Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Einstiegserleichterung für die Weiterentwicklung und Erweiterung der JavaScript- und HTML-GUI von Saros

Nina Weber
Matrikelnummer: 4781766
nina.weber@fu-berlin.de

Eingereicht bei: Prof. Lutz Prechelt
Betreuer: Franz Zieris

Berlin, 08. Juli 2016

Nina Weber

Zusammenfassung

Durch Bereitstellung des Saros-Plug-Ins, nicht nur für *eclipse* sondern auch für *IntelliJ IDEA*, wurde zur Vereinheitlichung der Benutzeroberfläche ein Web-Browser eingeführt, über den eine Webseite dargestellt werden kann. Die Implementierung dieser wurde von Bastian Sieker begonnen und soll die alte GUI in Zukunft vollständig ersetzen.

Das Ziel dieser Arbeit war anfänglich die nutzerorientierte Weiterentwicklung dieser *HTML-GUI*. Im Laufe meiner Tätigkeit veränderte sich der Fokus jedoch dahingehend, dass der Einstieg in die Entwicklung an der *JavaScript*- und *HTML*-basierten Oberfläche für nachfolgende Entwickler leichter gemacht wird. Dies geschah durch diverse Änderungen, die insgesamt zum Ziel hatten die Einarbeitung und den allgemeinen Workflow zu verbessern.

In diesem Zuge wurde die bestehende Ordnerstruktur neu gestaltet, sowie eine Lösung für den, durch das Build-Skript gestörten, Arbeitsfluss implementiert. Des Weiteren wurden mögliche Alternativen zu dem genutzten Template Engine evaluiert und zum Einsatz vorgeschlagen. Zusätzlich wurde die ursprüngliche Zielsetzung der Erweiterung verfolgt und auch ansatzweise umgesetzt, indem ein bislang fehlendes Feature umgesetzt, sowie der Grundstein für die Entwicklung eines weiteren gelegt wurde.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

08. Juli 2016

Nina Weber

Inhaltsverzeichnis

1 Einleitung

1.1 Paarprogrammierung

Die Paarprogrammierung ist eine von zahlreichen Praktiken des **Extreme Programming**. Bei dieser Methode arbeiten zwei Programmierer gleichzeitig vor einem Computer am selben Artefakt, zumeist Programmcode. Dabei implementiert einer der beiden das aktuelle Problem, während der andere ihm dabei zusieht und dafür sorgt, dass sich die Umsetzung „in den Kontext der umliegenden Anwendung“ einbetten lässt (s. [Mül08], S. 18). Neben einigen Vorteilen, die diese Methode bietet, hat sie allerdings auch den Schwachpunkt, dass sich beide Softwareentwickler stets am selben Ort befinden müssen, um direkt über den entstehenden Quellcode reden zu können.

> **Extreme Programming**
> Das *Extreme Programming*, kurz *XP*, ist eine von mehreren *agilen Methoden* der Softwareentwicklung. Neben sehr planungsintensiven Formen, wie dem *Wasserfallmodell*, in dem möglichst von Anfang an strikt feststeht wann welcher Schritt im Projekt getan werden muss, sind die Entwickler bei einer agilen Arbeitsweise weitaus flexibler. Ziele werden bei dieser eher kurz- oder mittelfristig geplant und geben die Möglichkeit etwas aufgrund von unabsehbaren Ereignissen zu ändern.
>
> Innerhalb des Extreme Programming gibt es verschiedene Praktiken, welche die Befürworter dieses Modells, an ihre persönlichen Bedürfnisse angepasst, befolgen. (vgl. [Wel99])

Die sogenannte *verteilte Paarprogrammierung*[1] ist eine Erweiterung der herkömmlichen Paarprogrammierung, welche die räumliche Einschränkung dieser aus dem Weg schafft. Bei dieser Arbeitsweise wird das gemeinsame Arbeiten auf zwei verschiedene Computer verteilt, was den Vorteil bietet, dass sich die Entwickler an beliebigen Orten befinden können. Um sich dennoch in Echtzeit über das entstehende Produkt austauschen zu können, benötigt es entsprechende technische Hilfsmittel, die den entstehenden Programmcode live an den Arbeitspartner übertragen können.

[1] auch *DPP*, von englisch: „Distributed Pair Programming“

1.2 Verteilte Paarprogrammierung mit Saros

Eine solche Softwarelösung entstand über die letzten Jahre in der Arbeitsgruppe „Software Engineering“ an der Freien Universität Berlin unter dem Namen Saros[2]. Dieses Programm ermöglicht es den Nutzern, in Form eines **Plug-ins** für die **IDE** *eclipse*[3] zu zweit oder in einer kleinen Gruppe mit Echtzeit-Synchronisierung an dem selben Artefakt zu arbeiten.

> **Plug-in**
> Ein *Plug-in* ist ein kleines Zusatzprogramm, welches in ein anderes, bereits bestehendes, Programm eingebunden werden und so dessen Funktionalitäten erweitern kann.

> **IDE**
> Eine *Integrated Development Environment*, kurz *IDE*, zu Deutsch etwa *integrierte Entwicklungsumgebung*, ist eine Anwendung, die einen Softwareentwickler bei seinen Programmiertätigkeiten unterstützt, indem sie häufig benötigte Arbeitsschritte, wie das Refaktoriseren oder Kompilieren, automatisiert beziehungsweise durch entsprechende grafische Elemente für den Programmierer leichter zugänglich macht und so dessen Arbeitsweise effizienter gestalten kann.

Um Saros nutzen zu können, wird ein **XMPP**-Account benötigt. Loggt man sich mit diesem ein, so ist es möglich weitere XMPP-Accounts als Kontakte hinzuzufügen. Möchte man nun mit einem oder mehreren von seinen gespeicherten Kontakten in Verbindung treten, um nach der Methode der verteilten Paarprogrammierung zu arbeiten, so kann man eine so genannte *Session*[4] starten. Nehmen die ausgewählten Kontakte die, in diesem Zuge gesendete, Sitzungseinladung an, so werden alle gewünschten Projekte, beziehungsweise Dateien, mit ihnen geteilt.

> **XMPP**
> Das *Extensible Messaging and Presence Protocol*, kurz *XMPP*[5], ist ein Protokoll, das für Instant Messaging Dienste (Chats) und Präsenzanzeigen (On- bzw. Offline) genutzt werden kann. Im Gegensatz zu vielen anderen Messaging-Diensten ist XMPP ein offenes System und kann somit Plattform-unabhängig und unter vielen verschiedenen Client-Programmen eingesetzt werden.

[2] http://www.saros-project.org/
[3] https://eclipse.org/
[4] zu Deutsch „Sitzung“

Wurden die entsprechenden Dateien erfolgreich geteilt, so können von diesem Punkt an alle beteiligten Saros-Nutzer gleichzeitig an den geteilten Dateien arbeiten. Zudem besteht die Möglichkeit seine Gedanken über einen integrierten Chat miteinander zu teilen, wodurch das lokale Paarprogrammieren bereits gut durch Saros imitiert werden kann.

1.3 Einführung eines Web-Browsers

Die geplante Bereitstellung des Saros-Plug-ins für weitere IDEs erfordert Rücksichtnahme auf die technischen Unterschiede zwischen diesen. Besonders im Fokus steht dabei die Entwicklungsumgebung *IntelliJ IDEA*[6]. Diese benutzt im Gegensatz zu *eclipse* nicht die Programmbibliothek *SWT*[7] für die Erstellung grafischer Oberflächen mit Java, sondern primär *Swing* und *AWT*[8]. Die Folge daraus ist, dass die gesamte Benutzeroberfläche in zweifacher Form vorliegen müsste, was doppelten Wartungsaufwand erfordern würde.

Diese Redundanz verstößt allerdings gegen das sogenannte **DRY-Prinzip**. Um ihm dennoch Folge leisten zu können, untersuchte Christin Cikryt den Nutzen eines Web Browsers der sowohl in *eclipse* als auch in *IntelliJ IDEA* gut integrierbar ist [Cik15]. Das Ergebnis war ein *SWT-Browser*, über den alle gewünschten Funktionalitäten in Form einer Website, unter Einsatz von **HTML** und **JavaScript**, realisiert und dargestellt werden können – und somit die Pflege und Wartung auf eine produktive Benutzeroberfläche reduziert.

> **DRY-Prinzip**
> D.R.Y. (bzw. DRY) steht für *Don't repeat yourself*, zu Deutsch etwa *Wiederhole dich nicht*. Es ist eines der Grundprinzipen in der angewandten Informatik und sollte nach Möglichkeit stets befolgt werden. Das Prinzip besagt, dass Informationen, wie zum Beispiel Codefragmente im Quelltext, nicht mehrfach auftauchen sollten.
>
> Wird es nicht eingehalten, so taucht „dieselbe Sache" an zwei oder mehr Stellen auf. Möchte man nun daran etwas ändern, so muss unbedingt dafür Sorge getragen werden, dass dieselbe Änderung auch an den anderen Stellen durchgeführt wird. In der Praxis ist dies nicht nur unbequem, sondern sorgt auch da-

[5] https://xmpp.org/
[6] https://www.jetbrains.com/idea/
[7] Standart Widget Toolkit
[8] Abstract Window Toolkit

für, dass früher oder später die ursprünglich gleichen „Sachen“ ungewollt von einander abweichen. (vgl. [HT99])

HTML
Die *HyperText Markup Language*, kurz HTML, ist eine so genannte Auszeichnungssprache, die für die Gliederung von Daten verwendet wird. Insbesondere für die Erstellung von Webinhalten wird sie häufig als rein statisches Grundgerüst verwendet.

JavaScript
JavaScript ist eine Skriptsprache, die vor allem für die Webentwicklung verwendet wird. Sie kann dazu verwendet werden, HTML-Seiten um dynamische Inhalte oder Funktionen zu erweitern.

1.4 Entwicklung der JavaScript und HTML-basierten GUI

Die Entwicklung grafischen Benutzeroberfläche, kurz *GUI*[9], die in dem von Cikryt vorgestellten *SWT-Browser* dargestellt werden sollte, im Folgenden *HTML-GUI* genannt, wurde zeitgleich mit der Integration des Browsers, sowie der Implementierung der benötigten **Schnittstelle** (zwischen dem Java-Backend und dem Web-Frontend) durch Matthias Bohnstedt [Boh15] durchgeführt. Die nötige Entwicklungsarbeit wurde von Bastian Sieker im Rahmen seiner Masterarbeit [Sie15] begonnen, konnte allerdings in der gegebenen Zeit nicht vollständig beendet werden.

Schnittstelle und API
„Der Begriff *Schnittstelle* - oder englisch *Interface* - bezeichnet grundsätzlich den Punkt einer Begegnung oder einer Koppelung zwischen zwei oder mehr Systemen und/oder deren Grenzen zueinander.“ ([Hal94], S. 168)

Eine besondere Schnittstelle in der Informatik ist dabei die *Softwareschnittstelle*, im Englischen *Application-Programming-Interface*, kurz *API*, die dazu benutzt wird, Daten zwischen zwei Softwaresystemen auszutauschen. (vgl.[Hal94], S. 169)

[9] Abkürzung GUI, von englisch: *graphical user interface*

1.5 Zielsetzung

Aufgrund der nicht fertiggestellten HTML-GUI, setzte ich mir die nutzerorientierte Weiterentwicklung dieser als Arbeitsaufgabe für diese Bachelorarbeit.

Dieses Ziel änderte sich jedoch während der Einarbeitungsphase. In dieser entschied ich, dass eine Erleichterung des Entwicklungseinstiegs für nachfolgende Programmierer sinnvoller sei, als eine reine Weiterentwicklung der *HTML-GUI*. Zunächst war in diesem Zuge eine vollständige Neustrukturierung der bestehenden Anwendung geplant. Im Ergebnis sollte die Anwendung wieder mindestens den gleichen Funktionsumfang aufweisen.

Nach einer umfangreichen Bewertung verschiedener Alternativen zu der bestehenden Technologie, musste ich meine vorige Entscheidung allerdings revidieren. Anstatt alles neu zu strukturieren, überarbeitete ich meine Zielsetzung erneut und entschied mich dafür, die bestehende Web-Anwendung nicht vollständig zu ersetzen, sondern nur Verbesserungen in den erkannten Problemfeldern einzuführen. Zudem wurde die anfänglich geplante nutzerorientierte Erweiterung wieder - in reduzierter Form - in den Aufgabenkatalog aufgenommen.

1.6 Gliederung dieser Arbeit

Diese Arbeit beschäftigt sich grundlegend mit den folgenden Themen:

In Abschnitt 2.1 auf der nächsten Seite werden die Mängel der bestehenden HTML-GUI analysiert. Im Anschluss werden mögliche Alternativen für das bisher genutzte Framework aufgezeigt und bewertet (siehe Abschnitt 2.2 auf Seite 10). Die anfänglich aufgezeigten Defizite werden in Abschnitt 3.1 auf Seite 19 teilweise behoben. Abschließend werden die Funktionalitäten der *HTML-GUI* in Abschnitt 3.2 auf Seite 26 erweitert.

