HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Bachelorarbeit

Architekturkonzepte moderner Web-Applikationen

Hochschule für Technik Rapperswil
Frühjahressemester 2013

Erstellt: 21. März 2013, 07:37

Autoren Manuel Alabor
Alexandre Joly
Michael Weibel

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

Our fancy abstract.. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Erklärung der Eigenständigkeit

Wir erklären hiermit,

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

Manuel Alabor *Alexandre Joly* *Michael Weibel*

Danksagungen

Inhaltsverzeichnis

Kapitel 1 **Management Summary**

1.1. Ausgangslage

1.2. Vorgehen

1.3. Ergebnisse

1.4. Ausblick

Kapitel 2 Analyse der Aufgabenstellung

Die Aufgabenstellung (siehe Anhang H) verzichtet bewusst auf funktionale Anforderungen an die zu erstellende Applikation. Weiter werden auch keine spezifischen Technologien zur Umsetzung vorgegeben.

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

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

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

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

Dieses Kapitel beantwortet im weiteren Verlauf dementsprechend folgende Fragen genauer:

1. Welche Schritte wurden im Bereich der Produktentwicklung durchlaufen? Auf welche Aspekte wurde besonders eingegangen?

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

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

2.1. Produktentwicklung

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

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

2.1.1. Workshop

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

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

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

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

Tabelle 2.1.: 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 (siehe Anhang E) 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.

2.1.2. Die finale Produktidee: Roomies

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

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

2.1.3. 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

Im Folgenden sind die Grundlegenden Stilelemente zur Referenz aufgeführt. Das Produktlogo wird zudem in verschiedenen, grössenoptimierten Varianten gezeigt.

Grundfarbpalette

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

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

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

Abbildung 2.1.: Branding Farbpalette

Logo & Logovariationen

Abbildung 2.2.: Roomies Logo im College Stil

Abbildung 2.3.: Roomies Logo in verschiedenen Grössen & Varianten

2.2. Architekturrichtlinien

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

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

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

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

2.2.1. Resource-oriented Client Architecture (ROCA)

Backend-Layer

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

Tabelle 2.2.: Die ROCA Architekturprinzipien: Backend

Frontend-Layer

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

Tabelle 2.3.: Die ROCA Architekturprinzipien: Frontend

Bewertung & Einschätzung

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

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

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

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

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

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

2.2.2. Building large web-based systems: 10 Recommendations

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

Tabelle 2.4.: Tilkovs Empfehlungen

Bewertung & Einschätzung

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

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

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

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

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

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

2.2.3. 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.

2.2.4. Richtliniendemonstration

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

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

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

Tabelle 2.5.: Mapping Architekturrichtlinien - Systemkomponenten

Nach erfolgreicher Umsetzung der Beispielapplikation gilt es die aufgestellten Demonstrationsstellen zu verifizieren und zu ggf. genauer zu beschreiben (Zeilennummer im Quellcode usw.).

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 Platzhirsche 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, Platzhirsche von einer engeren Auswahl auszuschliessen, war das Projektteam aufgrund der vorangegangenen Studienarbeit davon überzeugt, dass Java, insbesondere als Backendtechnologie, mit den "jungen Wilden" problemlos mithalten kann.

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

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

2.3.1. Bewertungskriterien & Gesamtbewertung

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

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

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

Tabelle 2.6.: Bewertungskriterien für Technologieevaluation

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

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

Abbildung 2.4.: Berechnungsformel Gesamtbewertung

2.3.2. Ruby

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

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

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

Quellcode 2.1: Negierte if-Abfrage in Java

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

Quellcode 2.2: Negierte if-Abfrage in Ruby

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

Bewertung Ruby on Rails

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

Tabelle 2.7.: 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, alles Andere als optimal wenn es darum geht, Architekturrichtlinien eindeutig und klar demonstrieren zu können.

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

2.3.3. Java

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

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

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

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

Tabelle 2.8.: Shortlist Analysekandidaten Java

Bewertungsmatrix

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

Tabelle 2.9.: 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* und *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 in länger in der Praxis bewähren können und sind lediglich genau in dieser Kombination evtl. weniger oft 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 [Goo] verdankt der früher für schlechte Performance und schlechte 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 [Goo] von Google hat 2009 zudem den Sprung

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

Auch im Bereich der Webframeworks gibt es auf diese Weise eine Menge an interessanten Libraries zu entdecken. Für die Evaluation im Bezug auf diese Bachelorarbeit werden folgende Frameworks genauer analysiert:

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

Tabelle 2.10.: Shortlist Analysekandidaten JavaScript

Bewertungsmatrix

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

Tabelle 2.11.: Bewertungsmatrix JavaScript Frameworks

Interpretation

Das Kandidatenfeld lässt sich grob in zwei Felder aufteilen:

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

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

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

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

Aufgrund der interessanten Ansätze welche Sails wählt, und dabei im Vergleich zu Express.js bereits ein ORM mitbringt, soll den Punkten zum Trotz *Sails* für JavaScript in der finalen Evaluationsrunde mit den restlichen Frameworks verglichen werden.

2.3.5. Entscheidung

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

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

Tabelle 2.12.: Finale Frameworkkandidaten für Technologieentscheidung

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

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

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

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

2.4. Evaluation Frameworks in Node.js

In der Technologie Evaluation zu JavaScript stachen die beiden Frameworks Express.js [Exp] und Sails.js [Balb] heraus.
Obwohl Express.js stabiler ist und weitaus mehr genutzt wird, ist Sails.js aufgrund der neuen Ideen und interessanten Ansätzen für einen ersten Prototyp ausgewählt worden.

2.4.1. Sails.js Prototyp

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

Sails.js verwendet Scaffolding um einerseits ein neues Projekt zu erstellen, andererseits auch um verschiedene Komponenten wie Model oder Controller zu erstellen. Wie im Entity-Relationship Diagramm beschrieben, werden u.a. ein Task und ein Resident Model (sowie entsprechende Tabelle) benötigt.
Um das ORM "Waterline" [Balc] zu testen, wurden diese beiden Models implementiert. Das Task-Model mittels Sails.js definiert sieht so aus:

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

Quellcode 2.3: Task Model in Sails.js

Mit einer Definition eines Models wird automatisch eine REST-API für dieses erstellt. Damit lassen sich einerseits CRUD-Operationen direkt über HTTP ausführen, andererseits existiert auch die Möglichkeit Socket.IO [Rau] zu aktivieren, um Models direkt von einem offenen Websocket zu verwenden.
Dieses Feature macht Sails.js sehr nützlich für Real-Time-Applikationen.

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

```
1 var TaskController = {
2   get: function(req, res) {
3     var id = req.param('id');
4     Task.find(id).done(function(err, task) {
5       User.find(task.userId).done(function(err, user) {
6         var response = {
7           'task': task,
```

```
8           'user': user,
9           'title': task.name
10        };
11
12        if (req.acceptJson) {
13          res.json(response);
14        } else if(req.isAjax && req.param('partial')) {
15          response['layout'] = false;
16          res.view(response);
17        } else {
18          res.view(response);
19        }
20      });
21    });
22  }
23
24};
25module.exports = TaskController;
```

Quellcode 2.4: Task Controller in Sails.js

In diesem Controller wird ein Task aufgrund des GET-Parameters “id” (Linien 3 und 4) geladen. Das Code-Stück zeigt die grosse Schwäche des ORMs Waterline [Balc]. Statt dass man auf dem “task”-Objekt direkt “.user” aufrufen könnte, muss man den Umweg über “User.find()” gehen.
Bei einem ausgereiften ORM würden solche Methoden wegen der Definition von Relationen direkt zur Verfügung stehen.

Der Controller antwortet auf eine Anfrage eines Browsers mit folgendem HTML:

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

Quellcode 2.5: Task Template

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

Schlussfolgerung

Wie bereits vorher angemerkt, ist das ORM von Sails.js nicht ausgereift. Weder Assoziationen zwischen Models [Bala] noch das setzen von Indizes ist möglich.
Für das Resident-Model ist es u.a. auch nötig, Facebook IDs zu speichern. Diese sind 64 Bit gross und wegen mangelhafter Unterstützung des ORMs wäre das gar nicht möglich.

Nebst dem ORM ist auch das Framework und die zugehörige Dokumentation wenig umfangreich. Auch die Community war zum Zeitpunkt der Evaluation (siehe Anhang F zur Technologieevaluation) sehr klein. Alles in allem spricht relativ viel dagegen, Sails.js weiter zu verwenden. Aus diesem Grund ist ein zweiter Prototyp unter Verwendung von Express.js erstellt worden.

