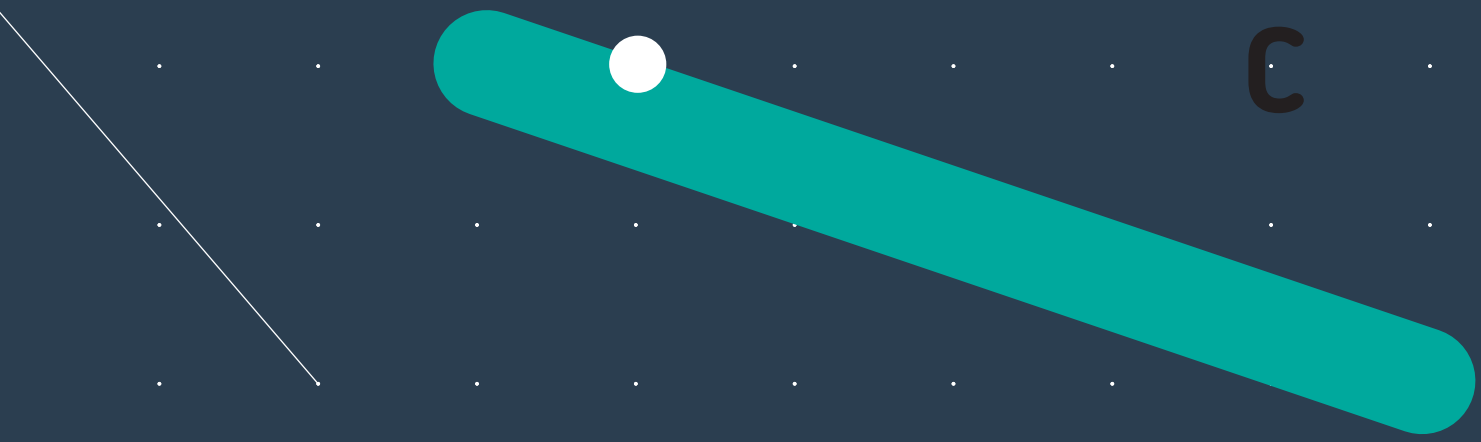
Hochschule Konstanz
Department of Computer Science

Autor
Jonas Reinwald

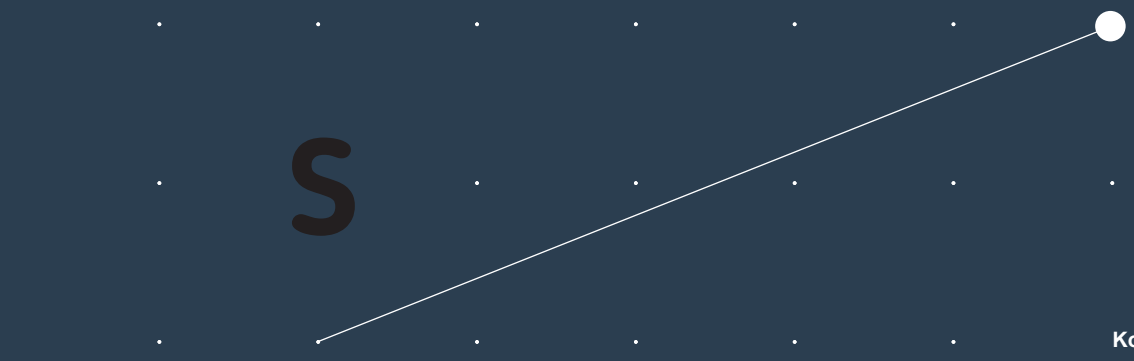
Immatrikulationsnummer
293320

E-Mail
jonas.reinwald@htwg-konstanz.de



Bachelorthesis

Konzipierung der Architektur einer Darstellungsschicht basierend auf aktuellen Webtechnologien im Hinblick auf eine zukünftige Anbindung an das Programm combit Relationship Manager



Konstanz, 10. Februar 2019

Hochschule Konstanz
Department of Computer Science

Bachelorthesis

Konzipierung der Architektur einer Darstellungsschicht basierend auf aktuellen Webtechnologien im Hinblick auf eine zukünftige Anbindung an das Programm combit Relationship Manager

von

Jonas Reinwald

Zur Erlangung des akademischen Grades eines

Bachelor of Science (B.Sc.)

an der Hochschule Konstanz für Technik, Wirtschaft und Gestaltung

Vorgelegt von:	Jonas Reinwald Wessenbergstr. 13 78462 Konstanz 293320
Erstbetreuer:	**Prof. Dr. Marko Boger**
Zweitbetreuer:	**Dipl.-Inf. Alexander Horak** combit GmbH Untere Laube 30 78462 Konstanz
Ausgegeben am:	01.12.2018
Eingereicht am:	28.02.2019

Zusammenfassung (Abstract)

Webapplikationen in Form von mobilen Apps, Single-Page-Applikationen, … FIXME: beispiel machen heute bereits einen sehr großen Teil häufig genutzter Software aus. In Zukunft wird der Anteil von webbasierter Software im Vergleich zu traditioneller Software noch weiter ansteigen (FIXME: quelle?). Auch für Unternehmen ist es langfristig wichtig diesem Trend zu folgen um Kunden flexiblere Softwarelösungen anbieten zu können und nicht den Eindruck zu erwecken man könne nicht mit modernen Entwicklungen mithalten (FIXME: formulierung). Die Firma combit GmbH entwickelt mit dem combit Relationship Manager eine Software für das Management von Kundenbeziehungen (Customer Relationship Management), die zum momentanen Zeitpunkt aus einer Desktopapplikation, einer Webapplikation und einer mobile Webapplikation besteht, welche unabhängig voneinander entwickelt und benutzt werden. Im Rahmen dieser Bachelorthesis soll ein Architekturkonzept für eine einheitliche Oberfläche (ThinClient FIXME: footnote?) basierend auf moderner Webtechnologie erarbeitet werden, mithilfe derer die drei momentanen Ausführungen des combit Relationship Managers abgelöst werden können.

Zur Erstellung dieses Konzepts beschäftigt sich diese Arbeit unter anderem mit der Evaluierung eines geeigneten Frontend-Frameworks bzw. einer Frontend-Bibliothek, der Kommunikation zwischen neuem UI-Layer und Backend und der Unterstützung aller bisherigen Features der Desktopapplikation. Zwei besonders spannende Probleme existieren hier zum einen in der Flexibilität der momentanen Oberfläche, welche von jedem Nutzer individuell anpassbar ist. Diese Flexibilität soll durch eine automatische Konvertierung zu einem von der neuen UI darstellbarem Format und weiterhin anpassbarem Format erhalten bleiben. Zum anderen bietet die Desktopapplikation die Möglichkeit Skripte und automatisierte Aufgaben zur Anpassung der zugrunde liegenden Daten, für Darstellungsbedingungen und weitere Zwecke direkt auf dem Client auszuführen. Das Ausführen auf dem Client ist bei Webapplikationen nicht ohne Weiteres möglich, weshalb eine andere Lösung für diese Funktionalität gefunden werden muss.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1

Einleitung

1.1. Motivation

Ein Großteil der heute genutzten Software kann über eine Weboberfläche angesprochen werden. Dies hat den Vorteil, dass keine langwierigen und komplizierten Installationen vorgenommen werden müssen und, eine bestehende Internetverbindung vorausgesetzt, die Applikation von überall erreichbar ist. Die Firma combit GmbH hat diese Entwicklung bereits früh erkannt und bietet für den combit Relationship Manager eine entsprechende Weboberfläche an. Diese existiert bereits seit etlichen Jahren und basiert, bedingt durch den damaligen Stand der Technik auf nicht mehr zeitgemäßen Technologien. Da der Fokus der Entwicklung – bestimmt durch die Nutzung der Kunden – auf der Desktopanwendung liegt waren auch nicht genug Ressourcen vorhanden um die Technologien nach und nach durch modernere Alternativen zu ersetzen. Webtechnologien haben außerdem nicht nur Vorteile für ihre Nutzer, sie bringen auch einige interessante Aspekte für das Entwicklerteam mit sich. Sehr viele technische Fortschritte fanden in den letzten Jahren in diesem Bereich statt, was einem nicht nur ermöglicht moderne Projekte und Konzepte kennen zu lernen, sondern auch, dass man eine breite Auswahl für eine solide Basis des eigenen Produkts hat. Ein spezielles Beispiel für einen solchen Fortschritt sind Programme, die es einem erlauben Webseiten wie normale Programme nativ auf einem Desktopcomputer darstellen zu können. Das Ziel hierbei ist, langfristig nur noch eine einzige Version für alle Umgebungen entwickeln zu müssen, wodurch Ressourcen gespart werden können und durch den einheitlichen Fokus, so die Erwartung, ein besseres Produkt entsteht. Dieses Potential der Webtechnologien wird dadurch noch weiter gesteigert, dass durch einfache Sprachen wie HTML, CSS

und JavaScript, welches nicht kompiliert werden muss und auf so gut wie jedem modernen Gerät lauffähig ist, schnelle Iterationsschritte von Software möglich sind. Wegen der genannten Gründe wurde entschieden, das Thema Web mit einer frischen Herangehensweise, einer neuen Perspektive und modernen Technologien neu anzugehen um den combit Relationship Manager sowohl extern als auch intern noch attraktiver zu machen.

1.2. Erwartete Schwierigkeiten

Bereits im Vorfeld soll ein kleiner Überblick über die Bereiche dieser Arbeit gegeben werden, welche im Laufe der Umsetzung zu Schwierigkeiten führen könnten. Dies dient sowohl der Kontrolle der eigenen Fähigkeit Probleme bereits im Voraus korrekt einschätzen zu können, als auch dazu Einschätzungen von Problemstellen zukünftiger Projekte zu verbessern indem am Ende der Arbeit mit den tatsächlichen problematischen Stellen verglichen wird. Weitere Informationen zu den hier aufgelisteten Punkten werden in Kapitel 2 aufgeführt.

Eine der größten Herausforderungen wird sein, eine Umsetzung des User Interface zu finden, welche auch weiterhin genauso oder zumindest annähernd anpassbar ist wie dies momentan der Fall ist. Da das Layout dateibasiert gespeichert wird und im Browser nicht auf diese Dateien zugegriffen werden kann muss ein neues Konzept für die Persistierung erarbeitet werden, etwa die Speicherung in einer Datenbank, mithilfe dessen die Verteilung über das Netzwerk einfacher möglich ist. Neben dem angesprochenen Layout gibt es auch noch weitere Informationen die momentan nur dateibasiert auf dem jeweiligen Clientrechner gespeichert sind, diese müssen ebenfalls alle in ein neues Konzept überführt werden.

Um auf alle diese Informationen und die eigentlichen Daten, die dem Nutzer angezeigt werden sollen, zugreifen zu können wird eine entsprechende Schnittstelle (API) benötigt. Das Design dieser API ist kritisch, da nachfolgend nötige Änderungen sich auf jede Komponente auswirkt und ebenso Änderungen an diesen erfordert.

Ebenfalls gilt es zu entscheiden, welche Features im Web überhaupt nicht oder nur mit zu vielen Schwierigkeiten, etwa Scripting-Unterstützung das auf eine Desktopumgebung ausgelegt ist oder die Interaktion mit anderen auf dem System laufenden Prozessen, umsetzbar ist und diese bei der Erstellung der Anforderungen entsprechend zu beachten. Hier muss abgewogen werden, wie stark der Nutzer im Vergleich zu bisher eingeschränkt wird und ob die Nutzung der neuen Oberfläche für ihn damit überhaupt attraktiv genug erscheint.

1.3. Kapitelübersicht

In Kapitel 2 werden Anforderungen (auch anhand bisheriger Features) ermittelt. In Kapitel 3 werden die in Betracht gezogenen Technologien und deren Zusammenspiel untereinander vorgestellt. Dazu gehören die zu nutzende Programmierumgebung (Sprache, Testframework, Continuous Integration), der Vergleich verschiedener Frontend-Frameworks die anhand der in Kapitel 2 formulierten Anforderungen bewertet werden und die Protokolle zur Kommunikation mit dem Backend. In Kapitel 4 wird die Ausarbeitung des Konzepts mit den in Kapitel 3 ausgesuchten Technologien und der Aufbau der API zur Kommunikation mit dem Backend beschrieben. In Kapitel 5 wird der Entwicklungsprozess der beispielhaften Umsetzung des erstellten Konzepts dargestellt. In Kapitel 6 wird das Ergebnis kritisch hinterfragt, die tatsächlich aufgetretenen Schwierigkeiten mit den zuvor Erwarteten verglichen und ein Fazit gezogen.