2 Meine Odyssee

Das Ziel dieser Arbeit war anfangs eine Erweiterung der bestehenden *HTML-GUI* durch zusätzliche Funktionalitäten. Dabei sollte auf die bereits vorangegangenen Arbeiten aufgesetzt werden. In diesem Zuge erfolgte zunächst eine Mängelanalyse.

Im Anschluss wurde das gesetzte Ziel dahingehend neu formuliert, dass die bestehende *HTML-GUI* neu strukturiert werden müsse um nachfolgenden Entwicklern einen leichteren Einstieg in die Erweiterung der selbigen zu ermöglichen. In diesem Rahmen erfolgte eine Bewertung der möglichen Alternativen des zugrundeliegenden Frameworks.

Abschließend wurde dieses Ziel erneut revidiert: Die *HTML-GUI* sollte nicht vollständig, sondern nur in Teilen umstrukturiert werden. Der bessere Entwicklungseinstieg blieb dabei weiterhin im Fokus.

2.1 Einarbeitung und Mängelanalyse

Bevor ich etwas an dem von Bastian Sieker im Rahmen seiner Masterarbeit [Sie15] entwickelten Frontend ändern oder erweitern konnte, musste ich mich zunächst in die von ihm bereitgestellten Artefakte einarbeiten. Während dieser Einarbeitungsphase, probierte ich mich versuchsweise an diversen kleinen Änderungen. Diese waren zumeist einfache Modifikationen einer Beschriftung oder die Abwandlung der Funktionalität eines Buttons. Dabei fiel mir auf, wie mühsam es oft war, das gewünschte Ergebnis innerhalb eines angemessenen Zeitrahmens zu erhalten. Die Gründe für den größten zeitlichen Aufwand werden im Folgenden näher beschrieben.

2.1.1 Ordnerstruktur

Die vorhandene Ordnerstruktur kann eine lange Suche in vielen verschiedenen Ordnern erfordern, um die Dateien zu finden, in denen die für ein bestimmtes Element verantwortlichen Stellen im Quellcode enthalten sind. Diese liegen in der Regel in drei unterschiedlichen Verzeichnissen, stehen jedoch in unmittelbarer Abhängigkeit zueinander.

2.1.2 Build-Skript

Ein weiterer, nicht unerheblicher Zeitfaktor ist der sogenannte Build-Schritt: Bei jeder Änderung im Quellcode muss ein **Production Build** erstellt werden, indem der Entwickler das Kommando `npm run build` ausführen und dann

auf die Beendigung des Skriptes warten muss, was mitunter eine mehr als zehn-sekündige Arbeitspause[10] in der Entwicklung verursacht.

> **Production Build**
> Ein *Production Build* ähnelt der klassischen Code-Kompilierung. In Zusammenhang mit JavaScript-Applikationen ist er im Grunde die Konkatenation und zumeist auch Kompression aller benötigten Dateien, typischerweise in einem `dist`-Ordner[11]. Dies bewirkt, dass der zuvor entstandene JavaScript-Quelltext nur noch in einer Datei, typischerweise in der `bundle.js`, zusammengefasst wird, sodass keine Querreferenzen mehr existieren und die Anwendung auch lokal nutzbar ist. [Gec16]

Dies verlängert insbesondere das in der Einarbeitung häufige Austesten von Funktionen und Möglichkeiten. Auswirkungen einer Änderung können aufgrund der ständigen Unterbrechungen nur schwerlich nachvollzogen werden.

Aber auch in der weiteren Entwicklung, in der davon auszugehen ist, dass der Programmierer weniger experimentieren wird als in der Anfangsphase, schränkt dieser Build-Schritt den allgemeinen Workflow erheblich ein. Gerade unter Berücksichtigung des Prinzips „Change one thing at a time“[12], wie es von David Agans vorgestellt wurde [Aga02] und in der modernen Softwareentwicklung praktiziert werden sollte, bedeutet dies, dass nach jedem minimalen Arbeitsschritt solch eine unfreiwillige Pause eingelegt werden muss, welche ein effizientes Arbeiten deutlich erschwert.

2.1.3 Jade-Templates

Die **Jade**-Templates, welche die Grundlage für die grafische Darstellung aller GUI-Elemente bilden, könnten für den Entwickler eine weitere Herausforderung darstellen. Auch wenn diese Sprache einige Vorteile bietet, so stellt sie, aufgrund ihrer sehr steilen Lernkurve, dennoch eine Hürde für den Entwicklungseinstieg dar (vgl. [Gor14]).

> **Jade**
> *Jade*[13] ist ein so genannter *Template Engine* und wandelt beim Kompilieren den in der Sprache *Jade* geschriebenen Code in HTML-Quelltext um, der vom Browser gelesen werden kann.

[10] vgl. Kommentare unter http://saros-build.imp.fu-berlin.de/gerrit/#/c/2997/

[11] Abkürzung von „distribution“, zu Deutsch: „Vertrieb“

[12] zu Deutsch etwa: „Ändere jeweils nur eine Sache“

Jade orientiert sich im Kern an HTML und „erweitert“ dieses, indem beispielsweise alle spitzen Klammern (`<` und `>`), sowie alle schließenden HTML-Tags (zum Beispiel `</p>` oder `</div>`) wegfallen. Dagegen wird die Zugehörigkeit von Textabschnitten mithilfe von Einrückungen und bestimmten Zeichen (zum Beispiel `=`, `.` oder `|`) bestimmt. Dies soll im Vergleich zu reinem HTML für ein aufgeräumteres und übersichtlicheres Erscheinungsbild sorgen und vor allem dazu führen, dass insgesamt weniger Code geschrieben werden muss.

Ich empfand die vorhandenen Templates allerdings eher als sehr zusammengedrängt und hatte Mühe die oben beschriebenen verwendeten Sonderzeichen richtig zu verstehen. Dadurch war es auch nicht leicht herauszufinden welche verwendeten Tags nun ineinander verschachtelt und welche nebeneinander auf einer Ebene waren. Hinzu kam, dass beim Ausprobieren auf Grund der strengen Whitespace[14]-Regeln, auch bei vermeintlich korrekt eingerückten *Jade*-Dokumenten, häufig Fehlermeldungen beim Kompilieren entstanden. In den meisten Fällen lag dies nur an einer Verwechslung von Tabulatoren und Leerzeichen, aber es führte bei mir dennoch schnell zu Frustration - insbesondere in Kombination mit der langen Ausführungszeit des Build-Skriptes (vgl. Abschnitt 2.1.2 auf Seite 6).

2.1.4 Ampersand.js

Ferner hatte ich starke Schwierigkeiten die Funktionen der einzelnen *JavaScript*-Dateien zu verstehen. Dies lag insbesondere daran, dass das genutzte **MV*-Framework Ampersand.js**[15] für mich nicht intuitiv verständlich war. Das Nachschlagen einzelner Optionen, die für das Unverständnis verantwortlich waren, gestaltete sich allerdings ebenfalls nicht als ausgesprochen einfach. Ein Grund dafür war die vergleichsweise kleine Community, aufgrund derer es kaum Antworten zu häufig gestellten Fragen im Internet zu finden gab. Die bereitgestellte Code-Dokumentation auf der offiziellen *Ampersand.js*-Webseite[16], lieferte ebenfalls nur spärlich Antworten auf aufkommende Fragen.

[13] http://jade-lang.com/
[14] Zwischenraumzeichen, wie Leerzeichen oder Tabulatoren
[15] https://ampersandjs.com/
[16] https://ampersandjs.com/docs/

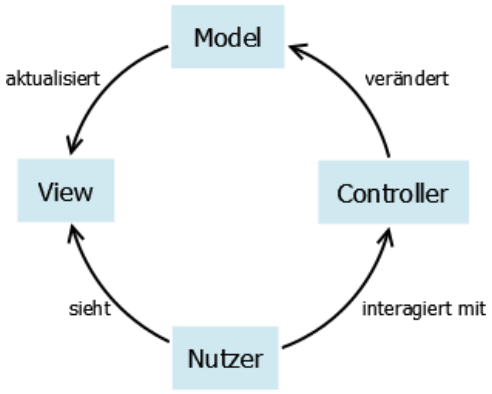

Abbildung 1: Interaktion der einzelnen MVC-Komponenten (vgl. [Fre10])

MV*-Framework
Ein *Framework* im Allgemeinen ist ein „Grundgerüst“, das dazu verwendet werden kann, dem Softwareentwickler einen grundlegenden Rahmen für seine Entwicklungstätigkeit zu geben.

Das *Model-View-Controller*-Konzept, kurz *MVC*, steht für eine Aufteilung der Applikation in drei Komponenten. Diese sind jeweils für einen bestimmten Teil der Anwendung verantwortlich. (vgl. [Osa12] und Abbildung 1)

1. In den **M**odels werden, grob gesagt, die Datentypen, wie zum Beispiel ein *User* repräsentiert.
2. Die **V**iews beschreiben zumeist das, was der Nutzer tatsächlich sehen soll. Sie „beobachten“ die Models und aktualisieren bei einer Änderung dementsprechend die sichtbaren Inhalte.
3. Die **C**ontroller reagieren auf Interaktionen des Nutzers, indem sie gemäß derer die Models verändern.

Das *MV**-Konzept basiert auf der oben beschriebenen *MVC*-Aufteilung. Der Unterschied dabei, ist, dass nicht zwangsläufig ein eigenständiger *Controller* benötigt wird, sondern dessen Verantwortungsbereich auch in das entsprechende Model oder die View integriert werden kann.

Ein *MV*-Framework* ist somit ein *Framework*, das genau dieses *MV**-Konzept implementiert. Gerade in der *JavaScript*-Entwicklung gibt es bereits unzählige solcher *Frameworks*[17], die alle genau diesen konzeptionellen Ansatz verfolgen.

2.2 Änderung statt Erweiterung

Diese Stolpersteine veranlassten mich dazu, ganz neu über meine Aufgabe als Entwickler nachzudenken: Selbst wenn ich mich nach einer längeren Einarbeitungszeit sicherlich gut in dem von Sieker geschriebenen Frontend und der Jade-Syntax auskannte, so fiele es dem nächsten Entwickler, der an dem Frontend arbeiten wollte, vermutlich mindestens genau so schwer wie mir sich einzuarbeiten. Die einzige Möglichkeit dies zu ändern, wäre, die bereitgestellte Dokumentation zu verbessern oder aber eine Änderung des Programmcodes an sich, um ein leichteres Verständnis zu ermöglichen.

Ich formulierte ein neues Ziel: Statt einer Weiterentwicklung der *HTML-GUI*, war nun eine Neustrukturierung dieser geplant. Am Ende sollte sich die *HTML-GUI* wieder mindestens auf dem damaligen Stand befinden. Da ich insbesondere so große Verständnisprobleme mit dem genutzten Framework, *Ampersand.js*, hatte, beschloss ich, dieses zugunsten eines anderen Frameworks auszutauschen. Dabei sollte vor allem auf eine leichtere Einarbeitung für nachfolgende Entwickler geachtet werden.

Sieker zog in seiner Arbeit neben der Verwendung von *Ampersand.js* insbesondere **AngularJS**[18] in Betracht (vgl. [Sie15], S.18-21). Interessant war vor allem wie Sieker *AngularJS* beschrieb:

> „AngularJS [...] allow[s] developers to build sophisticated UIs [...] without deep knowledge of JavaScript. [...] Many common problems can [...] be tackled fast and efficient, however, when [...] customising default behaviour [is required], things can get complicated. [...] AngularJS [has a] very big development community and [a] rich plugin and extension ecosystem.“
> — [Sie15, S. 19]

Im direkten Vergleich dazu, lassen sich für *Ampersand.js* lediglich folgende Vorteile finden:

> „AmpersandJS [...] offers the possibility to use BackboneJS components. [...] the codebase is small, well documented and readable, the level of abstraction is relatively low.“
> — [Sie15, S. 20f]

Nebenbei wird noch die verhältnismäßig kleine Community erwähnt, allerdings mit der Begründung „The impact of the problem is decreased by the

[17] vgl. http://todomvc.com/
[18] https://angularjs.org/

interoperatability with BackboneJS components“ [Sie15, S. 21], was wiederum die Frage offen lässt, warum dann die Tendenz nicht sowieso zu **Backbone.js**[19] ging.

Die von Sieker verwendeten Auswahlkriterien empfand ich als sehr gut. Jedoch hätte der Anzahl an Nutzern einer Programmiersprache, oder in diesem Fall eines Frameworks, aus den folgenden Gründen mehr Bedeutung beigemessen werden sollen: Je größer eine Community ist, desto mehr Lern- und Übungsmaterial lässt sich finden. Auch Probleme, auf die man während des Programmierens stößt, können in den meisten Fällen schneller gelöst werden, da mit einer höheren Wahrscheinlichkeit schon jemand anderes eine ähnliche Frage hatte oder man zumindest binnen kürzester Zeit mit Unterstützung rechnen kann.