2.4.2. Express.js Prototyp

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

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

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

```
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 };
```

Quellcode 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>
```

Quellcode 2.7: Template in Express.js

In den vorangegangenen zwei Quellcode-Listings 2.6 und 2.7 ist ersichtlich, dass der Applikations-Entwickler sehr grosse Kontrolle über Express.js hat.
Automatisch passiert in Express.js eigentlich nichts. Dies ist ein grosser Vorteil im Bezug auf die Veranschaulichung der Architekturrichtlinien.

ORM

Im Gegensatz zu den anderen evaluierten Frameworks ist bei Express.js kein ORM enthalten. Deswegen musste auch bzgl. des ORMs eine kurze Evaluation gemacht werden. Tabelle 2.13 zeigt die Kriterien und Gewichtung für die Evaluation des ORMs.

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

Tabelle 2.13.: Bewertungskriterien für ORM-Evaluation

Die Analyse ergab eine jeweils ähnliche Total-Bewertung bezüglich der Kriterien in 2.13. Trotzdem lässt sich in 2.14 den Grund für die Verwendung von Sequelize [Depb] auslesen.

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

Tabelle 2.14.: Bewertungsmatrix JavaScript ORMs

Wie erwähnt haben alle verglichenen ORMs eine ähnliche Gesamtbewertung. Bei "Sequelize" stechen jedoch die Produktreife und die Unterstützung für Relationen heraus. Diese zwei Gründe zusammen mit der Roadmap [Depa] von Sequelize haben schliesslich zur Überzeugung geführt, dass Sequelize die richtige Wahl ist.

Kapitel 3 **Anforderungsanalyse**

3.1. 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 3.1.: Funktionale Anforderungen

3.2. Nichtfunktionale Anforderungen

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

Tabelle 3.2.: Nichtfunktionale Anforderungen

3.3. Use Cases

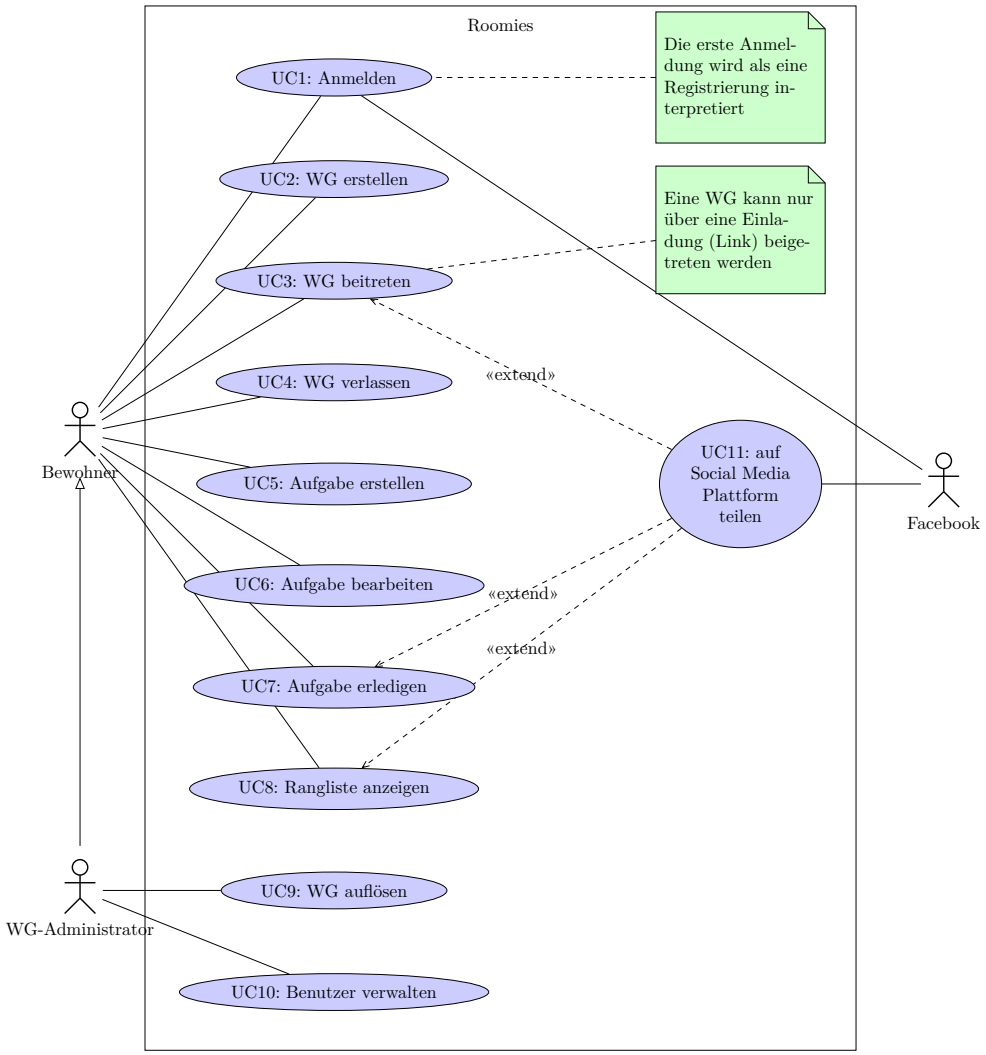


Abbildung 3.1.: Use Case Diagramm

3.3.1. Akteure

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

Tabelle 3.3.: Aktoren

3.3.2. UC1: Anmelden

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

Tabelle 3.4.: UC1: Anmelden

3.3.3. UC2: WG erstellen

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

Tabelle 3.5.: UC2: WG erstellen

3.3.4. UC3: WG beitreten

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

Tabelle 3.6.: UC3: WG beitreten

3.3.5. UC4: WG verlassen

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

Tabelle 3.7.: UC4: WG verlassen

3.3.6. UC5: Aufgabe erstellen

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

Tabelle 3.8.: UC5: Aufgabe erstellen

3.3.7. UC6: Aufgabe bearbeiten

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

Tabelle 3.9.: UC6: Aufgabe bearbeiten

3.3.8. UC7: Aufgabe erledigen

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

Tabelle 3.10.: UC7: Aufgabe erledigen

3.3.9. UC8: Rangliste anzeigen

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

Tabelle 3.11.: UC8: Rangliste anzeigen

3.3.10. UC9: WG auflösen

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

Tabelle 3.12.: UC9: WG auflösen

3.3.11. UC10: Benutzer verwalten

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

Tabelle 3.13.: UC10: Benutzer verwalten

