

Bachelorarbeit

Architekturkonzepte moderner Web-Applikationen

Hochschule für Technik Rapperswil
Frühjahrssemester 2013

Erstellt: 13. Juni 2013, 11:59

Autoren Manuel Alabor
Alexandre Joly
Michael Weibel

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

Abstract

Der Internetbrowser ist heute mehr denn je das Zuhause für komplexe Webapplikationen.

Erwartungsgemäss können traditionelle Architekturstile mit der neuen Technologie nicht mehr immer Lösungen für wiederkehrende Probleme bieten. Diese Bachelorarbeit analysiert Trends auf dem Gebiet der Webapplikationen kritisch und liefert gleichzeitig zukunftsweisende Vorschläge für das Unterrichtsmodul “Internettechnologien”.

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

Nach der ausführlichen Evaluation des Architekturstilkatalogs und der Auswahl von 22 Konzepten daraus entschied sich das Projektteam eine Aufgabenverwaltung für Wohngemeinschaften mit dem Namen “Roomies” umzusetzen. Der Fokus liegt so auf der Demonstration der Konzepte. Als Technologie wird server- als auch clientseitig JavaScript eingesetzt.

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

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 selbst und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurden.
- 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

Wir danken folgenden Personen für Ihre Unterstützung während der Bachelorarbeit:

- *Prof. Hans Rudin* für die Betreuung unserer Bachelorarbeit.
- *Kevin Gaunt* für viele interessante und hilfreiche Inputs während der gesamten Arbeit.
- Unseren Partnerinnen für Geduld, Rat und motivierende Worte.
- Unseren Korrekturlesern und Korrekturleserinnen.

Inhaltsverzeichnis

1.	Management Summary – SUBJECT TO CHANGE –	9
1.1.	Ausgangslage	9
1.2.	Vorgehen	9
1.3.	Ergebnisse	10
1.4.	Ausblick	10
2.	Analyse	11
2.1.	Architekturerrichtlinien	12
2.1.1.	Resource-oriented Client Architecture (ROCA)	13
2.1.2.	Building large web-based systems: 10 Recommendations	15
2.1.3.	Projektspezifische Richtlinien	16
2.1.4.	Richtliniendemonstration	17
2.2.	Produktentwicklung	18
2.3.	Technologieevaluation	19
2.3.1.	Bewertungskriterien & Gesamtbewertung	20
2.3.2.	Ruby	21
2.3.3.	Java	22
2.3.4.	JavaScript	24
2.3.5.	Entscheidung	26
2.4.	Proof of Concept	27
2.4.1.	Prototyp A: Sails.js	27
2.4.2.	Prototyp B: Express.js	29
3.	Technische Architektur	32
3.1.	Domainmodel	37
3.2.	Entity-Relationship Diagramm	39
3.3.	Software Layers	40
3.4.	Komponenten Diagramm	44
3.5.	Implementations-Sicht	45
3.5.1.	Rendering des User Interfaces und Event-Behandlung	45
3.5.2.	APIAdapter	46
3.5.3.	Quellcode Organisation	48
3.5.4.	Barefoot	49

3.5.5. JavaScript Callbacks	50
4. Richtliniendemonstration	53
4.1. Übersicht	54
4.2. RP1 REST	56
4.3. RP2 Application Logic	58
4.4. RP3 HTTP	60
4.5. RP4 Link	63
4.6. RP5 Non Browser	64
4.7. RP6 Should-Formats	65
4.8. RP7 Auth	66
4.9. RP8 Cookies	67
4.10. RP9 Session	69
4.11. RP10 Browser-Controls	70
4.12. RP11 POSH	71
4.13. RP12 Accessibility	73
4.14. RP13 Progressive Enhancement	74
4.15. RP14 Unobtrusive JavaScript	76
4.16. RP15 No Duplication	80
4.17. RP16 Know Structure	82
4.18. RP17 Static Assets	83
4.19. RP18 History API	86
4.20. TP3 Eat your own API dog food	87
4.21. TP4 Separate user identity, sign-up and self-care from product dependencies	88
4.22. TP7 Apply the Web instead of working around	92
4.23. TP8 Automate everything or you will be hurt	97
5. Schlussfolgerung	100
6. Persönliche Berichte	103
6.1. Manuel Alabor	103
6.2. Alexandre Joly	103
6.3. Michael Weibel	103
A. Abbildungen, Tabellen & Quelltexte	105
B. Glossar	110
C. Literatur	115
D. Projektrelevante URL's	126
E. Installationsanleitung	127
E.1. Mit Vagrant	127
E.2. Ohne Vagrant	127

F.	Technologie Einführung	129
F.1.	JavaScript	129
F.2.	Node.js	129
F.3.	Einführung für benutzte Libraries	130
G.	Technologieevaluation	131
H.	Produktentwicklung	134
H.1.	Workshop	134
H.2.	Branding	135
H.3.	Mindmap	136
I.	Anforderungsanalyse	138
I.1.	Funktionale Anforderungen	138
I.2.	Nichtfunktionale Anforderungen	139
I.3.	Use Cases	140
I.3.1.	Akteure	141
I.3.2.	UC1: Anmelden	141
I.3.3.	UC2: WG erstellen	141
I.3.4.	UC3: WG beitreten	142
I.3.5.	UC4: WG verlassen	142
I.3.6.	UC5: Aufgabe erstellen	142
I.3.7.	UC6: Aufgabe bearbeiten	143
I.3.8.	UC7: Aufgabe erledigen	143
I.3.9.	UC8: Rangliste anzeigen	143
I.3.10.	UC9: WG auflösen	144
I.3.11.	UC10: Benutzer verwalten	144
I.3.12.	UC11: auf Social Media Plattform teilen	144
I.4.	Umgesetzte Use Cases	145
J.	Projektplanung	146
J.1.	Iterationsplanung	146
J.2.	Meilensteine	148
J.3.	Artefakte	149
J.4.	Meetings	149
J.4.1.	Regelmässiges Statusmeeting	149
J.5.	Tools	150
J.5.1.	Projektverwaltung	150
J.5.2.	Entwicklungsumgebung	150
J.5.3.	Git Repositories	150
J.5.4.	Continuous Integration	150
J.6.	Qualitätsmanagement	151
J.6.1.	Baselining	151
J.6.2.	Testing	151

J.6.3. Continuous Integration	152
J.6.4. Coding Style Guideline	152
J.6.5. Code Reviews	152
K. Iterationsassesment	153
L. Auswertung Zeitrapportierung	154
L.1. Aufwand pro Person und Woche	154
L.2. Burndown	155
L.3. Zeitaufwand pro Aufgabenbereich	156
L.4. Zeitaufwand pro Iteration	157
M. Screenshot Tour: Beispielapplikation Roomies	158
N. Coding Style Guideline	162
O. Aufgabenstellung	164
P. Meetingprotokolle	168

Kapitel 1 **Management Summary -- SUBJECT TO CHANGE**

1.1. Ausgangslage

Der Internetbrowser ist heute mehr denn je das Zuhause für immer komplexer werdende Applikationen. Durch die Leistungssteigerung und immer ausgefeilteren Möglichkeiten nutzen ihn heutzutage nicht mehr nur Unternehmen 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 Browersetchnologie 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? Welche 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 sicher gestellt, 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



Abbildung 1.1.: Aufgabenverwaltung *Roomies* zur Konzeptdemonstration

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 bewerten 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

Aufgabenstellung

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

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

1. Die Beispielapplikation 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 sowohl interessierten Fachpersonen 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 Idee 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 O) 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.

Welche Richtlinien können auf der Beispielapplikation am einfachsten und effektivsten demonstriert werden? Diese Frage wird abschliessend aufgezeigt und auf die entsprechenden Stellen der Beispielapplikation hingewiesen.

2.1.1. Resource-oriented Client Architecture (ROCA)

Backend-Layer

ID	Prinzip	Erläuterung
<i>RP1</i>	REST	Die Kommunikation mit den Serverressourcen folgt dem REST-Prinzip [Fie00]
<i>RP2</i>	Application Logic	Die Businesslogik der Applikation soll im Backend bleiben.
<i>RP3</i>	HTTP	Ergänzend zu <i>RP1</i> findet die Kommunikation mit den Serverressourcen über wohldefinierte RESTful HTTP Requests [Irv+] statt
<i>RP4</i>	Link	Alle URI's weisen zu einer eindeutigen Ressource.
<i>RP5</i>	Non-Browser	Die Serverkomponente kann ohne Browser resp. Frontendkomponente (z.B. mit <i>wget</i> [Freb] oder <i>cURL</i> [Hax]) verwendet werden.
<i>RP6</i>	Should-Formats	Serverressourcen können ihre Daten in verschiedenen Formaten (JSON, XML) ausliefern.
<i>RP7</i>	Auth	<i>HTTP Basic Authentication over SSL</i> [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.
<i>RP8</i>	Cookies	Cookies werden nur zur Authentifizierung oder zum Tracking des Benutzers verwendet.
<i>RP9</i>	Session	Eine Webapplikation soll im Grossen und Ganzen zustandslos sein. Ausnahme bildet bspw. die Authentifizierung eines Benutzers.
<i>RP10</i>	Browser-Controls	Die Verwendung von Browser-Steuerelementen (Zurück, Aktualisieren usw.) müssen wie erwartet funktionieren und die Applikation nicht unerwartet beeinflussen.

Tabelle 2.1.: Die ROCA Architekturprinzipien: Backend

Frontend-Layer

ID	Prinzip	Erläuterung
<i>RP11</i>	POSH	Vom Backend generiertes HTML Markup ist semantisch korrekt [Pilb] und ist frei von Darstellungsinformationen
<i>RP12</i>	Accessibility	Alle Ansichten können von Accessibility Tools (z.B. Screen Reader für Sehbehinderte, siehe u.A. [Pic]) verarbeitet werden.
<i>RP13</i>	Progressive Enhancement	Die Darstellung des Frontends soll auf aktuellsten Browsern top aussehen, aber auch auf älteren mit weniger Features verwendbar sein.
<i>RP14</i>	Unobtrusive JavaScript	Die grundlegenden Funktionalitäten des Frontends müssen auch ohne JavaScript verwendbar sein.
<i>RP15</i>	No Duplication	Eine Duplizierung von Businesslogik auf dem Frontend-Layer soll vermieden werden (vgl. <i>RP2</i>)
<i>RP16</i>	Know Structure	Der Backendlayer soll so wenig wie möglich über die finale Struktur des HTML-Markups auf dem Frontend "kennen".
<i>RP17</i>	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.
<i>RP18</i>	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 der Richtlinien "RP13" und "RP14" 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 gehört “RP14” tendenziell zu einer unumgänglichen Richtlinie

2.1.2. Building large web-based systems: 10 Recommendations

ID	Empfehlung
<i>TP1</i>	Aim for a web of loosely coupled, autonomous systems.
<i>TP2</i>	Avoid session state wherever possible.
<i>TP3</i>	Eat your own API dog food.
<i>TP4</i>	Separate user identity, sign-up and self-care from product dependencies.
<i>TP5</i>	Pick the low-hanging fruit of front-end performance optimizations.
<i>TP6</i>	Don't bother readers with write complexity.
<i>TP7</i>	Apply the Web instead of working around it.
<i>TP8</i>	Automate everything or you will be hurt.
<i>TP9</i>	Know, design for & use web components
<i>TP10</i>	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 jede interne API-Methode 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 denn 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 jede Richtlinie an mindestens einer Systemkomponenten demonstriert werden kann.

	<i>Backend</i>	<i>Frontend</i>	<i>Tools</i>
	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 H “Produktentwicklung” weitere Details zum Prozess der konkreten Produktentwicklung. An dieser Stelle soll jedoch der Fokus auf der Grundlegenden Idee liegen.

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 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.

Logo & Branding



Abbildung 2.2.: Roomies Logo im College Stil

Im Anhang H sind zum Thema *Branding* Informationen wie Styleguide, Logovarianten etc. 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 sowohl in der Open Source- wie auch der Industrie-Community grösster Beliebtheit.

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

2.3.1. Bewertungskriterien & Gesamtbewertung

Die folgende Tabelle definiert 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. <i>Hohe Bewertung = Wenig Eigenkonzepte</i>	★★★
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? <i>Hohe Bewertung = Hohe Eignung</i>	★★★
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? <i>Hohe Bewertung = Hohe Produktreife</i>	★★★
TK4	Aktualität	Diese Arbeit befasst sich um "moderne Web-Applikationen". So sollte auch die zu verwendende Technologie gewissermassen nicht von "vorgestern" sein. <i>Hohe Bewertung = Hohe Aktualität</i>	★
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? <i>Hohe Bewertung = Hoher "Ease of use"-Faktor</i>	★★
TK6	Testbarkeit	Wie gut können die mit dem Framework oder der Technologie erstellte Komponenten durch Unit Tests getestet werden? <i>Hohe Bewertung = Hohe Testbarkeit</i>	★★

Tabelle 2.4.: Bewertungskriterien für Technologieevaluation

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

$$\frac{\sum_{n=1}^6 \text{Bewertung}_{TK_n} \times \text{Gewichtung}_{TK_n}}{6}$$

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 “leiserlichere” Syntax als man sich das beispielsweise von Java oder anderen verwandten Sprachen gewohnt ist. Folgende Codebeispiele bewirken dieselbe 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

	<i>TK1 Eigenkonzepte</i>	<i>TK2 Eignung</i>	<i>TK3 Produktreihe</i>	<i>TK4 Aktualität</i>	<i>TK5 "Ease of use"</i>	<i>TK6 Testbarkeit</i>	<i>Gesamtbewertung</i>
<i>Ruby on Rails</i>	★	★	★★★	★	★★★	★★	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 optimal wenn es darum geht, Architekturrichtlinien eindeutig und klar demonstrieren zu können.

Dies vor allem 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 Berücksichtigung 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 Servicelayer 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
<i>Studienarbeit-Zusammenstellung</i>	Die Zusammenstellung von <i>Google Guice</i> , <i>Jersey</i> , <i>Codehaus Jackson</i> sowie <i>EclipseLink</i> hat sehr gut harmoniert. Die Verwendung von einem Java-fremden Framework für die Implementierung des Frontends wäre jedoch erneut abzuklären.
<i>Spring [Incb]</i>	Spring hat sich in den letzten Jahren in der Industrie etablieren können. Es bietet eine Vielzahl von Subkomponenten (MVC, Beanmapping etc.).

Tabelle 2.6.: Shortlist Analysekandidaten Java (1/2)

Framework	Erläuterung
Plain JEE [Oraa]	Java Enterprise bietet von sich aus viele Features, welche die Frameworks von Dritten unter anderen Ansätzen umsetzen. Es gilt jedoch abzuwegen, wie gross der Aufwand ist, um beispielsweise eine REST-Serviceschnittstelle zu implementieren.
Vaadin [Ltd]	Vaadin baut auf Googles GWT [Goob] und erlaubt die serverzentrierte Entwicklung von Webapplikationen.
Play! Framework [Typ]	Seit dem Release der Version 2.0 im Frühjahr 2012 erfreut sich das Play! Frameworks grosser Beliebtheit. Insbesondere die integrierten Scaffolding-Funktionalitäten und MVC-Ansätze werden gelobt.

Tabelle 2.7.: Shortlist Analysekandidaten Java (2/2)

Bewertungsmatrix

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

Tabelle 2.8.: 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 in dieser Kombination 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 [Good] 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 [Good] von Google hat 2009 zudem den Sprung weg vom Browser geschafft: Node.js [Joyb] bietet die Engine als eigenständigen Skriptinterpretierer an und verfügt über eine umfangreiche, leistungsstarke und systemnahe API. Die neu entstandene Plattform erfreut sich in der Open Source Community grösster Beliebtheit. So sind seit der ersten Veröffentlichung eine Vielzahl hochwertiger Frameworks und Bibliotheken zu den verschiedensten Themengebieten entstanden.

Im Bereich der *Webframeworks* werden im Bezug auf diese Bachelorarbeit folgende Kandidaten genauer analysiert:

Framework	Erläuterung
Express.js	Express.js [Holf] baut auf dem Basisframework Connect [Senb] auf. Das damit durchgängig umgesetzte Middleware-Konzept ermöglicht die entkoppelte Entwicklung von simplen Webservern bis zu 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 <i>Unobtrusive JavaScripts</i> genügen mögen: Derby-Applikationen werden komplett im Browser gerendert und bieten keine Möglichkeit, HTML-Markup auf dem Server zu generieren.

Tabelle 2.9.: Shortlist Analysekandidaten JavaScript (1/2)

Framework	Erläuterung
<i>Geddy</i>	Geddy [Ged] ist ein weiterer 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.
<i>Sails</i>	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 2.10.: Shortlist Analysekandidaten JavaScript (2/2)

Bewertungsmatrix

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

Tabelle 2.11.: Bewertungsmatrix JavaScript Frameworks

Interpretation

Die Kandidaten lassen sich grob in zwei Kategorien 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 die Evaluation in der Kategorie JavaScript.

2.3.5. Entscheidung

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

	<i>TK1 Eigenkonzepte</i>	<i>TK2 Eignung</i>	<i>TK3 Produktreife</i>	<i>TK4 Aktualität</i>	<i>TK5 "Ease of use"</i>	<i>TK6 Testbarkeit</i>
<i>Ruby: Ruby on Rails</i>	★	★	★★★	★	★★★	★★
<i>Java: Studienarbeit-Zusammenstellung</i>	★★★	★★★		★★★	★★	
<i>Express.js</i>	★★★	★★★	★★	★★★	★★	★★★

Tabelle 2.12.: Finale Frameworkkandidaten für Technologieentscheidung

Im Entscheidungsmeeting vom 6. März (siehe Anhang P “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.

Die Betrachtung der verbleibenden Kandidaten verdeutlicht dass *Express.js* in den Kategorien *TK1 Eigenkonzepte*, *TK2 Eignung* und *TK4 Aktualität* weit vor *Ruby on Rails* liegt.

Zusätzlich war die Neugier auf eine unverbrauchte Technologie und die Herausforderung, etwas neues auszuprobieren sowohl beim Betreuer als auch beim Projektteam gross.

Aus diesen Gründen geht *JavaScript* mit dem Framework *Express.js* als Gewinner aus der Technologieevaluation hervor.

2.4. Proof of Concept

Der vorangegangene Abschnitt zur Technologieevaluation arbeitet das *JavaScript Framework Express.js* als Umsetzungstechnologie für die Beispielapplikation aus. Um diese Entscheidung fundiert bestätigen zu können hat sich das Projektteam für den Proof of Concept entschieden, sowohl einen Prototypen mit *Express.js* als auch einem zweiten *JavaScript Framework* zu erstellen.

Aufgrund der interessanten Ansätze im Bereich Real-Time wurde hierfür das vorgestellte *Sails.js* ausgewählt.

Wie der folgende Abschnitt “Proof of Concept” zeigt, wird sich das Ergebnis der Technologieevaluation bestätigen: *Express.js* ist die richtige Wahl für die Umsetzung der Beispielapplikation.

2.4.1. Prototyp A: Sails.js

Sails.js verwendet Scaffolding um einerseits ein neues Projekt zu erstellen, andererseits um verschiedene Komponenten wie Models oder Controllers zu erstellen. Wie im Entity-Relationship Diagramm (siehe Kapitel 3, Abschnitt 3.2) beschrieben, werden u.a. ein Task und ein Resident Model (mit entsprechenden Tabellen) benötigt.

Um das ORM “Waterline” [Balc] zu testen, wurden diese beiden Models implementiert. Das Task-Model sieht so aus wie im Quelltext 2.3 gezeigt.

```
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 der 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, Models direkt über einen 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.

```

1  function findTaskAndUser(id, next) {
2    Task.find(id).done(function(err, task) {
3      User.find(task.userId).done(function(err, user) {
4        next(task, user);
5      });
6    });
7  }
8
9  function renderResponse(req, res, response) {
10   if (req.acceptJson) {
11     res.json(response);
12   } else if(req.isAjax && req.param('partial')) {
13     response['layout'] = false;
14     res.view(response);
15   } else {
16     res.view(response);
17   }
18 }
19
20 var TaskController = {
21   get: function(req, res) {
22     var id = req.param('id');
23     findTaskAndUser(id, function(task, user) {
24       var response = {
25         'task': task,
26         'user': user,
27         'title': task.name
28       };
29       renderResponse(req, res, response);
30     });
31   }
32 };
33 module.exports = TaskController;

```

Quelltext 2.4: Task Controller in Sails.js

In diesem Controller wird ein Task aufgrund des GET-Parameters “id” (Linie 22) geladen. Das Code-Stück zeigt die grosse Schwäche des ORMs Waterline [Balc]. In anderem ORMs könnte man auf dem “task”-Objekt direkt “.user” aufrufen. Dies geht bei Waterline nicht und man muss den Umweg über “User.find()” (Zeile 3) gehen.

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

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

```

1 <div id="task-display">
2   <h1h1>
3   <ullili>
5     <lili>
6   </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 Skript am Ende des Task Templates wird aufgezeigt, wie man ohne Neuladen der Seite direkt das DOM ersetzen kann. Dies ist aber Framework-unabhängig.

Schlussfolgerung

Wie bereits 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. Die Community war zum Zeitpunkt der Evaluation (siehe Anhang G zur Technologieevaluation) sehr klein. Die Fakten deuten darauf hin, dass *Sails.js* für die konkrete Beispielapplikation eher ungeeignet ist.

2.4.2. Prototyp B: Express.js

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

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

Ein Beispielhafter Controller ist im Quelltext 2.6 zu sehen.

```

1 exports.index = function(req, res){
2   // first Parameter: Template File to use
3   // 2nd Parameter: Context to pass to the template
4   res.render('index', { title: 'Express' });

```

```
5 };
```

Quelltext 2.6: Beispiel eines Controllers in Express.js

Ein zugehöriges Template kann folgendermassen aussehen:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title><%= title %></title>
5     <link rel='stylesheet' href='/stylesheets/style.css' />
6     <%- LRScript %>
7   </head>
8   <body>
9     <h1><%= title %></h1>
10    <p>Welcome to <%= title %></p>
11  </body>
12 </html>
```

Quelltext 2.7: Template in Express.js

In den vorangegangenen zwei Quelltexten 2.6 und 2.7 ist ersichtlich, dass der Applikationsentwickler sehr grosse Kontrolle über Express.js hat.

Die Flexibilität von Express.js bietet sowohl Vor- als auch Nachteile für die erstellung von Webapplikationen. Im Bezug auf die Veranschaulichung der Architekturrichtlinien ist es jedoch ein grosser Vorteil, da wenig Logik fix in Express.js eingebaut ist.

ORM

Im Gegensatz zu den anderen evaluierten Frameworks ist in Express.js kein ORM enthalten. Aus diesem Grund wurde in einer weiteren Evaluation drei ORMs anhand der Kriterien in Tabelle 2.13 bewertet.

ID	Kriterium	Erläuterung	Gewichtung
<i>OK1</i>	Unterstützte DBs	Wieviele unterschiedliche Datenbanken unterstützt das ORM? Werden auch NoSQL-Datenbanken unterstützt? <i>Hohe Bewertung = Grosse Anzahl an Datenbanken</i>	★
<i>OK2</i>	Relationen	Sind Relationen zwischen Tabellen definierbar? Verwenden diese die datenbankspezifischen Foreign Keys dafür (falls möglich)? <i>Hohe Bewertung = Relationen möglich und verwendet Datenbank-spezifische Datentypen</i>	★★★
<i>OK3</i>	Produktreife	Wie gut hat sich das ORM bis jetzt in der Realität bewiesen können? Wie lange existiert es schon? Gibt es eine aktive Community und wird es aktiv weiterentwickelt? <i>Hohe Bewertung = Hohe Produktreife</i>	★★★
<i>OK4</i>	“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? <i>Hohe Bewertung = Hoher “Ease of use”-Faktor</i>	★
<i>OK5</i>	Testbarkeit	Wie gut können die mit dem Framework oder der Technologie erstellte Komponenten durch Unit Tests getestet werden? <i>Hohe Bewertung = Hohe Testbarkeit</i>	★★

Tabelle 2.13.: Bewertungskriterien für ORM-Evaluation

	<i>OK1 Unterstützung DBs</i>	<i>OK2 Relationen</i>	<i>OK3 Produktreife</i>	<i>OK4 “Ease of use”</i>	<i>OK5 Testbarkeit</i>	<i>Total</i>
<i>JugglingDB</i>	★★★	★	★	★★	★★	3
<i>Node-ORM2</i>	★★	★★	★	★★★	★	3
<i>Sequelize</i>	★	★★	★★	★★	★	3

Tabelle 2.14.: Bewertungsmatrix JavaScript ORMs

Alle verglichenen ORMs haben eine ähnliche Gesamtbewertung. Bei “Sequelize” stehen jedoch die Produktreife und die Unterstützung für Relationen heraus.

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

Kapitel 3 Technische Architektur

Einleitung

Die Architektur besteht aus folgenden zwei Haupt-Komponenten:

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

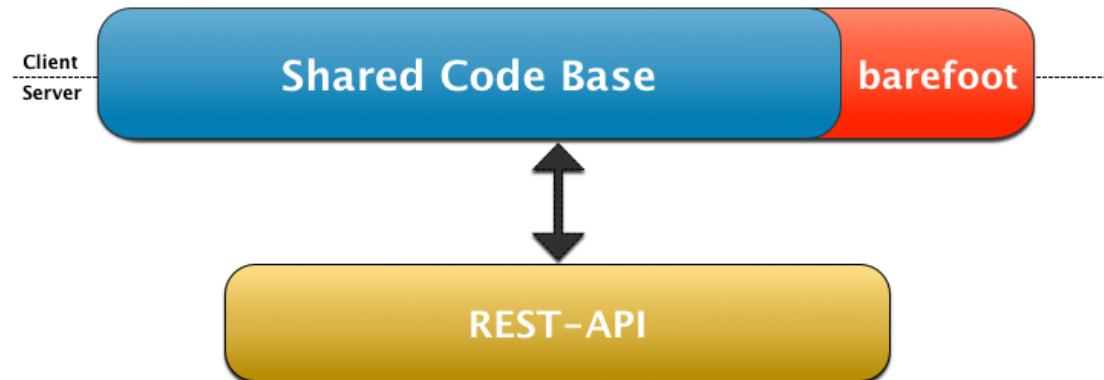


Abbildung 3.1.: Architektur

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 die MVC-Komponenten zu benutzen und setzt auf Backbone.js [Doca] auf.

Der Router dient hier als Beispiel:

```
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 3.1: Auszug aus Router der Beispielapplikation [AJWq]

Der Router aus Quelltext 3.1 wird sowohl vom Client als auch vom Server verwendet und registriert URLs mit den entsprechenden Funktionen.

Die "HomeView" ist in Quelltext 3.2 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 3.2: Ausschnitt aus HomeView der Beispielapplikation [AJWu]

API-Komponente

Auf Server-Seite ist 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 des Servers oder des Clients. Ruft der Client eine API auf, wird ein AJAX-Aufruf gemacht. Wird die API hingegen vom Server aus aufgerufen, werden direkt die entsprechend definierten Callbacks für die gewünschte Route lokal aufgerufen.

Quelltext 3.3 zeigt ein Beispiel für einen solchen Callback, während Quelltext 3.4 ein Beispiel für eine API-Route aufzeigt.

```

294 /**
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 3.3: API-Controller Beispiel [AJWg]

```
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 3.4: API-Route Beispiel [AJWi]

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

- Model ist ein traditionelles Model welches zwischen Server & Client geteilt wird
- View ist eine “Barefoot.View” [Alag] und rendert Templates
- Controller ist einerseits ein “Route”-Controller, welcher aufgrund von URLs die richtige View aufruft und andererseits ein API-Controller, welcher die REST-API-Logik kapselt

Diagramm 3.2 zeigt eine grobe Übersicht über das Zusammenspiel der verschiedenen Komponenten.



Abbildung 3.2.: MVC-Komponenten im Zusammenspiel

3.1. Domainmodel

Das Domainmodel in Abbildung 3.3 zeigt eine Übersicht über alle in der Problemdomäne enthaltenen Objekte. Zu jedem dieser ist im Anschluss eine genauere Erklärung zu finden.



Abbildung 3.3.: Domainmodel

Achievement (Erfolg)

Achievements werden an einen Resident vergeben, sobald dieser bestimmte Regeln ausreichend befriedigt hat.

Rule (Regel)

Rules beschreiben, welche Aktionen oder Verhalten notwendig sind, damit ein Resident ein bestimmtes Achievement 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 Achievement 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 einer Community sowie derer Tasks und Residents erhalten.

Task (Aufgabe)

Eine Community führt eine Liste von zu erledigenden 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.

3.2. Entity-Relationship Diagramm

Das ER-Diagramm in Abbildung 3.4 repräsentiert die Abbildung des Domainmodels auf Datenbankebene.

Für beide Modelle ist dieselbe Terminologie gültig.



Abbildung 3.4.: Entity-Relationship Diagramm

3.3. 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.

Technologieneutral



Abbildung 3.5.: Software Layers – Technologieneutral

Shared & Shared MVC

Die “Shared”-Schicht beinhaltet jeglichen Quelltext, welcher sowohl auf Client als auch auf Server-Seite instanziert wird. Der Hauptteil auf diesem Layer ist strukturiert anhand des *MVC*-Patterns

Templating Engine

Das Rendern der Views wird mithilfe einer Templating Engine vereinfacht.

API

Es existiert auf dem Server eine entkoppelte Service-Komponente welche von der “Shared”-Schicht für das Manipulieren von Daten verwendet wird.

Entities

Als Entities werden die ORM-Models bezeichnet, welche Datenbanktabellen abstrahieren und dadurch die Manipulation dieser ermöglichen.

ORM

Die ORM-Komponente abstrahiert die Datenbank-Verbindung sowie jegliche Interaktion mit der Datenbank.

Middleware & Auth

Die Middleware ist ein Konzept des verwendeten “Web Frameworks” und ermöglicht das Manipulieren von HTTP Requests und Responses unter Verwendung des *Filter*-Patterns.

Die Middleware-Schicht beinhaltet mehrere Middlewares. “Auth” ist dabei eine konkrete Implementation welche sich um die Authentifizierung von Benutzern resp.. der Kommunikation mit diesen kümmert.

Web Framework

Das Web-Framework dient zur Verarbeitung und Aufbereitung von HTTP Requests und Responses.

Validation

Die Validations-Schicht besteht aus mehreren Validatoren welche die datenbasierte Validierung übernehmen. Als datenbasierte Validierung bezeichnet man die Überprüfung von Informationen anhand deren Datentyp sowie formalen Korrektheit.

Code Sharing Framework

Während der Bachelorarbeit wurde ein “Code Sharing Framework” entwickelt, welches die aufgezeigte Architektur erst ermöglicht hat. Es erlaubt das Instanzieren des gleichen Quelltextes auf Server- wie auch auf Client-Seite.

Logging

Das Code Sharing Framework loggt Request und Fehler mithilfe einer Logging-Komponente.

MVC Framework

Das MVC Framework wird vom Code Sharing Framework verwendet, und bietet Basisklassen entsprechende Basisklassen wie *View*, *Model* und *Controller* dafür.

