Agile Project Dashboard

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2017

Autoren:	Tobias Schmitz
	Philipp Walder
Betreuer:	Prof. Dr. Markus Stolze
Projektpartner:	Zühlke Engineering AG
	Wiesenstrasse 10a 8952 Schlieren
	(Zürich)
Experte:	Thomas Kälin
Gegenleser:	Prof. Dr. Farhad D. Mehta

1 Inhaltsverzeichnis

Aufgabenstellung Bachelorarbeit Abteilung I, FS 2017
Tobias Schmitz, Philipp Walder

Scenarioo Agile Project Dashboard

1. Betreuer & Praxispartner

Betreuer dieser Arbeit ist
Prof. Dr. Markus Stolze

Co-Referent für diese Arbeit ist
Farhad Mehta

Externer Experte für diese Arbeit ist
Thomas Kälin

Anwendungspartner dieser Arbeit ist
Zühlke Engineering AG
Wiesenstrasse 10a
8952 Schlieren (Zürich)

2. Ausgangslage

Siehe separates Blatt (Aufgabenstellung Zühlke)

3. Ziele der Arbeit

Siehe separates Blatt (Aufgabenstellung Zühlke)

4. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die Dokumentation ist vollständig entsprechend den Anweisungen der Abteilung Informatik abzugeben (Upload), sowie ein Download-Link für Prof. Stolze und weitere Exemplare nach Absprache mit dem Co-Referenten und dem Experten.

Zudem ist eine kurze Projektresultatdokumentation im öffentlichen Wiki von Prof. M. Stolze zu erstellen.

5. Weitere Regeln und Termine

Im Weiteren gelten die allgemeinen Regeln zu Bachelor und Studienarbeiten „Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik“ (HSR Intranet) https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html)

Der Terminplan ist hier ersichtlich (HSR Intranet) https://www.hsr.ch/Termine-Bachelor-und-Studiena.5142.0.html

6. Rechte

Die resultierende Software und Dokumentation wird als Komponente des Open-Source Projekts Scenarioo mit der entsprechenden Lizenz veröffentlicht. Es gelten die sich daraus ergebenden Rechte. Die Arbeit (ohne allfällig geheime Anhänge) wird von der HSR im E-Prints Respository der HSR (eprints.hsr.ch) elektronisch veröffentlich. Hierbei ist von den Studenten darauf besonders darauf zu achten, dass in Screenshots und an anderen Stellen keine Kundendaten oder Namen sichtbar werden. Zudem sollten Interviews sorgfältig anonymisiert werden. Die zu publizierende Version sollte dem Praxispartner zur Überprüfung der Einhaltung dieser Regeln mit der Abgabe der Arbeit vorgelegt werden. Allfällig notwendige Anpassungen an der zu publizierenden Version werden Praxispartner binnen 4 Wochen gemeldet und danach von den Studenten entsprechend innerhalb einer Woche umgesetzt. Titel, Abstract und Praxispartner der Arbeit dürfen von der HSR und Studierenden schon während der Arbeit kommuniziert werden.

7. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 350h Arbeit für die Bachelorarbeit aufgewendet werden. Dies entspricht ungefähr 25h pro Woche (auf 14 Wochen) und damit ca. 3 Tage Arbeit pro Woche pro Student.

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterienliste.

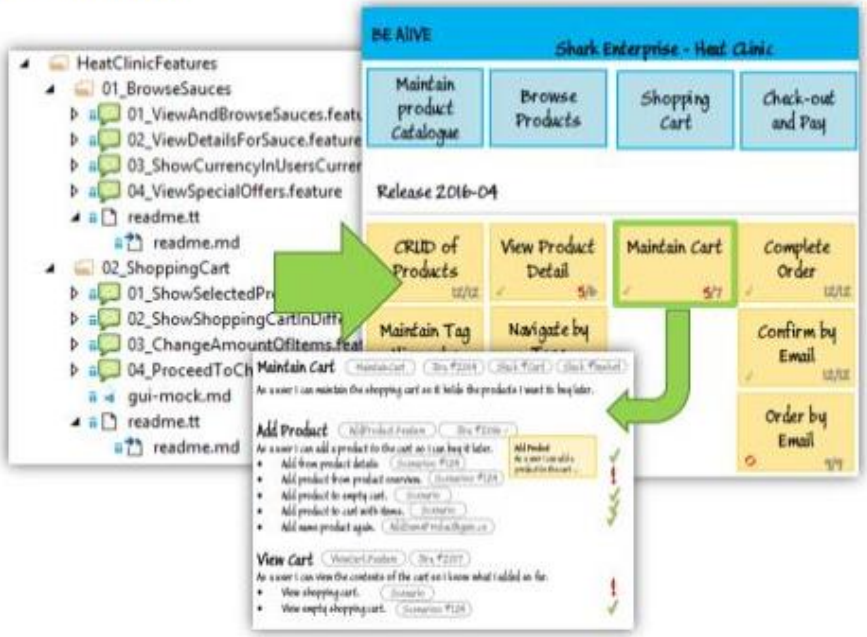
Rapperswil, 12.6.2017

Prof. Dr. Markus Stolze
Institut für Software
Hochschule für Technik Rapperswil

1. Ausschreibung Zühlke «Agile Project Dashboard»

4. Ziel der Arbeit

Über eine Webapplikation sollen alle Beteiligten an grösseren agilen Softwareprojekten jederzeit die aktuellsten Informationen zu allen System-Funktionalitäten schnell und übersichtlich zugreifen und durchsuchen können.



5. Ausgangslage

Dank dem agilen Vorgehen bei heutigen Entwicklungsprojekten entstehen Lösungen, die den Bedürfnissen der Anwender besser gerecht werden und den Kundennutzen ins Zentrum stellen. Es werden User Stories erstellt und diskutiert, welche dann in kurzen Sprints entwickelt und getestet werden. Dazu werden User Stories und Abnahmekriterien diskutiert, es werden GUI Sketches erstellt und es werden Sourcecode und Testfälle geschrieben. Im Verlaufe der agilen Projekt-Entwicklung entstehen weitere Artefakte, wie z.B. Markdown-Dokumentations-Files, ausführbare BDD-Spezifikationen, automatisierte Tests, Test-Reports etc. Dabei die Übersicht zu behalten, was wie zusammen gehört und was in welcher Version geändert hat, ist für die Beteiligten nicht immer einfach.

6. Problemstellung

Business Analysten, Entwickler und Tester haben oft Schwierigkeiten aktuelle und relevante Informationen, Beschreibungen, Testfälle und andere Dokumente zu Systemfunktionalitäten effizient zu finden.

Die zu entwickelnde Webapplikation soll helfen folgende Fragen zu beantworten:

- Überblick über die Funktionalitäten eines Systems
- Welche Beschreibungen und Testfälle gibt es zu einer Systemfunktionalität
- Welche Teile einer Funktionalität wurden bereits umgesetzt, welche sind noch in Planung
- In welcher Version der Software wurde welche Funktionalität wie geändert

Das erarbeitete Minimum Viable Product soll später möglichst in das Open Source Projekt «Scenarioo» von Zühlke integriert werden können oder unter Umständen gar direkt als Teil dieser Webapplikation umgesetzt werden. Dies wird im Konzept gemeinsam entschieden. Mehr dazu siehe http://www.scnearioo.org.

7. Vorgehen

- Erarbeitung der Vision und eines Konzeptes gemeinsam mit Zühlke
- Erstellen von GUI-Mockups oder Prototypen mit anschliessendem Review
- Backlog mit User Stories erstellen und priorisieren
- Agile Entwicklung eines «Minimum Viable Products» mit regelmässigen Reviews

8. Technologien, Methodik und Betreuung

- Webentwicklung mit AngularJS & Java
- Konzeptionelle und technische Unterstützung durch erfahrene Zühlke Mitarbeiter
- Open Source Entwicklung unter github.com
- Agiles Vorgehen

9.

10. Randbedingungen

Teamgrösse:	2-3 Personen
Arbeitsumfang:	P6 (360h pro Student) / P5 (180h pro Student)
Sprachen:	Deutsch oder Englisch

11. Auftraggeber

Zühlke Engineering AG
Wiesenstrasse 10a 8952 Schlieren
(Zürich)

3 Abstract

Diese Bachelorarbeit behandelt die Erweiterung des Open Source Projektes Scenarioo. Diese Erweiterung umfasst die Erarbeitung eines Konzeptes zusammen mit erfahrenen Mitarbeitern des Industriepartners und die Umsetzung eines Prototyps, integriert in Scenarioo, auf Basis des erarbeiteten Konzeptes. Auch die Umsetzung wurde in enger Zusammenarbeit mit dem Industriepartner realisiert. Dabei wurde die Datenstruktur um ein Composite-Element erweitert, damit aus der flachen Struktur eine beliebig tiefe hierarchische Struktur generiert werden kann. Diesem Composite-Element können neu auch weitere Files angehängt werden, welche der Dokumentation eines Projektes dienen. Diese Änderungen mussten auf der Client- und Serverseite implementiert werden.

Auf dem Server, welcher in Java geschrieben ist und über eine Rest-Schnittstelle bedient wird, musste der Importalgorithmus für die neue Struktur umgeschrieben werden. Auf der Clientseite, die mit AngularJS in JavaScript umgesetzt wurde, wurden neue Views erarbeitet, welche die neu gewonnenen Informationen mit abdecken.

Auf der Clientseite wurde die Benutzerführung mit den neuen Ansichten auf die neu hierarchische Struktur angepasst. Hierfür wurden wiederverwendbare Komponenten implementiert, um die neuen Daten einheitlich darzustellen. Die neuen Dokumentationsfiles, welche entweder ausführbare Testspezifikationen oder Markdown Dokumente sein können, können dem Benutzer anschaulich in der hierarchischen Struktur wiedergegeben werden.

4 Management Summary

4.1 Ausgangslage

Die Bachelorarbeit «Agile Project Dashboard» befasst sich mit der Weiterentwicklung des Tools Scenarioo. Dies ist ein UI-Testreporting Tool, welches von erfahrenen Zühlke Mitarbeitern entwickelt und in vorhergehenden Studentenprojekten erweitert wurde. Scenarioo kann unter anderem anhand von Screenshots, aus automatisierten UI-Tests, die Veränderungen zwischen unterschiedlichen Versionen einer GUI-Ansicht erkennen und anzeigen. Ziel dieser Bachelorarbeit ist es, Scenarioo im Sinne einer «Living-Documentation» so zu erweitern, dass alle Projektbeteiligten weitere hilfreiche Informationen aus dem Projekt darin auffinden können. Insbesondere liegt dabei der Fokus auf ausführbaren Spezifikationen wie zum Beispiel Gherkin-Files, Unit-Tests, automatisierte Integrationtests sowie Markdown Dokumentationen. Alle diese Informationen sollen in einer hierarchischen Struktur abgelegt und dem Benutzer übersichtlich dargestellt werden können.

4.2 Vorgehen

Die Konzeptionsphase war für dieses Projekt sehr wichtig, entsprechend wurde hierfür viel Zeit investiert. Für die Weiterentwicklung hin zum «Living-Documentation» Ansatz war zwar eine grobe Vision gegeben, diese musste aber in enger Zusammenarbeit mit dem Industriepartner in mehreren Schritten verfeinert. Dieser Prozess fand iterativ mit wöchentlichen Meetings statt. Hierfür wurden die Anpassungen am Datenmodell diskutiert und deren Auswirkungen evaluiert. Zeitgleich wurden Darstellungsmöglichkeiten als Mockup Skizzen erstellt und später als «Clickable Prototype» umgesetzt. Dieser wurde in Rücksprache mit dem Industriepartner weiter verfeinert.