2

Anforderungsanalyse

2.1. Produktüberblick

Um die Anforderungen erstellen und verstehen zu können muss zunächst einmal der Ausganszustand, also der jetzige Desktopclient des cRM (Eigenschreibweise für das Customer Relationship Management (CRM) Produkt der Firma combit GmbH) betrachtet werden. Wie am Begriff CRM bereits zu erkennen handelt es sich dabei um ein Programm das der Verwaltung und Vernetzung von Kunden- und Kontaktdaten dient. Neben den typischen Einsatzbereichen von CRM-Produkten wie die erwähnte Verwaltung von Kundendaten, Terminen und Aufgaben legt die combit GmbH allerdings viel Wert darauf, das Programm möglichst flexibel zu gestalten und damit viele weitere Einsatzbereiche von einer großen Anzahl verschiedener Kundengruppen zu ermöglichen. Einige Beispiele hierfür sind die mitgelieferten, fertigen Lösungen (im Kontext des cRM “Solutions” genannt) mit denen unter anderem Immobilien gehandhabt, die Vermittlung von Arbeitsstellen an Drittfirmen organisiert oder ein Ticket-System aufgebaut werden können. Zudem ist eine Musterlösung für den Datenschutz-Grundverordnung (DSGVO)-konformen Umgang mit all diesen Daten integriert. Unabhängig von diesen Lösungen stehen gewisse Funktionalitäten die jedoch auf den Daten einer Lösung aufbauen – Import und Export von Daten in viele verschiedene Formate, Versand von (Serien-)E-Mails oder etwa die Generierung von Berichten (Reports) durch ein zweites internes Produkt “List & Label” – immer zur Verfügung.

Eine Solution stellt die grundsätzliche Organisation der Daten in bestimmte Bereiche oder Szenarien dar und kann von jeder Firma individuell konfiguriert werden. Jede Solution ist durch eine Menge von Datenbanktabellen und darauf basierenden Ansichten zur

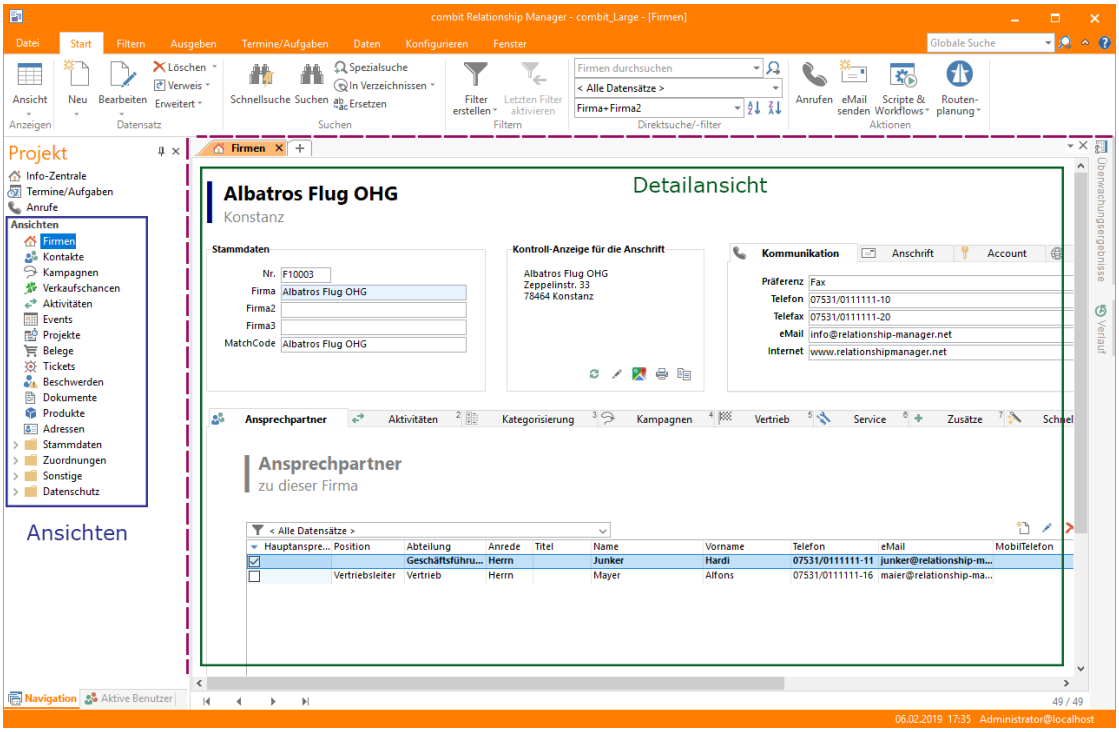


Abbildung 2.1: Grafische Oberfläche des cRM Desktopclient

Visualisierung der in den Tabellen enthaltenen Entitäten und Relationen definiert. Anhand der in Abbildung 2.1 gezeigten Standard-Solution “Large” ist links eine Liste der verfügbaren Ansichten und rechts die Detailansicht einer Entität der “Firmen”-Ansicht zu sehen. Außerdem wird hier auch die allgemeine Trennung zwischen Haupt-UI und ansichtsspezifischer UI ersichtlich. Anstelle der Detailansicht kann auch eine Listenansicht (tabellenförmige Auflistung aller Entitäten), eine Berichtsansicht, eine Webansicht (momentan mithilfe des Internet Explorer realisiert) und eine Kombinationsansicht, bei der sowohl die Übersichtsliste als auch die Details sichtbar sind, angezeigt werden.

Aus technischer Sicht ist der cRM in drei Schichten getrennt. Die zu verwaltenden Daten werden entweder in einer Microsoft SQL Server- oder einer PostgreSQL-Datenbank gespeichert. Eine Besonderheit hierbei ist die Tatsache dass Relationen zwischen Datenbanktabellen nicht durch Fremdschlüssel und zugehörige auf Datenbankebene sondern durch virtuelle Relationen zwischen Ansichten im C++-Kern der Anwendung, realisiert sind. Dies erlaubt einen flexibleren Umgang mit diesen und vereinfacht das Ändern von Datenbanktabelle, Ansichten und Relationen direkt aus der Oberfläche des cRM. Der cRM-Core bildet die zweite Schicht und ist für den Datenbankzugriff und die Business Logic FIXME: footnote verantwortlich. Diese beiden Schichten sollen von der Erstellung der neuen Weboberfläche möglichst unberührt bleiben. Einzig die API-Anbindung für die UI muss an dieser Stelle integriert werden – betrachtet wird

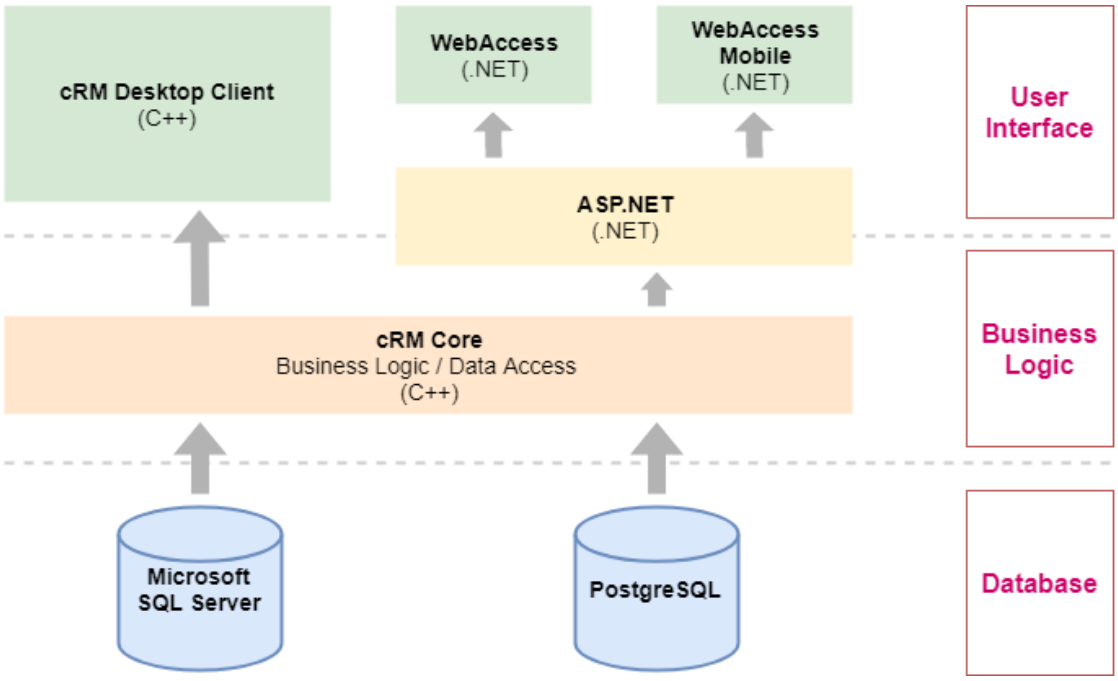


Abbildung 2.2: Aktueller technischer Aufbau des cRM

die API in dieser Arbeit jedoch nur aus Sicht der Oberfläche, die Integration im Backend findet zu einem späteren Zeitpunkt statt. Aufsetzend auf dem Kern existieren parallel der Desktopclient, ebenfalls in C++ geschrieben, der WebAccess für Desktopbrowser und der WebAccess Mobile für mobile Endgeräte. Zwischen Core und Webaccess (Mobile) gibt es noch einen mit ASP.NET FIXME: footnote? realisierten .NET-Wrapper, welcher den Zugriff auf Daten per HTTP ermöglicht.

Abbildung 2.2 zeigt eine Übersicht der momentanen Architektur, Abbildung 2.3 hingegen, wie die Architektur in Zukunft aufgebaut sein könnte.

2.2. Funktionale Anforderungen

In diesem Teil des Kapitels werden funktionale Anforderungen welche die Web-UI unterstützen soll aufgelistet und kurz erläutert.

2.2.1. combit Relationship Manager

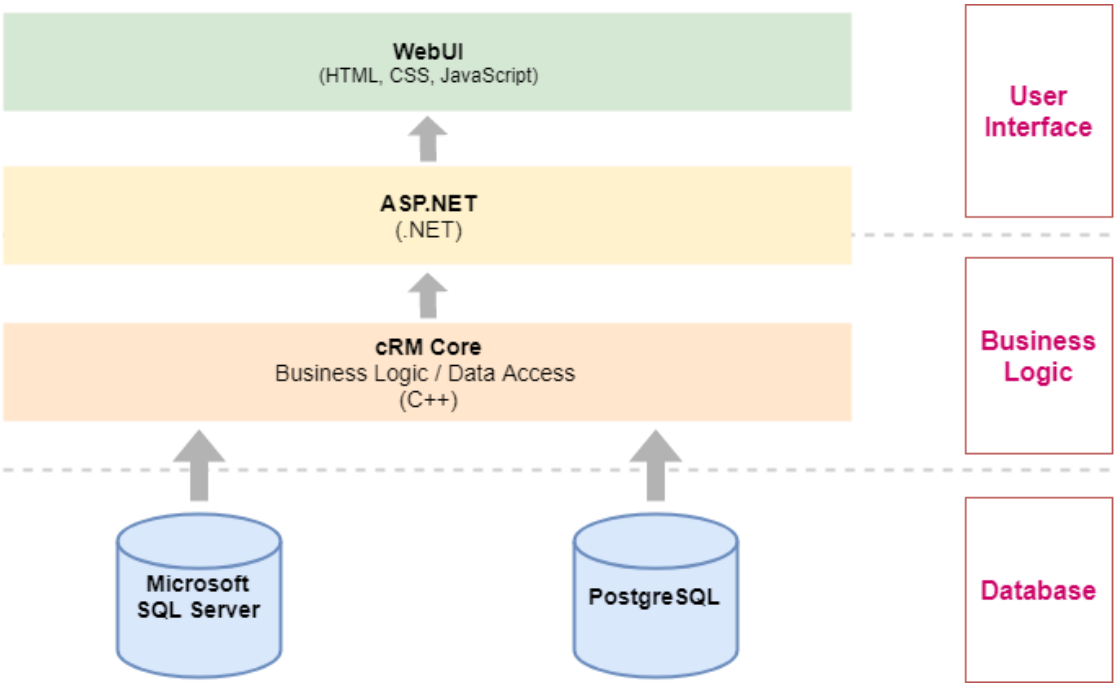


Abbildung 2.3: Möglicher zukünftiger technischer Aufbau des cRM

Automatisierte Erstellung

Die meisten Kunden haben entweder bereits ihre eigenen Ansichten erstellt oder benutzen angepasste Versionen der mitgelieferten Designs. Wenn das Ziel ist, langfristig die komplette UI-Schicht ins Web zu überführen muss dieser Prozess für die Benutzer leicht und zugänglich genug sein um nicht schon im Voraus abgelehnt zu werden. Es ist daher unumgänglich dass alle bisherigen Ansichten mit minimalem Aufwand seitens deren Nutzer im Web weiterverwendet werden können. Einige kleine Anpassungen sind tolerierbar, jede Ansicht neu zu erstellen jedoch nicht. Eines der Hauptziele dieser Arbeit ist es daher eine automatische Umwandlung bestehender Ansichten zu ermöglichen – hierfür muss die aktuelle UI-Spezifikation analysiert und eine neue Spezifikation erstellt werden, welche Implementierungen aus der vorherigen generieren kann.