3.3.12. UC11: auf Social Media Plattform teilen

Mapped Requirement	F11
Primary Actor	Bewohner
Story	Gewissen Interaktionen, wie UC3: WG beitreten, UC7: Aufgabe erledigen oder UC8: Rangliste anzeigen, sollen über die Social Media Plattform Facebook geteilt werden können. Wählt der Bewohner nach einer der aufgelisteten Use Cases die zu tun. So wird ein entsprechender Text erzeugt und die Möglichkeit geboten, diesen auf Facebook zu teilen.

Tabelle 3.14.: UC11: auf Social Media Plattform teilen

Kapitel 4 **Technische Architektur**

4.1. Einleitung

Wie in Kapitel 2.4, “Evaluation Frameworks in Node.js” beschrieben, wurde schlussendlich Express.js [Exp] für die Implementation der Beispielapplikation ausgewählt.

4.1.1. Code-Struktur

In Node.JS ist es unter den erfahreneren ([Hola], [Sch]) Entwicklern relativ üblich eine Applikation in mehr oder weniger komplett eigenständige Komponenten zu unterteilen. Dies sieht man u.a. auch an den installierbaren Komponenten auf NPM [Joya], in welchen es für sehr kleine Logiken eigene Komponenten hat.
Die Beispielapplikation “Roomies” folgt diesem Beispiel ebenfalls. Möglichst jede Komponente (z.B. auch ein Controller) ist abgetrennt von der Applikation und wird von dieser aufgerufen um eigene Initialisierungen zu machen.

4.2. Domainmodel

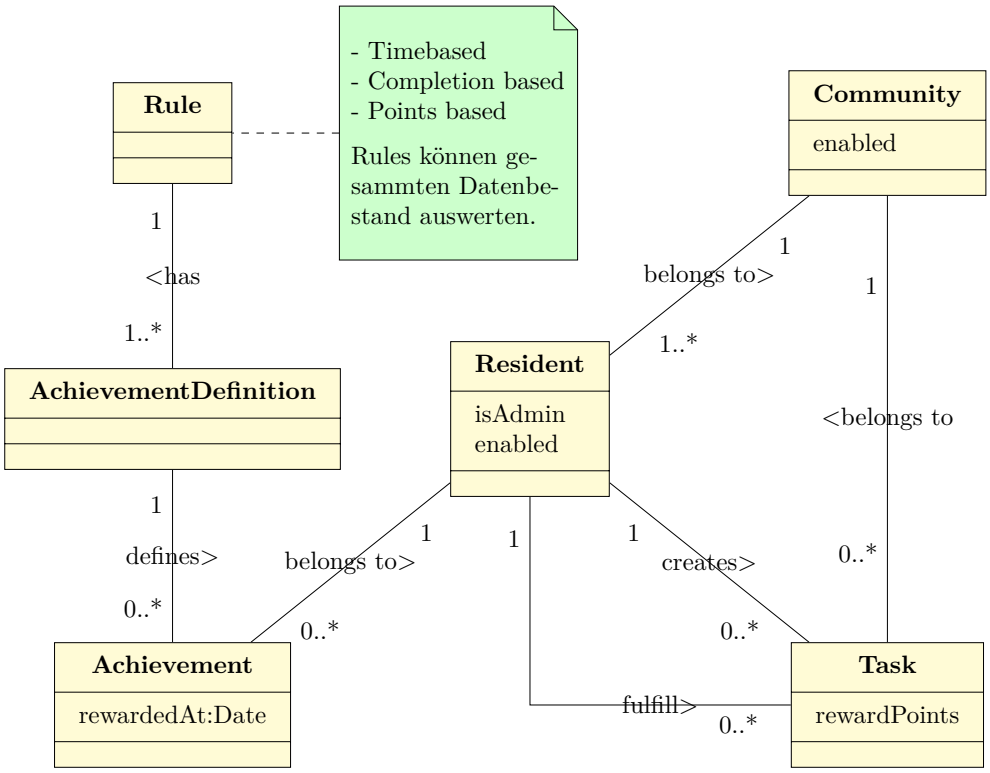


Abbildung 4.1.: Domainmodel

Achievement (Erfolg)

Achievements (Erfolge) werden an einen Resident vergeben, wenn Sie bestimmte Regeln (Rules) erreicht haben.

Rule (Regel)

Rules (Regeln) bestimmen, wann ein Resident Achievement erreicht hat und dieses bekommen. Diese Regeln können Zeitbasiert (z.B. ist kürzlich einer WG beigetreten), Punktebasiert (z.B. hat 10 Aufgaben erledigt) oder auch bei Komplettierung (z.B. hat sich registriert) vergeben werden.

AchievementDefinition (Erfolgsdefinition)

Damit die zu erreichenden Erfolge ersichtlich sind, müssen sie definiert werden. Hierfür wird AchievementDefinition (Erfolgsdefinition) benötigt. Die Erfolgsdefinition bestimmt wie ein Achievement auszusehen hat und mit welcher Regel es verknüpft ist.

Community (WG)

Community (WG) definiert die Wohngemeinschaft, in welcher sich die Bewohner befinden. Tasks werden immer einer WG zugeteilt.

Resident (Bewohner)

Resident (Bewohner) sind die Benutzer des Systems. Bewohner mit Administrotoren-Rechte (Eigenschaft “isAdmin”) können zusätzlich die Community (WG) administrieren.

Task (Aufgabe)

Tasks (Aufgaben) sind die eigentlichen “TODOs” der WG. Bewohner der WG können solche erstellen, bearbeiten und erledigen. Durch dass Erledigen einer Aufgabe gewinnt der Bewohner Punkte, welche sein Rang innerhalb der WG verbessert.

4.3. Entity-Relationship Diagramm

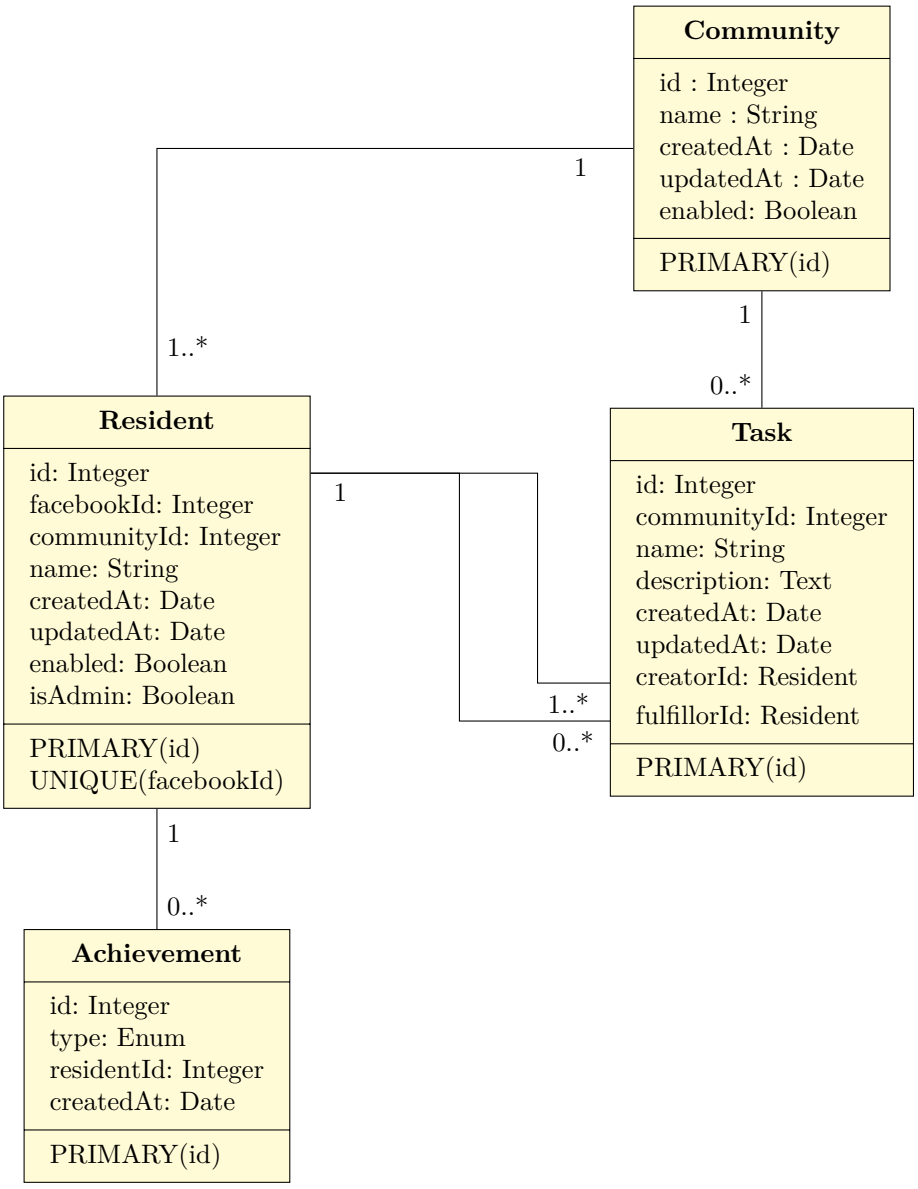


Abbildung 4.2.: Entity-Relationship Diagramm

4.4. Software Layers

Technologie-Unabhängig

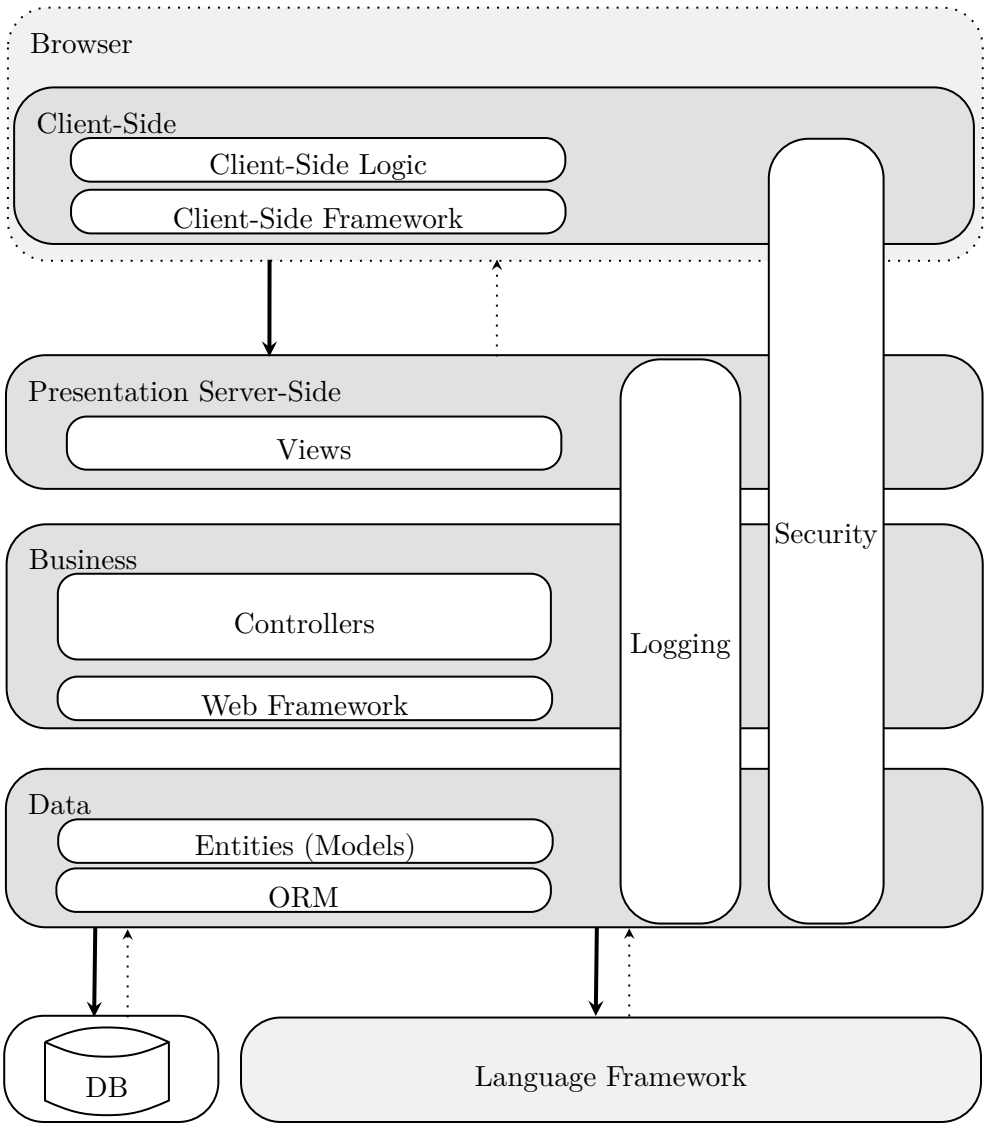


Abbildung 4.3.: Software Layers – Technologie-Unabhängig

Implementation

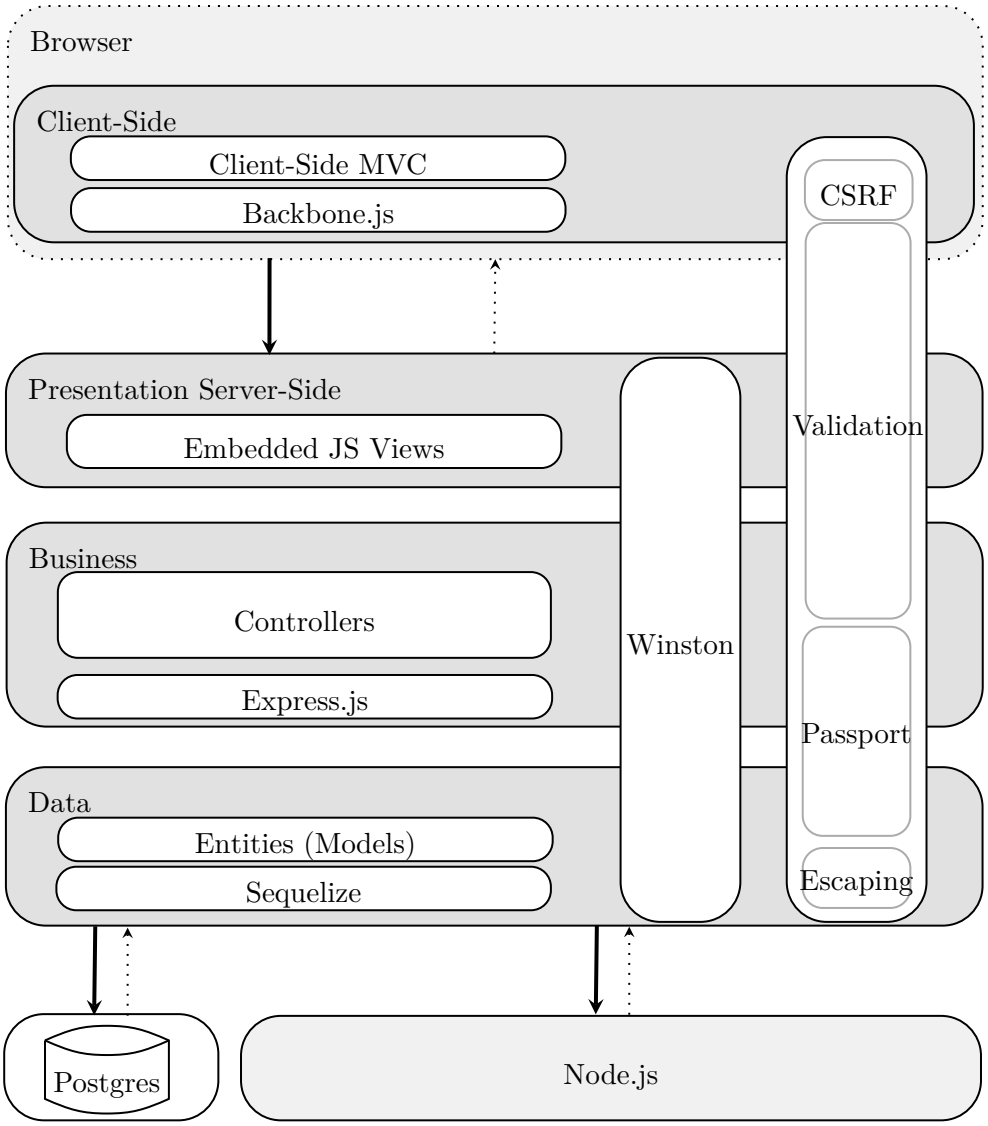


Abbildung 4.4.: Software Layers

Kapitel 5 Projektplanung

5.1. Phasenplanung

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

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

Tabelle 5.1.: Projektphasenbeschreibung

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