Eine kurze Recherche auf *Stack Overflow*[20]liefert konkrete Daten; so wurden über 180.000 Fragen mit dem „Tag“ *angularjs* versehen aber nur knapp 50 mit *ampersand.js*[21]. Auch wenn diese Daten die Größe der Community nicht exakt wiedergeben, da es noch viele weitere Hilfe-Foren und Ähnliches gibt, so halte ich gerade diesen immensen Größenunterschied für sehr repräsentativ.

Aufgrund dieser Ausführungen, die eine Entscheidung zu Gunsten von *AngularJS* hätten bewirken müssen, beschloss ich, dies nachträglich zu tun und die gesamte bis dahin entstandene Codebasis von *Ampersand.js* nach *AngularJS* zu portieren.

2.3 AngularJS

Bei der Einarbeitung in das von *Google, Inc.*[22] entwickelte Framework gewann ich schnell einen positiven Eindruck bezüglich seines Funktionsumfanges. Da ich viel Wert auf eine schnelle und leichte Einarbeitung für nachfolgende Entwickler legte, war ich speziell von der Vielzahl der im Internet verfügbaren Tutorials, sowohl von AngularJS direkt[23], als auch von zahlreichen anderen Anbietern, begeistert. Insgesamt wird es ermöglicht kostenlos und in nur kurzer Zeit einen guten Überblick über die grundlegenden Funktionsweisen dieses Frameworks zu erlangen.

[19] http://backbonejs.org/

[20] http://stackoverflow.com/

[21] vgl. http://stackoverflow.com/questions/tagged/angularjs/info und http://stackoverflow.com/questions/tagged/ampersand.js/info, Stand 07. Juli 2016

[22] https://www.google.com/about/company/

[23] http://campus.codeschool.com/courses/shaping-up-with-angular-js/ (Dieses Tuturial wurde von *Google* gesponsert und ist auf der offiziellen *AngularJS*-Seite[18] verlinkt)

2.3.1 Feature-orientierte Ordnerstruktur

Um bei einer Umstrukturierung möglichst ähnliche Probleme, wie in Abschnitt 2.1 auf Seite 6 beschrieben, zu vermeiden, berücksichtigte ich diese von Beginn an in meinen Überlegungen.

In der Entwicklung mit MV*-Frameworks ist es üblich, seine Dateien in einer Typen-basierten Ordnerstruktur, also, nach Dateiendung und entsprechender Funktion benannten Ordnern zu verwalten. Dies bedeutet, dass alle *Views* in einem „Views“ genannten Ordner liegen und alle *Model-* und *Controller*-Dateien befinden sich in dementsprechend benannten Ordnern, die sich wiederum in einem `js`-Ordner befinden. Dies ist zwar für kleinere Projekte meist ausreichend übersichtlich, sobald diese aber umfangreicher werden, ist diese häufig nicht mehr gewährleistet und führt dann zu Schwierigkeiten bei der Suche nach einzelnen Dateien, ohne sich vorher komplett einzuarbeiten (vgl. Abschnitt 2.1.1 auf Seite 6). Daher überlegte ich mir einen, für diese Zwecke, etwas unkonventionelleren Weg zu gehen: Ich erwog, nicht eben jene MV*-typische Ordnerstruktur aufzubauen, sondern eine Struktur, bei der eng verwandte Dateien in einen gemeinsamen Ordner liegen (vgl. Abbildung 2).

Nach Typ Sortiert

```
src
├── scripts
│   ├── controller
│   │   ├── contacts-controller.js
│   │   ├── project-tree-controller.js
│   │   ├── main-page-controller.js
│   │   └── session-wizard-controller.js
│   │
│   └── models
│       ├── contacts-model.js
│       ├── project-tree-model.js
│       ├── main-page-model.js
│       └── session-wizard-model.js
│
├── views
│   ├── contacts.html
│   ├── project.html
│   ├── main-page.html
│   └── session-wizard.html
│
├── app.js
└── index.html
```

Nach Feature sortiert

```
src
├── elements
│   ├── contacts
│   │   ├── contacts-controller.js
│   │   ├── contacts-model.js
│   │   └── contacts.html
│   │
│   └── project-tree
│       ├── project-tree-controller.js
│       ├── project-tree-model.js
│       └── project-tree.html
│
├── pages
│   ├── main-page
│   │   ├── main-page-controller.js
│   │   ├── main-page-model.js
│   │   └── main-page.html
│   │
│   └── session-wizard
│       ├── session-wizard-controller.js
│       ├── session-wizard-model.js
│       └── session-wizard.html
│
├── app.js
└── index.html
```

Abbildung 2: Unterschiede zwischen einer Typen-basierte Ordnerstruktur neben einer solchen Feature-orientierten

Aufgrund der Situation, dass die meisten Mitglieder des Saros-Teams im Zuge ihrer Abschlussarbeit an dem Produkt arbeiten und dementsprechend in den meisten Fällen an einem sehr stark spezifizierten Teil entwickeln, kann gesagt werden, dass dort eine **modulare Entwicklung** vorherrscht. Eine **Feature-orientierte Dateiorganisation** hat insbesondere bei dieser Entwicklungsweise den Vorteil, dass ein Entwickler, der derzeit an einem bestimmten Feature arbeitet, sich im Regelfall auch nur in einem konkreten

Ordner bewegen muss. Sie wird zwar nicht von dem offiziellen *AngularJS*-**Seed**[24] unterstützt, aber in einem beliebten, inoffiziellen Seed von John Papa stark befürwortet und umgesetzt [Pap14]. Dieser Seed wird auch von dem *AngularJS*-Team als „Best Practice“ empfohlen[25]. Aufgrund dieser besseren Eingliederung in den allgemeinen Workflow des Entwickler-Teams und die bessere Übersichtlichkeit, begann ich, genau eine solche Struktur zu entwerfen.

> **Modulare Entwicklung**
> Bei einer modularen Programmierung wird die zu implementierende Funktionalität in kleinere Teilblöcke, so genannte Module, zerlegt, die weitestgehend unabhängig voneinander operieren können.

> **Seed**
> Ein **Seed** ist eine minimale Vorlage oder ein Grundgerüst, das für die schnelle Aufsetzung einer Entwicklungsumgebung bestimmt ist. In ihm werden beispielsweise alle grundlegenden Einstellungen bereits vordefiniert, damit der Entwickler diese nur noch bei Bedarf anpassen muss und somit schneller an der eigentlichen Entwicklung arbeiten kann.

Im Anschluss daran begann ich die grundlegende Architektur der neu entstehenden JavaScript-Applikation zu erstellen, indem ich eine Haupt-Script-Datei, *app.js*, erstellte. In dieser definierte ich den Kern-Controller, auf den sich alle nachfolgenden Module stützen sollten. Damit die Kommunikation zu dem Java-Backend funktionieren würde, band ich in diesen Controller die bestehende *saros-api.js* ein.

> **saros-api.js**
> Die **saros-api.js** ist die JavaScript-seitige Implementierung der Schnittstelle zwischen Back- und Frontend in Saros. Sie beinhaltet alle Funktionen, die von der HTML-GUI aufgerufen und an das Java-Backend zur Ausführung weitergeleitet werden können.

Darauf aufbauend modellierte ich das *Account*-Modul und schrieb die in *Ampersand.js Model* genannte Struktur der einzelnen Accounts, also den Prototypen, in eine in *AngularJS* genannte *Factory* um. Während ich nach einer geeigneten Lösung in *AngularJS* für die Account-Liste[26] suchte, fiel

[24] https://github.com/angular/angular-seed

[25] http://angularjs.blogspot.de/2014/02/an-angularjs-style-guide-and-best.html

[26] Diese beinhaltet alle Accounts, mit denen der Nutzer sich anmelden kann

mir auf, dass viele der von *AngularJS* genutzten Funktionalitäten nur eingeschränkt nutzbar waren. Zum Beispiel die HTML-Attribute, wie `<ng-view>` und `<ng-click>`, konnten nicht von dem genutzten SWT-Browser verarbeitet werden. Da dieses Problem in gängigen Browsern, wie Mozilla Firefox und Google Chrome, nicht auftrat, musste davon ausgegangen werden, dass diese Einschränkungen ein schwer lösbares Kompatibilitätsproblem mit dem genutzten *SWT-Browser* darstellen.

2.4 Angular 2

Dies veranlasste mich dazu, meine ursprüngliche Entscheidung zugunsten von *AngularJS* erneut zu überdenken. Allerdings war ich den grundsätzlichen Konzepten des Frameworks immer noch sehr angetan. Den Nachfolger von *AngularJS*, welches zur besseren Übersicht im Folgenden *Angular 1* genannt wird, **Angular 2**[27], hatte ich bisher von vornherein ausgeklammert, da dieser sich derzeit noch in der Beta-Phase befindet und ich keine „unfertigen“ Produkte einbinden wollte. Über Sieker, den ursprünglichen Entwickler des gesamten *HTML*-Frontends, erfuhr ich allerdings, dass *Angular 2* bereits produktiv eingesetzt wird. Daraufhin informierte ich mich über den tatsächlichen Stand in der Entwicklung von *Angular 2* und musste feststellen, dass dieser tatsächlich sehr viel weiter fortgeschritten war, als ich bisher angenommen hatte. Diverse Webseiten[28] belegen außerdem, dass der Support für *Angular 1*-Applikationen innerhalb der nächsten Zeit, sobald die Mehrheit aller Nutzer zu *Angular 2* gewechselt sind, eingestellt werden würde. Dies hätte zur Folge, dass zukünftig wieder ein Entwickler benötigt wäre um die Applikation erneut umzustrukturieren, falls diese auf dem neusten Stand bleiben sollte.

Die Gründe, wegen derer ich mich ursprünglich für *Angular 1* entschieden hatte, blieben für *Angular 2* dieselben: Es sind zahlreiche Artikel, Videos, sowie Tutorials zum Einarbeiten[29] vorhanden und auch wenn die „Angular 2“-Community derzeit (noch) nicht so groß ist, wie die von *Angular 1*[30], so kann dennoch bereits jetzt ein steigendes Interesse daran ermittelt werden (vgl. Abbildung 3 auf der nächsten Seite).

Ich entschied mich dafür, meine bisher getane Arbeit zu verwerfen und erneut anzufangen, nur dieses Mal unter der Verwendung von *Angular 2* statt *Angular 1*.

---

[27] https://angular.io/

[28] z.B. https://jaxenter.com/angular-2-is-coming-soon-but-angular-1-is-not-going-anywhere-121678.html

[29] http://www.angular2.com/

[30] rund 8.500 „Stack Overflow“-Fragen, vgl. http://stackoverflow.com/questions/tagged/angular2

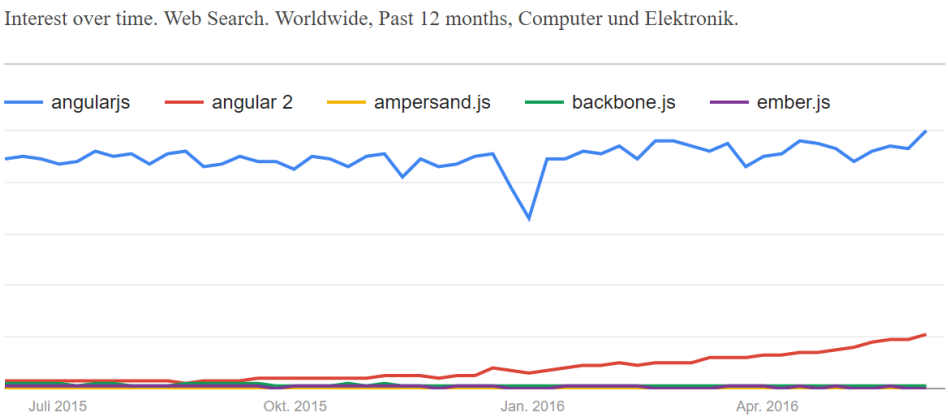

Abbildung 3: Vergleich der Suchanfragen auf http://www.google.com von *Angular 1*, *Angular 2*, *Ampersand.js*, sowie den beiden anderen von Sieker in Erwägung gezogenen Frameworks, Backbone.js und Ember.js — Quelle: http://www.google.com/trends

Ich begann wie zuvor mit dem Aufsetzen des Grundgerüsts. Ich legte zunächst eine *app.js* und eine *main.js*-Datei an, die jeweils dafür verantwortlich waren die komplette Applikation beziehungsweise die „main-page“ zu steuern. Bevor ich jedoch auch nur eine größere Funktionalität implementieren konnte, fiel mir schon beim Testen meiner bisherigen, nahezu statischen Seite auf, dass für das Laden der *Angular 2*-Module **XMLHttpRequests** vom Browser gesendet werden. Diese Requests werden jedoch nur von bestimmten Protokoll-Schemata unterstützt, wie z.B. **HTTP**, welche allerdings nicht lokal ausgeführt werden können.