In der Implementierungsphase wurden dann die Änderungen des Datenmodells im Scenarioo System umgesetzt, und mit denen von dem «Clickable Prototype» abgeleiteten Views ergänzt. Auch hier wurde wiederum Iterativ mit stetiger Rücksprache zum Industriepartner gearbeitet. Ziel dieser Phase war die Entwicklung eines Prototyps, als Proof of Concept.

4.3 Ergebnis

Das finale Ergebnis der Implementationsphase wurde unter der Open Source Lizenz GNU GPL auf GitHub publiziert und dem Scenarioo-Entwicklerteam übergeben. Das entwickelte Konzept dient dem Entwicklerteam als Anhaltspunkt für eine Integration des Prototyps in ein zukünftiges Release von Scenarioo. Mit den erarbeiteten Erweiterungen sollte es nun möglich sein, eine «Living-Documentation» über ein ganzes Projekt in Scenarioo zu überwachen.

5 Technischer Bericht

5.1 Danksagungen

Wir möchten uns besonders bedanken:

- beim Industriepartner der Zühlke Engineering AG.
- bei Markus Flückiger von der Zühlke, der uns in User Experience Fragen sehr geholfen hat.
- bei den Teilnehmern des Usability Test von der Zühlke welche uns sehr geholfen haben, die Qualität des Endresultates stark zu verbessern.
- bei den Betreuern von der Zühlke, die unsere Arbeit kompetent unterstützten und sich trotz einer hohen Belastung durch laufende Projekte immer für uns Zeit nahmen.
- bei Prof. Dr. Markus Stolze vom Institut für Software (IFS) der HSR Rapperswil, der diese Arbeit betreute und uns stets unterstützte um zu einem optimalen Resultat zu kommen.

5.2 Einleitung

Die Bachelorarbeit «Agile Project Dashboard» ist eine Weiterentwicklung des bestehenden Open-Source Projektes Scenarioo[1]. Scenarioo war bis anhin ein Werkzeug zum automatisierten Testen von User Interfaces. Ziel dieser Bachelorarbeit ist nun Scenarioo so zu erweitern, dass auch weitere Dokumentations-Artefakte hier einfliessen können. So soll in Zukunft Scenarioo zu einem Living-Documentation Werkzeug werden.

Ähnliche Projekte die wir gefunden haben waren, der «GuideAutomator: continuous delivery of end user documentation[2]», der zum Ziel hat, die Screenshots in einer Benutzeranleitung immer aktuell zu halten, auch wenn sich die Software verändert. Das zweite Projekt, «Generating automatically the documentation from PLC code by D4T3 to improve the usability and life cycle management of software in automation[3]», erstellt ebenfalls aus Code-Files eine Projektdokumentation. Dies sind nur zwei Beispiele, aber es zeigt die Nachfrage nach automatisierter Dokumentationserstellung und Aktualisierung ist vorhanden.

Zum einen beschreibt diese Bachelorarbeit das Erarbeiten eines Konzeptes, um diese Weiterentwicklung zu realisieren, zum anderen befasst sich die Arbeit aber auch mit der Implementation eines «Minimum-Viable Products», also eines Prototyps, welcher in Scenarioo selber entwickelt wurde.

Bevor hier aber auf die eigentliche Arbeit eingegangen wird, zuerst noch eine detailliertere Beschreibung zum bestehenden Projekt.

[1] Scenarioo Website: http://scenarioo.org/
[2] ACM Digital Library: http://dl.acm.org/ und Literaturverzeichnis
[3] IEEE Digital Library: http://ieeexplore.ieee.org/ und Literaturverzeichnis

5.2.1 Vorgabe

5.2.1.1 Systemübersicht

Die Scenarioo Applikation besteht grundsätzlich aus drei Hauptkomponenten:

- Server Applikation
- Client Applikation
- Writer-Library (eine pro Sprache)

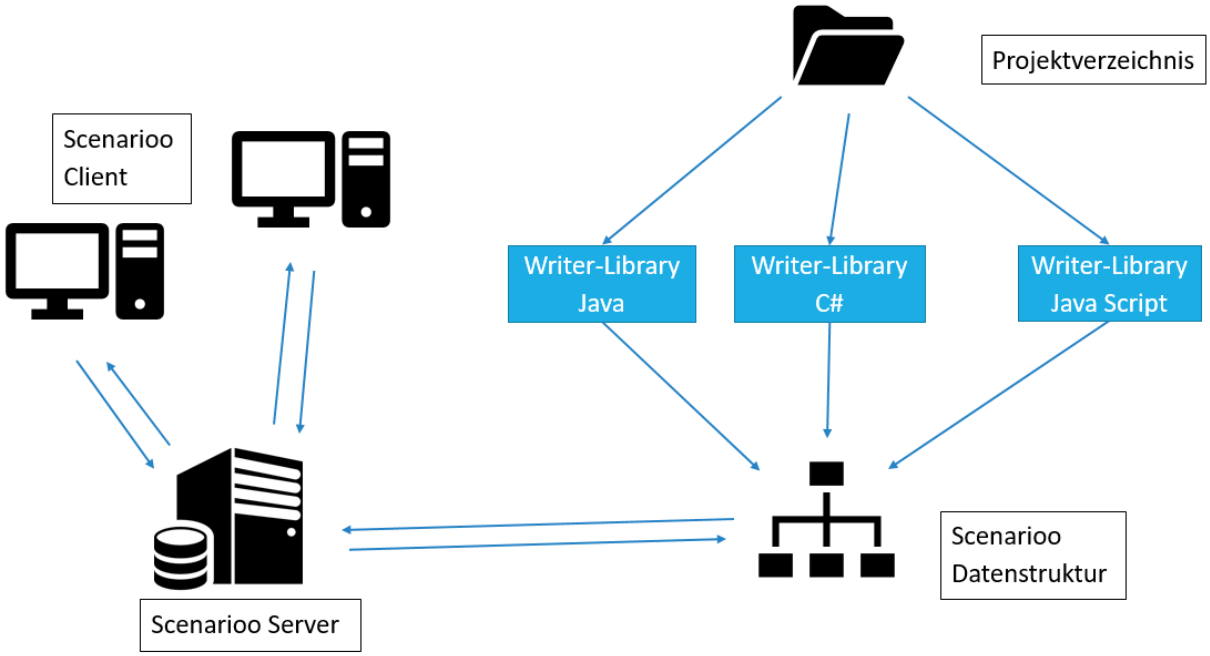


Abbildung 1: Systemübersicht Scenarioo

Writer-Libraries

Die Writer-Libraries generieren aus einem Projektverzeichnis die Senarioo Daten und speichern diese auf dem Server. Siehe Abbildung oben. Hierzu verwenden die Writer-Libraries Daten aus dem Sourcecode des zu dokumentierenden Projektes. Dieser Code kann auch im Source Repository eines Versionsverwaltungssystems, wie zum Beispiel GIT oder SVN, eingecheckt werden. Diese Daten müssen der API dieser Writer Library entsprechen. Die Writer-Libraries sind als Erweiterungen zu bestehenden Unit-Testing-Systemen, im Moment für die Programmiersprachen Java, C# und JavaScript, umgesetzt. Somit können die für Scenarioo geschriebenen UI-Tests jeweils auch von einem Continious-Integrationsystem ausgeführt werden. Dies hat den Vorteil, dass die Scenarioo Daten jeweils auf einem aktuellen Stand gehalten werden können.

Scenarioo Datenformat

Dies ist das Format, welches von den Writer-Libraries erzeugt werden muss und vom Scenarioo Server gelesen und importiert werden kann. Das folgende Modell zeigt eine Abstraktion dieses Datenformates welches von der Scenarioo Dokumentation übernommen wurde:

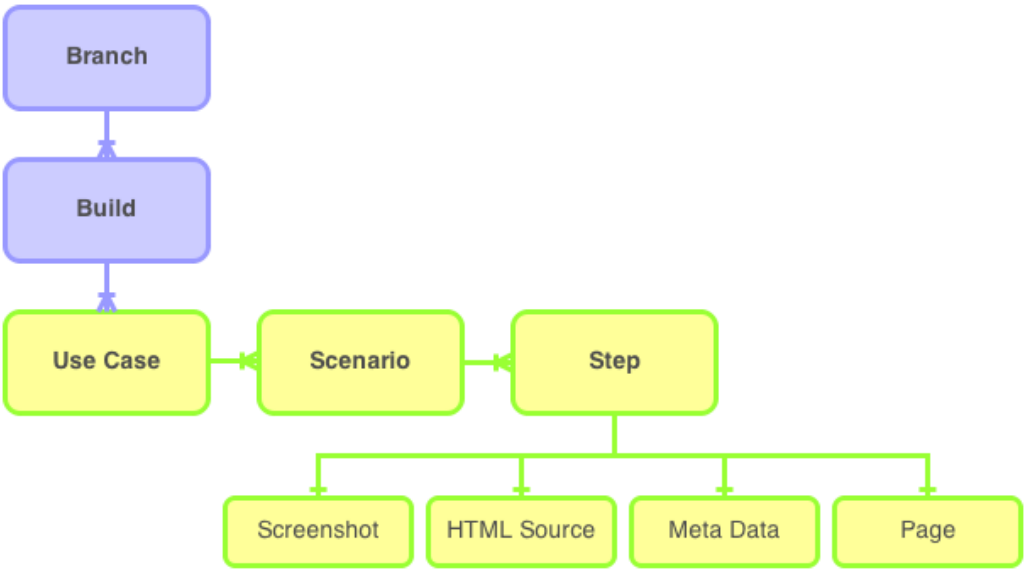


Abbildung 2: Scenarioo Domain Model

Quelle: http://scenarioo.org/docs/features/ Absatz: The Scenarioo Domain Model

Diese Anordnung der Daten wird auch im Filesystem so übernommen, inklusive der richtigen Dateinamen, damit ein reibungsloser Import auf der Serverseite gewährleistet ist. Dies ist in der Grafik auf der rechten Seite abgebildet. Die Dateien sind alle als .xml Files abgelegt.

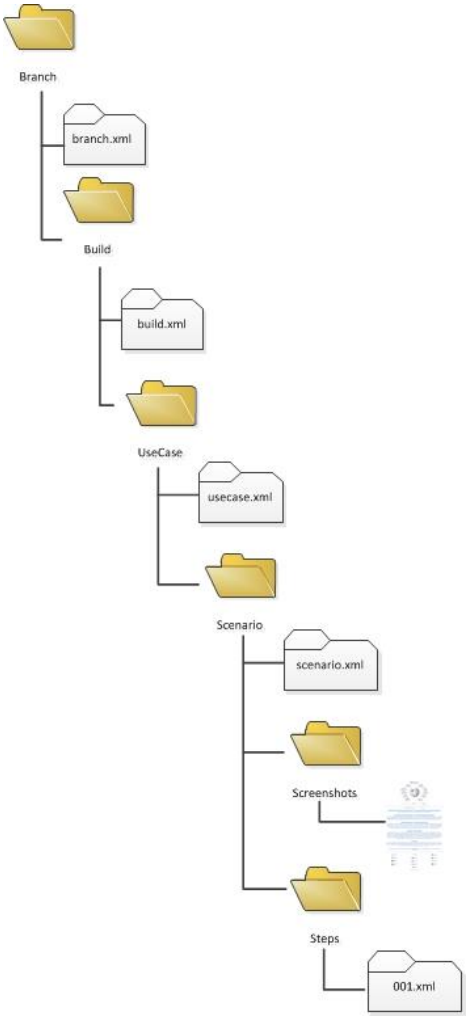


Abbildung 3: Scenarioo Ordnerstruktur

Quelle: http://scenarioo.org/docs/features/Scenarioo-Writer-Documentation-Format.html

Server

Der Server importiert die XML Dateien und führt den Screenshot-Vergleich mithilfe der «GraphicsMagick» Library durch.

