

Praxissemesterbericht

Entwicklung einer Admin Seite für eine SAP-Webanwendung mit
dem SAP User Interface 5 (SAPUI5) Framework

Vorgelegt von:

Luc Dreibholz

Matr.Nr.: 2210544

Bachelor-Studiengang Informatik

Hochschule Mannheim

Fakultät für Informatik

Unternehmen:

sovanta AG

Mittermaierstraße 31

69115 Heidelberg

Betreuer/in: Oliver Schmitt

Praktikumszeitraum: 01.03.2024 - 26.07.2024

Fertigstellung: 12.10.2024

Sichtvermerk der Firma:

Firmenstempel und Unterschrift der betreuenden Person(en)

Datum: _____ Unterschrift: _____

Datum: _____ Unterschrift: _____

Inhaltsverzeichnis

1	Firmenumfeld und Zielsetzung der Arbeit	1
1.1	Firmenumfeld	1
1.2	Zielsetzung der Arbeit	1
2	Wochenberichte	2
2.1	Einarbeitung in das SAPUI5 Frontend Framework	2
2.2	Vertiefung der SAPUI5 Kenntnisse und SAP Cloud Application Programming Model (SAP CAP) Backend Framework	2
2.3	Einarbeitung in die Project-Codebase und erste eigene Tickets	3
2.4	Erstes Sprint Review und weitere Bearbeitung von Tickets	4
2.5	Reihenfolge der Länder in der ganzen Anwendung ändern	4
2.6	Fertigstellung der Länderreihenfolge und eine neue Aufgabe	5
2.7	Bugfixing und ein neues Feature	6
2.8	Unittesting und Review Präsentation	7
2.9	Regression- und Unittests	7
2.10	Weitere Unittests und Beginn der Testphase	8
2.11	Bugfixing und Unittests	9
2.12	Komplizierteres Unittesting	9
2.13	Unittesting und der Start einer neuen Version	10
2.14	Probleme bei der Feature Implementierung	10
2.15	Ende der Implementierungsphase und Beginn der Regressionstests	11
2.16	Unittesting und Beginn des Refactorings	11
2.17	Fertigstellung des Standardwert-Refactorings und weitere Refactorings . . .	12
2.18	Refactoring und Unittests	12
3	Projektbericht	14
3.1	Zusammenfassung	14
3.2	Einleitung	14
3.3	Technologische Grundlagen	16
3.3.1	Einführung in TypeScript	16
3.3.2	Unterschiede zwischen JavaScript und TypeScript	18
3.3.3	Vorstellung weiterer verwendeter Technologien und Frameworks . . .	19

3.4	Anforderungsanalyse	22
3.4.1	Ist-Zustand	22
3.4.2	Anforderungen	23
3.5	Lösungskonzept	27
3.5.1	Designkonzept	27
3.5.2	Software-Design	31
3.6	Implementierung	33
3.6.1	Implementierungsdetails	33
3.6.2	Schrittweise Beschreibung der Implementierung	36
3.6.3	Herausforderungen und Problemlösungen	44
3.7	Fazit	45
	Literatur	46
	Abbildungsverzeichnis	47
	Tabellenverzeichnis	48
	Quellcodeverzeichnis	49
	Abkürzungsverzeichnis	50

1 Firmenumfeld und Zielsetzung der Arbeit

1.1 Firmenumfeld

Die sovanta AG ist ein in Heidelberg ansässiges Unternehmen, das sich auf die Entwicklung von Softwarelösungen für Unternehmen spezialisiert hat. Besonders im Bereich der SAP-Anwendungen ist sovanta darauf fokussiert, komplexe Geschäftsprozesse durch benutzerfreundliche und effiziente Lösungen zu verbessern. Das Unternehmen arbeitet eng mit verschiedenen Kunden aus unterschiedlichen Branchen zusammen, um maßgeschneiderte Software zu entwickeln, die genau auf deren Bedürfnisse abgestimmt ist.

Das Arbeitsumfeld bei sovanta zeichnet sich durch eine offene und teamorientierte Atmosphäre aus. In den Projektteams wird oft mit agilen Methoden wie Scrum gearbeitet, was eine flexible und dynamische Arbeitsweise ermöglicht. Der regelmäßige Austausch innerhalb der Teams und mit den Kunden spielt dabei eine wichtige Rolle, um schnell auf neue Anforderungen reagieren und eine hohe Qualität sicherstellen zu können. Die Kombination aus technischer Expertise und einer starken Fokussierung auf die Bedürfnisse der Nutzer ist ein wesentlicher Bestandteil der Firmenphilosophie von sovanta.

1.2 Zielsetzung der Arbeit

Während meines Praktikums war ich Teil eines agilen Scrum-Teams, das eine SAP-Anwendung für ein großes Unternehmen entwickelte. Ich arbeitete an Features sowohl im Frontend als auch im Backend.

Zusätzlich habe ich Bugs gefixt, um die Anwendung stabiler zu machen. Ein weiterer wichtiger Teil meiner Arbeit war das Schreiben von Tests, um die Qualität des Codes sicherzustellen.

Meine Aufgabe bestand darin, das Projekt voranzutreiben, indem ich neue Funktionen entwickelte und die bestehende Code-Qualität verbesserte.

2 Wochenberichte

2.1 Einarbeitung in das SAPUI5 Frontend Framework

In meiner ersten Praktikumswoche begann mit einem kurzen technischen Onboarding, indem mir mein Equipment bereitgestellt und alle sovanta-Accounts eingerichtet wurden. Danach habe ich mich intensiv mit dem SAPUI5-Framework auseinandergesetzt. Ziel war es, eine umfassende Einführung in die Grundlagen dieses Frameworks zu erhalten. Dabei lag der Fokus auf dem Verständnis von SAPUI5-spezifischen Konzepten wie Data Binding, Fragments, Views und Routing.

Die Einarbeitung begann ich mit einer Analyse der Dokumentation und Online-Ressourcen, um ein solides Fundament zu legen. Dabei kamen insbesondere Fragen zum effektiven Einsatz von Data Binding und zur korrekten Implementierung von Fragments auf.

Die Herausforderung bestand darin, die theoretischen Kenntnisse in der praktischen Anwendung zu vertiefen. Durch aktives Experimentieren und Umsetzen kleiner Beispiele gelang es mir, die Zusammenhänge besser zu verstehen. Schwierigkeiten traten vor allem bei der korrekten Integration von Data Binding-Funktionalitäten auf, wobei ich auf Online-Foren und Tutorials zurückgriff, um Lösungsansätze zu finden.

2.2 Vertiefung der SAPUI5 Kenntnisse und SAP CAP Backend Framework

Die zweite Woche meines Praktikums stand ganz im Zeichen der Vertiefung meines Verständnisses für das SAPUI5-Frontend-Framework und dem dazugehörigen SAP CAP Backend-Framework.

Aufgrund von Problemen mit dem Zugang zu der Codebase des Projektes, dem ich zugeeignet wurde, verzögerte sich die Einarbeitung in die Codebase, an der ich die nächsten

Monate arbeiten werde. Diese Zeit nutzte ich jedoch effektiv, indem ich mich eigenständig den fortgeschrittenen Konzepten von SAPUI5 und der den Grundlagen von SAP CAP widmete. Hierbei fing ich an die Dokumentation für das SAP CAP Framework zu lesen und erste kleine Projekte aufzusetzen.

Die größte Herausforderung dieser Woche war es, die erworbenen Kenntnisse in der Praxis anzuwenden und eine Verbindung zwischen einer eigenen SAPUI5-Anwendung und dem SAP CAP Backend herzustellen und vor allem das Konzept des Open Data Protocol (ODATA) Datenmodells zu verstehen und zu implementieren, welches für SAP-Anwendungen essenziell ist.

Durch kontinuierliche Experimente und die Analyse von Beispielcode gelang es mir jedoch, ein tieferes Verständnis für dieses Datenmodell zu gewinnen und dieses Verständnis in einer ersten Anwendung mit einem SAPUI5 Frontend und SAP CAP Backend umzusetzen.

2.3 Einarbeitung in die Project-Codebase und erste eigene Tickets

In der dritten Woche konnte ich endlich auf die Codebase des Projekts zugreifen, was mir die Einarbeitung in das laufende Projekt ermöglichte. Nachdem ich mich mit der Codebase etwas vertraut gemacht hatte, konnte ich mich auch direkt aktiv in das laufende Projekt einbringen, indem ich meine ersten eigenen Tickets bearbeitete. Diese Tickets beinhalteten die Implementierung verschiedener Features und Anpassungen, um die Funktionalität und Benutzerfreundlichkeit der Anwendung zu verbessern.

Die Hauptaufgabe diese Woche bestand darin, einen neuen Filter und eine neue Spalte, für bestehende Werte aus der Datenbank, zur Tabelle aller Kaufprojekte auf der Übersichtsseite hinzuzufügen. Zusätzlich zu dieser größeren Aufgabe bearbeitete ich auch kleinere Tickets, wie das Ändern von Dropdown-Optionen, die Anpassung von Abkürzungen in der gesamten Anwendung und das automatische Befüllen des "WährungFelds beim Erstellen eines bestimmten neuen Kaufprojektes.

Die größte Herausforderung bestand darin, die komplexe Codebase und Architektur eines schon so großen Projektes zu verstehen und dann an den richtigen Stellen Änderungen zu machen, um die Ergebnisse zu erzielen, die gefordert waren. Diese Herausforderung konnte ich durch viel ausprobieren und befragen meines Betreuers, zumindest teilweise,

bewältigen.

2.4 Erstes Sprint Review und weitere Bearbeitung von Tickets

Diese Woche lief ähnlich wie letzte Woche und ich arbeitete weiter an eigenen Tickets. Dazu kam dann allerdings auch, dass in dieser Woche das Sprint Review anstand, in dem ich dann die Ergebnisse, die ich in diesem Sprint erreicht habe, vor dem Kunden präsentieren sollte.

Für das Sprint Review habe ich mich vorher mit meinem Betreuer zusammengesetzt und er hat mir erklärt, wie das Review abläuft und wie er sich darauf vorbereitet. Anhand dieser Informationen habe ich mir dann einen Plan gemacht, was ich wie vorstellen möchte. Dafür habe ich dann auf dem Development System Beispiel Kaufprojekte angelegt, um daran die Änderungen besser vorstellen zu können und einen reibungslosen Ablauf zu garantieren.

Nach dem Review folgte dann das Sprint Planning, in dem mir dann eine neue, etwas größere, Aufgabe zugeteilt wurde. Die Aufgabe besteht darin, in der ganzen Anwendung die Länder und Ländercodes in eine, von dem Kunden vorgegebene, einheitliche Reihenfolge zu bringen. Dafür musste ich zuerst eine Liste mit allen Vorkommnissen von Ländern in der Anmeldung machen und deren Ursprung im Code herausfinden.

Diese Aufgabe zu lösen, stellte sich als größere Herausforderung heraus als eigentlich gedacht, da die Ländercodes an viele Stellen auch als zusammengesetzter String in den Datenbanktabellen der Kaufprojekte stehen und so diese Werte, bei bestehenden Projekten, nicht so einfach neu sortiert werden können. Für dieses Problem konnte ich diese Woche allerdings noch keine Lösung finden.

2.5 Reihenfolge der Länder in der ganzen Anwendung ändern

Diese Woche widmete ich mich vollkommen der Aufgabe eine Lösung für die neue Sortierung aller Ländervorkommnisse in der Anwendung zu finden.

Das größte Problem war hierbei, dass Kaufprojekte zum einen so genannte countryCodes

beinhalteten, welche eine Zugehörigkeit zu diesen Ländern aussagt. Aber auch ein `countryLabel`, welches ein im Backend berechneter String ist, welcher die `countryCodes` mit einem Trennzeichen miteinander verknüpft und in der Datenbank speichert. Da es keine Option war, in der Datenbank alle `countryLabel`, mit der neuen Reihenfolge zu aktualisieren, musste ich hier im Backend die Rückgabe der Projekte modifizieren. Bevor diese dann an das Frontend gesendet wurden, mussten diese, getrennt, neu sortiert und dann wieder mit dem richtigen Trennzeichen zusammengefügt werden.

Ein weiteres Problem war, dass nicht nur Projekte eine Zugehörigkeit zu einem oder mehreren Ländern haben, sondern es viele verschiedene Entitäten gibt, welche einen Ländercode gespeichert haben. Dieser Code war allerdings in vielen Fällen unter einem anderen Namen in den Entitäten zu finden. Das machte es schwierig, eine einzelne Funktion zum Sortieren all dieser Sonderfälle zu erstellen. Dafür musste ich die Sonderfälle separat abfragen und je nach Entität ein anderes Attribut zum Sortieren verwenden.