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

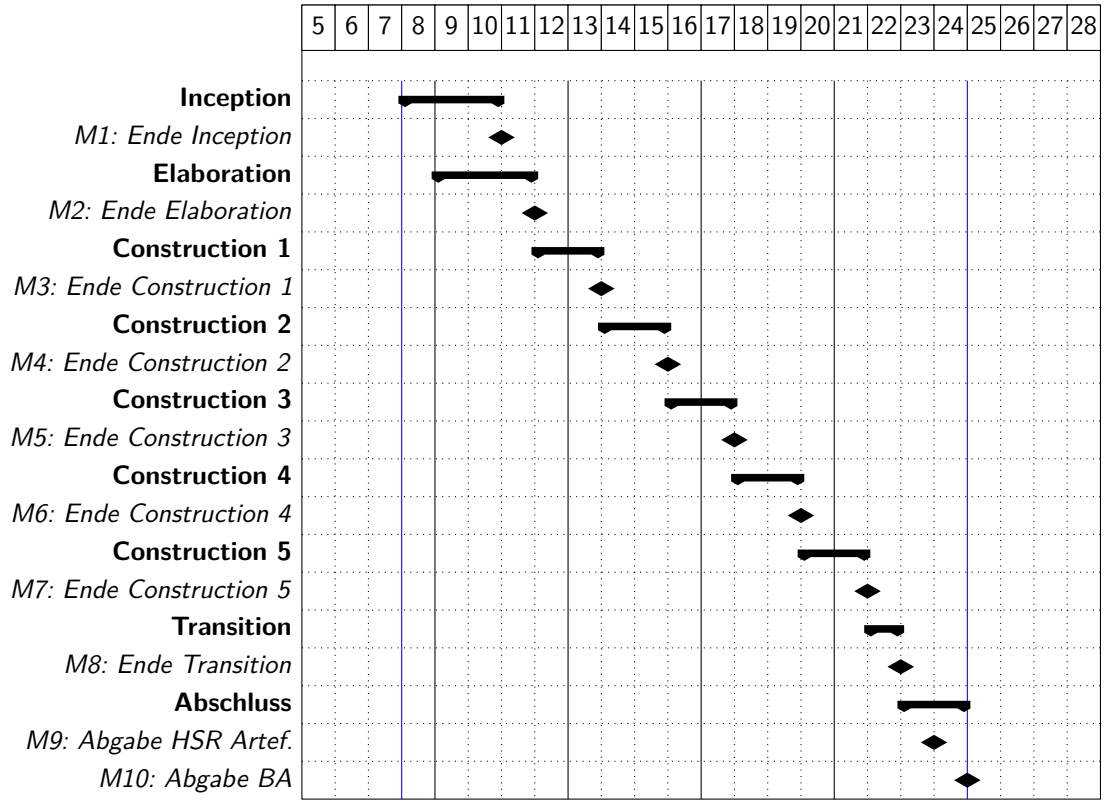


Abbildung 5.1.: Phasenübersicht mit Meilensteinen, Kalenderwochen Februar bis Juli 2013

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.

5.2. Meilensteine

ID	*Meilenstein*	*Termin*	*Beschreibung*
M1	Ende Inception	10.03.2013	Die Aufgabenstellung wurde gem. Auftrag klar definiert und die Projektinfrastruktur ist aufgesetzt. Eine initiale Projektplanung besteht.
M2	Ende Elaboration	17.03.2013	Konkrete Technologie und Guidelines sind definiert. Anforderungsdokumente sind erstellt und abgenommen. Initiale SAD und Architekturprototyp bereit.
M3	Ende Construction 1	31.03.2013	Das Fundament der Applikation wurde implementiert. Weiter wurden die ersten Use Cases der Priorität *Hoch* umgesetzt.
M4	Ende Construction 2	14.04.2013	Alle Use Cases der Priorität *Hoch* sind umgesetzt.
M5	Ende Construction 3	28.04.2013	
M6	Ende Construction 4	12.05.2013	
M7	Ende Construction 5	26.05.2013	
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 5.2.: Meilensteine

5.3. 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 & Phasenplanung
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	SAD	Beschreibung der angestrebten Architektur für die Beispielapplikation.
A26	M2	Architekturprototyp	Exemplarische Implementierung der angestrebten Technologie/Architektur *Typ: Quellcode/Applikation*
A80	M8	Quellcode Paket	Quellcode der Beispielapplikation zum eigenen, spezifischen Deployment. Bereits zur Weiterentwicklung. *Typ: Quellcode*
A81	M8	Vagrant Paket	VM-Image mit lauffähiger Version der Beispielapplikation. *Typ: Vagrant Image*
A82	M8	Heroku Paket	Beispielapplikation ist so vorbereitet, dass ein Deployment auf Heroku problemlos möglich ist. *Typ: Quellcode*
A83	M8	Installationsanleitung	Anleitung wie die verschiedenen Deployment-Pakete (Artefakte *A80-82*) eingesetzt/installiert werden können.
A100	M10	A0-Poster	Gem. HSR Vorgaben zu erstellendes Poster mit Übersucht zu dieser Bachelorarbeit.
A101	M10	Kurzfassung	Gem. HSR Vorgaben zu erstellende Kurzfassung dieser Bachelorarbeit.
A102	M10	Dokumentation	Alle bisherigen Dokumentationsartefakte zusammengefasst in einem Bericht. Wo nötig, sind entsprechende Kapitel dem Projektablauf entsprechend nachgeführt (bspw. *A25 SAD* etc.)

Tabelle 5.3.: Abzugebende Artefakte

5.4. Meetings

5.4.1. Regelmässiges Statusmeeting

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

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

5.5. 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.

5.5.1. 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.

5.5.2. 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.

5.5.3. Git Repositories

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

5.5.4. Continious Integration

Für das Projekt wird Travis CI zur Continious Integration Lösung verwendet. Nähere Informationen sind im Kaptiel 6 “Qualitätsmanagement” unter “Continuous Integration” zu finden.

Kapitel 6 **Qualitätsmanagement**

6.1. Coding Guideline

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

6.2. Test Guideline

Die Beispielapplikation wird nach der TDD-Methodik entwickelt.

6.2.1. Unit Testing

Mocha [Holb] 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 die “Should”-Formulierung verwendet:

```
1  var chai = require('chai')
2    ,config = require('../../config_test')
3    ,db = require('../../lib/db');
4
5    chai.should();
6
7    var sequelize = db(config);
8    describe('Community DAO', function(){
9      it('should not have any communities', function(done){
10       (function() {
11         sequelize.daoFactoryManager.getDAO('Community').all().success(
               function(communities) {
12           communities.should.have.length(0);
13           done();
14         }).error(function(error) {
15           throw(error);
16         })
17       }).should.not.throw();
18     });
19   });
```

Quellcode 6.1: Beispiel eines Unit Tests mit Mocha und Chai.js

6.2.2. Test Coverage Analysis

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

6.3. Continuous Integration

Mit Travis 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.

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.

6.4. Code Reviews

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

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

Anhang A **Abbildungen, Tabellen & Quellcodes**

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang B Literatur

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

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

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

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

[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).

[Dev] 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).

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

[Fie00] Roy Fielding. „Chapter 5, Representational State Transfer“. dissertation. University of California, Irvine, 2000. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

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

[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).

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

[Han] 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. *GNU Wget*. URL: http://curl.haxx.se/ (besucht am 13. 03. 2013).

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

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

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

[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).

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

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

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

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

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

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

[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).

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

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

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

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

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

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

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

[Wika] Wikipedia. *Bidirectional Streams Over HTTP*. URL: http://en.wikipedia.org/wiki/BOSH (besucht am 15. 03. 2013).

[Wikb] Wikipedia. *Responsive Webdesign*. URL: http://de.wikipedia.org/wiki/Responsive_Webdesign (besucht am 13. 03. 2013).

Anhang C Glossar

Benutzer

Ein Benutzer .... TOOODOOO. 31

Bewohner

Ein Bewohner .... TOOODOOO. 31

CI

Continuous Integration. 51, 59

CRUD

Abkürzung für “Create, Read, Update and Delete”; Die Zusammenfassung der Datenmanipulationsoperationen. 58

Gamification

Als Gamification oder Gamifizierung (seltener auch Spielifizierung) bezeichnet man die Anwendung spieltypischer Elemente und Prozesse in spielfremdem Kontext [Det+]. 10, 11, 44

Multiple Inheritance

Multiple Inheritance bezeichnet ein Konzept, welches es erlaubt, eine Klassen von mehr als einer Oberklasse erben zu lassen (vgl. Single Inheritance). 20

NoSQL

NoSQL Datenbanken ist eine relativ neue Art um Datenbanken ohne die SQL-Sprache zu entwickeln. Vielfach haben solche Datenbanken keine Relationen im Sinne der Relationalen Datenbanken implementiert.. 29

ORM

Ein “Object Relational Mapper” wird verwendet um Entitäten auf einer Relationalen Datenbank abzubilden und verwenden zu können. 23, 24, 26, 29

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 [Wika] sind solche Applikationen im Web realisierbar. 24, 26

REST

Representational State Transfer, definiert von Roy Fielding in seiner Dissertation [Fie00]. 26

RUP

Rational Unified Process; Iteratives Projektvorgehen [IBM]. 44

SAD

Software Architektur Dokument. 46, 47

Scaffolding

Scaffolding bezeichnet das toolunterstützte Generieren von Quellcodefragmenten. Beispiel: Erstellung einer Model-Klasse inkl. zugehörigen CRUD-Controller. 20, 21, 23, 26

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.. 50

VM

Virtual Machine. 47

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 [Dev]. 23, 26, 58

WG

Eine WG .... TOOODOOO. 10, 11, 31, 39, 40

Anhang D Projektrelevante URL's

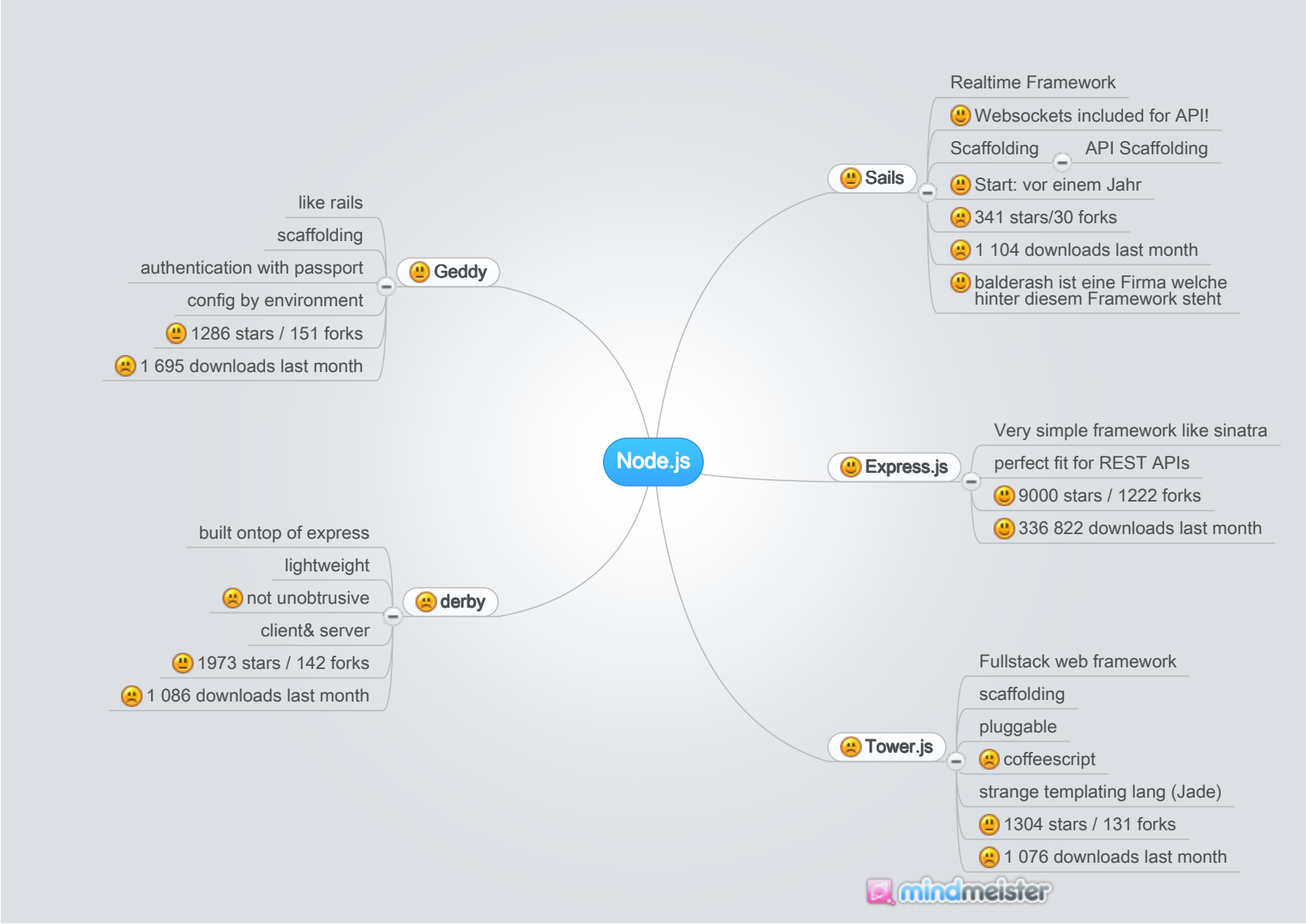
Ressource	*URL*
Projektverwaltung	`http://redmine.alabor.me/projects/ba2013`
Code: Git Repository	`https://github.com/mweibel/ba`
Code: CI	`https://travis-ci.org/mweibel/BA`
Thesis: Git Repository	`https://github.com/mweibel/BA-Dokumentation`
Thesis: PDF	`http://mweibel.github.com/BA-Dokumentation/thesis.pdf`
Thesis: CI	`https://travis-ci.org/mweibel/BA-Dokumentation`
Meeting Protokollierung	`https://github.com/mweibel/BA-Dokumentation/wiki/Meetings`

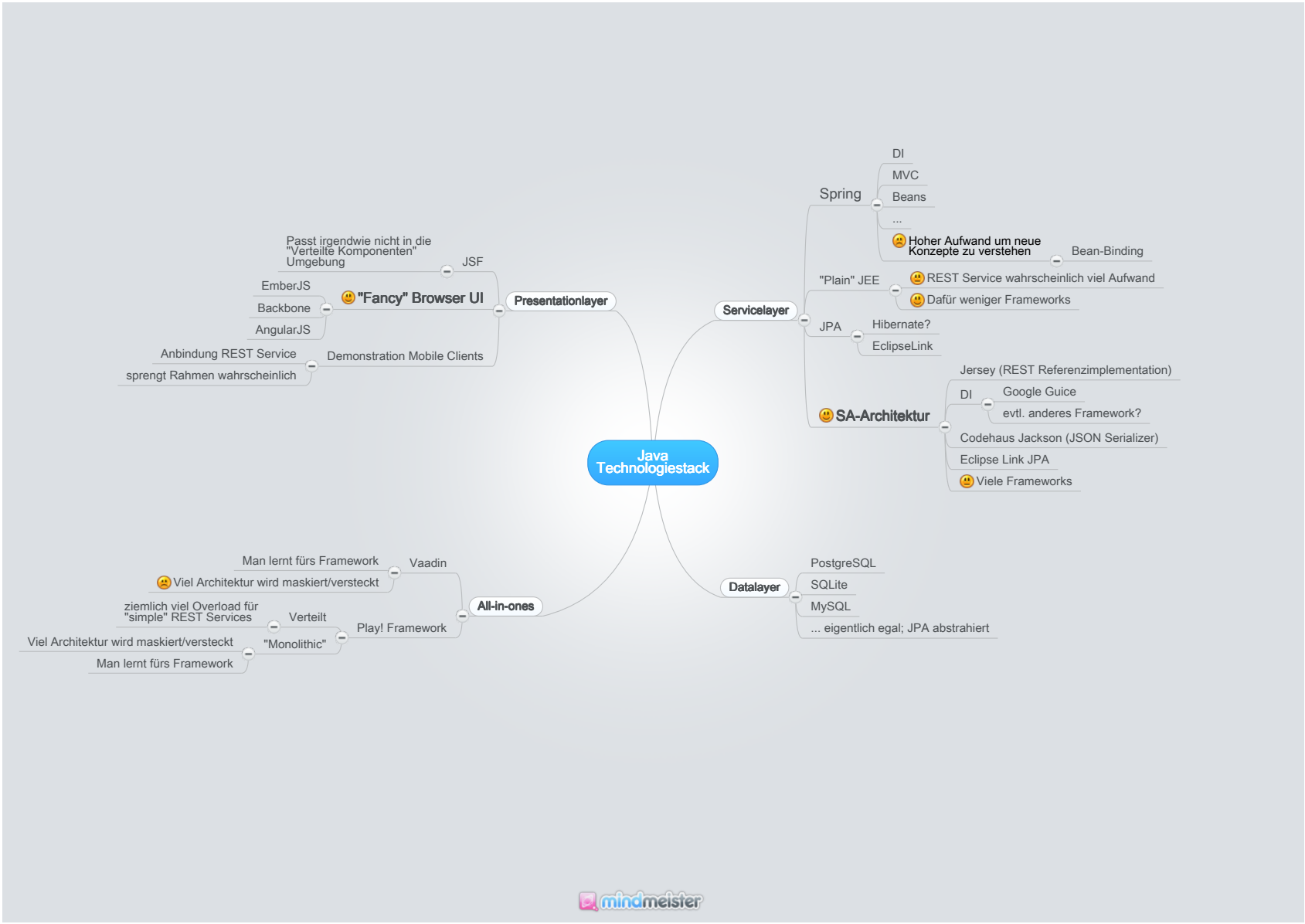
Tabelle D.1.: Projektrelevante URL's

Anhang E **Produktentwicklung**


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

Anhang F **Technologieevaluation**


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


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

Anhang G Coding Guideline

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

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

Original Repository: airbnb/javascript

Airbnb JavaScript Style Guide() {

Ein vernünftiger Ansatz für einen JavaScript-Style-Guide

Inhaltsverzeichnis

Datentypen

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

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

bar = 9;

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

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

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

bar[0] = 9;

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

[⬆]

Objekte

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

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

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

- Benutze keine reservierten Wörter für Attribute.

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

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

[⬆]

Arrays

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

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

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

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