HTTP
Das *Hypertext Transfer Protocol*, kurz *HTTP*, ist ein Dateiübertragungsprotokoll innerhalb des World Wide Webs.

XMLHttpRequest
Der *XMLHttpRequest*, kurz *XHR* ist eine API für die Versendung von HTTP-Anfragen an einen Server. Dies wird genutzt um beispielsweise Teile einer Webseite zu aktualisieren ohne sie vollständig neu zu laden. (vgl. [Net05] und [W3S])

Um dieses Problem zu umgehen, müsste man entweder einen lokalen Server mitliefern oder aber, wie bereits unter *Ampersand.js*, nach jeder Änderung im Quellcode einen *Production Build* durchführen, der alle benötigten Dateien vor der Laufzeit zusammenfasst.

Ein mitgelieferter Server würde einen unverhältnismäßig hohen Aufwand im Vergleich zu dem erwarteten Nutzen bedeuten - es geht schließlich immer noch „nur“ um die grafische Benutzeroberfläche. Ein Production Build dagegen käme mit den unter Abschnitt 2.1.2 auf Seite 6 beschriebenen Nebeneffekten einher, die möglichst vermieden werden sollten. Daher waren beide Möglichkeiten nicht ausreichend zufriedenstellend.

2.5 Vanilla JS

Vanilla JS
Vanilla JavaScript[31] ist natives, reines JavaScript ohne den Einsatz jeglicher Frameworks.

Im Zuge dieser anfänglichen Überlegungen entdeckte ich einen vollkommen neuen Lösungsansatz. Jegliche Probleme, die bisher aufkamen, wurden durch das verwendete Framework verursacht. So gab es unter der Verwendung von *Ampersand.js* Unklarheiten bezüglich der Verwendung der internen Funktionen, da es schwierig war entsprechende Dokumentation oder Beispiele zu finden, die diese verständlich erklärten.

Bei *Angular 1* gab es das Problem der nicht-nativen HTML-Attribute, die sich nicht mit dem verwendeten SWT-Browser vertrugen. Und abschließend nun unter *Angular 2* die Unannehmlichkeit des notwendigen Build-Schrittes, welche im Übrigen auch schon unter *Ampersand.js* bestand.

Warum wurde überhaupt ein Framework benötigt? Die zu entwickelnde Applikation war nicht besonders anspruchsvoll und benötigte im Grunde keine der von den verwendeten Frameworks angebotenen Features. Alles was auf den ersten Blick erforderlich war, war eine dynamische Datenbindung, so genanntes *Data-Binding*, zwischen der HTML-Datei und der dahinterliegenden JavaScript-Anwendung die bereits über einen einfachen Eventbus mit dem Java-Backend kommunizierte. Doch nur für diese verhältnismäßig kleine Funktionalität ein so umfangreiches Framework, wie es *Angular* (sowohl *1*, als auch *2*) ist, einzusetzen, erscheint etwas übertrieben. Gerade weil *Data-Binding* mithilfe der sowieso bereits eingesetzten Events relativ einfach und vor allem übersichtlich und intuitiv selbst in *Vanilla JS* realisiert werden könnte (vgl. [**Hyde2014)**].

Data-Binding
Als *Data-Binding* wird in Bezug auf MVC die Synchronisation der View mit dem Model bezeichnet.

[31] http://vanilla-js.com/

Der entscheidende Vorteil bei der Nutzung von purem JavaScript liegt darin, dass besonders in der Web-Entwicklung unerfahrene Programmierer nicht mit der Einarbeitung in gleich zwei Technologien (das Framework und JavaScript an sich) konfrontiert werden, sondern sich lediglich um das Verständnis von JavaScript bemühen müssen. Da auch grundsätzlich am Backend arbeitende Personen des Öfteren zu Testzwecken am Frontend Änderungen vornehmen müssen, ist dieser Punkt der Erlernbarkeit nicht zu verachten.

Nach kurzer Arbeitszeit stellte sich jedoch heraus, dass die wenigen Funktionen, die selbst implementiert werden müssten, etwas mehr als nur das Data-Binding umfassten. So sollte es beispielsweise ein Modell für die Kontakte oder Accounts geben, auf das bei Verwendung der API zurückgegriffen werden kann. Auch diese Aufgabe wäre lösbar, da zu beinahe jedem Problem, immer ein Beispiel in *Vanilla JS* zu finden ist, welches genau das löst[32]. Jedoch stellt sich hier die Frage nach dem Sinn: Warum etwas genau so aus einer externen Quelle „herauskopieren“, wenn man diese auch einfach nur referenzieren und dann benutzen kann?

Ideal wäre also eine Mischung aus einem großen, klobigen Framework, wie *Angular*, und der übersichtlichen, kleinen, selbstgeschriebenen Variante in *Vanilla JS*. Es sollte möglichst viele Funktionalitäten bereitstellen, ohne, dass sie zwangsläufig alle verwendet werden müssen. Etwas, das dem Entwickler die Möglichkeit bietet, ausschließlich auf die Funktionen zurückzugreifen, die er benötigt und ihm nicht alles in einem großen Paket liefert. Eine Art modulares Framework wäre also perfekt.

2.6 Zurück zu den Ursprüngen

> A highly modular, loosely coupled, non-frameworky framework for building advanced JavaScript apps.
> — https://ampersandjs.com/

Es stellte sich heraus, dass genau das was ich suchte, bereits von Anfang an vorhanden war. *Ampersand.js* ist genau die Art von Framework, die mir am Ende meines Arbeitsprozesses als die optimale Lösung erschien. Ich musste mir somit eingestehen, dass Siekers getroffene Entscheidung tatsächlich nicht so verkehrt war, wie es mir am Anfang erschien. Im Gegenteil - sie wurde zwar auf der Basis von, meiner Ansicht nach, nur unzureichenden Gründen getroffen, allerdings ist das grundlegende Konzept von *Ampersand.js* genau das, wonach ich am Ende meiner Odyssee suchte. So entschied ich mich

[32] z.B.: https://github.com/tastejs/todomvc/blob/master/examples/vanillajs/js/model.js

gegen die Einführung eines neuen JavaScript-Frameworks und für die weitere Verwendung von *Ampersand.js*.

Aufgrund dieses Beschlusses, formulierte ich das Ziel dieser Arbeit erneut um: Statt einer vollständigen Neustrukturierung der *HTML-GUI* durch den Austausch des zugrunde liegenden Frameworks, sollten nun die in Abschnitt 2.1 auf Seite 6 erkannten Defizite weitestgehend behoben werden. Mein Ziel war es, den Einstieg in die Weiterentwicklung für nachfolgende Entwickler zu vereinfachen. Zudem sollte - wie ursprünglich geplant - die *HTML-GUI* um einige Funktionalitäten erweitert werden.

3 Entwicklung

In Folge der anfänglichen Mängelanalyse wurde eine Suche nach einer bessere Alternative zu *Ampersand.js* durchgeführt, deren Ergebnis in der Erkenntnis darüber bestand, dass das bereits genutzte Framework doch die beste Wahl war.

Im nachfolgenden Kapitel wird nun die Umsetzung des zuletzt formulierten Ziels angestrebt, dass den leichteren Einstieg in die Weiterentwicklung der *HTML-GUI* durch konkrete Änderungen an dem bestehenden System, sowie die Erweiterung dessen beinhaltet.

3.1 Änderungen

Die anfänglich (vgl. Abschnitt 2.1 auf Seite 6) aufgeführten Problematiken des bestehenden Systems blieben natürlich bestehen. Der Einstieg in ein Framework wie Ampersand.js, welches lediglich über eine sehr überschaubare Dokumentation, sowie keinerlei Tutorials und eine nur sehr kleine Community verfügt, ist für einen Softwareentwickler immer eine Hürde. Insbesondere, wenn dieser sich nicht hauptsächlich mit dem Frontend beschäftigt, sondern dort nur eine kleine Änderung zu Testzwecken durchführen möchte, sollte nicht von ihm verlangt werden, sich zuerst mehrere Tage in dieses einarbeiten zu müssen. Daher sollten die anderen von mir beschriebenen Probleme, nicht für weitere Schwierigkeiten beim Verständnis sorgen, sondern möglichst auf eine Art und Weise behoben werden, sodass zumindest diese keinen weiteren erhöhten Lernaufwand bewirkten.

In der folgenden Tabelle liste ich alle von mir als den Entwicklungs- beziehungsweise Einarbeitungsfluss beeinträchtigend empfundene Sachverhalte auf. Außerdem fasse ich meine persönliche Einschätzung des jeweiligen Arbeitsaufwands und dem daraus resultierenden Nutzen zur Behebung des entsprechenden Problems in einem Bewertungssystem zusammen, wobei der Wert 0 keinen und 5 ein sehr großer Aufwand beziehungsweise Nutzen bedeutet. Nachfolgend werden die einzelnen Punkte eingehender besprochen.

Da viele nachfolgende Änderungen sich direkt auf die Projektstruktur bezogen (wie beispielsweise eine Umarbeitung des Build-Skripts, in dem die Lage der benötigten Dateien bekannt sein muss), wollte ich dieses Problem zuerst angehen. Auch war der Arbeitsaufwand einer Reorganisation des Aufbaus zeitlich sehr gut abschätzbar, da lediglich die vorhandenen Dateien in neue Ordner verschoben und die Referenzen der `require`-Statements am Anfang jeder *JavaScript*-Datei entsprechend aktualisiert werden müssten.

Da ich anfangs nicht einschätzen konnte, wie eine mögliche Lösung für das *Build-Skript-Problem* aussehen könnte, war es mir nicht möglich eine Zeitauf-

Problem	Erläuterung des Problems	Erläuterung der Lösung	geschätzter Aufwand	geschätzter Nutzen
Ordner-struktur	Abschnitt 2.1.1 auf Seite 6	Abschnitt 3.1.1	2	3
Build-Skript	Abschnitt 2.1.2 auf Seite 6	Abschnitt 3.1.2 auf Seite 22	?	5
Template Engine	Abschnitt 2.1.3 auf Seite 7	Abschnitt 3.1.3 auf Seite 25	4	2

Tabelle 1: Bewertung der einzelnen Probleme

wandsschätzung zu machen. Aufgrund des sehr hoch eingeschätzen Nutzens, wollte ich dieses Problem dennoch im Anschluss an die Reorganisation der Dateien angehen.

Eine Änderung des Template Engines, klammerte ich zunächst aus, da ich keinen besonders hohen Nutzen erwartete und den damit verbundenen Aufwand dagegen als eher unverhältnismäßig groß einschätzte.

3.1.1 Ordnerstruktur

Wie bereits in Abschnitt 2.3.1 auf Seite 12 erläutert, sah ich einen großen Nutzen in der Strukturierung der Dateien nach ihrer Funktion anstatt nach ihrer Dateiendung. Alle Elemente und Seiten sollten fortan in jeweils einem eigenen Ordner mit allen dazugehörigen *Templates*, *Views* und gegebenenfalls *Models* zu finden sein. Sollte nun etwas bestimmtes geändert werden, so wäre sofort klar, wo die entsprechenden Dateien liegen, auch ohne sich in die vorhandene *MV**-Struktur einzuarbeiten. Auch wenn anzunehmen ist, dass dies für Entwickler, die mit diesem Konzept bereits vertraut sind, keine enorm große Verbesserung darstellt, so ist doch davon auszugehen, dass es das Verständnis nicht beeinträchtigt. Dagegen erleichtert es allerdings die Suche nach Dateien für all jene Personen, die sich damit nicht auskennen.

Einen weiteren positiven Effekt hat diese Umstrukturierung auf die Bereitstellung von Änderungen. Da für jede Modifizierung eines bestimmten Elementes nur noch ein Ordner „angefasst“ werden muss, ist bei der Ansicht der veränderten Verzeichnisse zu erkennen, was geändert wurde.

Die Umsetzung dieser Neustrukturierung[33] resultierte in einem nach allgemeinen Dateien (wie CSS- und Font-Dateien), Elementen und Seiten geordneten Dateibaum. Jedes Element, bzw. jede Seite erhielt ein eigenes Unterverzeichnis in dem dazu gehörigen Ordner. In diesem liegen nun jeweils eine

[33] http://saros-build.imp.fu-berlin.de/gerrit/#/c/3063/

Template- (*Jade*), sowie die dazugehörige, für das Rendern verantwortliche View-Datei (*JavaScript*) paarweise vor.

Einige Elemente haben mehr als eine Ansicht, wie zum Beispiel die Kontakte, bei denen es abgesehen von der Einzelansicht (*contact*) noch eine Listenansicht aller Kontakte (*contacts*), sowie eine Listenansicht, bei der es möglich ist einzelne Kontakte auszuwählen (*selectable-contacts*), gibt. In diesen Element-Verzeichnissen gibt es dann analog mehr solcher *Jade-JavaScript*-Paare.