2.6 Fertigstellung der Länderreihenfolge und eine neue Aufgabe

Am Anfang dieser Woche habe ich mich hauptsächlich damit beschäftigt, die letzten Fehler, die beim Testen des Features, mit dem ich mich die letzte Woche beschäftigt habe, aufgekommen sind, zu beheben.

Das größte Problem war, dass das Sortieren in manchen Fällen nicht so funktioniert hat wie es sollte. Das lag daran, dass wir uns dazu entschieden haben, sogenannte Streams, welche mehreren Ländern zugehörig waren, anhand des ersten Landes in der Liste der Länder zu sortieren. Das hatte zur Folge, dass, wenn es mehrere Streams in einem Kaufprojekt gibt, welche mit demselben Land anfangen und eine unterschiedliche Anzahl an Ländern hatten, die Sortierung nicht mehr richtig funktioniert hat. Um dieses Problem zu beheben, musste die Sortierfunktion angepasst werden. Hier entschied ich mich, zusammen mit meinem Betreuer, dazu die Länder, die mit demselben Land beginnen, anhand der Länge der Länderliste zu sortieren.

Am Mittwoch war dann unser Sprint Review, in dem ich wieder meine Ergebnisse der letzten zwei Wochen vor dem Project Ownern von dem Kunden vorstellen durfte.

Im Sprint Planning habe ich dann wieder eine neue Aufgabe bekommen. Diesmal sollte ich eine neue Funktionalität in dem, vor kurzem erst hinzugefügten, Admin-UI entwickeln.

Da ein ähnliches Feature bereits existierte, konnte ich mich sehr stark daran orientieren und so schnell das neue Feature implementieren. Das hatte allerdings zur Folge, dass viel duplizierter Code entstanden ist, den es zu reduzieren galt. Dafür musste ich viele Funktionen abstrahieren, welche von den alten Funktionalitäten und von dem neu implementierten genutzt wurden oder nahezu identisch waren.

2.7 Bugfixing und ein neues Feature

Diese Woche fing damit an, dass ich Bugs beheben musste, welche unserem Tester aufgefallen sind. Dies waren allerdings alles nur kleine Bugs die schnell behoben werden konnten oder für die es schon einmal an einer anderen Stelle aufgetreten sind und somit schon eine Lösung dafür existierte. Daher konnte ich alle Bugs am Montag bereits beheben und Dienstag mit der Implementierung eines neuen Features beginnen.

Für das neue Feature sollte ein neues Feld auf der Kaufprojekt Übersichtsseite hinzugefügt werden. Dieses Feld soll es den Benutzern erlauben, bei Kaufprojekten für Textil Produkte, mehr Informationen über die Materialien zu hinterlegen. Die Anforderungen waren, dass das Feld eine Mehrfachauswahl ermöglicht, also mehrere Werte ausgewählt werden können und dass es nur für bestimmte Kaufprojektarten und Benutzerrollen sichtbar ist.

Um dieses Feature umzusetzen, musste ich Änderungen am Front- und Backend vornehmen. Die Änderungen im Frontend waren schnell umgesetzt, da es dort nur darum ging ein neues Dropdown Feld hinzuzufügen und an ein Feld in der Datenbank zu binden. Dieses Feld musste dann nur noch unter bestimmten Konditionen ausgeblendet werden. Dank dem OData data-binding von SAPUI5 und SAP CAP war das sehr schnell umzusetzen.

Die größten Änderungen passierten im Backend, da dort eine neue Entität erstellt werden musste und diese mit den möglichen Auswahlmöglichkeiten für das Feld gefüllt werden musste. Außerdem musste ein neues one-to-many Feld in einer Datenbank Tabelle angelegt werden.

Nachdem dieses Feature fertig war und von unserem Tester abgenommen wurde, habe ich angefangen Unittest zu schreiben, da es für den aktuellen Sprint keine weiteren Tickets zu bearbeiten gibt.

2.8 Unittesting und Review Präsentation

Diese Woche begann mit dem Quarterly Review, zu dem der Kunde uns eingeladen hat. In diesem 3 Stunden Meeting wurden die Fortschritte und Erfolge des letzten Quartals der verschiedenen Softwareprodukte von dem Kunden vorgestellt. Da ein paar dieser Produkte von sovanta entwickelt werden – darunter auch das BuyingCockpit, an dem ich beteiligt bin – war unser Team auch zu diesem Meeting eingeladen.

Am Dienstag habe ich mich auf unser Sprint Review vorbereitet, welches am Nachmittag stattfand. Hier sollte ich wieder die Features vorstellen, die ich in diesem Sprint implementiert habe.

Neben den beiden Reviews habe ich mich hauptsächlich mit den Unittests beschäftigt, mit denen ich in der letzten Woche angefangen habe. Für das Unittesting musste ich mich jedoch zuerst mit den Vorgehensweisen meines Teams vertraut machen, denn für das Mocken der Datenbank Aufrufe wurden von dem Team zum Beispiel Funktionen geschrieben, um das Mocken einfacher zu machen.

Nachdem ich mit den Unittests, an denen ich gearbeitet habe, fertig war, habe ich die Aufgabe bekommen Regressionstest durchzuführen. Da ein Teamkollege in den letzten Wochen unser Backendframework SAP CAP auf eine neue Version zu updaten, mussten wir nun sicher gehen, dass die Anwendung mit der neuen Version noch genau so funktioniert wie vorher. Daher mussten wir alle Funktionen der Anwendung manuelle testen und dies dokumentieren. Unser Haupttester hat dafür eine Testdatei erstellt, in der die verschiedenen Testfälle, die getestet werden sollen, aufgelistet sind. Damit habe ich dann Freitag angefangen.

2.9 Regression- und Unittests

Diese Woche stand ganz im Zeichen des Testens. Am Anfang der Woche habe ich mich hauptsächlich mit den Regressionstests, mit denen ich schon in der letzten Woche begonnen habe, beschäftigt. Hierbei sind mir einige kleinere Fehler aufgefallen, über die ich dann mit unserem Tester diskutiert habe, ob für diese Fehler ein Bug-ticket erstellt werden sollte und wie kritisch diese sind. Danach habe ich dann für die relevanten Fehler ein Bug-ticket erstellt, damit dieser dann so schnell wie möglich behoben werden kann.

Ein Bug, den ich gefunden habe, war ein blockierender Bug, weshalb ich die Regressionstests fürs erste pausieren musste. In ein paar Meetings mit meinen Kollegen haben wir dann diesen Fehler investigiert und versucht eine Lösung für das Problem zu finden. Schlussendlich hat dann ein Seniorentwickler das Problem identifizieren und beheben können.

Danach konnte ich dann die Regressionstests für die neue SAP CAP-Version erfolgreich abschließen.

Da in ein paar Wochen der Release der neuen Version bevor steht, werden keine neuen Features mehr implementiert, weshalb ich mich den Rest der Woche nur noch mit weiteren Unittests beschäftigt habe.

2.10 Weitere Unittests und Beginn der Testphase

Diese Woche verlief sehr ähnlich zur letzten Woche. Am Anfang der Woche habe ich weiter an Unittests gearbeitet. Inzwischen habe ich die Vorgehensweise meines Teams deutlich besser verstanden und konnte so deutlich schneller und effizienter die Tests schreiben. Außerdem bekam ich durch das Unittesting ein noch besseres Verständnis der Architektur und der Codebase, da ich für das Testen gezwungen war die Funktionen richtig zu verstehen und auch welche Rolle diese im Großen und Ganzen spielen.

Mitte der Woche hatten wir dann unser letztes Review vor dem Release der neuen Version, da nun die Testphase beginnt.

Als erste Instanz in der Testphase, sollte unser Team Regressionstest durchführen und alle Features manuell testen, bevor die UAT-Tests mit Benutzern des Kunden durchgeführt werden.

Hierfür wurden die verschiedenen Benutzer- und Ländergruppen zwischen allen Teammitgliedern aufgeteilt und bekamen dann Testfälle, die wir dann testen sollten. Mir wurden dann zwei Ländergruppen zugeteilt, die ich dann auf dem PRE-System testen sollte. Mit der ersten Ländergruppe beschäftigte ich mich dann den Rest der Woche.

2.11 Bugfixing und Unittests

Diese Woche hat damit angefangen, dass ich einen Bug welches mit einem Feature, welches ich vor ein paar Wochen implementiert habe, zusammenhing, beheben musste. Das Problem war, dass das Filtern auf der Übersichtsseite für manche Felder nicht mehr richtig funktionierte. Dies passierte, da mit einem neuen Feature, welches wir implementiert haben, die Funktionsweise wie wir standardmäßig die Projekte filtern, grundlegend verändert wurde. Somit funktionierten einige individuelle Filter nicht mehr und mussten an die neue funktionsweiße angepasst werden.

Nachdem ich diesen Bug beheben konnte, setzte ich mich wieder daran neue Unittests für verschiedene Dateien zu schreiben. Das habe ich dann den Rest der Woche gemacht und auch die Woche damit beendet.

2.12 Komplizierteres Unittesting

Diese ganze Woche war geprägt von weiteren Unittests. Jedoch war die Datei, welche ich diese Woche getestet habe, deutlich anspruchsvoller zu testen. Das Problem war, dass dies eine große Datei war mit vielen großen Funktionen, aber jedoch nur eine Funktion exportiert wurde und der Rest lokal war. Somit konnte ich nicht jede Funktion separat testen, sondern musste alle Funktionen indirekt über die eine exportierte Funktion abdecken. Das hatte zu folge, dass es deutlich schwieriger war, eine 100% Abdeckung aller Pfade zu erreichen.

Ich find damit an den gesamten Code zu analysieren und zu verstehen war dort genau passiert und was die Aufgabe des Codes ist. Nachdem ich das gemacht habe, konnte ich mir einen Plan machen, was für Fälle alle eintreten können, und welche getestet werden müssen. Als das erledigt war, musste ich mir überlegen, welche Daten ich mocken muss, um die verschiedenen Testfälle abzudecken. Nun musste ich nur noch alles zusammenfügen und die Testfälle mit den Mock Daten implementieren.

Am Ende der Woche konnte ich dann den Merge Request für diese etwas aufwendigeren Unittests erstellen.

2.13 Unittesting und der Start einer neuen Version

Am Anfang dieser Woche habe ich mich hauptsächlich mit einigen Unittests beschäftigt, um die restliche Zeit in diesem Sprint sinnvoll zu nutzen. Außerdem hatten wir am Montag ein Meeting, in dem die Ziele für die neue Version der Software besprochen wurden.

Am Mittwoch hat dann der neue Sprint, und damit auch die Entwicklung für die neue Version begonnen. Mir wurde dann ein neues Ticket zugeteilt, in dem es darum geht, ein neues Feld auf der Länderebene zu erstellen. Dieses Feld soll einen Einfluss auf den Workflow des Projektes haben, dafür muss das Feld mit an den Workflow Modulator übergeben werden. Diese Woche habe ich mich hauptsächlich damit beschäftigt herauszufinden, was geändert werden muss, um das Feature zu implementieren.

Am Donnerstag hatte ich noch meinen Onboarding-Day bei sovanta, an dem wurden uns Infos zu sovanta gegeben, Workshops gehalten und wir konnten viele neue Leute kennen lernen.

2.14 Probleme bei der Feature Implementierung

In dieser Woche habe ich mich zum großen Teil damit beschäftigt, das Feature, an dem ich in der letzten Woche abgefangen habe, fertig zu stellen.

Beim Versuch das Feature zu implementieren, ist mir allerdings ein Problem aufgefallen. Das Problem war, dass das Feld laut den Akzeptanzkriterien im Frontend drei Auswahlmöglichkeiten haben sollte, „Ja“, „Nein“ und „-“, (leer). Da allerdings im Workflow Modulator von dem anderen Team das Feld mit einem Boolean modelliert wurde, machte es das schwierig drei Auswahlmöglichkeiten zu implementieren.

Um dies zu lösen hatten wir nur zwei Möglichkeiten. Entweder das Feld auf der Anwendungsseite und der Workflow Modulator Seite als eine Assoziation zu modellieren. Dafür hätte jedoch das Team, welches für den Workflow Modulator zuständig ist, größere Änderungen vornehmen müssen. Die andere Möglichkeit, für welche wir uns dann, nach Absprache mit den verantwortlichen Personen, auch entschieden haben, war es das Feld nur auf der Anwendungsseite als eine Assoziation zu modellieren und dann bevor es an den Workflow Modulator gesendet wird einen Boolean zu mappen.

Nachdem wir diesen Lösungsansatz gefunden haben, war die Implementierung schnell umgesetzt und konnte dann auf das DEV-System deployed werden, um dort noch einmal ausführlich getestet zu werden.

2.15 Ende der Implementierungsphase und Begin der Regressionstests

Nachdem ich Ende letzter Woche mein Feature für diese Version erfolgreich deployen konnte hieß es für mich jetzt wieder Unittests schreiben, bis der Sprint Mitte der Woche zu Ende war.