Dazu erstellt der Server dann auch die XML Files der Comparison, um diese leichter in der gewünschten Form an den Client schicken zu können. Währenddessen wird auch für Elasticsearch ein Suchindex über teile des Modells erstellt.

Im Suchindex enthalten sind die Steps mit allen übergeordneten Informationen der Pages, Scenarios und Use Cases. Weiterhin wird für die Metadaten ebenfalls ein Suchindex angelegt. Weitere Informationen dazu in der folgenden Domainanalyse für die bestehende Applikation von Scenarioo.

Domainanalyse

Hier nun noch eine genauere Analyse der bestehenden Domain. In dieser Bachelorarbeit wurden «Breaking Changes» auf der Domainebene zusammen mit dem Auftraggeber geplant, weshalb eine genauere Analyse der vorgefundenen Domain unabdingbar ist.

Domainmodell

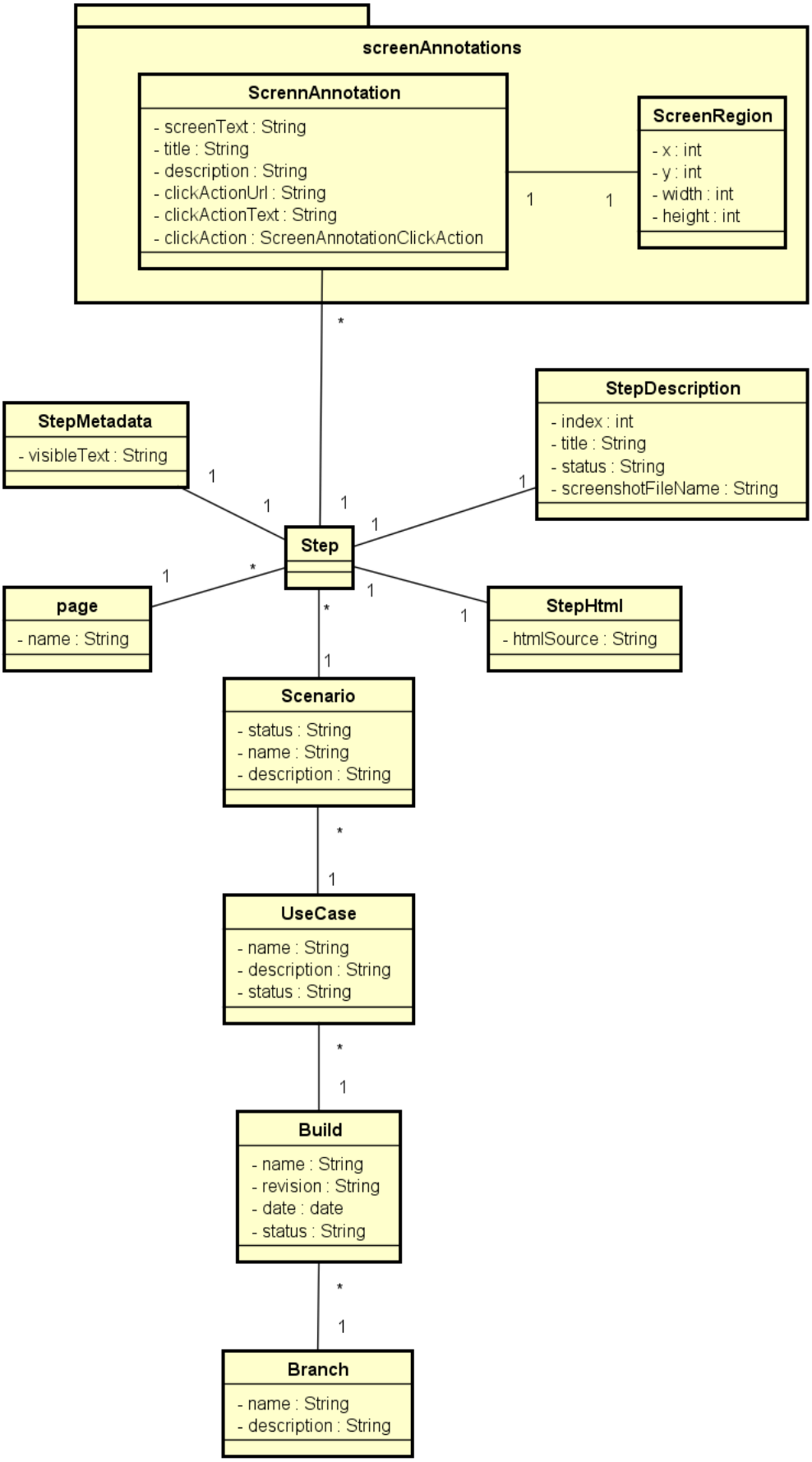


Abbildung 4:Scenarioo Domain Model ausführlich

Domainklassen

Branch und Build

Ein Branch beschreibt ein Projekt oder je nach Wunsch des Nutzers auch nur eine Version davon, beispielsweise den Branch eines Versionskontrollsystems wie GIT. Jeder Branch in Scenarioo benötigt einen eindeutigen Namen. Zu einem Branch können Aliase erstellt und verwaltet werden. Diese Alias-Mappings werden in der Scenarioo-Konfigurationsdatei gespeichert. Des Weiteren kann man dem Branch eine Beschreibung anfügen.

Ein Build beschreibt einen Snapshot in Scenarioo zu einem spezifischen Branch. So können mehrere Builds einem Branch zugeordnet werden.

Folgende Attribute weist ein Build auf:

Name	Typ	Beschreibung
name	String	Name des Builds.
revision	String	Eindeutige Erkennung innerhalb eines Branches. Kann zum Beispiel von einem Commit Hash abgeleitet werden.
date	Date	Zeitpunkt des Builds
status	String	Status Text. Scenarioo kennt «success» und «failed» alle anderen werden als «Ignored» behandelt.

Tabelle 1: Build Attribute

Zusammen bilden ein Branch und ein Build einen einzigartigen Projektstand ab. Dieser kann mithilfe des Comparison-Features mit anderen Ständen verglichen werden.

Use Case

Use Case ist jene Strukturierungskomponente welche in dieser Bachelorarbeit erweitert werden soll. Im alten Stand des Projektes Scenarioo konnten hiermit die UI-Testszenarien gruppiert werden.

Folgende Attribute weist ein Use Case auf:

Name	Typ	Beschreibung
name	String	Name des Use Cases. Eindeutig innerhalb eines Builds
description	String	Eine Beschreibung des Use Cases.
status	String	Status Text. Scenarioo kennt «success» und «failed» alle anderen werden als «Ignored» behandelt.

Tabelle 2: Use Case Attribute

Ein Build kann immer mehrere Use Cases beinhalten.

Scenario

Scenario beschreibt ein UI-Testszenario.

Folgende Attribute weist ein Scenario auf:

Name	Typ	Beschreibung
name	String	Name des Scenaro. Eindeutig innerhalb eines Use Cases
description	String	Eine Beschreibung des Scenarios.
status	String	Status Text. Scenarioo kennt «success» und «failed» alle anderen werden als «Ignored» behandelt.

Tabelle 3: Scenario Attribute

Ein Use Case kann mehrere Szenarien beinhalten.

Step

Ein Step beschreibt einen Schritt im Testszenario. Diese Schritte können im Client der Reihe nach angesehen werden. Mehr dazu im Absatz Client.

Ein Scenario kann mehrere Steps beinhalten.

StepDescription

Die StepDescription beschreibt einen Step. Die StepDescription weisst folgende Attribute auf:

Name	Typ	Beschreibung
index	int	Der Index gibt die Position des Steps im Scenario an.
title	String	Titel für den Step.
status	String	Status Text. Scenarioo kennt «success» und «failed» alle anderen werden als «Ignored» behandelt.
screenshotFile-Name	String	Den Namen des dazugehörigen Screenshots.

Tabelle 4: StepDescription Attribute

Ein Step hat immer eine StepDescription

StepMetadata

Die StepMetadata beschreibt, was auf dem Screenshot des Steps zu sehen ist. Dieser Text wird benötigt für die Volltextsuche.

Name	Typ	Beschreibung
visibleText	String	Text, welcher den Inhalt des Screenshots widerspiegeln kann, um eine Volltextsuche für den Step zu ermöglichen.

Tabelle 5: StepMetaData Attribute

Ein Step hat immer ein StepMetadata Objekt angehängt.

StepHtml

Falls das Testszenario von einer Website stammt, ist hier das HTML hinterlegt welches beim Erstellen des Screenshots im DOM geladen war.

Page

Eine Page beschreibt die Ansicht, welche auf dem Screenshot eines Steps dargestellt wird. Da mehrere Steps auf die gleiche Ansicht zeigen können, zum Beispiel durch das hin und her Navigieren zwischen zwei Ansichten, dient die Page als Gruppierungselement für die einzelnen Steps in einem Scenario. Eine Page kann mehrere Steps beinhalten.

Name	Typ	Beschreibung
name	String	Name der Page.

Tabelle 6: Page Attribute

screenAnnotation Package

Auf einem Step können Annotations angefügt werden. Diese beschreiben Aktionen, welche im Scenarioo Client dann als weitere Navigationsmöglichkeit dienen.

ScreenAnnotation

ScreenAnnotation beschreibt die auszuführende Aktion.

Name	Typ	Beschreibung
screenText	String	Text für die ScreenAnnotation.
title	String	Titel für den ScreenAnnotation.
description	String	Eine Beschreibung der ScreenAnnotation.
clickActionUrl	String	URL welche besucht wird.
clickActionText	String	Text für den Link.
clickAction	ScreenAnnotationClickAction	ScreenAnnotationClickAction ist ein ENUM mit folgenden Möglichkeiten: TO_NEXT_STEP: Geht zum nächsten Step im Szenario. TO_URL: Geht zur URL, welche als clickActionUrl angegeben wurde.

Tabelle 7: ScreenAnnotation Attribute

ScreenRegion

Gibt die Region in Pixeln auf dem Screenshot an, wo die ScreenAnnotation platziert werden soll.

Name	Typ	Beschreibung
x	int	Horizontale Position der ScreenAnnotation.
y	int	Vertikale Position der ScreenAnnotation.
width	int	Breite der ScreenAnnotation.

height	int	Höhe der ScreenAnnotation.

Tabelle 8: ScreenRegion Attribute

Labels

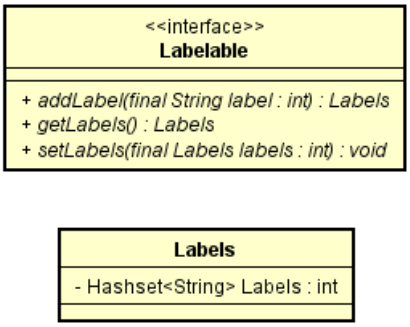


Abbildung 5: Scenarioo Labels

Eine weitere Funktionalität bieten Labels. Labels können an mehrere Klassen auf der Domainebene angehängt werden:

- UseCase
- Scenario
- StepDescription
- Page

Diese Klassen Implementieren das Interface Labelable und nutzen die Labels Klasse um Labels zu speichern.