Konkrete Implementation

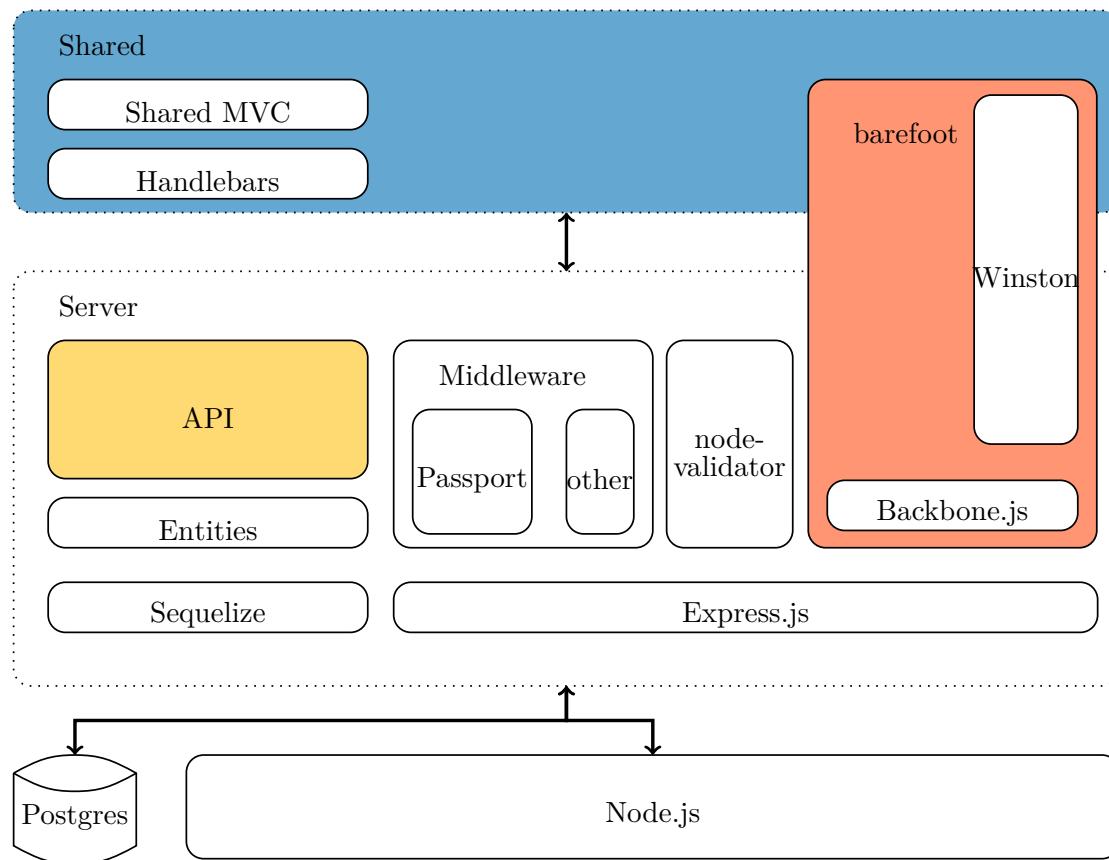


Abbildung 3.6.: Software Layers – Konkrete Implementation

Shared MVC

Das Shared MVC ist auf “barefoot” [Alaa] aufgebaut, welches wiederum Backbone.js [Doca] verwendet.

Handlebars

Die View des Shared MVC verwendet das präziserprobte “Handlebars” [Kat] für das Rendering effektiven HTML Markups.

Sequelize

Das aus Kapitel 2.3 Technologieevaluation bekannte Sequelize [Depb] wird als ORM eingesetzt und dient zur Abstraktion des Datenbanksystems.

Passport

Passport [Hanb] ist ein Authentisierungs-Framework für Node.js. Es bietet eine Vielzahl verschiedener Strategien (siehe auch Strategy-Pattern [Gam+94]) für die Authentifizierung über externe Identiy Provider (Facebook Login for Web [Faca] etc.).

Express.js

Express.js [Holf] ist ein erprobtes Web Framework für Node.js und wurde in der Technologieevaluation bereits näher erläutert.

node-validator

Eingehende Schnittstellenaufrufe werde mittels “node-validator” [OHa] auf vorher definierte Bedingungen überprüft und ggf. bereinigt.

barefoot

“barefoot” [Alaa] für das Code Sharing Framework zwischen Client und Server zuständig. Es wurde während dieser Bachelorarbeit entwickelt.

Winston

Winston [Fla] ist ein einfach konfigurierbares Logging-Framework für Node.js. Es wird von barefoot eingesetzt.

Backbone.js

“Backbone.js” [Doca] ist das MVC Framework und bietet die grundlegende Klassen welche “barefooot” kapselt und/oder erweitert:

- Barefoot.Events
- Barefoot.Model
- Barefoot.Collection
- Barefoot.Router
- Barefiit.View

3.4. Komponenten Diagramm

Das Diagramm 3.7 zeigt alle logischen Komponenten der Beispielapplikation.

Sie widerspiegelt sich zudem in der im Abschnitt 3.5 einsehbaren physischen Datei- und Ordnerstruktur des Quelltextes.

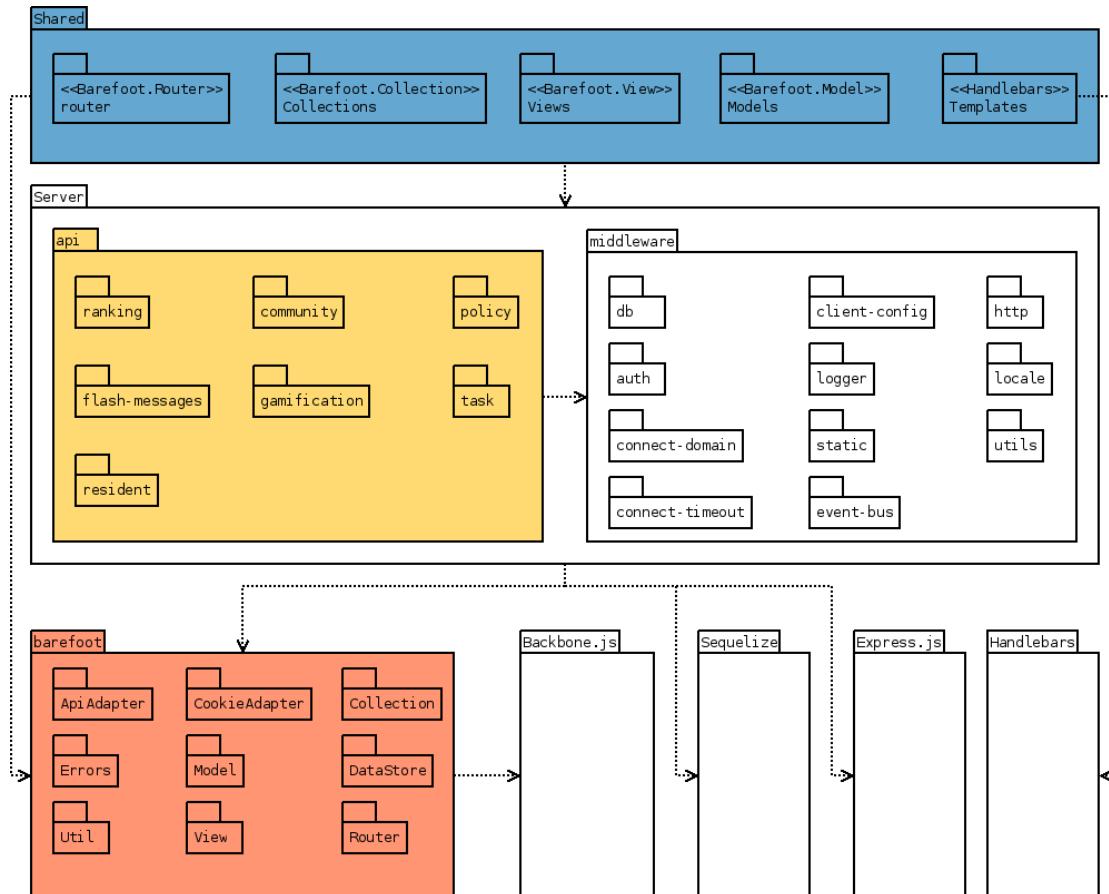


Abbildung 3.7.: “Roomies”: Logische Komponenten

3.5. Implementations-Sicht

3.5.1. 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. Diagramm 3.8 gerendert. Sollte der Client-Browser zudem JavaScript deaktiviert haben oder nicht unterstützen, so werden auch nachfolgende Requests nach dem gleichen Schema verarbeitet und gerendert.

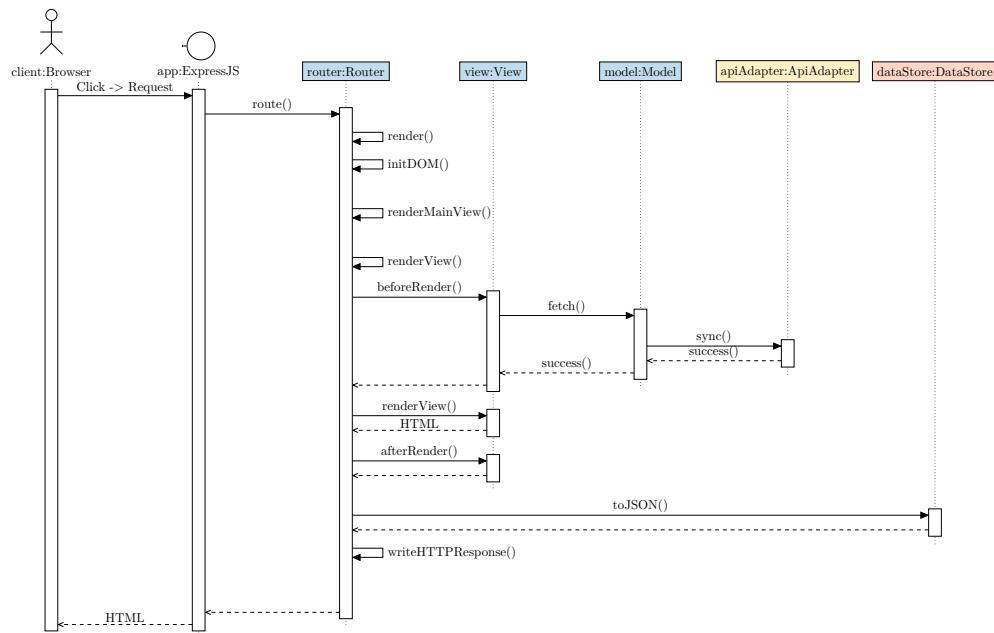


Abbildung 3.8.: Sequenzdiagramm: Rendering und Event-Verarbeitung auf dem Server

Client

Das Sequenzdiagramm 3.9 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.

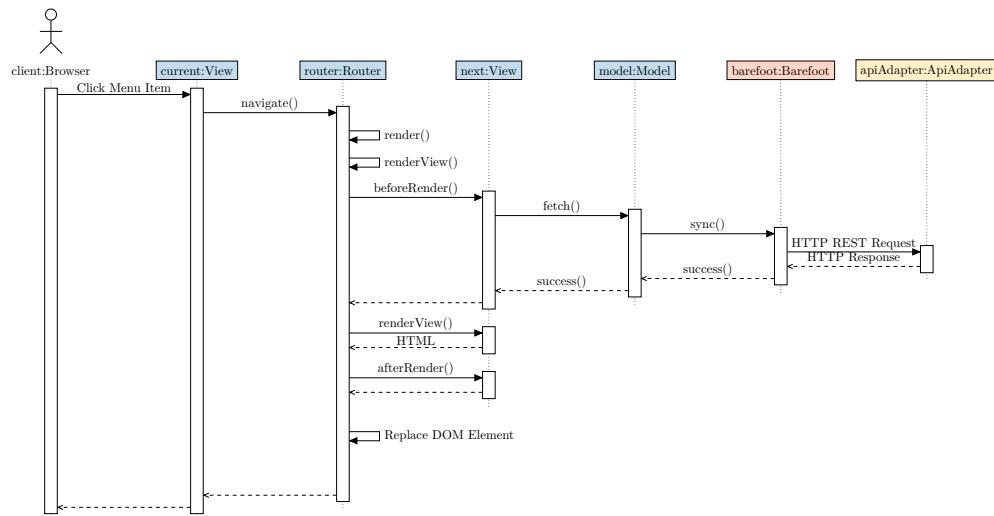


Abbildung 3.9.: Sequenzdiagramm: Rendering und Event-Verarbeitung auf dem Client

3.5.2. APIAdapter

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

Das Diagramm 3.10 verdeutlicht die Abläufe im innern des Adapters für den jeweiligen Anfragemodus.

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

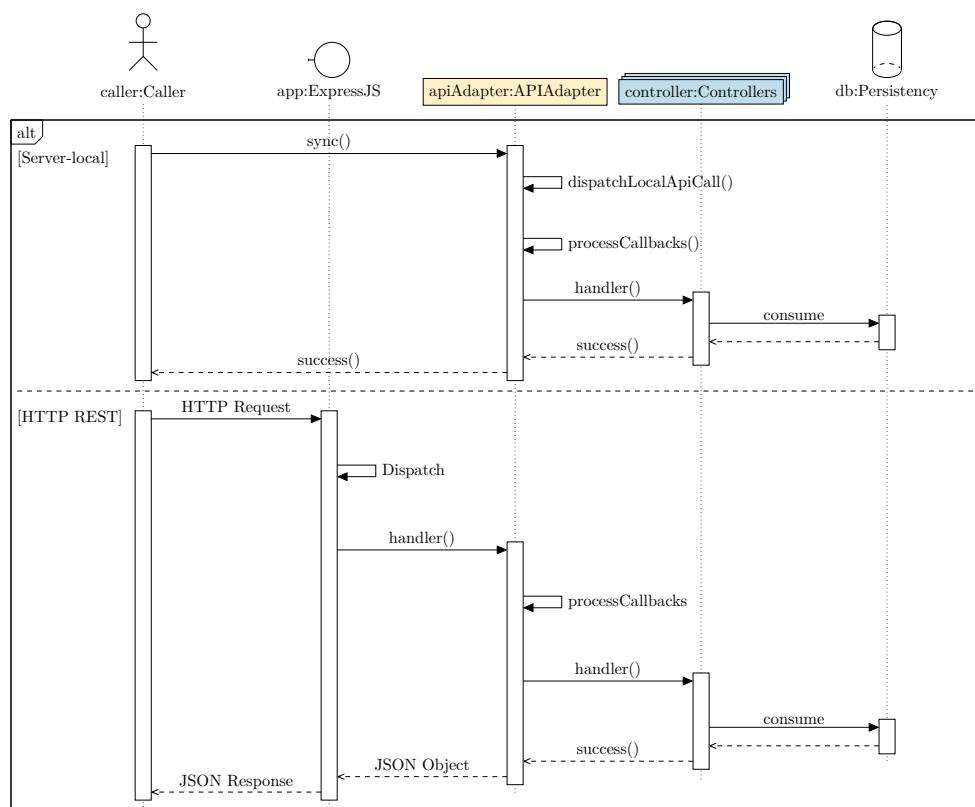


Abbildung 3.10.: Sequenzdiagramm: APIAdapter im server-lokalen sowie HTTP REST Modus

3.5.3. Quellcode Organisation

Für Applikationen mit Node.JS ist üblich ([Hold], [Sch]), den Quellcode soweit wie möglich eigenständige Komponenten zu unterteilen.

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

Die Beispielapplikation *Roomies* verwendet mehrere Komponenten aus NPM (siehe Quelltext 3.5) und ist dabei selbst in Komponenten unterteilt. Die Ordnerstruktur in Abbildung 3.11 veranschaulicht dieses Prinzip.

```

9 , "dependencies": {
10   "debug": "~0.7.2"
11   , "express": "~3.1.0"
12   , "sqlite3": "~2.1.7"
13   , "pg": "~0.15.1"
14   , "sequelize": "git://github.com/mweibel/sequelize.git#mekanics-mweibel-fix"
15     "
16   , "node-barefoot": "~0.0.11"

```

Quelltext 3.5: Auszug der verwendeten NPM Komponenten [AJWd]

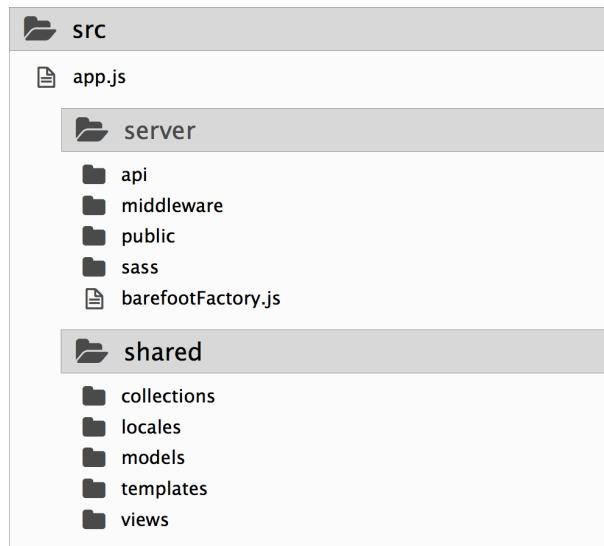


Abbildung 3.11.: Ordnerstruktur des *src*-Unterordners der Beispielapplikation *Roomies*

Jede Komponente mit JavaScript-Code kann mittels einem *require()*-Befehl eingebunden werden womit automatisch die enthaltene *index.js*-Datei geladen wird.

Im Folgenden werden die beiden wichtigsten Dateien für das Starten der Applikation erklärt.

```

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

Quelltext 3.6: Einbindung der Community-Komponente

barefootFactory.js

Die “barefoot”-Factory [AJWj] ist verantwortlich für das Setup der Server-Seite der Applikation. Unter anderem werden folgende wichtige Aufgaben ausgeführt:

- Definieren der JavaScript Dateien, welche zum Client übertragen werden sollen
- Einrichten des API-Adapters
- Express.js [Holf] Middlewares laden und initialisieren
- Express.js [Holf] HTTP Server starten
- Einrichten des Server Request Contexts (u.a. das Model für den eingeloggten Benutzer)

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

app.js

Die Datei “app.js” [AJWe] ist der Startpunkt der Applikation. Darin wird u.a. die “barefootFactory” instanziert und der “DataStore” initialisiert.

Der “DataStore” ist ein “barefoot”-Konzept [Alab]. Es wird verwendet um Models und Collections vom Server zum Client zu senden.

3.5.4. Barefoot

Barefoot [Alaa] ermöglicht die Verwendung desselben *Backbone.js* Quelltexts sowohl auf dem Server als auch auf dem Client. Dies wird durch gezieltes Einsetzen von Codefragmenten mittels Mixins erreicht.

Dazu werden neutrale Versionen jeder *barefoot*-Komponente während der Initialisierung des Frameworks (siehe Quelltext 3.7) mit umgebungsspezifischen Mixins ergänzt.

```

68 function boot(options) {
69   var Backbone = require('backbone')
70   , util = require('./util')
71   , environment = options.environment
72   , mixins = util.loadMixins(environment)
73   , Router = require('./router')(mixins.RouterMixin)
74   , View = require('./view')(mixins.ViewMixin)
```

Quelltext 3.7: Ermittlung der Runtime-Umgebung sowie anschliessendes applizieren der spezifischen Mixins [Alac]

Sobald die spezifischen Mixins (Zeile 72) geladen wurden, werden diese anschliessend ab Zeile 73 auf die neutralen Vorlagen appliziert.

3.5.5. JavaScript Callbacks

Ein Problem das insbesondere in JavaScript, aber auch in anderen Sprachen ein Thema ist, ist das der Callbacks oder auch "*Callback hell*".

Während der Implementation diverser APIs war auch immer wieder die Frage, wie man am Besten Code wie in Quelltext 3.8 strukturiert.

```
136 function createCommunity(req, res) {
137   var resident = req.user
138   , db = req.app.get('db')
139   , Community = db.daoFactoryManager.getDAO('Community')
140   , communityData = {
141     name: req.param('name')
142   };
143   createUniqueShareLink(db, function(err, link) {
144     if (err) {
145       return res.send(500);
146     }
147     communityData.shareLink = link;
148
149     resident.getCommunity()
150       .success(function getSuccess(community) {
151         if (community && community.enabled) {
152           req.flash('error',
153             'res.__("What exactly are you trying? You\'re ' +
154             'already in a community..."));
155           return res.redirect('/community');
156         }
157
158         Community.find({ where: { name: communityData.name }})
159           .success(function findResult(community) {
160             // ... more callbacks (see cited file)
161           })
162             .error(function findError() {
163               return res.send(500);
164             });
165           })
166             .error(function getError() {
167               return res.send(500);
168             });
169           });
170 }
```

Quelltext 3.8: Ausschnitt aus Community Controller mit Callback Hell [AJWf]

Quelltext 3.8 zeigt einen Ausschnitt der “createCommunity”-Funktion. Es ist sofort offensichtlich, dass dieser Quelltext sehr schwer lesbar ist und entsprechend refactored werden sollte.

Dazu gibt es zwei Ansätze:

- Funktional
- Promises / Futures

Funktional

Der funktionale Ansatz versucht jegliche Callbacks in eigene Funktionen auszulagern und dabei möglichst Seiteneffekte zu vermeiden.

Das vermeiden von Seiteneffekten ist ein gutes Prinzip. Gerade in Funktionen wie in Quelltext 3.8 zum Teil aber relativ umständlich, da alle benötigten Kontext-Variablen mitgegeben werden.

Quelltext 3.9 zeigt einen Ausschnitt aus der neusten Version derselben Funktionalität und ist mit dem funktionalen Ansatz umgeschrieben worden.

```
192 function createCommunity(success, error, data) {
193   var // ...
194   , forwardError = function forwardError(err) {
195     // ...
196   }
197   , afterCommunitySearch = function afterCommunitySearch(community) {
198     // ...
199     communityDao.create(communityData)
200       .success(afterCommunityCreate)
201       .error(forwardError);
202   }
203   , afterCommunityCreate = function afterCommunityCreate(community) {
204     // ...
205     createUniqueSlug(db, community.name, community.id
206       , function(err, slug) {
207         // ...
208         community.save()
209           .success(afterUpdateCommunitySlug)
210           .error(forwardError);
211       });
212   }
213   , afterUpdateCommunitySlug =
214     function afterUpdateCommunitySlug(community) {
215       // ...
216       updateResidentsCommunityMembership(
217         resident
218         , community
219         , function(err) {
220           // ...
```

```

222     }
223   );
224 }
225
226 createUniqueShareLink(db, function(err, shareLink) {
227   // ...
228   communityDao.find({ where: { name: communityData.name, enabled: true }})
229     .success(afterCommunitySearch)
230     .error(forwardError);
231 });
232 }
```

Quelltext 3.9: Ausschnitt aus dem neusten Community Controller [AJWh]

Nachteil dieser Version sind die jeweiligen Runtime-Neudefinitionen der Funktionen bei jedem Funktionsaufruf. Dies wurde aber bewusst zugunsten der Klarheit in Kauf genommen.

Promises / Futures

In “barefoot” sind mehrere Schritte notwendig, um bspw. eine View zu rendern (siehe Diagramm 3.8). Jeder Funktionsaufruf welcher im Diagramm 3.8 gezeigt wird ist asynchron. Der Ablauf dieser Funktionen ist aber vorgegeben. Deswegen wurden sogenannte Promises (auch bekannt als Futures [Wikg]) mithilfe der Q Bibliothek [Kow] implementiert.

Quelltext 3.10 zeigt die Implementation in “barefoot”.

```

215 Q.fcall(initDOM)
216   .then(renderMainView)
217   .then(renderView)
218   .then(serializeDataStore)
219   .done(writeHTTPResponse, writeHTTPError);
```

Quelltext 3.10: Ausschnitt aus Server router-mixin.js [Alad]

Mit diesen beiden Ansätzen bzw. Refactoring-Techniken ist es sehr einfach, gut les- und wartbaren Quelltext zu schreiben.

Kapitel 4 **Richtliniendemonstration**

Einleitung

In Kapitel 2, 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.

Das vorliegende Kapitel soll 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 Kapitels wird bewertet, wie gut die jeweiligen Richtlinien an der Beispielapplikation demonstriert werden konnten.

4.1. Übersicht

Die Tabelle 4.1 bietet eine Übersicht über die Analyse aller in Abschnitt 2.1.4 “Richtliniendemonstration” definierten Richtlinien.

Die Spalte “Nutzen” zeigt die Meinung des Projektteams bezüglich der Nützlichkeit der Richtlinie und kann dabei die Ausprägungen “Positiv”, “Neutral” oder “Negativ” haben. Die Spalten “Demonstriert” und “Bemerkung” zeigen auf, ob und unter welchen Umständen die jeweilige Richtlinie implementiert werden konnte.

Richtlinie	Nutzen	Demonstriert	Bemerkung
RP1 REST	😊	✓	Versionierung und 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	😊	!	<i>HTTP Basic Authentication over SSL</i> nicht umgesetzt
RP8 Cookies	😊	✓	
RP9 Session	😊	✓	
RP10 Browser-Controls	😊	✓	
RP11 POSH	😊	✓	
RP12 Accessibility	😊	✓	Wurde nicht explizit mit einem Hilfsgerät getestet
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	😊	✓	

Abbildung 4.1.: Übersicht Architekturrichtlinienanalyse (1/2)

<i>Richtlinie</i>	<i>Nutzen</i>	<i>Demonstriert</i>	<i>Bemerkung</i>
TP3 Eat your own API dog food	☺ ✓		Wiederverwendbare API umgesetzt
TP4 Separate user identity, sign-up (...)	☺ ✓		Facebook als Identity-Provider
TP7 Apply the Web instead of working around	☺ ✓		
TP8 Automate everything or you will be hurt	☺ ✓		CI, Make

Abbildung 4.2.: Übersicht Architekturrichtlinienanalyse (2/2)

4.2. RP1 REST

REST [Fie00] 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 Transportprotokoll und semantische Grundlage verwendet
- URIs sind Identifikatoren für Ressourcen auf dem Server
- Datenformat (XML, JSON, etc.) ist nicht vorgegeben

Die Zustandslosigkeit von REST ermöglicht die Implementation von skalierbaren Schnittstellen.

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 baut auf HTTP auf. 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 direkt ermöglicht. Voraussetzung dafür ist, dass die Software die entsprechenden Caching-Headers in den Antworten sendet.

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

Das Datenformat mit welchem Objekte und Collections übertragen werden ist hingegen nicht definiert. Dem Software Entwickler ist die Serialisierung der Daten somit selbst überlassen. Vielfach wird heute JSON eingesetzt, da es ein kompaktes und gut lesbares Datenformat ist (siehe [DuVa]).

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 auf der Server-Seite 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 wurde als separater Service-Layer implementiert. Auf diese wird Client- und Serverseitig transparent zugegriffen.

Das Beispiel im Quelltext 4.1 zeigt zwei Definitionen einer API-Route.

```

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 4.1: Community API Definition [AJWi]

Bei Zeile 34 wird eine “GET” API-Methode für das Abfragen einer Community 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 autorisiert ist.

Zeile 46 definiert eine “POST” API-Route um eine neue Community zu erstellen. Auch hier wird überprüft ob der Benutzer authentifiziert ist. Zusätzlich wird ein Daten-Validator verwendet, 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 durch ein zusätzliches Präfix für die API-Routes problemlos möglich.

Dasselbe gilt für das Caching. Jedes Objekt in der Datenbank hat eine Spalte mit der Information, wann es zuletzt modifiziert wurde. Durch diese Information kann ein generisches 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.

Dadurch dass REST keine Serialisierungsform festlegt, ist es dem Entwickler der API möglich, verschiedene Datenformate zu unterstützen (siehe auch 4.7 “RP6 Should-Formats”). Dies ermöglicht beispielsweise die Nutzung eines kompakten Datenformats, damit insgesamt weniger grosse Daten übertragen werden müssen.

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” [Comc].

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

4.3. RP2 Application Logic

Die Logik einer Applikation kann in zwei Kategorien unterteilt werden:

1. Businesslogik
2. Präsentationslogik

Businesslogik

Die Businesslogik beinhaltet das eigentliche Herz der Applikation. Sie legt fest, welche Zustandsübergänge für Ressourcen möglich sind, welche Daten erlaubt sind und was bei API-Aufrufen ausgeführt wird.

Zur Businesslogik gehört laut den “ROCA”-Autoren auch die *logikbasierte Validierung*. Diese beinhaltet [HST] Funktionalität zur Überprüfung der Korrektheit der übertragenen Daten im Sinne der Businesslogik.

Präsentationslogik

Bei jeder Aktion eines Benutzers müssen zwei Aktionen durchgeführt werden:

- API-Aufrufe (lokal und remote)
- Darstellung der nächsten Ansicht

Welche API-Aufrufe gemacht werden und was die nächste Ansicht beinhaltet, bzw. welche Ansicht die nächste ist, kann als Präsentationslogik bezeichnet werden.

Zur Präsentationslogik gehört laut den “ROCA”-Autoren die sogenannte *datenbasierte Validierung* [HST]. Diese überprüft die Korrektheit der Daten aufgrund ihres gewünschten Datentyps (z.B. Telefonnummern auf formale korrektheit etc.) und auch die Vollständigkeit der übertragenen Daten.

Damit eine Applikation “RP2”-konform ist, darf jegliche vorangehend als Businesslogik beschriebene Logik nur auf dem Server implementiert werden. Die Duplizierung auf den Client würde dem “*Don’t repeat yourself*”-Prinzip widersprechen und somit fundamentale Prinzipien des Software Engineerings verletzen.

Geplante Umsetzung

Die Beispielapplikation soll trotz Code-Sharing “RP2”-konform implementiert werden. Dies bedeutet, dass folgende Bedingungen beachtet werden sollen:

- Businesslogik soll in der API implementiert sein
- Logikbasierte Validierung gehört zur Businesslogik und soll somit ebenfalls in der API umgesetzt werden
- Die Präsentationslogik wird einmal implementiert und sowohl auf dem Client als auch auf dem Server verwendet
- Die datenbasierte Validierung wird auf der API implementiert und falls möglich auch mit dem Client geteilt.

Konkrete Umsetzung

Wie in Kapitel 3 “Technische Architektur” gezeigt, wurde sowohl ein API- als auch ein Shared-Layer implementiert.

Die geplante Umsetzung konnte erfolgreich durchgeführt werden. Die API beinhaltet alle Businesslogik sowie die logikbasierte Validierung, während die “Shared Codebase” Präsentationslogik beinhaltet.

Die datenbasierte Validierung wurde aus Zeitgründen nur auf der API-Schicht implementiert und nicht mit dem Client geteilt. Dies wäre eine mögliche Weiterentwicklung.

Diskussion

Ein Grundprinzip für Software Entwickler lautet “*Never trust the client*” [Wike]. Die Richtlinie *RP2* lässt sich direkt auf dieses Prinzip anwenden. Um sicherstellen zu können, dass jegliche übermittelte Daten eines Benutzers korrekt sind, müssen diese zwingend auf dem Server überprüft werden.

Diese Richtlinie ist für Client-Server Anwendungen wichtig. Was passiert aber, wenn Peer-to-Peer Anwendungen ohne zentralen Server implementiert werden sollen? Mit

WebRTC [Gooe] rückt dieses Thema vermehrt auch für Webapplikationen ins Zentrum der Aufmerksamkeit. Vermutlich kann dieses Prinzip nicht ohne weiteres auf solche Applikationen appliziert werden und bedarf darum weiterer Analysen von spezifischen Peer-to-Peer Patterns.

Im Projektteam ist man sich einig, dass diese Richtlinie für Client-Server Anwendungen umgesetzt werden muss. Falls man gewisse Codeteile sowohl auf dem Client und auf dem Server verwendet werden, kann es aber durchaus Sinn machen, schon auf dem Client diese Validierung zu machen. Dies dient aber vor allem der User Experience und darf niemals die Überprüfung auf dem Server ersetzen.

4.4. RP3 HTTP

Das HTTP Prinzip von ROCA ist ähnlich wie Abschnitt 4.2 “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 soll unter anderem über Backbone Models welche mittels den beiden Methoden “save()” und “sync()” bzw. “fetch()” (siehe [Docb]) mit dem Server kommunizieren.

Um 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 Umsetzung ist wie geplant ausgeführt worden. Als Beispiel zeigt Quelltext 4.2 ein *Barefoot.Model* [Alae], welches eine URL zur API definiert und über diese 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 4.2: Community Model [AJWp]

Um dieses Model mit Daten abzufüllen, kann wie in Quelltext 4.3 gezeigt, eine Instanz 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 4.3: Community Model Synchronisation [AJWt]

RESTful Forms

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

Die Unterstützung dieser zusätzlichen Methoden erfordert eine Hilfskonstruktion:

- Jedes Formular das nicht “POST” oder “GET” verwendet, erhält ein zusätzliches verstecktes Feld namens `_method` und dem Wert der gewünschten Methode
- Der Server interpretiert das Feld des vom Benutzer abgeschickten Formulars

- und ruft danach den entsprechenden Controller mit der entsprechenden Methode auf.