Am Ende des Sprints hatte wir wieder ein Review, in dem ich das Feature, an welchem ich gearbeitet habe, vorstellen durfte. Als Vorbereitung auf das Review habe ich mir im Vorfeld mehrere Testprojekte auf dem DEV-System erstellt und dort getestet, ob das Feature auch einwandfrei funktioniert und bereit für die Präsentation ist.

Mit Beginn des neuen Sprints begannen dann auch die Regressionstests für die neue Version. Während der Durchführung der Tests sind mir ein paar Defekte aufgefallen, für welche ich dann ein Ticket erstellt habe und mich mit Kollegen darüber ausgetauscht habe.

Leider konnten wir die Regressionstests diese Woche noch nicht abschließen, da einige Features aktuell nicht testbar waren und wir darauf warten mussten, bis ein anderes Team den Fehler behoben hat.

2.16 Unittesting und Beginn des Refactorings

Diese Woche habe ich damit begonnen, dass ich die Unittests, an denen ich vor dem Beginn der Regressionstests gearbeitet habe, fortgesetzt habe. An diesen habe ich dann noch bis Mitte der Woche gearbeitet.

Nachdem ich mit den Unittests fertig haben wir beschlossen, dass wir nun mit dem Refactoring des Backend Quellcodes anfangen möchten. Dafür wurden bereits einige Tickets für Dateien oder Funktionsweisen, welche überarbeitet werden sollen, erstellt.

Mit wurde ein Ticket zugewiesen, in dem es darum ging die Logik und die Werte für das befüllen verschiedener Formulare mit Standardwerten vom Frontend in das Backend zu verlegen. Dies sollte eine bessere Wartbarkeit dieser Standardwerte für die Zukunft zu schaffen.

Dafür musste ich mir zuerst einen Überblick verschaffen, wo diese Werte im Frontend gesetzt werden und welches Land mit welchen Werten befüllt wird, das ist nämlich für jedes Land unterschiedlich. Am Ende der Woche hatte ich dann einen guten Überblick, was geändert werden muss und welche Werte zu welchem Land gehören.

2.17 Fertigstellung des Standardwert-Refactorings und weitere Refactorings

Diese Woche fing damit an das Refactoring, mit welchem ich in der letzten Woche begonnen habe, fertig zu stellen. Nachdem ich in der letzten Woche alles ausführlich analysiert habe, konnte ich nun mit der Implementierung beginnen.

Dafür fing ich an im Backend eine neue Entität zu erstellen, welche die Standardwerte für jedes Land beinhaltet. In dieser Entität werden zu jedem Land die entsprechenden IDs der Werte gespeichert. Da aber im Frontend nicht nur die IDs benötigt werden, sondern teilweise auch die Werte die zu den IDs gehören mussten all diese Tabellen beim abfragen der Werte gejoint werden.

Nachdem die Änderungen im Backend fertig waren, konnte ich im Frontend eine Funktion schreiben, welche die Werte aus dem Backend, anhand der Länderzugehörigkeit des Users, abfragt und dann das Formular mit diesen Werten auffüllt. Mit diesen Änderungen konnte das Refactoring dann auch deployed werden und ausführlich getestet werden.

2.18 Refactoring und Unittests

Diese Woche habe ich damit angefangen, eine Datei zu analysieren und zu schauen was in diese Datei überarbeitet werden könnte. Dafür habe ich mir dann eine Liste mit allen Problemen oder Optimierungsmöglichkeiten erstellt um einen Überblick zu behalten und

um damit das Ticket für das Refactoring zu erstellen.

Nachdem ich das Ticket für das Refactoring erstellt habe, konnte ich damit anfangen die Änderungen um zu setzen.

Dadurch das während dem Refactoring viele tote Codestücke gelöscht wurden, liefen die Unittests für diese Datei nicht mehr fehlerfrei durch. Nach einer Absprache mit meinem Betreuer, sind wir zu dem Entschluss gekommen, dass es am sinnvollsten ist, die Unittests für diese Datei komplett neu zu schreiben, da sich zu viel geändert hat und die Tests somit keinen Sinn mehr ergeben haben.

Somit fing ich an die Unittests für die Überarbeitete Datei komplett neu zu schreiben.

3 Projektbericht

3.1 Zusammenfassung

Dieser Bericht dokumentiert die Entwicklung und Implementierung einer eigenständigen Admin-Seite für eine Webanwendung, die es Endnutzern ermöglicht, administrative Aufgaben unabhängig von technischen Entwicklern durchzuführen. Vor der Einführung dieser Lösung waren Änderungen an den Datenbankinhalten manuell und zeitintensiv, was zu Verzögerungen und erhöhten Kosten führte.

Die neue Admin-Seite erlaubt den Nutzern, Projekte zu löschen, Informationen zu aktualisieren und andere administrative Tätigkeiten selbstständig zu erledigen. Dabei wurde ein besonderer Fokus auf die Umstellung von JavaScript auf TypeScript im Frontend gelegt, um die Codequalität zu verbessern und den Entwicklungsprozess effizienter zu gestalten.

Die Implementierung der Admin-Seite führte zu einer spürbaren Effizienzsteigerung und einer Reduzierung der Abhängigkeit von den Entwicklern. Diese Arbeit beleuchtet die Herausforderungen während der Entwicklung sowie die ergriffenen Lösungen und zeigt, wie die neue Admin-Seite zu einer signifikanten Verbesserung der internen Abläufe beigetragen hat.

3.2 Einleitung

In Umfeld einer Webanwendung ist die Pflege der dazugehörigen Daten ein essenzieller Bestandteil. Diese Pflege der Daten musste bisher manuell von den Entwicklern übernommen werden, indem Änderungen direkt in der Datenbank vorgenommen wurden. Für den Kunden führte diese Abhängigkeit von den Entwicklern zu einer Reihe von Problemen.

Zum einen entstanden Verzögerungen, da selbst kleine Änderungen in den Daten, wie z.B. das Aktualisieren von Projektdetails oder das Löschen von nicht mehr benötigten Einträgen, oft Wartezeiten zur Folge hatten, bis die Entwickler verfügbar waren. Dies war besonders problematisch bei einer hohen Auslastung der Entwickler oder bei dringenden Korrekturen. Zum anderen verursachte dieses Vorgehen unnötige Kosten, da für jede Änderung technisches Personal hinzugezogen werden musste, auch wenn die Aufgaben relativ trivial

waren.

Darüber hinaus war die Fehleranfälligkeit durch manuelle Datenbankänderungen erhöht, da Änderungen ohne Benutzeroberfläche eine präzise Kenntnis der Datenbankstruktur voraussetzten. Jede kleine Ungenauigkeit konnte zu Dateninkonsistenzen oder anderen Problemen führen, die zusätzlichen Aufwand erforderte.

Um diese Probleme zu beseitigen und den Arbeitsfluss des Kunden zu verbessern, wurde in diesem Projekt eine eigenständige Admin-Seite entwickelt, die es den Endnutzern ermöglicht, administrative Aufgaben selbstständig und effizient durchzuführen. Damit wird das Unternehmen des Kunden unabhängiger von Entwicklern, was zu einer schnelleren Bearbeitung von Aufgaben und geringeren Kosten führt.

Das Ziel dieser Arbeit ist es, die Entwicklung und Implementierung dieser Admin-Seite zu dokumentieren und die aufgetretenen Herausforderungen sowie deren Lösungen zu beschreiben. Ein besonderer Fokus liegt dabei auf der Umstellung von JavaScript auf TypeScript im Frontend, um dort die Codequalität und Wartbarkeit der Codebase zu verbessern.

Die neue Admin-Seite soll es den Nutzern ermöglichen, Projekte zu löschen, Informationen zu aktualisieren und andere administrative Aufgaben eigenständig durchzuführen, ohne auf die Hilfe von Entwicklern angewiesen zu sein. Dies trägt zu einer spürbaren Effizienzsteigerung im Betrieb des Kunden bei.

3.3 Technologische Grundlagen

3.3.1 Einführung in TypeScript

TypeScript ist eine Open-Source-Programmiersprache, welche erstmals 2012 von Microsoft veröffentlicht wurde [1]. TypeScript ist ein Superset von JavaScript und erweitert JavaScript um eine statische Typisierung und andere erweiterte Funktionen. Das Ziel von TypeScript ist es, die Entwicklung von großer und komplexer JavaScript-Anwendungen einfacher und sicherer zu machen [3].

Warum TypeScript?

JavaScript ist dynamisch typisiert, was bedeutet, dass Variablen während der Laufzeit jeden beliebigen Typ annehmen können. Das kann zu Laufzeitfehlern führen, welche schwer zu debuggen sind. In TypeScript wird dieses Problem gelöst, indem es den Entwicklern ermöglicht, Typen explizit zu definieren, was mehrere Vorteile bietet:

- **Fehlererkennung zur Entwicklungszeit:**

Durch die statische Typisierung werden viele Fehler bereits während dem Schreiben des Codes erkannt und nicht erst zur Laufzeit des Programms.

- **Verbesserte Integrated Development Environment (IDE)-Unterstützung:**

Die Entwicklungsumgebung wird von TypeScript durch eine Autovervollständigung, Refactoring-Tools und IntelliSense erheblich verbessert. [2]

- **Bessere Dokumentation:**

Durch die Typenannotation ist TypeScript Code selbstdokumentierend, was die Verständlichkeit und Wartbarkeit erhöht.

- **Skalierbarkeit:**

Die Entwicklung und Wartung großer Codebasen wird durch klare Typdefinitionen und Modulunterstützungen verbessert.

Grundlegende Konzepte

- **Typen:**

TypeScript erweitert JavaScript um ein starkes Typensystem. Einiger der grundlegenden Typen sind im folgenden Listing gezeigt.

```
1 var isDone: boolean = true;
2 var count: number = 42;
3 var name: string = "Alice";
4 var list: number[] = [1, 2, 3];
```

Listing 3.1. Beispiel: Typen in TypeScript

- **Interfaces:**

Interfaces definieren die Struktur eines Objekts und können sicherstellen, dass Objekte diese Form einhalten.

```
1 interface Person {
2     firstName: string;
3     lastName: string;
4 }
5
6 function greet(person: Person) {
7     return "Hello, " + person.firstName + " " + person.lastName;
8 }
9
10 let user = { firstName: "Jane", lastName: "Doe" };
11 console.log(greet(user)); // "Hello, Jane Doe"
```

Listing 3.2. Beispiel: Interfaces in TypeScript

- **Klassen:**

TypeScript unterstützt objektorientierte Programmierung durch Klassen, was Vererbung, Kapselung und andere Objektorientierte Programmierung (OOP)-Konzepte ermöglicht.

```
1 class Animal {
2     name: string;
3
4     constructor(name: string) {
5         this.name = name;
6     }
7     move(distance: number = 0) {
8         console.log(`${this.name} moved ${distance}m.`);
9     }
10 }
11
12 class Dog extends Animal {
13     bark() {
```

```
14     console.log("Woof! Woof!");
15   }
16 }
17
18 let dog = new Dog("Buddy");
19 dog.bark();
20 dog.move(10);
```

Listing 3.3. Beispiel: Klassen in TypeScript

- **Generics:**

Generics ermöglichen die Erstellung wiederverwendbarer Komponenten, die mit verschiedenen Typen arbeiten können.

```
1 function identity<T>(arg: T): T {
2     return arg;
3 }
4
5 let output1 = identity<string>("myString");
6 let output2 = identity<number>(100);
```

Listing 3.4. Beispiel: Generics in TypeScript

3.3.2 Unterschiede zwischen JavaScript und TypeScript

Da TypeScript auf JavaScript aufbaut, gibt es viele Gemeinsamkeiten zwischen den beiden Sprachen. Jedoch gibt es zwei wesentliche Punkte, in denen sich die Sprachen grundsätzlich unterscheiden:

1. **Statische Typisierung**

In JavaScript werden Variablen dynamisch typisiert und Typen zur Laufzeit überprüft. In TypeScript werden diese statisch typisiert, was es Entwicklern erlaubt, Typen explizit zu deklarieren und zur Kompilierzeit zu überprüfen. Dies reduziert die Wahrscheinlichkeit von Typfehlern und verbessert die Codequalität.

```
1 // JavaScript
2 let count = 42; // Dynamisch typisiert
3
4 // TypeScript
5 let count: number = 42; // Statisch typisiert
```

2. **Typanmerkungen und -inferenz**

In TypeScript können Typen explizit angegeben werden oder von der TypeScript-Engine automatisch inferiert werden. Diese Inferenz ermöglicht eine flexible Typisierung, welche sowohl die Sicherheit als auch die Bequemlichkeit erhöht.

```
1 let message: string = "Hello, World!"; // Explizite Typanmerkung
2 let count = 42; // Typinferenz: TypeScript erkennt automatisch, dass count
  vom Typ number ist
```

3.3.3 Vorstellung weiterer verwendeter Technologien und Frameworks

Das SAPUI5 Framework

SAPUI5 ist ein auf JavaScript basierendes UI-Framework, um plattformübergreifende Webanwendungen für Unternehmen effizient zu entwickeln.