Zudem gibt es zu einigen Elementen ein oder mehrere passende Modelle, auf die sich in den Views bezogen wird. Bei den Kontakten gibt es so beispielsweise jeweils ein Modell für einen Kontakt, sowie eine Kontaktliste (wieder *contact* bzw *contacts* genannt). Diese liegen in einem gesonderten Verzeichnis innerhalb des entsprechenden Elemente-Ordners, um eine Verwechslung mit den Views zu vermeiden.

Das Ergebnis dieser Änderung sieht aus, wie in Abbildung 4 gezeigt.

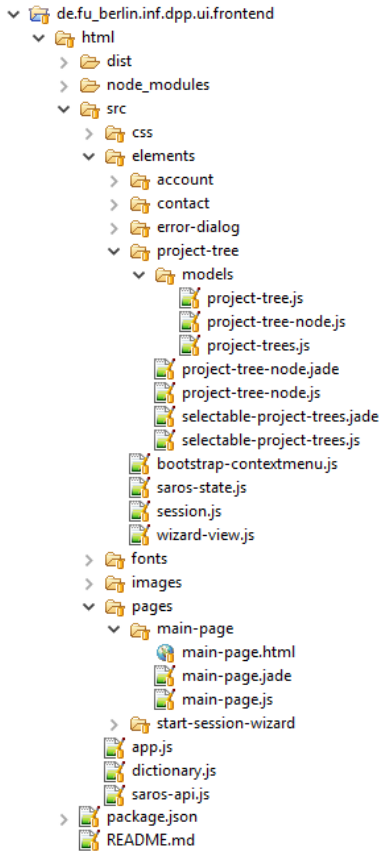

Abbildung 4: Neue Ordnerstruktur

3.1.2 Watch-Skript

Das unter Abschnitt 2.1.2 auf Seite 6 erläuterte Problem des durch das Build-Skript gestörten Arbeitsflusses ist für eine reibungslose Entwicklung das störendste.

Es wurde bereits Anfang diesen Jahres ein möglicher Lösungsansatz vorgestellt[34]. Dieser behob allerdings nur einen kleinen Teil des gesamten Problems: das manuelle Ausführen des Kommandos. Das Skript an sich benötigte nach wie vor mehrere Sekunden zur Ausführung, während derer das Testen des Ergebnisses nicht möglich war. Hinzu kam, dass die automatische Kommandoausführung nur funktionierte, wenn innerhalb der IDE *eclipse* entwickelt wurde. Da diese allerdings für die JavaScript-Entwicklung nur bedingt gut einsetzbar ist, war die vorgestellte Lösung, alles in allem, nicht zufriedenstellend.

Ich versuchte also, die Bewältigung des Problems auf eine andere Weise anzugehen. Dazu untersuchte ich zuerst die Ursache der langen Ausführungszeit. Diese wird durch den Umfang der von dem Skript durchgeführten Änderungen verursacht. Es werden jedes mal mehrere Ordner erstellt, in denen immer die selben Dateien hineinkopiert werden und der komplette JavaScript-Baum wird vollständig neu aufgebaut. Dies ist in den meisten Fällen gar nicht nötig, da stets nur eine minimale Änderung erfolgte.

Der gesamte Build-Prozess kann in die folgenden Schritte unterteilt werden:

1. **prebuild** erstellt einen *bundle*- und einen *dist*-Ordner, sowie in die Unterordner *bundle*, *css* und *fonts*.
2. **build** führt *build:jade* und *build:js* aus:
   2.1. **build:jade**: Das *Templatizer*-Modul[35] erstellt aus allen *Jade*-Templates eine *templates.js*, in der die einzelnen Templates in Form von HTML-Strings geladen und dann wiederum in einem Array exportiert werden. So kann im nachfolgenden *build:js*-Schritt auf diese zugegriffen werden.
   2.2. **build:js**: Das *Browserify*-Modul[36] fasst alle benötigten JavaScript-Dateien in eine *bundle.js*-Datei zusammen. Dabei wird von der *app.js*, dem Kernstück der Anwendung, ausgegangen und alle mit `require('')` angeforderten Module und wiederum deren Abhängigkeiten mit in die *bundle*-Datei hinein geladen.
3. **postbuild** kopiert abschließend alle *HTML*-, *CSS*- und *Font*-Dateien,

[34] http://saros-build.imp.fu-berlin.de/gerrit/#/c/2997/
[35] https://github.com/HenrikJoreteg/templatizer
[36] http://browserify.org/

sowie die gebildete `bundle.js` in den dist-Ordner beziehungsweise in einen der respektiven Unterordner.

Betrachtet man diese Liste, so ist schnell ersichtlich, dass ein Großteil der Teilprozesse jeweils nur in seltenen Fällen durchgeführt werden müssten und nicht nach jeder Änderung des Quellcodes:

So zum Beispiel der komplette prebuild, in dem jedes Mal Ordner angelegt werden, die allerdings nur zu Beginn der Entwicklung fehlen. Zu jedem nachfolgenden Zeitpunkt bestehen die Ordner bereits. Ihre Neuerstellung kostet jeweils unnötig viel Zeit.

Auch der postbuild-Schritt ist meist nicht notwendig. Auf das Kopieren der `bundle.js` kann insofern vollständig verzichtet werden, als dass diese Datei automatisch generiert wird und somit die Ausgabedatei einfach direkt an der gewünschte Stelle innerhalb des `dist`-Verzeichnisses erstellt werden kann.

Der andere Teil des postbuilds, das Kopieren der *HTML*-, *CSS*- und *Font*-Dateien, ist in den meisten Fällen ebenfalls nicht erforderlich. Diese Dateien werden nur in außergewöhnlichen Fällen verändert und werden daher zumeist überflüssigerweise erneut in den `dist`-Ordner kopiert. Nötig ist das Aktualisieren dieser Dateien, so wie das Erstellen des `dist`-Ordners und seine Unterverzeichnisse, also nur am Anfang einer Entwicklung und eben in jenen seltenen Fällen in denen etwas an ihnen verändert wurde.

Im Grunde ist nach einem typischen Entwicklungsschritt lediglich das Erstellen beziehungsweise Aktualisieren der `bundle.js`, inklusive der Kompilierung aller möglicherweise geänderten *Jade*-Dateien, notwendig. Es ist also möglich, das Erstellen der Ordner und das Kopieren der Dateien in einem Schritt durchzuführen und in der weiteren Entwicklung nur noch die Kommandos `build:js` beziehungsweise `build:jade` auszuführen, je nach dem was geändert wurde.

Der Übersichtlichkeit halber erstellte ich zwei neue Skripte, angelehnt an die von Sieker geschriebenen pre- und postbuild-Skripte, die die Ordnererstellung und das Kopieren übernehmen. Diese wurden nun innerhalb des `pre`- und `postbuild`s aufgerufen. Es ist zu beachten, dass aufgrund der geänderten Ordnerstruktur nicht nur *HTML*-, *CSS*- und *Font*-Dateien in den `/dist`-Ordner kopiert werden, sondern auch Bilder, die seit der Umstrukturierung in einem eigenen `/images`-Ordner zu finden sind.

```
"prebuild": "npm run createdist",
"postbuild": "npm run copysrcs",
"createdist": "mkdirp dist/bundle/ dist/css dist/fonts",
"copysrcs": "cp src/pages/*/*.html dist/ && cp src/css/*.* dist
   /css/ && cp src/fonts/*.* dist/fonts/ && cp src/images/*.*
   dist/images/"
```

Um die Ausführungszeit noch weiter zu reduzieren untersuchte ich die verwendeten Module. Das „bundlen“ der *JavaScript*-Dateien wird von *Browserify* durchgeführt. Von diesem Modul gibt es eine Variante, *Watchify*[37], die es ermöglicht alle in der `app.js` direkt und indirekt referenzierten Dateien zu überwachen und bei einer Änderung die `bundle.js` automatisch zu aktualisieren, sodass das manuelle Eintippen des Kommandos `npm run build` nicht mehr nötig wäre. Dies erschien mir eine sinnvolle und vor allem zeitsparende Möglichkeit für den Build zu sein, sodass ich diese in einem „Watch“-Skript umsetzte. Um bei der Ausführung dessen immer alle Ordner, sowie *HTML*-, *CSS* und Font-Dateien zu haben, führte ich zudem ein „prewatch“-Kommando ein, welches stets vor dem eigentlich „watch“ ausgeführt werden würde und, genau wie der `pre-` und `postbuild`, erst den `createdist` und anschließend den `copysrcs` Befehl ausführte.

```
"prewatch": "npm run createdist && npm run copysrcs",
"watch": "watchify -v -d src/app.js -o dist/bundle/bundle.js"
```

Da diese Lösung allerdings nur die *JavaScript*-Dateien beobachtete suchte ich anschließend nach einer ähnlichen Möglichkeit für die Überwachung der *Jade*-Dateien. Dabei stieß ich auf *Jadeify*[38], welches als Modul von *Browserify* verwendet werden kann. Mit dessen Hilfe ist es möglich, dass *Browserify* direkt die benötigten *Jade*-Dateien kompilieren kann und nicht auf ein externes Modul, wie *Templatizer*, angewiesen ist. Da *Watchify* mit *Browserify* kompatibel ist[39], kann dieses Modul auch davon verwendet werden. Es war nur eine minimale Änderung des `watch`-Skriptes nötig und sofort wurde auch auf die Änderung an *Jade*-Dateien geachtet.

```
"watch": "watchify -v -d -t jadeify src/app.js -o dist/bundle/
    bundle.js"
```

Von nun an musste nicht mehr nach jeder Änderung `npm run build` ausgeführt werden, sondern lediglich einmal zu Beginn der Entwicklung `npm run watch`. Die Ausführungszeit dessen liegt beim Starten aufgrund des `prewatch` -Skriptes bei rund zehn Sekunden. Jede weitere Änderung erfolgt allerdings in aller Regel in unter einer Sekunde. Dies bringt dem Entwickler nicht nur einen entscheidenden zeitlichen Vorteil, auch muss das Kommando zum „Build“ nicht mehr manuell ausgeführt werden, was die Entwicklung erheblich erleichtert, da die eigenen Gedankengänge nicht mehr unterbrochen werden.

[37] https://github.com/substack/watchify

[38] https://github.com/domenic/jadeify

[39] “Use watchify, a browserify compatible caching bundler [...]“ (http://browserify.org/#more, 04. Juli 2016)

3.1.3 Jade-Templates

Die von Sieker eingeführten *Jade*-Templates beurteilte ich anfangs als schwer erlernbar (vgl. Abschnitt 2.1.3 auf Seite 7). Sie warfen bei mir bereits in der Einarbeitungsphase die Frage auf, warum für die wenigen, kurzen Templates, die keine der von Jade unterstützten Features nutzten, überhaupt ein Engine verwendet wurde. Meiner ersten Einschätzung nach, könnten die benötigten Vorlagen auch einfach in leichter erlernbarem HTML geschrieben werden.

Bei näherer Betrachtung dieser Einschätzung musste ich allerdings bald feststellen, dass die Einführung von schlichtem HTML anstelle der Jade-Dateien mehr Probleme als erwartet mit sich bringen würde. Insbesondere die dynamisch generierten Inhalte, die mithilfe des Data-Bindings an die Templates übermittelt werden, könnten nur mit viel Aufwand in eine reine HTML-Oberfläche integriert werden.

Daher untersuchte ich den Nutzen einer Einführung eines anderen, möglicherweise einfacher erlernbaren Template Engines. Hierzu folgt eine Liste einiger Alternativen, welche ich in Bezug auf ihre Syntax evaluiere. Diese sollte möglichst strukturiert und auch mit einer nur geringen Einarbeitungszeit zu verstehen sein.

Des Weiteren sollte die Wahl auf einen der Engines fallen, welcher über eine ausreichend große Community verfügt, um bei nicht in der offiziellen Dokumentation enthaltenen Fragen eine externe Quelle finden zu können, die diese beantworten kann. Hierzu wird, wie bereits in Abschnitt 2.2 auf Seite 10 beschrieben, die Anzahl an „Tags“ auf *StackOverflow*[20] als Bewertungsmaßstab genommen.

1. **EJS**[40]

   EJS (*Embedded JavaScript*) hat keine Whitespace-Regelungen, wie *Jade* sondern benutzt im weitesten Sinne schlichtes *HTML*. Dennoch ist die Übersichtlichkeit des in *EJS* geschriebenen Templates durch die Verwendung von zahlreichen `<% function()%>`- beziehungsweise `<%= variable %>`-Tags stark eingeschränkt. Aufgrund dessen, kann dieser Template Engine nicht als Verbesserung von *Jade* bezeichnet werden.

2. **Dust.js**[41]

   Dust.js wirkt verglichen mit *Jade* und auch *EJS* bereits deutlich übersichtlicher. Syntaktisch wird eine Mischung aus purem *HTML* in Kombination mit speziellen Kennzeichnungen in Form von geschwungenen Klammern (`{ }`) genutzt.