Quelltext 4.4 zeigt die Anbindung der entsprechenden “MethodOverride” Middleware [Sena] an eine Express-Applikation auf Zeile 38.

```

18 /**
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 4.4: HTTP Middleware [AJWm]

Als Beispiel zeigt Quelltext 4.5 das Formular für das abschliessen einer Aufgabe. Auf Zeile 7 wird das entsprechende Feld definiert.

```

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

```

Quelltext 4.5: Formular mit verstecktem `_method` Feld [AJWs]

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 wird ist diese Hilfskonstruktion sicher ein gehbarer Weg.

4.5. RP4 Link

Das “Link-Prinzip” ist eng verbunden mit Abschnitt 4.11, “RP10 Browser-Controls” und stellt die Anforderung, dass jede Seite eindeutig per URL 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 Nutzer häufig nicht nachvollziehbar und die User Experience leidet damit.

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

Geplante Umsetzung

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

Konkrete Umsetzung

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

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

Diskussion

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

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

Das Projektteam ist davon überzeugt, dass diese Richtlinie umgesetzt werden muss.

4.6. 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 einhält, insbesondere Abschnitt 4.2, “RP1 REST”.

Geplante Umsetzung

Eine REST-API wird laut Abschnitt 4.2 umgesetzt und somit ist es möglich, 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 wird trotzdem nicht geplant.

Konkrete Umsetzung

Eine REST-API wurde umgesetzt. Durch die Anbindung an den Identity Provider “Facebook” (siehe Abschnitt 4.21) ist es unumgänglich, vorher ein gültiges 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

```
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 4.6: cURL Request auf Roomies

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

Wie geplant wurde aus Zeitgründen kein weiterer Login-Mechanismus implementiert.

Diskussion

Durch den Einsatz einer generalisierten API-Schnittstelle mittels REST kann eine Software die entsprechenden Daten auch ohne HTML-Parser 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 in Abschnitt 4.2 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 es weiter, einen zusätzlichen Authentisierung-Provider (z.B. OAuth [oau]) zu implementieren, falls andere Komponenten auf die API zugreifen sollen.

4.7. RP6 Should-Formats

Wie bereits der Name ROCA sagt, sollen Anwendungen ressourcen-orientiert sein. Damit diese Ressourcen auch in anderen Anwendungen als in einem Browser verwendet werden können, muss entweder das erzeugte HTML maschinenlesbar sein oder es müssen alternative Formate (bspw. JSON, XML etc.) angeboten werden.

Geplante Umsetzung

Im Abschnitt 4.2 “RP1 REST” wird darauf hingewiesen, dass eine REST API mit JSON als Datenformat umgesetzt werden soll.

Konkrete Umsetzung

Abschnitt 4.2 “RP1 REST” zeigt, dass eine API umgesetzt wurde. Die Schnittstelle liefert JSON als Ausgabeformat. Dies wird garantiert, indem ein JavaScript Objekt an *Express.js* übergeben und anschliessend als JSON Response [Holg] an den Client gesendet [Hole] wird.

```
333  /** Function: getCommunityWithSlug
334   * Looks up a community with a specific slug.
335   *
336   * Parameters:
337   *   (Function) success - Callback on success. Will pass the community data
338   *     as
339   *       first argument.
340   *   (Function) error - Callback in case of an error
341   *   (String) slug - The slug of the community to look for.
342 */
342 function getCommunityWithSlug(success, error, slug) {
343   debug('get community with slug');
344
345   var communityDao = getCommunityDao.call(this);
346
347   communityDao.find({ where: { slug: slug, enabled: true }})
```

```
348     .success(function findResult(community) {
349       if(!_.isNull(community)) {
350         success(community.dataValues);
351       } else {
352         error(new errors.NotFoundError('Community with slug ' + slug +
353           ' does not exist.'));
354       }
355     })
356     .error(function daoError(err) {
357       error(err);
358     });
359 }
```

Quelltext 4.7: Community API getCommunityWithSlug [AJWg]

Quelltext 4.7 zeigt, wie das Community Objekt an den Success-Handler weitergegeben wird. Der Success-Handler ist von *barefoot* definiert und reicht den übergebenen Parameter an *Express.js* weiter. Express.js generiert daraus einen JSON String.

Diskussion

Auf den ersten Blick scheint diese Richtlinie keinen sonderlichen Mehrwert zu liefern. Liefert die API einer Applikation bereits JSON, warum soll sie zusätzlich bspw. auch noch XML ausgeben können?

“RP6 Should-Formats” ist anders zu verstehen: Es soll zusätzlich zum Generieren von HTML (welches aufgrund von “RP14 Unobtrusive JavaScript” ausdrücklich erforderlich ist) ein Ausgabeformat generiert werden, welches von Maschinen einfach lesbar ist, wie eben JSON oder XML.

Um die beiden Richtlinien “RP5 Non Browser” und “RP14 Unobtrusive JavaScript” umsetzen zu können muss auch *RP6* umgesetzt werden. Ob JSON oder XML verwendet wird ist dabei gleichgültig und muss von der umsetzenden Applikation entschieden werden.

Das Projektteam kann die Umsetzung dieser Richtlinie *RP6 Should-Formats* uneingeschränkt empfehlen.

4.8. RP7 Auth

Mit *Basic Access Authentication* [Uni+] steht seit Version 1.0 von *HTTP* eine einfache Möglichkeit zur Verfügung, Ressourcen vor dem Zugriff Unbefugter zu schützen. Dank der hohen Verbreitung, bedingt durch die frühe Integration in den *HTTP Standard*, bringt der Verzicht auf Cookies, Session ID's oder Anmeldeseiten Eleganz durch Verwendung grundlegender Protokollfeatures.

Der Server kann durch Senden eines *WWW-Authentication* Headers die Authentifizierung des Clients verlangen. Dieser wiederum sendet über den *Authorization*-Header in seiner Antwort Benutzername und Passwort, welche mittels Base64 [Wikc] kodiert sind.

Wird *HTTP Basic Auth* in dieser Form verwendet, werden Benutzername und Passwort in Klartext über das Internet übertragen. Abhilfe schafft *Digest Access Authentication*, wenn auch nur unbefriedigend. Hierbei werden die Identifizierungsmerkmale vor dem Übertragen mit einem Hashing-Algorithmus, bspw. MD5 maskiert. Der Server vergleicht dann den übertragenen Wert mit dem selbst berechneten Hash.

Insbesondere die Verwendung des MD5-Hashes gilt als unsicher [Comb]. Die Einfachheit von *Basic Access Authentication* kann unter Verwendung des *Secure Socket Layer* Protokolls [FPP] problemlos beibehalten werden. Dabei wird die Kommunikation zwischen Client und Server in einen verschlüsselten *SSL*-Tunnel verpackt. Alle übertragenen Informationen sind so nicht mehr ohne weiteres von Dritten lesbar.

Um eine effiziente Sicherung von Ressourcen zu gewährleisten, schlägt *RP7 Auth* die Verwendung des vorgestellten *HTTP Basic Access Authentication over SSL* vor.

Geplante Umsetzung

Die Umsetzung von *HTTP Basic Authentication over SSL* ist nicht geplant. Im Bereich *Security* und *Authentication* soll der Fokus auf “TP4 Separate user identity, sign-up and self-care from product dependencies” gelegt werden.

Konkrete Umsetzung

Den Erwartungen entsprechend wurde das von *RP7* vorgeschlagene *HTTP Basic Authentication over SSL* nicht in der Beispielapplikation *Roomies* implementiert.

Diskussion

Die ROCA Richtlinie *Auth* ergänzt die Forderungen von “RP5 Non Browser”: Die Kombination ermöglicht den browserfreien Zugriff auf geschützte REST-API-Ressourcen.

Für die Beispielapplikation *Roomies* wurde aufgrund der umzusetzenden Anforderungen auf die Demonstration von *RP7 Auth* verzichtet. Zwar bietet auch die in 4.21 “TP4 Separate user identity, sign-up and self-care from product dependencies” vorgestellte Lösung eine sichere Variante, einen Benutzer für den Zugang zur Applikation zu authentisieren. Wie der Abschnitt 4.6 jedoch ausführlich beschreibt, kann dies nur mit einem Browser praktikabel funktionieren.

Nach Umsetzung der Beispielapplikation empfiehlt das Projektteam darum, sollte es mit den Aufwänden vereinbar sein, *RP7 Auth* umzusetzen und die API via *HTTP Basic Auth over SSL* zugänglich zu machen.

4.9. 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.

Konkrete Umsetzung

Das Geplante konnte mit Erfolg umgesetzt werden. Um einem User eine Session zuzuweisen zu können verwendet Express.js [Holf] 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 muss mit allen künftigen Requests mitgeschickt werden.

In der Beispielapplikation wird die Session-Middleware wie in Quelltext 4.8 initialisiert:

```
31 app.use(express.session({
32   store: new SequelizeStore({
33     db: db
34   })
35   , secret: config.sessionSecret
36 }));
```

Quelltext 4.8: Connect Session Middleware [AJWn]

Der “SequelizeStore” [Weia] übernimmt die Speicherung der Session-Daten innerhalb der Datenbank.

Diskussion

Das Einbinden von mehr Informationen in Cookies hat mehrere Nachteile:

- Sicherheit

Wenn Benutzername und Passwort in ein Cookie gespeichert werden kann dies ohne Probleme von Drittpersonen mitgelesen werden. Falls HTTPS eingesetzt (auch wenn dies nur direkt im Browser geschieht) bildet dies ein nicht tragbares Risiko. Außerdem ist es einem Benutzer möglich seine Cookies abzuändern. Dies könnte nur mithilfe von Validierung und dem Einsetzen eines Message Authentication Codes erkennbar gemacht werden.

- Datenmenge

Cookies werden mit jeder Anfrage und jeder Antwort zwischen Client und Server mitgeschickt. Zwar ist die Grösse eines Cookies beschränkt, trotzdem wird die Datenmenge unnötig grösser.

Aus diesen Gründen sollte unbedingt darauf verzichtet werden, Cookies für andere Zwecke als für die wiederkehrende Authentifizierung über eine Session-ID oder für das Tracking zu verwenden. Für diese Zwecke eignen sich Sessions oder die Speicherung in die Datenbank besser.

4.10. RP9 Session

Eine Session sollte ausschliesslich für simple Authentifizierungsinformationen verwendet werden. Dabei wird eine eindeutige Session-ID generiert, mit welcher der angemeldete Benutzer unmissverständlich erkannt werden kann. Dies reicht für die wiederholte Authentifizierung aus.

Geplante Umsetzung

Der Session-Kontext soll nicht als Datenspeicher missbraucht werden. Aus diesem Grund soll dessen Verwendung auf die beschriebenen Authentisierungsinformationen beschränkt sein.

Konkrete Umsetzung

Die für Benutzer-Authentifikation verwendete Bibliothek *Passport.js* [Hanb] (siehe auch 4.21 “TP4 Separate user identity, sign-up and self-care from product dependencies”) speichert beim erstmaligen Anmelden eines Benutzers dessen Benutzerdaten serialisiert in die Session. Ab diesen Punkt hat *Roomies* bis zum Löschen der Session Zugriff auf diese Informationen.

Um die User Experience zu steigern wurde entschieden, zusätzlich auch transiente Informationen im Session-Kontext zwischenzuspeichern. Konkret handelt es sich dabei um die Zielseite einer Weiterleitung, wie folgendes Beispiel verdeutlicht:

Ein Benutzer von “Roomies” liebt es, sich mit den anderen Mitbewohner zu vergleichen. Deswegen besucht er oft die Rangliste. Auch einen Favorit hat er auf die Seite gesetzt, damit er direkt auf der Rangliste landet.

Ist der Benutzer noch nicht angemeldet und möchte die Rangliste besuchen wird er vom System aufgefordert sich anzumelden. Nach dem Anmeldevor-gang wird ihm wie gewünscht jene Seite gezeigt.

Das Weiterleiten auf die Rangliste ist nur möglich, da der Server die URL vor dem Anmelden in den Session-Kontext speichert, wie der Quelltext 4.9 zeigt.

```
225
226  /** Function: redirectIfNotAuthorized
227  * If the client is not authorized it will redirect. Otherwise it returns
228  * false to indicate that the calling method can continue work.
229  */
230 , redirectIfNotAuthorized: function redirectIfNotAuthorized() {
231   if(!this.isAuthorized()) {
232     var req = this.apiAdapter.req;
233     req.session.redirectUrl = req.originalUrl;
234     this.navigate('', {trigger: truereturn true;
236   }
237   return false;
238 }
```

Quelltext 4.9: Router - Autorisationskontrolle [AJWq]

Diskussion

Wie in der konkreten Umsetzung beschrieben, kann diese Richtlinie nicht immer ohne Weiteres eingehalten werden.

Das Speichern vergleichsweise einfacher Informationen wie der Weiterleitungs-URL ist aus Sicht des Projektteams vertretbar. Kritischer zeigt sich die Situation bei extensiver Nutzung des Kontexts. Wird zum Beispiel ein kompletter Warenkorb eines Webshops im Session-Kontext abgelegt, hat dies mehrere negative Auswirkungen: Zum einen gehen die Informationen verloren, sobald der Benutzer die Session beendet (Browser schliessen usw.), zum anderen skaliert diese Implementation bei vielen Sessions schlecht, da die Arbeitsspeicherauslastung linear zur Anzahl Sessions steigt. Für persistente, komplexe Session-Daten sollte entsprechend auf den Persistence-Layer zurückgegriffen werden.

Das Projektteam gibt entsprechend folgenden Ratschlag: Solange nur kleine, simple und temporäre Daten im Session-Kontext gespeichert werden, ist dies problemlos vertretbar. Es muss aber bei jeder Implementation genau überlegt werden, ob und welche Trade-Offs entstehen. Weiter ist das Projektteam der Meinung, dass geschäftsprozess-kritische, transiente Informationen in keiner Situation im Session-Kontext abgelegt werden sollten.

4.11. 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 klickt man auf den Zurück-Knopf 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 kein fiktives Beispiel, sondern ein reelles Szenario aus dem Online-Shop des Elektronikgrosshändlers Digitec [Dig].

Das ROCA-Prinzip “Browser-Controls” will genau dieses unangenehme Erleben verhindern. Dabei ist es eng mit dem Prinzip “RP4 Link” verbunden. Indem jede Seite ihre eigene URI hat und diese beim Navigieren in die “History” des Browsers hinzugefügt wird, funktionieren die Standardbedienelemente (Zurück, Vorwärts und Aktualisieren) eines Browsers weiterhin erwartungsgemäss.

Geplante Umsetzung

Jeder Seitenwechsel innerhalb der Beispielapplikation *Roomies* soll über den Verlauf im Browser nachvollziehbar sein und so dem Benutzer erlauben, die im Browser eingebauten Funktionen zur Navigation zu benutzen.

Konkrete Umsetzung

Mit der Funktion “Backbone.history.start()” [Docc] bietet *Backbone.js* resp. *barefoot* eine Anbindung an die History API [Mozd] und ermöglicht damit JavaScript Applikationen das Navigieren inkl. einer 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 4.10: Aktivierung der History in Barefoot [Alaf]

Diskussion

Das einleitende Beispiel *Digitec* ist keine Seltenheit: Oft leidet die User Experience auf entsprechenden Angeboten sehr. Diese Internetseiten verletzen zudem meist absichtlich RP4 Link um Funktionalitäten wie *Frames* einbinden zu können.

In der Zeit der Social Networks gehört es zum Alltag, interessante Links mit Freunden zu teilen. Gerade hier wird dem Endbenutzer oft bewusst, dass das geteilte Angebot hintergründig nicht *RP4* und somit *RP10* befolgt. Dem Projektteam ist es unverständlich, warum nicht mehr Gebrauch von neuen Features wie dem *History API* gemacht wird. Der Mehrwert für den Benutzer bzw. die Verbesserung der User Experience ist oft immens.

Abschliessend gehört für das Projektteam die Umsetzung von *RP10 Browser-Controls* zu einem Muss.

4.12. RP11 POSH

“POSH”, ausgeschrieben “Plain Old Semantic HTML”, bezeichnet das Erstellen von HTML-Seiten mithilfe von semantischen 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

Das HTML Markup der Beispielapplikation sollen eine klare und logische Struktur aufweisen.

Alle Seiten haben einen Titel, eine Überschrift und entsprechenden Inhalt.

Für Überschriften werden die Tags *h1*, für die grösste und wichtigste Überschrift, bis *h6* für untergeordnete.

Tabellen sollen für tabellarische Daten verwendet werden und nicht zur gestalterischen Strukturierung einer Seite.

Konkrete Umsetzung

Um semantische Inkorrekttheiten durch Wiederholungsfehler zu reduzieren, wurde in *Roomies* mit Templates gearbeitet. So können bspw. Menüs oder Fusszeilen einmalig definiert und wiederverwendet werden. Dies geht sehr stark nach dem “Don’t repeat yourself” Prinzip. Fehler werden verhindert, indem man sie gar nicht zu schreiben hat.

Quelltext 4.11 zeigt einen Auschnitt des HTML-Markups vom Haupttemplate von *Roomies*. Die Seitenstruktur ist klar zu erkennen: Oben beginnt der Header mit dem Menü, gefolgt von einem Platzhalter für allfällige Fehler-, Warnungs- oder Informationsmeldungen. Weiter ist der Hauptbereich mit der ID “main” ersichtlich, welcher zur Laufzeit mit eigentlichen Applikationsinhalten gefüllt wird. Abschliessend steht ein semantisch korrektes *footer*-Element, welches final den Link zur Abmeldung des Benutzers enthalten 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 4.11: Layout Definition [AJWk]

Diskussion

Semantisches HTML ist aus zwei Gründen eine gute Idee:

- Suchmaschinenoptimierung
- Barrierefreiheit

Suchmaschinenoptimierung

Suchmaschinen wenden komplexe Algorithmen an, um Webseiten zu analysieren und in verwertbare Suchresultate zu transformieren. Ein Entwickler von Webseiten und -applikationen kann durch die Verwendung von semantisch korrekten Tag-Elementen viel zur Optimierung des Suchprozesses beigetragen werden. Diese ermöglichen den Suchmaschinen ein fehlerfreies Interpretieren der untersuchten Informationen.

Ergänzend dazu helfen Standards wie *schema.org* [IIC]. Unterstützt von Google, Yahoo

und Microsoft, hat dieser zum Ziel, die Maschinenlesbarkeit von Webseiten zu erhöhen und spezifischen Teilen einer Seite analysierbaren *Sinn* zu geben.

Mithilfe von *schema.org* können bspw. Attribute von Produkten (Name, Preis etc.) gekennzeichnet und gezielt für Suchmaschinen verwertbar gemacht werden. Sucht ein Benutzer nun nach Produkten, kann die Suchmaschine die nun strukturierten Informationen besser durchsuchen und massgeschneiderte Suchergebnisse dem Benutzer präsentieren.

Für Dienstanbieter ergibt sich so ein enormes Potential zur Monetarisierung.

Barrierefreiheit

Nebst der Verbesserung der Sichtbarkeit bei Suchmaschinen ist ein semantisch korrektes HTML-Markup auch aus Sicht der Barrierefreiheit sinnvoll.

Ein signifikanter Teil der Bevölkerung ist auf barrierefreie Webseiten angewiesen [Wikb]. Das Erstellen von barrierefreien Webseiten unterstützt Geräte wie *Screen Reader*, Inhalte zu analysieren und in einem für den gehandicapten Benutzer gerechten Format zugänglich zu machen.

Semantische Tags sind einer der wichtigsten Schritte auf dem Weg zu einem barrierefreien Webangebot.

Das Projektteam ist aufgrund der genannten Gründe der Meinung, dass semantische Tags und somit *RP11 POSH* sehr wichtig ist und auch für eine Webapplikation umgesetzt werden sollte.

4.13. RP12 Accessibility

Aufbauend auf dem Prinzip “RP11 POSH” besagt das Prinzip *Accessibility*, dass alle Seiten für Hilfesoftware/-geräte wie *Screen Reader* oder *Braillezeile* zugänglich sein müssen.

“Barrierefreiheit schliesst sowohl Menschen mit und ohne Behinderungen als auch Benutzer mit technischen (Textbrowser oder PDA) oder altersbedingten Einschränkungen (Sehschwächen) [...] ein.” – [Wikb]

RP12 Accessibility beschränkt sich dabei auf Zugänglichkeit über Hilfeanwendungen. So wird die in der Definition erwähnte Problematik *Farbblindheit* (richtige Farbenwahl etc.) oder die Navigation ohne Maus nicht abgedeckt.

Geplante Umsetzung

Damit Hilfesoftware- und Hilfegeräte das User Interface der Beispielapplikation interpretieren kann, soll das HTML Markup, wie unter 4.12 “RP11 POSH” erklärt, eine semantisch korrekte Struktur aufweisen.

Konkrete Umsetzung

Roomies verwendet, wo nötig, semantisch korrekte HTML-Elemente.

Diskussion

In “RP11 POSH” wird ausführlich über die Wichtigkeit eines barrierefreien Internets diskutiert. Das Prinzip *RP12 Accessibility* strebt durch die notwendigen Vorehrungen in die selbe Richtung, hat aber die Unterstützung gehandicappter Personen zum Ziel.

Leider gehört die Berücksichtigung von Farbenblindheit o.Ä. nicht zum Repertoire von *RP12*. Für das Projektteam gehören aber auch diese Themen klar zum Aufgabenbereich der Richtlinie *Accessibility*.

Das Projektteam schlägt darum die Ergänzung von *RP12 Accessibility* um die erwähnten Punkte vor. Machen es die Anforderung nötig, empfiehlt das Projektteam die Umsetzung von *RP12*.

4.14. RP13 Progressive Enhancement

Die Client-Technologien HTML und CSS haben sich über Jahre weiterentwickelt. Dies auch meistens mit dem Hintergedanken “Progressive Enhancement” oder Rückwärtskompatibilität. Das bedeutet, dass die Entwicklung vielfach versucht hat, Rücksicht auf die älteren Browserversionen zu nehmen. Deutlich sieht man diese Rückwärtskompatibilität beim neuen HTML5 Standard. Neue Formularfeldtypen wie “tel” oder “email” wurden eingeführt. Browser die HTML5 nicht unterstützen, wechseln bei einem für sie unbekannten Typ zum normalen Textfeld zurück.

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

Quelltext 4.12: Formularfeld mit HTML5, welches eine Telefonnummer erwartet

Natürlich gibt es keine Regel ohne Ausnahme. Manche Tags, die im HTML 5 Standard eingeführt wurden, werden überhaupt nicht beachtet. Dies kann vor allem bei Versionen *kleiner Neun* des Internet Explorers beobachtet werden.

Geplante Umsetzung

Die Beispielapplikation soll, wie bereits in Kapitel I.2 erwähnt, folgende Versionen unterstützen: Internet Explorer 8 und höher, Chrome 25 und höher, Firefox 19 und höher und Safari 6 und höher.

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 nicht alle geplanten und zu unterstützenden Browser HTML5 interpretieren können, musste ein kleiner Trick angewendet werden, um so die Rückwärtskompatibilität zu erhöhen. Dieser Trick heisst “modernizr” [Ate], eine JavaScript-Bibliothek, welche gezielt den verwendeten Browser auf seinen Funktions- und Unterstützungsumfang von HTML und CSS 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 einem "div"-Tag ersetzt werden müssen, wie dies sonst der Fall wäre.

Um diese Bibliothek einzubinden, hat man nichts anderes zu tun als das Skript im Header hinzuzufügen.

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

Quelltext 4.13: Einbinden von modernizr [AJWl]

Mit "modernizr" konnte erreicht werden, dass in allen geplanten Browser alle Elemente 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.

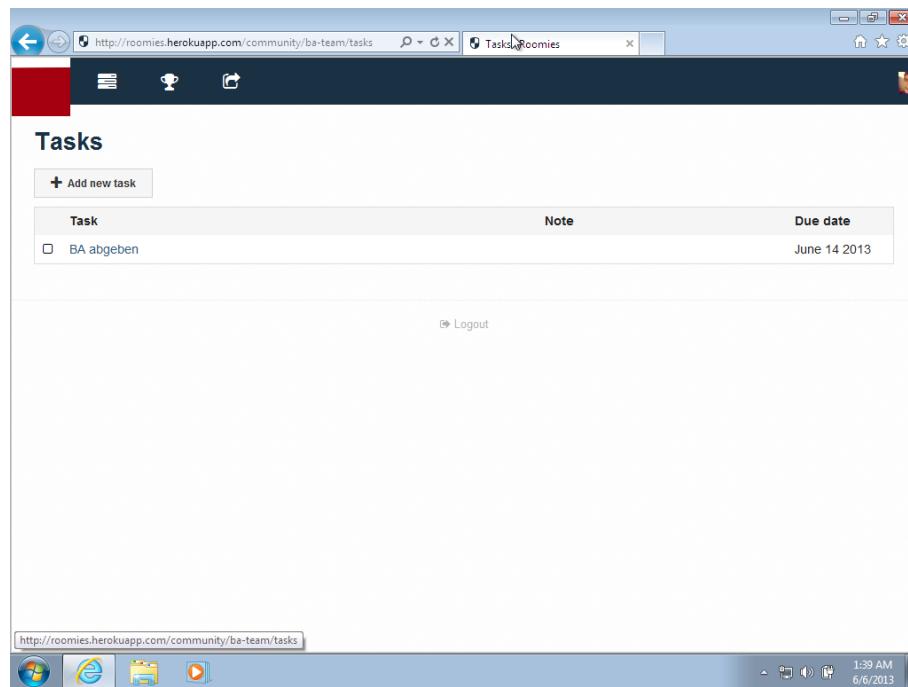


Abbildung 4.3.: Fehlerhafte Darstellung im Internet Explorer 8

Diskussion

Abhängig vom Zielpublikum der zu erstellenden Webapplikation gewinnt oder verliert die Unterstützung älterer Browser und somit "Progressive Enhancement" an Wichtigkeit. Mithilfe von Tools wie *modernizr* kann Kompatibilität einfacher gewährleistet werden. Die Implementation der Beispielapplikation hat jedoch bewiesen, dass diese Werkzeuge kein Allheilmittel darstellen.

Manuelles Testen und insbesondere eventuelle manuelle Korrekturen sind und bleiben wichtig.

Das Projektteam ist sich einig: *Progressive Enhancement* ist wichtig. Insbesondere die Sicherstellung der Kompatibilität zu älteren Versionen von Internet Explorer stellt immer noch eine Herausforderung dar und verursacht meist immense Kosten.

Weiter ist das Projektteam der Meinung, dass diese Probleme in Zukunft zwar nicht komplett wegfallen werden, aber immerhin reduziert werden könnten.

Microsoft verfolgt mit Internet Explorer 10 eine ähnliche Updatestrategie wie Mozilla und Google mit ihren Browsern: Häufige Veröffentlichung von Updates in kurzen Abständen [Kei] ermöglichen Microsoft künftig die vermehrte Partizipation bei der Entwicklung von Web-Standards. In absehbarer Zeit wird dies Webapplikationsentwicklern helfen, bestmögliche Kompatibilität zwischen verschiedenen Browsern, innerhalb kürzester Zeit, zu gewährleisten.

4.15. 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 4.4.: Kategorisierung aktueller Webapplikationen

Die Kategorie *Statisch* zeichnet sich durch 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 Serverkomponente.

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 ist, neue Inhalte über einen AJAX Request. Nach Erhalt des vorgerenderten HTML Markups aus der Antwort ersetzt JavaScript 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 4.2 "RP1 REST").

Die Verlagerung des User Interface Quelltexts direkt in den Browser hat den Vorteil, dass UI Elemente effizienter verändert und aktualisiert werden können. Soll bspw. nur ein kleiner Teil der Benutzeroberfläche aktualisiert werden, kann dies gezielt und ohne Umweg über einen erneuten Request an den Server geschehen.

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

Es ist bereits zu erkennen, dass diese Art von Webapplikation ohne JavaScript-Unterstützung im Browser nicht ausgeführt werden kann. Ein Beispiel hierfür liefert der *Google Drive* [Gooa] WebClient. Wie in Abbildung 4.5 ersichtlich verweigert dieser ohne aktiviertes JavaScript die Funktion und zeigt ein leeres Standardlayout mit einer entsprechenden Meldung an.

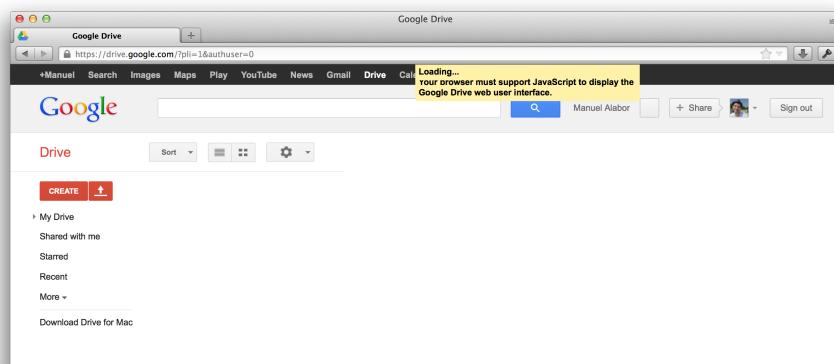


Abbildung 4.5.: *Google Drive* in Firefox 21.0 mit deaktiviertem JavaScript

Die Richtlinie *RP14 Unobtrusive JavaScript* aus dem ROCA Manifest verlangt, dass eine Webapplikation auch bei deaktiviertem JavaScript weiter funktionstüchtig bleibt. Soll JavaScript für ein modernes User Interface verwendet werden, muss gem. *RP14* also eine Mischform aus den vorgestellten Applikationstypen verwendet werden.

Neben dem "Wie und Wo" ein User Interface gerendert wird, ist die Richtlinie *Unobtrusive Javascript* auch eng mit der Thematik *Codequalität* verbunden: Die Vermengung von HTML Markup und JavaScript Code soll unterbunden werden. Hierzu zeigt der Quelltext 4.14 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 4.14: Beispiel einer Vermischung von HTML Markup und JavaScript

In den Quelltexten 4.15 und 4.16 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 4.15: Beispiel eines sauberen HTML Markups 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 4.16: 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 der entsprechender Programmcode die Umsetzung der im einleitenden Abschnitt vorgestellten Funktionalitäten eines vollwertigen *JavaScript Clients*.

Zugriffe auf persistente Applikationsdaten sollen wie in 4.2 “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, dass die entstehende Applikation zwar wie erwartet die gewünschte hybride Form aufweisen wird, aber keinesfalls mit der ROCA Richtlinie *RP15 No Duplication* vereinbar sein wird.