Es wurde 2011 von SAP entwickelt und wird bis heute weiterentwickelt. Seit 2013 gibt es von dem Framework auch eine Open-source-Variante namens OpenUI5. Der Hauptunterschied der beiden Frameworks ist allerdings nur die Lizenz.

Das Hauptziel des Frameworks ist es, eine konsistente, reaktionsschnelle und intuitive Benutzererfahrung über alle SAP-Anwendungen hinweg zu bieten, die sowohl auf Desktops als auch auf mobilen Geräten reibungslos funktioniert.

Grundlegende Architektur von SAPUI5

SAPUI5 basiert auf der Model-View-Controller (MVC) Architektur [4], ein gängiges Entwurfsmuster, welches eine klare Trennung der verschiedenen Anwendungskomponenten gewährleistet:

- **Model:**

Das Model repräsentiert die Datenstruktur und -logik der Anwendung. Es beinhaltet die Daten, welche aus unterschiedlichen Quellen geladen werden können, wie zum Beispiel OData-Services, JSON-Dateien oder anderen APIs. Diese Quellen können dann durch sogenannte Bindings direkt in das View eingebunden werden und von dort auf die Daten zugegriffen und in diese geschrieben werden.

- **View:**

Die View beschreibt das Layout und Design der Benutzeroberfläche. SAPUI5 bietet verschiedene Ansätze für die Definition der Views, darunter XML, HTML, JSON und JavaScript. Der beliebteste Ansatz sind die XML-Views, da sie die Strukturierung und Wiederverwendbarkeit fördern.

- **Controller:**

Der Controller verarbeitet die Benutzereingaben und steuert die Geschäftslogik. Er dient als Vermittler zwischen dem Model und der View, indem er die Daten abrufen und an die View weiterleitet oder Benutzereingaben verarbeitet und entsprechende Aktionen

Diese Trennung ermöglicht eine bessere Wartbarkeit und Skalierbarkeit von Anwendungen und erleichtert die Zusammenarbeit in Entwicklerteams.

Vorteile und Schlüsselmerkmale von SAPUI5

SAPUI5 bietet eine Reihe von Vorteilen und Funktionen, die es zu einem leistungsstarken Framework für die Entwicklung von Unternehmensanwendungen machen:

- **Responsive Design:**

Durch SAPUI5 werden Anwendungen automatisch an verschiedene Bildschirmgrößen angepasst, wodurch sie auf Desktop-Computern und mobilen Geräten gut funktionieren.

- **UI-Komponenten:**

Das Framework bietet eine umfangreiche Bibliothek von vorgefertigten UI-Komponenten wie Tabellen, Diagrammen, Formularen und vielem mehr. Diese Komponenten sind dafür ausgelegt, eine einheitliche Benutzererfahrung zu gewährleisten und Anwendungen effizienter entwickeln zu können.

- **Internationalisierung und Lokalisierung:**

Mit SAPUI5 können Anwendungen mithilfe des sogenannten i18n-Modells an verschiedene Sprachen und Regionen angepasst werden.

- **Integration mit SAP-Systemen:**

Das Framework ist speziell darauf ausgelegt, nahtlos mit SAP-Backend-Systemen zu arbeiten, insbesondere mit OData-Services.

SAPUI5 in Verbindung mit TypeScript

SAPUI5 in TypeScript ist eine relativ neue Erweiterung zu SAPUI5, die 2021 von SAP veröffentlicht wurde und immer weiter verbessert wird. Sie bringt alle Vorteile, die TypeScript gegenüber JavaScript mitbringt, zu SAPUI5, was die Entwicklung der Anwendungen stark verbessert.

Unterschied zwischen JavaScript und TypeScript in SAPUI5 anhand eines Beispiels

Im Folgenden werden die Unterschiede eines SAPUI5 BaseControllers in JavaScript und TypeScript gezeigt.

JavaScript:

In JavaScript werden SAPUI5 Controller über eine interne Funktion definiert. Diese erwartet ein Array mit Pfaden zu Controllern, die importiert werden sollen, und einer Callback-Funktion, welche die Funktionalität des Controllers definiert.

```
1 sap.ui.define([
2     "sap/ui/core/mvc/Controller"
3 ], function (Controller) {
4     "use strict";
5
6     return Controller.extend("de.example.controller.BaseController", {
7         getRouter: function () {
8             return this.getOwnerComponent().getRouter();
9         },
10        getModel: function (sName) {
11            return this.getView().getModel(sName);
12        }
13    })
14 })
```

Listing 3.5. Beispiel: JavaScript BaseController.js

TypeScript:

Anders als in JavaScript werden Controller in TypeScript über Klassen definiert. Damit der TypeScript Kompilier die Klasse jedoch in einen SAPUI5 Controller konvertieren kann, muss ein Kommentar mit dem passenden namespace hinzugefügt werden. Controller können mit einem TypeScript Import importiert werden.

```
1 import Controller from "sap/ui/core/mvc/Controller";
2
3 /**
4  * @namespace de.example.controller
5  */
6 export default class BaseController extends Controller {
7     public getRouter(): Router {
8         return (this.getOwnerComponent() as UIComponent).getRouter();
9     }
10
11     public getModel(name?: string): Model {
12         this.getView().getModel(name);
13     }
14 }
```

Listing 3.6. Beispiel: TypeScript BaseController.ts

3.4 Anforderungsanalyse

In diesem Kapitel wird eine kurze Analyse der aktuellen Situation durchgeführt und die Anforderungen an das Admin-UI aufgeführt. Zunächst wird der Ist-Zustand beschrieben, um die Probleme und Herausforderungen zu identifizieren. Danach werden die Anforderungen aufgestellt, die zur Lösung des Problems erforderlich sind.

3.4.1 Ist-Zustand

Bei der Webanwendung, für die das Admin-UI entwickelt wird, handelt es sich um eine Plattform, die es dem Kunden ermöglicht, Kaufprojekte zu erstellen und den Benutzer Schritt für Schritt durch die notwendigen Aufgaben zu führen, die für die Umsetzung dieses Projekts erforderlich sind. Jedoch gibt es ein Problem mit der Art und Weise, wie Änderungen an den Projektdaten durchgeführt werden. Derzeit können Änderungen ausschließlich über direkte Structured Query Language (SQL)-Abfragen in der Datenbank von einem Entwickler vorgenommen werden.

Ein Beispiel für diese Daten betrifft Kaufprojekte, bei denen regelmäßig administrative Aufgaben anfallen, wie z.B.:

- Das Löschen von Kaufprojekten, die nicht mehr relevant sind,
- Das Ändern des Verantwortlichen für bestimmte Kaufprojekte, wenn sich Zuständigkeiten im Team ändern.

Das größte Problem an dieser Herangehensweise ist, dass die Nutzer des Systems vollständig auf Entwickler angewiesen sind, um selbst kleinere Änderungen durchführen zu können. Dies führt zu unnötigen Verzögerungen im Arbeitsablauf, da die Entwickler nicht immer sofort verfügbar sind. Zudem entsteht für die Entwickler vermeidbare Arbeit, da sie Aufgaben übernehmen müssen, die prinzipiell von den Endnutzern selbst erledigt werden könnten.

Dieses Problem zeigt sich insbesondere in Situationen, in denen schnelle Anpassungen benötigt werden, um auf sich ändernde Anforderungen im Projektgeschäft zu reagieren, etwa wenn kurzfristig ein Kaufprojekt entfernt oder der zuständige Mitarbeiter aktualisiert werden muss.

Um dieses Problem zu lösen, wird in diesem Projekt eine eigenständige Admin-Seite entwickelt. Diese Admin-Seite ist eine Erweiterung der bestehenden Anwendung und soll es den Endnutzern ermöglichen, administrative Aufgaben, wie das Löschen oder Bearbeiten von Projektdaten, ohne die Hilfe von Entwicklern durchzuführen. So wird die Abhängigkeit von Entwicklern reduziert und der Arbeitsfluss des Kunden effizienter gestaltet.

3.4.2 Anforderungen

Dieses Kapitel befasst sich mit den Anforderungen an das Admin-UI, welche von dem Kunden zu Beginn des Projektes bereits verfasst wurden. Die Funktionalität aller Seiten des Admin-UIs wurden in diesen Anforderungen festgehalten und dienen dann ebenfalls als Akzeptanzkriterien.

A1 - Reaktionszeit der Anwendung

Tabelle 3.1. Anforderung A1 - Reaktionszeit der Anwendung

A1	Reaktionszeit der Anwendung
Nicht-Funktional	Die Anwendung soll in einer angemessenen Zeit antworten.

A2 - Sprache der Anwendung

Tabelle 3.2. Anforderung A2 - Sprache der Anwendung

A2	Sprache der Anwendung
Nicht-Funktional	Die Anwendung soll je nach Standort oder Einstellung des Nutzers in Deutsch oder Englisch angezeigt werden.

A3 - Intelligente Vorschläge für Eingabefelder

Tabelle 3.3. Anforderung A4 - Intelligente Vorschläge für Eingabefelder

A3	Intelligente Vorschläge für Eingabefelder
Funktional	Eingabefelder sollen dem Nutzer Vorschläge anhand der bereits eingegeben Zeichen machen. Die Vorschläge sollen sich bei der Eingabe oder Löschung von Zeichen aktualisieren.

A4 - Eingabefelder der Seite zum Löschen der Kaufprojekte

Tabelle 3.4. Anforderung A4 - Eingabefelder der Seite zum Löschen der Kaufprojekte

A4	Eingabefelder der Seite zum Löschen der Kaufprojekte
Funktional	Die Seite hat ein Eingabefeld, in dem eine oder mehrere ProjektIDs, die gelöscht werden sollen, eingetragen werden können. Das Feld ist ein Pflichtfeld und muss ausgefüllt werden.

A5 - Eingabefelder der Seite zum Aktualisieren des Projektowners

Tabelle 3.5. Anforderung A4 - Eingabefelder der Seite zum Aktualisieren des Projektowners

A5	Eingabefelder der Seite zum Aktualisieren des Projektowners
Funktional	<p>Die Seite hat drei Eingabefelder:</p> <ul style="list-style-type: none"> • Der alte Projektowner (Pflichtfeld) • Der neue Projektowner (Pflichtfeld) • Eine oder mehrere Sub Commodity Groups (SCGs) (Optional) <p>Pflichtfelder müssen ausgefüllt werden. Optionale Felder müssen keinen Wert beinhalten.</p>

A6 - Eingabefelder der Seite zum Aktualisieren des Projektmanagers und der Käufergruppe

Tabelle 3.6. Anforderung A6 - Eingabefelder der Seite zum Aktualisieren des Projektmanagers und der Käufergruppe

A6	Eingabefelder der Seite zum Aktualisieren des Projektmanagers und der Käufergruppe
Funktional	<p>Die Seite hat fünf Eingabefelder:</p> <ul style="list-style-type: none"> • Der alte Projektmanager (Pflichtfeld) • Der neue Projektmanager (Pflichtfeld) • Die alte Käufergruppe (Pflichtfeld) • Der neue Käufergruppe (Pflichtfeld) • Eine oder mehrere SCGs (Optional) <p>Pflichtfelder müssen ausgefüllt werden. Optionale Felder müssen keinen Wert beinhalten.</p>

A7 - Speichern und Verwerfen Aktionen

Tabelle 3.7. Anforderung A7 - Speichern und Verwerfen Aktionsleiste

A7	Speichern und Verwerfen Aktionsleiste
Funktional	<p>Auf jeder Seite des Admin-UIs soll eine Aktionsleiste mit zwei Knöpfen vorhanden sein:</p> <ul style="list-style-type: none"> • Speichern: Öffnet ein Pop-Up in dem der Nutzer seine Wahl bestätigen muss. Bei Bestätigung wird die jeweilige Aktion der Seite durchgeführt. Der Knopf ist nur Verwendbar, wenn alle Pflichtfelder ausgefüllt sind. • Verwerfen: Setzt alle Eingabefelder zurück.

