dieses Buch quasi unumgänglich. Es klärt über die Unzulänglichkeiten von JavaScript auf und gibt wertvolle Tipps, die JavaScript angenehm machen.

- Javascript Module Pattern [Che]
  Damit grössere Applikationen strukturiert werden können benötigt es momentan noch (bis ECMAScript 6 Standard ist) klar definierte Patterns. Das Module Pattern ist nach Ansicht der Autoren eines der besten davon.

- Secrets of the JavaScript Ninja [RB12]
  Vom jQuery-Author beschreibt dieses Buch Techniken um zum JavaScript-Guru aufzusteigen.

Weitere Links

- Mozilla Developer Network Wiki [Mozd]
  Ausführliches Nachschlagewerk für JavaScript, CSS und andere Web–APIs und –Technologien.

F.2. Node.js

Einführung

- Felix's Node.js Guide [Gei]
  Felix Geisendörfer ist ein Kern-Entwickler von node.js und hat verschiedene Tutoriale über Node.js geschrieben.

- Node Tutorials [Tei]
- Mastering Node.js [Holc]
  Kostenloses Buch vom Express.js [Expb] etc. Entwickler TJ Holowaychuck [Holf]. Beinhaltet sowohl Informationen für Anfänger wie auch für Fortgeschrittene.
- Node.js API Docs [Nod]

F.3. Einführung für benutzte Libraries

- Express.js Guide [Holb]
  Erste Schritte mit Express.js können getrost mithilfe des Guides von Express.js gemacht werden. Sobald man die grundsätzliche Struktur einer Applikation mithilfe des Frameworks versteht, ist es hilfreich, einige Beispiele [Hola] anzuschauen. Die "Roomies" Applikation kann natürlich auch als Beispiel fungieren.
- Handlebars.js [Kat]
  Die Handlebars.js Website bietet direkt eine Einführung in das Templating.
- Developing Backbone.js Applications [Osm]
  Dieses Buch beinhaltet eine Einführung in die relevanten Konzepte von Backbone.js, zeigt Beispielapplikationen und auch weiterführende Libraries/Konzepte.

G.1. Workshop

Um die zweitrangige Prozedur der Ideenfindung pragmatisch abhandeln zu können wurde ein Workshop mit Brainstorming und anschliessender Diskussionsrunde durchgeführt. Folgende Tabelle zeigt die Favoriten aus einem Pool generierter Ideen.

Die Spalte *Potential* bewertet jede Idee nach subjektiver Einschätzung des Projektteams unter Berücksichtigung folgender Faktoren:

- Funktionsumfang
- Konzeptionelle und technische Herausforderung
- Attraktivität (Für Projektteam)
- Attraktivität (Für Studierende des Moduls Internettechnologien)

Idee	***Pro***	***Contra***	***Potential***
WG-Aufgabenverwaltung	Viele Studierende identifizieren sich tendenziell damit, da sie selber in einer WG wohnen, Faktor *Gamification* sehr interessant		★★★
Instant Messenger	Attraktive Features wären realisierbar (Realtime, Websockets etc.)	Funktionelle Anforderungen könnten Rahmen sprengen	★★
Aufgabenverwaltung	Evtl. gute Verwendung des Produkts	"*Gibt's wie Sand am Meer*", viele bestehende Beispielapplikationen [OS]	★★
Chat		Bereits oft verwendet in bestehender Vorlesung, abgenutzte Thematik	★
Forum		"*Gibt's wie Sand am Meer*"	★

Tabelle G.1.: Produktideenpool

Am verheissungsvollsten wurde die Idee des WG Aufgabenverwaltungstool eingeschätzt und gefiel dem gesamten Team von Beginn an ziemlich gut. Die Thematik Gamification in einem konkreten Produkt umsetzen zu können eliminierte schlussendlich die letzten Zweifel.

In einem nächsten Schritt wurde die rohe Produktidee mit einem Mindmap weiter ausgebaut und die ersten funktionalen Anforderungen wurden entwickelt. Daneben konnte eine konkrete Kurzbeschreibung sowie das erste Branding für das geplante Produkt formuliert resp. entworfen werden.

G.2. Branding

Grundfarbpalette

Neutral #FFFFFF, RGB(255,255,255)

Akzent 1 #1A3143, RGB(26,49,67)

Akzent 2 #A4000F, RGB(164,0,15)