Dank der durchgängigen Verwendung von JavaScript hätten viele Codefragmente wie die View-Templates (Beispiel siehe Quelltext 4.17) 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       
38     </a>
39   </li>
40 </ul>
41 {{/if}}
```

Quelltext 4.17: Ausschnitt aus dem *Handlebars* [Kat] Template zur Darstellung von Benutzerinformation in der Menüleiste von *Roomies* [AJWr]

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 3.5).

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

Zwar gibt es mit *Rendr* von *Airbnb* [Aird] eine Konkurrenzbibliothek auf diesem Gebiet. Aufgrund einiger, insbesondere designbedingter Nachteile, entschied das Projektteam jedoch, mit *barefoot* eine eigene Implementation umzusetzen:

Problem von Rendr	Lösung durch barefoot
Unterstützt nur <i>Handlebars</i> [Kat] für View-Templates	Keine Vorgabe resp. ohne Template-Engine verwendbar
Quelltext für Client kann nicht automatisch zusammengefasst und zur Übertragung optimiert werden	<i>Barefoot</i> verwendet hierfür die <i>browserify-middleware</i> (siehe auch 4.18 "RP17 Static Assets")
<i>Rendr</i> ist mittels <i>CoffeeScript</i> [Ash] implementiert	<i>Barefoot</i> setzt auf pures JavaScript unter Verwendung von <i>Underscore</i> [Aire]
Wiederholte Überprüfung der aktuellen Runtime-Umgebung (Client oder Server) mittels entsprechender <i>if</i> -Statements	Einmalige Überprüfung, anschliessende Verwendung umgebungsspezifischer Mixins

Abbildung 4.6.: *Rendr* vs. *barefoot*

Diskussion

Ähnlich den zwei Typen von Webapplikationen sind hier zwei kontroverse Strömungen in der Entwicklergemeinschaft erkennbar [Comd]: Die eine Gruppe drängt zur alleinigen Nutzung der neusten Features und tendiert daher 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 müsse
- zum anderen die Umsetzung einer ebensolchen *unobtrusive* Lösung entsprechend aufwändig sei.

Beiden Lagern kann das Projektteam mit der umgesetzten Beispiellapplikation 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, einfacher eine "unobtrusive" Lösung umzusetzen.

4.16. RP15 No Duplication

Um *RP15* einfach erklären zu können, soll folgendes Beispiel dienen:

In einer Webapplikation sollen die Benutzereingaben aus einem Formular auf formale Korrektheit hin geprüft werden. Beim Versenden des Formulars werden dazu die übertragenen Informationen in der Backendkomponente überprüft und ggf. mit einer Fehlermeldung zurückgewiesen.

Für eine Verbesserung der User Experience soll nun bereits vor dem Versenden des Formulars im Frontend eine Prüfung der Eingaben gemacht werden. Da die Backendkomponente mit PHP implementiert wurde, entscheidet der zuständige Entwickler den bestehenden Code mit JavaScript auf den Client zu portieren.

Das Beispiel verdeutlicht, welche Stellen einer Webapplikation tendenziell besonders anfällig für duplizierten Quelltext sein können.

Die Richtlinie 15 *No Duplication* soll die Erstellung von doppelten Codefragmenten minimieren resp. komplett verhindern.

Geplante Umsetzung

Die Aufhebung der Sprachbarriere, welche durch Verwendung von JavaScript sowohl auf Client- als auch auf Serverseite resultiert, soll bereits zu einem grossen Teil zur Vermeidung von doppelten Codefragmenten beitragen.

Das Projektteam will zudem durch geschickte Erstellung von Modulen die Wiederverwendbarkeit des enthaltenen Quelltexts erleichtern.

Konkrete Umsetzung

Mit der durchgängigen Verwendung von *barefoot* [Alaa] für die Implementation der Beispielapplikation konnte der Anspruch von *RP15 No Duplication* besser als erwartet umgesetzt werden.

Wie unter “Konkrete Umsetzung” im Abschnitt 4.15 “RP14 Unobtrusive JavaScript” bereits ausführlich beschrieben wurde, konnte eine durchgängige und duplikatfreie Codebasis umgesetzt werden.

Diskussion

Unabhängig von der Entwicklung von Webapplikationen kennt der Software Engineer das Prinzip von *Don't repeat yourself*. Dementsprechend bietet *RP15 No Duplication* eigentlich keine grundlegenden Neuerungen. Wie in der Beispielapplikation aufgezeigt werden konnte, erleichtert die Verwendung der gleichen Programmiersprache in Front- und Backend die Umsetzung von *RP15* zudem zusätzlich.

Lassen es daher die Umstände zu, empfiehlt das Projektteam aufgrund des besser wartbaren Codes die Umsetzung von *RP15 No Duplication* uneingeschränkt.

4.17. 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 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 aufbaut.

```
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 4.18: Beispiel eines HTML Gerüsts zum Rendering eines User Interfaces

Zeile 9 im Quelltext 4.18 zeigt beispielhaft die Einbindung der JavaScript-Datei aus Quelltext 4.19. Nach Beendigung des Ladevorgangs wird unter Verwendung des DOM-Manipulators *jQuery* [Fouc] dynamisch ein Titel-Element in das `<div>`-Element mit der ID *main* eingefügt.

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

Quelltext 4.19: JavaScript-Datei *app.js* zu Quelltext 4.18

Das ROCA Prinzip 17 *Know Structure* beschreibt den oben aufgezeigten Aufbau und erachtet es als wichtig, dass die Backendkomponente keine Kenntnis über das vom Frontendteil gerenderten User Interface hat. Das Backend soll lediglich die initiale Struktur des Grundgerüsts aus Quelltext 4.18 kennen und später nur noch als Datenlieferant via einer API dienen.

Geplante Umsetzung

Unter Berücksichtigung des Prinzips *RP14 Unobtrusive JavaScript*, näher beschrieben im Abschnitt 4.15, wird nicht geplant, *RP16 Know Structure* in seiner essentiellen Form innerhalb der Beispielapplikation *Roomies* zur Anwendung zu bringen.

Konkrete Umsetzung

Wie in Abschnitt 4.15 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 den Abschnitten 4.16 und 4.15 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 [AJWk], in welches sowohl im Front- als auch Backend die UI Elemente gerendert werden.

Diskussion

Separation of concerns [Wikl] 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 von *Know Structure*.

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.

4.18. RP17 Static Assets

Die Verwendung von JavaScript macht die Ausführung von dynamisch generiertem Quelltext einfach:

```
1 function evilFunction(message) {  
2     eval('alert(' + message + ')');  
3 }
```

Quelltext 4.20: Ausführung von dynamischem Quelltext mittels *eval*

Der Quelltext 4.20 ist nicht nur schwer wartbar, die Verwendung von *eval* [Mozb] stellt zusätzlich ein nicht abschätzbares Sicherheitsrisiko dar: Das übergebene String-Argument, welches den auszuführenden Quellcode enthält, kann von Angreifern problem-

los angepasst werden. Auf diese Weise ist es ohne grossen Aufwand möglich, Schadcode in eine Applikation einzuschleusen.

Die *Static Assets* ROCA Richtlinie will, dass jeglicher JavaScript Code in statischer Form vorliegt. Diese Anforderung erweitert sie zusätzlich auf alle CSS Formatierungsinformationen aus.

Geplante Umsetzung

CSS Stylesheets

Die verwendeten CSS Stylesheets sollen mit dem SASS Präprozessor [CWE] erstellt werden. Zwar muss der eigentliche Stylesheet Code im Voraus 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

Der JavaScript-Quelltext für den Client soll auf verschiedene Dateien aufgeteilt werden. Die Codefragmente in dieser Form an den Browser auszuliefern ist jedoch aus Gründen wie Verbindungsoverhead und der damit verbundenen schlechteren User Experience nicht optimal.

Aus diesem Grund sollen auf dem Backend alle Client-Quellcode-Dateien zusammengefasst 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 dies 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] gibt es zwar eine geteilte Codebasis für Client und Server. Jedoch stellt sich auch hier die Herausforderung, über verschiedene Dateien verteilte Programm-

logik in eine einzige, größenoptimierte JavaScript-Datei zusammenzufassen.

Zu diesem Zweck verwendet *barefoot* 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 wird eine eigenständige JavaScript-Datei erzeugt, welche als Ganzes an den Client ausgeliefert wird. Besonders für den produktiven Betrieb wichtig, kann diese zudem von Kommentaren und unnötigen Füllzeichen befreit und mittels Gzip [Wikh] komprimiert werden.

```

95 /* Browserify: */
96 // Roomies uses browserify to package all necessary 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 4.21: Konfiguration der browserify Middleware [AJWa]

Pseudo-dynamischer Quelltext mittels Funktionen

Rendert *barefoot* ein User Interface auf der Serverkomponente, verwendet es einen *DataStore* [Alab] um den Zustand aller beteiligten Models zum Client zu übertragen.

Der *DataStore* serialisiert seine Informationen dabei in ein JSON-Objekt welches wie in Quelltext 4.22 gezeigt in eine Funktion verpackt wird.

```

1 <script>function deserializeDataStore(){return {"currentUser":{ "dataStoreModelIdentifier": "ResidentModel", "data":/* ... */[{"name": "BA abgeben", "description": "", "reward": 5, "fulfilledAt": null, "dueDate": "2013-06-14T00:00:00.000Z", "id": 1, "createdAt": "2013-06-03T20:28:58.000Z", "updatedAt": "2013-06-03T20:28:58.000Z", "CommunityId": 1, "creatorId": 1, "fulfillorId": null}]};</script>
```

Quelltext 4.22: Ausschnitt eines von *barefoot* serialisierten *DataStores* im HTML Markup von *Roomies*

Wird *barefoot* resp. die implementierende Applikation auf dem Client gestartet, wird diese Funktion aufgerufen und damit der Zustand des *DataStores* rekonstruiert.

Zwar kann man hier tendenziell von dynamischen Code sprechen, in dieser Form wird aber kein dynamisch erzeugter String ausgewertet.

Diskussion

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

Im Bereich der clientseitigen JavaScript Entwicklung ermöglicht die erwähnte *browserify-middleware* oder andere Bibliotheken wie RequireJS [jrb] erst das effiziente Verteilen von Quellcode auf mehrere 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 immense Vereinfachung der Entwicklung von CSS-Stylesheets, welche mit verschiedenen Browsern kompatibel sind.

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.

4.19. RP18 History API

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

Geplante Umsetzung

Mithilfe von “Backbone.History” [Docc] 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 4.11 “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 4.23 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 4.23: Aktivierung des *History*-API's in barefoot [Alaf]

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. Wenn jemand eine URL mit dem Hashbang eintippte, mussten die Webapplikationen deshalb folgenden Workaround verwenden:

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 je länger je mehr darauf verzichtet (siehe z.B. den Blogpost von Twitter über die Abschaffung des Hashbangs [Twia]).

Die History API wird heute von allen modernen Browsern unterstützt. Man kommt also nicht mehr um diese API herum. Das Projektteam empfiehlt zudem, Hashbangs ohne zwingenden Grund nicht zu verwenden.

4.20. 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 zur gegenseitigen Interaktion definieren müssen.

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

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

Langfristig besteht die Gefahr, dass ein Flickwerk aus anwendungfallspezifischen Interfaces resp. Interfacemethoden entsteht.

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 deshalb auch der Grundsatz zu seiner Forderung, dass keine privaten Methoden existieren sollen.

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 4.2 “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 in Abschnitt 4.2 des Kapitels “Richtliniendemonstration” erläutert, konnte die generische Serviceschnittstelle für alle Objekte aus der *Roomies* Problemdomäne (siehe 3.1 “Domainmodell”) umgesetzt werden.

Es wurde komplett auf private Methoden verzichtet. Zum Schutz sensibler Daten wurde wie in den Abschnitten 4.8, 4.9 sowie 4.10 beschrieben ein Session-basierter Authentifizierungsmechanismus via Facebook (siehe 4.21 “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 abzuweichen.

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.

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

Viele Dienste im Internet verlangen heute die 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 sowohl 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 Termin- und Zeitangaben, Personalisierung der Frontseite usw.)
- Sicherstellung der Identität: Jedes Benutzerkonto resp. jeder Benutzername wird immer von derselben 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 selbst 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 omnipräsenzen sozialen Netzwerke erleichtern resp. forcieren eben diese Auf trennung 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 und welche nicht.

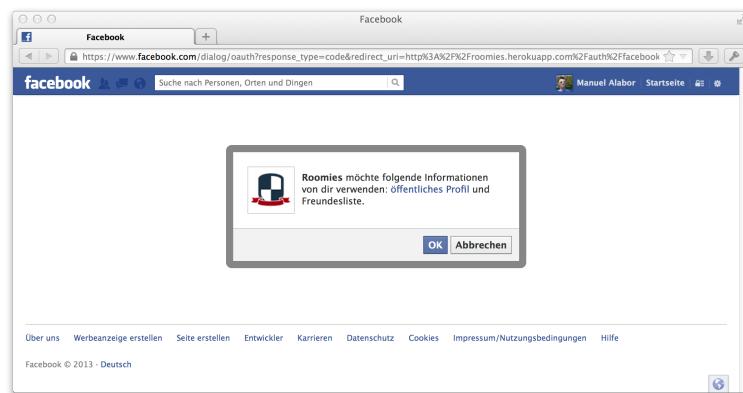


Abbildung 4.7.: Bestätigungsdialog von Facebook, ob *Roomies* auf die persönlichen Daten des Benutzers zugreifen darf.

Visualisiert sieht das Konzept der getrennten Datenhaltung in Bezug auf applikations- und identitätsspezifische Informationen wie in Abbildung 4.8 aus.

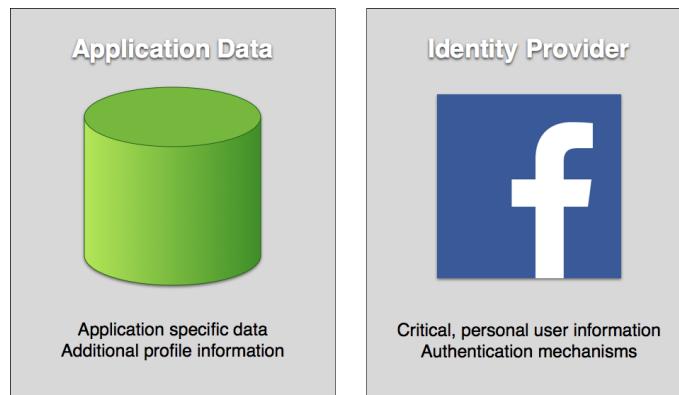


Abbildung 4.8.: Separierung der Applikations- und Identitätsinformationen

Zudem ist in Abbildung 4.8 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 Implementierungen.

Geplante Umsetzung

Für die Beispielapplikation *Roomies* hat das Projektteam geplant, Facebook als hauptsächlichen und einzigen *Identity Provider* 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.

Konkrete Umsetzung

Die tatsächliche Anbindung des “*Facebook Login for Web*” [Faca] SDK’s wurde wie in Abschnitt 3.3 “Software Layers” des Kapitels “Technische Architektur” beschrieben, mithilfe der quelloffenen Bibliothek *Passport.js* [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 der Name des 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 4.9.: Facebook Profilbild des angemeldeten Benutzers der Menüleiste der Beispiapplikation *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.js* [Hanb] verwendet werden können.

So ist es ohne grössere Zusatzaufwände möglich (eine entsprechende Architektur vorausgesetzt) 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 bereits gemachte Anbindung an Facebook.

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

- Datenschutz
- Potentielle Akzeptanz bei Benutzern
- Verbreitung spezifischer *Identity Providers* bei Benutzern
- Usability & User Experience

- Neue oder bestehende Applikation?

Gerade bei 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 wettgemacht 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.

4.22. TP7 Apply the Web instead of working around

Der moderne Internetbrowser kapselt eine Vielzahl von leistungsfähigen Funktionen des jeweiligen Hostrechners in ein für den Software Entwickler leicht zu verwendendes Interface. Die Integration von systemnahen Komponenten wie GPU's [Netc] gehört dabei genauso dazu wie die Möglichkeit, gleichzeitig mehrere Fenster oder Tabs für dieselbe oder auch verschiedene Applikationen resp. Internetseiten offen zu halten.

Das Hin- und Herspringen zwischen besuchten Seiten mittels Vorwärts- und Zurück-Schaltflächen gehört seit Beginn der Web-Ära 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 Farbverläufen [Mozf]) 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., siehe auch 4.12 “RP11 POSH”)
- 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 *Responsive Design* resp. den zugrundeliegenden *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, wie von 4.19 "RP18 History API" bereits gefordert, zwingend darauf geachtet werden, dass die Verwendung der Browser-Funktionen *Vorwärts*, *Zurück* und *Aktualisieren* zu keinem unerwarteten Verhalten führt.

Konkrete Umsetzung

HTML Markup

Wie geplant verwendet das gerenderte HTML Markup die vom HTML 5 Standard eingeführten Funktionen. Quelltext 4.24 zeigt die Verwendung des `<header>` sowie `<nav>` Tags zur Beschreibung des Applikationsmenüs der Beispielapplikation.