[40] http://embeddedjs.com/
[41] http://www.dustjs.com/

Allerdings besteht hier das Problem, der nur sehr kleinen Community - auf *StackOverflow* lassen sich lediglich knapp 400 Fragen dazu finden[42]. Aus diesem Grund ist von einer Wahl diesen Template Engines abzusehen.

3. **mustache**[43]

   Der Template Engine *mustache* weist eine ähnlich übersichtliche Syntax wie *dust.js* auf. Er verwendet dabei ausschließlich Ausdrücke in doppelt geschwungen Klammern (`{{ }}`) . Diese stören den Lesefluss des Dokuments nicht, sodass der Leser den Eindruck erhält, er befasse sich mit reinem *HTML*, das lediglich um wenige Details ergänzt wurde. Auch auf *StackOverflow* lassen sich beinahe 1400 Fragen finden[44], was für eine ausreichend große Community spricht.

4. **Handlebars.js**[45]

   Handlebars.js, im Folgenden *Handlebars* genannt, basiert auf *mustache* und erweitert dieses noch um einige nützliche Features[46]. Hinzu kommt, dass *Handlebars* in Bezug auf die Performance deutlich bessere Ergebnisse erzielt als *mustache*.

   Auch die Suche auf *StackOverflow* liefert gute Ergebnisse: Mit knapp 5000 Fragen hat *Handlebars* verglichen mit den anderen hier vorgestellten Template Engines eine vielfach größere Nutzergemeinde.

Die beste Alternative zu den bisher verwendeten *Jade*-Templates ist somit eindeutig *Handlebars*. Da es für diesen zudem ebenfalls ein unter *Browserify* genutztes Modul gibt[47], wären auch nach einer Einführung von *Handlebars*, die in Abschnitt 3.1.2 auf Seite 22 umgesetzten Verbesserungen am Build-Skript nicht zunichte gemacht.

3.2 Erweiterungen

Neben den oben genannten Veränderung an den vorhandenen Artefakten, bei denen die Entwicklung einer entwicklerfreundlicheren, internen Struktur im Vordergrund stand, war das zweite Ziel dieser Arbeit die nutzerorientierte Erweiterung der bestehenden HTML-GUI. Da für eine Web-Oberfläche, die den gleichen Funktionsumfang der in SWT realisierten Oberfläche aufweist, noch einige Funktionen implementiert werden müssten, fasste ich alle

[42] http://stackoverflow.com/tags/dust.js/info, Stand 29. Juni 2016
[43] https://mustache.github.io/
[44] http://stackoverflow.com/tags/mustache/info, Stand 29. Juni 2016
[45] http://handlebarsjs.com/
[46] https://github.com/wycats/handlebars.js#differences-between-handlebarsjs-and-mustache
[47] https://github.com/dlmanning/browserify-handlebars

noch fehlenden Komponenten zusammen und evaluierte den Aufwand, sowie den vorraussichtlichen Nutzen der Realisation. Diese Ergebnisse werden in Tabelle 2 zusammengefasst. Die Skala der Bewertung des Aufwandes und des Nutzens reicht wieder, wie bereits bei Tabelle 1 auf Seite 20, von 0 (kein Aufwand/Nutzen) bis 5 (sehr hoher Aufwand/Nutzen).

Fehlende Komponente	Kurze Erläuterung	Anmerkung	geschätzter Aufwand	geschätzter Nutzen
StartSession-Wizard	Um eine neue Session zu initieren benötigt es einen Wizard in dem die geteilten Projekte, sowie die teilnehmenden Kontakte ausgewählt werden können.	Bereits angefangen, API komplett vorhanden	3	5
JoinSessionDialog	Erhält man eine Einladung zu einer Session, so benötigt es ein Dialogfenster, in dem man diese akzeptieren oder ablehnen kann.	API in Entwicklung	3	5
Chat	Ein Chatfenster, mit dem es möglich ist eine Nachricht an alle Session-Mitglieder gleichzeitig zu senden.		5	4
Configure-AccountsDialog[48]	Ein Einstellungsfenster in dem die Accounts, mit denen man sich einloggen kann, verwaltet werden können.		3	3
PreferencesDialog	Ein Dialog, in denen allgemeine Einstellungen beispielsweise bezüglich des Netzwerkes oder den visuellen Präferenzen getroffen werden können.		3	3

Tabelle 2: Auflistung und Evaluation der fehlenden Komponenten

Aufgrund dieser Einschätzung und vor allem des aktuellen Standes der in der API vorhandenen Funktionen entschied ich mich dafür, den bereits angefangenen *StartSessionWizard* zu vollenden. Im Anschluss daran plante ich, den gesamten Einladungsprozess zu komplettieren, indem ich den *JoinSessionWizard* genannt, implementierte. Als Ziel dieser Erweiterungen setzte ich mir, in der HTML-GUI einen vollständigen Session-Aufbau-Prozess tätigen zu können.

3.2.1 StartSessionWizard

Zur Vervollständigung des StartSessionWizard musste zunächst erörtert werden, was bereits vorhanden ist. Anders als von Sieker angegeben, war der Wizard zu Beginn meiner Arbeit noch nicht „fully functional“ (vgl. [Sie15], S. 39). Grundlegendes, wie zum Beispiel das Öffnen des Dialogfensters, sowie die Darstellung der *Wizard-Buttons* (*back*, *next*, *cancel* und *finish*), war zwar bereits vorhanden, jedoch fehlten noch die folgenden Punkte:

1. Funktionalität der *Wizard-Buttons*
2. Bezeichner an der Liste der auswählbaren Projekte
3. Liste der auswählbaren Kontakte in der entsprechenden Ansicht

Zuerst überprüfte ich, warum die Buttons in der Fußzeile des Wizards nicht funktionierten und ergründete, dass diese nicht korrekt von der Eltern-View, dem „AmpersandWizard“ vererbt wurden. Die missglückte Vererbung der *events* lag daran, dass der „StartSessionWizard“ nicht direkt mit `AmpersandView.extend` erweitert wird, sondern noch eben jene hierarchische Zwischenstufe des „AmpersandWizards“ genommen wird. Dieser wird nicht, wie der Name vermuten lässt, von *Ampersand.js* bereitgestellt, sondern ist eine selbst implementierte Vorlage[49]. Eine Vererbung zwischen zwei selbst implementierten Klassen erfolgt allerdings nicht automatisch, sondern muss explizit gemacht werden[50].

```
module.exports = WizardView.extend({
  // Inherit all events from the WizardView Prototype
  events: WizardView.prototype.events,
})
```

Des Weiteren war der JavaScript-seitige `sendInvitation`-Aufruf noch nicht fertiggestellt, so dass ein Klick auf den *finish*-Button nichts bewirkte. Diese Funktion wurde der *saros-api.js* noch hinzugefügt[51].

Im Anschluss daran untersuchte ich den Grund für die Absenz der Projekt- und Verzeichnisnamen in der entsprechenden Auswahl-Liste. Im Prinzip waren bereits alle nötigen Vorkehrungen getroffen worden, sodass diese eigentlich hätten angezeigt werden müssen. Es stellte sich heraus, dass es sich nur um eine Abweichung der Bezeichner im Backend beziehungsweise Frontend handelte. Unter *Java* wurden die Bezeichner `label` genannt, unter *JavaScript*,

[49] Um dieser Verwirrung vorzubeugen benannte ich den AmpersandWizard in WizardView um (http://saros-build.imp.fu-berlin.de/gerrit/#/c/3072)

[50] vgl. https://github.com/AmpersandJS/ampersand-view#events-ampersandviewextend-events---events-hash---

[51] http://saros-build.imp.fu-berlin.de/gerrit/#/c/3071

hießen sie allerdings `displayName`. Da die Kommunikation zwischen diesen beiden „Welten“ gleiche Bezeichner (zumindest an der jeweiligen Schnittstelle) vorraussetzt, scheiterte dies hier. Die Lösung war, im gesamten Frontend ebenfalls `label` zu verwenden[52]. Das Ergebnis ist in Abbildung 5 zu sehen.

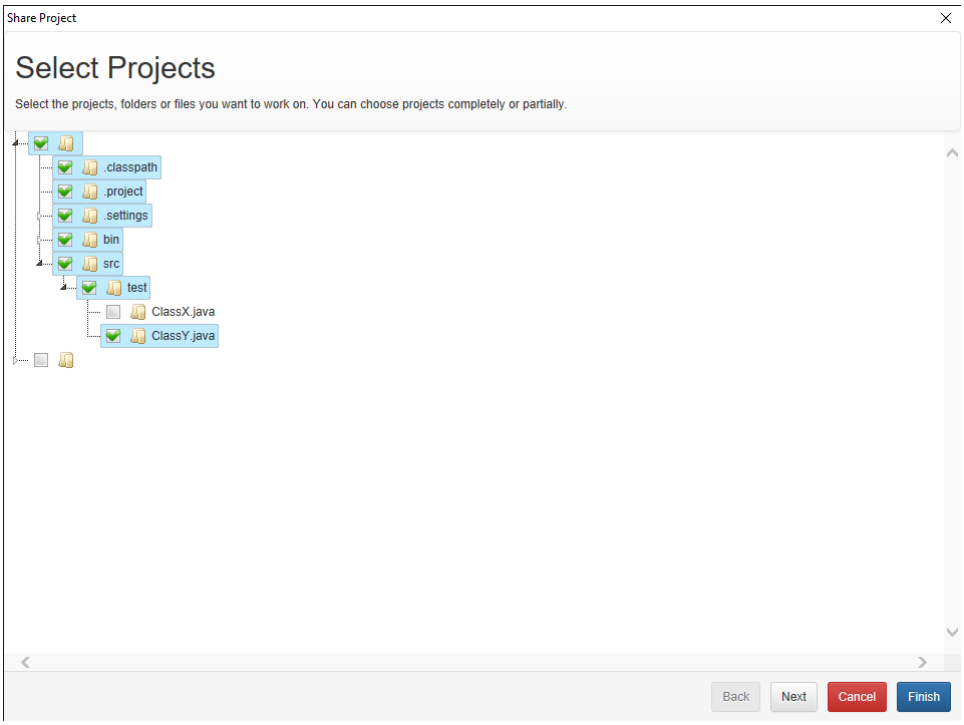

Abbildung 5: Ansicht der Projektauswahl im *StartSessionWizard*

Die Liste der auswählbaren Kontakte konnte nicht angezeigt werden, da innerhalb der `prepareView`-Funktion der Aufruf `app.state.contactList.getAvailable()` scheiterte. Diese Funktion hätte alle Kontakte ausgeben sollen, die derzeit *online* sind. Dies hat allerdings den Nachteil, dass bei keinen derzeit anwesenden Kontakten nur eine leere Seite angezeigt werden würde. Da auch keine weitere Nachricht angezeigt wird, die den Nutzer informiert, dass keiner seiner Kontakte derzeit online ist, ist davon auszugehen, dass dies Verwirrung und Unsicherheit bei dem Nutzer stiften würde. Daher entschied ich mich, diesen Ansatz nicht weiter zu verfolgen.

Stattdessen sollten immer alle Kontakte ausgegeben werden, auch wenn diese nicht *online* sind und somit auch keine Session mit ihnen zustande kommen soll. Diese derzeit nicht erreichbaren Kontakte werden allerdings grau unterlegt und können nicht angeklickt werden, sodass klar ist, dass sie nicht ausgewählt werden dürfen. Das hat zur Folge, dass immer Kontakte an-

[52] http://saros-build.imp.fu-berlin.de/gerrit/#/c/3066

gezeigt werden und der Nutzer mutmaßlich nicht ratlos vor einem weißen Bildschirm sitzt. Das Ergebnis[53] ist in Abbildung 6 zu sehen.