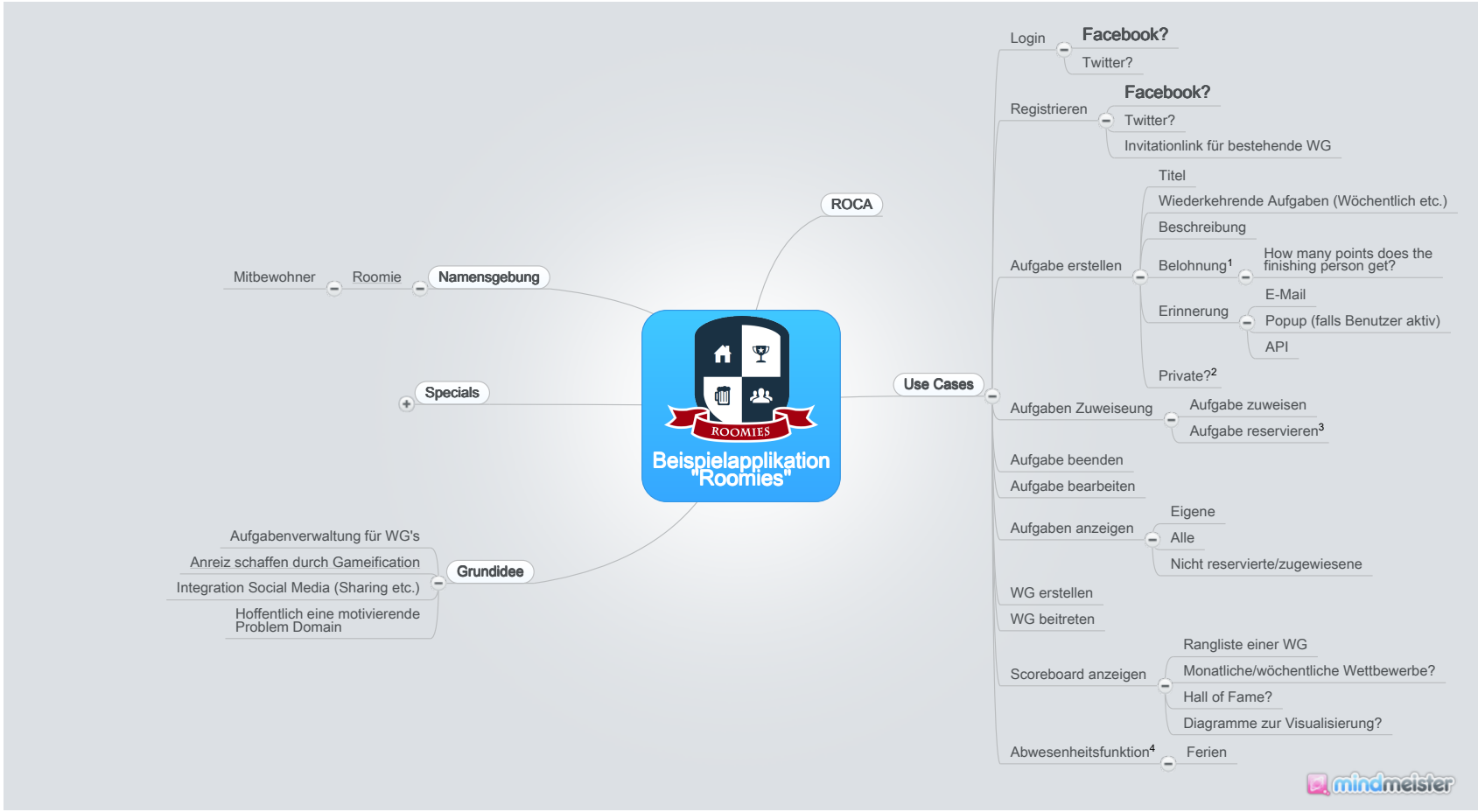
Abbildung G.1.: Branding Farbpalette

Logo & Logovariationen

Abbildung G.2.: Roomies Logo im College Stil

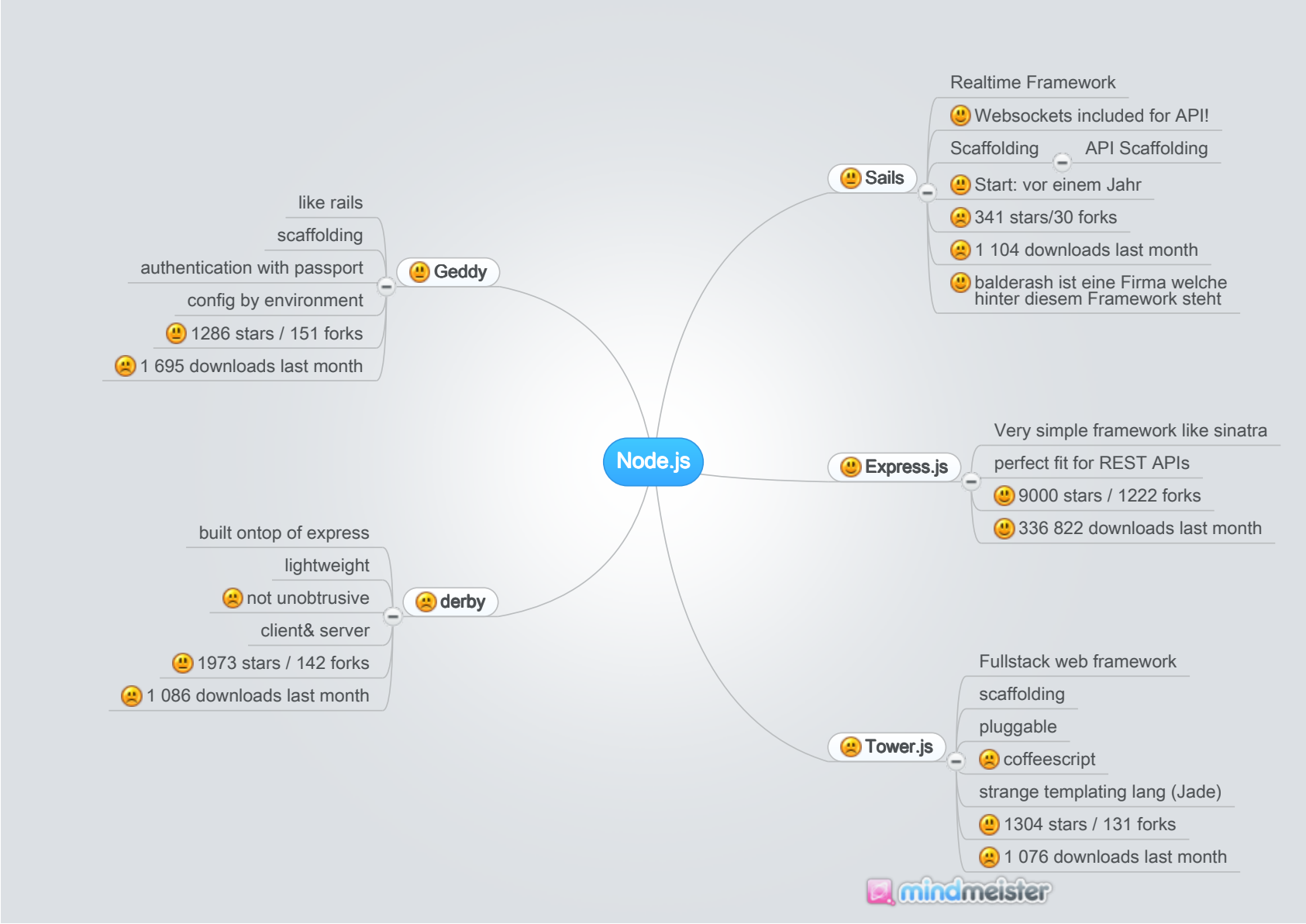
Abbildung G.3.: Roomies Logo in verschiedenen Grössen & Varianten