Die Labels werden dann beim Client angezeigt.

Generic Details

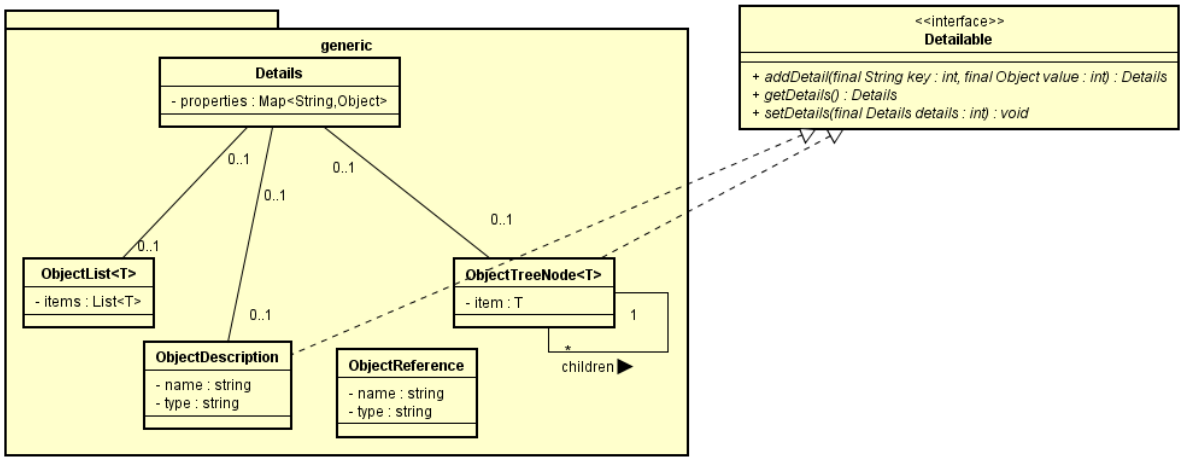


Abbildung 6: Scenarioo Details

Generic Details beschreiben custom Metadaten, welche an verschiedenen Objekten der Domainebene angehängt werden können. Hierzu muss die Klasse des Objektes das Interface Detailable implementieren.

Dieses Interface implementieren folgende Klassen:

- Branch
- Build
- UseCase
- Scenario
- StepDescription
- StepMetadata
- ScreenAnnotation
- Page
- ObjectDescription
- ObjectTreeNode<T>

Klassenbeschreibungen

Details

Details ist die Hauptklasse der Generic Details, wenn eine Klasse das Interface Detailable implementiert, so besitzt die klasse auch immer eine Reference auf Details.

Details beschreibt einen Key-Value Storage. Als Value kommen alle Klassen des Packages generic in Frage, sowie die Java Klasse String.

Als Key wird jeweils ein String verwendet.

ObjectList<T>

ObjectList beschreibt die Implementierung einer Liste. Diese Klasse dient der Darstellung der in der Liste gespeicherten Objekte. Diese Objekte können wiederum alle Klassen des Packages generic oder Strings sein.

ObjectTreeNode<T>

ObjectTreeNode beschreibt die Implementierung einer Baumstruktur. Diese Klasse dient der Darstellung der in der Baumstruktur gespeicherten Objekte. Diese Objekte können wiederum alle Klassen des Packages generic oder Strings sein. Des Weiteren kann jede TreeNode weitere TreeNodes als «Children» besitzen.

ObjectReference

ObjectReference beschreibt eine Referenz auf ein anderes Objekt welches in den Generic Details gespeichert ist. Hier können der Name und der Typ des Objektes angegeben werden.

ObjectDescription

ObjectDescription beschreibt ein Objekt mit einem Typ. Für jeden unterschiedlichen «type» welcher in den Generic Details abgespeichert ist, erstellt der Server einen Suchindex für die Volltextsuche.

Name	Typ	Beschreibung
name	String	Name des Objektes
type	String	Typ des Objektes, für welchen ein Suchindex existiert.

Tabelle 9: ObjectDescription Attribute

Client

In der Webansicht des Clients kann der Benutzer einen Branch und einen Build aussuchen, um dann genau diesen Projektstand zu betrachten. Des Weiteren kann er auch eine «Comparison» auswählen, falls überhaupt eine vorhanden oder konfiguriert ist, um die Änderungen der UI-Testszenarien zu einem anderen Build zu sehen. Diese Funktion erlaubt es detaillierte Vergleiche zwischen den Änderungen der in den einzelnen Steps gespeicherten Screenshots zu erhalten.

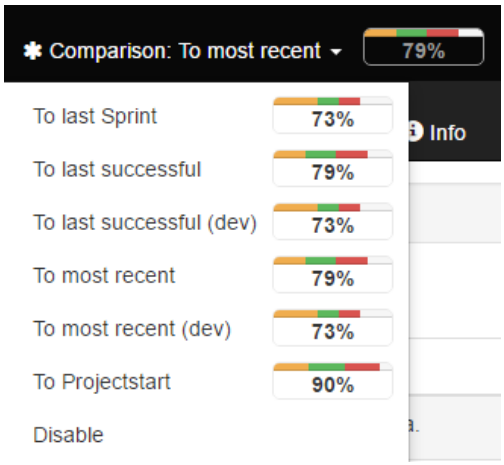


Abbildung 7: Comparison Auswahl

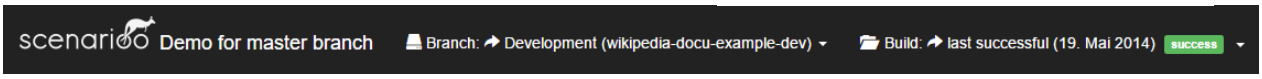


Abbildung 8: Branch und Build Auswahl

Wenn der Nutzer einen Brach und einen Build gewählt hat, so werden die Use Cases, welche in diesem Build vorhanden sind, aufgelistet. In dieser Tabelle kann der Nutzer den jeweiligen Status, den Namen, die Beschreibung und die Anzahl darunter geordneter Scenarios sehen. Falls eine Comparison ausgewählt wurde, werden die Veränderungen des Use Cases in einer weiteren Spalte angezeigt. Diese zeigt zum Beispiel ob ein Use Case neu hinzugekommen ist, wenn ein Use Case gelöscht wurde, oder die prozentuale Änderung aller Screenshots in den Scenarios des Use Cases.

Auf der rechten Seite neben der Tabelle werden weitere Informationen zum gewählten Branch und zum gewählten Build angezeigt.

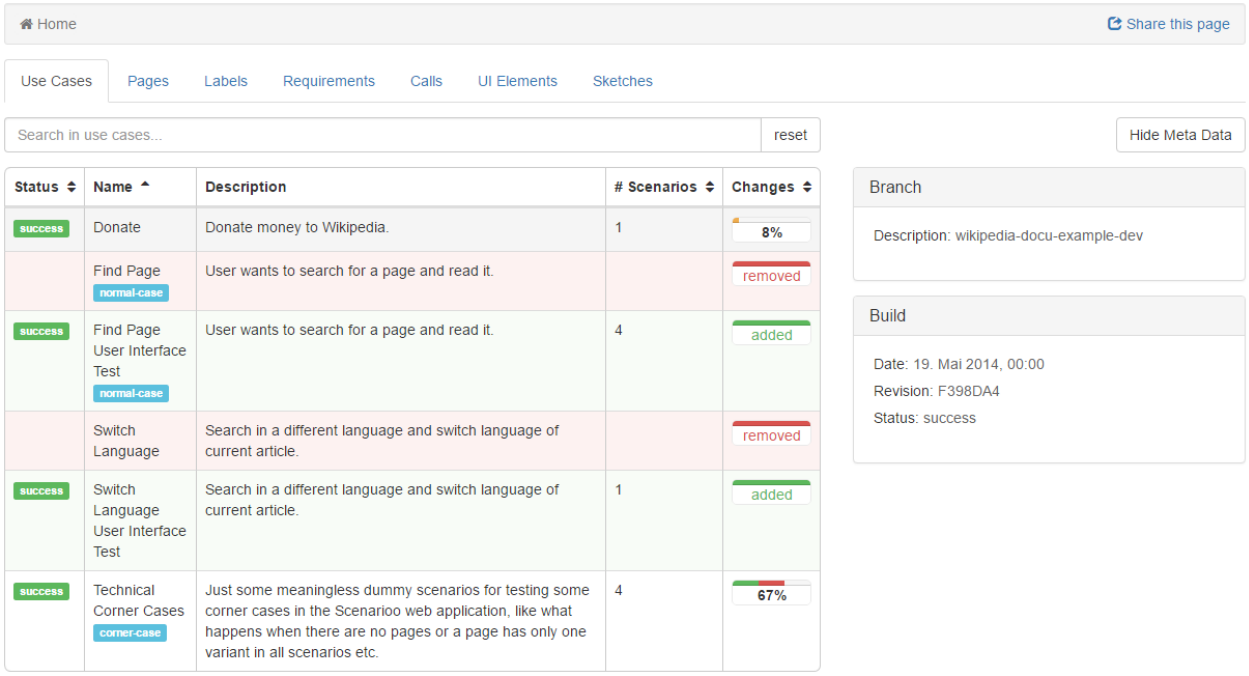


Abbildung 9: Bestehende Use Cases-View

Auf der linken Seite die Use Cases, auf der rechten Seite die Metadaten dazu.

Der Nutzer kann nun einen dieser Use Cases auswählen und sieht nun einen Überblick über den gewählten Use Case.

In einer ähnlichen Tabelle sind nun die Informationen zu den Scenarios aufgelistet, welche dem Use Case untergeordnet sind. In den Spalten ist wiederum der Status, der Name, weiterführende Actions, eine Beschreibung und die Anzahl der Steps für den Use Case aufgelistet. Falls eine Comparison ausgewählt ist, wird auch hier die Veränderung zu einem anderen Build angezeigt, analog zum Use Case.

Auf der rechten Seite neben der Tabelle sind Detail Informationen zum Use Case, Labels und Custom Metadata, welche dem Use Case angehängt ist, angezeigt.