```
var someStack = [];

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

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

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

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

// schlecht
```

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

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

[⬆]

Zeichenketten

- Benutze einfache Anführungszeichen `''` für Zeichenketten

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

// gut
var name = 'Bob Parr';

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

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

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

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

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

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

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

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

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

```
length = messages.length;

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

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

  return items + '</ul>';
}

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

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

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

[⬆]

Funktionen

- Funktionsausdrücke:

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

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

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

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

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

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

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

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

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

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

[⬆]

Eigenschaften

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

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

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

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

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

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

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

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

[⬆]

Variablen

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

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

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

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

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

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

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

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

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

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

- Weise den Wert einer Variable, wenn möglich, immer am Anfang des Gültigkeitsbereichs zu. Dies hilft Problemen mit der Variablendeklaration vorzubeugen.

```
// schlecht
function() {
  test();
  console.log('doing stuff..');

  //..other stuff..

  var name = getName();

  if (name === 'test') {
    return false;
  }

  return name;
}

// gut
function() {
  var name = getName();

  test();
  console.log('doing stuff..');

  //..other stuff..

  if (name === 'test') {
    return false;
  }

  return name;
}

// schlecht
function() {
  var name = getName();

  if (!arguments.length) {
    return false;
  }

  return true;
}

// gut
function() {
```

```
  if (!arguments.length) {
    return false;
  }

  var name = getName();

  return true;
}
```

[⬆]

Hoisting

- Variablendeklarationen werden vom Interpreter an den Beginn eines Gültigkeitbereichs genommen, genannt (`hoisting`). Wohingegen die Zuweisung an der ursprünglichen Stelle bleibt.

```
// Dies wird nicht funktionen (angenommen
// notDefined ist keine globale Variable)
function example() {
  console.log(notDefined); // => throws a ReferenceError
}

// Wird eine Variable nach seiner ersten
// Referenzierung deklariert, funktioniert
// dies dank des hoistings.
// Beachte aber, dass die Zuweisung von true
// erst nach der Referenzierung stattfindet.
function example() {
  console.log(declaredButNotAssigned); // => undefined
  var declaredButNotAssigned = true;
}

// Der Interpreter nimmt die Variablendeklaration
// an den Beginn des Gültigkeitbereichs.
// So kann das Beispiel wiefolgt umgeschrieben
// werden:
function example() {
  var declaredButNotAssigned;
  console.log(declaredButNotAssigned); // => undefined
  declaredButNotAssigned = true;
}
```

- Anonyme Funktionen `hoisten` ihren Variablennamen, aber nicht die Funktionszuweisung.

```
function example() {
  console.log(anonymous); // => undefined

  anonymous(); // => TypeError anonymous is not a function

  var anonymous = function() {
    console.log('anonymous function expression');
  };
}
```

- Benannte Funktionen `hoisten` ihren Variablennamen, aber nicht der Funktionsname oder Funktionskörper.

```
function example() {
  console.log(named); // => undefined

  named(); // => TypeError named is not a function

  superPower(); // => ReferenceError superPower is not defined

  var named = function superPower() {
    console.log('Flying');
  };
```

```
  // Das gleiche gilt, wenn der Funktionsname
  // derselbe ist, wie der Variablenname
  function example() {
    console.log(named); // => undefined

    named(); // => TypeError named is not a function

    var named = function named() {
      console.log('named');
    };
  }
}
```

- Funktionsdeklarationen `hoisten` ihren Namen und ihren Funktionskörper.

```
function example() {
  superPower(); // => Flying

  function superPower() {
    console.log('Flying');
  }
}
```

- Für weitere Informationen siehe hier: JavaScript Scoping & Hoisting by Ben Cherry

[⬆]

Bedingungen und Gleichheit

- Ziehe `===` und `!==` gegenüber `==` und `!=` vor.
- Bedingungsausdrücke werden immer gezwungen der `ToBoolean` Methode ausgewertet zu werden. Diese folgt den folgenden einfachen Grundregeln:
  - **Objekte** werden als **true** gewertet
  - **Undefined** wird als **false** gewertet
  - **Null** wird als **false** gewertet
  - **Booleans** werden als **der Wert des Booleans** gewertet
  - **Zahlen** werden als **false** gewertet sofern **+0, -0, or NaN**, ansonsten als **true**
  - **Zeichenketten** werden als **false** gewertet, sofern sie leer ist `''`, ansonsten als **true**

```
if ([0]) {
  // true
  // Arrays sind Objekte und Objekte werden als true ausgewertet
}
```

- Benutze `shortcuts`

```
// schlecht
if (name !== '') {
  // ...stuff...
}

// gut
if (name) {
  // ...stuff...
}

// schlecht
if (collection.length > 0) {
  // ...stuff...
}

// gut
if (collection.length) {
  // ...stuff...
}
```

- Für weitere Informationen siehe hier: Truth Equality and JavaScript by Angus Croll

[⬆]

Blöcke

- Benutze geschweifte Klammern für alle mehrzeiligen Blöcke.

```
// schlecht
if (test)
  return false;

// gut
if (test) return false;

// gut
if (test) {
  return false;
}

// schlecht
function() { return false; }

// gut
function() {
  return false;
}
```

[⬆]

Kommentare

- Benutze `/** ... */` für mehrzeilige Kommentare. Daran kann eine Beschreibung, eine Typendefinition und Werte für alle Parameter und den Rückgabewert angegeben werden.

```
// schlecht
// make() returns a new element
// based on the passed in tag name
//
// @param <String> tag
// @return <Element> element
function make(tag) {

  // ...stuff...

  return element;
}

// gut
/**
 * make() returns a new element
 * based on the passed in tag name
 *
 * @param <String> tag
 * @return <Element> element
 */
function make(tag) {

  // ...stuff...

  return element;
}
```

- Benutze `//` für einzeilige Kommentare. Platziere den Kommentar auf einer separaten Zeile oberhalb der beschriebenen Zeile. Vor den Kommentar kommt eine Leerzeile.

```
// schlecht
```

```
var active = true;  // is current tab

// gut
// is current tab
var active = true;

// schlecht
function getType() {
  console.log('fetching type...');
  // set the default type to 'no type'
  var type = this._type || 'no type';

  return type;
}

// gut
function getType() {
  console.log('fetching type...');

  // set the default type to 'no type'
  var type = this._type || 'no type';

  return type;
}
```

[⬆]

Whitespace

- Benutze Tabulatoren.

```
// schlecht
function() {
····var name;
}

// schlecht
function() {
·var name;
}

// gut
function() {
–var name;
}
```

- Platziere ein Leerzeichen vor einer öffnenden Klammer.

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

// gut
function test() {
  console.log('test');
}

// schlecht
dog.set('attr',{
  age: '1 year'
  , breed: 'Bernese Mountain Dog'
});

// gut
dog.set('attr', {
  age: '1 year'
  , breed: 'Bernese Mountain Dog'
```

```
});
```

- Platziere eine Leerzeile an das Ende der Datei.

```
// schlecht
(function(global) {
  // ...stuff...
})(this);
```

```
// gut
(function(global) {
  // ...stuff...
})(this);
```

- Rücke bei langen Methodenverkettungen ein.

```
// schlecht
$('#items').find('.selected').highlight().end().find('.open').updateCount();

// gut
$('#items')
  .find('.selected')
    .highlight()
    .end()
  .find('.open')
    .updateCount();

// schlecht
var leds = stage.selectAll('.led').data(data).enter().append("svg:svg").class('led', true)
    .attr('width',  (radius + margin) * 2).append("svg:g")
    .attr("transform", "translate(" + (radius + margin) + "," + (radius + margin) + ")")
    .call(tron.led);

// gut
var leds = stage.selectAll('.led')
    .data(data)
  .enter().append("svg:svg")
    .class('led', true)
    .attr('width',  (radius + margin) * 2)
  .append("svg:g")
    .attr("transform", "translate(" + (radius + margin) + "," + (radius + margin) + ")")
    .call(tron.led);
```

[⬆]

Führende Kommas

- **Ja.**

```
// schlecht
var once,
    upon,
    aTime;

// gut
var once
    , upon
    , aTime;

// schlecht
var hero = {
  firstName: 'Bob',
  lastName: 'Parr',
  heroName: 'Mr. Incredible',
  superPower: 'strength'
};
```

```
// gut
var hero = {
    firstName: 'Bob'
    , lastName: 'Parr'
    , heroName: 'Mr. Incredible'
    , superPower: 'strength'
};
```

[⬆]

Semikolons

- **Ja.**

```
// schlecht
(function() {
  var name = 'Skywalker'
  return name
})()

// gut
(function() {
  var name = 'Skywalker';
  return name;
})();
```

[⬆]

Typumwandlung

- Benutze `type coercion` am Anfang eines Statements.
- Bei Zeichenketten:

```
//  => this.reviewScore = 9;

// schlecht
var totalScore = this.reviewScore + '';

// gut
var totalScore = '' + this.reviewScore;

// schlecht
var totalScore = '' + this.reviewScore + ' total score';

// gut
var totalScore = this.reviewScore + ' total score';
```

- Benutze immer `parseInt` für Zahlen und gebe immer eine Basis für die Typumwandlung an.
- Wenn man aus Performancegründen kein `parseInt` verwenden will und ein `Bitshifting` benutzt, sollte man einen Kommentar hinterlassen, wieso dies gemacht wurde.