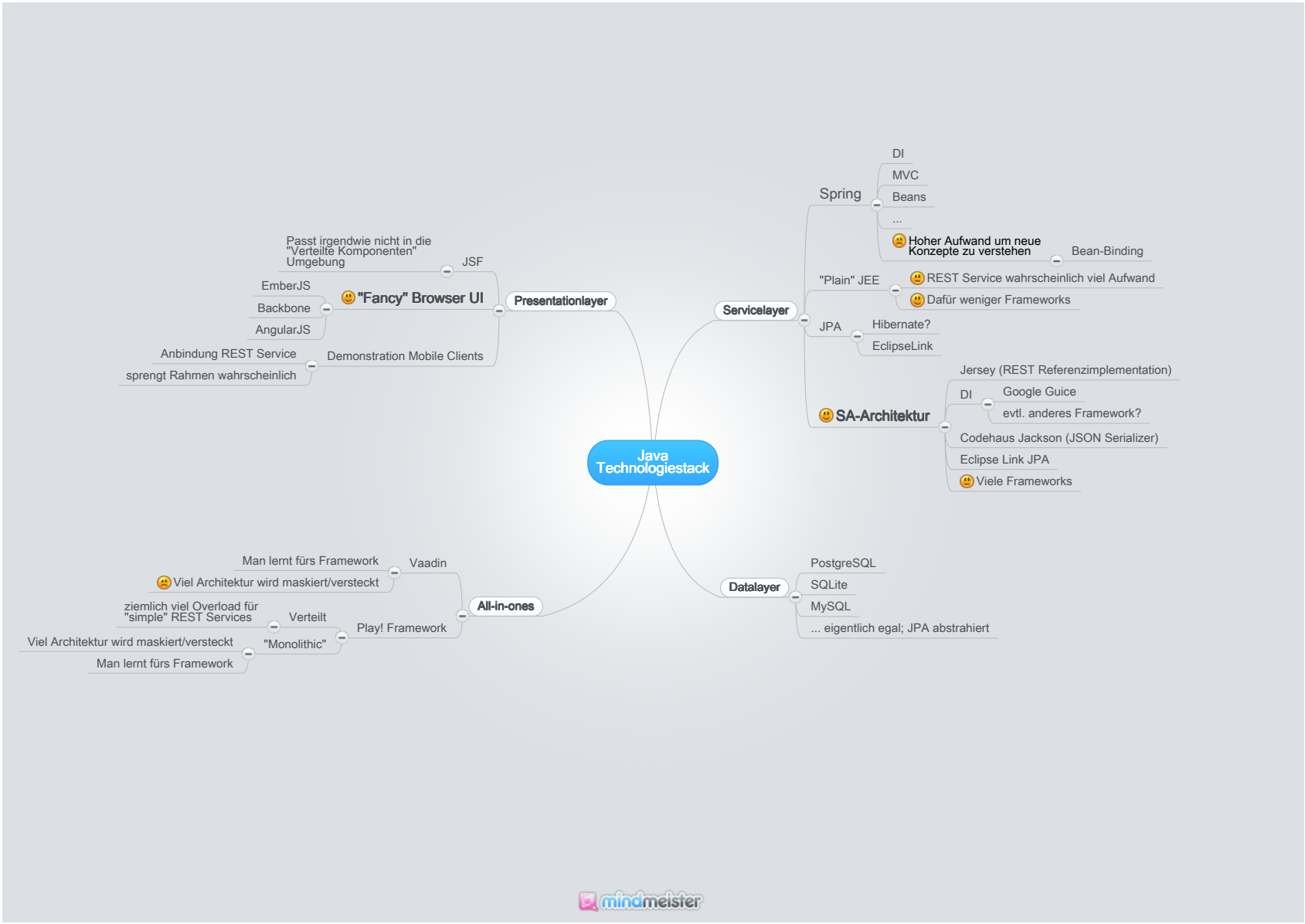
G.3. Mindmap


Beispielapplikation "Roomies"
ROOMIES
Namensgebung
Mitbewohner
Roomie
Specials
Grundidee
Aufgabenverwaltung für WG's
Anreiz schaffen durch Gameification
Integration Social Media (Sharing etc.)
Hoffentlich eine motivierende Problem Domain
ROCA
Use Cases
Login
Facebook?
Twitter?
Registrieren
Facebook?
Twitter?
Invitationlink für bestehende WG
Aufgabe erstellen
Titel
Wiederkehrende Aufgaben (Wöchentlich etc.)
Beschreibung
Belohnung[1]
How many points does the finishing person get?
Erinnerung
E-Mail
Popup (falls Benutzer aktiv)
API
Private?[2]
Aufgaben Zuweiseung
Aufgabe zuweisen
Aufgabe reservieren[3]
Aufgabe beenden
Aufgabe bearbeiten
Aufgaben anzeigen
Eigene
Alle
Nicht reservierte/zugewiesene
WG erstellen
WG beitreten
Scoreboard anzeigen
Rangliste einer WG
Monatliche/wöchentliche Wettbewerbe?
Hall of Fame?
Diagramme zur Visualisierung?
Abwesenheitsfunktion[4]
Ferien
mindmeister

Anhang H Technologieevaluation

Auf den nächsten Seiten finden sich Mindmaps welche in der Evaluationsphase für die Entscheidungsfindung benutzt wurden.


Node.js
Sails
Realtime Framework
Websockets included for API!
Scaffolding
API Scaffolding
Start: vor einem Jahr
341 stars/30 forks
1 104 downloads last month
balderash ist eine Firma welche hinter diesem Framework steht
Geddy
like rails
scaffolding
authentication with passport
config by environment
1286 stars / 151 forks
1 695 downloads last month
Express.js
Very simple framework like sinatra
perfect fit for REST APIs
9000 stars / 1222 forks
336 822 downloads last month
derby
built ontop of express
lightweight
not unobtrusive
client& server
1973 stars / 142 forks
1 086 downloads last month
Tower.js
Fullstack web framework
scaffolding
pluggable
coffeescript
strange templating lang (Jade)
1304 stars / 131 forks
1 076 downloads last month
mindmeister


Java Technologiestack
Presentationlayer
"Fancy" Browser UI
EmberJS
Backbone
AngularJS
JSF
Passt irgendwie nicht in die "Verteilte Komponenten" Umgebung
Demonstration Mobile Clients
Anbindung REST Service
sprengt Rahmen wahrscheinlich
Servicelayer
Spring
DI
MVC
Beans
...
Hoher Aufwand um neue Konzepte zu verstehen
Bean-Binding
"Plain" JEE
REST Service wahrscheinlich viel Aufwand
Dafür weniger Frameworks
JPA
Hibernate?
EclipseLink
SA-Architektur
Jersey (REST Referenzimplementation)
DI
Google Guice
evtl. anderes Framework?
Codehaus Jackson (JSON Serializer)
Eclipse Link JPA
Viele Frameworks
Datalayer
PostgreSQL
SQLite
MySQL
... eigentlich egal; JPA abstrahiert
All-in-ones
Vaadin
Man lernt fürs Framework
Viel Architektur wird maskiert/versteckt
Play! Framework
Verteilt
ziemlich viel Overload für "simple" REST Services
"Monolithic"
Viel Architektur wird maskiert/versteckt
Man lernt fürs Framework
mindmeister

Anhang I Coding Style Guideline

Dieses Kapitel enthält die JavaScript Coding Style Guideline, welche zur Erstellung von entsprechendem Quellcode im Rahmen dieser Bachelorarbeit benutzt wird.

Die Guideline basiert auf einem von Airbnb [Aira] veröffentlichten Dokument. Sie wurde zusätzlich gemäss eigenen Präferenzen [Airb] des Projektteams erweitert und optimiert.

Original Repository: airbnb/javascript

Airbnb JavaScript Style Guide() {

Ein vernünftiger Ansatz für einen JavaScript-Style-Guide

Inhaltsverzeichnis

Datentypen

- **Primitive Typen**: Bei primitiven Datentypen wird immer direkt auf deren Wert zugegriffen.
  - `string`
  - `number`
  - `boolean`
  - `null`
  - `undefined`

```
var foo = 1
    , bar = foo;

bar = 9;

console.log(foo, bar); // => 1, 9
```

- **Komplexe Typen**: Bei komplexen Datentypen wird immer auf eine Referenz zugegriffen.
  - `object`
  - `array`
  - `function`

```
var foo = [1, 2]
    , bar = foo;

bar[0] = 9;

console.log(foo[0], bar[0]); // => 9, 9
```

[⬆]

Objekte

- Benutze die `literal syntax`, um Objekte zu erzeugen.

```
// schlecht
var item = new Object();

// gut
var item = {};
```

- Benutze keine reservierten Wörter für Attribute.

```
// schlecht
var superman = {
  class: 'superhero'
  , default: { clark: 'kent' }
  , private: true
};

// gut
var superman = {
  klass: 'superhero'
  , defaults: { clark: 'kent' }
  , hidden: true
};
```

[⬆]

Arrays

- Benutze die `literal syntax`, um Arrays zu erzeugen.

```
// schlecht
var items = new Array();

// gut
var items = [];
```

- Wenn du die Array-Länge nicht kennst, benutze `Array#push`.

```
var someStack = [];

// schlecht
someStack[someStack.length] = 'abracadabra';

// gut
someStack.push('abracadabra');
```

- Wenn du ein Array kopieren möchtest, benutze `Array#slice`. jsPerf

```
var len = items.length
    , itemsCopy = []
    , i;

// schlecht
```

```
for (i = 0; i < len; i++) {
  itemsCopy[i] = items[i];
}

// gut
itemsCopy = Array.prototype.slice.call(items);
```

[⬆]

Zeichenketten

- Benutze einfache Anführungszeichen `''` für Zeichenketten

```
// schlecht
var name = "Bob Parr";

// gut
var name = 'Bob Parr';

// schlecht
var fullName = "Bob " + this.lastName;

// gut
var fullName = 'Bob ' + this.lastName;
```

- Zeichenketten die länger als 80 Zeichen lang sind, sollten mit Hilfe von `string concatenation` auf mehrere Zeilen aufgeteilt werden.
- Beachte: Benutzt man `string concatenation` zu oft kann dies die performance beeinträchtigen. jsPerf & Discussion

```
// schlecht
var errorMessage = 'This is a super long error that was thrown because of Batman. When you stop to think about hou

// schlecht
var errorMessage = 'This is a super long error that \
was thrown because of Batman. \
When you stop to think about \
how Batman had anything to do \
with this, you would get nowhere \
fast.';

// gut
var errorMessage = 'This is a super long error that ' +
  'was thrown because of Batman.' +
  'When you stop to think about ' +
  'how Batman had anything to do ' +
  'with this, you would get nowhere ' +
  'fast.';
```