Benutzerverwaltung

Die Benutzer und deren Rechte werden vollständig im Backend verwaltet. Es muss allerdings bei der Umsetzung der Konfigurationsoberfläche und bei der Umsetzung der Rechte besonders sorgfältig geprüft werden, ob weiterhin der volle Funktionsumfang gewährleistet ist. Sollten bestimmte Rechte für Benutzer oder Gruppen nicht mehr anpassbar sein oder fehlerhaft angewendet werden ist unter Umständen der komplette cRM nicht mehr vernünftig nutzbar.

Edit-Modus

Änderungen in der Detailansicht werden nicht direkt an die Datenbank übertragen, sondern nur als geändert markiert. Der Benutzer hat dann die Möglichkeit diese zu verwerfen oder den entsprechenden Datensatz zu speichern und die Informationen in die Datenbank zu schreiben.

Individualisierung

Die Individualisierung der beiden Hauptansichten für Datenentitäten, die Detail- und Listenansicht, ist zur effektiven Visualisierung bestimmter Sachverhältnisse ausschlaggebend. Bei der Listenansicht können beispielsweise Datenbankspalten ein- bzw. ausgeblendet und frei angeordnet werden, Spalten je nach Datentyp formatiert und spezielle Werte nach selbst definierten Regeln hervorgehoben werden. Die Detailansicht ist noch mächtiger, hier kann jedes UI-Element frei platziert werden, mit Sichtbarkeits- und Formatierungsbedingungen versehen und mit anderen Feldern verknüpft werden (Änderung in Feld A löst Änderung in Feld B aus). Um diese Einstellungen vorzunehmen existiert ein kompletter Eingabemaskendesigner welcher dem Benutzer per What You See Is What You Get (WYSIWYG)-Prinzip FIXME: footnote? das Designen erleichtert. Die Weboberfläche sollte diese Funktionalität möglichst vollständig abbilden.

Berichts- und Webansicht

Wie in der Übersicht erklärt werden Berichte von einem zweiten Produkt der Firma combit GmbH, List & Label, gehandhabt. Dieses besitzt bereits zum jetzigen Zeitpunkt einen entsprechende Weboberfläche die zukünftig für die Erstellung und das Anzeigen von Berichten genutzt werden kann. Auch die Webansicht kann weiterhin unterstützt werden indem darin anzuzeigende Ressourcen in ein iframe-Tag FIXME: footnote? eingebettet werden.

Suche / Filter / Sortierung

Suchen und Filter von Datensätzen können weiterhin mit bestehenden Techniken gelöst werden indem das Backend Daten vor dem Senden entsprechend im Vorfeld verarbeitet. Da es aber sinnvoll ist Sortierungen in der Übersichtsliste auf dem Client auszuführen – eine Garantie der API, dass Daten immer in einer entsprechenden Reihenfolge gesendet und auch empfangen werden schränkt das Design dieser zu sehr ein – ist es zumindest überlegenswert, ob Suchen und Filter ebenfalls zusätzlich direkt Clientseitig unterstützt werden sollen (viele JavaScript-Projekte bieten hierfür bereits fertige und effiziente Lösungen an).

Scripting

Die Desktopanwendung des cRM unterstützt momentan das Ausführen von Skripten in den Sprachen VBScript und C#. Für die programmatische Ansteuerung existiert ei-

ne Component Object Model (COM)-API welche im Kontext des Clients, auf dem der Prozess ausgeführt wird, benutzt werden kann. Diese Technologien sind in einer Browserumgebung nicht verfügbar, daher kann ein Skript zwar über die neue UI ausgelöst, aber nicht lokal sondern ausschließlich im Kontext des Servers ausgeführt werden.

Import / Export

Import und Export von Daten ist ein wichtiges Feature wenn es darum geht verschiedene Programme in einem Workflow zu vereinen. Der cRM unterstützt diese Funktion mit einer Vielzahl an Formaten (Excel, Outlook, Datenbanken (ODBC) und weitere). Diese Funktion kann wie viele andere auch weiterhin bestehen bleiben, es muss aber eine Möglichkeit geschaffen werden die Daten über das Netzwerk zur Verfügung zu stellen (Down- und Upload).

Terminverwaltung und Aufgabenplanung

Die interne Termin- und Aufgabenverwaltung ist analog zu anderen Daten über Datenbanktabellen realisiert, diese können ebenso vom Backend an die Web-UI übertragen und dort dargestellt werden. Der Desktopclient unterstützt aber zudem auch die Verwaltung von Terminen von externen Programmen (z.B. Microsoft Outlook), eine Anbindung dieser kann nur gewährleistet werden wenn entsprechende HTTP-APIs von den Produkten angeboten werden.

Design für mobile und stationäre Endgeräte

Da die neue Oberfläche alle bisherigen UI-Versionen ablösen soll ist es wichtig alle Endgeräte ihrer Möglichkeiten nach zu unterstützen. Eine bewährte Herangehensweise (“mobile-first” FIXME: footnote) hierfür ist es, das mobile Design zuerst zu gestalten um hier die wichtigsten Eigenschaften und Fähigkeiten direkt zu beachten. Das Design für größere Bildschirme kann dann darauf aufbauend mit zusätzlichen Features, die nur hier sinnvoll sind, ergänzt werden. Es ist zusätzlich auch wichtig zu beachten, dass viele mobile Nutzer je nach Standort keine gute oder zumindest nur eine teilweise gute Internetverbindung besitzen und es daher wichtig ist möglichst wenig Daten übertragen zu müssen bevor eine Webseite angezeigt wird. CSS das spezifisch für mobile Geräte ist sollte daher zuerst geladen werden, da dies weitere Downloads von irrelevanten Daten vermeidet. Auf stationären Geräten spielt es überwiegend keine Rolle, ob ein paar Byte für mobiles CSS zusätzlich geladen und anschließend durch die korrekten Styles ersetzt wird.

Vorerst nicht unterstützt

Einige momentane Features sind im Web entweder gar nicht oder nur mit sehr viel Aufwand umzusetzen. Diese werden daher vorerst nicht weiter beachtet und eventuell zu einem späteren Zeitpunkt erneut evaluiert:

- Clientseitiges Scripting

  Skripte werden momentan immer auf dem Client ausgeführt und evaluiert, dabei existiert u.a. Zugriff auf das Dateisystem und andere native Anbindungen des Betriebssystems. Diese Anbindungen können in einer Browserumgebung nicht angeboten werden.

- Interaktion mit anderen Prozessen

  Es war bisher möglich mit anderen Prozessen auf dem Client zu interagieren (Starten von externen Prozessen, Interkation mit diesen per COM, etc.). Ein Beispiel hierfür ist ein Hilfsprogramm mit dem Anrufe direkt am PC entgegengenommen und automatisch im cRM protokolliert werden konnten. Eine derartige Interaktion ist ein einer Browserumgebung, vor allem in einem von fast allen gängigen Browsern benutzten Sandbox-Modus, nicht möglich.

- Ereignisse

  Ereignisse sind Events die zu bestimmten Zeitpunkten der Programmlaufzeit ausgelöst werden und als Aktion zum Beispiel ein Skript starten oder eine E-Mail versenden können. Bevor diese in der Web-UI implementiert werden können muss zuerst evaluiert werden, welche dieser Zeitpunkte noch unterstützt werden können (Gegenbeispiel: “Nach Programmstart” – anstatt einem Prozess existiert nur noch eine Webseite, das Laden dieser ist nicht gleichbedeutend mit dem Starten eines Prozess)

2.2.2. Web-Apps

Zusätzlich zu diesen Anforderungen gibt im Webbereich noch die Bereiche die unterstützt werden sollten, um den Nutzern eine möglichst gute Erfahrung bei der Bedienung der Seite zu bieten und solche Bereiche, die den Entwicklungsprozess positiv beeinflussen. Einige wichtige dieser Anforderungen werden im Folgenden kurz vorgestellt und erläutert.

i18n / i10n

Die Akronyme “i18n” und “i10n” stehen für Internationalisierung beziehungsweise Lokalisierung [**i18n_i10n_ishida_w3c_miller_boeing_2018**]. Gemeint ist damit, dass für jede Sprachressource auf einer Webseite anstelle des Strings an sich ein Identifikator genutzt wird der mit Strings verschiedener Sprachen verknüpft ist. Je nach Einstellung (Auswahl durch Benutzer, Erkennung der Browsersprache, etc.) können dann automatisch alle Texte durch Texte in einer anderen Sprache ausgetauscht werden. Der cRM

unterstützt bereits mehrere Sprachen, die Ressourcen hierfür sind allerdings direkt in das Programm einbettet. Um die gleiche Technologie für die Webseite benutzen zu können müssten also alle Texte immer aus dem Backend angefordert und über das Internet übertragen werden. Dies wäre nicht nur langsam und eine unnötige Belastung für den Server (Texte müssten bei jedem Aktualisieren der UI-Schicht neu geladen werden), sondern auch sehr fehleranfällig. Bei einer unterbrochenen Verbindung könnten nicht einmal Fehlertexte angezeigt werden. Besonders im Hinblick auf den weiter unten beschriebenen “offline-first”-Ansatz sollten daher alle Texte in der UI-Schicht gespeichert und jederzeit abrufbar sein.

Barrierefreiheit

Barrierefreiheit von Software bedeutet, dass diese auch von Menschen mit körperlichen Einschränkungen gut genutzt werden kann. Dies ist selbstverständlich immer ein wünschenswertes Ziel, viele Standards im Webbereich machen ein solches Vorhaben aber besonders einfach. So gibt es beispielsweise bestimmte HTML-Tags die extra dafür geschaffen wurden, um von Sprachsoftware vorgelesen zu werden oder die Möglichkeit per austauschbarem CSS einen “Dark-Mode” FIXME: footnote? anzubieten, sofern ein System von Beginn an entsprechend aufgebaut wird. Es daher sollte evaluiert werden, welche Möglichkeiten für eine hohe Barrierefreiheit existieren und wie diese bei der Entwicklung beachtet und umgesetzt werden können.

Anleitungs-Modus / “Feature-Tour”

Um neuen Nutzern den Einstieg beim Erlernen eines Produkts zu erleichtern kann ein interaktiver Anleitungs-Modus, welcher entweder beim ersten Aufruf einer Webseite und / oder bei der ersten Nutzung einzelner Features ausgelöst wird, hilfreich sein. Ein solcher Modus hebt wichtige Bedienelemente hervor und gibt kurze Erklärungen zu diesen, welche entweder direkt weggeklickt werden können oder intelligent seltener erscheinen, je länger der jeweilige Nutzer das Produkt bereits kennt. Eine mögliche, fertige Umsetzung eines solchen Modus ist in Abbildung 2.4 von Intro.js FIXME: footnote? dargestellt.