```
1 <header id="menu">
2   <div class="fixed-navigation">
3     <nav class="navigation" role="navigation">
4       <div class="title-area">
5         <a href="/" class="banner" title="Roomies"><h1>Roomies</h1></a>
6       </div>
7       <section class="nav-section">
8         <ul class="left">
9           <li>
10             <a href="/community/ba-team/tasks" title="Aufgaben">
11               <i class="icon-tasks icon-large"></i>
12               <span class="item-label"> Aufgaben</span>
13             </a>
14           </li>
15           <!-- ... more items -->
16         </ul>
17         <!-- ... displaying the facebook profile picture -->
18       </section>
19     </nav>
20   </div>
21 </header>
```

Quelltext 4.24: 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 4.10.: Datumsauswahl für ein Textfeld vom Typ *date* in *Safari für iPhone*

CSS

Der erstellte CSS Quellcode macht von den neusten CSS 3 Features ausführlichen Gebrauch. So wird für die Darstellung von visuellen Effekten oft auf *box-shadow* oder halb-transparente 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 4.25: Einbindung des `@border-radius` Mixins von *Bourbon* [AJWo]

Im Bereich *Responsive Design* wurde keine Lösung von Grund auf selbst 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 der Endgeräte reagieren kann.

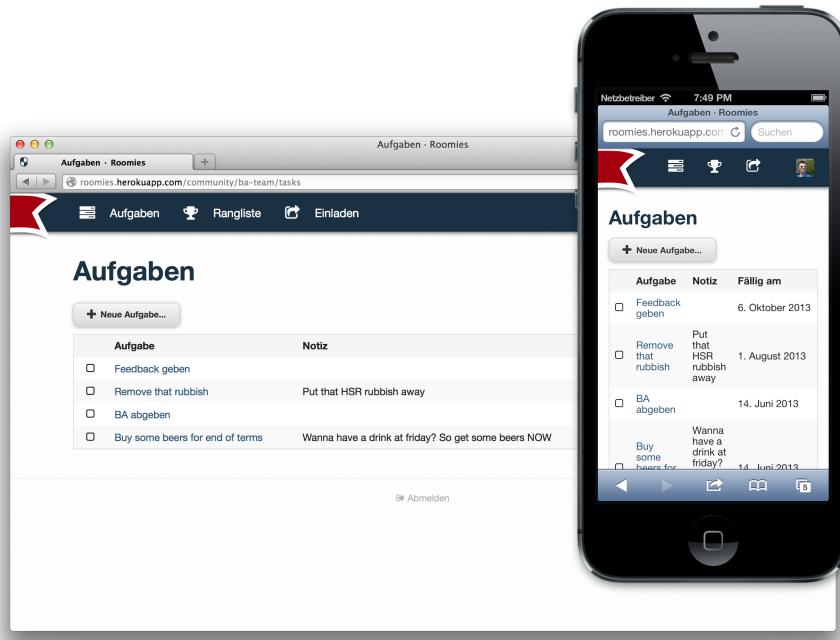


Abbildung 4.11.: Media Queries ermöglichen dynamische Anpassung des Layouts entsprechend der verfügbaren Bildschirmgröße

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 in Abschnitt 4.11 “RP10 Browser-Controls” sowie 4.19 “RP18 History API” enthalten.

iOS Webapp Kompatibilität

Zusätzlich zu den geplanten Features wurde die von Apple definierte Spezifikation für *Web Applications* [Inca] 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 4.12) als auch angepasste Ladebildschirme während dem *Aufstarten* der Applikation.



Abbildung 4.12.: *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 Helper 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 aber trotzdem 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.

Aus diesem Grund ermutigt das Projektteam neuste Funktionalitäten zu verwenden, ermahnt jedoch, immer eine Fallbacklösung bereitzuhalten, sollte ein Feature auf einem Browser nicht verfügbar sein.

4.23. 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, zum anderen werden auch wiederkehrende Routineaufgaben automatisiert.

Entwickelt man den „*Don't repeat yourself*“-Ansatz weiter, so landet man unweigerlich bei der Verwendung von automatisierten Tests und Deployments mittels Continuous Integration Systemen.

Geplante Umsetzung

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

Tabelle 4.1 zeigt eine Auflistung der zu automatisierenden Aufgaben:

ID	Aufgabe
TP8.1	Starten der Beispielapplikation
TP8.2	Ausführung der Unit Tests
TP8.3	Formale Überprüfung des Quellcodes
TP8.4	Umwandlung von SASS zu CSS Stylesheets
TP8.5	Erstellung von Quellcode Dokumentation

Tabelle 4.1.: Automatisierte Aufgaben (geplant)

Konkrete Umsetzung

Neben den oben erwähnten Aufgaben wurden in der konkreten Implementation des Projektes weitere Tasks erfolgreich automatisiert:

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

Tabelle 4.2.: Automatisierte Aufgaben (umgesetzt)

Als Kernkomponente für die Automatisierung der Aufgaben in Tabelle 4.2 wird ein *Makefile* ([AJWc] und [AJWb]) verwendet. Dieses wird von *GNU Make* [Frea] interpretiert und ermöglicht so das Ausführen verschiedenster Operationen.

Der Quelltext 4.26 zeigt exemplarisch die Befehlsdefinition für die Erstellung der Quellcode Dokumentation mittels *Natural Docs* [DuVb].

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

Quelltext 4.26: Ausschnitt Makefile: Code Dokumentation erstellen [AJWc]

Durch den einfache Kommandozeilenbefehl *make docs* wird der in 4.26 definierte Befehl ausgeführt.

Continuous Integration

Die in Tabelle 4.2 vorgestellten Aufgaben TP8.2 bis 8.8 werden sowohl lokal auf dem Entwicklerrechner als auch auf dem Continuous Integration System ausgeführt.

Entsprechend der Definition in Kaptiel J.6 “Qualitätsmanagement” wird hierzu die Open Source Plattform Travis CI [CI] verwendet. Der Quelltext 4.27 zeigt einen Ausschnitt der Datei *.travis.yml* [AJWv]. 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 4.27: Ausschnitt aus .travis.yml [AJWv]

Diskussion

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

Die Praxis zeigt, dass Zeitersparnis und Effektivitätssteigerung die positiven Folgen der Automatisierung von Routineaufgaben sind. 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.

Kapitel 5 **Schlussfolgerung**

Roll Up

In den vorangegangenen drei Kapiteln hat sich das Projektteam ausführlich mit verschiedenen Aspekten moderner Webapplikationen befasst. Es wurde eine Zusammenstellung von Architekturrichtlinien analysiert und dabei jedes Konzept aus unterschiedlichen Perspektiven sowohl theoretisch als auch praktisch bewertet.

Nach der Evaluation dreier Technologiekandidaten und mehrerer Bibliotheken wurde unter Verwendung von *JavaScript* und *Express.js* mit *Roomies* eine Applikation zur Veranschaulichung der ausgewählten Konzepte entwickelt.

Mit den ersten Arbeiten an der Beispielmöglichkeit wurde dem Projektteam schnell klar, dass die Anforderungen der *Unobtrusive JavaScript*-Richtlinie einiges mehr an Aufwand verschlingen würden als zuerst angenommen wurde. Die Anstrengungen in diesem Gebiet wurden mit dem wiederverwendbaren Framework *barefoot* belohnt: Unter dessen Integration kann *Roomies*' Quellcode ohne mehrfache Implementation sowohl auf dem Server als auch auf dem Client verwendet werden.

Fazit

Architekturrichtlinien

Aus über 25 Richtlinien, Architekturkonzepten und Prinzipien wurden während der Vorselektion insgesamt 22 ausgewählt, um in einem weiteren Schritt am praktischen Beispiel *Roomies* demonstriert zu werden.

Bis auf einige wenige Ausnahmen konnten alle Konzepte befriedigend implementiert werden. Dabei stechen *No Duplication* und *Unobtrusive JavaScript* aus dem ROCA-Katalog besonders hervor:

Während der Implementation von *Roomies* flossen über 40 Prozent des Gesamtaufwandes in die Umsetzung dieser Richtlinien. Insbesondere die Integration bestehender Codefragmente mit *barefoot* verschlang immens Zeit.

Die Qualität von *barefoot* hat von diesem Prozess hingegen sehr profitiert. In dessen aktueller Version *0.0.11* sind viele Fehler behoben und wichtige Features wurden ergänzt oder neu konzipiert.

Negativ aufgefallen sind lediglich die Konzepte *Auth* und *Know Structure*: Letzteres

war mit der Umsetzung von *Unobtrusive JavaScript* nicht vereinbar, da diese beiden Prinzipien sich widersprechen. Erstes wurde gezielt nicht umgesetzt, um den Workload auf das Team zu optimieren.

Gesamtheitlich schätzt das Projektteam die bearbeiteten Architekturempfehlungen als zweckmäßig und sehr hilfreich ein. Vielerorts muss jedoch wie dokumentiert von Situation zu Situation entschieden werden, welchen Konzepten aufgrund der vorliegenden Anforderungen der Vorzug gewährt wird.

Tauglichkeit des für Unterrichtmodul Internettechnologien

Seit der gemeinsamen Definition der Aufgabenstellung gehörte die Idee, neue Vorschläge für das Unterrichtmodul *Internettechnologien* zu liefern, fest zur Idee dieser Bachelorarbeit dazu.

Neben der praktischen Demonstration sollte insbesondere auch der theoretische Teil der Arbeit darüber Aufschluss geben, inwiefern die neuen, zu untersuchenden Architekturkonzepte für den Unterricht und damit den Entwickleralltag tauglich sind.

Mit der Beispielapplikation *Roomies* sind alle Richtlinien bis auf zwei Ausnahmen praktisch umgesetzt und demonstriert worden. Das vorliegende Produkt verfügt über eine klare und durchgehende Komponentenarchitektur. Das Projektteam ist davon überzeugt, dass Aspekte wie die *REST*-basierte API oder das umgesetzte *MVC*-Pattern für das User Interface Mustergültigkeit haben.

Im grösseren Zusammenspiel, insbesondere in Kombination mit dem *Express.js Middleware*-Konzept sowie dem experimentellen *Code Sharing*-Ansatz *barefoots*, ergibt sich eine komplexe Kombination der für sich einfachen Komponenten. Das Projektteam ist sich darum nicht sicher, in wie weit die Beispielapplikation als ganzes tatsächlich in den Unterricht integriert werden kann.

Die vollständige Analyse der Richtlinien im Kapitel 4 "Richtliniendemonstration" bietet eine optimale Grundlage für das Übertragen der untersuchten Prinzipien in den Unterricht. Ausführliche Quellenangaben und Querverweise erleichtern zudem das Erarbeiten weiterführender Themen. So hält das Projektteam diesen Teil des Arbeitsergebnisses für uneingeschränkt wertvoll und wiederverwendbar.

Kontroverse: JavaScript

Während der Technologieevaluation war umstritten, ob *JavaScript* wirklich für den geplanten Einsatz vollenfänglich geeignet sei. Zwar wurden bereits Lösungen mit *Node.js* für den produktiven Einsatz umgesetzt und auch das Projektteam hat in der Studienarbeit positive Erfahrung auf der Clientseite machen können. Trotzdem waren altbekannte Unsicherheiten teils weiterhin sowohl im Betreuer- als auch Projektteam spürbar:

- Seriöses Software Engineering (*Separation Of Concerns*, TDD, CI etc.) ist mit *JavaScript* nicht möglich
- Schlechte serverseitige Performance

- Schlechte Strukturierung von Quellcode (*Callback Hell* etc.)

Die gemachten Erfahrungen mit der Umsetzung von *Roomies* beweisen, dass *JavaScript* sich sehr wohl mit bekannten Technologien wie *Ruby* messen kann. Gerade in den Belangen Flexibilität und Skalierbarkeit braucht es sich keinesfalls zu verstecken.

Werkzeuge wie *JSHint*, *Mocha*, *Travis CI* oder *JSCoverage* ermöglichen die Sicherstellung der Codequalität von *JavaScript* Quelltext-Artefakten.

JavaScript ist definitiv seinen Kinderschuhen entwachsen und wird heute rege weiterentwickelt. Das Projektteam hat die Entscheidung für *JavaScript* keine Sekunde bereut und kann auch nach erfolgreicher Durchführung dieser Arbeit voll und ganz hinter dieser Entscheidung stehen.

Ausblick

Kapitel 6 **Persönliche Berichte**

6.1. Manuel Alabor

Dokumentation

Implementation

Fazit

6.2. Alexandre Joly

Dokumentation

Implementation

Fazit

6.3. Michael Weibel

Die letzten 16 Wochen Bachelorarbeit waren vollgepackt mit einem interessanten Thema, viel Arbeit und interessanten Diskussionen. Hier ein Überblick über meine persönliche Erfahrung.

Evaluation

Nach der initialen Themenfindung war schnell klar, dass wir eine Beispielapplikation benötigen um die Architekturrichtlinien zu zeigen.

Ich widmete mich dem Thema JavaScript, welches schlussendlich aufgrund der Kriterien und der persönlichen Interessen des Projektteams und des Betreuers Hans Rudin ausgewählt wurde. Dies hat mich gefreut, schliesslich befasse ich mich schon länger intensiv mit JavaScript und Node.js.

Die Evaluationphase und deren Kriterien für die Auswahl der Sprache und des Frameworks war unter anderem aus Zeitgründen mit persönlichen Vorlieben geprägt. In einem anderen Projekt müssten wohl die Kriterien spezifischer gewählt werden und eine grössere Bandbreite an Technologien mit einem Proof of Concept testen.

Beispielapplikation Roomies

Als nächstes kam ein grosser Teil Implementation. Bald wurde klar, dass sich das Projektteam in zwei Teile aufteilen wird, um einerseits die Use Cases zu implementieren und andererseits ein Code Sharing Framework zu entwickeln.

Ich war im Team zusammen mit Alexandre Joly um die Use Cases zu implementieren. Wir kamen relativ gut voran, jedoch haben wir zuwenig auf eine entkoppelte API geachtet. Wenn wir das gemacht hätten, wäre der spätere Umstieg auf "barefoot" wohl einfacher und schneller vonstatten gegangen.

Insgesamt war die Implementation aber gut und wir konnten eine Menge lernen. Das entstandene Framework "barefoot" ist sehr interessant und ich könnte mir vorstellen, dies auch bei anderen Projekten einzusetzen.

Architekturrichtlinien

Die Architekturrichtlinien sind meiner Meinung nach definitiv hilfreich und ein grosser Teil wird wohl an vielen Orten unbewusst umgesetzt.

Die Richtlinien an sich waren für mich nicht neu. Viele davon sind mir aufgrund meiner Erfahrung klar. Die intensive Beschäftigung damit gibt jedoch eine neue Sicht darauf und ein gutes Vokabular.

Es gibt einen Punkt, welchen ich an den Richtlinien bemängle: Viele davon sagen eigentlich dasselbe aus oder sind zumindest sehr eng verwandt. Natürlich kann argumentiert werden, dass diese eng verwandten Richtlinien entsprechend viele verschiedene Sichten auf sehr ähnliche Probleme geben. Dennoch denke ich, dass die Richtlinien insgesamt gekürzt werden könnten.

Fazit

Insgesamt war die Bachelorarbeit lehrreich für mich. Dies insbesondere auch bezüglich der Serverseitigen Verwendung von JavaScript.

Die Zusammenarbeit im Team war wie schon in der Studienarbeit sehr gut. Die Zusammenarbeit mit den Betreuern war ebenfalls sehr angenehm und produktiv.

Anhang A **Abbildungen, Tabellen & Quelltexte**

Abbildungsverzeichnis

1.1.	Aufgabenverwaltung <i>Roomies</i> zur Konzeptdemonstration	10
2.1.	Mapping Architektrichtlinien - Systemkomponenten	17
2.2.	Roomies Logo im College Stil	18
2.3.	Berechnungsformel Gesamtbewertung	20
3.1.	Architektur	32
3.2.	MVC-Komponenten im Zusammenspiel	36
3.3.	Domainmodel	37
3.4.	Entity-Relationship Diagramm	39
3.5.	Software Layers – Technologieneutral	40
3.6.	Software Layers – Konkrete Implementation	42
3.7.	“Roomies”: Logische Komponenten	44
3.8.	Sequenzdiagramm: Rendering und Event-Verarbeitung auf dem Server .	45
3.9.	Sequenzdiagramm: Rendering und Event-Verarbeitung auf dem Client .	46
3.10.	Sequenzdiagramm: APIAdapter im server-lokalen sowie HTTP REST Modus	47
3.11.	Ordnerstruktur des <i>src</i> -Unterordners der Beispielapplikation <i>Roomies</i> .	48
4.1.	Übersicht Architektrichtlinienanalyse (1/2)	54
4.2.	Übersicht Architektrichtlinienanalyse (2/2)	55
4.3.	Fehlerhafte Darstellung im Internet Explorer 8	75
4.4.	Kategorisierung aktueller Webapplikationen	76
4.5.	<i>Google Drive</i> in Firefox 21.0 mit deaktiviertem JavaScript	77
4.6.	<i>Rendr</i> vs. <i>barefoot</i>	80
4.7.	Bestätigungsdialog von Facebook, ob <i>Roomies</i> auf die persönlichen Daten des Benutzers zugreifen darf.	90
4.8.	Separierung der Applikations- und Identitätsinformationen	90
4.9.	Facebook Profilbild des angemeldeten Benutzers der Menüleiste der Beispielapplikation <i>Roomies</i>	91
4.10.	Datumsauswahl für ein Textfeld vom Typ <i>date</i> in <i>Safari für iPhone</i> . .	94

4.11. Media Queries ermöglichen dynamische Anpassung des Layouts entsprechend der verfügbaren Bildschirmgrösse	95
4.12. <i>Roomies</i> als Webapp auf dem iPhone Homescreen	96
H.1. Branding Farbpalette	135
H.2. Roomies Logo im College Stil	136
H.3. Roomies Logo in verschiedenen Grössen & Varianten	136
I.1. Use Case Diagramm	140
J.1. Iterationsübersicht mit Meilensteinen, Kalenderwochen Februar bis Juli 2013	147
L.1. Burndown	155
L.2. Zeitaufwand pro Aufgabenbereich	156
L.3. Zeitaufwand pro Iteration	157
M.1. Roomies – Anmelden	158
M.2. Roomies – Aufgaben	159
M.3. Roomies – Aufgabe erstellen	159
M.4. Roomies – Aufgabe bearbeiten	160
M.5. Roomies – Rangliste	160
M.6. Roomies – Einladen	161
M.7. Roomies – Bewohnerprofil	161

Tabellenverzeichnis

2.1. Die ROCA Architekturprinzipien: Backend	13
2.2. Die ROCA Architekturprinzipien: Frontend	14
2.3. Tilkovs Empfehlungen	15
2.4. Bewertungskriterien für Technologieevaluation	20
2.5. Bewertung Ruby on Rails	21
2.6. Shortlist Analysekandidaten Java (1/2)	22
2.7. Shortlist Analysekandidaten Java (2/2)	23
2.8. Bewertungsmatrix Java Frameworks	23
2.9. Shortlist Analysekandidaten JavaScript (1/2)	24
2.10. Shortlist Analysekandidaten JavaScript (2/2)	25
2.11. Bewertungsmatrix JavaScript Frameworks	25
2.12. Finale Frameworkkandidaten für Technologieentscheidung	26
2.13. Bewertungskriterien für ORM-Evaluation	31
2.14. Bewertungsmatrix JavaScript ORMs	31
4.1. Automatisierte Aufgaben (geplant)	97

4.2.	Automatisierte Aufgaben (umgesetzt)	98
D.1.	Projektrelevante URL's	126
H.1.	Produktideenpool	134
I.1.	Funktionale Anforderungen	138
I.2.	Nichtfunktionale Anforderungen	139
I.3.	Akteure	141
I.4.	UC1: Anmelden	141
I.5.	UC2: WG erstellen	141
I.6.	UC3: WG beitreten	142
I.7.	UC4: WG verlassen	142
I.8.	UC5: Aufgabe erstellen	142
I.9.	UC6: Aufgabe bearbeiten	143
I.10.	UC7: Aufgabe erledigen	143
I.11.	UC8: Rangliste anzeigen	143
I.12.	UC9: WG auflösen	144
I.13.	UC10: Benutzer verwalten	144
I.14.	UC11: auf Social Media Plattform teilen	144
I.15.	Resultate Use Case Umsetzung	145
J.1.	Projektierungsbeschreibung	146
J.2.	Meilensteine	148
J.3.	Abzugebende Artefakte	149
K.1.	Iterationsassessment	153
L.1.	Zeitrapport - Kalenderwochen 7-13	154
L.2.	Zeitrapport - Kalenderwochen 14-19	154
L.3.	Zeitrapport - Kalenderwochen 20-24	154

Quellcodeverzeichnis

2.1.	Negierte if-Abfrage in Java	21
2.2.	Negierte if-Abfrage in Ruby	21
2.3.	Task Model in Sails.js	27
2.4.	Task Controller in Sails.js	28
2.5.	Task Template	28
2.6.	Beispiel eines Controllers in Express.js	29
2.7.	Template in Express.js	30
3.1.	Auszug aus Router der Beispielapplikation [AJWq]	32

3.2.	Ausschnitt aus HomeView der Beispielapplikation [AJWu]	33
3.3.	API-Controller Beispiel [AJWg]	34
3.4.	API-Route Beispiel [AJWi]	35
3.5.	Auszug der verwendeten NPM Komponenten [AJWd]	48
3.6.	Einbindung der Community-Komponente	49
3.7.	Ermittlung der Runtime-Umgebung sowie anschliessendes applizieren der spezifischen Mixins [Alac]	49
3.8.	Ausschnitt aus Community Controller mit Callback Hell [AJWf]	50
3.9.	Ausschnitt aus dem neusten Community Controller [AJWh]	51
3.10.	Ausschnitt aus Server router-mixin.js [Alad]	52
4.1.	Community API Definition [AJWi]	57
4.2.	Community Model [AJWp]	60
4.3.	Community Model Synchronisation [AJWt]	61
4.4.	HTTP Middleware [AJWm]	62
4.5.	Formular mit verstecktem <code>_method</code> Feld [AJWs]	62
4.6.	cURL Request auf Roomies	64
4.7.	Community API <code>getCommunityWithSlug</code> [AJWg]	65
4.8.	Connect Session Middleware [AJWn]	68
4.9.	Router - Autorisationskontrolle [AJWq]	69
4.10.	Aktivierung der History in Barefoot [Alaf]	71
4.11.	Layout Definition [AJWk]	72
4.12.	Formularfeld mit HTML5, welches eine Telefonnummer erwartet	74
4.13.	Einbinden von modernizr [AJWI]	75
4.14.	Beispiel einer Vermischung von HTML Markup und JavaScript	77
4.15.	Beispiel eines sauberen HTML Markups ohne JavaScript	78
4.16.	Beispiel Event-Handler in ausgelagerter JavaScript Datei	78
4.17.	Ausschnitt aus dem <i>Handlebars</i> [Kat] Template zur Darstellung von Benutzerinformation in der Menüleiste von <i>Roomies</i> [AJWr]	79
4.18.	Beispiel eines HTML Gerüsts zum Rendering eines User Interfaces	82
4.19.	JavaScript-Datei <code>app.js</code> zu Quelltext 4.18	82
4.20.	Ausführung von dynamischem Quelltext mittels <code>eval</code>	83
4.21.	Konfiguration der browserify Middleware [AJWa]	85
4.22.	Ausschnitt eines von <i>barefoot</i> serialisierten <i>DataStores</i> im HTML Markup von <i>Roomies</i>	85
4.23.	Aktivierung des <i>History</i> -API's in <i>barefoot</i> [Alaf]	86
4.24.	Ausschnitt des gerenderten HTML Markups der Menüleiste <i>Roomies</i>	93
4.25.	Einbindung des <code>@border-radius</code> Mixins von <i>Bourbon</i> [AJWo]	94
4.26.	Ausschnitt Makefile: Code Dokumentation erstellen [AJWc]	98
4.27.	Ausschnitt aus <code>.travis.yml</code> [AJWv]	98
E.1.	Installationsanleitung Vagrant	127
E.2.	Installationsanleitung ohne Vagrant	128

J.1.	Beispiel eines Unit Tests mit Mocha und Chai.js	151
N.1.	Führende Kommas - Coding Guidelines [Aira]	162
N.2.	Tabulatoren - Coding Guidelines [Aira]	163

Anhang B **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]. 76

Benutzer

Ein Benutzer verwendet “Roomies” und ist in den meisten Fällen ein Bewohner..
138

Bewohner

Ein Bewohner ist einer WG zugeordnet.. 110, 138

Boilerplate

Codefragmente, welche in unveränderter Form immer wieder verwendet werden können. 85

CI

Continuous Integration. 55, 126

CLI

Kurzform für *Command Line Interface*, Kommandozeile oder Terminal. 98

CRUD

Abkürzung für “Create, Read, Update and Delete”; Die Zusammenfassung der Datenmanipulationsoperationen. 112

DOM

Document Object Model, Repräsentation eines, HTML Dokuments im Speicher.
76, 77, 82

Don't repeat yourself

DRY ist eines der wichtigsten Software-Entwicklungs Prinzipien und verfolgt den Ansatz, den gleichen Code nicht zu duplizieren.. 59, 72, 97

Frames

Ein Frame ist ein Teilbereich einer HTML-Seite, in dem eine andere HTML-Seite dargestellt werden kann. *Secure Socket Layer* [Wikf]. 71

Gamification

Als Gamification oder Gamifizierung (seltener auch Spielifizierung) bezeichnet man die Anwendung spieltypischer Elemente und Prozesse in spielfremdem Kontext [Det+]. 18, 135, 146

GPU

Graphical Processing Unit, Prozessor auf einer Grafikkarte. 92

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 aufgrund dessen eine Möglichkeit gefunden, um Hashbang-URLs zu crawlen (siehe [Gooc]). Nach der Entwicklung der History API haben viele Webseiten dies aber wieder komplett abgeschafft (u.a. Twitter [Twia]).. 86, 87

JSON

JavaScript Object Notation. 56, 85, 88

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 [Wiki].. 68

Middleware

Middlewares in Express.js [Holf] werden verwendet, um Funktionalitäten wie Cookies, Sessions etc. einer Applikation hinzuzufügen. Dies erlaubt es, jede Applikation genau auf die benötigten Funktionalitäten zu beschränken.. 24, 41, 68

Mixin

Als Mixin wird in der objektorientierten Programmierung ein zusammengehöriges, mehrfach verwendbares Bündel von Funktionalität bezeichnet, das zu einer Klasse hinzugefügt werden kann. [Wikj]. 49, 50, 80, 108

Multiple Inheritance

Multiple Inheritance bezeichnet ein Konzept, welches es erlaubt, eine Klasse 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. 19, 24, 25, 43

NoSQL

NoSQL Datenbanken sind eine relativ neue Art, um Datenbanken ohne SQL-Sprache zu entwickeln. Vielfach haben solche Datenbanken keine Relationen im Sinne der Relationalen Datenbanken implementiert.. 31

ORM

Ein “Object Relational Mapper” wird verwendet um Entitäten auf einer Relationalen Datenbank abzubilden und verwenden zu können. 25, 27, 30, 41

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. [Wikd]. 11, 27

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 [Wikd] sind solche Applikationen im Web realisierbar. 25–27

REST

Representational State Transfer, definiert von Roy Fielding in seiner Dissertation [Fie00]. 27, 45, 56, 58, 60, 63–65, 88

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.. 60, 61

RUP

Rational Unified Process; Iteratives Projektvorgehen [IBM]. 146

SAD

Software Architektur Dokument. 148, 149

Scaffolding

Scaffolding bezeichnet das toolunterstützte Generieren von Quellcodefragmenten. Beispiel: Erstellung einer Model-Klasse inkl. dazugehörigem CRUD-Controller. 21, 22, 24, 27

SDK

Software Development Kit, eine Code-Bibliothek eines Anbieters um dessen Dienste in einer eigenen Applikation anzubinden.. 90, 91

Search Engine Spider

Computerprogramme, welche vor allem von Suchmaschinen verwendet werden, um Webseiten zu durchsuchen und zu analysieren.. 71

SSL

Abkürzung für *Secure Socket Layer* [FPP]. 67

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.. 101, 151

URI

Uniform Resource Identifier [Wikm], identifiziert eine abstrakte oder physische Ressource.. 56, 70, 113

URL

Uniform Resource Locator [Wikn], identifiziert z.B. eine Webseite in einem Netzwerk. Oft ein Synonym für *Internetadresse*. Es ist eine Unterart von URI.. 63, 69, 70, 87, 91

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.. 111

VM

Virtual Machine. 149

WebRTC

Web Real-Time-Communication [Gooe] 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 Anwendungen (inbesondere aber Audio/Video-Konferenz-Software und Spiele).. 60

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, 27, 112

WG

Eine WG ist eine Wohngemeinschaft und kann auf “Roomies” eröffnet werden. . 18, 38, 110, 134, 135, 138

Anhang C 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 (Updated)*. URL: <https://github.com/hsr-ba-ajw-2013/javascript-style-guide> (besucht am 10.06.2013).
- [Airb] Airbnb. *Airbnb JavaScript Style Guide (Updated)*. URL: <https://github.com/timofurrer/javascript-style-guide> (besucht am 10.06.2013).
- [Airc] Airbnb. *Airbnb JavaScript Style Guide (Updated)*. URL: <https://github.com/airbnb/javascript> (besucht am 10.06.2013).
- [Aird] Airbnb. *airbnb/rendr*. URL: <https://github.com/airbnb/rendr> (besucht am 10.06.2013).
- [Aire] Airbnb. *Underscore.js*. URL: <http://underscorejs.org/> (besucht am 10.06.2013).
- [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/package.json at master*. URL: <https://github.com/hsr-ba-ajw-2013/BA/blob/master/package.json> (besucht am 09.06.2013).
- [AJWe] 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).

- [AJWf] Manuel Alabor, Alexandre Joly und Michael Weibel. *BA/src/lib/community/controller.js at 5d62b8699b64f7a919ecd6fa9c5f7b4f0f50827b*. URL: <https://github.com/hsr-ba-ajw-2013/BA/blob/5d62b8699b64f7a919ecd6fa9c5f7b4f0f50827b/src/lib/community/controller.js#L136> (besucht am 13.06.2013).
- [AJWg] 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/controller.js> (besucht am 05.06.2013).
- [AJWh] 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/controller.js#L192> (besucht am 13.06.2013).
- [AJWi] 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).
- [AJWj] 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).
- [AJWk] 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).
- [AJWl] 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).
- [AJWm] 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).
- [AJWn] 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).
- [AJWo] 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).
- [AJWp] 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).
- [AJWq] 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).

- [AJWr] 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).
- [AJWs] 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).
- [AJWt] 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).
- [AJWu] 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).
- [AJWv] 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.DataStore*. URL: <http://swissmanu.github.io/barefoot/docs/files/lib/datastore-js.html> (besucht am 09.06.2013).
- [Alac] Manuel Alabor. *barefoot/lib/index.js*. URL: <https://github.com/swissmanu/barefoot/blob/master/lib/index.js#L68> (besucht am 10.06.2013).
- [Alad] Manuel Alabor. *barefoot/lib/server/router-mixin.js at master*. URL: <https://github.com/swissmanu/barefoot/blob/master/lib/server/router-mixin.js#L215> (besucht am 13.06.2013).
- [Alae] Manuel Alabor. *Barefoot.Model*. URL: <http://swissmanu.github.io/barefoot/docs/files/lib/model-js.html> (besucht am 09.06.2013).
- [Alaf] Manuel Alabor. *Barefoot.Start.Client Source*. URL: <https://github.com/swissmanu/barefoot/blob/master/lib/client/start.js> (besucht am 05.06.2013).
- [Alag] 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. *encryption - MD5 security is fine? - Stack Overflow*. URL: <http://stackoverflow.com/questions/6774345/md5-security-is-fine> (besucht am 11.06.2013).
- [Comc] 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).
- [Comd] 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).
- [Cov] Coveralls. *Coveralls - Code Coverage History and Statistics*. URL: <https://coveralls.io/> (besucht am 10.06.2013).
- [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. *Backbone.js - Backbone#sync*. URL: <http://backbonejs.org/#Sync> (besucht am 05.06.2013).
- [Docc] DocumentCloud. *Backbone.js - History*. URL: <http://backbonejs.org/#History-start> (besucht am 05.06.2013).
- [DuVa] Adam DuVander. *1 in 5 APIs Say “Bye XML”*. URL: <http://blog.programmableweb.com/2011/05/25/1-in-5-apis-say-bye-xml/> (besucht am 09.06.2013).
- [DuVb] Adam DuVander. *Natural Docs*. URL: <http://www.naturaldocs.org/> (besucht am 09.06.2013).
- [Eat] Alf Eaton. *Login to Facebook using cURL*. URL: <https://gist.github.com/hubgit/306638> (besucht am 10.06.2013).
- [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).
- [FPP] A. Freier, Netscape Communications P. Karlton und Independent Consultant P. Kocher. *The Secure Sockets Layer (SSL) Protocol Version 3.0*. URL: <http://tools.ietf.org/html/rfc6101> (besucht am 11.06.2013).
- [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 Geisendorfer. *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. *Google Web Toolkit - Google Developers*. URL: <https://developers.google.com/web-toolkit> (besucht am 09.06.2013).
- [Gooc] Google. *Making AJAX Applications Crawlable*. URL: <https://developers.google.com/webmasters/ajax-crawling/> (besucht am 06.06.2013).
- [Good] Google. *V8 JavaScript Engine*. URL: <https://code.google.com/p/v8/> (besucht am 16.03.2013).
- [Gooe] 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. *Express API Reference - res.send*. URL: <http://expressjs.com/api.html#res.send> (besucht am 07.06.2013).
- [Holf] TJ Holowaychuk. *Express - node.js web application framework*. URL: <http://expressjs.com/> (besucht am 27.02.2013).
- [Holg] TJ Holowaychuk. *express/lib/response.js at master - visionmedia/express*. URL: <https://github.com/visionmedia/express/blob/master/lib/response.js%5C#L117> (besucht am 12.06.2013).
- [Holh] TJ Holowaychuk. *Mocha*. URL: <http://visionmedia.github.com/mocha/> (besucht am 17.03.2013).
- [Holi] TJ Holowaychuk. *TJ Holowaychuck*. URL: <http://tjholowaychuk.com/> (besucht am 14.04.2013).
- [HST] Falk Hoppe, Till Schulte-Coerne und Stefan Tilkov. *ROCA: Resource-oriented Client Architecture*. URL: http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2012/Architekturen/hoppe_schulte-coerner_tilkov_OS_Architekturen_12_hgz8.pdf (besucht am 11.06.2013).
- [IBM] IBM. *IBM Rational Unified Process (RUP)*. URL: <http://www-01.ibm.com/software/awdtools/rup/>.
- [IIC] Google Inc., Yahoo Inc. und Microsoft Corporation. *Schema.org*. URL: <http://schema.org> (besucht am 12.06.2013).
- [Inca] 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).

- [Incb] Go Pivotal Inc. *SpringSource.org*. URL: <http://www.springsource.org/> (besucht am 09.06.2013).
- [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).
- [Kei] Gregg Keizer. *Microsoft quickens browser pace with IE10, goes for annual upgrades*. URL: http://www.computerworld.com/s/article/9215766/Microsoft_quickens_browser_pace_with_IE10_goes_for_annual_upgrades (besucht am 12.06.2013).
- [Kow] Kris Kowal. *kriskowal/q*. URL: <https://github.com/kriskowal/q> (besucht am 13.06.2013).
- [Lc] Jake Luer und individual contributors. *Chai*. URL: <http://chaijs.com>.
- [Ltd] Vaadin Ltd. *Vaadin - thinking of U and I - vaadin.com*. URL: <https://vaadin.com/home> (besucht am 09.06.2013).
- [Moza] Mozilla. *box-shadow / MDN*. URL: <https://developer.mozilla.org/en-US/docs/Web/CSS/box-shadow> (besucht am 05.06.2013).
- [Mozb] Mozilla. *eval - JavaScript / MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval (besucht am 09.06.2013).
- [Mozc] 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).
- [Mozd] Mozilla. *Manipulating the browser history*. URL: https://developer.mozilla.org/en-US/docs/DOM/Manipulating_the_browser_history (besucht am 15.04.2013).
- [Moze] Mozilla. *Mozilla Developer Network*. URL: <https://developer.mozilla.org/> (besucht am 14.03.2013).
- [Mozf] 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).
- [Oraa] Oracle. *Java EE at a Glance*. URL: <http://www.oracle.com/technetwork/java/javaeeforbeginners/overview/index.html> (besucht am 09.06.2013).
- [Orab] 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).
- [Pic] Heydon Pickering. *Structural Semantics — The Importance Of HTML5 Sectioning Elements / Smashing Coding*. URL: <http://coding.smashingmagazine.com/2013/01/18/the-importance-of-sections/> (besucht am 09.06.2013).
- [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).
- [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://nodedtuts.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://bit.ly/VZzu9i> (besucht am 12.03.2013).
- [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).
- [Typ] TypeSafe. *Play Framework - Build Modern & Scalable Web Apps with Java and Scala*. URL: <http://www.playframework.com/> (besucht am 09.06.2013).
- [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. *Connect-Session-Sequelize*. URL: <https://npmjs.org/package/connect-session-sequelize> (besucht am 07.06.2013).
- [Weib] Michael Weibel. *Express.js Prototyp - app.js*. URL: <https://github.com/hsr-ba-ajw-2013/BA/blob/prototype-express/app.js> (besucht am 16.03.2013).

- [Wika] Wikipedia. *AJAX (Programmierung)*. URL: [http://de.wikipedia.org/w/index.php?title=Ajax_\(Programmierung\)&oldid=118744583](http://de.wikipedia.org/w/index.php?title=Ajax_(Programmierung)&oldid=118744583) (besucht am 06. 06. 2013).
- [Wikb] Wikipedia. *Barrierefreies Internet*. URL: http://de.wikipedia.org/w/index.php?title=Barrierefreies_Internet&oldid=119350081 (besucht am 12. 06. 2013).
- [Wikc] Wikipedia. *Base64*. URL: <http://de.wikipedia.org/w/index.php?title=Base64&oldid=118052316> (besucht am 11. 06. 2013).
- [Wikd] Wikipedia. *Bidirectional Streams Over HTTP*. URL: <http://en.wikipedia.org/w/index.php?title=BOSH&oldid=555755444> (besucht am 15. 03. 2013).
- [Wike] Wikipedia. *Defensive Programming*. URL: http://en.wikipedia.org/w/index.php?title=Defensive_programming&oldid=541456737 (besucht am 11. 06. 2013).
- [Wikf] Wikipedia. *Frames (HTML)*. URL: [http://de.wikipedia.org/w/index.php?title=Frame_\(HTML\)&oldid=118246657](http://de.wikipedia.org/w/index.php?title=Frame_(HTML)&oldid=118246657) (besucht am 12. 06. 2013).
- [Wikg] Wikipedia. *Futures and promises*. URL: http://en.wikipedia.org/w/index.php?title=Futures_and_promises&oldid=556613152 (besucht am 13. 06. 2013).
- [Wikh] Wikipedia. *gzip*. URL: <http://de.wikipedia.org/w/index.php?title=Gzip&oldid=116452018> (besucht am 14. 04. 2013).
- [Wiki] 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).
- [Wikj] Wikipedia. *Mixin - Wikipedia*. URL: <https://de.wikipedia.org/w/index.php?title=Mixin&oldid=118972775> (besucht am 10. 06. 2013).
- [Wikk] Wikipedia. *Proof Of Concept*. URL: https://de.wikipedia.org/w/index.php?title=Proof_of_Concept&oldid=116970837 (besucht am 24. 03. 2013).
- [Wikl] Wikipedia. *Separation of Concerns*. URL: http://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=543036001 (besucht am 09. 04. 2013).
- [Wikm] Wikipedia. *Uniform Resource Identifier*. URL: http://de.wikipedia.org/w/index.php?title=Uniform_Resource_Identifier&oldid=118628990 (besucht am 05. 06. 2013).
- [Wikn] Wikipedia. *Uniform Resource Locator*. URL: http://de.wikipedia.org/w/index.php?title=Uniform_Resource_Locator&oldid=119183264 (besucht am 05. 06. 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 D Projektrelevante URL's

Ressource	URL
<i>Projektverwaltung</i>	http://redmine.alabor.me/projects/ba2013
<i>Code: Git Repository</i>	https://github.com/hsr-ba-ajw-2013/BA
<i>Code: CI</i>	https://travis-ci.org/hsr-ba-ajw-2013/BA
<i>Code: Dokumentation</i>	http://hsr-ba-ajw-2013.github.com/BA/docs
<i>Code: Unit Test Coverage Report</i>	https://coveralls.io/r/hsr-ba-ajw-2013/BA
<i>Thesis: Git Repository</i>	https://github.com/hsr-ba-ajw-2013/BA-Dokumentation
<i>Thesis: PDF</i>	http://hsr-ba-ajw-2013.github.com/BA-Dokumentation/thesis.pdf
<i>Thesis: CI</i>	https://travis-ci.org/hsr-ba-ajw-2013/BA-Dokumentation
<i>Meeting Protokollierung</i>	https://github.com/hsr-ba-ajw-2013/BA-Dokumentation/wiki

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* [Orab]

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 selbst 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

Links für Fortgeschrittene

- **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 sind momentan noch (bis ECMAScript 6 Standard ist) klar definierte Patterns nötig. Das Module Pattern ist nach Ansicht der Autoren eines der besten davon.
- **Secrets of the JavaScript Ninja** [RB12]
Der jQuery-Author beschreibt in diesem Buch Techniken um zum JavaScript-Guru aufzusteigen.

Weitere Links

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

F.2. Node.js

Einführung

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

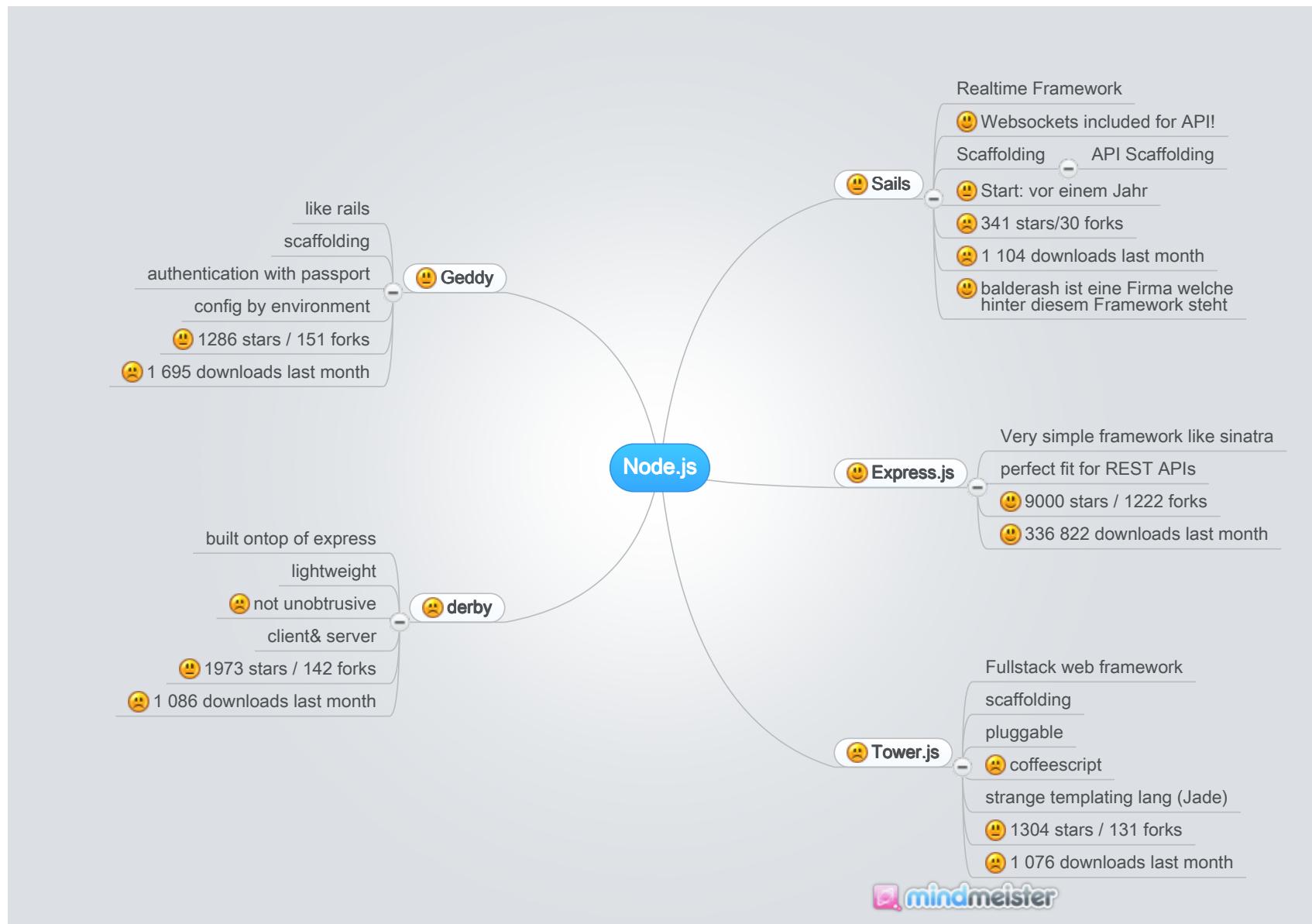
- Node Tutorials [Tei]
- Mastering Node.js [Holc]
Kostenloses Buch vom Express.js [Holf] etc. Entwickler TJ Holowaychuck [Holi]. Beinhaltet sowohl Informationen für Anfänger wie auch für Fortgeschrittene.
- Node.js API Docs [Nod]

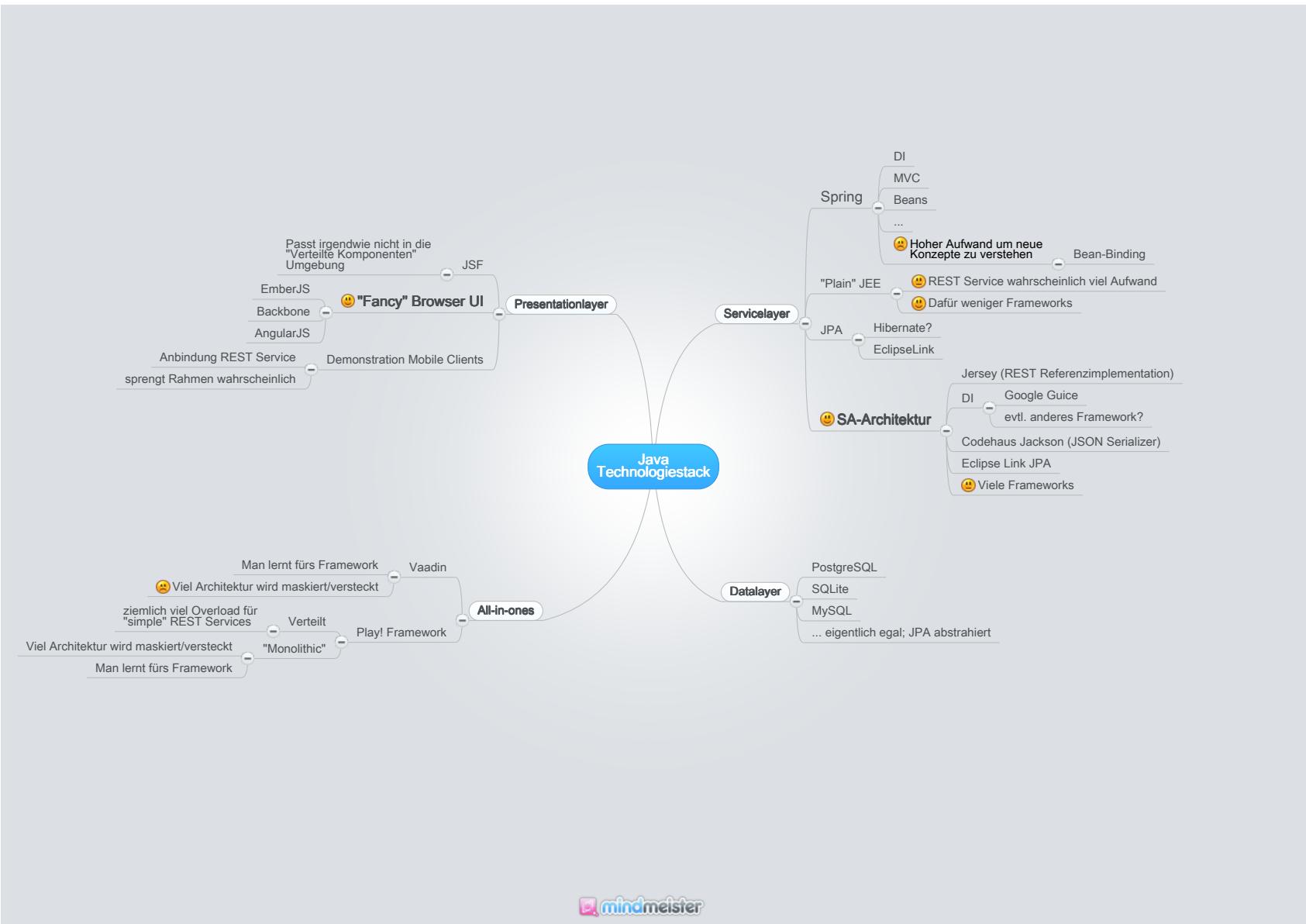
F.3. Einführung für benutzte Libraries

- 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.

Anhang G **Technologieevaluation**

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





Anhang H **Produktentwicklung**

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

Tabelle H.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.

H.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

Grundfarbpalette



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



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



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

Abbildung H.1.: Branding Farbpalette

Logo & Logovariationen

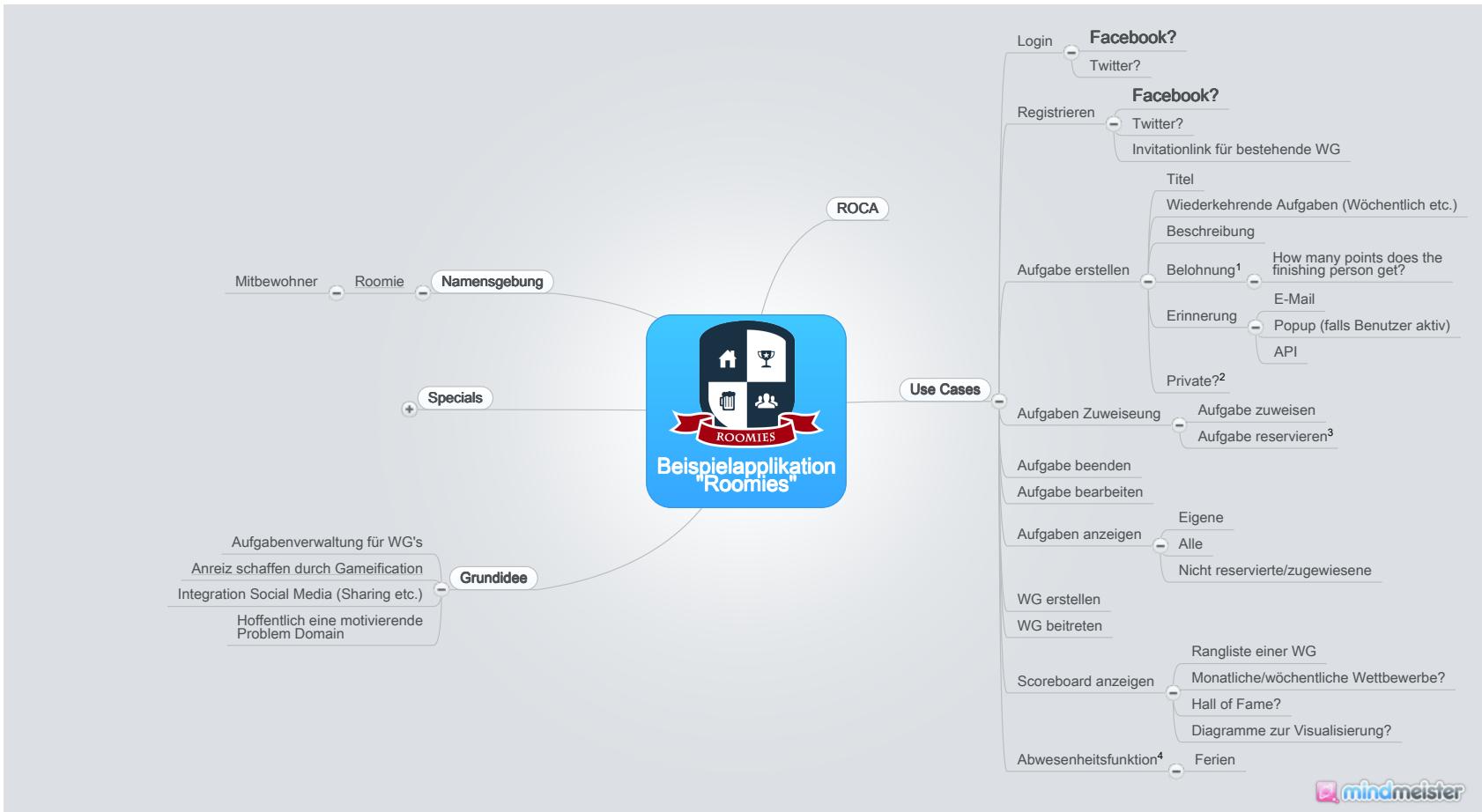


Abbildung H.2.: Roomies Logo im College Stil



Abbildung H.3.: Roomies Logo in verschiedenen Größen & Varianten

H.3. Mindmap



Anhang I 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.

I.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 I.1.: Funktionale Anforderungen

I.2. Nichtfunktionale Anforderungen

ID	Name	Beschreibung
<i>NF1</i>	Antwortzeit	Die Applikation antwortet bei normalen Anfragen innerhalb von 0.2s.
<i>NF2</i>	Desktop Browserkompatibilität	Die Applikation unterstützt Internet Explorer 8 und höher, Chrome 25 und höher, Firefox 19 und höher sowie Safari 6 und höher.
<i>NF3</i>	Mobile Browserkompatibilität	Die Applikation unterstützt Safari 6.0 und Android Browser 4.0.
<i>NF4</i>	Sicherheit	Die Applikation kontrolliert den Zugriff auf geschützte Ressourcen.
<i>NF5</i>	Architekturrichtlinien	Die Applikation entspricht den in 2.1.3 "Projektspezifische Richtlinien" erarbeiteten Architekturrichtlinien.

Tabelle I.2.: Nichtfunktionale Anforderungen

I.3. Use Cases

Das Diagramm I.1 zeigt eine Übersicht der zu implementierenden Use Cases und deren Akteure.

Darauffolgend wird jeder die Use Case genauer beschrieben.

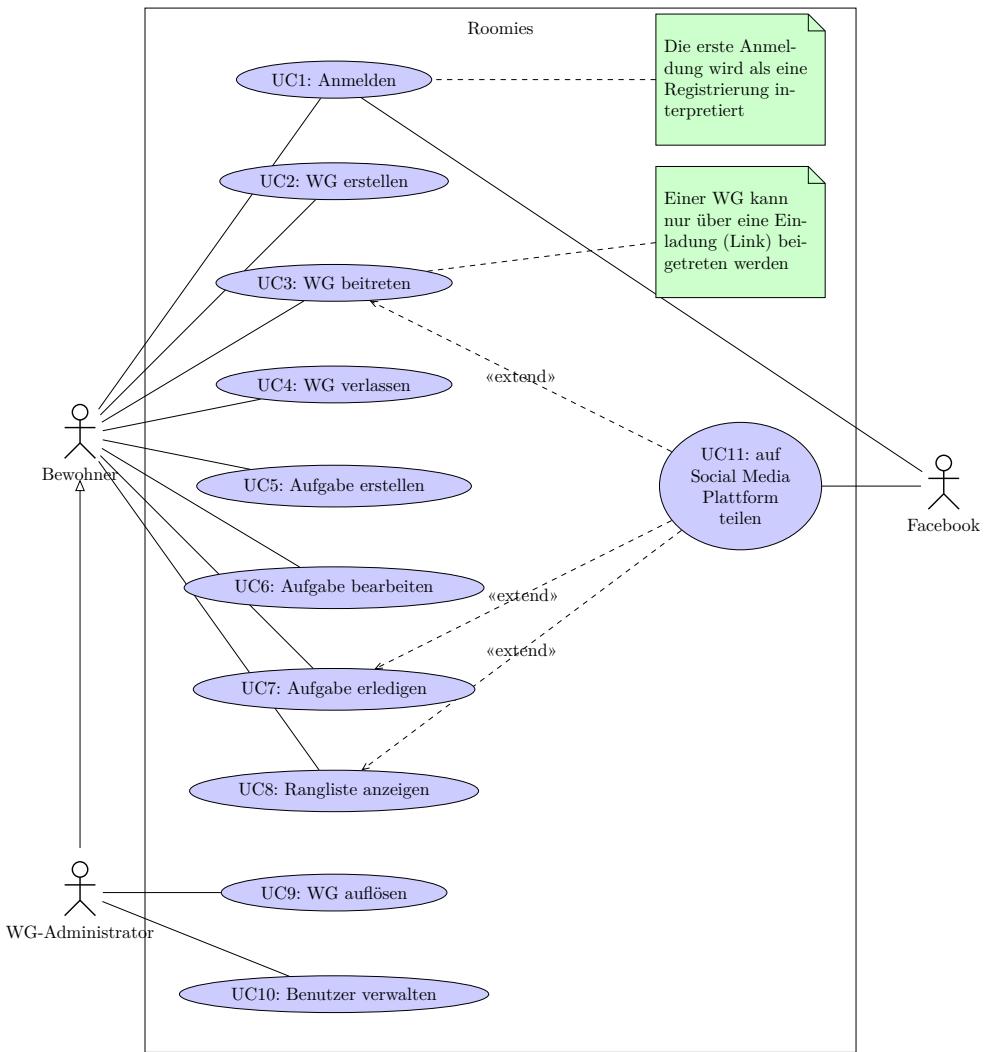


Abbildung I.1.: Use Case Diagramm

I.3.1. Akteure

Name	Beschreibung
Bewohner	Als <i>Bewohner</i> wird ein Anwender der <i>Roomies</i> Anwendung bezeichnet, der zu einer <i>WG</i> gehört. Dieser besitzt die Rechte <i>Aufgaben</i> seiner <i>WG</i> zu verwalten.
WG-Administrator	Der <i>WG-Administrator</i> ist eine Erweiterung des Akteurs <i>Bewohner</i> . Der Ersteller einer <i>WG</i> wird automatisch zu deren <i>Administrator</i> . Diese Rolle kann an <i>Bewohner</i> weitergegeben werden (siehe UC4: WG verlassen und UC10: Benutzer verwalten).
Facebook	<i>Facebook</i> ist die Schnittstelle zur Social Media Plattform. Sie ermöglicht das Anmelden bei <i>Roomies</i> sowie das Teilen von Daten und Links.

Tabelle I.3.: Akteure

I.3.2. UC1: Anmelden

Mapped Requirement	NF4
Primary Actor	Bewohner
Secondary Actor	Facebook
Story	Der <i>Bewohner</i> startet <i>Roomies</i> . Das System zeigt das Anmeldeformular. Der Benutzer meldet sich mittels seines Facebook-Logins an. Das System überprüft die Daten. Sind sie gültig wird der Benutzer auf die <i>WG</i> -Startseite weitergeleitet.

Tabelle I.4.: UC1: Anmelden

I.3.3. UC2: WG erstellen

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

Tabelle I.5.: UC2: WG erstellen

I.3.4. UC3: WG beitreten

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

Tabelle I.6.: UC3: WG beitreten

I.3.5. UC4: WG verlassen

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

Tabelle I.7.: UC4: WG verlassen

I.3.6. UC5: Aufgabe erstellen

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

Tabelle I.8.: UC5: Aufgabe erstellen

I.3.7. UC6: Aufgabe bearbeiten

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

Tabelle I.9.: UC6: Aufgabe bearbeiten

I.3.8. UC7: Aufgabe erledigen

<i>Mapped Requirement</i>	F4
<i>Primary Actor</i>	Bewohner
<i>Story</i>	In der Aufgabenliste kann ein <i>Bewohner</i> eine Aufgabe als erledigt markieren. Als Folge davon kann "UC11: auf Social Media Plattform teilen" ausgeführt werden.

Tabelle I.10.: UC7: Aufgabe erledigen

I.3.9. UC8: Rangliste anzeigen

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

Tabelle I.11.: UC8: Rangliste anzeigen

I.3.10. UC9: WG auflösen

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

Tabelle I.12.: UC9: WG auflösen

I.3.11. UC10: Benutzer verwalten

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

Tabelle I.13.: UC10: Benutzer verwalten

I.3.12. UC11: auf Social Media Plattform teilen

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

Tabelle I.14.: UC11: auf Social Media Plattform teilen

I.4. Umgesetzte Use Cases

Tabelle I.15 gibt darüber Aufschluss, welche Use Cases in der finalen Version von *Roomies* umgesetzt wurden.

<i>ID</i>	<i>Umsetzung</i>	<i>Use Case</i>	<i>Bemerkung</i>
UC1	✓	Anmelden	
UC2	✓	WG erstellen	
UC3	✓	WG beitreten	
UC4	!	WG verlassen	Aus Zeitgründen nicht umgesetzt
UC5	✓	Aufgabe erstellen	
UC6	✓	Aufgabe bearbeiten	
UC7	✓	Aufgabe erledigen	
UC8	✓	Rangliste anzeigen	
UC9	✓	WG auflösen	
UC10	!	Benutzer verwalten	Zwecks <i>Unobtrusive JavaScript</i> fallengelassen
UC11	!	auf Social Media Platform teilen	Nur für Einladungs-Link umgesetzt

Tabelle I.15.: Resultate Use Case Umsetzung

Anhang J Projektplanung

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

Tabelle J.1.: Projektierungsbeschreibung

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

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

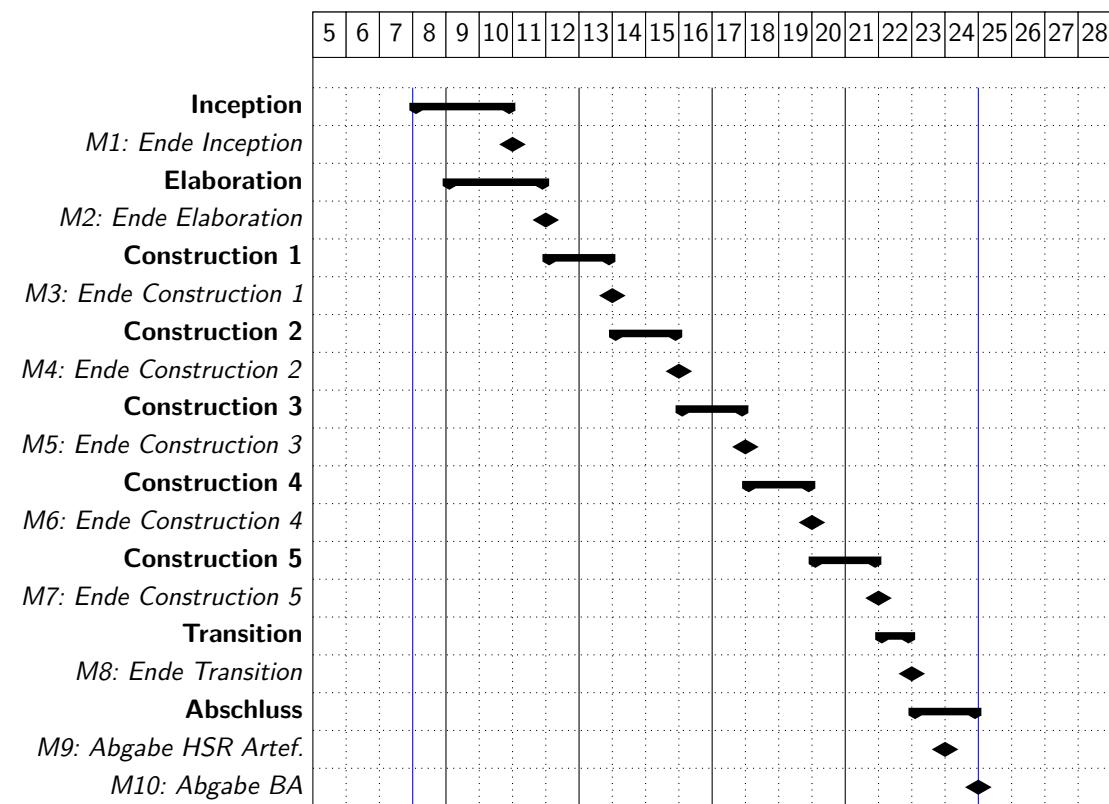


Abbildung J.1.: Iterationsübersicht mit Meilensteinen, Kalenderwochen Februar bis Juli 2013

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.

J.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 <i>Hoch</i> umgesetzt.
M4	Ende Construction 2	14.04.2013	Alle Use Cases der Priorität <i>Hoch</i> 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 J.2.: Meilensteine

J.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 <i>Typ: Quellcode/Applikation</i>
A80	M8	Quellcode Paket	Quellcode der Beispielapplikation zum eigenen, spezifischen Deployment. Bereits zur Weiterentwicklung. <i>Typ: Quellcode</i>
A81	M8	Vagrant Paket	VM-Image mit lauffähiger Version der Beispielapplikation. <i>Typ: Vagrant Image</i>
A82	M8	Installationsanleitung	Anleitung wie die verschiedenen Deployment-Pakete (Artefakte A80-81) eingesetzt/installiert werden können.
A100	M10	A0-Poster	Gem. HSR Vorgaben zu erstellendes Poster mit Übersicht 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 J.3.: Abzugebende Artefakte

J.4. Meetings

J.4.1. Regelmässiges Statusmeeting

Während der gesamten Projektdauer findet jeweils am Mittwoch um 10 Uhr ein wöchentliches Statusmeeting statt. Die Sitzung wird abwechselungsweise von einer Person

aus dem Projektteam geführt sowie von einer anderen protokolliert.

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

J.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.

J.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.

J.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.

J.5.3. Git Repositories

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

J.5.4. Continuous Integration

Für das Projekt wird Travis CI als Continuous Integration Lösung verwendet. Nähere Informationen sind Abschnitt “Qualitätsmanagement” unter “Continuous Integration” zu finden.

J.6. Qualitätsmanagement

J.6.1. Baselining

Beim Erreichen der Meilensteine wird auf dem Quelltext- als auch auf Dokumentations-Git-Repository ein Tag mit der 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.

J.6.2. Testing

Die Beispielapplikation wird nach der TDD-Methodik entwickelt.

Unit Testing

Mocha [Holh] 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(
12         function(community) {
13           community.should.have.length(0);
14         done();
15       }).error(function(error) {
16         throw(error);
17       })
18     }).should.not.throw();
19   });
20 });
```