- Wenn man im Programmverlauf eine Zeichenkette dynamisch zusammensetzen muss, sollte man `Array#join` einer `string concatenation` vorziehen. Vorallem für den IE. jsPerf.

```
var items
    , messages
    , length, i;

messages = [{
    state: 'success'
    , message: 'This one worked.'
},{
    state: 'success'
    , message: 'This one worked as well.'
},{
    state: 'error'
    , message: 'This one did not work.'
}];
```

```
length = messages.length;

// schlecht
function inbox(messages) {
  items = '<ul>';

  for (i = 0; i < length; i++) {
    items += '<li>' + messages[i].message + '</li>';
  }

  return items + '</ul>';
}

// gut
function inbox(messages) {
  items = [];

  for (i = 0; i < length; i++) {
    items[i] = messages[i].message;
  }

  return '<ul><li>' + items.join('</li><li>') + '</li></ul>';
}
```

[⬆]

Funktionen

- Funktionsausdrücke:

```
// anonyme Fuktionsausdrücke
var anonymous = function() {
  return true;
};

// benannte Funktionsausdrücke
var named = function named() {
  return true;
};

// direkt ausgeführte Funktionsausdrücke (IIFE)
(function() {
  console.log('Welcome to the Internet. Please follow me.');
})();
```

- Vermeide Funktionen in `non-function blocks` zu deklarieren. Anstelle sollte die Funktion einer Variablen zugewiesen werden. Dies hat den Grund, dass die verschiedenen Browser dies unterschiedlich interpretieren.

- **Beachte:** ECMA-262 definiert einen Block als eine Abfolge/Liste von Statements. Eine Funktion hingegen ist **kein** Statement. Read ECMA-262's note on this issue.

```
// schlecht
if (currentUser) {
  function test() {
    console.log('Nope.');
  }
}

// gut
if (currentUser) {
  var test = function test() {
    console.log('Yup.');
  };
}
```

- Benenne einen Parameter nie `arguments`, denn dies wird das `arguments`-Objekt, dass in jedem Funktionskörper zur Verfügung steht, überschreiben.

```
// schlecht
function nope(name, options, arguments) {
  // ...stuff...
}

// gut
function yup(name, options, args) {
  // ...stuff...
}
```

[⬆]

Eigenschaften

- Benutze die Punktnotation, um auf die Eigenschaften eines Objekts zuzugreifen.

```
var luke = {
  jedi: true
  , age: 28
};

// schlecht
var isJedi = luke['jedi'];

// gut
var isJedi = luke.jedi;
```

- Benutze die Indexnotation `[]`, um auf die Eigenschaften eines Objekts zuzugreifen, sofern der Index eine Variable ist.

```
var luke = {
  jedi: true
  , age: 28
};

function getProp(prop) {
  return luke[prop];
}

var isJedi = getProp('jedi');
```

[⬆]

Variablen

- Benutze immer `var`, um Variablen zu deklarieren. Tut man dies nicht, werden die Variablen im globalen Namespace erzeugt – was nicht gewüscht werden sollte.

```
// schlecht
superPower = new SuperPower();

// gut
var superPower = new SuperPower();
```

- Benutze immer nur ein `var`, um mehrere aufeinanderfolgende Variablen zu deklarieren. Deklariere jede Variable auf einer eigenen Zeile.

```
// schlecht
var items = getItems();
var goSportsTeam = true;
var dragonball = 'z';

// gut
var items = getItems()
    , goSportsTeam = true
    , dragonball = 'z';
```

- Deklariere Variablen ohne direkte Zuweisung immer als letztes. Dies ist vorallem hilfreich, wenn man später eine Variable anhand einer zuvor deklarierten Variable initialisieren möchte.

```
// schlecht
var i, len, dragonball
    , items = getItems()
    , goSportsTeam = true;

// schlecht
var i, items = getItems()
    , dragonball
    , goSportsTeam = true
    , len;

// gut
var items = getItems()
    , goSportsTeam = true
    , dragonball
    , i
    , length;
```

- Weise den Wert einer Variable, wenn möglich, immer am Anfang des Gültigkeitsbereichs zu. Dies hilft Problemen mit der Variablendeklaration vorzubeugen.

```
// schlecht
function() {
  test();
  console.log('doing stuff..');

  //..other stuff..

  var name = getName();

  if (name === 'test') {
    return false;
  }

  return name;
}

// gut
function() {
  var name = getName();

  test();
  console.log('doing stuff..');

  //..other stuff..

  if (name === 'test') {
    return false;
  }

  return name;
}

// schlecht
function() {
  var name = getName();

  if (!arguments.length) {
    return false;
  }

  return true;
}

// gut
function() {
```

```
  if (!arguments.length) {
    return false;
  }

  var name = getName();

  return true;
}
```

[⬆]

Hoisting

- Variablendeklarationen werden vom Interpreter an den Beginn eines Gültigkeitbereichs genommen, genannt (`hoisting`). Wohingegen die Zuweisung an der ursprünglichen Stelle bleibt.

```
// Dies wird nicht funktionen (angenommen
// notDefined ist keine globale Variable)
function example() {
  console.log(notDefined); // => throws a ReferenceError
}

// Wird eine Variable nach seiner ersten
// Referenzierung deklariert, funktioniert
// dies dank des hoistings.
// Beachte aber, dass die Zuweisung von true
// erst nach der Referenzierung stattfindet.
function example() {
  console.log(declaredButNotAssigned); // => undefined
  var declaredButNotAssigned = true;
}

// Der Interpreter nimmt die Variablendeklaration
// an den Beginn des Gültigkeitbereichs.
// So kann das Beispiel wiefolgt umgeschrieben
// werden:
function example() {
  var declaredButNotAssigned;
  console.log(declaredButNotAssigned); // => undefined
  declaredButNotAssigned = true;
}
```

- Anonyme Funktionen `hoisten` ihren Variablennamen, aber nicht die Funktionszuweisung.

```
function example() {
  console.log(anonymous); // => undefined

  anonymous(); // => TypeError anonymous is not a function

  var anonymous = function() {
    console.log('anonymous function expression');
  };
}
```

- Benannte Funktionen `hoisten` ihren Variablennamen, aber nicht der Funktionsname oder Funktionskörper.

```
function example() {
  console.log(named); // => undefined

  named(); // => TypeError named is not a function

  superPower(); // => ReferenceError superPower is not defined

  var named = function superPower() {
    console.log('Flying');
  };
```

```
  // Das gleiche gilt, wenn der Funktionsname
  // derselbe ist, wie der Variablenname
  function example() {
    console.log(named); // => undefined

    named(); // => TypeError named is not a function

    var named = function named() {
      console.log('named');
    };
  }
}
```

- Funktionsdeklarationen `hoisten` ihren Namen und ihren Funktionskörper.

```
function example() {
  superPower(); // => Flying

  function superPower() {
    console.log('Flying');
  }
}
```

- Für weitere Informationen siehe hier: JavaScript Scoping & Hoisting by Ben Cherry

[⬆]