“Offline-First”-Ansatz

In jüngster Zeit ist es möglich mithilfe von Service-Worker FIXME: footnote? und lokalen Speichermöglichkeiten im Browser Webseiten zu erstellen, die auch beim Aussetzenden Internetverbindungen mit minimalen Funktionen weiterhin funktionieren. So können etwa Änderungen welche an den Server geschickt werden müssen zwischengespeichert werden und sobald wieder eine Verbindung zum Internet besteht losgeschickt werden. Ein weiteres Beispiel ist das Anzeigen von Daten die vom letzten Besuch einer Seite stammen, wenn beim aktuellen Aufruf die Verbindung nicht stabil genug ist. Natürlich sollte ein Nutzer sowohl wenn er sich gerade im Offline-Modus befindet, als auch

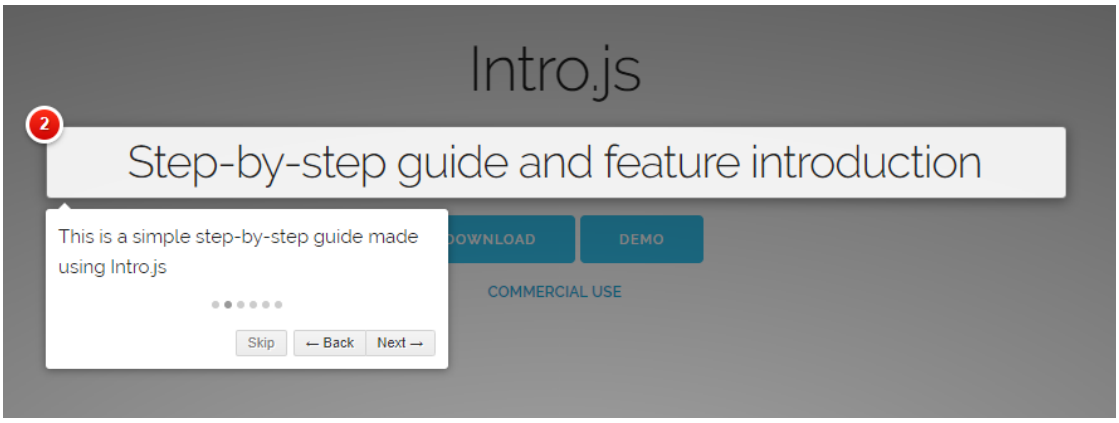


Abbildung 2.4: Anleitungs-Modus von Intro.js

wenn wieder eine Verbindung besteht immer entsprechend informiert werden. Vor allem auch in Kombination mit dem Ansatz die Bedienung einer Webseite für mobile Nutzer komfortabel zu gestalten ist es wichtig Verbindungsprobleme bereits mit einzuplanen und Nutzer nicht mit einer nicht-funktionalen Software alleine zu lassen.

Intelligente Fehlerbehandlung

Softwarefehler auf eine für den Benutzer angenehme Art und Weise zu behandeln und im Nachhinein detailliert zu analysieren ist bei traditioneller Software bereits ein nicht zu unterschätzendes Problem. Bei Webapplikationen kommt aber erschwerend hinzu, dass ein Teil der Logik nicht in einer Umgebung ausgeführt wird, welche man selbst kontrolliert (Browser auf Clientrechner). Umso wichtiger ist es, ein Konzept zu entwickeln um dort auftretende Fehler dennoch mit zu bekommen und auch genug Informationen zur künftigen Behebung dieser zu erhalten. Eine Lösung hierfür ist es, auftretende Fehler immer direkt zum Server zu schicken um sie vor Ort analysieren zu können – wenn die eigene Infrastruktur dafür nicht ausreichend ausgebaut ist gibt es auch entsprechende Online-Dienste bei denen man ein solches Fehlerlog speichern kann. Zusätzlich dazu müssen auch Parameter für Serveranfragen mit höchster Vorsicht ausgewählt werden. Timeouts müssen lange genug sein, um eine Serverantwort nicht fälschlicherweise zu verwerfen und kurz genug um bei einem tatsächlichen Fehler die Applikation nicht zu träge wirken zu lassen. Zudem sollten im Fehlerfall erneute Versuche intelligent gestartet werden, also weder zu häufig um den Server nicht unnötig zu belasten als auch nicht zu selten um dennoch einen responsiven Eindruck zu vermitteln.

Etablierte Authentifizierung

Der cRM besitzt zwar bereits ein integriertes Benutzersystem mit Rechteverwaltung, für die Portierung in das Internet sollte aber in Erwägung gezogen werden diese Rechteverwaltung mit einem oft genutzten, öffentlichen Protokoll wie OAuth oder ähnlichen

Protokollen zu verbinden, um die Verbindung über HTTP und den Schutz der Authentisierungsdaten nicht mit einer selbst entwickelten, schlechter funktionierenden Lösung handhaben zu müssen.

Caching

Um Ressourcen zu sparen ist es von Vorteil, komplexe Berechnungen und lange dauernde Übertragungen nicht mehrmals auszuführen. Im Bereich des Webs gibt es mehrere Möglichkeiten, solche Daten temporär zu speichern: Die einfachste Form besteht darin, Antworten auf eingehende Anfragen auf dem Server und ebenso danach auf dem Client zu wieder zu verwenden. Wenn der Client selbst kein Caching unterstützt kann mit “bedingten Anfragen” dennoch der Cache des Servers genutzt werden – es wird hierbei eine Anfrage an den Server geschickt und, sollte sich die Antwort im Vergleich zur letzten Anfrage nicht geändert haben, keine Daten, sondern nur ein entsprechender Hinweis an den Client zurück gesendet. Auch das HTTP-Protokoll unterstützt eingebaute Mechanismen (Cache-Header FIXME: footnote? um Caching beispielsweise über Cache-Proxies zu erleichtern. Bei allen Formen des Caching ist es jedoch kritisch, dass man die richtige Strategie zum Invalidieren der Daten auswählt um niemals mit zu stark veralteten Werten zu arbeiten, aber dennoch eine Effizienzsteigerung zu erreichen.

Container

Container-Technologien wie Docker oder Kubernetes helfen dabei, Abhängigkeiten der eigenen Software zu verwalten und auf jedem Entwicklungsrechner eine identische Umgebung zu schaffen. Auch für die Bereitstellung für Kunden ist es sehr viel einfacher einen bereits korrekt konfigurierten Container nur zu starten, anstatt eine komplette Umgebung auf einem Server einzurichten, vor allem wenn sich dieser Server bei externen Dienstleistern befindet. Da diese Konzepte bei der combit GmbH noch nicht eingesetzt werden ist es sinnvoll zu untersuchen, an welchen Stellen diese zu mehr Produktivität in Entwicklung und Verteilung der Software beitragen können.

Serverless

Unter “Serverless” versteht man das Konzept, dass als Firma keine eigenen Server für Kunden eingerichtet werden müssen, sondern stattdessen auf Lösungen von Drittherstellern (etwa AWS von Amazon oder FIXME: weiteres Beispiel) gesetzt wird, auf denen die eigenen Serverkomponenten dann laufen. Firmenintern betreibt man also keine Server, ist also “Serverless”. Der Vorteil von diesem Ansatz ist, dass die Verantwortung über die Serverhardware nicht selbst getragen werden muss und die Server durch die großen Kapazitäten der Hersteller je nach Belastung skaliert werden können. Im Gegenzug werden aber eigene Daten fremden Servern anvertraut, je nach Art und Sensibilität der Daten (Betriebsgeheimnisse, DSGVO FIXME: footnote?) kann dies nicht

gewünscht sein. Ein solcher Ansatz muss daher gut abgewogen werden, bevor man ihn weiter verfolgt.

2.3. Anforderungen an den Entwicklungsprozess

Neben den funktionalen Anforderungen ist es ebenso wichtig die richtigen Anforderungen für die Projektumgebung und -Entwicklung zu formulieren, um diesen Prozess möglichst effizient und skalierbar zu halten. Daher wird empfohlen folgende Technologien und Praktiken bei der Entwicklung des neuen Projektes zu nutzen:

- Tests

  Alle in Kapitel 3 vorgestellten UI-Frameworks enthalten bereits Lösungen um Tests auszuführen. Eine hohe Testabdeckung ist daher nicht nur nützlich um die Code zu erhalten und Regressionen zu erkennen sondern auch ohne hohe Kosten und Aufwand erreichbar.

- Continuous Integration

  Ein entsprechender TeamCity-Server von der Firma JetBrains ist firmenintern bereits vorhanden, daher sollten Tests von neuen Features immer auch auf diesem Server ausgeführt werden.

- Codereviews

  Pro Feature sollte auf einem gesonderten Branch gearbeitet und beim Zurückführen in den Masterbranch ein Review durch mindestens einen anderen Entwickler stattfinden um die Codequalität stets auf einem hohen Niveau zu halten. Auch für Reviews gibt es firmenintern bereits einen Upsource-Server (ebenfalls von der Firma JetBrains), welcher mit dem TeamCity-Server zusammen genutzt werden kann.

- Git

  Für bestehende Projekte wird bei der combit GmbH momentan SVN FIXME: footnote als Versionskontrollsystem eingesetzt. Um die oben genannten Punkte besser umsetzen zu können wäre es von Vorteil neue Projekte zukünftig mit Git zu verwalten. Im Vergleich zu SVN ist das Erstellen und Integrieren eines Branches, welcher für Features und die damit verbundene CI und Reviews genutzt werden soll, bei Git simpler, da weniger Konflikte entstehen. FIXME: quelle

2.4. Hauptaugenmerk dieser Arbeit

Eine komplette Umsetzung aller Anforderungen wäre im Rahmen dieser Thesis nicht zu bewältigen. Es gäbe zu viele Aspekte (geeignetes Hosting, eine sichere Authentifizierungsmethode, ein neuer Aktualisierungsmechanismus, etc.) die beachtet werden müssten, summiert reicht die Zeit nicht aus um Konzepte für all diese Aspekte in einer entsprechenden Qualität und Sorgfalt umzusetzen. Wie in diesem Kapitel bereits beschrieben kann die Oberfläche des cRM in zwei Hauptbereiche unterteilt werden. Zum einen diejenigen UI-Elemente welche sich nicht mit den anzuzeigenden Daten ändern, in Abbildung 2.1 zum Teil links und oberhalb der Detailansicht zu sehen (weitere Dialoge und Oberflächen werden kontextbasiert angezeigt), und zum anderen die Detail- und Listenansicht selbst. Die nicht datenabhängigen Elemente und Dialoge bestehen fast vollständig aus statischen Elementen mit teilweise speziellen Anforderungen welche bei der Umsetzung zwar zeitaufwändig wären, aber technisch kein interessantes Problem darstellen. Es wird im Weiteren daher nur die Oberfläche der Datenansichten (Detail- und Listenansicht) und die damit verbundenen Anforderungen betrachtet.

Aus der Liste der funktionalen Anforderungen des cRM also folgende Punkte:

- Automatisierte Erstellung
- Editier-Modus
- Individualisierung
- Suche / Filter / Sortierung

Ein Vorteil von moderner, komponentenbasierter Webtechnologie (alle in Kapitel 3 betrachteten Frameworks sind darauf ausgelegt) erlaubt es trotz der Trennung dieser UI-Bestandteile und der restlichen Oberfläche diese problemlos miteinander zu kombinieren. Es können sogar jeweils unterschiedliche Technologien für die jeweiligen Teile genutzt werden, durch das Transpilieren und herkömmliches HTML, CSS und JavaScript sind zur Laufzeit alle Umsetzungen miteinander kompatibel. Bei dieser zweistufigen Umsetzung kann das Ergebnis dieser Arbeit außerdem bereits in der Webansicht der bestehenden Desktopapplikation angezeigt und als Alternative zu den nativen Oberflächen angeboten werden. So können Kunden dieses Feature bereits im Vorfeld testen und evaluieren und wichtiges Feedback für die weitere Entwicklung einbringen.

3

Technologien

Die richtige Technologie zur Lösung einer Problemstellung auszuwählen garantiert noch keinen Erfolg, ist aber ein wichtiger Grundstein um überhaupt erfolgreich sein zu können. In diesem Sinne sollte nicht zu wenig Augenmerk auf den Vergleich und die Entscheidung zwischen auf dem Markt verfügbaren Alternativen gelegt werden. Dieses Kapitel beschreibt, welche Technologien für welchen Teil der Architektur in Frage kommen, stellt diese gegenüber und begründet die Entscheidung für die am besten geeignetste Alternative.

3.1. Entwicklungsumgebung

Die Entwicklung des combit Relationship Managers findet momentan ausschließlich mit Visual Studio statt. Wie in Kapitel 2 ist gibt es die Core-Komponente und Desktopkomponente, welche in C++ geschrieben sind, und eine auf den Core aufsetzende, in C# geschrieben, .NET-Komponente zur Bereitstellung der Webfunktionalität. Der bestehende Entwicklungsprozess soll nicht verändert und auch nicht weiter thematisiert werden, da die neu zu entwickelnde UI-Schicht eine unabhängige Codebasis besitzt und auch konzeptionell von den bisherigen Schichten getrennt ist.