Tabelle 3.8. Anforderung A7.1 - Speicheraktion auf der Seite zum Löschen der Kaufprojekte

A7.1	Speicheraktion auf der Seite zum Löschen der Kaufprojekte
Funktional	<p>Wird die Speicheraktion ausgeführt sollen alle ausgewählten Kaufprojekte in der Datenbank als gelöscht markiert werden und dürfen danach nicht mehr angezeigt werden.</p>

Tabelle 3.9. Anforderung A7.2 - Speicheraktion auf der Seite zum Aktualisieren des Projektowners

A7.2	Speicheraktion auf der Seite zum Aktualisieren des Projektowners
Funktional	<p>Wird die Speicheraktion ausgeführt, soll der alte Projektowner mit dem neuen Projektowner in allen Projekten ersetzt werden. Diese Aktualisierung soll auf Projekt und Länder -Ebene geschehen.</p> <p>Für den Fall, dass eine oder mehrere SCGs mit angegeben wurden, beschränkt sich die Aktualisierung auf Projekte, die mit den angegebenen SCGs übereinstimmen.</p>

Tabelle 3.10. Anforderung A7.3 - Speicheraktion auf der Seite zum Aktualisieren des Projektmanagers und der Käufergruppe

A7.3	Speicheraktion auf der Seite zum Aktualisieren des Projektmanagers und der Käufergruppe
Funktional	<p>Wird die Speicheraktion ausgeführt, soll der alte Projektmanager und die alte Käufergruppe mit dem neuen Projektmanager und der neuen Käufergruppe in allen Projekten ersetzt werden. Diese Aktualisierung soll auf Projekt und Länder -Ebene geschehen.</p> <p>Für den Fall, dass eine oder mehrere SCGs mit angegeben wurden, beschränkt sich die Aktualisierung auf Projekte, die mit den angegebenen SCGs übereinstimmen.</p>

3.5 Lösungskonzept

3.5.1 Designkonzept

Das Design des Admin-UIs wurde von einem separaten Designerteam erstellt, welches nah mit dem Kunden zusammengearbeitet hat.

Die Mockups orientieren sich an den Anforderungen A4, A5, A6 und A7. Sie fügen jedoch noch andere Elemente hinzu, um für eine gute User Experience (UX) zu sorgen. Dazu gehören:

- Die Unterteilung der Eingabefelder mit der dazu passenden Überschrift.
- Platzhalter in den Eingabefeldern, um besser zu erkennen, was in die Felder eingetragen werden muss.

Anhand der Kundenanforderungen und den UX-Anforderungen hat wurden von dem Designteam einige Mockups für das Admin-UI erstellt, welche dem Entwicklerteam als Leitbild dienen sollte.

Notiz: Um die Anonymität des Kunden zu wahren, wurden einige Stellen der Mockups geschwärzt.

Mockups für das Admin-UIs

BUYING PROJECTS ▾

🔍 🔔 ⚙️

BUYING COCKPIT - ADMIN USER INTERFACE

VERSION
1.5

PROJECT OWNER >

BUYING MANAGER /
PURCHASING GROUP >

DELETE BUYING
PROJECT >

PROJECT OWNER

OLD PROJECT OWNER*
MAX.MUSTERMANN@.COM

NEW PROJECT OWNER*
MAXIMILIAN.MUSTERMANN@.COM

SCGS

SCGS
00010101.00010102...

SAVE CHANGES

CLEAR

Abbildung 3.1. Mockup: Admin-UI Projekowner Seite

3 Projektbericht

BUYING PROJECTS ▾

🔍 🔔 ⚙️

BUYING COCKPIT - ADMIN USER INTERFACE

VERSION
1.5

PROJECT OWNER >

BUYING MANAGER / PURCHASING GROUP >

DELETE BUYING PROJECT >

BUYING MANAGER / BUYING DIRECTOR

OLD BUYING MANAGER*

MAX.MUSTERMANN@...COM

NEW BUYING MANAGER*

MAXIMILIAN.MUSTERMANN@...COM

PURCHASING GROUP

OLD PURCHASING GROUP*

G33

NEW PURCHASING GROUP*

G100

SCGS

SCGS

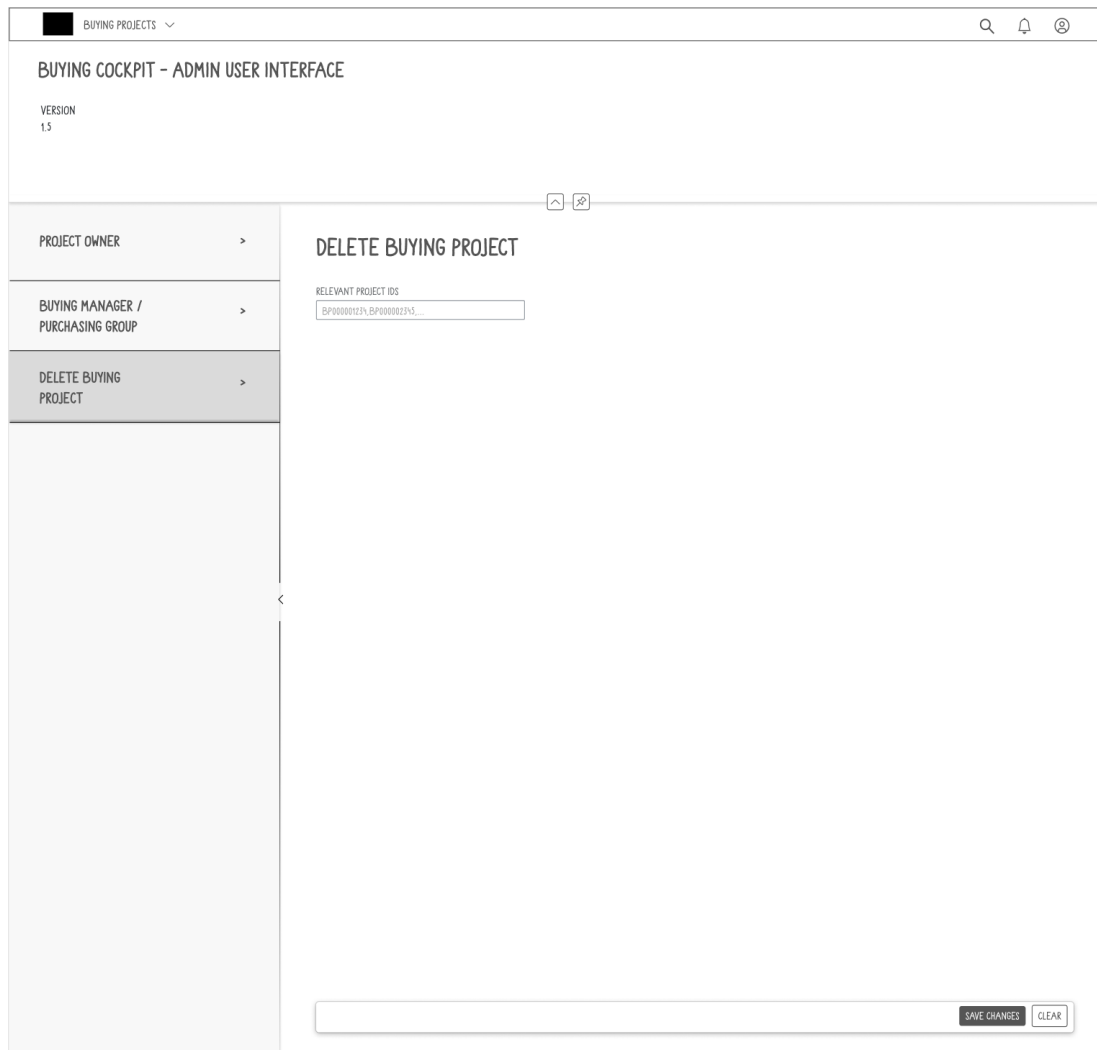
00010101.00010102...

SAVE CHANGES

CLEAR

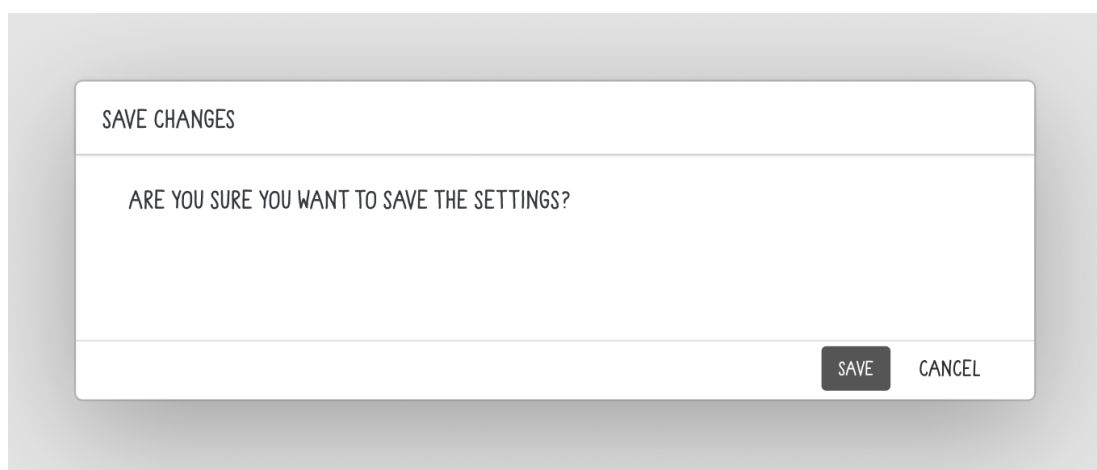
Abbildung 3.2. Mockup: Admin-UI Projektmanager und Käufergruppe Seite

3 Projektbericht



The mockup shows a web interface for 'BUYING COCKPIT - ADMIN USER INTERFACE'. At the top, there is a header bar with a logo, the text 'BUYING PROJECTS', and search, notification, and user icons. Below the header, the page title 'BUYING COCKPIT - ADMIN USER INTERFACE' and version '1.5' are displayed. A sidebar on the left contains three menu items: 'PROJECT OWNER', 'BUYING MANAGER / PURCHASING GROUP', and 'DELETE BUYING PROJECT', each with a right-pointing arrow. The 'DELETE BUYING PROJECT' item is highlighted. The main content area is titled 'DELETE BUYING PROJECT' and includes a section for 'RELEVANT PROJECT IDS' with a text input field containing 'BP0000001234_BP0000001234...'. At the bottom right of the main area, there are two buttons: 'SAVE CHANGES' and 'CLEAR'.

Abbildung 3.3. Mockup: Admin-UI Löschungs-Seite



The mockup shows a modal dialog box with a title bar 'SAVE CHANGES'. The main text inside the dialog asks 'ARE YOU SURE YOU WANT TO SAVE THE SETTINGS?'. At the bottom right, there are two buttons: 'SAVE' and 'CANCEL'.

Abbildung 3.4. Mockup: Admin-UI Pop-Up

3.5.2 Software-Design

Im folgenden Kapitel wird das Software-Design des Admin-UI und der dazu gehörenden Anwendung näher beschrieben. Es bietet einen Überblick über die Architektur der Anwendung, die verwendeten Technologien und die Struktur der Daten.

Grundlegende Architektur

Die grundlegende Architektur der Anwendungen wird in Abbildung 3.5 etwas vereinfacht dargestellt, Komponenten, welche nicht relevant für die Entwicklung des Admin-UIs sind, wurden ausgelassen.

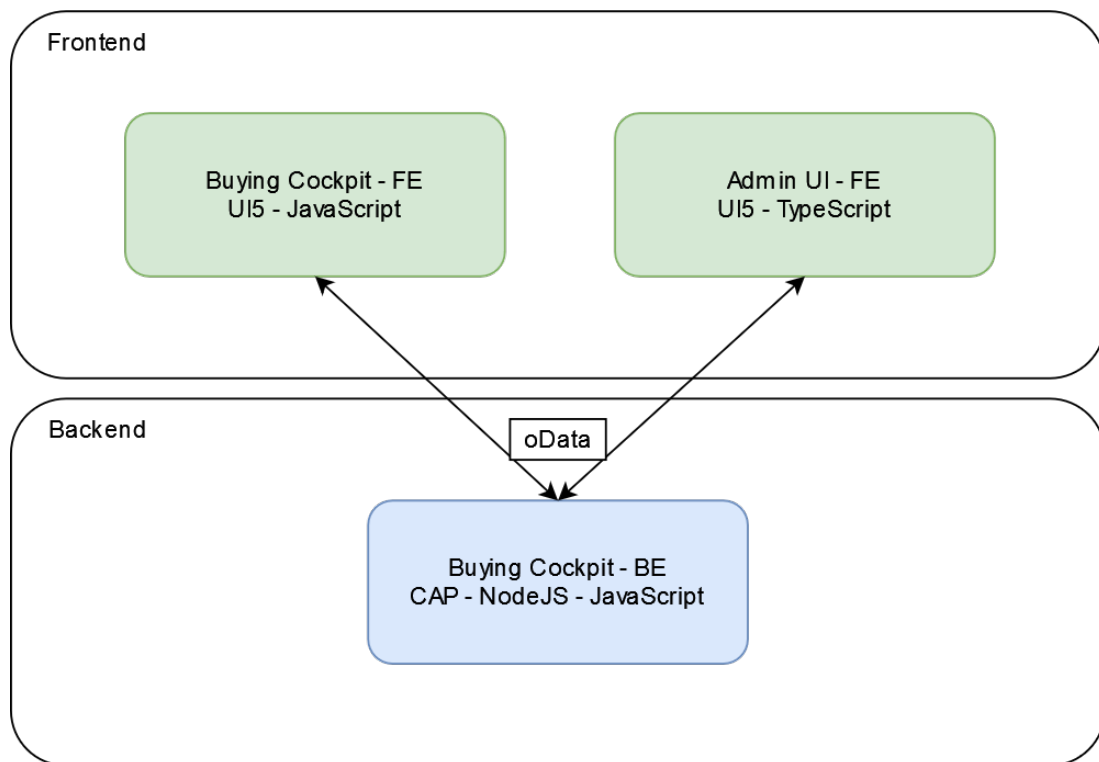


Abbildung 3.5. Grundlegende Architektur der Anwendungen

Die Architektur der Anwendungen ist in Frontend und Backend aufgeteilt und diese Ebenen kommunizieren über ein standardisiertes Protokoll miteinander.