Bedingungen und Gleichheit

- Ziehe `===` und `!==` gegenüber `==` und `!=` vor.
- Bedingungsausdrücke werden immer gezwungen der `ToBoolean` Methode ausgewertet zu werden. Diese folgt den folgenden einfachen Grundregeln:
  - **Objekte** werden als **true** gewertet
  - **Undefined** wird als **false** gewertet
  - **Null** wird als **false** gewertet
  - **Booleans** werden als **der Wert des Booleans** gewertet
  - **Zahlen** werden als **false** gewertet sofern **+0, -0, or NaN**, ansonsten als **true**
  - **Zeichenketten** werden als **false** gewertet, sofern sie leer ist `''`, ansonsten als **true**

```
if ([0]) {
  // true
  // Arrays sind Objekte und Objekte werden als true ausgewertet
}
```

- Benutze `shortcuts`

```
// schlecht
if (name !== '') {
  // ...stuff...
}

// gut
if (name) {
  // ...stuff...
}

// schlecht
if (collection.length > 0) {
  // ...stuff...
}

// gut
if (collection.length) {
  // ...stuff...
}
```

- Für weitere Informationen siehe hier: Truth Equality and JavaScript by Angus Croll

[⬆]

Blöcke

- Benutze geschweifte Klammern für alle mehrzeiligen Blöcke.

```
// schlecht
if (test)
  return false;

// gut
if (test) return false;

// gut
if (test) {
  return false;
}

// schlecht
function() { return false; }

// gut
function() {
  return false;
}
```

[⬆]

Kommentare

- Benutze `/** ... */` für mehrzeilige Kommentare. Daran kann eine Beschreibung, eine Typendefinition und Werte für alle Parameter und den Rückgabewert angegeben werden.

```
// schlecht
// make() returns a new element
// based on the passed in tag name
//
// @param <String> tag
// @return <Element> element
function make(tag) {

  // ...stuff...

  return element;
}

// gut
/**
 * make() returns a new element
 * based on the passed in tag name
 *
 * @param <String> tag
 * @return <Element> element
 */
function make(tag) {

  // ...stuff...

  return element;
}
```

- Benutze `//` für einzeilige Kommentare. Platziere den Kommentar auf einer separaten Zeile oberhalb der beschriebenen Zeile. Vor den Kommentar kommt eine Leerzeile.

```
// schlecht
```

```
var active = true;  // is current tab

// gut
// is current tab
var active = true;

// schlecht
function getType() {
  console.log('fetching type...');
  // set the default type to 'no type'
  var type = this._type || 'no type';

  return type;
}

// gut
function getType() {
  console.log('fetching type...');

  // set the default type to 'no type'
  var type = this._type || 'no type';

  return type;
}
```

[⬆]

Whitespace

- Benutze Tabulatoren.

```
// schlecht
function() {
····var name;
}

// schlecht
function() {
·var name;
}

// gut
function() {
–var name;
}
```

- Platziere ein Leerzeichen vor einer öffnenden Klammer.

```
// schlecht
function test(){
  console.log('test');
}

// gut
function test() {
  console.log('test');
}

// schlecht
dog.set('attr',{
  age: '1 year'
  , breed: 'Bernese Mountain Dog'
});

// gut
dog.set('attr', {
  age: '1 year'
  , breed: 'Bernese Mountain Dog'
```

```
});
```

- Platziere eine Leerzeile an das Ende der Datei.

```
// schlecht
(function(global) {
  // ...stuff...
})(this);
```

```
// gut
(function(global) {
  // ...stuff...
})(this);
```

- Rücke bei langen Methodenverkettungen ein.

```
// schlecht
$('#items').find('.selected').highlight().end().find('.open').updateCount();

// gut
$('#items')
  .find('.selected')
    .highlight()
    .end()
  .find('.open')
    .updateCount();

// schlecht
var leds = stage.selectAll('.led').data(data).enter().append("svg:svg").class('led', true)
    .attr('width',  (radius + margin) * 2).append("svg:g")
    .attr("transform", "translate(" + (radius + margin) + "," + (radius + margin) + ")")
    .call(tron.led);

// gut
var leds = stage.selectAll('.led')
    .data(data)
  .enter().append("svg:svg")
    .class('led', true)
    .attr('width',  (radius + margin) * 2)
  .append("svg:g")
    .attr("transform", "translate(" + (radius + margin) + "," + (radius + margin) + ")")
    .call(tron.led);
```

[⬆]

Führende Kommas

- **Ja.**

```
// schlecht
var once,
    upon,
    aTime;

// gut
var once
    , upon
    , aTime;

// schlecht
var hero = {
  firstName: 'Bob',
  lastName: 'Parr',
  heroName: 'Mr. Incredible',
  superPower: 'strength'
};
```

```
// gut
var hero = {
    firstName: 'Bob'
    , lastName: 'Parr'
    , heroName: 'Mr. Incredible'
    , superPower: 'strength'
};
```

[⬆]

Semikolons

- **Ja.**

```
// schlecht
(function() {
  var name = 'Skywalker'
  return name
})()

// gut
(function() {
  var name = 'Skywalker';
  return name;
})();
```

[⬆]

Typumwandlung

- Benutze `type coercion` am Anfang eines Statements.
- Bei Zeichenketten:

```
//  => this.reviewScore = 9;

// schlecht
var totalScore = this.reviewScore + '';

// gut
var totalScore = '' + this.reviewScore;

// schlecht
var totalScore = '' + this.reviewScore + ' total score';

// gut
var totalScore = this.reviewScore + ' total score';
```

- Benutze immer `parseInt` für Zahlen und gebe immer eine Basis für die Typumwandlung an.
- Wenn man aus Performancegründen kein `parseInt` verwenden will und ein `Bitshifting` benutzt, sollte man einen Kommentar hinterlassen, wieso dies gemacht wurde.

```
var inputValue = '4';

// schlecht
var val = new Number(inputValue);

// schlecht
var val = +inputValue;

// schlecht
var val = inputValue >> 0;

// schlecht
var val = parseInt(inputValue);
```

```
// gut
var val = Number(inputValue);

// gut
var val = parseInt(inputValue, 10);

// gut
/**
 * parseInt was the reason my code was slow.
 * Bitshifting the String to coerce it to a
 * Number made it a lot faster.
 */
var val = inputValue >> 0;
```

- Bei Booleans:

```
var age = 0;

// schlecht
var hasAge = new Boolean(age);

// gut
var hasAge = Boolean(age);

// gut
var hasAge = !!age;
```

[⬆]

Namenskonventionen

- Benutze keine `einzeichigen` Namen. Die Namen sollten beschreibend sein.

```
// schlecht
function q() {
  // ...stuff...
}

// gut
function query() {
  // ..stuff..
}
```

- Benutze `camelCase`, um Objekte, Funktionen und Instanzen zu benennen.

```
// schlecht
var OBJEcttssss = {};
var this_is_my_object = {};
var this-is-my-object = {};
function c() {};
var u = new user({
  name: 'Bob Parr'
});

// gut
var thisIsMyObject = {};
function thisIsMyFunction() {};
var user = new User({
  name: 'Bob Parr'
});
```

- Benutze `PascalCase`, um Klassen und Konstrukturen zu benennen.

```
// schlecht
function user(options) {
  this.name = options.name;
```

```
}

var bad = new user({
  name: 'nope'
});

// gut
function User(options) {
  this.name = options.name;
}

var good = new User({
  name: 'yup'
});
```

- Benutze führende Untenstriche `_`, um private Eigenschaften zu benennen.

```
// schlecht
this.__firstName__ = 'Panda';
this.firstName_ = 'Panda';

// gut
this._firstName = 'Panda';
```

- Um eine Referenz an `this` zuzuweisen, benutze `_this`.

```
// schlecht
function() {
  var self = this;
  return function() {
    console.log(self);
  };
}

// schlecht
function() {
  var that = this;
  return function() {
    console.log(that);
  };
}

// gut
function() {
  var _this = this;
  return function() {
    console.log(_this);
  };
}
```

- Gib deinen Funktionen einen Namen. Dies ist hilfreich für den `stack trace`.

```
// schlecht
var log = function(msg) {
  console.log(msg);
};

// gut
var log = function log(msg) {
  console.log(msg);
};
```

[⬆]

Zugriffsmethoden

- Zugriffsmethoden für Objekteigenschaften sind nicht von Nöten.