Für das in Kapitel 2 angesprochene Tool zum Einlesen und Konvertieren der bisherigen UI-Repräsentation wurde wegen der vorhandenen internen Infrastruktur ebenfalls Visual Studio verwendet. Die Wahl der Programmiersprache fiel auf C#, da der Zugriff auf das Dateisystem und speziell auf XML-Dateien, das Format der momentanen Repräsentation, sehr einfach ist.

Für die neu entstehenden Webtechnologien kann jeder beliebige Texteditor benutzt werden, da keine speziellen Funktionen einer IDE oder eines Editors notwendig sind. Der in den nachfolgenden Kapiteln entwickelte Code wurde aufgrund der guten Autovervollständigung, der Integration mit Versionsverwaltungstools und der einfachen Erweiterbarkeit, mit Visual Studio Code von Microsoft erstellt. Wichtiger als der Editor ist die Sprache für welche er benutzt wird. Im Bereich der Webentwicklung hat sich JavaScript ohne Konkurrenz durchgesetzt und ist die einzige Sprache, die von allen Browsern unterstützt wird. Das spiegelt sich auch in der Auswahl der Frontend-Frameworks wieder, welche alle entweder direkt oder indirekt in JavaScript geschrieben sind.

3.2. Frontend-Frameworks

3.2.1. Traditionelle Lösungen

Frameworks und Bibliotheken die versuchen Entwicklern das Erstellen von UIs zu erleichtern gibt es schon sehr lange und in einer sehr großen Anzahl. Traditionelle Projekte wie jQuery-UI (Mobile), Bootstrap und neuere Namen (Materialize, Semantic-UI, Pure, etc.) setzen auf die Gestaltung von fertigen Komponenten die per CSS-Klasse in eine vorhandene Webseite integriert werden können. Durch diesen Fokus auf pures Styling mit CSS sind diese Projekte meist sehr klein und können schnell integriert werden. Sie eignen sich damit für rapides Prototyping, jedoch nicht um komplette Webseiten von Grund auf zu erstellen – vielmehr können sie mit den unten angesprochenen JavaScript-Projekten im Zusammenspiel benutzt werden.

3.2.2. Moderne, JavaScript-basierte Lösungen

In jüngster Zeit haben komponentenbasierte Frameworks, Single-Page-Applications und Shadow-DOM starken Wachstum zu verzeichnen. FIXME: erklärung Auch in diesem Bereich gibt es sehr viel Auswahl, aufgrund der zeitlichen Limitierung werden jedoch nur die größten drei Mitstreiter Vue, React und Angular FIXME: quelle, npm stats oder jahresrückb miteinander verglichen. Weitere Möglichkeiten für spätere Analysen sind Polymer, Ember, Knockout, Riot, etc.

3.2.3. Vergleich Vue, React & Angular

Generell

- Vue

  Entwickler legen viel Wert darauf möglichst kleine und dennoch performante Laufzeit zu entwickeln. Der Einstieg in Vue soll möglichst einfach sein, was durch eine sehr gute Dokumentation und ein hervorragendes CLI-Tool zum Erstellen neuer Projekte gelingt. Abhängigkeiten zwischen Daten und UI verfolgt Vue selbstständig, es werden daher immer nur Elemente aktualisiert wenn sich auch deren anzuzeigende Daten geändert haben. Das Erstellen von nativen Apps ist bei Vue nur mit Software von Drittherstellern möglich, die Entwickler arbeiten mit diesen aber zusammen um den Prozess zu verbessern. Das Verwenden von JavaScript-Templates (JSX) und Typescript ist mit etwas Konfiguration ebenfalls möglich. Vue hat im Vergleich noch ein kleines Ökosystem, kann aber dafür den stärksten Wachstum aller Frameworks verzeichnen.

- React

  React diktiert im Vergleich zu Vue noch weniger Vorgaben an die Entwickler, alle typischen Problemstellungen werden durch Bibliotheken gelöst. Dies ist möglich, da das Ökosystem sehr groß ist und eine breite und tiefe Auswahl an entsprechenden Bibliotheken bietet. Native Apps können mit dem vom gleichen Team entwickelten React-Native erstellt werden ohne dafür weitere Software zu benötigen. Durch die Konzentration auf wesentliche Features und Verbesserungen und eine stabile API stellt das React-Team sicher, dass Upgrades auf neue Versionen gar keine bis wenige Änderungen am bestehenden Code verlangen. Auch Typescript-Integration und damit verbundenes Tooling (Linter, Type checks, Autocompletion) sind problemlos möglich.

- Angular

  Es werden für alle typischen Anforderungen die bei der Entwicklung anfallen bereits Tools und Konzepte mit ausgeliefert, um diese Anforderungen zu erfüllen. Dadurch ist Angular an sich sehr viel mächtiger als Vue und React. Die Chance dafür dass man abgesehen von Angular noch weitere große Bibliotheken benötigt ist gering, jedoch erfordert die Nutzung dieser mitgelieferten Möglichkeiten auch dass man sie gut genug kennt. Der Einstieg in Angular wird von vielen daher als mühsam und zeitintensiv beschrieben. Die Dokumentation ist sehr umfangreich, teilweise aber auch unübersichtlich und das Ökosystem von Angular ist zwar größer als das von Vue, es hat jedoch auch negatives Wachstum zu verzeichnen.

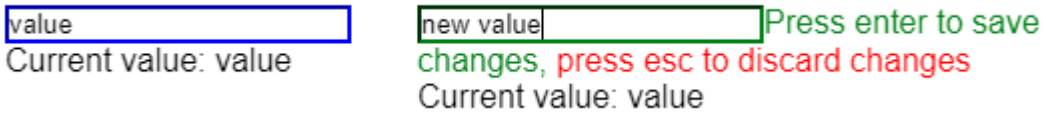


Abbildung 3.1: Beispielkomponente für die Umsetzung mit allen Frameworks

Aufbau von Apps (mit Beispielkomponente)

Um zumindest ein Gefühl für das Benutzen der jeweiligen Frameworks zu bekommen wurde eine Beispielkomponente ausgewählt die mit allen drei Frameworks umgesetzt werden soll. In Abbildung 3.1 sieht man die fertige Komponente. Bei Klick in das Edit-Feld ändert sich die Farbe des Randes um den Fokus zu signalisieren. Ändert man den Eintrag kann diese Änderung mit Esc verworfen oder mit Enter gespeichert werden. Der Text darunter gibt immer den momentan gespeicherten Wert wieder. Die Komponente wurde in Online-Editoren erstellt welche bereits eine fertig Umgebung mit dem jeweiligen Framework bieten. Auf die Benutzung weiterer Bibliotheken wurde hier explizit verzichtet.

- Vue

  Die Vue-Komponente wird komplett als Instanz einer Vue-Klasse angelegt, der man Daten und Methoden übergeben muss. Im HTML-Template werden die Vue-eigenen Attribute ersichtlich.

```
1  Vue.component('custom-edit-field', {
2      data: () => {
3          return {
4          editing: false,
5          inputValue: '',
6          value: '',
7          borderColor: 'blue'
8          }
9      },
10     methods: {
11         onFocus: function() {
12             this.editing = true
13             this.borderColor = 'green'
14         },
15         onFocusLost: function() {
16             this.editing = false
17             this.borderColor = 'blue'
18         },
19         onSubmit: function() {
20             if (this.editing) {
21                 this.value = this.inputValue
```

```
22          }
23          },
24          onAbort: function() {
25              if (this.editing) {
26              this.inputValue = this.value
27              }
28          }
29      },
30      template: `
31          <div>
32              <input
33                  v-on:focusin="onFocus" v-on:focusout="onFocusLost"
34                  v-on:keyup.enter="onSubmit" v-on:keyup.esc="onAbort"
35                  type="text" v-model="inputValue"
36                  class="inputForm"
37                  v-bind:style="{ borderColor: borderColor }"
38              >
39              <span v-if="editing" class="submitInfo">Press enter to
                    save changes, </span>
40              <span v-if="editing" class="abortInfo">press esc to
                    discard changes</span>
41              <div>
42                  Current value: {{value}}
43              </div>
44          </div>
45      `
46  })
```

Listing 3.1: Umsetzung der Beispielkomponente mit Vue

- React

  Eine React-Komponente kann entweder als herkömmliche Klasse oder nur als Funktion, welche die darzustellenden UI-Elemente als Rückgabewert enthält, angegeben werden.

```
1  class InputValue extends React.Component {
2      constructor(props) {
3          super(props)
4          this.state = {
5              editing: false,
6              inputValue: '',
7              value: '',
8              borderColor: 'blue'
9          }
10
11         this.input = React.createRef();
```

```
12
13          this.onFocus = this.onFocus.bind(this);
14          this.onFocusLost = this.onFocusLost.bind(this);
15          this.handleKeyPress = this.handleKeyPress.bind(this);
16          this.onChange = this.onChange.bind(this);
17      }
18
19      onFocus() {
20          this.setState({editing: true, borderColor: 'green'});
21      }
22
23      onFocusLost() {
24          this.setState({editing: false, borderColor: 'blue'});
25      }
26
27      handleKeyPress(e) {
28          if (e.key === 'Enter') {
29              this.setState({value: this.state.inputValue})
30          } else if (e.key === 'Escape') {
31              this.input.current.value = this.state.value;
32              this.setState({inputValue: this.state.value})
33          }
34      }
35
36      onChange(event) {
37          this.setState({inputValue: event.target.value})
38      }
39
40      render() {
41          return (
42              <div>
43              <input
44                  onFocus={this.onFocus}
45                  onBlur={this.onFocusLost}
46                  type="text"
47                  className={`inputForm ${this.state.editing ? '
                        editBorder' : 'normalBorder'}`}
48                  onChange={this.onChange}
49                  onKeyDown={this.handleKeyPress}
50                  ref={this.input}
51                  />
52                  {this.state.editing ?
53                      <span>
54                      <span className="submitInfo">Press enter to save
                            changes, </span>
55                      <span className="abortInfo">press esc to discard
                            changes</span>
```

```
                </span> : null
                }
                <div>
                Current value: {this.state.value}
                </div>
            </div>
        )
    }
}
```

Listing 3.2: Umsetzung der Beispielkomponente mit React

```
const InputValueHooks = () => {
    const [editing, setEditing] = useState(false);
    const [inputValue, setInputValue] = useState('');
    const [value, setValue] = useState('');
    const [borderColor, setBorderColor] = useState('blue');

    const input = useRef(null);

    const handleKeyPress = (e) => {
        if (e.key === 'Enter') {
            this.setState({value: this.state.inputValue})
        } else if (e.key === 'Escape') {
            this.input.current.value = this.state.value;
            this.setState({inputValue: this.state.value})
        }
    }

    return (
        <div>
            <input
                onFocus={() => {setEditing(true); setBorderColor('
                    green');}}
                onBlur=={() => {setEditing(false); setBorderColor('
                    blue');}}
                type="text"
                className={`inputForm ${editing ? 'editBorder' : '
                    normalBorder'}`}
                onChange={(event) => {setInputValue(event.target.
                    value);}}
                onKeyDown={handleKeyPress}
                ref={input}
            />
            {editing ?
                <span>
                <span className="submitInfo">Press enter to save
```

```
                    changes, </span>
32                  <span className="abortInfo">press esc to discard
                        changes</span>
33                  </span> : null
34              }
35              <div>
36                  Current value: {value}
37              </div>
38          </div>
39      )
40  }
```

Listing 3.3: Funktionale Umsetzung der Beispielkomponente mit React

- Angular

  Bei Angular besteht eine Komponente wie bei React aus einer herkömmlichen Klasse. Diese wird jedoch mit einer speziellen Annotation mit Informationen über das Aussehen und die Platzierung der Komponente versehen.

```
1  @Component({
2      selector: 'input-element',
3      template: `
4          <input #inputelement class="inputElement" [ngStyle]="{'border
               -color': editing ? 'green' : 'blue' }" (focus)="onFocus(
               $event)" (blur)="onBlur($event)" (keydown)="onKeyDown(
               $event)">
5          <span *ngIf="editing" class="submitInfo">Press enter to save
               changes, </span>
6          <span *ngIf="editing" class="abortInfo">press esc to discard
               changes</span>
7          <div>Current value: {{value}}</div>
8      `,
9      styles: [`.inputElement:focus {outline: none;} .submitInfo {
           color: green; } .abortInfo { color: red; }`]
10 })
11
12 export class InputElement  {
13     value = '';
14     editing = false;
15
16     @ViewChild('inputelement') input;
17
18     onFocus() {
19         this.editing = true;
20     }
21
```

```
22      onBlur() {
23          this.editing = false;
24      }
25
26      onKeyDown(event: KeyboardEvent) {
27          if (event.key === 'Enter') {
28              this.value = this.input.nativeElement.value;
29          } else if (event.key === 'Escape') {
30              this.input.nativeElement.value = this.value;
31          }
32      }
33  }
```

Listing 3.4: Umsetzung der Beispielkomponente mit Angular

State-Management

- Vue

  Standard ist, dass jede Komponente ihr eigenes State-Objekt besitzt. Zusätzlich dazu kann noch auf das State-Objekt der Vue-Instanz in der sich die Komponente befindet (Parent) zugegriffen werden. Dieses kann entweder direkt verändert oder per Store-Pattern FIXME: footnote? verwaltet werden. Vuex ist eine direkt von Vue entwickelte Bibliothek welche das genannte Store-Pattern umsetzt.