```
var inputValue = '4';

// schlecht
var val = new Number(inputValue);

// schlecht
var val = +inputValue;

// schlecht
var val = inputValue >> 0;

// schlecht
var val = parseInt(inputValue);
```

```
// gut
var val = Number(inputValue);

// gut
var val = parseInt(inputValue, 10);

// gut
/**
 * parseInt was the reason my code was slow.
 * Bitshifting the String to coerce it to a
 * Number made it a lot faster.
 */
var val = inputValue >> 0;
```

- Bei Booleans:

```
var age = 0;

// schlecht
var hasAge = new Boolean(age);

// gut
var hasAge = Boolean(age);

// gut
var hasAge = !!age;
```

[⬆]

Namenskonventionen

- Benutze keine `einzeichigen` Namen. Die Namen sollten beschreibend sein.

```
// schlecht
function q() {
  // ...stuff...
}

// gut
function query() {
  // ..stuff..
}
```

- Benutze `camelCase`, um Objekte, Funktionen und Instanzen zu benennen.

```
// schlecht
var OBJEcttssss = {};
var this_is_my_object = {};
var this-is-my-object = {};
function c() {};
var u = new user({
  name: 'Bob Parr'
});

// gut
var thisIsMyObject = {};
function thisIsMyFunction() {};
var user = new User({
  name: 'Bob Parr'
});
```

- Benutze `PascalCase`, um Klassen und Konstrukturen zu benennen.

```
// schlecht
function user(options) {
  this.name = options.name;
```

```
}

var bad = new user({
  name: 'nope'
});

// gut
function User(options) {
  this.name = options.name;
}

var good = new User({
  name: 'yup'
});
```

- Benutze führende Untenstriche `_`, um private Eigenschaften zu benennen.

```
// schlecht
this.__firstName__ = 'Panda';
this.firstName_ = 'Panda';

// gut
this._firstName = 'Panda';
```

- Um eine Referenz an `this` zuzuweisen, benutze `_this`.

```
// schlecht
function() {
  var self = this;
  return function() {
    console.log(self);
  };
}

// schlecht
function() {
  var that = this;
  return function() {
    console.log(that);
  };
}

// gut
function() {
  var _this = this;
  return function() {
    console.log(_this);
  };
}
```

- Gib deinen Funktionen einen Namen. Dies ist hilfreich für den `stack trace`.

```
// schlecht
var log = function(msg) {
  console.log(msg);
};

// gut
var log = function log(msg) {
  console.log(msg);
};
```

[⬆]

Zugriffsmethoden

- Zugriffsmethoden für Objekteigenschaften sind nicht von Nöten.

- Macht man dennoch Zugriffsmethoden, benutze `getVal()` und `setVal('hello')`.

```
// schlecht
dragon.age();

// gut
dragon.getAge();

// schlecht
dragon.age(25);

// gut
dragon.setAge(25);
```

- Wenn die Eigenschaft ein Boolean ist, benutze `isVal()` oder `hasVal()`.

```
// schlecht
if (!dragon.age()) {
  return false;
}

// gut
if (!dragon.hasAge()) {
  return false;
}
```

- Es ist in Ordnung `get()` - und `set()` -Methoden zu erstellen, aber sei konsistent.

```
function Jedi(options) {
  options || (options = {});
  var lightsaber = options.lightsaber || 'blue';
  this.set('lightsaber', lightsaber);
}

Jedi.prototype.set = function(key, val) {
  this[key] = val;
};

Jedi.prototype.get = function(key) {
  return this[key];
};
```

[⬆]

Konstruktoren

- Weise die Methoden dem `prototype` des Objektes zu, anstelle den `prototype` mit einem neuen Objekt zu überschreiben. Wenn man den `prototype` überschreibt wird eine Vererbung unmöglich, denn damit wird die Basis überschrieben!

```
function Jedi() {
  console.log('new jedi');
}

// schlecht
Jedi.prototype = {
  fight: function fight() {
    console.log('fighting');
  }

  , block: function block() {
    console.log('blocking');
  }
};

// gut
Jedi.prototype.fight = function fight() {
  console.log('fighting');
```

```
};

Jedi.prototype.block = function block() {
  console.log('blocking');
};
```

- Methoden können `this` zurückgeben, um eine Methodenverkettung zu ermöglichen.

```
// schlecht
Jedi.prototype.jump = function() {
  this.jumping = true;
  return true;
};

Jedi.prototype.setHeight = function(height) {
  this.height = height;
};

var luke = new Jedi();
luke.jump(); // => true
luke.setHeight(20) // => undefined

// gut
Jedi.prototype.jump = function() {
  this.jumping = true;
  return this;
};

Jedi.prototype.setHeight = function(height) {
  this.height = height;
  return this;
};

var luke = new Jedi();

luke.jump()
  .setHeight(20);
```

- Es ist in Ordnung eine eigene `toString()` -Methode zu schreiben, aber man sollte sicherstellen, dass diese korrekt funktioniert und keine Nebeneffekte hat.

```
function Jedi(options) {
  options || (options = {});
  this.name = options.name || 'no name';
}

Jedi.prototype.getName = function getName() {
  return this.name;
};

Jedi.prototype.toString = function toString() {
  return 'Jedi - ' + this.getName();
};
```

[⬆]

Module

- Ein Modul sollte mit einem `!` beginnen. Dies stellt sicher, dass wenn in einem Modul das abschliessende Semikolon vergessen wurde, keine Fehler entstehen, wenn die Scripte zusammengeschnitten werden.
- Eine Datei sollte in `camelCase` benannt sein, in einem Ordner mit dem selben Namen liegen und dem Namen entsprechen mit dem es exportiert wird.
- Benutze eine Methode `noConflict()` , welche das exportierte Modul auf die vorhergehende Version setzt und diese zurück gibt.
- Deklariere immer `'use strict';` am Anfang des Moduls.

```
// fancyInput/fancyInput.js
```

```
!function(global) {
  'use strict';

  var previousFancyInput = global.FancyInput;

  function FancyInput(options) {
    this.options = options || {};
  }

  FancyInput.noConflict = function noConflict() {
    global.FancyInput = previousFancyInput;
    return FancyInput;
  };

  global.FancyInput = FancyInput;
}(this);
```

[⬆]

jQuery

- Stelle allen jQuery-Objektvariablen ein `$` voran.

```
// schlecht
var sidebar = $('.sidebar');

// gut
var $sidebar = $('.sidebar');
```

- Speichere `jQuery lookups`, sofern sie mehrmals gebraucht werden.

```
// schlecht
function setSidebar() {
  $('.sidebar').hide();

  // ...stuff...

  $('.sidebar').css({
    'background-color': 'pink'
  });
}

// gut
function setSidebar() {
  var $sidebar = $('.sidebar');
  $sidebar.hide();

  // ...stuff...

  $sidebar.css({
    'background-color': 'pink'
  });
}
```

- Für DOM-Abfragen benutze `Cascading` : `$('.sidebar ul')` oder parent > child `$('.sidebar > .ul')` . jsPerf
- Benutze `find` mit `scoped jQuery object queries`

```
// schlecht
$('.sidebar', 'ul').hide();

// schlecht
$('.sidebar').find('ul').hide();

// gut
$('.sidebar ul').hide();

// gut
```

```
$('.sidebar > ul').hide();

// gut (langsamer)
$sidebar.find('ul');

// gut (schneller)
$($sidebar[0]).find('ul');
```

[⬆]

ECMAScript 5 Kompatibilität

- Verweis auf Kangax's ES5 Kompatibilitätstabelle

[⬆]

Testing

- **Ja.**

```
function() {
  return true;
}
```

[⬆]

Performance

- On Layout & Web Performance
- String vs Array Concat
- Try/Catch Cost In a Loop
- Bang Function
- jQuery Find vs Context, Selector
- innerHTML vs textContent for script text
- Long String Concatenation
- Loading...

[⬆]

Ressourcen

Lese dieses

- Annotated ECMAScript 5.1

Andere Styleguides

- Google JavaScript Style Guide
- jQuery Core Style Guidelines
- Principles of Writing Consistent, Idiomatic JavaScript

Andere Styles

- Naming this in nested functions - Christian Johansen
- Conditional Callbacks

Bücher

- JavaScript: The Good Parts - Douglas Crockford
- JavaScript Patterns - Stoyan Stefanov
- Pro JavaScript Design Patterns - Ross Harmes and Dustin Diaz
- High Performance Web Sites: Essential Knowledge for Front-End Engineers - Steve Souders
- Maintainable JavaScript - Nicholas C. Zakas
- JavaScript Web Applications - Alex MacCaw

- Pro JavaScript Techniques - John Resig
- Smashing Node.js: JavaScript Everywhere - Guillermo Rauch

Blogs

- DailyJS
- JavaScript Weekly
- JavaScript, JavaScript...
- Bocoup Weblog
- Adequately Good
- NCZOnline
- Perfection Kills
- Ben Alman
- Dmitry Baranovskiy
- Dustin Diaz
- nettuts

[⬆]

In the Wild

Dies ist eine Liste von Organisationen, welche diesen Style Guide benutzen. Sende uns einen `Pull request` oder öffne einen `issue` und wir werden dich der Liste hinzufügen.

- **Airbnb**: airbnb/javascript
- **American Insitutes for Research**: AIRAST/javascript
- **ExactTarget**: ExactTarget/javascript
- **GoCardless**: gocardless/javascript
- **GoodData**: gooddata/gdc-js-style
- **How About We**: howaboutwe/javascript
- **MinnPost**: MinnPost/javascript
- **National Geographic**: natgeo/javascript
- **Razorfish**: razorfish/javascript-style-guide
- **Shutterfly**: shutterfly/javascript

[⬆]

Übersetzungen

Dieser Styleguide ist in den folgenden Sprachen erhältlich:

- :en: **Englisch**: airbnb/javascript
- **Japanisch**: mitsuruog/javacript-style-guide

[⬆]

The JavaScript Style Guide Guide

- Reference

[⬆]

Contributors

- View Contributors

[⬆]

Lizenz

[⬆]