- Macht man dennoch Zugriffsmethoden, benutze `getVal()` und `setVal('hello')`.

```
// schlecht
dragon.age();

// gut
dragon.getAge();

// schlecht
dragon.age(25);

// gut
dragon.setAge(25);
```

- Wenn die Eigenschaft ein Boolean ist, benutze `isVal()` oder `hasVal()`.

```
// schlecht
if (!dragon.age()) {
  return false;
}

// gut
if (!dragon.hasAge()) {
  return false;
}
```

- Es ist in Ordnung `get()` - und `set()` -Methoden zu erstellen, aber sei konsistent.

```
function Jedi(options) {
  options || (options = {});
  var lightsaber = options.lightsaber || 'blue';
  this.set('lightsaber', lightsaber);
}

Jedi.prototype.set = function(key, val) {
  this[key] = val;
};

Jedi.prototype.get = function(key) {
  return this[key];
};
```

[⬆]

Konstruktoren

- Weise die Methoden dem `prototype` des Objektes zu, anstelle den `prototype` mit einem neuen Objekt zu überschreiben. Wenn man den `prototype` überschreibt wird eine Vererbung unmöglich, denn damit wird die Basis überschrieben!

```
function Jedi() {
  console.log('new jedi');
}

// schlecht
Jedi.prototype = {
  fight: function fight() {
    console.log('fighting');
  }

  , block: function block() {
    console.log('blocking');
  }
};

// gut
Jedi.prototype.fight = function fight() {
  console.log('fighting');
```

```
};

Jedi.prototype.block = function block() {
  console.log('blocking');
};
```

- Methoden können `this` zurückgeben, um eine Methodenverkettung zu ermöglichen.

```
// schlecht
Jedi.prototype.jump = function() {
  this.jumping = true;
  return true;
};

Jedi.prototype.setHeight = function(height) {
  this.height = height;
};

var luke = new Jedi();
luke.jump(); // => true
luke.setHeight(20) // => undefined

// gut
Jedi.prototype.jump = function() {
  this.jumping = true;
  return this;
};

Jedi.prototype.setHeight = function(height) {
  this.height = height;
  return this;
};

var luke = new Jedi();

luke.jump()
  .setHeight(20);
```

- Es ist in Ordnung eine eigene `toString()` -Methode zu schreiben, aber man sollte sicherstellen, dass diese korrekt funktioniert und keine Nebeneffekte hat.

```
function Jedi(options) {
  options || (options = {});
  this.name = options.name || 'no name';
}

Jedi.prototype.getName = function getName() {
  return this.name;
};

Jedi.prototype.toString = function toString() {
  return 'Jedi - ' + this.getName();
};
```

[⬆]

Module

- Ein Modul sollte mit einem `!` beginnen. Dies stellt sicher, dass wenn in einem Modul das abschliessende Semikolon vergessen wurde, keine Fehler entstehen, wenn die Scripte zusammengeschnitten werden.
- Eine Datei sollte in `camelCase` benannt sein, in einem Ordner mit dem selben Namen liegen und dem Namen entsprechen mit dem es exportiert wird.
- Benutze eine Methode `noConflict()` , welche das exportierte Modul auf die vorhergehende Version setzt und diese zurück gibt.
- Deklariere immer `'use strict';` am Anfang des Moduls.

```
// fancyInput/fancyInput.js
```

```
!function(global) {
  'use strict';

  var previousFancyInput = global.FancyInput;

  function FancyInput(options) {
    this.options = options || {};
  }

  FancyInput.noConflict = function noConflict() {
    global.FancyInput = previousFancyInput;
    return FancyInput;
  };

  global.FancyInput = FancyInput;
}(this);
```

[⬆]

jQuery

- Stelle allen jQuery-Objektvariablen ein `$` voran.

```
// schlecht
var sidebar = $('.sidebar');

// gut
var $sidebar = $('.sidebar');
```

- Speichere `jQuery lookups`, sofern sie mehrmals gebraucht werden.

```
// schlecht
function setSidebar() {
  $('.sidebar').hide();

  // ...stuff...

  $('.sidebar').css({
    'background-color': 'pink'
  });
}

// gut
function setSidebar() {
  var $sidebar = $('.sidebar');
  $sidebar.hide();

  // ...stuff...

  $sidebar.css({
    'background-color': 'pink'
  });
}
```

- Für DOM-Abfragen benutze `Cascading` : `$('.sidebar ul')` oder parent > child `$('.sidebar > .ul')` . jsPerf
- Benutze `find` mit `scoped jQuery object queries`

```
// schlecht
$('.sidebar', 'ul').hide();

// schlecht
$('.sidebar').find('ul').hide();

// gut
$('.sidebar ul').hide();

// gut
```

```
$('.sidebar > ul').hide();

// gut (langsamer)
$sidebar.find('ul');

// gut (schneller)
$($sidebar[0]).find('ul');
```

[⬆]

ECMAScript 5 Kompatibilität

- Verweis auf Kangax's ES5 Kompatibilitätstabelle

[⬆]

Testing

- **Ja.**

```
function() {
  return true;
}
```

[⬆]

Performance

- On Layout & Web Performance
- String vs Array Concat
- Try/Catch Cost In a Loop
- Bang Function
- jQuery Find vs Context, Selector
- innerHTML vs textContent for script text
- Long String Concatenation
- Loading...

[⬆]

Ressourcen

Lese dieses

- Annotated ECMAScript 5.1

Andere Styleguides

- Google JavaScript Style Guide
- jQuery Core Style Guidelines
- Principles of Writing Consistent, Idiomatic JavaScript

Andere Styles

- Naming this in nested functions - Christian Johansen
- Conditional Callbacks

Bücher

- JavaScript: The Good Parts - Douglas Crockford
- JavaScript Patterns - Stoyan Stefanov
- Pro JavaScript Design Patterns - Ross Harmes and Dustin Diaz
- High Performance Web Sites: Essential Knowledge for Front-End Engineers - Steve Souders
- Maintainable JavaScript - Nicholas C. Zakas
- JavaScript Web Applications - Alex MacCaw

- Pro JavaScript Techniques - John Resig
- Smashing Node.js: JavaScript Everywhere - Guillermo Rauch

Blogs

- DailyJS
- JavaScript Weekly
- JavaScript, JavaScript...
- Bocoup Weblog
- Adequately Good
- NCZOnline
- Perfection Kills
- Ben Alman
- Dmitry Baranovskiy
- Dustin Diaz
- nettuts

[⬆]

In the Wild

Dies ist eine Liste von Organisationen, welche diesen Style Guide benutzen. Sende uns einen `Pull request` oder öffne einen `issue` und wir werden dich der Liste hinzufügen.