Quelltext J.1: Beispiel eines Unit Tests mit Mocha und Chai.js

Test Coverage Analyse

Die Test Coverage Analyse wird innerhalb des CI Prozesses erstellt. Dabei übernimmt “Coveralls” [Cov] die Auswertung der Coverage-Rohdaten und stellt entsprechende Reports zur Verfügung.

J.6.3. 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.

J.6.4. Coding Style Guideline

Die Coding Style Guideline ist im Anhang N “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.

J.6.5. Code Reviews

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

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

Anhang K Iterationsassessment

Iteration	Status	Bemerkung / Begründung
Inception	✓	Slowstart aufgrund unklarer Auftragslage. Zeit investiert in Projektsetup.
Elaboration	✓	
Construction 1	✓	
Construction 2	✓	
Construction 3	!	Verschiebung Gamification auf Construction 4 sowie Vorzug UC8 Rangliste anzeigen auf Construction 3 wegen Abwesenheiten im Team
Construction 4	!	Verzögerungen durch Unobtrusive JavaScript-Umsetzung. Meilenstein M6 um 24 Stunden versäumt
Construction 5	!	Aufgabe UC4 WG verlassen & UC10 Benutzer verwalten zu Gunsten Unobtrusive JavaScript-Umsetzung, Meilenstein M7 mit M8 zusammengelegt
Transition	!	Wegen abschliessender Arbeiten Meilenstein M8 um 24 Stunden versäumt
Abschluss	✓	

Tabelle K.1.: Iterationsassessment

Anhang L Auswertung Zeitrapportierung

L.1. Aufwand pro Person und Woche

Mitglied	2013-7	2013-8	2013-9	2013-10	2013-11	2013-12	2013-13
<i>Manuel Alabor</i>	1	6.45	16	22	31.5	23.75	23.25
<i>Michael Weibel</i>	1	6.67	13.5	20.66	30.24	24.75	26
<i>Alexandre Joly</i>	1	5.2	7.25	16.5	31.25	22	22
Gesamtzahl	3	18.32	36.75	59.16	92.99	70.5	71.25

Tabelle L.1.: Zeitrapport - Kalenderwochen 7-13

Mitglied	2013-14	2013-15	2013-16	2013-17	2013-18	2013-19
<i>Manuel Alabor</i>	10.05	25.75	26	12.75	21.5	16.75
<i>Michael Weibel</i>	18	24.25	14	5	18.5	17.25
<i>Alexandre Joly</i>	13	24.25	16.25	19.75	14.5	22
Gesamtzahl	41.05	74.25	56.25	37.5	54.5	56

Tabelle L.2.: Zeitrapport - Kalenderwochen 14-19

Mitglied	2013-20	2013-21	2013-22	2013-23	2013-24	Gesamtzahl
<i>Manuel Alabor</i>	24.25	22.75	22.25	49.3	15.25	370.55
<i>Michael Weibel</i>	22.25	23.5	37.25	44.25	18.5	365.57
<i>Alexandre Joly</i>	18.25	22.5	26	37.75	19.25	338.7
Gesamtzahl	64.75	68.75	85.5	131.3	53	1074.82