- React

  React unterstützt nur einen unidirektionalen Datenfluss mit “Props”. Diese werden Komponenten von ihren Elternkomponenten mitgegeben. Jede Komponente hat ihren eigenen State der per Props an Kinder weitergegeben (aber nicht direkt verändert) werden kann. Es gibt einige bekannte Bibliotheken welche das Verwaltung von State auf andere Arten lösen. Redux setzt das funktionale Flux-Pattern um, dabei werden Änderungen per Aktions-Event ausgelöst, die Events von Reducern ausgewertet und der State entsprechend verändert. Der neue State wird den entsprechenden UI-Elementen wieder unveränderbar (immutable) übergeben. MobX und react-easy-state setzen das bei Vue erwähnte Store-Pattern in React um.

- Angular

  State wird direkt in den Klasseninstanzen der Komponenten gespeichert und kann durch herkömmliche OOP-Konzepte weitergegeben und verändert werden. Zusätzlich existiert auch NGXS, welches auf dem gleichen Prinzip wie Redux beruht, aufgrund moderner Typescript-Features laut Angular aber einfacher zu benutzen sei.

Routing

- Vue

  Routing ist direkt in Vue integriert. Routen werden in JavaScript mit Komponenten verknüpft und bei Aufruf dieser Route wird die verknüpfte Komponente an einen zuvor festgelegten Platzhalter im DOM gerendert.

- React

  Nicht integriert, es gibt aber mehrere Alternativen als Bibliotheken. Bei Aviator werden Routen als geschachteltes Objekt übergeben, bei dem jedes Element eine Route und eine Zielfunktion enthält. Die Zielfunktion wird aufgerufen wenn zur entsprechenden Route navigiert wird und muss wissen wie und wohin (etwa über Dokument-Selektoren) die entsprechende Komponente gerendert werden muss. Die Nutzung von JSX ist dabei nicht mehr möglich. Bei react-router hingegen werden die Routen deklarativ in den JSX-Templates hinterlegt. Sie enthalten als Property die Route für die sie gerendert werden sollen und die eigentliche Komponente die angezeigt werden soll.

- Angular

  Analog zu Vue ist auch hier eine Lösung direkt integriert und funktioniert auch nach dem exakt gleichen Prinzip.

Testing

- Vue

  Vue enthält Test-Tools mit denen man Komponenten in Variablen rendern kann. Diese gespeicherten Komponenten sind herkömmliche JavaScript-Objekte und können dann mit weiteren (externen) Tools auf bestimmte Zustände und Eigenschaften geprüft werden.

- React

  Das Testen von React-Komponenten funktioniert analog zu Vue.

- Angular

  Alle zum Testen benötigten Tools sind bereits enthalten und sehr ausführlich dokumentiert. Wenn nur bestimmte Teilaspekte einer Anwendung wie etwa die UI-Komponenten getestet werden sollen ist der Aufwand dies einzurichten durch die Notwendigkeit sich mit dem kompletten Test-Konzept auseinander setzen zu müssen höher als bei Vue und React

Server-Side-Rendering

Bei allen Frameworks gibt es auch die Möglichkeit nur wenige Seiten “vor-gerendert” auf dem Server liegen zu haben anstatt sie dynamisch bei entsprechenden Anfragen zu generieren. Hierfür kann das Node.js Tool Prerender.io benutzt werden.

- Vue

  React liefert einen dafür vorgesehenen Renderer (vue-server-renderer) der Markup als Text generiert. Das kann beim initialen Laden der Seite ausgeliefert werden (erfordert einen Node.js Server).

- React

  Auch React liefert einen dafür vorgesehenen Renderer (ReactDOMServer) der Markup als Text generiert. Das kann beim initialen Laden der Seite ausgeliefert und auf Clientseite mit ReactDom.hydrate() mit Inhalten gefüllt werden (erfordert einen Node.js Server).

- Angular

  Angular benötigt für Server-Side-Rendering (SSR) zusätzliche Abhängigkeiten, Änderungen an der Konfiguration, ein zusätzliches Build / Bundle Target und ein zusätzliches Modul mit extra Konfiguration das auf dem Server läuft und dafür zuständig ist das neue JS-Bundle auszuliefern.

3.2.4. Fazit

Alle betrachteten Alternativen sind valide Optionen mit denen jegliche Anforderungen umgesetzt werden können. Der Einstieg in Angular scheint allerdings schwierig, da dort eine Vielzahl an Technologien für verschiedenste Zwecke mitgeliefert werden. Da es seitens der Funktionalität kein Unterschiede gibt kommt es mehr oder weniger darauf an welches Projekt einem Entwickler (-team) am meisten zusagt bzw. mit welchem es am besten arbeiten kann. React besitzt mit Abstand die größte Community, es kann daher davon ausgegangen werden, dass die Weiterentwicklung, viele aufsetzende Bibliotheken und Ressourcen im Netz garantiert sind. Ein weiterer Vorteil, die Benutzung von Typescript vorausgesetzt, ist die IDE-Unterstützung beim Schreiben der Komponenten-Templates. Im Vergleich zu Vue wird bei React hier ausschließlich JavaScript genutzt, was es erlaubt die Typisierung von Typescript zu nutzen um Autovervollständigung oder ähnliche Hilfen anzuwenden. Vue hingegen schreibt JavaScript-Funktionsaufrufe in Strings und nutzt eigene, für nicht Vue-Entwickler unbekannte, HTML-Attribute für Schleifen und Verzweigungen – da es sich dabei um Vue-Sonderfälle handelt kann die

IDE oft nicht helfen, es liegt also am Entwickler das entsprechende Wissen aufzubauen und keine Fehler zu machen.

Für den Prototypen wird vorerst React genutzt. Da die die Technologien zur Laufzeit alle kompatibel sind (sie werden alle in normales HTML, CSS und JavaScript transpiliert), kann dieser Code später aber auch mit Vue zusammen genutzt oder nach und nach ersetzt werden.

3.3. TypeScript

Bei JavaScript handelt es sich um eine dynamisch typisierte Sprache, dass heißt das sich Typen von Variablen zur Laufzeit ändern können und vor der Nutzung einer Variablen entsprechend überprüft werden müssen. Dies ist ein Vorteil wenn schnell kleinere Skripte geschrieben werden bei denen aufgrund ihrer überschaubaren Größe entsprechende Überprüfungen trivial sind oder es schlichtweg keine Rolle spielt ob ein Skript Fehler enthält. Für die Entwicklung von größeren Projekten ist diese Eigenschaft jedoch ein gravierender Nachteil, da Typen von Variablen im Code durch Entfernung von Deklaration und Nutzung nicht ersichtlich sind und daraus resultierende Fehler bei der Benutzung immer erst zur Laufzeit der betroffenen Zeilen auftreten oder sogar gar nicht als Fehler erkennbar sind und lediglich ein falsches (aber nicht unbedingt als falsch erkennbares) Ergebnis liefern.

Um dieser Art von subtilen Bugs vorzubeugen bietet Microsoft seit 2012 TypeScript an. Es handelt sich dabei um eine typisierte open-source Sprache welche als syntaktische Obermenge von JavaScript (normaler JavaScript-Code ist also ebenso gültiger TypeScript-Code FIXME: footnote?) beschrieben werden kann und die zu gewöhnlichem JavaScript transpiliert FIXME: footnote wird und somit trotz ihrer Vorteile keinen Kompatibilitätsnachteil besitzt. Da die Syntax auf JavaScript basiert ist das Erlernen dieser für JavaScript-Entwickler trivial. Durch Nutzung von typisierten Daten werden fehlerhafte Nutzungen von Variablen bereits bei der Entwicklung, eine entsprechende Integration des Editors vorausgesetzt, oder spätestens beim Transpilieren bemerkt und dem Entwickler als solche angezeigt. Ebenso erlaubt eine Editorintegration das Anbieten von Codevervollständigung und damit zu einem effektiveren Entwicklungsprozess.

Der einzige Nachteil von TypeScript ist, dass ein zusätzlicher Schritt zum Übersetzen in JavaScript-Code notwendig ist. Da die meisten Projekte im Webbereich aber sowieso nicht komplett ohne ähnliche Tooling-Schritte auskommen und der TypeScript-Compiler einfach in diese bestehenden Prozesse integrierbar ist handelt es sich hierbei nur um eine überschaubare Einschränkung. Dieser Nachteil kann aber auch positiv aus-

gelegt werden, da es durch den Übersetzungsschritt möglich ist, bereits Features und Standards zu benutzen, welche noch nicht von allen Browsern unterstützt werden. Bei der Übersetzung werden diese in semantisch identischen Code umgewandelt, der unter Einhaltung von älteren Standards gültig ist, aber für den Entwickler schwieriger zu schreiben wäre.

Aufgrund dieser Argumentation soll zur Unterstützung der Entwickler und Vorbeugen von Fehlern bei die Entwicklung der React-Seite ausschließlich TypeScript benutzt werden.

3.4. API-Anbindung

Die Anbindung der API kommen zwei Ansätze in Frage: Rest und GraphQL. Bei Rest handelt es sich um einen Architekturstil bei dem Ressourcen über URI-Endpunkte angesprochen und abgerufen werden. GraphQL hingegen ist eine von Facebook entwickelte Query-Sprache zur Abfrage von vorhandenen Daten. In diesem Abschnitt werden speziell die für dieses Projekt relevanten Vor- und Nachteile beider Ansätze angesprochen.

3.4.1. Rest

Rest-APIs sind heutzutage die Norm, um die Architektur allerdings korrekt umzusetzen ist aber mit viel Arbeit und gewissen Nachteilen verbunden. Dies könnte einer der Gründe dafür sein, dass ein großer Teil der angesprochenen Rest-APIs nur “Rest-like” sind. Als “Rest-like” werden die Implementationen bezeichnet, welche das sogenannte HATEOAS [1]-Konzept nicht umsetzen. Das Konzept beschreibt, wie ein Client durch die API navigieren soll – alle validen Übergänge vom momentanen State in den nächsten sind bereits in der momentanen Antwort einer Anfrage in Form von Links und Metadaten verfügbar. Ein Konsument muss damit nicht mehr wissen, welche Anfragen zu welchem Zeitpunkt gültig sind und kann immer genau die Aktionen anbieten, die auch von der API angeboten werden. Eine Versionierung der API (und damit stärkere Kopplung von Clients und Server) entfällt ebenfalls, neue Features können als weitere Aktion-Links auftauchen und alte Features zusätzlich zu einem Link die Information erhalten, dass die Aktion bald nicht mehr zur Verfügung stehen wird. Trotz verschiedener Standards die beim Erstellen der Struktur der Daten, vor allem bezgl. der Metadaten und wie diese ausgelesen werden können, helfen, ist es aufgrund des Mehraufwands bei jedem Endpoint dennoch mühsam eine Rest-API mit HATEOAS korrekt zu implementieren. Zusätzlich

[1] Hypermedia as the Engine of Application State

dazu, und dieser Nachteil wird im Gegenzug zu der gerade beschriebenen Flexibilität absichtlich in Kauf genommen, können solche APIs nur navigiert werden, indem vielen Links gefolgt und damit viele Netzwerkanfragen gemacht werden. Die Alternative, Rest-APIs ohne HATEOAS zu erstellen, ist zwar für die Entwickler einfacher, das Konsumieren der API wird dadurch aber deutlich erschwert. Eine Anbindung ist nur möglich indem dauerhaft die Dokumentation (welche in entsprechender Qualität vorhanden sein muss) konsultiert wird. Bei Änderungen an der API müssen auch die Clients angepasst werden, da sie keine Möglichkeit haben die Änderungen direkt mit zu bekommen. Um die Erstellung von richtigen Rest-APIs zu vereinfachen gibt es den OData-Standard der Entwicklern unter anderem die Definition und die Erstellung von Metadaten abnimmt, sodass diese sich auf die eigentliche Programmlogik konzentrieren können.