- **Airbnb**: airbnb/javascript
- **American Insitutes for Research**: AIRAST/javascript
- **ExactTarget**: ExactTarget/javascript
- **GoCardless**: gocardless/javascript
- **GoodData**: gooddata/gdc-js-style
- **How About We**: howaboutwe/javascript
- **MinnPost**: MinnPost/javascript
- **National Geographic**: natgeo/javascript
- **Razorfish**: razorfish/javascript-style-guide
- **Shutterfly**: shutterfly/javascript

[⬆]

Übersetzungen

Dieser Styleguide ist in den folgenden Sprachen erhältlich:

- :en: **Englisch**: airbnb/javascript
- :jp: **Japanisch**: mitsuruog/javacript-style-guide

[⬆]

The JavaScript Style Guide Guide

- Reference

[⬆]

Contributors

- View Contributors

[⬆]

Lizenz

(The MIT License)

[⬆]

```
};
```

Anhang J Aufgabenstellung

Die folgenden drei Seiten enthalten die offizielle Aufgabenstellung dieser Bachelorarbeit.

Aufgabenstellung Bachelorarbeit für Manuel Alabor, Alexandre Joly und Michael Weibel „Architekturkonzepte moderner Web-Applikationen“

1. Auftraggeber, Betreuer und Experte

Bei dieser Arbeit handelt es sich um eine HSR-interne Arbeit zur Unterstützung des Moduls Internettechnologien.

Auftraggeber/Betreuer:

- Prof. Hans Rudin, HSR, IFS hrudin@hsr.ch +41 55 222 49 36 (Verantw. Dozent, Betreuer)
- Kevin Gaunt, HSR,IFS kgaunt@hsr.ch +41 55 222 4662 (Betreuer)

Experte:

- Daniel Hiltebrand, Crealogix

2. Studierende

Diese Arbeit wird als Bachelorarbeit an der Abteilung Informatik durchgeführt von

- Manuel Alabor malabor@hsr.ch
- Alexandre Joly ajoly@hsr.ch
- Michael Weibel mweibel@hsr.ch

3. Ausgangslage

Das Modul Internettechnologien ist stark Technologie-zentriert. Wünschbar ist eine Weiterentwicklung (Aktualisierung, Verbesserung) mit vermehrter Beachtung von konzeptionellen und Architektur-Fragen. In letzter Zeit haben sich Prinzipien und Konzepte herauskristallisiert, nach denen Web-Applikationen am besten aufgebaut werden. Siehe zum Beispiel [1] oder [2]. Um diese Prinzipien und Konzepte anschaulich zu vermitteln, braucht es neben Erläuterungen möglichst anschauliche Beispiele und Übungsaufgaben. Ziel dieser Arbeit ist es, die Weiterentwicklung des Moduls Internettechnologien entsprechend zu unterstützen.

4. Aufgabenstellung

In dieser Arbeit sollen die in [1], [2] und weiteren Quellen dargestellten Prinzipien und Konzepte analysiert werden. Gemeinsam mit dem Betreuer sollen daraus in das Modul Internettechnologien zu transferierende Prinzipien und Konzepte ausgewählt werden, und es soll überlegt werden, wie diese Inhalte anschaulich für den Unterricht aufbereitet werden können. In der Folge sollten entsprechende Resultate erarbeitet werden, welche das Unterrichten der ausgewählten Inhalte möglichst gut unterstützen. Eine wichtige Rolle dürfte dabei eine anschauliche Beispielapplikation bilden.

Details werden im Verlauf der Arbeit zwischen Studierenden und Betreuer vereinbart.

5. Zur Durchführung

Mit dem Betreuer finden wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen sind von den Studierenden mit einer Traktandenliste vorzubereiten, die Besprechung ist durch die Studierenden zu leiten und die Ergebnisse sind in einem Protokoll festzuhalten, das den Betreuern und dem Auftraggeber per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

6. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html). Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Alle Resultate sind vollständig auf CD/DVD in 3 Exemplaren abzugeben. Der Bericht ist ausgedruckt in doppelter Ausführung abzugeben.

7. Referenzen

[1] Stefan Tilkov
Building large web-based systems: 10 Recommendations
Präsentation an der OOP 2013, München
PDF als Beilage

[2] http://roca-style.org
ROCA Resource-oriented Client Architecture - A collection of simple recommendations for decent Web application frontends

8. Termine

Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html.

Montag, den 18. Februar 2013	**Beginn der Bachelorarbeit,** **Ausgabe der Aufgabenstellung durch die Betreuer**
7. Juni 2013	Abgabe Kurzbeschreibung und A0-Poster. Vorlagen stehen unter den allgemeinen Infos Diplom-, Bachelor- und Studienarbeiten zur Verfügung.
14. Juni 2013, 12:00	**Abgabe** der Arbeit an den Betreuer bis 12.00 Uhr. Fertigstellung des A0-Posters bis 12.00 Uhr. Abgabe der Posters im Abteilungssekretariat 6.113.
14. Juni 2012	HSR-Forum, Vorträge und Präsentation der Bachelor- und Diplomarbeiten, 16 bis 20 Uhr
5.8. - 23.08.2013	Mündliche Prüfung zur Bachelorarbeit

9. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von 30 Stunden budgetiert (Siehe auch Modulbeschreibung der Bachelorarbeit https://unterricht.hsr.ch/staticWeb/allModules/19419_M_BAI.html.).

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/6
2. Berichte (Abstract, Mgmt Summary, technischer u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/6
3. Inhalt*)	3/6
4. Mündliche Prüfung zur Bachelorarbeit	1/6

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit mit den Studierenden festgelegt.

Im Übrigen gelten die Bestimmungen der Abteilung Informatik für Bachelorarbeiten.

Rapperswil, den 20. Februar 2013

Prof. Hans Rudin
Institut für Software
Hochschule für Technik Rapperswil

Anhang K **Meetingprotokolle**

Bachelorarbeit Vorbesprechung 14. Februar 2013

Teilnehmer

- Hans Rudin, HRU (HSR)
- Kevin Gaunt, KGA (HSR)
- Daniel Hiltebrand, DHI (Crealogix)
- Manuel Alabor, MAL (Team)
- Alexandre Joly, AJO (Team)
- Michael Weibel, MWE (Team, Protokoll)

Traktanden

- StoryboardBuilder - Was ist der aktuelle Stand?
- Oder was habt ihr für Ideen?

Meeting

StoryboardBuilder

Einführung DHI

- Ende Januar Entscheid: Nicht weiterentwickeln
- Allerdings nicht weil Produkt/Markt nicht interessant wäre
- Ziel war: damit die Crealogix UX-Services zu unterstützen
- Crealogix wird keine UX-Services gegen aussen mehr anbieten
    - mehr interne Projekte betreuen

- Zwei Schwerpunkte: Education & Financial Services

    - StoryboardBuilder gehört nicht in einen solchen Schwerpunkt

- Dies obwohl gutes Potential gesehen wird für das Produkt
- Anforderungsspezifikation verfeinert bis Ende Januar
- Technischer Prototyp gestartet, aber wieder gestoppt aufgrund der Neuorientierung
- Idee wäre: Storyboardbuilder an externe Firma weitergeben
- Bestehende Mitbewerber bauen ihre Angebote aus