Tabelle L.3.: Zeitrapport - Kalenderwochen 20-24

L.2. Burndown

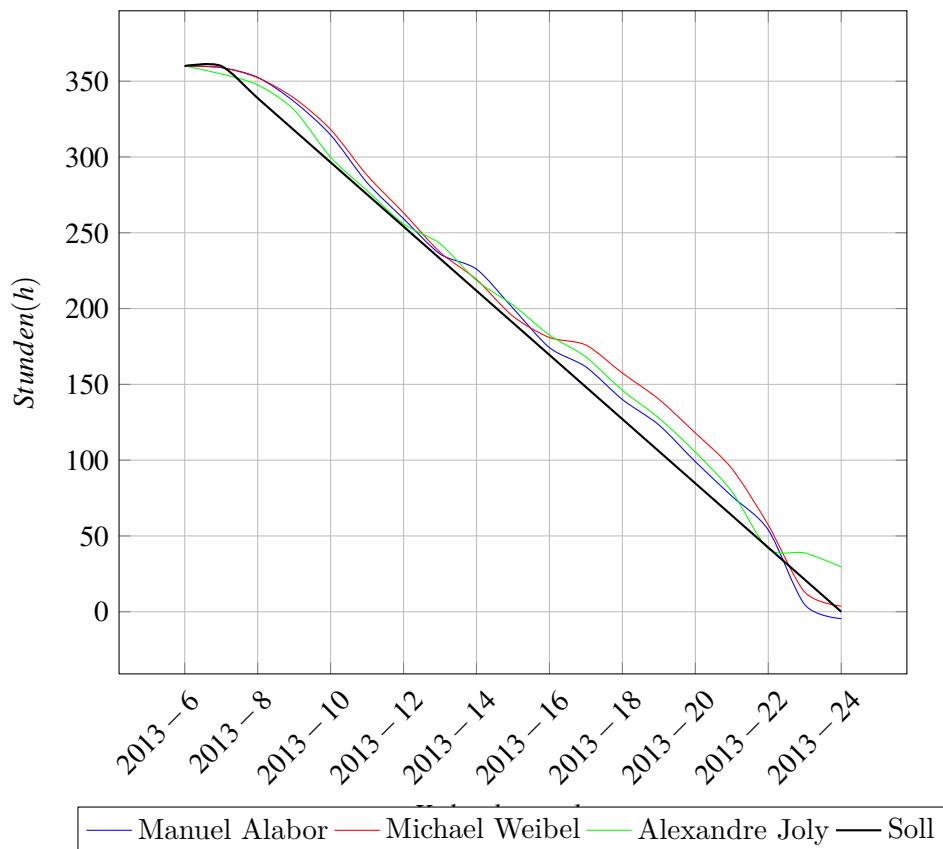


Abbildung L.1.: Burndown

L.3. Zeitaufwand pro Aufgabenbereich

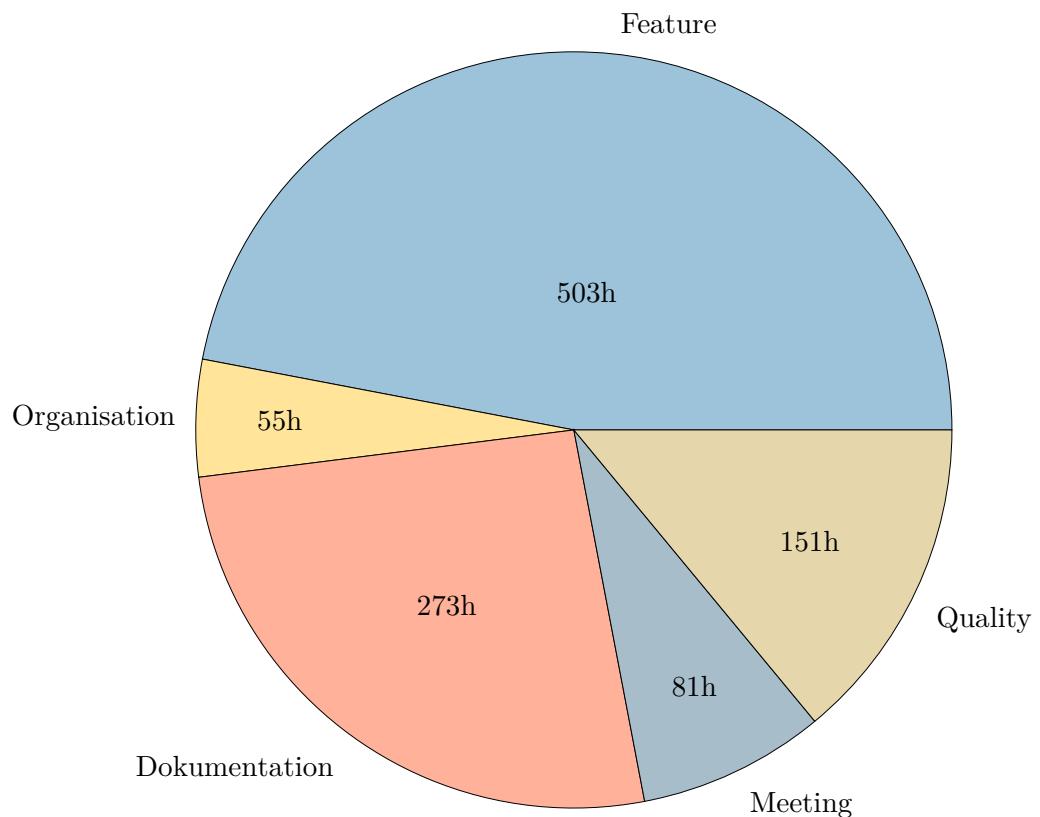


Abbildung L.2.: Zeitaufwand pro Aufgabenbereich

L.4. Zeitaufwand pro Iteration

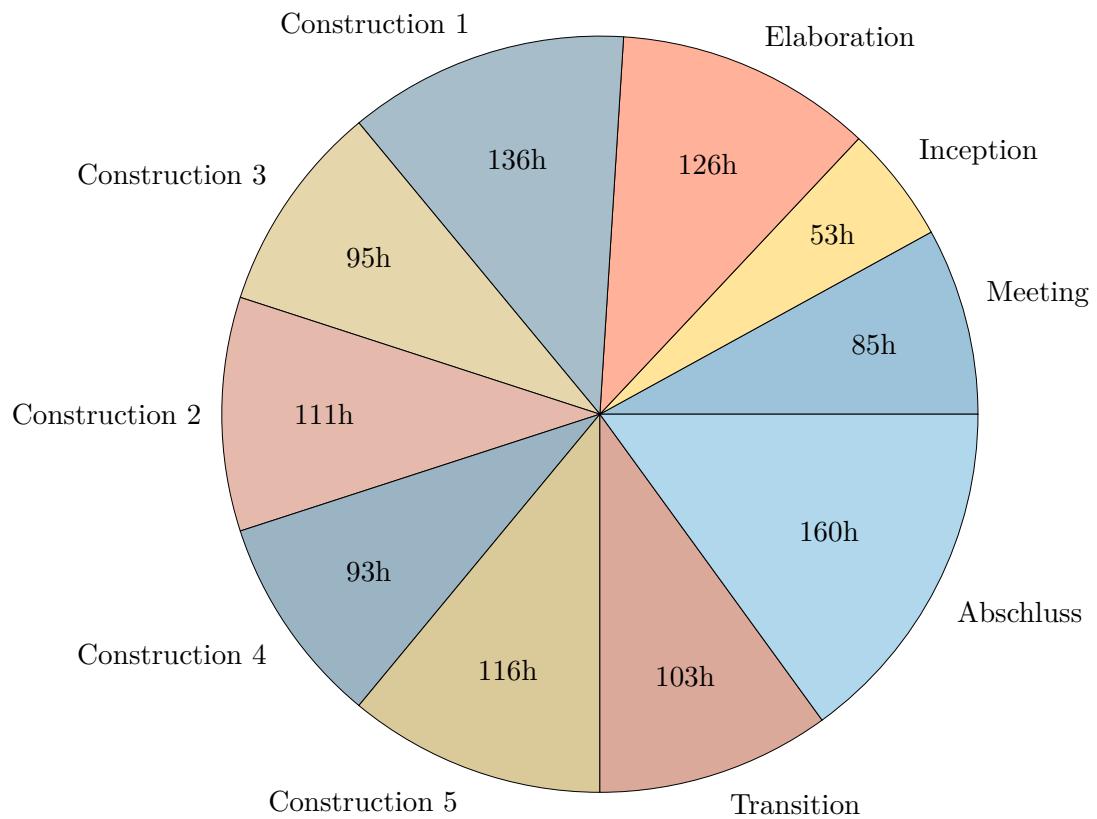


Abbildung L.3.: Zeitaufwand pro Iteration

Anhang M Screenshot Tour: Beispielapplikation Roomies

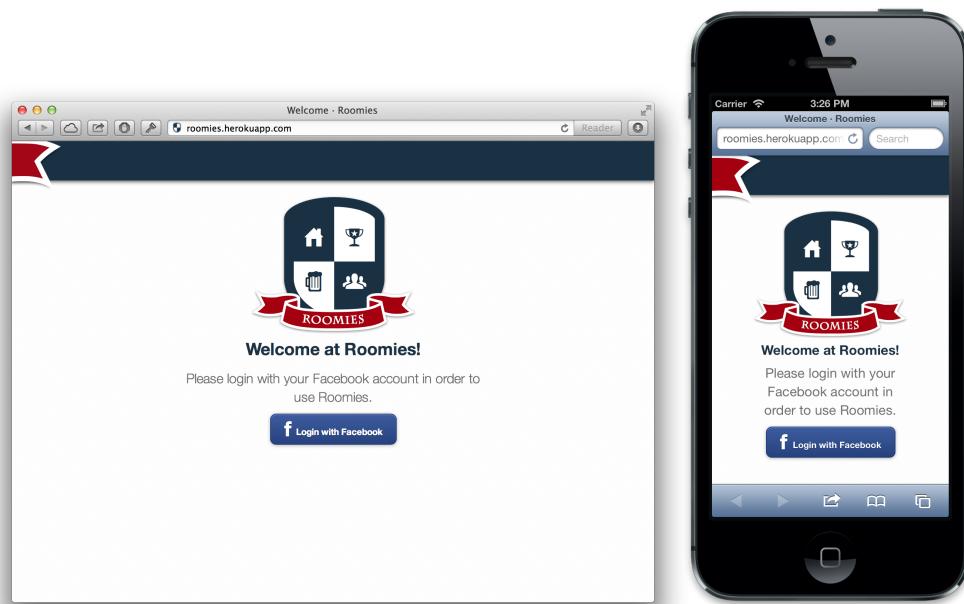


Abbildung M.1.: Roomies – Anmelden

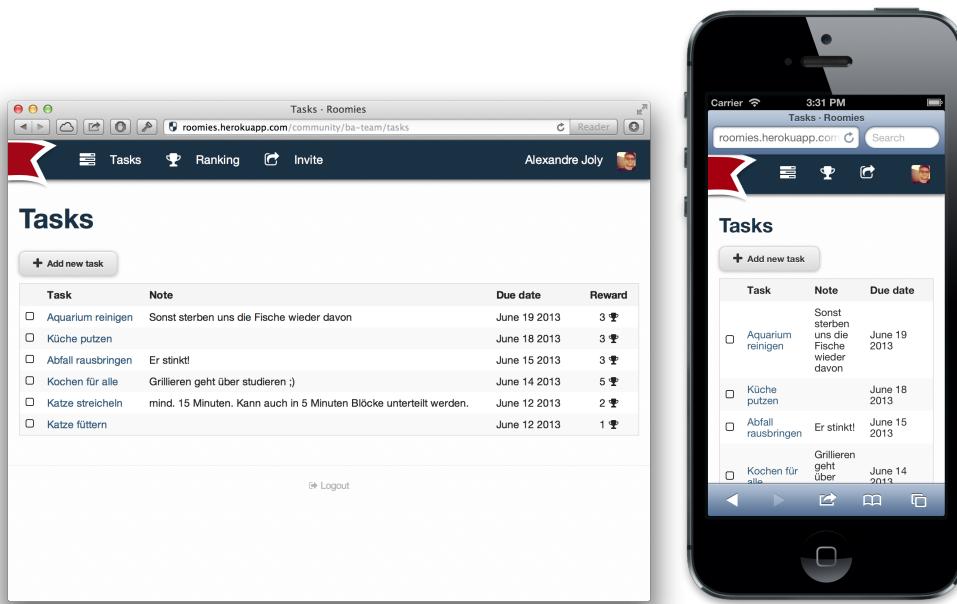


Abbildung M.2.: Roomies – Aufgaben

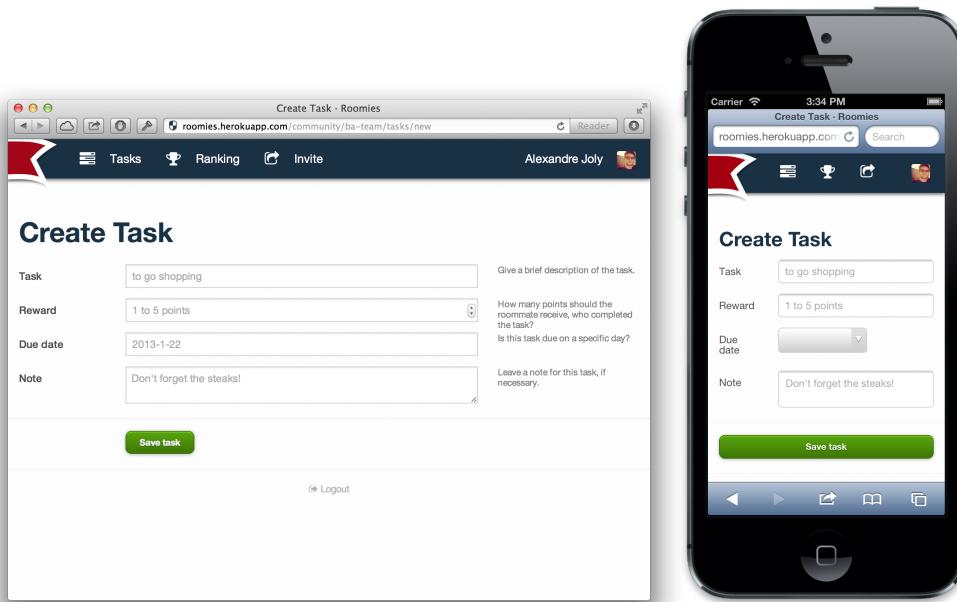


Abbildung M.3.: Roomies – Aufgabe erstellen

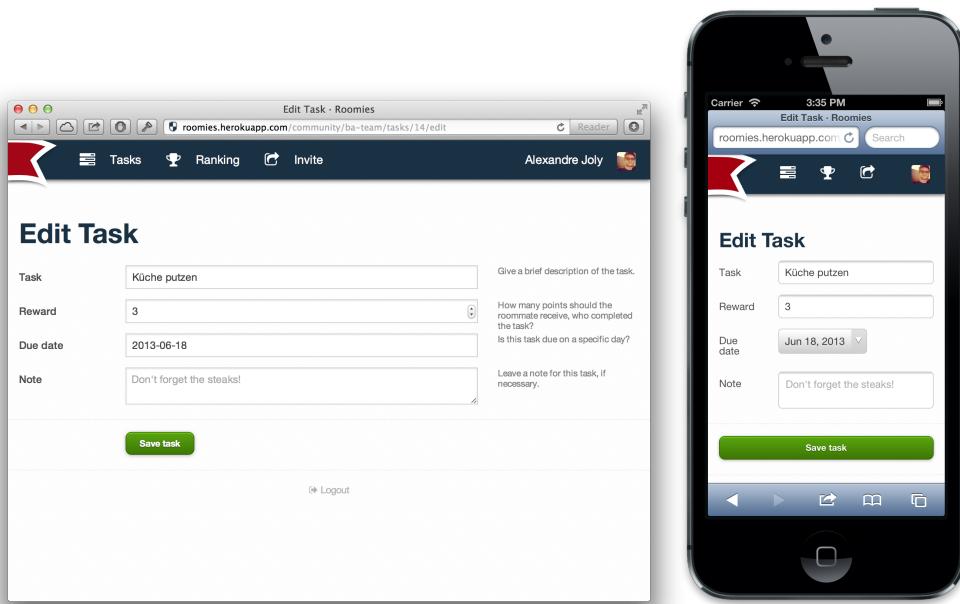


Abbildung M.4.: Roomies – Aufgabe bearbeiten

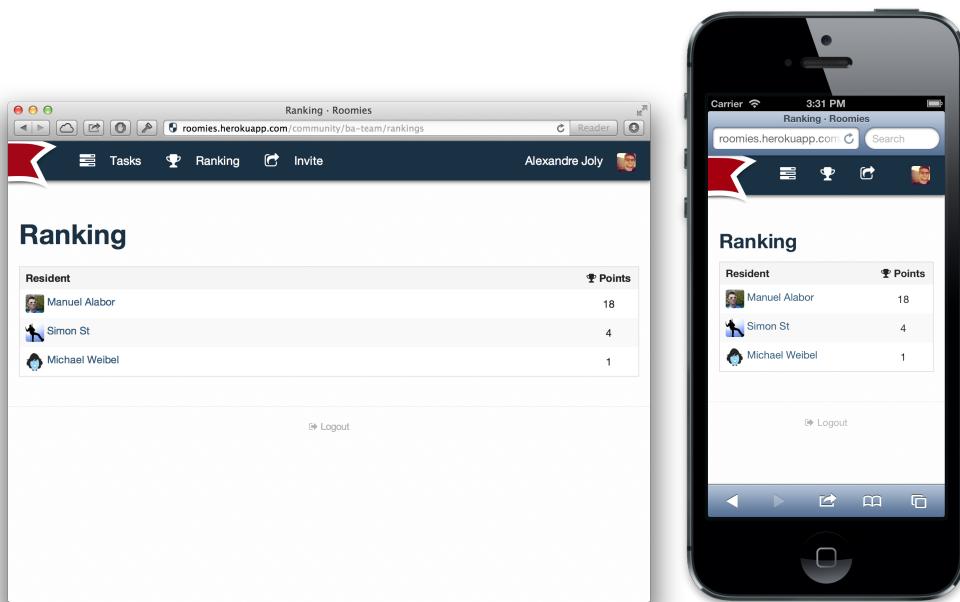


Abbildung M.5.: Roomies – Rangliste

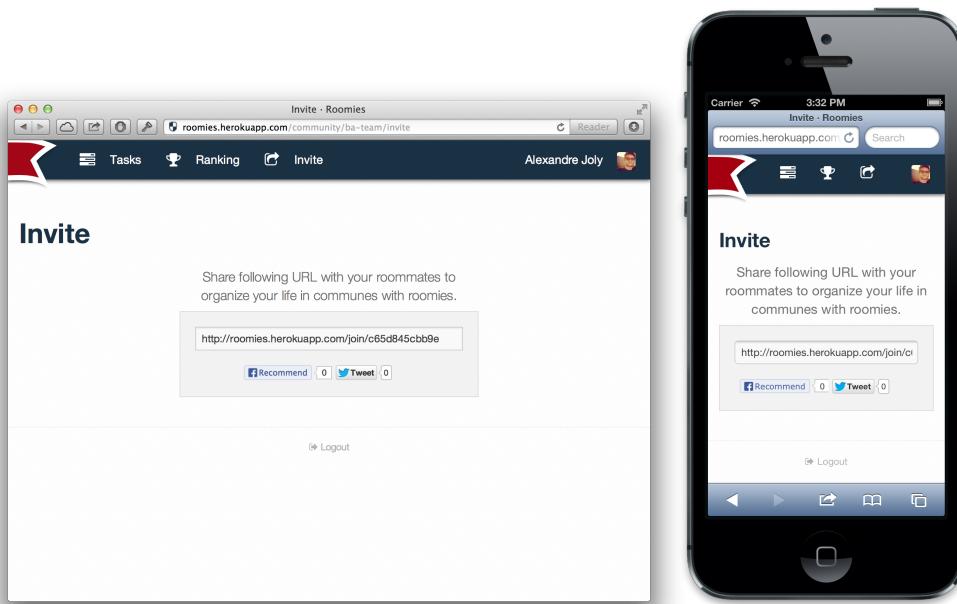


Abbildung M.6.: Roomies – Einladen

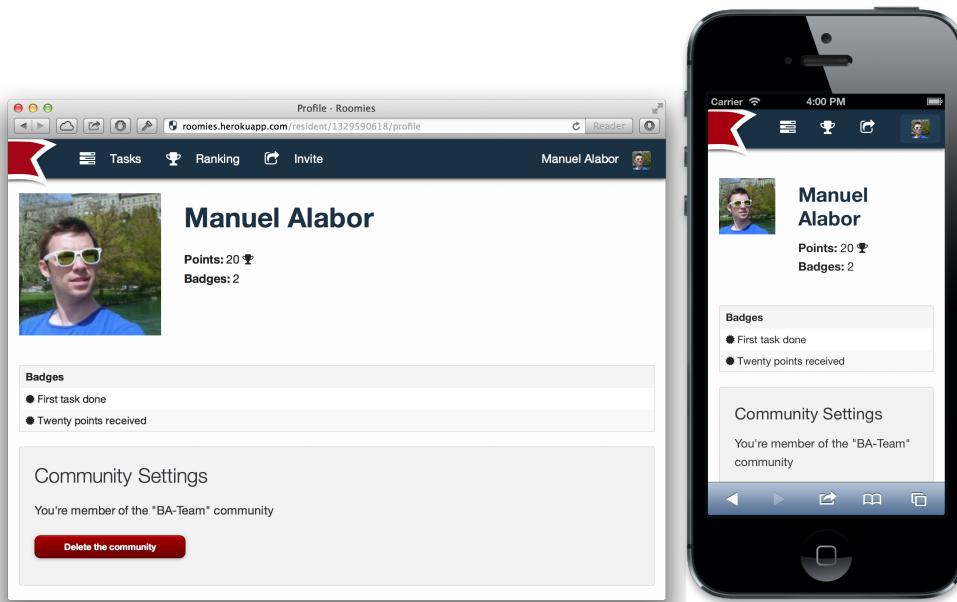


Abbildung M.7.: Roomies – Bewohnerprofil

Anhang N Coding Style Guideline

Basierend auf den übersetzten Guidelines von Timo Furrer [Airb] (ursprünglich von Airbnb [Airc] veröffentlicht) wurde gemäss den Präferenzen des Projektteams eine erweiterte und optimierte Version dieser erstellt [Aira].

Diese Coding Style Guidelines wurden für die Erstellung von Quelltext im Rahmen dieser Bachelorarbeit benutzt.

Führende Kommas

```
1 // schlecht
2 var once,
3     upon,
4     aTime;
5
6 // gut
7 var once
8   , upon
9   , aTime;
10
11 // schlecht
12 var hero = {
13   firstName: 'Bob',
14   lastName: 'Parr',
15   heroName: 'Mr. Incredible',
16   superPower: 'strength'
17 };
18
19 // gut
20 var hero = {
21   firstName: 'Bob'
22   , lastName: 'Parr'
23   , heroName: 'Mr. Incredible'
24   , superPower: 'strength'
25 };
```

Quelltext N.1: Führende Kommas - Coding Guidelines [Aira]

Tabulator anstatt Leerzeichen

```
1 // schlecht
2 function() {
3 ...var name;
4 }
5
6 // schlecht
7 function() {
8 .var name;
9 }
10
11 // gut
12 function() {
13 →var name;
14 }
```

Quelltext N.2: Tabulatoren - Coding Guidelines [Aira]

Anhang O **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 P **Meetingprotokolle**

Bachelorarbeit Vorbesprechung -

14.02.2013

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 zu entwickeln?
 - 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 seitens 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 Firmenanstände, 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 Teilaспект 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 einem 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 interessiert vor allem 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 vor allem Mobile interessant
 - arbeitet auch vor allem 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 interessantes)
 - 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 ob was anderes geben würde
 - gibt den 3 Studierenden möglichst freie Hand
- KGA: Termin bis wann die Entscheidung möglich sein
 - HRU: allerspätestens 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 Themen sehr interessant für Team?
- Team: Ja, insbesondere auch zu gross für die kurze Zeit (Thema 2)
- HRU: Reservationssystem vor allem 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

Bachelorarbeit Entscheidung Thema

- 20.02.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- SA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO) [Protokoll]
 - Manuel Alabor (MAL)

Kickoff

- Termin am Mi 27.2.13 10:10
- Weekly Meetings jeweils am Mi 10:10

TODO

- (HRU) Aufgabenstellung bis 20.2.13 Abends
- (SA Team) Grobablauf und Vorstellungen bis Mi 27.2.13

Weekly Meeting - 27.02.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- SA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: AJO
- Protokoll: MAL

Agenda

- Aktuelle Themen
- Offene Fragen
- Ausblick, nächste Schritte

Diskussion: Eine vs. mehrere Beispielapps

- TEAM: Aufbauende, iterative Snapshots einer einzige Applikation wären sehr schön
 - Pro Iteration 1-2 Prinzipien
 - Optimal für Unterricht
 - Student muss sich nicht immer neu einarbeiten

- **HRU: Entscheidung: Eine einzige App**
- Form des Endergebnisses:
 - HRU: Fokus soll auf guter Beispielapplikation liegen
 - HRU: Sekundär soll "padagogischer" Aufbau für Unterricht sein
- Ideen Problemdomain:
 - HRU: Könnte schwierig werden Prinzipien auf nur eine Problemdomain anzuwenden
 - AJO: WG Achievments:
 - HRU: Motivationsfaktor für Studenten da
 - KGA: Gefahr dass ROCA Principles evtl. Overkill sein könnte besteht (immer)

Technologiestacks

- MAL: Machen zwei verschiedene Stacks Sinn?
 - 1x Opensource Schiene (node.js, ember.js)
 - 1x Enterprise Schiene (Java EE, ...)
- HRU: Fokus:
 - Hauptfokus soll auf Architekturen liegen
 - Technologieübersicht eher sekundär
- Technologie-Wertung:
 - HRU: Keine Optionen:
 - .NET
 - Flash
 - HRU: Positiv:
 - node.js
 - Java EE
 - Ruby on Rails
- HRU: PHP kommt bis anhin nicht vor in InTe
- MWE: Technologieentscheid sollte nächste Woche fallen
- HRU: Zweischieniger Ansatz gefällt sehr gut
 - Jedoch: Aufwand für doppelte Umsetzung
 - Dafür: Spannend um Architekturkonzepte zu vergleichen

- **Multi Tier Architekturen:**
 - MWE: Komponenten-Architektur wäre interessant
 - HRU: Anbindung an Servicelayer wurde bis anhin immer ausgespart in InTe
 - KGA: Multitier-Architektur sehr interessanter Ansatz da bis anhin nirgends im Studium
 - KGA: Idee: Mit Java EE arbeiten, gegen Schluss "mit Bruchteil von Code" gleichen Server mit NodeJS zeigen
- MWE: Klar Aufzeigen welches Framework was kann
- KGA: "*Cognitive-Overload*" vermeiden
 - MAL: Ein Frontendframework "and stick with it"

Gefühl "Doku vs. Code"

- HRU: Thesis hat Visibility
- HRU: Fokus aber "eben schon" auf Beispielapplikation
- HRU: Weiterverwendung des Produkts noch nicht vollends festgelegt:
 - HRU: Ziel ist nicht, fertige Übungen zu haben
- KGA: Ziel: "HRU kennt sich komplett mit App aus"
- HRU: Idee: Aufbau analog Beispiel JSF@work
 - Pro Iteration ein Kapitel
 - Bezug auf entsprechende ROCA Prinzip(ien) nehmen
 - Erklärung wie dies nun in Applikation umgesetzt wurde

Bereich "Andere Themen"

- HRU: Fokus liegt auf ROCA
- HRU: Andere Themen nach "gutdünken"
- HRU: Falls ROCA Prinzip/Teil davon überflüssig, gerne kommunizieren

Projektmanagement & -vorgehen

- HRU: Wie aufgesetzt?

- Analog SA
 - Redmine: <http://redmine.alabor.me/>
 - Code: <https://github.com/mweibel/BA>
 - Thesis: <https://github.com/mweibel/BA-Dokumentation>
- MWE: Idee Vorgehen:
 - Beschreibung der Applikation (Domainmodel)
 - Iterations Beschreibung

Ausblick

- Nächste Sitzungen wieder 10:00, gleicher Ort
- Technologiestack:
 - 2-3 Vorschläge
 - Entscheidung
- Dokumentation:
 - Wie soll sie aussehen?
- Umfang Beispielapplikation, konkreter Vorschlag:
 - Brief Description zu Features
 - *Keine Fully Suited Use Cases!*
 - *Pragmatisch*
- Konkrete Iterationsplanung

Weekly Meeting - 06.03.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- SA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: MAL
- Protokoll: MWE

Agenda

- Vorstellung Technologiestacks
- Diskussion & Entscheidung Technologiestack
- Vorstellung Entwurf Iterationsplanung
- Kurzes Update: Problem Domain Idee "Roomies"
- Offene Fragen
- Ausblick, nächste Schritte

Letztes Protokoll

Ist i.O.

Vorstellung Technologiestacks

- REST?
- Authentication?
- MVC?

AJO: Ruby on Rails

- REST sehr einfach
 - aus Model direkt erstellbar (scaffolding)
- Formate XML & JSON
 - *.plist als format erweiterbar (iOS)
- Authentication
 - gleiche Möglichkeiten wie sonst überall
 - User management gibts gute plugins
 - AJO sieht keine Einschränkungen
- MVC: Rails ist MVC

MWE: node.js

- Überblick über frameworks (mindmap)
- Express oder Sails als Vorschlag

MAL: Java

- <http://www.mindmeister.com/266971852>
 - Kann ich Lese-Zugriff bekommen? kevin.gaunt@gmail.com
 - --> Dropbox
- Service Layer
 - Spring
 - Hoher Initialaufwand
 - "Plain" J2EE
 - MAL nicht gross Erfahrung damit
 - Grosser initialaufwand deswegen
 - HRU: Jersey nicht dabei?
 - MAL weiss es nicht genau, aber denkt eher nicht

- JPA ein muss, egal welches Framework
- SA-Architektur
 - Viele Frameworks
 - Aber: schon Erfahrung, initialaufwand kleiner
- Presentation Layer
 - JSF: nicht wirklich eine Option
 - vieles passiert auf Server
 - nicht wirklich ein "verteiltes System"
 - HRU: stimmt
 - Vorschlag: "Fancy" Browser UI
 - modernes Javascript framework
 - Demonstration mobile clients
 - wahrscheinlich zuviel
- All-In-One
 - Play!
 - fancy
 - wie rails
 - versteckt vieles
 - zuviel Overload für simple REST services
 - Vaadin
 - man lernt fürs Framework
 - viel wird versteckt

MALs Favorit: SA-Architektur + Fancy Browser UI

Diskussion & Entscheidung

Technologiestack

- Unobtrusive JS schwierig
- HRU: also: was machen wir?
- MAL: voreingenommen für Node.js
- AJO: node.js oder ruby/rails
- MWE: node.js oder ruby/rails

- alle 3 sachen sind vorstellbar
- node.js mit frontend js
- rails mit frontend js
- java REST-API, node.js presentation layer, frontend js
 - mehraufwand
- MAL: Idee von KGA wäre interessant
 - java im backend, node.js pres. layer, frontend js
 - KGA: müsste aber java fast als "legacy" betrachten
 - ist das sinnvoll?
 - MAL: evtl. in 2 Jahren ist die Architektur die wir hier machen legacy - dann könnte man dies so machen
- MAL: Ist in Rails wirklich so viel versteckt?
 - HRU: scaffolding, aber schlussendlich ist alles Ruby
 - HRU: viele Helper (viele Handarbeit wird abgenommen)
- HRU: node.js ist unbestritten eine eigene Schiene
- HRU: Rails/Java ist "etwas" konservativer
- HRU: bei Java wäre es so
 - Glassfish oder so ein ähnlicher Server
 - JSF im Frontend
 - damit Webapplikation machen
- MAL: wirklich 2 schieniger Technologiestack?
 - MAL: letztes Mal keine def. Entscheidung
- MAL: wenn Kontrast zw. 2 Technologien gewünscht ist, dann wäre JSF wirklich ein Ansatz
 - Primefaces vornedran und dann siehts doch relativ ähnlich aus wie die "gehypten" Frameworks/Technologien
- MAL: Wenn 2 technologien
 - JSF
 - Node.js mit Sails
 - MWE: Ok.
 - MWE: Wenn nur 1 Technologie: Rails - als logische Schlussfolgerung

- AJO: stimmt
 - evtl. zu wenig neue Technologien an HSR
 - HRU: somit wäre das ein Plädoyer für Node.js
- MAL: wie sehen sie das HRU?
 - HRU: node.js wäre am besten um zu lernen für HRU
 - HRU: Rails wäre interessant
 - HRU: Zuviel verschiedene Technologie (Klagen)
 - AJO: deswegen vielleicht nur 1 Technologie?
 - HRU: Stimmt. Obwohl natürlich JSF nicht komplett wegfallen wird
- MAL: Wie würde denn die Vorlesung gestaltet werden?
 - HRU: Fände es spannender, Node.js anzuschauen
- MAL: Somit definitiv node.js?
- HRU & KGA akzeptieren.

--> Node.js

Vorstellung Entwurf Iterationsplanung

- Problematik: Iterationssnapshot pro Principle Set
 - Vorschlag: Mapping statt Snapshots
 - MAL: Backend geht i.O. mit Roca
 - MAL: Frontend mit Roca aufsplitten eher schwierig
 - MAL: Statt Snapshot, gut Dokumentieren welches Roca Prinzip wo zu finden ist, wäre das i.O.?
 - HRU: stimmt zu, ist eine gute Lösung
 - HRU: Anleitung/Guide wo was zu sehen ist
- MAL: zeigt Excel-Dokument
- MAL: ist das in Ordnung?
 - HRU: Elaboration fertig bis nächste Woche -- evtl. zu früh?
 - MAL: Zeitaufwand bis jetzt war nicht übermäßig, deswegen sollte es gehen

Kurzes Update: Problem Domain Idee "Roomies"

- <http://www.mindmeister.com/264681082>
 - Kann ich Lese-Zugriff bekommen? kevin.gaunt@gmail.com
 - --> Dropbox
- MAL: Stellt vor
- KGA: Funktionsumfang i.O.
 - **WG vergleichen** wäre auch noch interessant
 - Team: wird wohl nicht gemacht, aber im Hinterkopf behalten, coole Idee!
- HRU: Funktionsumfang i.O.
- (*ausserhalb des BA-Themas*) KGA: Gameification interessiert euch?
 - KGA: hat paar Bücher gekauft vor paar Jahren
 - könnte falls Interesse besteht, ausgeliehen werden

Offene Fragen

- TDD: OK?
 - MAL: stellt vor
 - HRU: Ok.
- Vagrant: Developer Provisioning -> OK?
 - MAL: stellt vor (was ist vagrant etc.)
 - HRU: interessant, ist ok
- Was muss alles in die Dokumentation?
 - Abstract
 - Management Summary
 - Analyse über Inception Phase bzw. Evaluationsphase
 - Prozess wie man auf Technologie-Entscheidung gekommen ist kurz beschreiben
 - 5-6 Seiten (HRU)

- Projektplanung
- Use Cases
- Guidelines
 - Code / Test guidelines
- SAD
- Installationsanleitung
- Testbericht (Coverage etc.)
 - Systemtest (Anforderungsdokumentation!)
- Persönliche Berichte
- Eigenständigkeitserklärung
- Poster! (KGA)
- Für Broschüre: Nicht ganz Abstract (bisschen länger als Abstract)
 - bisschen Marketing (Marktschreier)

Ausblick

- Fertigstellung Projektplanung
 - Milestones
 - Iterationen
- Guidelines
 - Coding Guidelines
 - Testing Guidelines
- Software Architecture Document
- Domain Modellierung
- Minimale Anforderungsdokumentation
 - funktional & nicht funktionale Anforderungen
- Use Cases
 - nicht im Detail, kleine Abschnitte pro Use case
- kleiner Architekturprototyp
 - evtl. sehr einfach mit Scaffolding

TODO

- Mindmap export
 - [Done] Die Mindmaps wurden in die Dropbox unter
1_Project Management gespeichert

Weekly Meeting - 13.03.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- SA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: AJO
- Protokoll: MAL

Agenda

- Letztes Protokoll
- Projektplanung
- Guidelines
 - Coding Guidelines
 - Testing Guidelines
- Software Architecture Document
- Domainmodel & ER Diagramm
- Anforderungsdokumentation
 - funktional & nicht funktionale Anforderungen
- Use Cases
- Architekturprototyp

- Offene Fragen
- Ausblick, nächste Schritte

Letztes Protokoll

- Ist i.O.
- Beschlüsse in einem Abschnitt zusammenfassen

Projektplanung

- Phasenplanung
 - Erklären warum Inception & Elaboration überlappen
- Milestones
 - KGA: Bezeichnung Milestones könnten sprechender sein

Guidelines

- AJO: JavaScript Guideline von Airbnb
- AJO: Testing Guideline bis Ende Woche
- KGA: Warum Airbnb Guideline?
 - MAL: Verbreitet in Community, schon länger Bestand

Anforderungsdokumentation

Funktionale & Nichtfunktionale Anforderungen

- HRU: Einleitung für Requirements
- KGA: F5: WG verlassen statt nur "Verlassen"
- MAL: NF5: Nicht nur ROCA sondern auch Tilkov Erwähnen

Use cases

- HRU: Mapping auf Requirements machen
- AJO: Muss Registrieren & Login auseinandergekommen werden?
 - HRU: Nein, Fallunterscheidung kann im Use Case geschehen

- AJO: Erläuterung wie "Invite with Link" funktioniert
 - HRU: Expliziter erklären dass die Links ausserhalb des Systems verteilt werden
- AJO: Kann "UC10 Benutzer verwalten" so zusammengefasst werden?
Enthält ja CRUD & Rechtevergabe
 - HRU: Kein Problem, einfach im Use Case wieder per Fallunterscheidung erwähnen
 - HRU: Casual Style wählen --> wikipedia suche von gestern ;)
- HRU: Es muss klar sein, wie der Link zur Einladung verteilt/kopiert werden kann
- AJO: Akteure
 - HRU: Facebook fehlt
- HRU: Neue Use Case Formulierung ist gut so
- AJO: Muss extend-Use Case einzeln beschrieben werden?
 - HRU: Ja

Technische Architektur

Domainmodel

- AJO: Deutsche/Englische Begriffe
 - HRU: Übergänge deutsche/englische Begriffe klar definieren
 - HRU: Konzepte Beschreiben
- AJO: Erklärung Rollenkonzept
 - AJO: Admin-Rolle wird als nice to have klassifiziert
 - AJO: Resident & CommunityAdmin sollen umgesetzt werden
 - AJO: User, welcher Community erstellt, wird automatisch CommunityAdmin
 - HRU: Problematik Rollenmodellierung
 - Link Security Patterns: <http://swissmanu.github.com/hsr-apf-2013/patterns.pdf>, Abschnitt 1.2
 - KGA: Naming "CommunityAdmin" unglücklich... Was hat er mit dem normalen Admin zu tun? Besser "WG Verantwortlicher"

- MAL: Pragmatischerer Vorschlag für Rollenkonzept
- AJO: Erläuterung Achievement-System
 - HRU: Braucht es Verbindung zwischen Rule <-> Task?
- HRU: Domainmodell braucht Attribute!

ER Diagramm

- HRU: Multiplicity RoleToUser <-> User falsch
- AJO: Überspringen da Dank neuen Erkenntnissen überholt

Software Layers

- Business Layer:
 - Businesslogic:
 - HRU: Validation gehört eher in Models...?
 - HRU: Was ist API?
 - MWE: Service API, in unserem Falle REST
 - HRU: API/Controllers: Wirklich auf gleicher Ebene?
- MAL: Presentation Layer für Server & Client differenzieren
- KGA: Layerdiagramm sollte Logging & Security nicht als einzelne "Cross Cutting" Komponente darstellen
- HRU: "Tiefere" bitte in UML Notation
 - Komponentendiagramme
 - Sequenzdiagramme für Kommunikation

Architekturprototyp

- MWE: Warum kein Sails?
 - Hauptgrund: ORM unbrauchbar (keine Relations etc.)
 - Testability kein Thema im Moment
 - Schlechte Dokumentation
- MWE: Entscheid am letzten Freitag:
 - express.js

- HRU: Erklären warum kein Sails sondern nun express verwendet wird
- MWE: Warum JugglingDB als ORM?
 - Relations funktionieren
 - Viele DB-Adapter vorhanden
 - Weniger optimal im Moment: PostgreSQL Adapter erstellt keine Foreign Keys
- MWE: Erläuterung "Passport"
 - Authentication mit verschiedenen Identity Providers (Facebook, Twitter, Google, ...)

Offene Fragen

- Urheberrechtsvereinbarung nötig?
 - HRU: Ja, machen wir um einfach sicher zu sein

Ausblick

- Elaboration abschliessen: Siehe Beschlüsse

Varia

- KGA: InfoQ Vortrag Video Mitschnitt von Stefan Tilkov ist interessant!

Beschlüsse

- Projektplanung
 - Kurz Sachverhalt erklären, warum Phasen überlappen
 - Milestones sprechender benennen sobald Inhalt fix ist
- Technische Architektur
 - Zusätzlicher Abschnitt "Node.JS & express.js"-Einführung im Kapitel technische Dokumentation
 - inkl. Erklärung warum kein Sails sondern nun express verwendet wird
 - Rollenkonzept "pragmatisieren"

- Attribute im Domainmodell hinzufügen!
 - Achievementmodellierung "pragmatisieren"
 - Komponentendiagramme für genauere Abhängigkeiten (spätere Projektphase)
 - Sequenzdiagramme
 - ERM gem. neuem Domainmodell überarbeiten
 - Layerdiagramm sollte Logging & Security nicht als einzelne "Cross Cutting" Komponente darstellen
- Transition zwischen Sails & express.js erklären/verkaufen

Weekly Meeting - 20.03.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- BA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: MWE
- Protokoll: AJO

Agenda

- Letztes Protokoll
- Reviewergebnisse Dokumentation
- Vagrant Installation
- Offene Fragen
- Ausblick, nächste Schritte

Letztes Protokoll

- Ist i.O.