```
};
```

Anhang H 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 I **Meetingprotokolle**

Bachelorarbeit Vorbesprechung 14. Februar 2013

Teilnehmer

- Hans Rudin, HRU (HSR)
- Kevin Gaunt, KGA (HSR)
- Daniel Hiltebrand, DHI (Crealogix)
- Manuel Alabor, MAL (Team)
- Alexandre Joly, AJO (Team)
- Michael Weibel, MWE (Team, Protokoll)

Traktanden

- StoryboardBuilder - Was ist der aktuelle Stand?
- Oder was habt ihr für Ideen?

Meeting

StoryboardBuilder

Einführung DHI

- Ende Januar Entscheid: Nicht weiterentwickeln
- Allerdings nicht weil Produkt/Markt nicht interessant wäre
- Ziel war: damit die Crealogix UX-Services zu unterstützen
- Crealogix wird keine UX-Services gegen aussen mehr anbieten
    - mehr interne Projekte betreuen

- Zwei Schwerpunkte: Education & Financial Services

    - StoryboardBuilder gehört nicht in einen solchen Schwerpunkt

- Dies obwohl gutes Potential gesehen wird für das Produkt
- Anforderungsspezifikation verfeinert bis Ende Januar
- Technischer Prototyp gestartet, aber wieder gestoppt aufgrund der Neuorientierung
- Idee wäre: Storyboardbuilder an externe Firma weitergeben
- Bestehende Mitbewerber bauen ihre Angebote aus

Diskussion BA

- Frage an die Runde: ist es interessant für euch, den Storyboardbuilder in der BA weiter zuentwickeln?
    - MAL: Wie würde das aussehen?
        - DHI: Beispiel aufgrund gewählter Technologie zu entwickeln
        - DHI: Auf Basis der fachlichen Spezifikation der Crealogix
        - HRU: BA sollte nicht zu einer Fleissarbeit werden
        - HRU: Was wären denn die Herausforderungen wenn das weiterentwickelt werden würde in BA?
        - HRU: Die momentane Ausgangslage ist anders, da kein wirklicher Kunde existiert
        - DHI: Es sind sicher einige Ideen da, die technisch Herausfordernd sind
        - DHI: Grafische Repräsentation auf Screen, Objektmodell umsetzen
        - DHI: Wie gesagt, hat auch nichts dagegen, wenn eine neue Arbeit gemacht werden würde
        - DHI: Laut Kenntnisstand von DHI sollte auch von seiten Industriepartner weniger Betreuung beinhalten, mehr von Team
- KGA: Was ist nun der Anspruch an das Meeting? Müssen wir am Ende des Meetings schon wissen was gemacht werden soll?
    - DHI: Ist offen, hat keinen Anspruch auf Entscheid jetzt - sollte aber bald geschehen, da Bachelorarbeit bald startet
    - DHI:

- DHI: Hat div. Alternativen die man anschauen könnte
- MAL: für ihn ist das Durchführen der BA mit dem Storyboardbuilder nicht mehr besonders interessant, aufgrund Änderung seitens Crealogix
    - DHI: Ja das stimmt, BA würde nicht mehr zu einem realen Produkt führen

- MWE: gehts ähnlich wie MAL
- DHI: Thema 1:
    - **Kino reservations system**
    - Kennt Kinobesitzer in rapperswil
    - Buchungssystem
    - Mit anbindung an Kassensystem
    - nicht nur einzelne Plätze
    - sondern auch Firmenanlässe, Frauenkino, Catering etc.
    - Konzeptionell entwerfen
    - soweit wie möglich implementieren
    - könnte sehr interessant sein, DHI's Meinung nach

- DHI: Thema 2
    - **Personal Finance Management**
    - Zusatz zu Ebanking
    - Eigene Zahlungen analysieren
    - Verschiedene Banken
    - Soviel für Versicherung, Einkauf, etc.
    - Basiert auf einer isländisch-schwedischen Firma
    - Integrationsarbeit (in ein Bankensystem einbauen)
    - MAL: Geht es darum, das zu integrieren und anschaulich darzustellen, und Data-Mining ist schon gemacht?
    - DHI: Ja
        - Schwierigkeit Sicherheit
        - Zertifizierung, Authentifizierung
        - nur Teilaspekt möglich zum lösen
        - Versch. Komponenten
        - Aspekt wichtig auf welcher sich die BA konzentrieren soll

    - DHI: Müsste das genauer anschauen wie das machbar wäre
        - Da es ein sehr grosses System wäre
        - Müsste verifizieren ob das gehen soll?

- HRU: Thema 2 wohl eher interessant (Team bejaht)
  - HRU: Wäre das so kurzfristig machbar?
    - DHI: müsste angeschaut werden
    - DHI: Verträge bestehen
    - DHI: anderes ist Ebanking in Java mit Oracle
      - DHI: Verfügbar machen möglich
  - HRU: Zwei Komponenten, was ist Webfrontend?
    - DHI: Netty server von airlock für authentifizierung
    - DHI: möglichst HTML das übers web geht
    - DHI: J2EE - JSP vorne
  - HRU: Isländische Software
    - DHI: .NET basiert
    - DHI: hat aber ein WCF/REST/AJAX schnittstelle welche relativ gut ins Frontend integrierbar wäre
    - DHI: von eienm System ins andere System transferieren (Oracle zu MSSQL DB)
    - DHI: Statistisch aufwerten, und wieder anzeigen
    - DHI: machbarkeit unklar, muss verifiziert werden
- MAL: Was hat HRU für Projekte
  - MAL: Realtime sachen wären interessant (Mobile, Chat, Messaging?)
  - HRU: hat keine Projekte
- DHI: Hat evtl. noch andere Projekte, müsste das aber noch anschauen
- MAL: Bis Testumgebung steht würden wohl wochen vergehen
  - DHI: stimmt wohl
- MWE: Persönlich intressiert vorallem WebRTC
- HRU: was ist mit web realtimecommunication (WebRTC) gemeint?
- MWE: Near-realtime communication (Daten, Video, Audio) zw. Browsern
- MWE: Was wäre denn für Crealogix interessant - zentrale Frage?
  - DHI: Crealogix muss nicht unbedingt dabei sein, wenn nicht nötig
- DHI: update maus-scanner
  - MAL: wäre denn mobile auch ein Thema?
  - DHI: Mobile ist sehr zentral

- HRU: gibt es fraktionen (web/mobile) im Team?
  - MWE: Web-Mensch, aber Mobile wäre auch sehr interessant

- DHI: Fragt bei Crealogix CEO/entwicklungsleiter nach bzgl. Mobile
- MAL: Elearning wäre auch interessant bzlg. Mobile
  - DHI: könnte nachfragen obs da auch was geben würde?

- AJO: Auch vorallem Mobile interessant
  - arbeitet auch vorallem in Mobile
  - HRU: Auch sie MAL ;)

- DHI: müsste nachfragen, aber wäre sicher interessant
- MAL: Wäre sicher interessant auf DHI's Themen zu warten, andererseits müsste auch Teamintern bzw. mit KGA/HRU geschaut werden
- DHI: wann beginnt die Arbeit? - Montag
- DHI: 3 Bereiche Education
  - Campusmanagement
  - Time to learn
    - 30'000 die mit TTL arbeiten
    - Mit Center for young professional zusammenarbeit (evtl. da was interssantes)
    - In Bubikon

- DHI: was machen wenn nichts herauskommt?
  - DHI: würden gerne mit euch zusammenarbeiten, wenn möglich

- HRU: Wäre schon der Weg zum gehen
  - parallel müssten überlegungen angestellt werden obs was anderes geben würde
  - gibt den 3 Studierenden möglichst freie Hand

- KGA: Termin bis wann die Entscheidung möglich sein
  - HRU: allerspätistens erste Woche
  - DHI: wird noch heute mit den 3 Crealogix leuten schauen
  - DHI: Bis morgen, 15.02. Antwort wenn möglich
  - MAL: Mittwoch zu spät oder zu früh?
  - HRU: Wann sind sie @HSR?
    - MAL: MO/DI/MI

    - HRU: Mittwoch wäre nicht unbedingt zu spät
    - MAL: Mittwoch wäre Zusammenkunft, um definitiv zu entscheiden. Aber mit Kommunikation bis dann
    - MAL: DHI kann sicher schnell entscheiden, je nach dem wäre pers. Anwesenheit nicht nötig
    - DHI: würde gerne persönlich dabei sein

- DHI: gibt morgen Feedback
- HRU: das ist gut - in der Runde Team/HSR diskutieren was gemacht werden kann
- KGA: Was wird mit UX passieren? (Expert Talks)
    - DHI: Prio 1 hat interne Aufträge und Prio 2 externe

Diskussion im Team

- KGA: also ist keines der beiden Thema sehr interessant für Team?
- Team: Ja, insbesondere auch zu gross für die kurze Zeit (Thema 2)
- HRU: Reservationssystem vorallem Businessanalyse
- HRU: Sicher gut zu schauen was DHI einbringt
- HRU: Aber auch schauen was wir für Ideen haben
- KGA: was wäre ursprüngliche Idee für SA gewesen
- MWE: XMPP Server in node.js modular, flexibel bzgl. Datenbankanbindung
- MAL: und auch Skalierbarkeit von node.js interessant
- MAL: interessant wäre vielleicht frontend framework
- HRU: Alle miteinander auf dem Laufenden halten bzgl. Ideen

Nächstes Meeting

- Mittwoch, 20. Februar 2013, 10:10 Uhr