Diskussion BA

- Frage an die Runde: ist es interessant für euch, den Storyboardbuilder in der BA weiter zuentwickeln?
    - MAL: Wie würde das aussehen?
        - DHI: Beispiel aufgrund gewählter Technologie zu entwickeln
        - DHI: Auf Basis der fachlichen Spezifikation der Crealogix
        - HRU: BA sollte nicht zu einer Fleissarbeit werden
        - HRU: Was wären denn die Herausforderungen wenn das weiterentwickelt werden würde in BA?
        - HRU: Die momentane Ausgangslage ist anders, da kein wirklicher Kunde existiert
        - DHI: Es sind sicher einige Ideen da, die technisch Herausfordernd sind
        - DHI: Grafische Repräsentation auf Screen, Objektmodell umsetzen
        - DHI: Wie gesagt, hat auch nichts dagegen, wenn eine neue Arbeit gemacht werden würde
        - DHI: Laut Kenntnisstand von DHI sollte auch von seiten Industriepartner weniger Betreuung beinhalten, mehr von Team
- KGA: Was ist nun der Anspruch an das Meeting? Müssen wir am Ende des Meetings schon wissen was gemacht werden soll?
    - DHI: Ist offen, hat keinen Anspruch auf Entscheid jetzt - sollte aber bald geschehen, da Bachelorarbeit bald startet
    - DHI:

- DHI: Hat div. Alternativen die man anschauen könnte
- MAL: für ihn ist das Durchführen der BA mit dem Storyboardbuilder nicht mehr besonders interessant, aufgrund Änderung seitens Crealogix
    - DHI: Ja das stimmt, BA würde nicht mehr zu einem realen Produkt führen

- MWE: gehts ähnlich wie MAL
- DHI: Thema 1:
    - **Kino reservations system**
    - Kennt Kinobesitzer in rapperswil
    - Buchungssystem
    - Mit anbindung an Kassensystem
    - nicht nur einzelne Plätze
    - sondern auch Firmenanlässe, Frauenkino, Catering etc.
    - Konzeptionell entwerfen
    - soweit wie möglich implementieren
    - könnte sehr interessant sein, DHI's Meinung nach

- DHI: Thema 2
    - **Personal Finance Management**
    - Zusatz zu Ebanking
    - Eigene Zahlungen analysieren
    - Verschiedene Banken
    - Soviel für Versicherung, Einkauf, etc.
    - Basiert auf einer isländisch-schwedischen Firma
    - Integrationsarbeit (in ein Bankensystem einbauen)
    - MAL: Geht es darum, das zu integrieren und anschaulich darzustellen, und Data-Mining ist schon gemacht?
    - DHI: Ja
        - Schwierigkeit Sicherheit
        - Zertifizierung, Authentifizierung
        - nur Teilaspekt möglich zum lösen
        - Versch. Komponenten
        - Aspekt wichtig auf welcher sich die BA konzentrieren soll

    - DHI: Müsste das genauer anschauen wie das machbar wäre
        - Da es ein sehr grosses System wäre
        - Müsste verifizieren ob das gehen soll?

- HRU: Thema 2 wohl eher interessant (Team bejaht)
  - HRU: Wäre das so kurzfristig machbar?
    - DHI: müsste angeschaut werden
    - DHI: Verträge bestehen
    - DHI: anderes ist Ebanking in Java mit Oracle
      - DHI: Verfügbar machen möglich
  - HRU: Zwei Komponenten, was ist Webfrontend?
    - DHI: Netty server von airlock für authentifizierung
    - DHI: möglichst HTML das übers web geht
    - DHI: J2EE - JSP vorne
  - HRU: Isländische Software
    - DHI: .NET basiert
    - DHI: hat aber ein WCF/REST/AJAX schnittstelle welche relativ gut ins Frontend integrierbar wäre
    - DHI: von eienm System ins andere System transferieren (Oracle zu MSSQL DB)
    - DHI: Statistisch aufwerten, und wieder anzeigen
    - DHI: machbarkeit unklar, muss verifiziert werden
- MAL: Was hat HRU für Projekte
  - MAL: Realtime sachen wären interessant (Mobile, Chat, Messaging?)
  - HRU: hat keine Projekte
- DHI: Hat evtl. noch andere Projekte, müsste das aber noch anschauen
- MAL: Bis Testumgebung steht würden wohl wochen vergehen
  - DHI: stimmt wohl
- MWE: Persönlich intressiert vorallem WebRTC
- HRU: was ist mit web realtimecommunication (WebRTC) gemeint?
- MWE: Near-realtime communication (Daten, Video, Audio) zw. Browsern
- MWE: Was wäre denn für Crealogix interessant - zentrale Frage?
  - DHI: Crealogix muss nicht unbedingt dabei sein, wenn nicht nötig
- DHI: update maus-scanner
  - MAL: wäre denn mobile auch ein Thema?
  - DHI: Mobile ist sehr zentral

- HRU: gibt es fraktionen (web/mobile) im Team?
  - MWE: Web-Mensch, aber Mobile wäre auch sehr interessant

- DHI: Fragt bei Crealogix CEO/entwicklungsleiter nach bzgl. Mobile
- MAL: Elearning wäre auch interessant bzlg. Mobile
  - DHI: könnte nachfragen obs da auch was geben würde?

- AJO: Auch vorallem Mobile interessant
  - arbeitet auch vorallem in Mobile
  - HRU: Auch sie MAL ;)

- DHI: müsste nachfragen, aber wäre sicher interessant
- MAL: Wäre sicher interessant auf DHI's Themen zu warten, andererseits müsste auch Teamintern bzw. mit KGA/HRU geschaut werden
- DHI: wann beginnt die Arbeit? - Montag
- DHI: 3 Bereiche Education
  - Campusmanagement
  - Time to learn
    - 30'000 die mit TTL arbeiten
    - Mit Center for young professional zusammenarbeit (evtl. da was interssantes)
    - In Bubikon

- DHI: was machen wenn nichts herauskommt?
  - DHI: würden gerne mit euch zusammenarbeiten, wenn möglich

- HRU: Wäre schon der Weg zum gehen
  - parallel müssten überlegungen angestellt werden obs was anderes geben würde
  - gibt den 3 Studierenden möglichst freie Hand

- KGA: Termin bis wann die Entscheidung möglich sein
  - HRU: allerspätistens erste Woche
  - DHI: wird noch heute mit den 3 Crealogix leuten schauen
  - DHI: Bis morgen, 15.02. Antwort wenn möglich
  - MAL: Mittwoch zu spät oder zu früh?
  - HRU: Wann sind sie @HSR?
    - MAL: MO/DI/MI

    - HRU: Mittwoch wäre nicht unbedingt zu spät
    - MAL: Mittwoch wäre Zusammenkunft, um definitiv zu entscheiden. Aber mit Kommunikation bis dann
    - MAL: DHI kann sicher schnell entscheiden, je nach dem wäre pers. Anwesenheit nicht nötig
    - DHI: würde gerne persönlich dabei sein

- DHI: gibt morgen Feedback
- HRU: das ist gut - in der Runde Team/HSR diskutieren was gemacht werden kann
- KGA: Was wird mit UX passieren? (Expert Talks)
    - DHI: Prio 1 hat interne Aufträge und Prio 2 externe

Diskussion im Team

- KGA: also ist keines der beiden Thema sehr interessant für Team?
- Team: Ja, insbesondere auch zu gross für die kurze Zeit (Thema 2)
- HRU: Reservationssystem vorallem Businessanalyse
- HRU: Sicher gut zu schauen was DHI einbringt
- HRU: Aber auch schauen was wir für Ideen haben
- KGA: was wäre ursprüngliche Idee für SA gewesen
- MWE: XMPP Server in node.js modular, flexibel bzgl. Datenbankanbindung
- MAL: und auch Skalierbarkeit von node.js interessant
- MAL: interessant wäre vielleicht frontend framework
- HRU: Alle miteinander auf dem Laufenden halten bzgl. Ideen

Nächstes Meeting

- Mittwoch, 20. Februar 2013, 10:10 Uhr