1. Frontend-Komponenten:

- **Buying Cockpit - Frontend (FE):**

Diese Komponente ist mit SAPUI5 und JavaScript entwickelt und bildet die Hauptanwendung und ist die Benutzeroberfläche für die Nutzer.

- **Admin-UI:**

Dies ist die zu entwickelnde Administrationsoberfläche für das Buying Cockpit und kann nur von bestimmten Nutzern verwendet werden.

2. Backend-Komponente:

- **Buying Cockpit - Backend (BE):** Das Backend des Systems basiert auf SAP CAP und wird mit NodeJS und JavaScript entwickelt. Es stellt die benötigten Daten und Services über standardisierte Schnittstellen beiden Frontend-Anwendungen zur Verfügung.

3. Kommunikationsprotokoll:

Die Kommunikation zwischen dem Frontend und dem Backend erfolgt über das **OData-Protokoll**. OData ermöglicht den standardisierten Austausch von Daten über HTTP und bietet eine effiziente Möglichkeit, auf Daten zuzugreifen und diese zu manipulieren.

Datenstruktur

Im folgenden wird die Datenstruktur für die Anwendungen vereinfacht dargestellt und Attribute, Entitäten und Relationen, welche nicht relevant für das Admin-UI sind, ausgelassen.

In Abbildung 3.6 sind die drei Entitäten, welche im Admin-UI manipuliert werden sollen, in einen Entity-Relationship-Modell (ER-Modell) dargestellt:

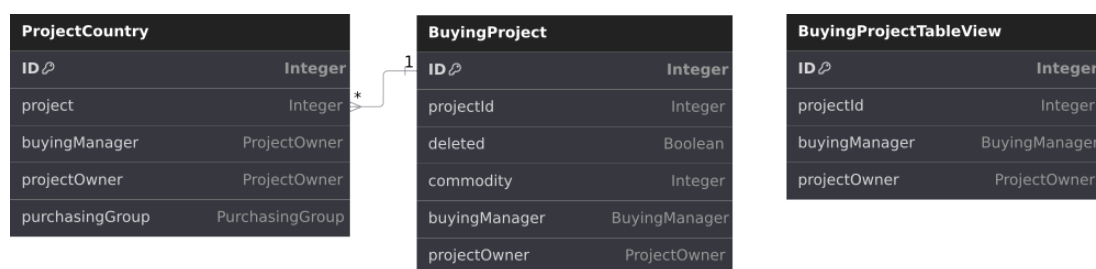


Abbildung 3.6. Vereinfachte Darstellung der Datenstruktur

Die Entität *BuyingProjectTableView* ist eine Entität, welche für das Filtern auf der Übersichtsseite der BuyingCockpit Anwendung benötigt wird. Daher müssen die Werte aus der *BuyingProject*-Entität mit dieser übereinstimmen und somit immer beide simultan aktualisiert werden.

3.6 Implementierung

Dieses Kapitel beschreibt die Umsetzung des Projekts im Detail. Es wird ein Überblick über den Implementierungsprozess gegeben, der Vorgehensweise bei der Entwicklung und ein Einblick in Probleme, welche während der Entwicklung aufgetreten sind. Neben einer schrittweisen Darstellung der Implementierung werden auch exemplarisch Codebeispiele vorgestellt.

3.6.1 Implementierungsdetails

In den Implementierungsdetails werden die Vorgehensweisen und Konzepte, welche für die Implementierung des Frontends und Backends benötigt werden, erläutert.

Frontend

Das SAPUI5-Framework verwendet zur Darstellung der Seiten sogenannte *Views* welche in Extensible Markup Language (XML) Dateien definiert werden. Diese Views müssen mit der Dateierweiterung **".view.xml"** enden, damit sie von SAPUI5 als View erkannt werden.

Views sind eine Art Container für SAPUI5-Elemente (Buttons, Input Felder, Entitäten, Listen, ...) und SAPUI5-Layouts (Flex Box, HBox, VBox). Views können jedoch auch andere Views beinhalten und so einen verschachtelten Aufbau der Seite schaffen. Die SAPUI5-Elemente sind vorgefertigte Komponenten welche von SAPUI5 bereitgestellt werden, um eine einheitliche Benutzeroberfläche zu schaffen, welche dann auch über verschiedene SAP-Anwendungen in einem Unternehmen konsistent ist.

Für die Admin-UI Seite sollen die Views und Komponenten wie in Abbildung 3.7 angeordnet und strukturiert werden. Die Unterseiten für die Eingabefelder aus den Anforderungen 3.4, 3.5 und 3.6 werden in eigenen Views angezeigt, zwischen denen der Benutzer über die Navigationsleiste wechseln kann.

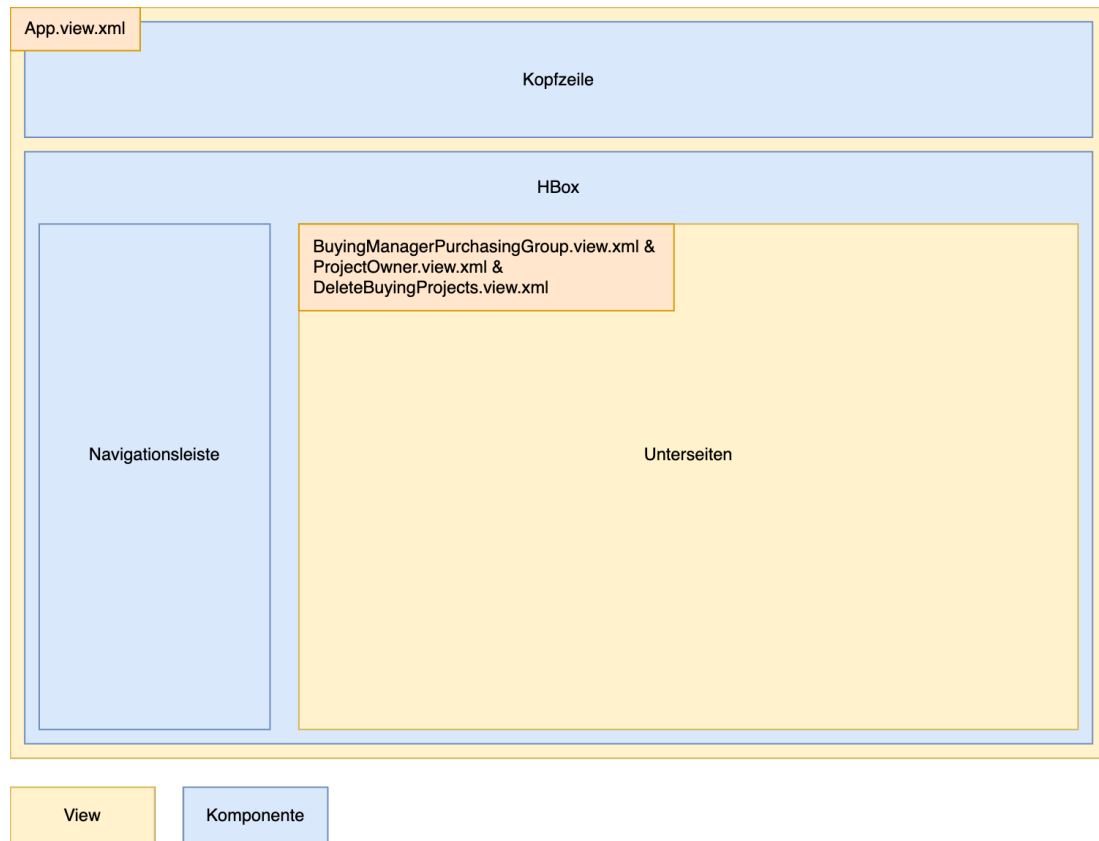


Abbildung 3.7. Darstellung der Anwendungsstruktur

Für jeden View kann ein *Controller* erstellt werden, welcher die Funktionalität für die Views bereitstellt und Elemente dynamisch laden kann. Ein Controller muss, so wie die View, eine besondere Dateiergung haben, damit SAPUI5 die Datei als Controller erkennen kann. Für Controller ist diese Endung **".controller.ts"**.

Für das Admin-UI soll für jeden View ein Controller erstellt werden. Zudem soll es einen sogenannten *BaseController* geben, von dem alle anderen Controller seine Funktionen erben. Dieser BaseController benötigt, anders also normale Controller, nicht die Controller spezifische Endung, sondern nur die TypeScript-spezifische Dateiergung **".ts"**. Denn der BaseController wird keinem View explizit zugeordnet und muss daher auch nicht von SAPUI5 als Controller erkannt werden.

Die Funktion eines BaseControllers ist es, Funktionen, welche von jedem Controller benötigt werden, in diesem zu definieren, damit diese an einer zentralen Stelle definiert sind und nicht in jedem Controller neu definiert werden müssen. Dazu gehören meist Helferfunktionen für zum Beispiel das Routing oder im Fall des Admin-UIs auch die Funktionalität der Aktionsleiste (siehe Anforderung 3.7), welche auf jeder Unterseite eine sehr ähnliche Funktion hat.

In den Controllern für die einzelnen Unterseiten sollen Funktionen, die spezifisch für die Unterseiten sind stehen, wie die intelligenten Vorschläge für die Eingabefelder (Anforderung 3.3) und das Senden der Daten an das Backend.

Backend

Im Backend sollen die Daten, welche von dem Frontend gesendet wurden, validiert und verarbeitet werden.

Für das Löschen eines Kaufprojekts muss lediglich die *projectId* des zu löschenden Projektes gesendet werden, wofür eine SAP CAP *function* verwendet werden soll.

Für das Ändern der Projekt Owner und Manager, sowie der Purchasing Organisation, soll für jede Änderung ein neuer Eintrag in eine Datenbanktabelle geschrieben werden, der die alten und neuen Werte, sowie den Benutzer der die Änderung getätigt hat und den Zeitpunkt der Änderung beinhaltet. Diese Datenbanktabellen sind in Abbildung 3.8 dargestellt. Dies soll dazu dienen, die Änderungen auch im Nachhinein nachvollziehen zu können oder auch rückgängig zu machen.

PurchasingGroupMassChange	
ID	UUID
oldBuyingManager	String
newBuyingManager	String
oldPurchasingGroup	String
newPurchasingGroup	String
acgs	String
created_at	Timestamp
user_id	UUID

ProjectOwnerMassChange	
ID	UUID
oldProjectOwner	String
newProjectOwner	String
acgs	String
created_at	Timestamp
user_id	UUID

Abbildung 3.8. Datenbanktabellen für Massenänderungen

Wenn aus dem Frontend ein neuer Eintrag in einer der Entitäten erstellt wird, soll mit einer sogenannten *Hook* in SAP CAP das Ändern der Daten geschehen. Diese *Hooks* fangen das Erstellen der Entität ab und führen dann individuellen Code aus, in dem die gesendeten Daten verarbeitet werden können.

Bevor jedoch die Daten verarbeitet werden, muss überprüft werden, ob der Benutzer, der die Daten gesendet hat, die Berechtigung hat, diese Änderungen durchzuführen. Falls dies nicht der Fall ist, soll ein Error an das Frontend gesendet und eine entsprechende Fehlermeldung angezeigt werden. Besitzt der Benutzer jedoch die benötigte Berechtigung, sollen die Daten verarbeitet und validiert werden und die Änderungen in der Datenbank vorgenommen werden.

3.6.2 Schrittweise Beschreibung der Implementierung

In diesem Abschnitt werden die Implementierungsdetails aus dem vorherigen Abschnitt anhand von Codebeispielen schrittweise erläutert und erklärt.

Frontend

Die Implementierung des Frontends teilt sich in zwei Abschnitte auf. Zum einen die Erstellung der Views, welche die Darstellung der Seiten sein werden, und zum anderen die Erstellung der Controller, die Views mit Funktionalität zu versehen.

Erstellung der XML-Views:

Für die Darstellung wurden, wie in Abbildung 3.7 bereits dargestellt wurde, vier verschiedene XML-Views angelegt. Die *App.view.xml*, *BuyingManagerPurchasingGroup.view.xml*, *ProjectOwner.view.xml* und die *DeleteBuyingProjects.view.xml*.