Caching ist bei Rest einfacher möglich als bei GraphQL, da jedes Endpoint-Daten-Paar als ein Eintrag im Cache angesehen werden kann. Werden entsprechende Cache-Header in den HTTP-Nachrichten gesetzt werden die Daten vom Browser (und eventuell vorhandenen Cache-Proxies) automatisch verwaltet. Dies funktioniert, weil Anfragen einer Ressource bei Rest immer alle Daten erhalten, die es zu dieser Ressource gibt. Um nicht immer alle Daten übertragen zu müssen besitzen viele Rest-APIs Parameter, mit denen man eine Anfrage einschränken kann. Je granularer diese Einschränkungen sind desto weniger überflüssige Daten müssen übertragen werden, gleichzeitig greift damit aber auch der Cache immer seltener, weil dauerhaft andere Endpoint-Daten-Paare angefragt werden.

3.4.2. GraphQL

Einer der Hauptgründe für die Entwicklung von GraphQL ist die höhere Netzwerklast bei Rest-APIs, die insbesondere bei mobilen Clients sehr negativ auffallen kann. Ein Hauptaugenmerk der Technologie ist deswegen, in diesem Bereich möglichst sparsam zu sein und möglichst wenig Daten zu übertragen. GraphQL ist laut npm FIXME: quelle die im Webbereich am meisten wachsende Technologie überhaupt, als Folge dessen wird es in Zukunft wahrscheinlich in vielen Projekten benutzt werden, der Aufbau von entsprechendem Know-How in diesem Bereich ist daher wichtig. Eine Abfrage bei GraphQL hat den gleichen Aufbau wie ein JSON-Dokument, mit der Besonderheit dass anstatt Schlüssel-Werte-Paaren nur Schlüssel bzw. der Name von Ressourcen angegeben werden. Eine Antwort hat immer den gleichen Aufbau wie eine Anfrage, die Schlüssel bzw. Ressourcennamen werden dabei wie in Listing **??** zu sehen mit den ausgelesenen Daten ergänzt. Durch die Nutzung von Typen für diese Abfragen kann garantiert werden, dass eine von den Tools akzeptierte Anfrage auf jeden Fall ein Ergebnis vom Server liefern wird. Die Abfragen müssen dabei nicht alle Felder einer Ressource ent-

halten, es können immer genau die Daten abgefragt werden die in einer spezifischen Situation gerade benötigt werden. Ebenfalls können mehrere unabhängige Anfragen zu einer großen Anfrage zusammengefasst und mit nur einem HTTP-Roundtrip beantwortet werden. Das typisierte Schema erlaubt weiterhin, Konsumenten jederzeit Informationen über sich selbst (Metadaten), etwa den Type eines Feldes oder weitere mögliche Felder zur Abfrage, zur Verfügung zu stellen. Vorausgesetzt ein Client kennt einen den Einsprungpunkt eines Graphen kann er mit diesen Metainformationen alle weiteren Informationen programmatisch auslesen.

```
Query:
{
  me {
    name
  }
}

Result:
{
  "me": {
    "name": "Luke Skywalker"
  }
}
```

Listing 3.5: Beispielquery aus der GraphQL Dokumentation FIXME: quelle?

Um Entwickler zu unterstützen existiert ein visueller Query-Editor “GraphiQL” mit Autovervollständigung und aus dem Schema generierter Dokumentation. Mithilfe diesen Editors kann sehr leicht in der API navigiert und direkt code-seitig benutzbare Abfragen generiert werden. Viele moderne Webseiten basieren nicht mehr nur auf der Annahmen, dass irgendwann Daten vom Server abgefragt werden, sondern auch darauf dass serverseitige Events / Änderungen das Laden oder Anzeigen von Daten auslösen können. GraphQL spezifiziert für diese Anforderung ein Abonnement-System: ein Client registriert sich beim Server für alle Daten, deren Änderung er sofort mitbekommen möchte und gibt an, welche Abfrage dafür ausgeführt werden soll. Bei einer Änderung wird diese Abfrage dann ausgeführt und die Daten über eine Websocket-Verbindung sofort an den Client übertragen.

Um diese ganzen Eigenschaften zu ermöglichen ist es jedoch notwendig, dass sowohl der Client als auch der Server mit dem gleichen GraphQL-Schema arbeiten und damit aneinander gekoppelt sind. Ein weiterer Nachteil ist es, dass Aufgrund der Nutzung von GraphQL (nur HTTP-POST und identischer Endpoint) Caching nicht auf HTTP-Ebene geschehen kann und damit die Verantwortung für gute Cache-Strategien hauptsächlich beim Client liegen.

Zwei bekannte Bibliotheken welche die GraphQL-Spezifikation in JavaScript umsetzen und das Erstellen von Queries, Caching und Debuggen erleichtern sind Apollo und Relay. Relay wurde ebenfalls von Facebook entwickelt, dennoch hat Apollo eine sehr viel größere Community. Für die Umsetzung im Backend (.NET) existieren ebenfalls Implementierungen, die am meisten genutzte davon ist graphql-dotnet.

3.4.3. Fazit

Rest und GraphQL können beide für den gleichen Zweck benutzt werden, sie konkurrieren jedoch nicht direkt miteinander. Eine API kann ebenso beide Ansätze entweder ergänzend oder parallel anbieten. Bestehende Rest-APIs können durch das Resolver-Konzept von GraphQL auf einfache Art und Weise gewrapped werden und so auf beide Arten angesprochen werden. Rest ist flexibler als GraphQL, um diese Flexibilität aber richtig zu nutzen und damit eine API zu schreiben welche viele Jahre genutzt und skaliert werden kann ist aber viel Erfahrung und Aufwand notwendig. GraphQL scheint daher als die richtige Wahl für Firmen oder Personen die noch nicht viel Erfahrung in diesem Bereich sammeln konnten. GraphQL benötigt sowohl eine Server als auch eine Clientkomponente und hat damit mehr Abhängigkeiten als dies bei Rest der Fall ist, im Gegenzug ist es dadurch aber möglich effiziente, typsichere Anfragen zu erstellen. Ebenso wird mit dem mitgelieferten GraphiQL-Tool ein Abfrage-Editor mit Autovervollständigung und Fehlerbeschreibungen bereitgestellt, der es Nutzern sehr einfach macht eine API und deren Möglichkeiten zu erkunden sowie für alle Bestandteile direkt eine Dokumentation einzusehen. Ein weiterer Vorteil von GraphQL ist es, dass durch die typisierten Abfragen eine automatische Generierung von Mocking-Daten möglich ist. Mithilfe solcher Mocking-Daten kann das Entwicklerteam den API Client im Frontend entwickeln und testen, bevor das Backend mit den Echtdaten zur Verfügung steht.

Wegen der einfacheren Nutzung von GraphQL und der besseren Unterstützung von Entwicklern durch Typisierung und mitgelieferten Tools wird im Prototyp diese Technologie verwendet.

4

Ausarbeitung des Architekturkonzepts

4.1. Analyse bisheriger UI-Aufbau

4.1.1. DLI

4.1.2. VLC

4.2. React-Komponenten

4.2.1. Erste Layout-Überlegungen

Zu Beginn erschien die zentrale Frage, wie es technisch möglich ist, die Komponenten der Detailansicht vom Benutzer anpassbar anzuordnen. Eine naive Herangehensweise kann in Abbildung 4.1 gesehen werden. Umgesetzt wurde dies mit einem CSS-Grid (rote Umrandung) das entweder horizontal oder vertikal in zwei Hälften getrennt werden kann. Jede dieser Hälften stellt abermals ein CSS-Grid dar das beliebig zwischen Elterncontainern verschoben werden kann. Abstrakt handelt es sich bei diesem Ansatz um einen nicht balancierten binären Baum an dessen Endpunkten (Blätter) sich genau eine UI-Komponente (blaue Umrandung) befindet. Es wurde schnell klar dass diese Lösung in ihrer binären Form nicht flexibel genug ist um vorhandene Layouts abzubilden. Da ein Ausbau auf eine Struktur mit variabler Anzahl an Verzweigungen sehr komplex und zeitaufwändig gewesen wäre

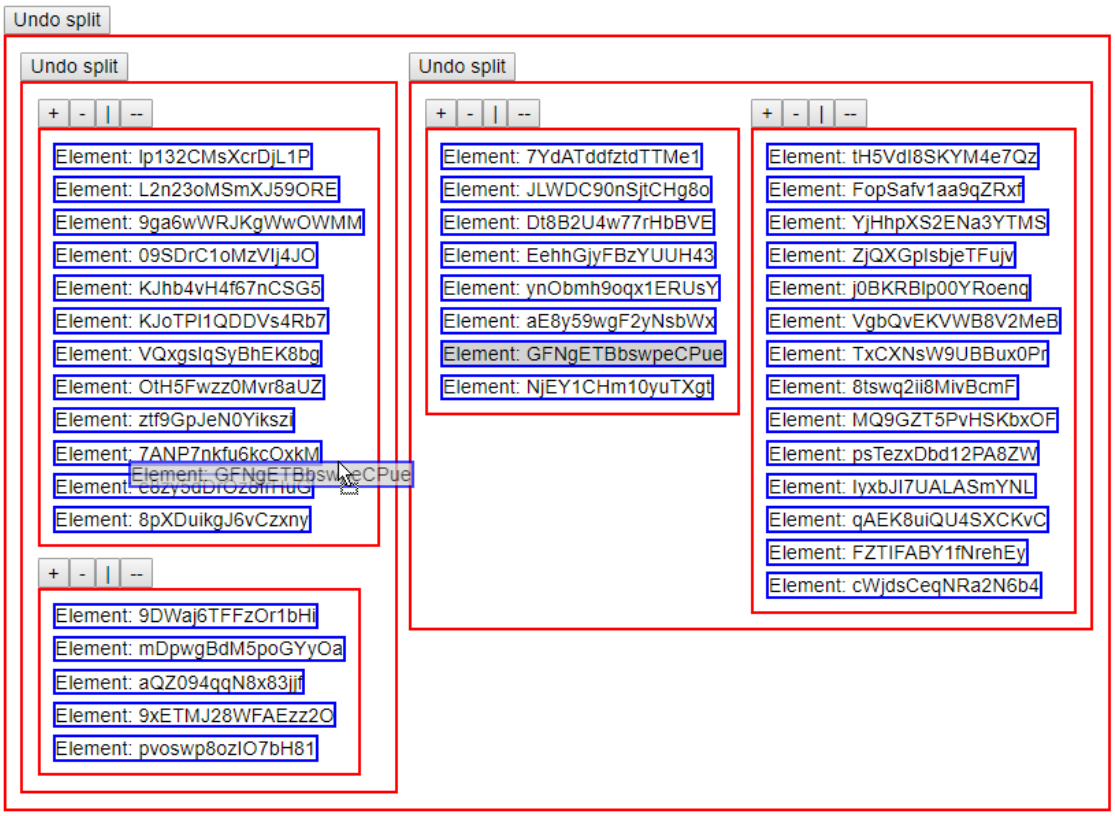


Abbildung 4.1: Eigener Layout-Prototyp mit CSS-Grids

wurde der Ansatz gänzlich verworfen und nach einer Alternativen Lösung gesucht. FIXME: bessere erklärung und begründung warum nicht flexibel genug

4.2.2. Verwendung der Komponenten

4.2.3. Identifikation auf Server

Um die UI-Elemente mit Daten aus der Datenbank zu befüllen muss eine entsprechende Identifikation möglich sein. Es wird vorausgesetzt dass diese eindeutige ID, ob sie aus Tabellennamen plus Spaltenname der Datenbank oder aus anderen Informationen besteht, zum Zeitpunkt der Übersetzung einer Ansicht bereits bekannt ist und mit ausgelesen werden kann. Bei Anfragen an den Server werden alle IDs der beteiligten Elemente mit an den Server übertragen, ebenso wie dieser bei Antworten immer die IDs der Elemente, für welche die Antwortdaten gedacht sind, sendet.