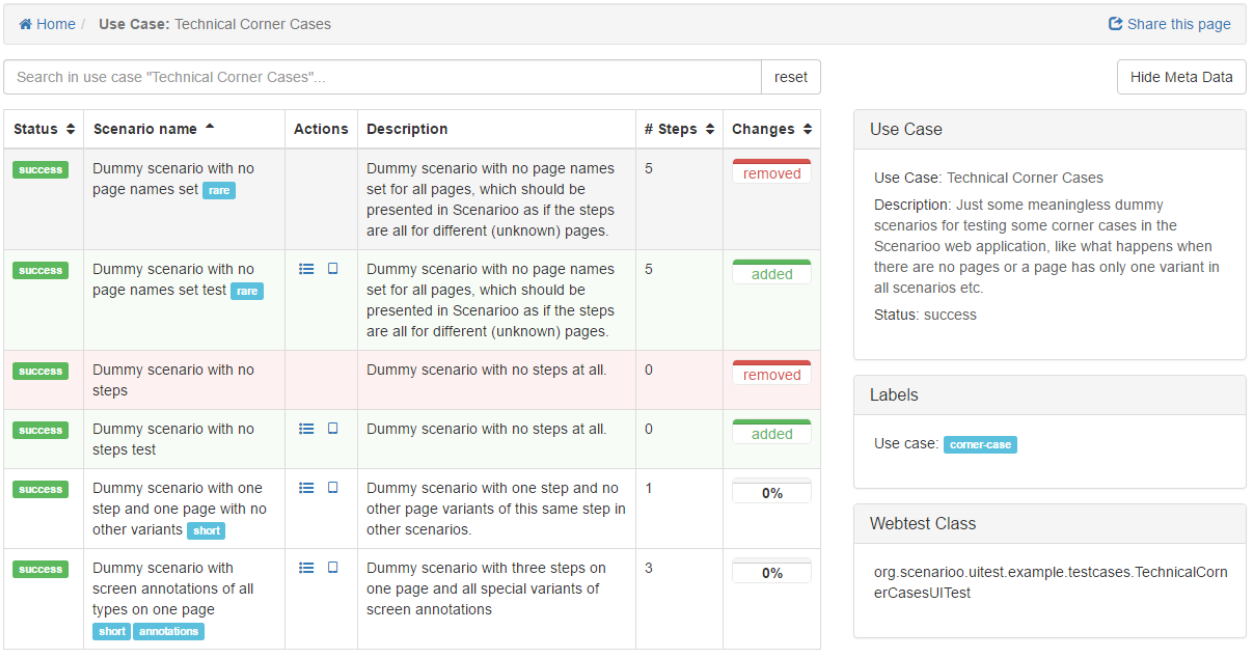


Abbildung 10: Bestehende Scenario-View

Auf der linken Seite die Szenarien eines Use Cases, auf der rechten Seite Metadaten zum Use Case und Labels.

In der Tabelle kann der Nutzer nun ein Scenario auswählen, oder die Actions nutzen.

Die erste Action bewirkt dasselbe wie das auswählen des Scenarios aus der Tabelle. Dem Nutzer wird die Übersichtsseite des Scenarios angezeigt.

Die zweite Action überspringt diese Übersicht und geht gleich zur Detail-View für den ersten Step.

Auf der Übersichtsseite des Scenarios werden Previews der einzelnen Pages angezeigt und die ersten Steps für diese Page. Falls eine Comparison ausgewählt wurde, wird über der Preview jeweils noch die Änderungen der Page angezeigt.

Auf der rechten Seite sieht der Nutzer wiederum Detailinformationen zu dem gewählten Scenario und Custom Metadata welche an das Scenario angehängt ist.

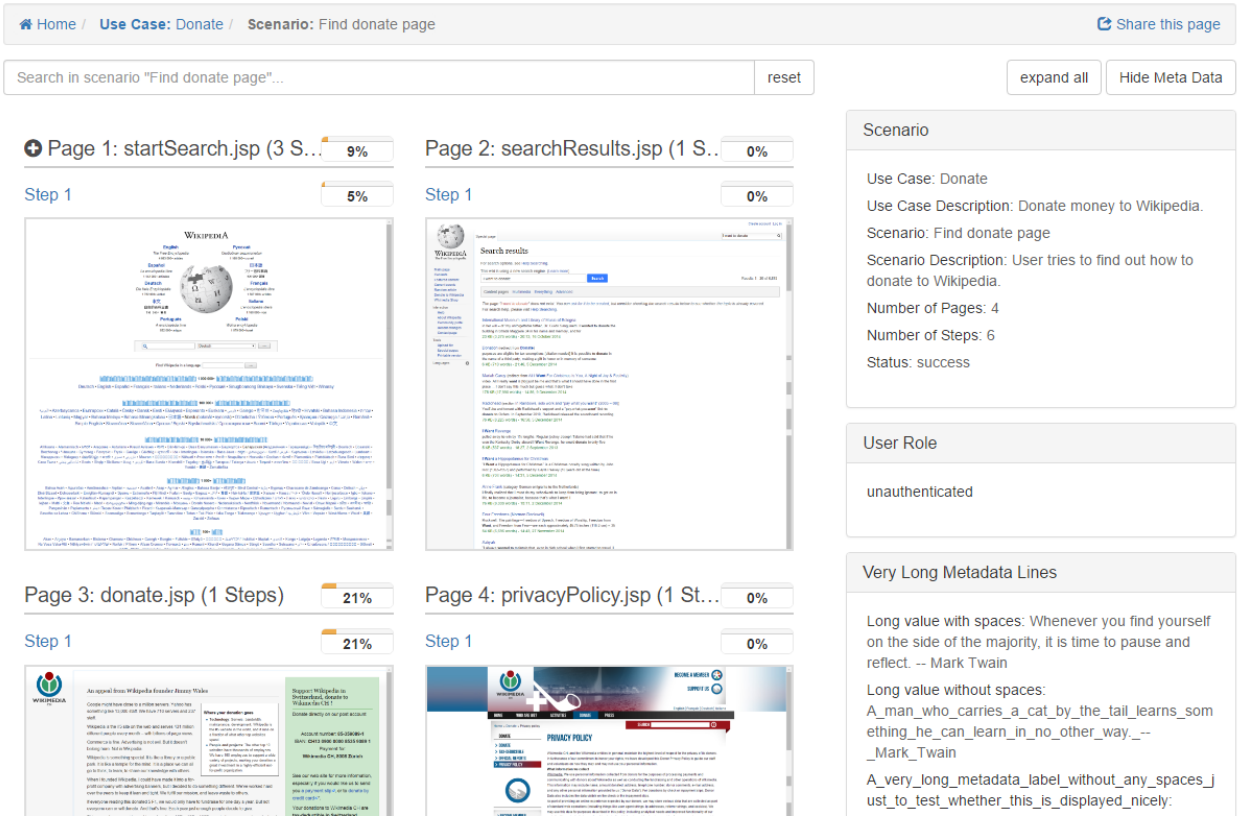


Abbildung 11: Bestehende Step-View

Auf der linken Seite die einzelnen Pages zu welchen Scenario Steps existieren und auf der rechten Seite wieder Metadaten.

Mit dem Plus Button neben der ersten Page, können auch direkt hier alle Steps angezeigt werden, auch die Steps, welche auf derselben Seite bleiben, wie zum Beispiel das eingeben von Informationen in ein Inputfeld.

Nach dem Auswählen eines Steps kommt die Detail-View:

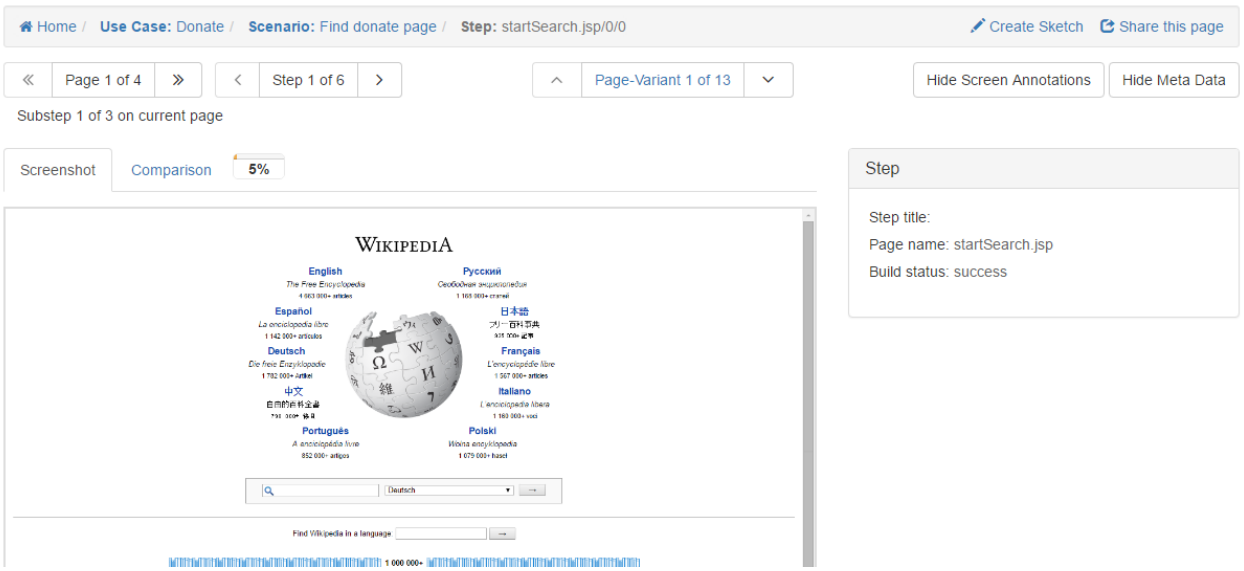


Abbildung 12: Bestehende Page-View

Auf der linken Seite die jeweilige Ansicht, auf der rechten Seite die Metadaten.

Oberhalb sind drei Navigationsmöglichkeiten

1. Page Navigation innerhalb des Scenarios (Page 1 of 4)

   Hier können die verscheiden Pages durchgeblättert werden.

2. Step Navigation innerhalb des Scenarios (Step 1 of 6)

   Hier können die Steps ausgewählt werden, welche im aktuellen Scenario. In der Reihenfolge des jeweiligen Scenarios

3. Page-Variant Navigation (Page-Variant 1 of 13) diese ist Use Case und Scenario übergreifend.

   Hier können alle Steps angesteuert werden, welche im aktuell gewählten Build existieren.

Wird auf der Detail-View auf den Tab Comparison gewechselt, sieht der Nutzer direkt die Veränderungen des gewählten Steps.

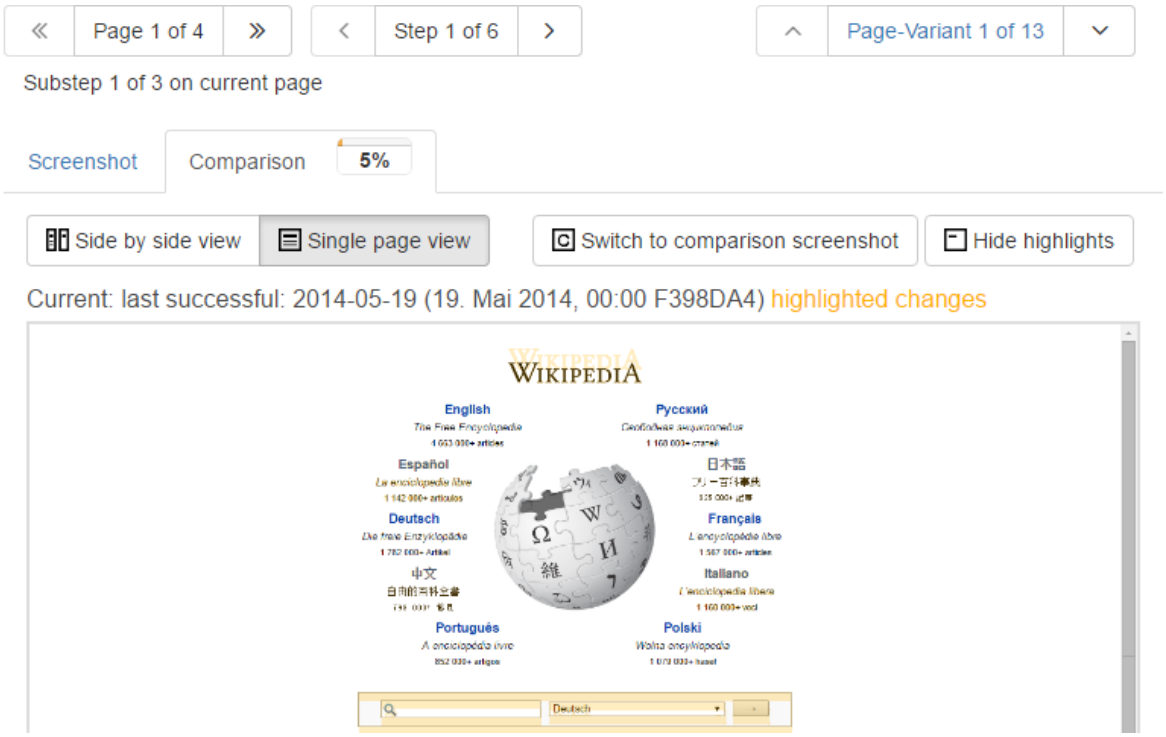


Abbildung 13: Page Comparison

Oben rechts kann Hier zusätzlich ein Sketch zu der jeweiligen Seite erstellt werden:

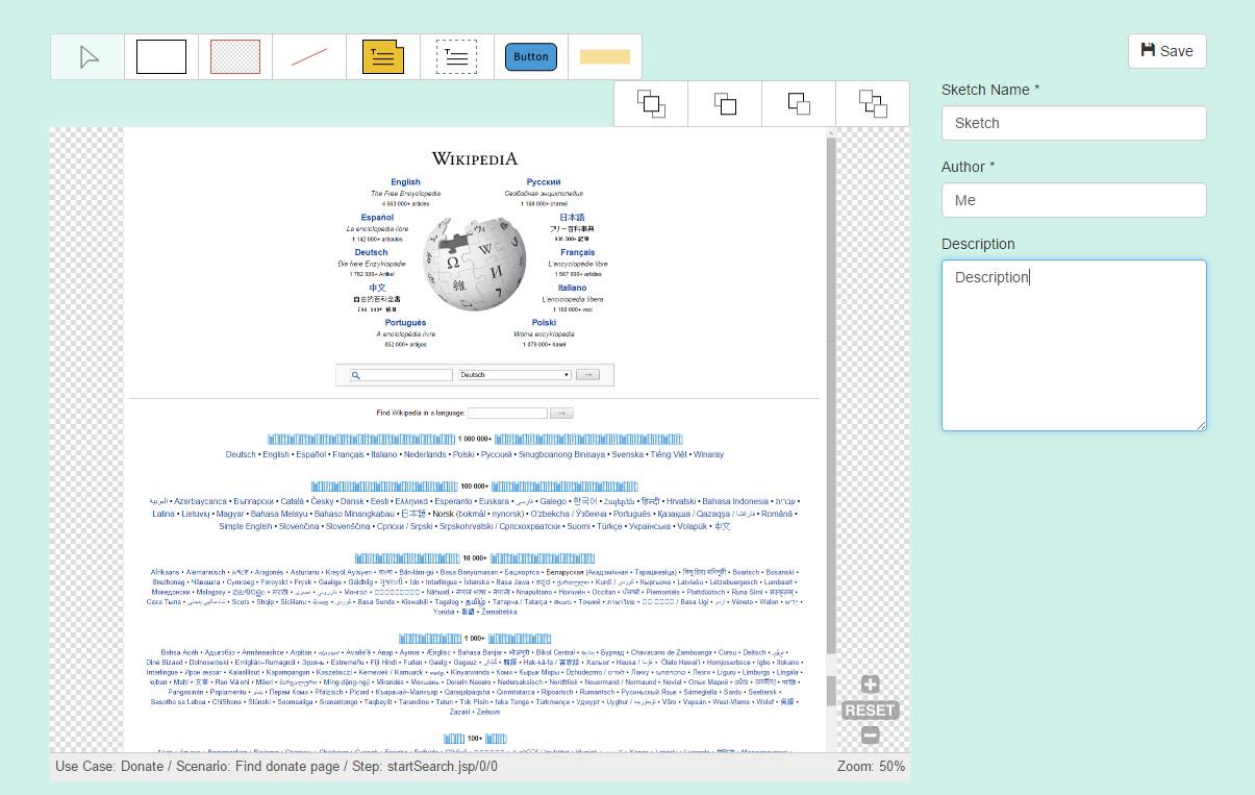


Abbildung 14: Sketch Editor

Der Sketch Editor verfügt über folgende Funktionen:

- Rechteck zeichnen
- Roten rechteckigen Rahmen zeichnen
- Rote Linie zeichnen
- Notizzettel erstellen und platzieren
- Text platzieren
- Button zeichnen
- Rechteckige Marker Funktion

Die Elemente können angeordnet werden:

- Bring to Front
- Bring forward
- Send backward
- Send to back

Des Weiteren verfügt der Sketch Editor über eine Zoomfunktion.

Durch das eingeben eines Namens und eines Autors und das klicken auf den Button «Save» kann der Sketch gespeichert werden. Hier kann auch noch eine optionale Beschreibung hinterlegt werden.

Nach dieser Abgrenzung zu dem gegebenen Projekt, nun aber weiter im Kontext. Für eine Bachelorarbeit in der Software Entwicklung, ist Planung wichtig, auch wenn die agilen Methodiken immer populärer werden, darf auch bei der Anwendung dieser, nicht ganz auf eine Projektplanung verzichtet werden.

5.3 Planung

Wie bereits in der Einleitung angedeutet, ist diese Arbeit in grob zwei Teile trennbar. Einerseits das Konzept, welches für die Weiterentwicklung von Scenarioo, in enger Zusammenarbeit mit Mitgliedern des Scenarioo Projektes entwickelt wurde, andererseits die Implementation des Prototyps, welcher wiederum in Rücksprache mit besagten Mitarbeitern entwickelt wurde.

5.3.1 Projektplan

5.3.1.1 Zeitbudget

Für die Umsetzung der Bachelorarbeit haben wir 17 Wochen Zeit und ein Stundenpensum von 2 x 350 Stunden.

5.3.1.2 Iterationen

Geplant sind jeweils zweiwöchige Iterationen, plus eine Woche Reserve.

Die Iterationen gehen jeweils von Montag bis Sonntag, zwei Wochen später.

In den Wochen vor Semesterende, also den ersten 15 Wochen ist ein Stundenpensum von durchschnittlich 21 Stunden pro Person geplant.

In der Letzen Woche der Iteration (T1) Transition ist ein Stundenpensum von ca. 28 Stunden geplant.

Die Verbleibenden 7 Stunden werden in die Reserve genommen, um die Arbeit abzuschliessen.

Die Entwicklung wird in die Phasen Inception, Elaboration, Construction und Transition aufgeteilt. Die einzelnen Iterationen tragen die Namen der Phasen, wobei die Iterationen innerhalb einer Phase durchnummeriert werden. Die Iterationen werden in Redmine verwaltet.

Für die Inception haben wir uns in diesem Fall die ersten zwei Wochen reserviert, da der Kickoff-Termin auf den Donnerstag in der ersten Woche viel. Somit haben wir die zweite Woche genutzt zur Einarbeitung und zum Verständnis des bestehenden Projektes.

Hiernach werden drei Iterationen zur Elaboration und zum Abstecken der Problemdomäne Verwendet.

Die Konzeptionsphase geht nach End-Of-Elaboration fliessend in die Implementationsphase über. Hier wird jedoch keine klare Trennung gemacht, da sich diese beiden Teile der Bachelorarbeit teilweise in den Tätigkeiten überlappen.

Die detaillierte Planung und Verwaltung der Arbeitspakete erfolgt in Redmine. Die Planung wird laufend aktualisiert und den gegebenen Umständen in Absprache mit dem Industriepartner angepasst.

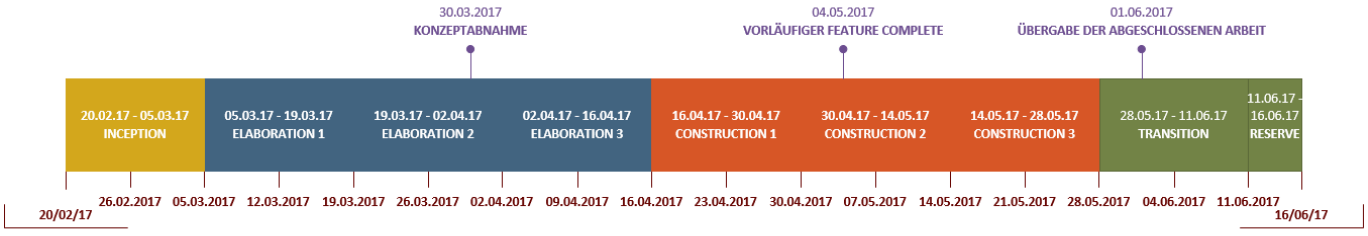


Abbildung 15: Zeitsrahl der Projektplanung

5.3.1.3 Meilensteine

Zur Überprüfung des Fortschritts werden vier Meilensteine gesetzt:

Datum	Beschreibung	Name
30.03.2017	Konzeptabnahme durch Zühlke	MS1
04.05.2017	Vorläufiger Feature Complete	MS2
09.06.2017	Übergabe der abgeschlossenen Arbeit	MS3
16.06.2017	Abgabe Bericht und Arbeit an den Betreuer	

Tabelle 10: Meilensteine

5.3.1.4 Weitere Temine

Usability-Test

Am 18.05.2017 wurde ein Usability-Test mit den Mitarbeitern der Zühlke gemacht um sicherzustellen, dass unser Produkt auch nutzerfreundlich und benutzbar ist.

Besprechungen mit Betreuer

Für die Besprechungen mit dem Betreuer der Bachelorarbeit ist jeweils der Freitagnachmittag um 15:00 Uhr eingeplant. Terminänderungen je nach Verfügbarkeit vorbehalten.

5.3.1.5 Nachträge

Ausschluss des Features Testreport / Testlog

In Absprache und durch die Priorisierung der weiteren Anforderungen wurde die Anforderung «Testlog / Testreport», dokumentiert in dem Use Case 11: «Testreport/Testlog zu Feature ansehen» und in der Domainanalyse Kapitel: Testreport/Testlog, vom Implementationsscope dieser Bachelorarbeit ausgeschlossen.

5.4 Requirements

5.4.1 Vorgehen

In mehreren Sitzungen mit der Zühlke wurde das neue Struktur-Konzept entwickelt. Dies wurde in einem iterativen Prozess mit Hilfe und mit den Ideen der Zühlke Mitarbeiter entwickelt. Der iterative Prozess umfasste mehrere Meetings in welchen das Konzept diskutiert und dadurch verfeinert werden konnte. Das Endprodukt dieser Phase ist in diesem Teil der Dokumentation ersichtlich.

In diesem iterativen Prozess wurden die benötigten Ansichten für die Weiterentwicklung, eine Domainanalyse für die Weiterentwicklung und die benötigten Anforderungen, in Form von Use Cases, User Stories, User Centered Design Scenarios und nicht-Funktionalen Anforderungen, für die Weiterentwicklung erstellt.

Mitte Projekt wurden Testdaten für die Weiterentwicklung benötigt, diese konnten zu dem Zeitpunkt noch nicht vom Industriepartner geliefert werden, weshalb für dieses Projekt eine Self-Documentation auf der neuen Datenstruktur realisiert wurde.

5.4.2 Domainanalyse

Hier werden nur die Teile der Domainebene nochmals behandelt welche sich im Laufe der Arbeit geändert haben. Für eine komplette Übersicht siehe im Kapitel Vorgabe/Domainanalyse.