Für die App-View wurden drei Hauptkomponenten benötigt. Die Kopfzeile mit dem Titel der Seite, die Navigationsleiste zur Navigation zwischen den Unterseiten und den Unterseiten an sich, welche dynamisch mithilfe der Navigationsleiste angezeigt werden. All diese Komponenten wurden innerhalb einer *DynamicPage*-Komponente definiert, welche bereits Funktionalitäten für eine einfahrbare Kopfzeile und Navigationsleiste bietet. So konnte die Kopfzeile als eine *DynamicPageTitle* und *DynamicPageHeader*-Komponente als Titel und Header der *DynamicPage* hinzugefügt werden. Wie das dann im Code für das Admin-UI aussieht, zeigt das folgende Listing:

```
1 <f:DynamicPage class="sapUiNoContentPadding">
2     <f:title>
3         <f:DynamicPageTitle>
4             <f:heading>
5                 <Title text="{i18n>titHeader}" class="
6 sapUiSmallMarginTop" />
7             </f:heading>
8         </f:DynamicPageTitle>
9     </f:title>
10    <f:header>
11        <f:DynamicPageHeader>
12            <VBox>
13                <Label text="{i18n>lblVersion}" />
14            </VBox>
15        </f:DynamicPageHeader>
16    </f:header>
17    ...
18 </f:DynamicPage>
```

Listing 3.7. Kopfzeile des App-Views

Das *text*-Attribut liefert den Text, der in dem Titel und dem Header angezeigt wird. Dieser kommt in unserem Fall aus dem *i18n*-Model, welches Text in verschiedenen Sprachen bereitstellt. Das *class*-Attribut ist für das Styling der Elemente verantwortlich. Hier wird eine von SAPUI5 vorgefertigte Klasse angegeben, um ein einheitliches Aussehen auf der ganzen Seite zu garantieren.

Für die Navigationsleiste und die Unterseiten wurden zwei *Panel-Komponenten* verwendet, diese dienen als Container für den Inhalt der Komponenten und haben die Möglichkeit, sich ein- und ausklappen zu lassen. Die Navigationsleiste wurde als eine Liste von *ActionListItem*s definiert, welche eine Art Knöpfe sind, die durch Attribute als ausgewählt angezeigt werden können. Diese *ActionListItem*s werden anhand von vordefinierten Routen dynamisch generiert und sind somit jeweils an eine Route, beziehungsweise an eine Unterseite gebunden. Durch Anklicken eines der *ActionListItem*s wird über den SAPUI5-Router das richtige XML-View in den *Panel-Container* für die Unterseiten geladen.

In dem folgenden Listing wird der Code für die beiden *Panel-Komponenten* dargestellt, es wurden jedoch Attribute und Elemente, welche lediglich für das Aussehen der Komponenten zuständig sind, ausgelassen.

```
1  ...
2  <Panel>
3      <List items="{routes}>/routes}">
4          <ActionListItem
5              text="{
6                  parts: [
7                      'routes>name'
8                  ], formatter: '.formatter.formatNavItemText'
9              }"
10             type="Navigation"
11             navigated="{= ${routes>name} === ${routes>/currentRoute}}"
12             press="onNavItemPress" />
13      </List>
14  </Panel>
15  <Panel>
16      <NavContainer id="navContainer" width="100%" height="100%" />
17  </Panel>
18  ...
```

Listing 3.8. Navigationsleisten- und Unterseiten-Container des Admin-UIs

Die drei XML-Views für die Unterseiten sind alle gleich aufgebaut und verwenden alle dieselben Komponenten zur Darstellung der *Toolbar* zum Speichern der Änderungen und den Eingabefeldern.

Für die Toolbar wird die *Toolbar-Komponente* von SAPUI5 verwendet und beinhaltet zwei Knöpfe. Ein Knopf zum Speichern und ein Knopf, um alle Eingabefelder zu leeren. Im

folgenden Listing wird der Code für die Toolbar, welche auf allen Seiten weitestgehend identisch ist, dargestellt:

```
1 <Toolbar id="toolbar" design="Solid">
2     <ToolbarSpacer />
3     <Button text="{i18n>btnSaveChanges}" type="Emphasized" press="
        onSaveChangesPress" />
4     <Button text="{i18n>btnClear}" press="onClearPress" />
5 </Toolbar>
```

Listing 3.9. Toolbar der Unterseiten

Die Eingabefelder wurden mit *Input-Komponenten* umgesetzt, da diese bereits Funktionalitäten für intelligente Vorschläge, wie in Anforderung 3.3 beschrieben, besitzen. Für eine bessere Formatierung wurde jedes Eingabefeld und dessen Titel, welcher als *Label-Komponente* dargestellt wird, in einer *VBox-Komponente* platziert. In dem folgenden Listing wird der Code für das Eingabefeld beispielhaft an einem Eingabefeld auf der ProjectOwner.view.xml Unterseite dargestellt:

```
1 <VBox width="100%">
2     <Label text="{i18n>lblOldBuyingManager}" required="true" />
3     <Input
4         id="oldBuyingManagerInput"
5         value="{form>/oldBuyingManager}"
6         placeholder="{i18n>txtEmailPlaceholder1}"
7         required="true"
8         suggestionItems="{/AppUsers}"
9         showSuggestion="true"
10        suggest="onSuggest($event, 'user')"
11        suggestionItemSelected="onSuggestionItemSelected">
12         <core:Item key="{id}" text="{id}" />
13     </Input>
14 </VBox>
```

Listing 3.10. Eingabefeld für Unterseiten

Die Attribute *suggest* und *suggestionItemSelected* der Input-Komponente bekommen die Namen von Funktionen übergeben, welche in einem Controller für die Views definiert sind. Die Funktion dieser Funktionen ist es, die intelligenten Vorschläge anhand der Benutzereingabe anzuzeigen und beim Auswählen eines der Vorschläge diesen Wert für das Feld zu übernehmen. Das Attribut *required* ist dafür da anzuzeigen, ob das Eingabefeld ein Pflichtfeld ist. Wird dieses auf *true* gesetzt, wird vor dem Speichern überprüft, ob in diesem Feld etwas eingegeben wurde.

Erstellung der Controller:

Einige Funktionalitäten sind in allen Unterseiten identisch, daher wurde für diese Funktionen ein BaseController erstellt. Funktionen in diesem Controller stehen allen Views zur

Verfügung, wodurch der Code für diese Funktionalitäten nur einmal geschrieben werden musste.

Andere Funktionalitäten sind jedoch spezifisch für eine Unterseite oder mussten unterschiedlich implementiert werden, da es zum Beispiel unterschiedliche Eingabefelder auf den Seiten gibt. Eine dieser Funktionen ist die *onSaveChangesPress*-Funktion aus Listing 3.9, die für das Speichern der Eingabefelder zuständig ist.

```
1 onSaveChangesPress(event: Button$PressEvent): void {
2     const formModel = this.getModel<JSONModel>("form"),
3         isValid = this.validateInputFields(this.allInputIDs);
4
5     if (!isValid) {
6         event.getSource().setEnabled(false);
7         return;
8     }
9
10    if (!formModel) return;
11
12    const {
13        oldProjectOwner,
14        newProjectOwner,
15        SCGs
16    } = formModel.getData();
17
18    this.createSaveChangesMessageBox(
19        "/ProjectOwnerMassChange",
20        {
21            oldProjectOwner,
22            newProjectOwner,
23            acgs: SCGs
24        },
25        this.allInputIDs
26    );
27 }
```

Listing 3.11. *onSaveChangesPress* Funktion

In dieser Funktion werden die Eingabefelder validiert, indem überprüft wird, ob alle Pflichtfelder ausgefüllt sind. Wenn dies der Fall ist, wird mit den Werten der Eingabefelder ein neuer Eintrag in der passenden Entität im Backend erstellt. Der Tabellename ist für die verschiedenen Unterseiten unterschiedlich und daher wird dieser zusammen mit den Daten an die *createSaveChangesMessageBox*-Funktion im BaseController gesendet. Diese Funktion ist im folgenden Listing zu sehen:

```
1 protected createSaveChangesMessageBox(sPath: string, formData: object, inputIDs:
2     string[]): void {
3     const resourceBundle = this.getI18nResourceBundle();
4     if (!resourceBundle) {
```

```

4         return;
5     }
6     const saveAction = resourceBundle.getText("btnSave") || "Save";
7     MessageBox.confirm(
8         resourceBundle.getText("txtConfirmDialog") || "",
9         {
10             title: resourceBundle.getText("titConfirmDialog") || "",
11             actions: [saveAction, MessageBox.Action.CANCEL],
12             emphasizedAction: saveAction,
13             onClose: (action: string) => {
14                 if (action === saveAction) {
15                     this.submitChanges(sPath, formData, inputIDs);
16                 }
17             }
18         }
19     );
20 }

```

Listing 3.12. createSaveChangesMessageBox Funktion

Dies ist eine Funktion im BaseController, welche eine *MessageBox-Komponente* erstellt, um den Nutzer um eine weitere Bestätigung zu bitten. Bestätigt der Benutzer seine Entscheidung, wird die *submitChanges*-Funktion aufgerufen.

```

1 protected submitChanges(sPath: string, oData: object, inputIDs: string[]): void
2 {
3     const viewModel = this.getModel<JSONModel>("objectView"),
4         model = this.getModel<ODataModel>();
5
6     if (!model || !viewModel) return;
7
8     viewModel.setProperty("/busy", true);
9     model.create(sPath, oData,
10         {
11             success: () => {
12                 this.clearAllFields(inputIDs);
13                 this.onSubmitSuccess();
14             },
15             error: this.onSubmitError.bind(this),
16         }
17     );
18 }

```

Listing 3.13. submitChanges Funktion

Diese Funktion ist dafür verantwortlich, die übergebenen Daten an das Backend zu senden. Um das zu tun, wird in dem OData-Model, welches mit dem Backend verbunden ist, ein neuer Eintrag erstellt. Dafür benötigt es die Namen der Entität, sowie die Daten, die in die Entität geschrieben werden sollen. Ist das Erstellen des neuen Eintrages erfolg-

reich, werden alle Eingabefelder geleert und es wird eine Meldung angezeigt, dass das Speichern erfolgreich war. Im Fall, dass das Erstellen nicht erfolgreich war, wird eine Fehlermeldung angezeigt.

Backend

Nachdem die Daten von dem Frontend an das Backend gesendet wurden, müssen diese dort noch verarbeitet werden und die von den Änderungen betroffenen Entitäten müssen bearbeitet werden. Die Beschreibung dieser Funktionalität wird im Folgenden schrittweise beschrieben.

Sobald im Frontend eine Anfrage an das Backend gesendet wird, um einen neuen Eintrag in der Entität zu erstellen, wird diese Anfrage von SAP CAP abgefangen. Dies geschieht in der `srv.on` Funktion, welche bestimmte Events abfangen kann. Sie hat drei Parameter, das Event, welches abgefangen werden soll, den Namen der Entität und eine Handlerfunktion, die aufgeführt wird, wenn eine Anfrage abgefangen wurde.

In diesem Fall ist das Event das *CREATE* Event, welches beim Erstellen eines neuen Eintrags aufgerufen wird. Der Name der Entität ist abhängig von der Unterseite, von der die Anfrage gestellt wurde. Die Handlerfunktion hat zwei Parameter, die Anfrage, in der die Daten die gesendet wurden sowie weitere Informationen über die Anfrage, wie den Benutzer, der die Anfrage gestellt hat, stehen.

Im Folgenden wird das Codebeispiel nur für eine der Funktionalitäten dargestellt, da der einzige Unterschied die aufgerufene Funktion ist.

```
1  srv.on('CREATE', 'PurchasingGroupMassChange', async (req, next) => {
2    const whitelistedMassChangeUsers = process.env.ADMIN_UI_USERS ? process.env.
      ADMIN_UI_USERS.split(',') : []
3    if (whitelistedMassChangeUsers.indexOf(req.user.id.toLowerCase()) < 0) {
4      req.error(400, 'Invalid user')
5    } else {
6      try {
7        await projectService.purchasingGroupMassChange(req)
8        return await next()
9      } catch (e) {
10       logAndThrowError(log, 'performing purchasing group mass change', e,
11         req.data)
12     }
13  })
```

Listing 3.14. `srv.on` Funktion zum Abfangen der Anfrage

In unserer Handlerfunktion ist das Erste, was geprüft wird, ob der Benutzer, der die An-

frage gestellt hat, die passenden Rechte für eine solche Anfrage besitzt. Ist dies nicht der Fall, wird eine Fehlernachricht zurückgesendet. Besitzt der Benutzer die passenden Rechte, wird eine Funktion aufgerufen, welche die Validierung und Verarbeitung der Daten übernimmt.