Reviewergebnisse Dokumentation

- [HRU] Gegenleser wird Prof. Dr. Ruedi Stoop sein

Kap. 2 Einleitung

- [HRU] Involvierte Personen entfernen
- [MAL] Kapitel ganz entfernen?
 - [HRU] Ja

Kap. 3 Analyse der Aufgabenstellung

- [HRU] Produktentwicklung ist ok. vielleicht in Anhang (nicht sicher)
- [HRU] Architekturrichtlinien: sehr gut
- [HRU] Technologieevaluation: insgesamt gut
 - Bewertungskriterien TK1: für Neueinsteigen erwähnen (Unterrichtsmaterial)
- [HRU] versucht spannend zu schreiben, manchmal zu 'süffig'
- [HRU] Begriff Realtime Applikation definieren.
 - [MAL] ist im Glossar
 - [HRU] referenzieren

Kap. 4 Anforderungsspez.

- [HRU] ist OK

Kap. 5 Technische Architektur

- [HRU] bisher i.O.
 - [HRU] JavaScript noch schwer verständlich
 - [HRU] [MAL] Code-Beispiele in Zukunft 'lesbarer', selbsterklärend darstellen.

Sonstiges

- [BA-Team] review des Dokuments bis Ende Iteration wäre gut

Facebook

- [MWE] Facebook Dev. stellt die Möglichkeit Dummy-User zur erstellen
 - für Test sehr nützlich
 - <https://developers.facebook.com/apps/>

Vagrant Installation

- [HRU] MacBook Air
 - alte vagrant box wurde entfernt (\$ vagrant box delete ba-box virtualbox)
 - clone ba projekt von github
 - vagrant image wurde neu herunter geladen und gestartet (\$ vagrant up)
 - [MAL] Forwarding Ports sind ersichtlich. Sind die Ports, die von der VirtualMachine auf der locale Machine weitergeleitet werden.
 - Port sind bereits besetzt ... :/-> ein Reboot tut immer gut -> (\$ vagrant up) -> Problem solved
 - (\$vagrant ssh) auf machine verbinden
 - /vagrant wird mit lokale Machine geshared.
 - Inhalt des Projekts
 - (\$ npm install) installiert alle nötigen Komponenten
 - (\$ npm start) -> ergibt Fehler -> config.js fehlt
 - config.example.js duplizieren nach config.js
 - facebook infos in config.js anpassen
 - (\$ npm start) -> erfolgreich
 - auf lokale Machine: <http://localhost:9001> -> Server ist erreichbar
 - mit (\$ vagrant halt) kann die VirtualMachine heruntergefahren werden

Offene Fragen

- [MWE] Wie soll der Wechsel von Sails zu Express.js dokumentiert werden? So wie im moment i.O. oder soll Technologieevaluation dem

- späteren Entscheid bereits vorgreifen?
 - [HRU] Sails bereits in Evaluation integrieren.

Ausblick

- Anleitung für Full-Setup der Entwicklungsumgebung bis Ende Iteration

Beschlüsse

- Prototype-Beschrieb in Evaluation integrieren
- Kapitel 2 aus Doku entfernen
- Nächster Doku-Review Ende Iteration 1

Weekly Meeting - 27.03.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- BA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: MAL
- Protokoll: MWE

Agenda

- Letztes Protokoll
- Rückblick
- Ausblick
- Offene Fragen

Letztes Protokoll

- Wird so akzeptiert

Rückblick

- MWE: Verfeinerung Architektur (Code Demo)

- keine Fragen dazu (vielleicht später)
- MAL: UI Design (Mockup Demo)
 - KGA: Kleine Labels statt grosse Beschreibung
 - KGA: Beschreibung on Focus oder mit Hilfeicon(Klick - auf Mobile) (zu viel Text)
 - könnte aber auch zuviel Abstand sein (zw. Label und Input-Field)
 - MAL: Evtl. auch Beschreibung wie die Error-Meldungen (aber mit Grau)
 - KGA: Würde mit Floating Feld (on Focus) arbeiten
 - MAL: Zeigt auch Mobile-View
 - KGA: "Du wohnst noch in keiner WG" - der User wird sagen "DOCH!"
 - Neutralere Texte verwenden (nicht System addressiert den User)
 - KGA: Habt ihr Twitter Bootstrap verwendet?
 - MAL: nein - [Zurb Foundation](#)
 - Weil SASS statt LESS
- AJO: Umsetzung Use Cases (Live Demo)
 - Anmelden
 - WG erstellen
 - -> keine Fragen
- MAL: Supporting Actions
 - Thesis: Baselining
 - Thesis: Einarbeitung erste Reviewbefunde HRU
 - Code Dokumentation: NaturalDocs

Diskussion

- HRU: Zeitaufwand?
 - Ist in Ordnung
- KGA: Kontext - wo sind wir insgesamt?

- KGA/HRU: Mitte kommt langsam
- HRU: Daten nach Abschluss
 - Datum für Prüfung
 - 20min Präsentation
 - 40min Fragen/Prüfen
 - Einführung in Thema für Gegenleser vorab (Kurze Präsentation)
 - eher informal
 - vor Abschluss (mitte Semester)
 - Evtl. Ende Construction 3 (anfang Mai)
- KGA: Wie ist das für euch, seid ihr unter Druck oder gehts?
 - AJO: ROCA-Prinzipien wichtig, ansonsten gut
 - MWE: das selbe
 - MAL: relativ optimistisch
 - KGA: will auf die Frage hinaus, ob der Slow Start sehr hinderlich war oder nicht
 - Team glaubt nicht

Ausblick

- Abschluss Construction 1
 - Termin: Sonntag 31.03.2013
 - Baseline M3 wird reviewbereit sein
 - Einschätzungen?
- Ab Montag: Construction 2
 - Use Case: WG beitreten
 - Use Case: Aufgabe erstellen
 - Use Case: Aufgabe erledigen
- Ongoing: Code Reviews
- Abwesenheit MAL 02. - 05.04.2013 (Militär)

Beschlüsse

- Wiki: Repository kaputt? Fixen (MWE) [DONE]

- Formulare werden nochmals überarbeitet (MAL) [DONE]
- Button-Styling wird überarbeitet (MAL) [DONE]
- Wording "Du wohnst noch in keiner WG" etc. anpassen (MAL) [DONE]
- Facebook Login langsam? überprüfen (MWE) [Wird auf nächsten Meilenstein verschoben]
 - Performance allgemein
- Zeitpunkt für Prüfung (alle, Terminvorschlag durch Team)
 - evtl. zwischen 24.6. und 28.6. (Geht nicht, HRU hat dann Ferien)
 - zwischen 8.7. und 12.7.? HRU fragt nach
 - HRU fragt Daniel Hiltebrand und Ruedi Stoop wann er Zeit haben würde
- Zeitpunkt für Information Gegenleser (alle, Terminvorschlag durch Team)
 - Evtl. Ende Construction 3 (anfang Mai)
- Dokumentation
 - *Doku Architektur* (MWE) [Wird auf nächsten Meilenstein verschoben]
 - **Installationsanleitung** (MWE) [DONE]
- Vagrant Image aktualisieren (MWE) [DONE]
- ROCA-Prinzipien in Dokumentation aufnehmen (MAL) [DONE]
 - und in Code veranschaulichen

Weekly Meeting - 03.04.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- SA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)

Abwesend

- Manuel Alabor (Militär)

Organisation

- Leitung: MWE
- Protokoll: AJO

Agenda

- Genehmigung des letzten Protokolls
- Feedback Review
- Ausblick

Genehmigung des letzten Protokolls

Das letzte Protokoll wurde von allen Teilnehmern akzeptiert.

Feedback Review

- [HRU] wird PDF mit Kommentaren versenden
- [HRU] auf gutem Weg

Kap. 2

- [HRU] Begriff 'Produktentwicklung' ändern mit 'Beispielapplikation' o.Ä.
- [HRU] Branding: von anfang an Logo zeigen
- [HRU] Gut referenziert
- [HRU] Technologieevaluation: ist gut

Kap. 3

- [HRU] Braucht ein bisschen Text
- [HRU] ist schon viel besser

Kap. 4

- [HRU] Software-Layer: zu jedem Kästchen braucht's Text

Kap. 5

- [HRU] ist Gut

Kap. 6

- [HRU] Wort 'Phase' wird teilweise für 'Iteration' gebraucht
 - -> einheitlich pflegen

Sonstiges

- [MWE] BA geht eigentlich um die ROCA-Prinzipien. In Thesis wird am Anfang viel über die Anwendung gesprochen...
 - [HRU] Kap. 2.2 vor 2.1
 - [HRU][MWE] Ordnung überarbeiten
- [KGA] Persönliche Meinung einbringen (Kap. 5). Wäre schön, denn es gibt genug Dokumentationen über z.B. Cookies

- spannender zum lesen
- einfacher zum schreiben
- [KGA] nimmt sich vor die Thesis am Fr zu lesen
- [AJO] Installationsanleitung auf github ergänzen

Ausblick

- Construction 2
 - ROCA Prinzipien
 - UC3: WG Beitreten
 - UC5: Aufgabe erstellen
 - UC7: Aufgabe erledigen
- Doku Architektur
- FB Login Performance / Performance allgemein
 - [HRU] Was könnten die Ursachen sein?
 - [KGA] Vagrant hat zu wenig Speicher(?)
 - [MWE] Performance FB-Seitig, da in Sandbox
 - etc.
- HRU ist am Mi 10.4 um 10h verhindert
 - Meeting wird auf 15:10 verschoben
 - KGA ist noch unsicher
- HRU ist die Woche vom 24.4 abwesend
- MWE ist am 24.4 ebenfalls abwesend

Beschlüsse

- Thesis anhang der Kommentare von HRU überarbeiten
- Reihenfolge der Kapitel ändern, damit die ROCA-Prinzipien in vordergrund treten.

Neue Tasks

- (MWE)
 - FB Login Performance [DONE]
 - Sessions [DONE]
 - SAD
 - Doku überarbeiten [DONE]
- (AJO)
 - UC3
 - SAD
 - Doku überarbeiten
- (MAL)
 - SAD
 - Doku überarbeiten

Weekly Meeting - 10.04.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
- BA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)
- Abwesend
 - Kevin Gaunt (KGA)

Organisation

- Leitung: AJO
- Protokoll: MAL

Agenda

- Genehmigung des letzten Protokolls
- Rückblick
 - MAL: Backbone on Server
 - AJO: UC3 (WG Beitreten)
 - MWE: Dokumentation
- Feedback Review
- Aktuelle Themen
 - Update bzgl. Termine Vortrag/Prüfung und Introduction des Gegenlesers?
- Ausblick

Genehmigung des letzten Protokolls

Das letzte Protokoll wurde von allen Teilnehmern akzeptiert.

Rückblick

MAL

- MAL Erzählt das Konzept von Backbone on Server (Soweit wie möglich gleicher Code auf Frontend und Backend)
 - Aufgrund vom Unobtrusive JavaScript Requirement
 - rendr von AirBNB ist nicht ausgereift und nicht anwendbar für die BA
 - selber gestartet damit, ist aber noch nicht fertig - aber auf gutem Weg
 - Mit Backbone als sehr flexibles und leichtgewichtiges Framework ist es eher möglich als z.B. mit Ember.js
- HRU fragt: hat das schon jemand gemacht?
 - MAL: rendr verwendet auch Backbone
- aktueller Stand: Vorallem Refactoring von Client/Server Unterschiede heute
- Nested Views der nächste Task

AJO

- Umsetzung Use Cases
 - WG beitreten
 - AJO erläutert Arbeitsfortgang mit geklärten Unklarheiten
 - Demonstration Umsetzung Use Case
 - Aufgabe erstellen
 - Aufgabe bearbeiten
- AJO meint, ein Storyboard mit allen Views inkl. zugrundeliegenden REST Resources etc. wäre wünschenswert
- AJO demonstriert die Routen-Definition der Community-Komponente

MWE

- MWE erläutert die Überarbeitung der Dokumentation
 - Reviewbefunde von HRU eingearbeitet
 - Mehr oder weniger Abgeschlossen
- HRU Evtl. Tutorialverzeichnis bereits in der Architekturdoku?
 - MAL Ein Kapitel im Anhang "How To Get Startet With Our Technologies" wäre evtl. passender
 - HRU Stimmt zu

Termine

- HRU Anfangs Mai eine kurze Einführung für Ruedi Stoop
 - Am optimalsten wäre ein Montag
- HRU Prüfungstermin passt in der vorgeschlagenen Woche leider nicht
 - Muss doch in die Prüfungssession verlegt werden

Ausblick

- Abschluss Construction 2 bis Sonntag Abend
 - UC5 & UC7 (Aufgaben)
- Dokumentation erweitern
- "Backbone auf Server" weitertreiben
- HRU: Falls neue Dokumente zum durchlesen, dann bitte bis zum 16.04. liefern

Beschlüsse

- Installationsanleitung auf Github ergänzen/verfeinern
- Im Anhang eine Leseliste zusammenstellen, wie man sich am Besten in die verschiedenen Technologien einarbeiten kann
- Terminvorschläge für Einführung Ruedi Stoop
- Terminvorschläge für mündl. Prüfung (innerhalb Prüfungssession, verschiedene Wochen vorschlagen)

- Mittwoch tendenziell schlecht
- HRU 9. - 11.09. nicht möglich

Weekly Meeting - 17.04.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
- SA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Abwesend

- Kevin Gaunt (KGA)

Organisation

- Leitung: MAL
- Protokoll: MWE

Agenda

- Genehmigung des letzten Protokolls
- Rückblick
- Feedback Review
- Ausblick

Genehmigung des letzten Protokolls

Das letzte Protokoll wurde von allen Teilnehmern akzeptiert.

Rückblick

MWE

- Dokumentation
- [Coveralls.io Coverage Statistik für Bachelorarbeit Beispielapplikation](#)

MAL

- Backboneify
 - soweit i.O. - guter Fortschritt

AJO

- DEMO
 - UC3: WG Beitreten
 - UC5: Aufgabe erstellen
 - UC7: Aufgabe erledigen
 - alles i.O.
 - MAL: logout button auf mobile funktioniert nicht

Feedback Review

- wird später eingereicht durch HRU/KGA

Ausblick

- Construction 3
 - Projektrichtlinien
 - Gamification
 - Umstellung auf BackboneJS
 - HRU: d.h. es ist noch im Experimentierstadium?
 - MAL: es ist noch nicht fertig, aber auf gutem Weg
 - MAL: falls es nicht geht, müsste die Beschreibung zur Umsetzung der entsprechenden Projektrichtlinien angepasst werden

Beschlüsse

- Termine für Besprechung mit Ruedi Stoop: 6. Mai und 13. Mai als Vorschlag
- Termine für Prüfung: 27.08. - 02.09. als Vorschlag

Neue Tasks

- HRU: Terminfindung für Prüfung / Besprechung

Weekly Meeting 24.04.2013

Teilnehmer

- HSR/IFS
 - Kevin Gaunt (KGA)
- BA Team
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)
- Abwesend
 - Hans Rudin (HRU)
 - Michael Weibel (MWE)

Organisation

- Leitung: AJO
- Protokoll: MAL

Agenda

- Genehmigung des letzten Protokolls
- Rückblick
- Aktuelle Themen
 - Replanning
- Ausblick

Genehmigung des letzten Protokolls

Das letzte Protokoll wurde von allen Teilnehmern akzeptiert.

Rückblick

AJO

- AJO Erläutert Fortschritt von Ranking List
 - Erläuterung Row Queries
 - KGA Klar Aussagen was Problemstellung mit Row Queries ist und Dokumentieren (Warum und wie gelöst?)

MAL

- MAL demonstriert Barefoot Demo Applikation
 - Erklärung Aufbau
 - Erläuterungen zu View & Model Restoration auf Client

Feedback Dokumentation

- KGA "Das wird ziemlich cool!"
- KGA Idee: Zu Beginn der Richtliniendemonstration könnte man die folgenden Ergebnissen noch einmal in einer Ähnlichen Tabelle wie in der Aufgabenanalyse zusammen
- KGA Code Beispiele: Evtl. auch mit Verweisen auf Codefiles arbeiten
- KGA Perfekter Schliff fürs Literaturverzeichnis: Wikipedia mit Permalink referenzieren. Kein Must-Have, aber als Anmerkung :)
- KGA Feedback als Mail
- AJO/MAL Dokumentation muss nicht zwingend einem neuen Review unterzogen werden; zu wenige Neuerung. Besser in ein bis zwei Wochen.
 - KGA OK so

Replanning

- Gameification eine Woche später
 - Grund: Komplettes Team für Kernkomponente nötig
 - Letzte Woche Probleme mit Testing, darum Verzögerung
 - KGA: OK so

- "Use Case: Rangliste" dafür bereits diese Woche
 - KGA: OK so

Ausblick

- Abschluss Construction 3
 - Ranking List
- Beginn Umsetzung Backbone/Barefoot
- Übersichtstabelle Richtliniendemonstration

Beschlüsse

- Dokumentation erst in ein bis zwei Wochen wieder reviewen

Weekly Meeting - 01.05.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- BA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: MWE
- Protokoll: AJO

Agenda

- Letztes Protokoll
- Rückblick
- Ausblick
- Offene Fragen

Letztes Protokoll

- Wird so akzeptiert

Rückblick

- MAL: Backbone on the server

- [MAL] Erklärung welche Probleme (CookieAdapter) aufgetaucht sind und welche Vorteile es bringt
- MWE: Dokumentation
 - [MWE] Bringt die Doku aufgrund der Kommentaren auf dem neusten Stand
 - [MWE] UC 'Task bearbeiten' konnte bereits abgeschlossen werden
 - Test machen Probleme
 - Tests reagieren komisch...
 - Viel Zeit geht verloren
 - Falls es bis Fr nicht funktionieren sollte, wird auf code coverage verzichtet
 - [HRU] ok
- AJO: Use cases
 - Gamification wurde in Construction 4 verschoben, dafür wurde UC Ranking anzeigen vorverschoben.
 - Begründung kann im Protokoll vom [[24.04.2013 - Weekly Meeting]] nachgelesen werden
 - [MWE] 'Task Bearbeiten' wird kurz vorgestellt

Diskussion

- [HRU] Könnte die Dokumentation möglicherweise publiziert werden?
 - [MAL] Library wird OpenSource zur Verfügung gestellt
 - [MAL] Slides könnten erstellt werden...
 - [HRU] Publikation in JavaSPEKTRUM?
 - [MWE] Kleine Zusammenfassung
 - [HRU] Im Normalfall bis zu ~3 Seiten
 - [KGA] Know How für Tipps ist vorhanden
 - [BA-Team] Magazin JavaSPEKTRUM wird angeschaut

Ausblick

- Construction 4:
 - Gamification

- Aufgabe bearbeiten (DONE)
- WG Auflösen
- Backbone on the server

Offene Fragen

- Status Datum für Zwischenpräsentation etc.?
 - 1. und 13. stehen zur Auswahl. Wird heute (01.05.13) von HRU abgeklärt.

Beschlüsse

- Doku wird erst wieder Ende Construction angeschaut
- [HRU] Termin für Zwischenpräsentation wird abgeklärt

Weekly Meeting - 08.05.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- BA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: MAL
- Protokoll: MWE

Agenda

- Letztes Protokoll
- Rückblick
- Ausblick
- Offene Fragen

Letztes Protokoll

- Wird so akzeptiert

Rückblick

- MWE: Gamification

- MAL: Kann das auch Zeitgesteuert (mit cronjob) gemacht werden?
 - KGA: Aus welcher Überlegung?
 - MAL: damit z.B. pro Woche ein Achievement vergeben werden kann
 - MWE: ja, das ist möglich über z.B. eigene Events
- AJO: Features Construction 4
 - Community löschen
 - Offen: Div. checks nach löschen der Community
 - Mit ranking gekämpft
 - einzelne Bugs in Sequelize gefixt
 - Keine Raw SQL Query mehr (datenbank unabhängig)
 - KGA: Wie ist das "admin"-flag gelöst?
 - AJO: Mit isAdmin flag direkt im Resident --> weil ein Resident nur in einer Community sein kann
- MAL: Portierung zu Backbone
 - Ein grösserer Teil des Controller codes muss wohl neugeschrieben werden
 - zeigt Struktur
 - KGA: Könnte das auch gezeigt werden?
 - HRU: Evtl. ja: ob Backbone genutzt wird oder nicht

Diskussion

- Vorbereitung Präsentation Gegenleser
 - MAL: was sind die Erwartungen?
 - MAL: macht es Sinn eine Technische Einführung zu geben?
 - HRU: ja schon auch
 - HRU: weiss aber nicht ob er JavaScript kennt
 - HRU: Grundprinzipien erklären (MVC?)
 - Was sind die Leistungen des Teams?
 - Was wird erreicht mit der Arbeit?
 - KGA: Sehr wichtig: Herr Stoop muss die Dokumentation als wichtiger

als die Beispielapplikation erachten nach der Präsentation

- HRU: Powerpoint Präsentation mit gutem Aufbau
 - HRU: hilft insbesondere um auch neue Leute in die Dokumentation einzuführen
- KGA: Klar machen, dass es eine Forschungsarbeit ist (akademisch) und nicht eine reine Website-Entwicklung
- KGA: Darf ruhig auch technisch werden an einem Ort, aber eher ein kleiner Teil rauspicken und zeigen

Ausblick

- Abschluss Construction 4
- Fortführung Portierung Backbone
- Vorbereitung Präsentation Gegenleser

Beschlüsse

Weekly Meeting - 15.05.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- BA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: AJO
- Protokoll: MAL

Agenda

- Letztes Protokoll
- Rückblick
- Ausblick
- Offene Fragen

Letztes Protokoll

- i.O.

Rückblick

- AJO: Abschluss Construction 4

- Use Cases abgeschlossen
- Dokumentation wurde nicht weiter verfeinert
- Backbone/Barefoot ongoing
- MWE: Zwischenpräsentation
 - Siehe *Diskussion*
- MAL: Backbone on the Server
 - Demo Vergleich: "Alter" & neuer API Controller Code

Diskussion

- Abschlusspräsentation:
 - Schriftsprache
 - Total 60min
 - 20min Präsentation/Demo
 - 40min Fragen
 - HRU: Gezielt visualisieren wo sinnvoll
 - HRU: Ideen Aufbau:
 - Thema
 - Prinzipien nicht aufzählen, sondern grafisch darstellen:
"Quelle ROCA" & "Quelle Tilkov" führen zu eigenem Richtlinien Pool
 - Nicht alle Prinzipien erklären, sondern schlechtes Beispiel zeigen , dann zeigen wie es mit der Arbeit umgesetzt werden kann
 - Technologieevaluation, Bleeding-Edge irgendwie visualisieren
 - Was heisst shared code base?
 - KGA: Soll zeigen, warum wir Bleeding-Edge gewählt haben
 - HRU: Demo sollte enthalten sein
 - MAL: Einführung geben "Es gibt eine WG, ein Mitwohner möchte organisieren" und dann in Demo gehen

Ausblick

- AJO: Construction 5:
 - Sharing Funktionalitäten
 - Migration "alter Code" auf Backbone/Barefoot
 - KGA: Migration Unit Tests
- Dokumentation

Offene Fragen

- AJO: Ist Einführung in ein bestimmtes Thema für HRU gewünscht?
 - HRU: Eher nein
 - HRU: Beschreibung der Frameworks sollte im Rahmen des Kapitels "Technische Architektur" abgehandelt werden
 - HRU: "Technische Architektur" sollte aber Sequenzdiagramm(e) enthalten wie ein Request bspw. auf Client & Server gehandelt wird
- MWE: Java Spektrum?
 - MAL: Eher nicht im Rahmen der "normalen" BA; Runde stimmt zu
- Rahmenbedingungen für Schlusspräsentation?
 - Siehe *Diskussion*

Weekly Meeting - 22.05.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- BA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
 - Manuel Alabor (MAL)

Organisation

- Leitung: MWE
- Protokoll: AJO

Agenda

- Letztes Protokoll
- Rückblick
- Themen/Diskussion
- Ausblick
- Offene Fragen

Letztes Protokoll

- i.O.

Rückblick

- Portierung nach Backbone
 - [MAL] Alle mit Portierung beschäftigt
 - (MAL) & (AJO) API
 - (MWE) Unit Tests
 - [MWE] Code coverage konnte erhöht werden: ~80%
 - [MWE] geht weiter so bis Ende Iteration

Themen/Diskussion

- MAL: Promises & Server Rendering
 - [MAL] Demo
 - [MWE] Sequenzdiagramm wäre hilfreich
 - (Promise == Future-Pattern)
- MWE: Mock Testing mit JavaScript
 - [MWE] Demo
 - [MWE] Mocking braucht kein Framework mehr, da jetzt die API direkt getestet werden kann.
 - Nun sind es wirkliche Unit Tests :)

Ausblick

- Abschluss Portierung
 - Funktionalität beenden
- Dokumentation
 - Je nach Stand der Portierung
 - [MWE] Sequenzdiagramm sind wichtig
- (MAL) am 29.5 abwesend (Militär)
- (KGA) am 29.5 abwesend (Zahnarzt)
 - Nachmittags möglich?
 - Meeting auf 14h00 verschoben

Beschlüsse

- Meeting vom 29.5 auf 14h00 verschoben

Weekly Meeting - 29.05.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- BA Team
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)
- Abwesend
 - Manuel Alabor (MAL) (Militär)

Organisation

- Leitung: AJO
- Protokoll: MWE

Agenda

- Letztes Protokoll
- Rückblick
- Themen/Diskussion
- Ausblick
- Offene Fragen

Letztes Protokoll

- Wird so akzeptiert.

Rückblick

- Meilenstein Construction 5: Aktueller Stand
 - Warum kein Tag?
 - Mischt sich bereits mit dem Task Bugfixing
 - AJO erklärt warum
- MWE: Demo Barefooted-Roomies
- Abstract
 - Screenshot statt Logo
 - evtl. mit Mobile & Desktop
 - ▪ immer noch grosses Logo (speziell hervorheben)
- Poster
 - mehr Visualisierung (z.B. Back-Button zeigen)
 - Roomies Screenshot
 - Braucht die Layer?
 - Ergebnis: Ein Statement! Kontrovers evtl.
 - Studie <-> Beispielapplikation interagierend
 - Wer ist Zielgruppe?
 - ROCA fehlt? Evtl. als weiterführender Link
 - Allgemein wo sind mehr Infos zu finden?
 - QR-Code? TinyURL?
 - Gleicher Code auf Client & Server - evtl. auf Poster bringen
- Abstract für HSR nicht das gleiche wie Abstract bei der BA (keine Bilder z.B.)

Themen/Diskussion

Ausblick

- Abschluss Portierung bis spätestens Ende dieser Woche
 - AJO: Community Delete
 - AJO: Ranking Query improvement
 - AJO: Mark task as done

- MWE: Flash messages
- MWE: Ajaxify
- MWE: Validator messages in Form
- **Merge Barefoot branch to master branch**
- Transition
 - Vagrant Paketierung
 - Source Paketierung
 - Bugfixing (Transition)
 - Dokumentation

Offene Fragen

- 14.Juni 2012 findet vom 16 bis 20 Uhr die Ausstellung der Bachelorarbeiten 2013
 - Wie sieht das aus?
 - --> Alle Bachelorstudenten sind bei ihren Postern und erklären an interessierten das Thema (müssen aber nicht immer alle da sein, aber min. einer schon)
- Dokumentationsreview?
 - spätestens bis Freitag Mittag (7.6.) Dokumentation schicken (wenn möglich früher)
 - spezifisch sagen was genau anschauen
 - falls fragen: direkt auch grad fragen
 - dafür am Montag, 10.6.13, 10 Uhr

Beschlüsse

- am nächsten Mittwoch (5.6.) keine Sitzung
- nächstes Meeting: Montag, 10.6.13, 10 Uhr

Last Weekly Meeting - 10.06.2013

Teilnehmer

- HSR/IFS
 - Hans Rudin (HRU)
 - Kevin Gaunt (KGA)
- BA Team
 - Manuel Alabor (MAL)
 - Michael Weibel (MWE)
 - Alexandre Joly (AJO)

Organisation

- Leitung: MAL
- Protokoll: AJO

Agenda

- Letztes Protokoll
- Rückblick
- Besprechung Reviews
- Offene Fragen
- Ausblick

Letztes Protokoll

- Wird so akzeptiert

Rückblick

- Abschluss Implementation
 - [MWE] Abgeschlossen
 - [MWE] Link & Anleitungen wurden verschickt
 - [MWE] Zusätzliche Task (Benutzerverwaltung & Post to FB) aufgrund von mangelnder Zeit nicht gemacht
 - [MWE] Anwendung wurde auf Heroku veröffentlicht:
<http://roomies.herokuapp.com>
- Dokumentation
 - Restrukturierung: Req. Spec. in Anhang
 - HRU mit ähnlichen Gedanken

- Restrukturierung: Projekt- & Qualitymanagement in Anhang
 - [HRU] ist ok

Besprechung Reviews

Abstract

- [KGA] kann gekürzt werden und sollte ok bleiben
- [KGA] könnte prägnanter sein

Kap. 2

- [KGA] sieht Zusammenhang zwischen Progressive Enhancement und Responsive Design nicht
- Bewertung & Einschätzung: Nachteile?
 - Vor- und Nachteile ganz kurz zeigen
- 2.2.1: RP15 No Duplication: HTML & CSS kann auch viel Duplication haben

Kap. 4 Technische Architektur

- [MWE] ist noch nicht fertig
- [HRU] Kapitel ist eine Herausforderung bei so vielen Technologien
 - Wie sollen die Technologien erklärt werden...
- [MWE] braucht's die Software Layers wirklich?
 - [KGA] unbedingt drin lassen
- [KGA] Quellcode: warum wurden keine Konstanten gebraucht?
- [HRU] Barefoot muss noch erklärt werden
- [KGA] GET Request & Klick auf Menü in Sequenzdiagramm.
 - Wäre schön wenn der selbe Fall gezeigt wäre

Kap. 5

- RP1: [KGA] 'intern' und 'extern' API umschreiben
- RP2: [KGA] Diskussion muss angepasst werden.
- RP5: [HRU] Vielleicht begründen: aus Zeit Gründen, ...
- RP8: [KGA] 'Cookies für ... sollte verzichtet werden' für diesen Fällen eignet sich ...
- RP13: [KGA] IE8 Screen Shot vergleichen mit Soll
- RP14: [KGA] Vergleich mit Mila oder Google sind gut

Kap. 6

- [HRU] Planung kann schlecht verfolgt werden, ob alles wie geplant abgelaufen werden konnte --> Iterationsassessment

Kap. 7

- Testing: Resultat zeigen

Abschliessende Bewertung

- Empfinden, Persönliche Sicht

Coding Style

- [MAL] nur referenzieren?
 - [HRU] ja, das reicht
 - [HRU] dürfen es drin lassen
- [HRU] es wurden aber nur ~10% angepasst
- wird nur referenziert

Allgemein

- [MWE] Poster ebenfalls in Anhang?
 - [HRU] kann

Offene Fragen

- Wie ist das gemeint bzgl. den Meeting protokollen?
- Ist eine Auswertung der Zeitrapportierung gewünscht?
 - [HRU] wäre schön
 - BurnDown-Chart

Ausblick

- Fertig machen
 - SAD
 - Reviews
- [KGA] Mehr Beispiele, auch für die Ausstellung

Beschlüsse

- Coding Style aus der Doku nehmen und referenzieren

Protokoll

- Dieses Protokoll wird von allen Teilnehmer akzeptiert