5.4.2.1 Feature-Struktur

Die erste grosse Änderung wurde am ehemaligen Use Case vorgenommen. Dieser wurde in dieser Bachelorarbeit zum komplexeren Feature umgestaltet. Komplexer in der Hinsicht, dass es nun möglich ist, aus mehreren Features eine Baumstruktur zu erstellen. Dies dient der Möglichkeit einem Projekt eine grössere Strukturierungsfreiheit zu geben. Bis anhin waren die Use Cases lediglich als Liste oder Tabelle darstellbar. Eine Baumstruktur soll zum Beispiel die Gruppierung einzelner Teile eines Projektes nach Themengebiet im Projekt ermöglichen.

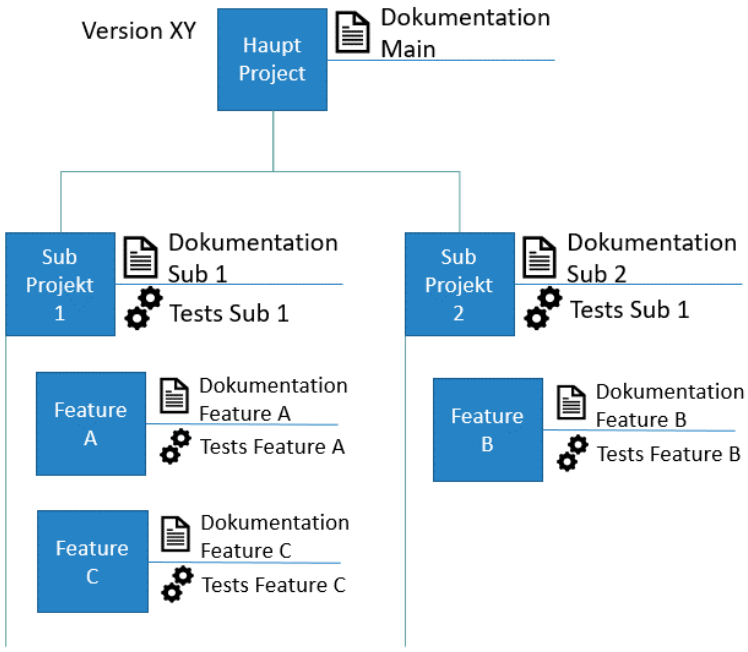


Abbildung 16:Strukturbeispiel

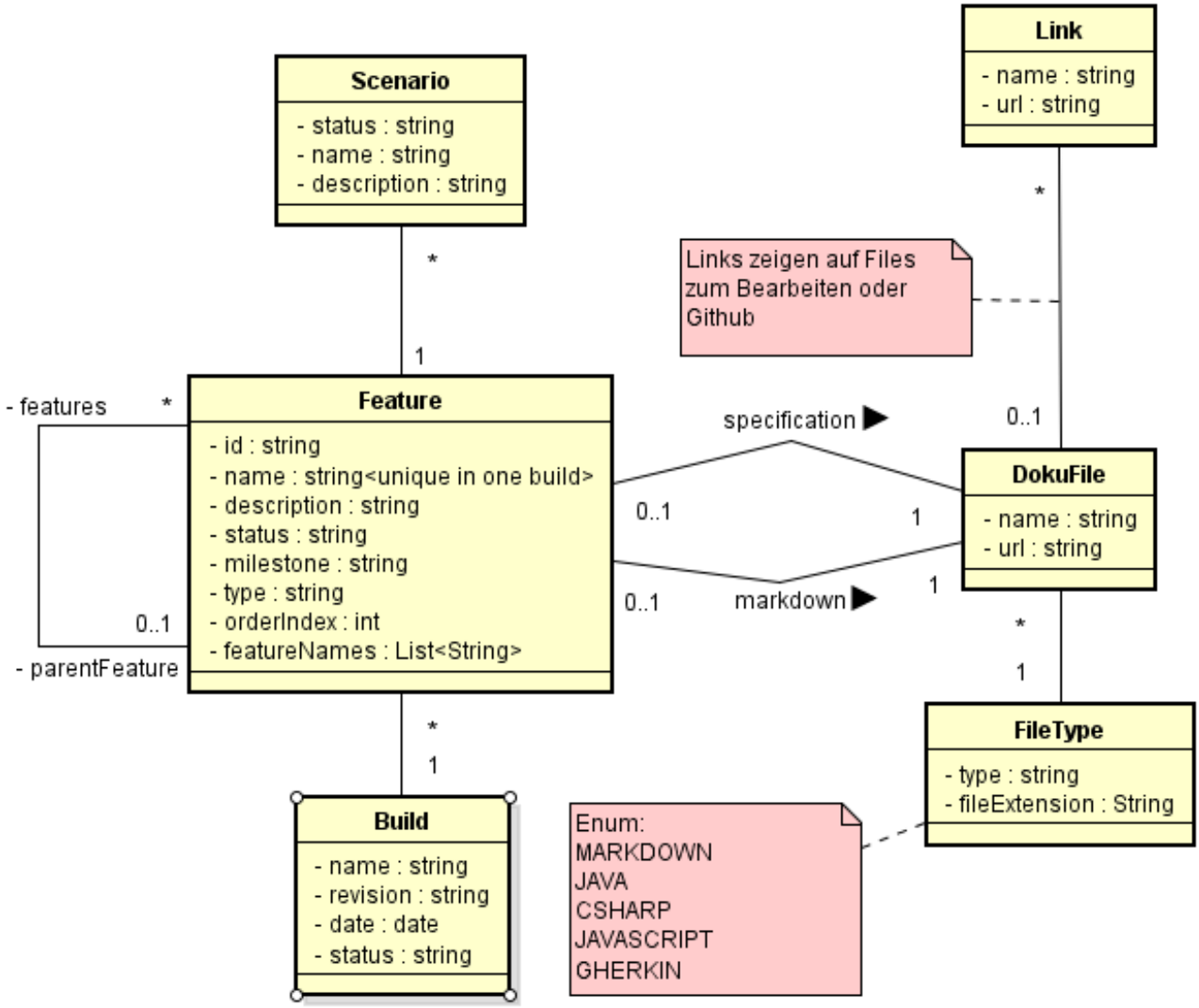


Abbildung 17: Neue Domain Objekte

Feature

Feature ist ein elementares Strukturelement. Hiermit kann ein in Scenarioo dokumentiertes Projekt, die einzelnen Elemente dieses Projektes abbilden.

Ein Feature gehört immer zu einem Build und ein Build kann mehrere Features beinhalten.

Ein Feature kann mehrere Child-Features beinhalten, sogenannte Subfeatures. Somit kann mit diesem Element eine hierarchische Baumstruktur erstellt werden, welche in einem Projekt zur Ordnung von einzelnen Teilen genutzt werden kann.

Ein Feature weisst folgende Attribute auf:

Name	Typ	Beschreibung
id	String	Eindeutiger Name, muss mit Ordnernamen für den Import übereinstimmen.
name	String	Name des Features

description	String	Beschreibung des Features
status	String	Status Text. Scenarioo kennt «success» und «failed» alle anderen werden als «Ignored» behandelt.
milestone	String	Sortierbarer Text, welcher z.B. den jeweiligen Release des Features angibt. Wird im Frontend für die Swimlanes auf der Map verwendet.
type	String	Benutzerdefinierbarer Typ welcher im Frontend angezeigt wird
orderIndex	int	Zahlenwert, zur Sortierung der Position in der Map-View
featureNames	List of Strings	Ids der Subfeatures. Diese müssen mit der jeweiligen Id und dem Ordnernamen des Subfeatures übereinstimmen.

Tabelle 11: Feature Attribute

Relationen:

Name	Typ	Beschreibung
features	List of Feature	Subfeatures welche dem Feature angehängt sind. Dieses Attribut wird automatisch erzeugt und muss nicht von der Writer Library geschrieben werden.
documentation	DocFile	Angehängtes Markdown File. Zur Dokumentation des Features.
specification	DocFile	Angehängtes Spec File, dies kann z.B. ein Gherkin File, oder sonst ein Test File, für z.B. automatisierte Tests sein.
parentFeature	Feature	Wird auf der Clientseite generiert, dient der Rückwärtsnavigation
scenarios	List of Scenario	Testszenarios welche zum Feature gehören.

Tabelle 12: Feature Relationen

Link

Das Link Element wurde eingeführt, um das Erstellen von benutzerdefinierbaren Links für die Dokumentations-Artefakte zu externen Quellen, wie zum Beispiel der Originaldatei in einem Souce-Code Versionsverwaltungssystems wie GIT auf GitHub, zu ermöglichen. Diese Links sollen dem Nutzer mit einem benutzerdefinierbaren Namen angezeigt werden.

Attribute:

Name	Typ	Beschreibung
name	String	Lesbarer Name des Links
url	String	Die URL zur externen Ressource, beispielsweise eine Webseite.

Tabelle 13: Link Attribute

DocFile

Ein Feature hat maximal zwei Referenzen zu DocFiles. Dies ist einmal ein Markdown-File und zum zweiten ein Spezifikationsfile, welches eine ausführbare Testdefinition des Features enthält. Dies können Beispielsweise Gherkin-Files, Unit-Testfiles oder automatisierbare Integrationtestfiles sein.

Attribute:

Name	Typ	Beschreibung
name	String	Name des Dokuments
url	String	URL zum Dokument. Dies kann entweder eine URL zu einer externen Resource sein, oder aber auch ein relativer Pfad im Documentation Folder des gewählten Builds. Mehr hierzu im Kapitel Implementation.

Tabelle 14: DocFile Attribute

Relationen:

Name	Typ	Beschreibung
links	List of Link	Ein File kann mit diversen Links ergänzt werden, z.B. zum Original auf GitHub oder einem Link ins Internet für weitere Infos
type	FileType	Jedes DocFile hat einen der vorgegebenen Typen.

Tabelle 15: DocFile Relationen

FileType

FileType definiert die akzeptierten Formate welche als DocFile angehängt werden können. Jedes DocFile muss eines dieser Formate besitzen um angezeigt werden zu können.

Folgende Formate sind akzeptiert:

- JAVA (mit dem «type»:"Java" und der «fileExtension»:".java")
- MARKDOWN (mit dem «type»:"Markdown" und der «fileExtension»:".md")
- CSHARP (mit dem «type»:"C #" und der «fileExtension»:".cs")
- JAVASCRIPT (mit dem «type»:"JavaScript" und der «fileExtension»:".js")
- GHERKIN (mit dem «type»:"Gherkin" und der «fileExtension»:".feature")

Attribute:

Name	Typ	Beschreibung
type	String	Ist der Name des Typs als ausgeschriebenes Wort
fileExtension	String	Ist die offizielle Endung des Dateiformats.

Tabelle 16: FileType Attribute

5.4.2.2 Testreport/Testlog

Eine weitere mögliche Erweiterung ist das Darstellen von Testreports und Testlogs im Client. Wie bereits in der Planung beschrieben, wurde dieses Feature in Rücksprache mit dem Industriepartner jedoch von dem Implementationsscope ausgeschlossen. Hier wird aber dennoch die Veränderung, welche in einem Meeting mit den Mitarbeitern der Zühlke besprochen wurde auf der Domainebene festgehalten.