Hierbei wurde jedoch zwischen der Funktionalität, Kaufprojekte zu löschen und Kaufprojekte zu bearbeiten, unterschieden. Die Funktion, Kaufprojekte zu löschen, benötigte keine weitere Verarbeitung der Daten. Um ein Kaufprojekt zu löschen, beziehungsweise ein Kaufprojekt als gelöscht zu markieren, musste lediglich das Attribut *deleted* auf *true* gesetzt werden und der Eintrag in der *BuyingProjectTableView* Entität gelöscht werden.

Für die Funktionalitäten des Aktualisierens der Projektmanager und der Käufergruppe sowie des Projektowners, müssen die Daten noch validiert und verarbeitet werden. Dafür wird zuerst überprüft, ob alle benötigten Daten angekommen sind und ob diese das korrekte Format besitzen. Falls dies nicht der Fall ist, wird eine Fehlernachricht zurückgesendet. Nachdem die Validierung abgeschlossen ist und es zu keinem Fehler gekommen ist, werden die betroffenen Entitäten aktualisiert.

Die verschiedenen Unterseiten erfordern eine unterschiedliche Art und Weise, wie die Daten aktualisiert werden, daher wurden in diesem Schritt verschiedene Funktionen aufgerufen. Für die Aktualisierung der Projektmanager und Käufergruppen wird zuerst die neue Käufergruppe hinzugefügt und dem neuen Projektmanager zugewiesen.

```
1 async function insertNewPurchGroup(db, newPurchGroup) {
2   const { PurchasingGroup } = db.entities('com.company.buyingcockpit.
      buyingProject')
3   await db.run(INSERT.into(PurchasingGroup)
4     .entries({ ID: newPurchGroup, Name: newPurchGroup }))
5 }
6
7 async function updatePurchGroupOfNewManager(db, newPurchGroup, newBuyingManager)
8   {
9     const { AppUser } = db.entities('com.company.buyingcockpit.common')
10    await db.run(UPDATE(AppUser)
11      .with({ purchaseGroup: newPurchGroup })
12      .where({ ID: newBuyingManager })))
13 }
```

Listing 3.15. Einfügen und Aktualisieren der Käufergruppe

Die *db.run*-Funktion ist eine Funktion von SAP CAP die verschiedene Datenbankoperationen durchführen kann. In den beiden Funktionen wird *INSERT* zum Hinzufügen der neuen Käufergruppe verwendet und *UPDATE*, um den Projektmanager zu aktualisieren.

Nachdem die Käufergruppe aktualisiert wurde, müssen die Projektländer aktualisiert werden. Die Projektländer mussten hierfür nach zwei Kriterien gefiltert werden. Zum einen

danach, dass der alte Projektmanager als Projektmanager in dem Projektland eingetragen ist, sowie, dass das Kaufprojekt, zu dem das Projektland gehört, die SCG besitzt, die der Benutzer gegebenenfalls angegeben hat.

Das Filtern nach dem aktuellen Projektmanager konnte mit einer *where*-Bedingung der *SELECT*-Operation geschehen:

```
1 const projectCountriesToUpdate = await db.run(  
2   SELECT.from(ProjectCountry)  
3     .columns('ID', 'project_ID')  
4     .where({ buyingManager_ID: oldBuyingManager, or: { purchasingGroup_ID:  
       oldPurchGroup } })  
5 )
```

Listing 3.16. SELECT-Operation für Projektländer

Um diese Projektländer dann nach den SCGs zu filtern, wurde mehr Berechnungsaufwand benötigt, da diese nicht in den Projektländern gespeichert sind, sondern in dem dazugehörigen Kaufprojekt. Daher wurde eine weitere Funktion geschrieben, welche die vor gefilterten Projektländer nach den SCGs filtert:

```
1 async function filterProjectCountriesBySCGs(db, projectCountries, scgs) {  
2   const { BuyingProject } = db.entities('com.company.buyingcockpit.  
   buyingProject')  
3   const projectIds = [...new Set(projectCountries.map((pc) => pc.project_ID))]  
4  
5   if (scgs.length > 0 && projectIds.length > 0) {  
6     const projects = await db.run(  
7       SELECT.from(BuyingProject)  
8         .columns((project) => {  
9           project('ID')  
10          project.commodity((commodity) => {  
11            commodity('Code')  
12          })  
13        })  
14        .where({ ID: { in: projectIds } })  
15      )  
16  
17      const projectsMap = {}  
18      for (const project of projects) {  
19        projectsMap[project.ID] = project  
20      }  
21  
22      return projectCountries.filter((pc) => {  
23        const project = projectsMap[pc.project_ID]  
24        return !!project && !!project.commodity && scgs.indexOf(project.  
          commodity.Code) >= 0  
25      })  
26    } else {
```

```
27         return projectCountries
28     }
29 }
```

Listing 3.17. SCG Filterfunktion für Projektländer

In dieser Funktion werden zuerst alle Kaufprojekte (*BuyingProject*) mit der *SELECT*-Operation gesucht, welche in den Projektländern als Kaufprojekt angegeben sind. Anhand dieser Kaufprojekte werden dann die Projektländer (*ProjectCountries*) gefiltert. Dafür wird überprüft, ob der *Commodity Code* (auch SCG) des zu dem Projektland gehörenden Kaufprojektes in der Liste der angegebenen SCGs ist. Diese gefilterte Liste wird dann zurückgegeben. Anhand dieser Liste konnten dann die betroffenen Entitäten aktualisiert werden.

Sind diese Entitäten aktualisiert, ist die Verarbeitung der Daten vollendet. Das Letzte was nun noch geschieht, ist, dass ein Eintrag mit den Änderungen, die vorgenommen wurden, erstellt wird, um diese zu einem späteren Zeitpunkt nachvollziehen zu können und gegebenenfalls rückgängig zu machen.

3.6.3 Herausforderungen und Problemlösungen

Ein zentrales Problem war die Handhabung der unterschiedlichen Anforderungen in den Controllern. Während einige Funktionalitäten spezifisch für eine Unterseite waren, mussten andere auf mehreren Seiten gleich funktionieren. Um das DRY-Prinzip (Don't Repeat Yourself) zu beachten, war es wichtig, den Code, der von mehreren Seiten genutzt wird, zu abstrahieren.

Um dieses Problem zu lösen, wurden die gemeinsamen Logiken der Controller in den *BaseController* ausgelagert. Dadurch konnte sichergestellt werden, dass wiederkehrende Logik nicht mehrfach in den einzelnen Controllern dupliziert wird. Anstatt den Code für API-Aufrufe oder Datenvalidierung in jedem Controller zu wiederholen, wurden diese Funktionen in den *BaseController* integriert. Die spezifischen Controller konnten dann von dieser Klasse erben und nur die für die Unterseiten benötigten Funktionen implementieren. So blieb der Code flexibel und konnte den spezifischen Anforderungen der Unterseiten gerecht werden, ohne redundanten Code schreiben zu müssen.

Zusätzlich wurden Hilfsfunktionen erstellt, die häufig genutzte Logik wie die Validierung von Benutzereingaben oder die Verarbeitung von Daten zentralisiert. Diese Hilfsfunktionen konnten dann von allen Controllern aufgerufen werden, was den Code nicht nur wiederverwendbar, sondern auch wartbarer machte.

Durch diese Maßnahmen konnten die unterschiedlichen Anforderungen der Unterseiten effizient gemanaget und gleichzeitig sichergestellt werden, dass das DRY-Prinzip beachtet wird.

3.7 Fazit

In diesem Projekt wurde erfolgreich eine neue Admin-Seite entwickelt, die es den Nutzern ermöglicht, administrative Aufgaben eigenständig und effizient durchzuführen. Diese Lösung hat die Abhängigkeit von Entwicklern erheblich reduziert und die Effizienz der Arbeitsabläufe gesteigert.

Die neue Admin-Seite erlaubt es den Endnutzern, Projekte zu löschen und Informationen selbst zu aktualisieren, ohne auf technische Unterstützung angewiesen zu sein. Dadurch können kleine Änderungen nun direkt und zeitnah umgesetzt werden, was zu einer spürbaren Effizienzsteigerung führt.

Die Entscheidung, TypeScript anstelle von JavaScript zu verwenden, hat sich als äußerst vorteilhaft erwiesen. Die Integration von TypeScript hat nicht nur dazu beigetragen, die Codequalität zu verbessern, sondern auch den Entwicklungsprozess zu optimieren. Durch die verbesserte Lesbarkeit und Struktur des Codes konnte schneller gearbeitet und Fehler frühzeitig erkannt werden, was den gesamten Entwicklungszyklus beschleunigt hat.

Zusammenfassend lässt sich sagen, dass die neue Admin-Seite einen bedeutenden Fortschritt für das Unternehmen des Kunden darstellt. Sie bietet eine benutzerfreundliche Lösung für administrative Aufgaben und legt gleichzeitig den Grundstein für zukünftige Entwicklungen. Die positiven Effekte auf die Effizienz und Kostensenkung werden langfristig zu einer verbesserten Zusammenarbeit zwischen den Nutzern und den Entwicklern führen.

Literatur

- [1] Microsoft. "TypeScript Documentation". en. (2024), Adresse: <https://www.typescriptlang.org/> (besucht am 14.08.2024).
- [2] Microsoft. "TypeScript in Visual Studio Code". (8. Jan. 2024), Adresse: <https://code.visualstudio.com/docs/languages/typescript> (besucht am 14.08.2024).
- [3] Microsoft. "Why does TypeScript exist?" Paragraph: "What Problems Can TypeScript Solve?" (2024), Adresse: <https://www.typescriptlang.org/why-create-typescript/> (besucht am 14.08.2024).
- [4] SAP. "Model View Controller (MVC)". (2024), Adresse: <https://sapui5.hana.ondemand.com/sdk/#/topic/91f233476f4d1014b6dd926db0e91070.html> (besucht am 26.09.2024).

Abbildungsverzeichnis

3.1	Mockup: Admin-UI Projekowner Seite	28
3.2	Mockup: Admin-UI Projektmanager und Käufergruppe Seite	29
3.3	Mockup: Admin-UI Löschungs-Seite	30
3.4	Mockup: Admin-UI Pop-Up	30
3.5	Grundlegende Architektur der Anwendungen	31
3.6	Vereinfachte Darstellung der Datenstruktur	32
3.7	Darstellung der Anwendungsstruktur	34
3.8	Datenbanktabellen für Massenänderungen	35

Tabellenverzeichnis

3.1	Anforderung A1 - Reaktionszeit der Anwendung	23
3.2	Anforderung A2 - Sprache der Anwendung	23
3.3	Anforderung A4 - Intelligente Vorschläge für Eingabefelder	23
3.4	Anforderung A4 - Eingabefelder der Seite zum Löschen der Kaufprojekte .	23
3.5	Anforderung A4 - Eingabefelder der Seite zum Aktualisieren des Projektow- ners	24
3.6	Anforderung A6 - Eingabefelder der Seite zum Aktualisieren des Projektma- nagers und der Käufergruppe	24
3.7	Anforderung A7 - Speichern und Verwerfen Aktionsleiste	25
3.8	Anforderung A7.1 - Speicheraktion auf der Seite zum Löschen der Kaufpro- jekte	25
3.9	Anforderung A7.2 - Speicheraktion auf der Seite zum Aktualisieren des Pro- jektowners	25
3.10	Anforderung A7.3 - Speicheraktion auf der Seite zum Aktualisieren des Pro- jektmanagers und der Käufergruppe	26

Listings

3.1	Beispiel: Typen in TypeScript	17
3.2	Beispiel: Interfaces in TypeScript	17
3.3	Beispiel: Klassen in TypeScript	17
3.4	Beispiel: Generics in TypeScript	18
3.5	Beispiel: JavaScript BaseController.js	21
3.6	Beispiel: TypeScript BaseController.ts	21
3.7	Kopfzeile des App-Views	36
3.8	Navigationsleisten- und Unterseiten-Container des Admin-UIs	37
3.9	Toolbar der Unterseiten	38
3.10	Eingabefeld für Unterseiten	38
3.11	onSaveChangesPress Funktion	39
3.12	createSaveChangesMessageBox Funktion	39
3.13	submitChanges Funktion	40
3.14	srv.on Funktion zum Abfangen der Anfrage	41
3.15	Einfügen und Aktualisieren der Käufergruppe	42
3.16	SELECT-Operation für Projektländer	43
3.17	SCG Filterfunktion für Projektländer	43

Abkürzungsverzeichnis

BE	Backend
ER-Modell	Entity-Relationship-Modell
FE	Frontend
IDE	Integrated Development Environment
OData	Open Data Protocol
OOP	Objektorientierte Programmierung
SAP CAP	SAP Cloud Application Programming Model
SAPUI5	SAP User Interface 5
SCG	Sub Commodity Group
SQL	Structured Query Language
UX	User Experience
XML	Extensible Markup Language