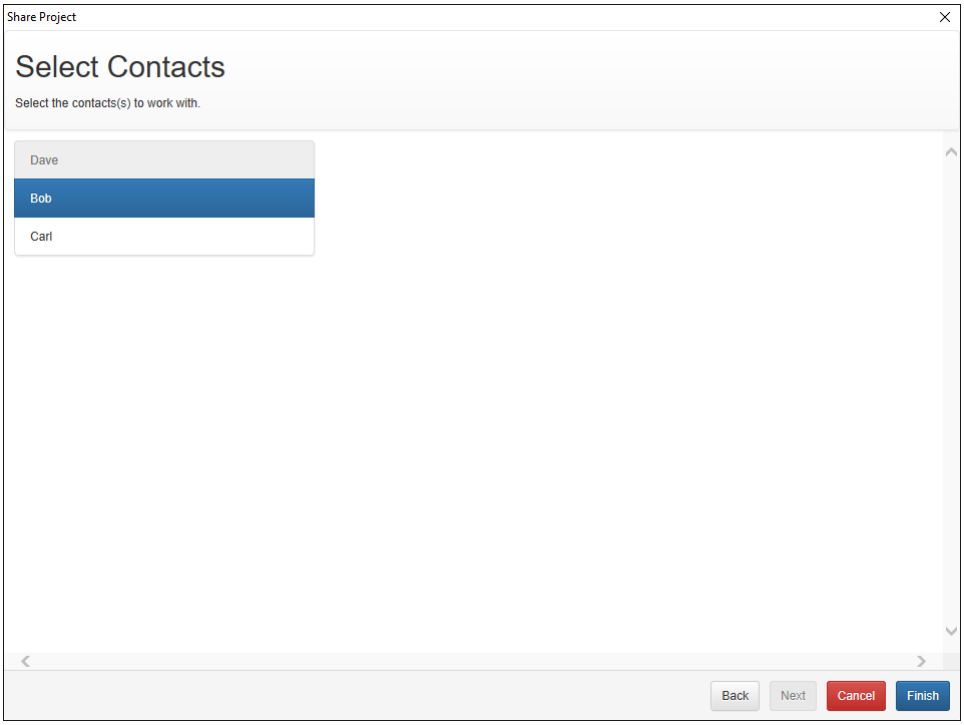

Abbildung 6: Ansicht der Kontaktauswahl im *StartSessionWizard*

3.2.2 JoinSessionWizard

Der *JoinSessionWizard* ist ein Dialogfenster, das bei einer ankommenden Session-Einladung automatisch geöffnet wird. Zunächst wird der Nutzer gefragt, ob er die Einladung annehmen oder ablehnen möchte. Akzeptiert er sie, so schließt sich in der alten GUI dieser Dialog und ein neuer, der *SessionNegotiationWizard* wird geöffnet. In diesem wird nach einem lokalen Speicherort für die von dem Session-Host geteilten Dateien gefragt.

Häufig ist die Pause zwischen dem Schließen des einen und dem Öffnen des anderen Fensters allerdings so lang, dass es einige Nutzer bereits zu der Annahme geführt hat, dass der Prozess abgebrochen wurde. Da dies natürlich keine erwünschte Reaktion ist, sollte die beschriebene Unterbrechung möglichst vermieden werden, was beispielsweise durch die Zusammenlegung der beiden Dialogfenster in einen gemeinsamen Wizard realisiert werden könnte.

[53] http://saros-build.imp.fu-berlin.de/gerrit/#/c/3071/

In der HTML-GUI soll aus diesem Grund bei einer Annahme der Einladung der *JoinSessionWizard* **nicht** geschlossen, sondern auf eine zweite Seite gesprungen werden, in der die Eingabe des gewünschten Speicherorts gefordert wird. Der *ProjectNegotiationWizard* wird nicht mehr benötigt.

4 Fazit

Das folgende Kapitel beschreibt zunächst eine rückblickende Einschätzung bezüglich der wichtigsten Entscheidungen, die ich im Laufe dieser Arbeit getroffen habe. Anschließend wird der Beitrag, den diese Arbeit insgesamt geleistet hat, in einen größeren Kontext gesetzt und bewertet.

4.1 Rückblickende Bewertung der getroffenen Entscheidungen

Die wichtigen Entscheidungen dieser Arbeit werden in der entsprechenden zeitlichen Reihenfolge im Folgenden aufgezählt. Es wird jeweils kurz erläutert, welche Gründe zu meiner jeweiligen Wahl führten und ob sie auch rückblickend betrachtet noch als gut zu bewerten ist.

1. **Änderung des Ziels**

   Abschnitt 2 auf Seite 6

   Anstatt die bestehenden Funktionalitäten zu erweitern, änderte ich meinen Fokus auf die Verbesserung der Arbeit als Entwickler. Der Grund hierfür war die steile Lernkurve, die jeder nachfolgende Entwickler meistern müsste. Auch retrospektiv empfinde ich diese Entscheidung als sehr gut. Der Einstieg der nachfolgenden Programmierer musste eindeutig erleichtert werden um ein Voranschreiten des HTML-GUI-Projekts gewährleisten zu können.

2. **Änderung des genutzten Frameworks**

   Abschnitt 2.2 auf Seite 10

   Das Verwerfen eines Großteils der bisherigen Arbeiten zu Gunsten eines neuen Frameworks war zum Zeitpunkt der Entscheidung insbesondere dadurch begründet, dass kaum gute Lern- und Übungsmaterialien auffindbar waren. Die bestehende Dokumentation war eher gering und deckte ausschließlich Standardprobleme ab. Sobald eine spezielleren Fragestellung vorlag, konnte darauf nur schwerlich eine Antwort gefunden werden.

   Rückblickend betrachtet war dieser Beschluss nicht besonders nützlich. Insbesondere die nachfolgende Revidierung dieser Entscheidung (vgl. Punkt 3) sollte dies kenntlich machen. Allerdings würde ich vermutlich auch mit dem dazu erworbenen Erfahrungsschatz auf Basis der damals geltenden Umstände dieselbe Entscheidung erneut treffen.

3. **Keine Änderung des genutzten Frameworks**

(Abschnitt 2.6 auf Seite 17)

Die Entscheidung *Ampersand.js* weiter zu benutzen, gründete insbesondere darauf, dass kein so „klobiges“ Framework, wie *Angular 2* benutzt werden sollte. Dass der Gedanke der Benutzung von *Angular 2* schnell wieder verworfen wurde, lag wiederum insbesondere an dem benötigten Build-Schritt, der vermieden werden sollte. Jedoch konnte dafür im Zuge der Verbesserung der Version mit *Ampersand.js* eine Lösung gefunden werden, die mit Sicherheit auch für *Angular 2* hätte gefunden werden können.

Daher würde ich insbesondere das Verwerfen von *Angular 2* als etwas voreilig bezeichnen und wüsste nicht, ob ich in einer ähnlichen Situation heute nicht anders entscheiden würde. Es ist allerdings anzumerken, dass in dem Augenblick dieser Entscheidung bereits die Idee der Verwendung von *Vanilla JS* entstanden war, die insbesondere innerhalb meines Antrittsvortrages am Fachbereich großen Anklang fand. Dies wurde insbesondere durch den Gedanken begründet, dass nachfolgende Programmierer ausschließlich Kenntnisse in purem JavaScript bräuchten, den ich auch nach wie vor noch als sehr gut bewerte.

4. **Umgestaltung der Ordnerstruktur**

   Der Wechsel von einer Typen- zu einer Feature-orientierten Ordner-Struktur war durch die Schwierigkeit bei der Suche nach den zusammengehörigen Dateien begründet. Diese Entscheidung ist auch im Rückblick nicht falsch gewesen. Es konnten leider aufgrund von zeitlichem Mangel keine konkreten Daten erhoben werden, ob die Umstrukturierung tatsächlich den gewünschten Effekt, gesuchte Dateien schneller zu finden, brachte. Jedoch bin ich nach wie vor davon überzeugt, dass diese Struktur dem allgemeinen Verständnis der Anwendung zuträglich ist.

4.2 Ergebnis

Im Ergebnis hat diese Arbeit ihr Ziel, die Erleichterung des Einstiegs in die Entwicklung der HTML-GUI, sowie die Erweiterung derselbigen,grundlegend erreicht. Einige Aspekte wurden entsprechend der in Abschnitt 3 auf Seite 19 vorgestellten Vorgehensweise noch nicht umgesetzt. Dies wird allerdings noch, wie in Abschnitt 5.3 auf Seite 39 beschrieben, erfolgen und somit in die folgende Ergebnisbewertung mit einbezogen.

Alles in allem stellen die vorgetragenen Änderungen, meiner Ansicht nach, auf jeden Fall eine Verbesserung zu dem vormalig bestehenden System dar. Leider konnte aufgrund des knappen Zeitfensters, das für die Bearbeitung

vorgesehen ist, keine Feldstudie oder ähnliches durchgeführt werden, um meine getroffenen Entscheidungen in ihrer Güte evaluieren zu können. Interessant wäre hierbei insbesondere eine Gegenüberstellung der benötigten Zeit zur Einarbeitung, sowie der dabei entstandenen Probleme und wie sie gelöst werden konnten unter der Verwendung der von mir vorgestellten Änderungen im Vergleich zu dem vormalig bestehenden System.

4.3 Arbeit im Saros-Team

Die Arbeitsweise des Saros-Teams habe ich insgesamt als gut empfunden. Es wurde sich gegenseitig unterstützt so gut es ging und bei aufkommenden Fragen, wurde gemeinsam versucht eine Lösung zu finden. Dazu waren insbesondere die zweimal wöchentlich stattfindenden Treffen sehr förderlich, bei denen jeweils ein kurzer Zwischenstandbericht der zuletzt geleisteten und als nächstes geplanten Arbeit erstattet wurde.

Thematisch gab es zu den anderen Arbeiten leider nur wenig Berührungspunkte. Lediglich ein Teil einer anderen Arbeit ([Sun16]) war für meinen Arbeitsbereich annähernd relevant. Aufgrund der theoretisch notwendigen, immensen Einarbeitungszeit in arbeitsfremde Themengebiete, war die gegenseitige Unterstützung bei bereichsspezifischen Problemen während der gesamten Arbeitszeit leider nur selten gegeben.

5 Ausblick

Im Folgenden werden alle in der HTML-GUI zum Zeitpunkt der Abgabe dieser Arbeit fehlenden oder fehlerhaften Komponenten aufgelistet. Dabei wird zwischen den in der alten GUI vorhanden, aber in der HTML-GUI noch nicht implementierten Features und derzeit noch bestehenden Problemen, die für eine gute *User-Experience* behoben werden müssen, unterschieden.

Im Anschluss daran wird aufgezählt, welche dieser Punkte noch von mir entwickelt beziehungsweise behoben werden, und die weitere geplante Vorgehensweise wird erläutert.

5.1 Fehlende Features

#1 Diverse Komponenten

Bis auf den *StartSessionWizard* (s. Abschnitt 3.2.1 auf Seite 28), sowie den *JoinSessionWizard* (s. Abschnitt 3.2.2 auf Seite 30 und Punkt **#2**) fehlen noch alle in Tabelle 2 auf Seite 27 aufgeführten Komponenten. In Zusammenhang mit dem *ConfigureAccountsDialog* sei zu erwähnen, dass eine reduzierte Variante - der *AddAccountDialog* - zum Hinzufügen eines Nutzerkontos ebenfalls noch nicht vorhanden ist.

#2 *JoinSessionWizard*

Da die Schnittstelle für das Öffnen beziehungsweise Schließen des Wizards, sowie die Weiterleitung der Nutzereingaben an das Backend sich zum Zeitpunkt der Abgabe noch in der Entwicklung befand[54], konnte auch der in Abschnitt 3.2.2 auf Seite 30 Wizard noch nicht fertig implementiert werden.

#3 Template-Engine

Der in Abschnitt 3.1.3 auf Seite 25 begründete Austausch des Template-Engines *Jade* zu Gunsten von *Handlebars* wurde bisher noch nicht umgesetzt.

#4 Dokumentation

Abgesehen von den anfänglich erörterten Problemen, sollte zudem die bestehende Dokumentation[55] erweitert werden, um den nachfolgenden

[54] vgl. http://saros-build.imp.fu-berlin.de/gerrit/#/c/3019/
[55] http://www.saros-project.org/html-gui

Entwicklern den Einstieg zu erleichtern. Außerdem muss sie aufgrund der Umstrukturierung der Dateien (Abschnitt 3.1.1 auf Seite 20) und dem hinzugefügten „Watch“-Skript (Abschnitt 3.1.2 auf Seite 22) entsprechend ergänzt werden.

#5 Diverse Buttons

Diverse kleinere Knöpfe, wie beispielsweise der *Enter follow mode*-Button, mit dem die Eingabe eines anderer Nutzers fokussiert werden kann oder auch der *Stop Session*-Button wurden bisher noch nicht implementiert. Diese erfordern, meiner Einschätzung nach, keinen besonders großen Frontend-seitigen Arbeitsaufwand, müssen aber zunächst noch in der Schnittstelle implementiert werden.

5.2 Bestehende Probleme

#6 Bei bestehender Session wird das Starten einer weiteren im Frontend nicht unterbunden beziehungsweise nicht davor gewarnt

Nach dem Starten einer Session ist es weiterhin möglich auf den *Start Session*-Knopf und im sich öffnenden *StartSessionWizard* auf *finish* zu klicken. Dies wirft im Backend die Warnung `(SarosSessionManager.java :218)could not start a new session because a session has already been started` aus, die allerdings nicht durch eine Meldung für den Nutzer sichtbar gemacht wird. Die beste Lösung für dieses Problem wäre es, den *Start Session*-Knopf bei einer laufenden Sitzung in einen *Stop Session*-Button umzuwandeln. Damit wäre zudem eine weitere, in Punkt **#5** erwähnte, Teilkomponente implementiert.

#7 Session lässt sich ohne Projekte oder Kontakte starten

Der *finish*-Button des *StartSessionWizards* lässt sich auswählen, auch wenn keine Projekte oder Kontakte ausgewählt wurden. Die Session startet dann zwar nicht, da dieser Fall im Backend abgefangen wird, der Nutzer erhält darüber jedoch keine Auskunft in Form einer Fehlermeldung.