4.3. Tests und Continuous Integration

5

Implementierung eines Prototypen

In diesem Kapitel wird die Entwicklung eines auf dem zuvor ausgearbeiteten Konzept basierten Prototypen beschrieben. Dieser Prototyp enthält nicht alle im Konzept beschriebenen Anforderungen und befindet sich auch nicht in einem finalen Entwicklungsstadium, kann aber mit weiteren Entwicklungsressourcen als Grundlage für eine finale Implementierung genutzt werden. Er soll zeigen, wie sich das entworfene Konzept umsetzen lässt und die darin benutzten Technologien miteinander interagieren.

5.1. Tool zum Parsen der vorhandenen UI-Persistierung

Wie in Kapitel 4 beschrieben müssen die Dateien welche das momentane UI-Layout enthalten in ein web-freundlicheres Format (JSON) übersetzt werden. Bei diesem Schritt ist es auch direkt möglich Informationen die zukünftig nicht mehr benötigt werden nicht mit zu übernehmen und andere Informationen in eine optimalere Struktur zu überführen. Für diesen Zweck wurde ein kleines Hilfstool in C# geschrieben, welches sowohl die “.dli”-Datei der Detailansicht als auch die “.vlc”-Datei der Übersichtsliste einer einzelnen cRM-Ansicht als Input erhält und daraus eine “.json”-Datei mit allen benötigten Informationen erstellt. Um die Anpassbar- und Wiederverwendbarkeit des Tools zu maximieren wurde das Visitor-Pattern FIXME: Erklärung? angewandt. In Abbildung 5.1 ist der Aufbau des Tools erkennbar.

Zunächst wird für jedes einzulesende Token (XML-Element) eine eine Klasse vom “Acceptor”-Interface (IDLIAcceptor / IVLCAcceptor) abgeleitet, dieses Interface enthält eine einzige Methode “Apply”, welche einen Visitor (IDLIVisitor / IVCLVisitor) überge-

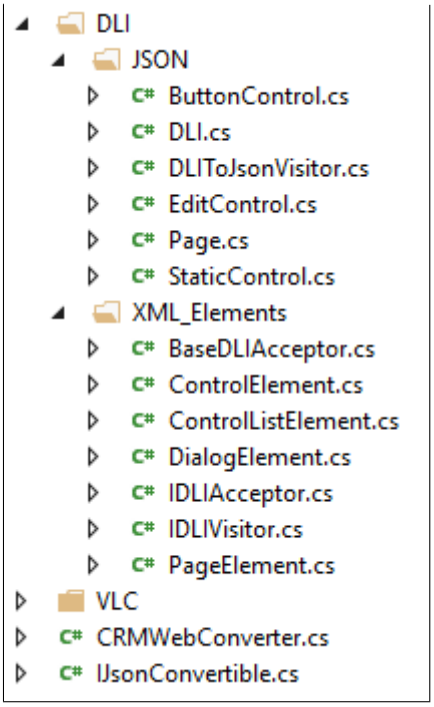


Abbildung 5.1: Datei-Struktur des Konvertierungstools

ben bekommt. Die jeweiligen Token-Klassen werden mit dem von ihnen verwalteten XML-Element (XElement) initialisiert. Wie im Quellcodeauszug 5.1 anhand der Klasse “PageElement” beispielhaft zu sehen werden aus diesem XML-Element die relevanten Informationen über das Element selbst (Name und Metadaten) und dessen verschachtelte Kinder-Elemente (Liste von Controls) ausgelesen.

```
1  public PageElement(XElement element) : base(element)
2  {
3      Name = element.Attribute("name").Value;
4
5      foreach (var childElement in element.Elements())
6      {
7          if (CheckAttribute(childElement, "name", "Title"))
8          {
9              MetaData = childElement.Value;
10             break;
11         }
12
13
14         if (CheckAttribute(childElement, "name", "Controls"))
15         {
16             ControlList = new ControlListElement(childElement);
17         }
18     }
```

```
19 }
```

Listing 5.1: Initialisierung der PageElement-Klasse

FIXME: syntax highlights Nachdem die Informationen der XML-Datei auf diese Art und Weise in ihre einzelnen Token-Instanzen übersetzt wurden wird die Apply-Methode, zu sehen in Quellcodeauszug 5.2, des zentralen Tokens (DialogElement) mit einer Visitor-Instanz aufgerufen. Der Visitor erhält als Parameter eben diese Instanz und extrahiert alle für ihn relevanten Informationen. Anschließend ruft er rekursiv die Apply-Methoden der Kinder-Token auf und liest auch aus diesen die relevanten Informationen aus. Diese Aufrufe sind im Quellcodeauszug 5.3 zu sehen. Nachdem alle Tokens vollständig besucht wurden können die gewonnenen Daten als JSON-String ausgegeben werden.

```
1 public override void Apply(IDLIVisitor visitor)
2 {
3     visitor.Visit(this);
4 }
```

Listing 5.2: Apply-Methode der DialogElement-Klasse

```
1 public void Visit(DialogElement dialog)
2 {
3     foreach (var page in dialog.Pages)
4     {
5         page.Apply(this);
6     }
7 }
8
9 public void Visit(PageElement page)
10 {
11     if (_currentPage != null)
12     {
13         throw new InvalidOperationException("Page must be processed
               completely before continuing to next page");
14     }
15
16     _currentPage = new Page(page.Name);
17     _currentPage.AddMetaData(page.MetaData);
18
19     page.ControlList.Apply(this);
20
21     _dli.Add(_currentPage);
22     _currentPage = null;
23 }
24
```

```
25 public void Visit(ControlListElement controlList)
26 {
27     if (_currentPage == null)
28     {
29         throw new InvalidOperationException("Page must be valid to process
               its control list");
30     }
31
32     foreach (var control in controlList.Controls)
33     {
34         control.Apply(this);
35     }
36 }
37
38 public void Visit(ControlElement control)
39 {
40     if (_currentPage == null)
41     {
42         throw new InvalidOperationException("Page must be valid to process
               its controls");
43     }
44
45     _currentPage.AddControl(CreateControlByType(control.Type, control.
           Items));
46 }
47
48 private IJsonConvertible CreateControlByType(string type, Dictionary<
       string, string> items)
49 {
50     switch (type)
51     {
52         case "STATIC":
53             return StaticControl.ConstructWithItems(items);
54         case "BUTTON":
55             return ButtonControl.ConstructWithItems(items);
56         case "EDIT":
57             return EditControl.ConstructWithItems(items);
58         default:
59             return null;
60     }
61 }
```

Listing 5.3: Besuchen von und Extrahieren relevanter Informationen aus XML-Tokens durch den JSON-Visitor

Die Flexibilität dieser Architektur, welche in der Abbildung 5.2 nochmals übersichtlich als Klassendiagramm dargestellt wird, ist ebenso daran zu erkennen, dass der einzi-

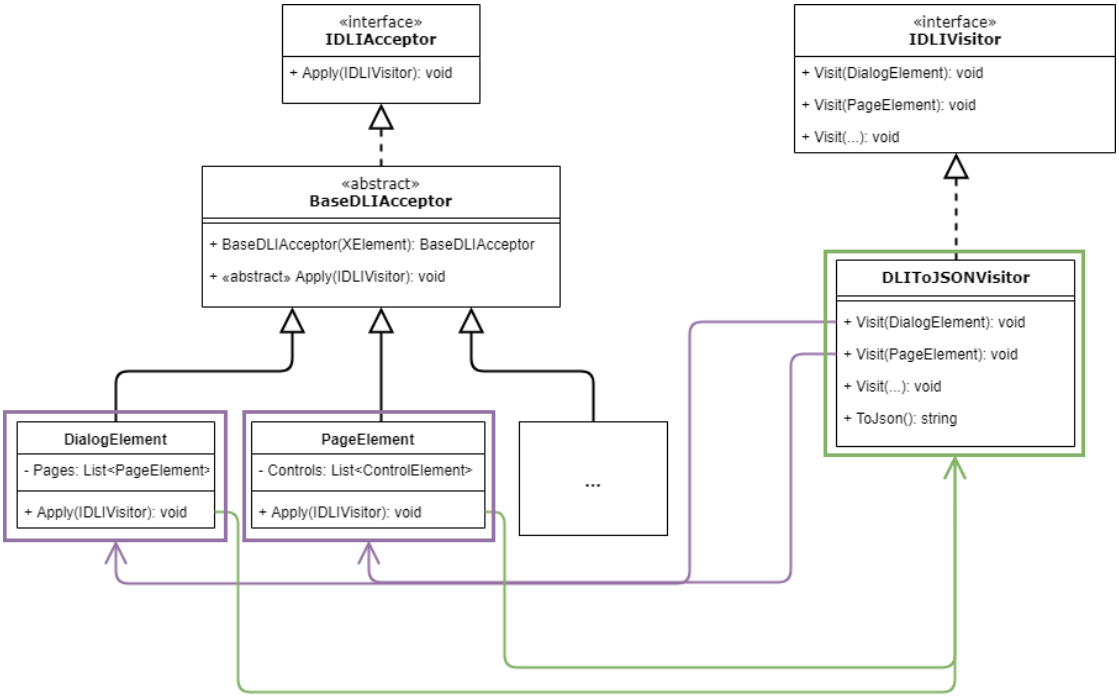


Abbildung 5.2: Klassendiagramm der Visitor-Struktur des Konvertierungstools

ge Unterschied beim Auslesen von Detailansicht-Datei und Übersichtslisten-Datei in der Implementierung der Interfaces besteht. Sowohl Acceptor- FIXME: some kind of highlight, italize? als auch Visitor-Klassen können sehr leicht einzeln angepasst oder ersetzt werden. Ebenso ist es möglich weitere Tokens, welche eventuell zu einem späteren Zeitpunkt benötigt werden, aus den XML-Dateien auszulesen, indem man weitere Acceptor- und Visitor-Implementierungen hinzufügt.

Der Input und das Endergebnis in Form eines vom GraphQL-Server direkt verwertbaren JSON-Dokument ist in den beiden Quellcodeauszügen 5.4 und 5.5 anhand eines kleines Auszuges ersichtlich. FIXME: vollständigeres beispiel und erklärung?

```
<list name="Page0">
    <list name="Controls">
        <list name="Control">
        <item name="TypeIndex">1</
            item>
        <item name="Type">STATIC</
            item>
        <item name="Rect(l,t,w,h)">
            (18,28,420,11)</item>
        <item name="ColorBackground
            ">Transparent</item>
        <item name="CondVisible">
            Deactivated</item>
        <item name="InputLCID">0</
            item>
        <item name="
            EnvironmentRestriction"
            >0</item>
        <item name="UIModifiers">0
            :UkdCKDI1NSwwLDAp,1
            :IlNlZ29lIFVJIg==,2
            :RmFsc2U=,3:RmFsc2U=,4
            :RmFsc2U=,5:RmFsc2U=,6
            :MTMuMA==</item>
        <item name="Font">
            {(255,0,0)
            ,13,-17,0,0,0,400,0,0,0,0,3,2,1,34,
            Segoe UI}</item>
        <item name="Calculated">
            TRUE</item>
        <item name="Formula">«&quot
            ;--- deaktiviert seit &
            quot; + Date$(Date(
            DeactivatedOn),&quot
            ;%02d.%02m.%04y %02H:
            %02i:%02s&quot;) + Cond
            (not IsNullOrEmpty(
            DeactivatedReason), &
            quot;, Grund: &quot; +
            DeactivatedReason) + &
            quot; ---&quot;»</item>
        <item name="Orientation">0<
            /item>
        </list>
        ...
```

Listing 5.4: XML-Input

```
"pages": [
    {
    "name": "Page0",
    "title": "",
    "controls": [
        {
        "type": "static",
        "text": "",
        "textFromServer":
            true
        },
        ...
```

Listing 5.5: JSON-Ergebnis

5.2. Webseite

5.2.1. CRA-Skelett

5.2.2. Benutzung der vorgefertigten UI-Elemente

5.2.3. Storybook und Tests

5.2.4. GraphQL-Mock-Server und Resolver

6

Fazit

6.1. Ausblick