#8 Fehlerhafte Accountauswahl

Bei der Auswahl eines Accounts, mit dem der Nutzer sich anmelden könnte werden teilweise Konten angezeigt, mit denen dieser sich noch nie angemeldet hat. Dies ist zum einen in Bezug auf die alte GUI inkonsistent (siehe Abbildung 7 auf der nächsten Seite), aber vor allem

führt die Auswahl eines fälschlicherweise aufgeführten Accounts zu einer `IllegalArgumentException` (siehe Listing 1 auf der nächsten Seite).

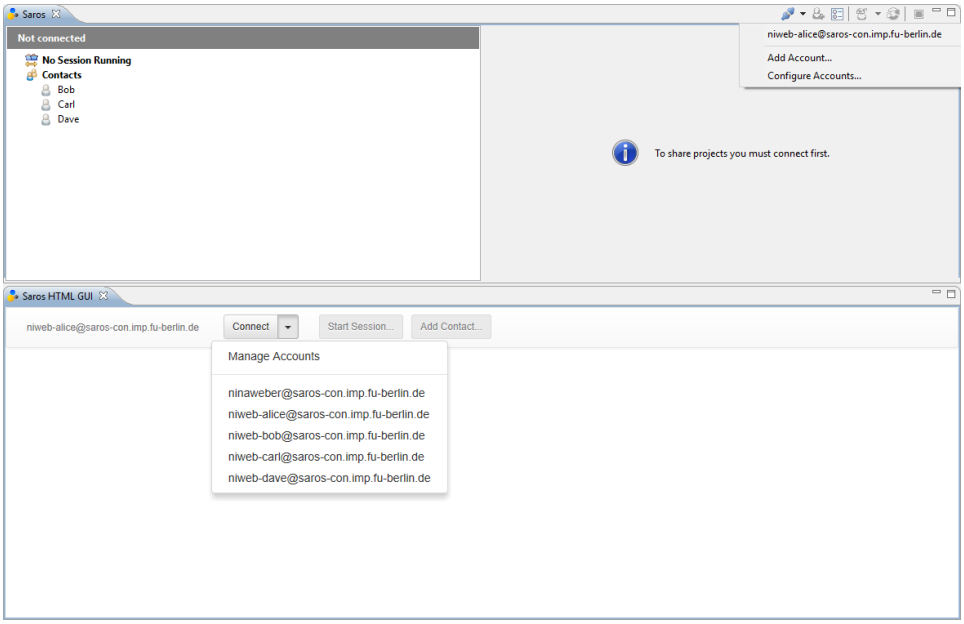

Abbildung 7: Fehlerhafte Accountauswahl

```
ERROR 11:36:28,448 [BrowserFunction-connect] (
    TypedJavascriptFunction.java:214) connect
    threw an exception, args: [de.fu_berlin.inf.
    dpp.ui.model.Account@f3fa3c]
java.lang.IllegalArgumentException: account 'null'
     is not in the current account store
at de.fu_berlin.inf.dpp.account.XMPPAccountStore.
    setAccountActive(XMPPAccountStore.java:341)
at de.fu_berlin.inf.dpp.ui.core_facades.
    StateFacade.connect(StateFacade.java:52)
at de.fu_berlin.inf.dpp.ui.browser_functions.
    ConnectAccount.connect(ConnectAccount.java:35)
at sun.reflect.NativeMethodAccessorImpl.invoke0(
    Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(
    NativeMethodAccessorImpl.java:57)
at sun.reflect.DelegatingMethodAccessorImpl.invoke
    (DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java
    :606)
at de.fu_berlin.inf.dpp.ui.browser_functions.
    TypedJavascriptFunction$1.run(
    TypedJavascriptFunction.java:158)
at de.fu_berlin.inf.dpp.util.ThreadUtils$1.run(
    ThreadUtils.java:38)
at java.lang.Thread.run(Thread.java:745)
```

Listing 1: Fehlermeldung bei Auswahl eines fälschlicherweise aufgeführten Kontaktes (siehe Punkt #8 auf Seite 36)

#9 Kontaktliste in der Hauptansicht verschwindet teilweise

Wird die Breite der Saros-View auf unter 768 Pixel verringert, so rutscht die auf der *MainPage* vorhandene Kontaktliste unter die Menüleiste (siehe Abbildung 8 auf der nächsten Seite). Dies verschlechtert nicht nur die *User-Experience* aufgrund der verminderten Ästhetik, sondern schränkt auch die Funktionalität dahingehend ein, dass die Kontaktliste durch Klick auf den stilisierten, nach oben zeigenden Pfeil nicht mehr eingeklappt werden kann.

#10 Projektname wird nicht angezeigt

Bei der Projekt-Auswahl im *StartSessionWizard* werden zwar die Namen aller in einem Projekt enthaltenen Verzeichnisse angezeigt, jedoch nicht der Name des gesamten Projekts.

#11 Automatische Auswahl der Unterordner

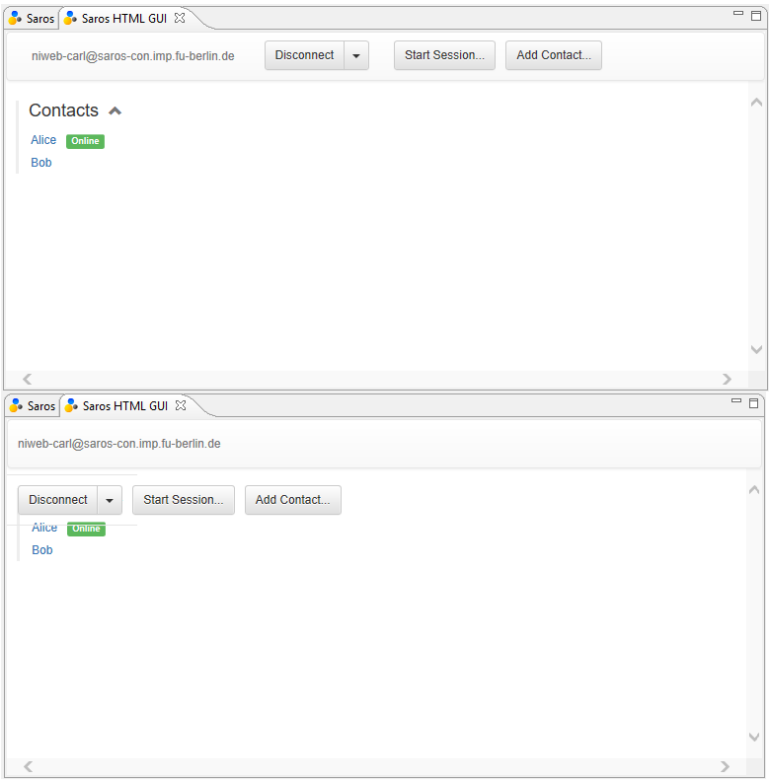

Abbildung 8: Verschwindende Kontaktliste

Wird ein Projekt beziehungsweise ein Ordner innerhalb eines Projekts beim Starten einer neuen Session aus- oder abgewählt, so wird entgegen der anzunehmenden Erwartung des Nutzers nicht die Auswahl aller Unterordner entsprechend aktualisiert.

5.3 Geplante Umsetzung

Für die nachfolgenden Punkte wurde bereits eine konkrete Lösungsstrategie ausgearbeitet, die jeweils kurz erläutert wird. Die Umsetzung erfolgt nach Einreichung dieser Arbeit.

#2 Der *JoinSessionWizard* wird nach Beendigung der Schnittstellenimplementierungen entsprechend der in Abschnitt 3.2.2 auf Seite 30 getätigten Entscheidungen fertig entwickelt werden.

#3 Mithilfe des Moduls *jade-to-handlebars*[56], sollten die vorhandenen *Ja-*

[56] https://www.npmjs.com/package/jade-to-handlebars

de-Templates in verhältnismäßig kurzer Zeit in entsprechende *Handlebars*-Templates umgewandelt werden können. Die notwendigen Aktualisierungen der Referenzen in den *JavaScript*-Dateien müssen allerdings manuell getätigt werden.

Es wird mit keinen Komplikationen gerechnet, dennoch wird dieser Arbeitsschritt aufgrund der händischen Neureferenzierung von etwas längerer Dauer sein.

#4 Die Dokumentation wird, wie in Punkt **#4** auf Seite 35 geschildert, aktualisiert und erweitert. Dies wird einige Zeit in Anspruch nehmen, ist aber für eine weiterführende Entwicklung unabdingbar.

#7 Zunächst ist zu klären, ob vor dem Starten einer „leeren“ Session frontend-seitig gewarnt werden soll oder dies sogar ganz zu unterbinden ist. Hierzu erfolgt zunächst eine Abstimmung mit dem Saros-Team.

Soll diese Option vollständig unterbunden werden, so ist die einfachste Lösung das Deaktivieren des *finish*-Buttons, solange nicht mindestens ein Artefakt und ein Kontakt ausgewählt wurde.

Wenn es prinzipiell weiterhin möglich sein sollte ohne ein ausgewähltes Projekt oder einen Kontakt eine neue Session zu starten, so benötigt es dieselbe Validierung. Der *finish*-Button sollte dann allerdings nicht blockiert, sondern bei einem Klick, im entsprechenden Fall, lediglich eine Warnung ausgegeben werden. Diese könnte entweder im selben oder in einem neuen Dialogfenster erscheinen. Diese Entscheidung wird in diesem Fall erneut unter Absprache mit dem Projekt-Team erfolgen.

Die Umsetzung sollte, egal wie die Entscheidungen ausfallen, nicht zu viel Zeit beanspruchen, da sowohl die dazu benötigte Validierung bereits angefangen, als auch das eventuell erforderliche Dialogfenster zur Ausgabe des Warnhinweises bereits grundlegend implementiert wurde.

#9 Das Verschwinden der Kontaktliste ist durch das Verrutschen der Button-Gruppe aus der Navigationsleiste heraus begründet. Dies ist mit großer Wahrscheinlichkeit lediglich der Effekt einer nicht ordnungsgemäßen Strukturierung, die sich zwar in den *Jade*-Templates nicht besonders gut herauslesen lässt, nach der Einführung von *Handlebars* (siehe Punkt 5.3 auf der vorherigen Seite) aber mutmaßlich leicht ersichtlich und dementsprechend schnell korrigierbar sein sollte.

Literatur

[Aga02] David J. Agans. *Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems*. 2002.

[Boh15] Matthias Bohnstedt. „Entwicklung einer IDE-unabhängigen Benutzeroberfläche für Saros“. Masterarbeit. Freie Universität Berlin, 2015.

[Cik15] Christian Cikryt. „Evaluating the Use of a Web Browser to Unify GUI Development for IDE Plug-ins“. Masterarbeit. Freie Universität Berlin, 2015.

[Fre10] Regis Frey. *MVC-Process*. 11. Mai 2010. URL: https://commons.wikimedia.org/wiki/File:MVC-Process.svg (besucht am 24. 06. 2016).

[Gec16] Minko Gechev. *Building an Angular 2 Application for Production*. 26. Juni 2016. URL: http://blog.mgechev.com/2016/06/26/tree-shaking-angular2-production-build-rollup-javascript/ (besucht am 07. 07. 2016).

[Gor14] Alex Gorbatchev. „Comparing JavaScript Templating Engines: Jade, Mustache, Dust and More“. In: *StrongLoop* (11. Nov. 2014). URL: https://strongloop.com/strongblog/compare-javascript-templates-jade-mustache-dust/.

[Hal94] Wulf R. Halbach. *Interfaces: Medien- und kommunikationstheoretische Elemente einer Interface-Theorie*. 1994.

[HT99] Andrew Hunt und David Thomas. *The Pragmatic Programmer. From Journeyman to Master*. 1999. Kap. The Evils of Duplication, S. 26.

[Mül08] Matthias Müller. *Analyse leichtgewichtiger Softwareentwicklungsmethoden*. 2008.

[Net05] Mozilla Developer Network. *XMLHttpRequest*. 4. Aug. 2005. URL: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest (besucht am 03. 07. 2016).

[Osa12] Addy Osami. *Journey Through The JavaScript MVC Jungle*. 27. Juli 2012. URL: https://www.smashingmagazine.com/2012/07/journey-through-the-javascript-mvc-jungle/ (besucht am 24. 06. 2016).

[Pap14] John Papa. *Angular 1 Style Guide*. 2014. URL: https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md (besucht am 02. 06. 2016).

[Sie15] Bastian Sieker. „User-Centered Development of a JavaScript and HTML-based GUI for Saros“. Masterarbeit. Universität Paderborn, 2015.

[Sun16] David Sungaila. „Verbesserung und Erweiterung der Core-Bestandteile von Saros“. 15. Apr. 2016.

[W3S] W3Schools. *AJAX Create an XMLHttpRequest Object*. URL: http://www.w3schools.com/ajax/ajax_xmlhttprequest_create.asp (besucht am 03. 07. 2016).

[Wel99] Don Wells. *Extreme Programming: A gentle introduction*. 1999. URL: http://www.extremeprogramming.org/ (besucht am 24. 06. 2016).