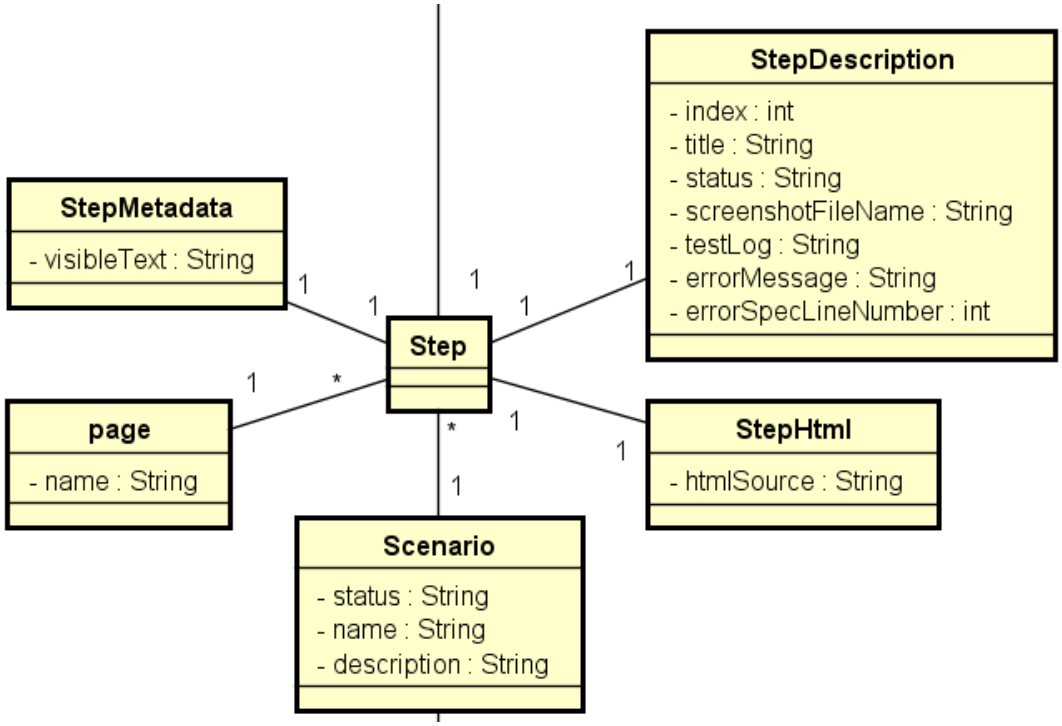


Abbildung 18: Idee für das Anzeigen von Testlogs

Diese Erweiterung befasst sich mit der StepDescription. Drei weitere Attribute müssen ergänzt werden.

Name	Typ	Beschreibung
testLog	String	Test Output oder anderer Benutzerdefinierbarer Text welcher bei diesem Step angezeigt werden soll.
errorMessage	String	Die Fehlermeldung eines allfällig Fehlgeschlagenen Tests
errorSpecLineNumber	int	Zeilennummer des am Feature angehängten Specification Files auf welcher der in «errorMessage» geschriebene Fehler passiert ist.

Tabelle 17: StepDescription Erweiterung

5.4.3 Ansichten

In der Konzeptphase wurden in Zusammenarbeit mit dem Industriepartner Ansichten zuerst in Form von Mockup-Skizzen erstellt, dann verfeinert. Später wurde zur Planung dieser Ansichten ein «Clickable-Prototype» mit Dummy-Daten implementiert um die Views besser mit dem Industriepartner besprechen zu können.

5.4.3.1 Navigationskonzept

Da Scenarioo auch nach der Erweiterung dieser Bachelorarbeit immer noch eher als Software für «Experten» Nutzer gedacht ist, sprich eine gewisse Einarbeitung in die Konzepte vorausgesetzt werden kann, ist es umso wichtiger ein einheitliches Benutzerführungskonzept zu haben.

Dazu wurden für die neuen Ansichten einige wichtige Punkte in Absprache mit dem Industriepartner festgehalten, welche hier Dokumentiert sind.

Kontextgebundene Navigation

Unter kontextgebundener Navigation wird hier der Punkt beschreiben, dass im Frontend immer ein aktuell gewähltes Feature gespeichert wird. Dem Nutzer sollen auf allen neuen Ansichten Informationen zu diesem gewählten Feature angezeigt werden. Hierzu kommt der Punkt, dass von dem aktuell gewählten Feature entweder weiter zu seinen Subfeatures navigiert werden kann oder wieder zurück zu seinem «parentFeature». Auf der Scenario-View, auf welcher alle Scenarios zu einem Feature aufgelistet sind, soll zudem ein Link zu dem entsprechenden Testszenario existieren.

Navigation zu Subfeatures

Auf drei der vier neuen Ansichten soll es möglich sein zu den Subfeatures weitere Informationen zu sehen und zu diesen navigieren zu können. Auf der Feature-View sollen alle Subfeatures zu dem aktuell gewählten Feature in einer Tabelle aufgelistet werden, auf der Documentation-View sollen alle angehängten Dokumentations-Artefakte zum aktuell gewählten Feature und zu allen Subfeatures des aktuell gewählten Features angezeigt werden können. Die Subfeatures sollen ausgewählt werden können um so zu ihnen zu navigieren. Auf der Map-View sollen sowohl die Subfeatures, als «Backbone» der Map angezeigt werden, wie auch die Subfeatures der Subfeatures gruppiert nach dem entsprechenden «milestone» Attribut.

Navigieren zum Parent-Feature

Um eine simple Aufwärtsnavigation zu gewährleisten, wurden die Breadcrumbs vom bestehenden Scenarioo übernommen und für die neue Struktur angepasst. Somit kann auch nach dem Wechsel in ein Scenario noch in das aktuell gewählte Feature und seine Parent-Features durch die Breadcrumbs erreicht werden.

Navigieren im der hierarchischen Struktur

Ein weiteres Feature, welches für diese Erweiterung geplant ist, soll eine Möglichkeit bieten, dem Benutzer auf schnelle Weise zu ermöglichen sämtliche Features in der hierarchischen Struktur der Features zu erreichen. Hierfür soll möglichst jedes Feature in der hierarchischen Struktur in jeder der neuen Ansichten erreichbar sein.

5.4.3.2 Documentation-View

Auf dieser Ansicht sollen alle Dokumentationsartefakte, also Markdown Dokumentation und Specification File des aktuell gewählten Features und aller Subfeatures des aktuell gewählten Features angezeigt werden. Für diese Darstellung wurden im Verlaufe des Projektes verschiedene Versionen diskutiert. Diese sind hier kurz aufgezeigt.

Nachfolgend die Entstehung der Docs-View:

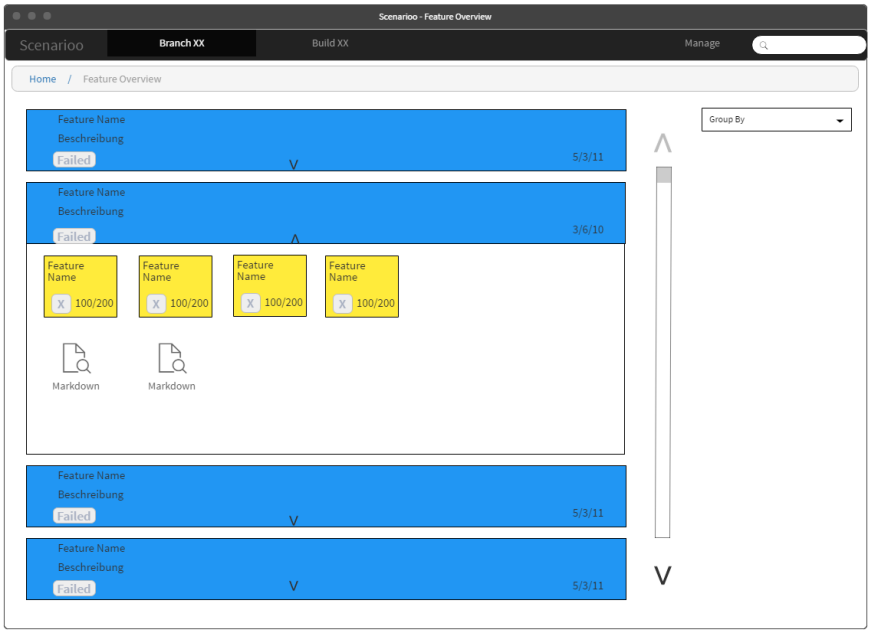


Abbildung 19: Erster Entwurf als Accordion

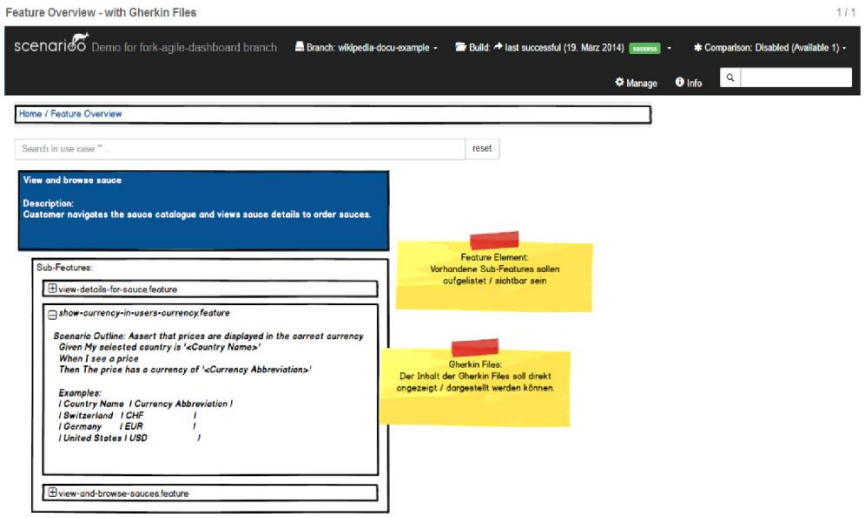


Abbildung 20: Zweiter Entwurf mit Dropdown

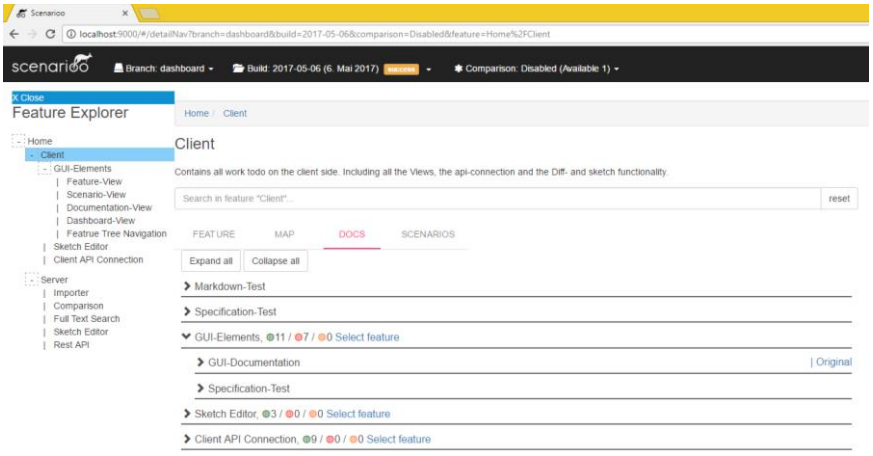


Abbildung 21: Resultat schlicht mit Dropdown

5.4.3.3 Map-View

In der Map-View sollen alle Subfeatures und alle Subfeatures der Subfeatures in einer Art Story-Map dargestellt werden. Im «Backbone» der Map werden alle Subfeatures des gewählten Features angezeigt. Diese sollen sortiert werden können. Darunter werden alle Subfeatures der Subfeatures in einer vertikalen Liste dargestellt, gruppiert nach dem «milestone» Attribut.

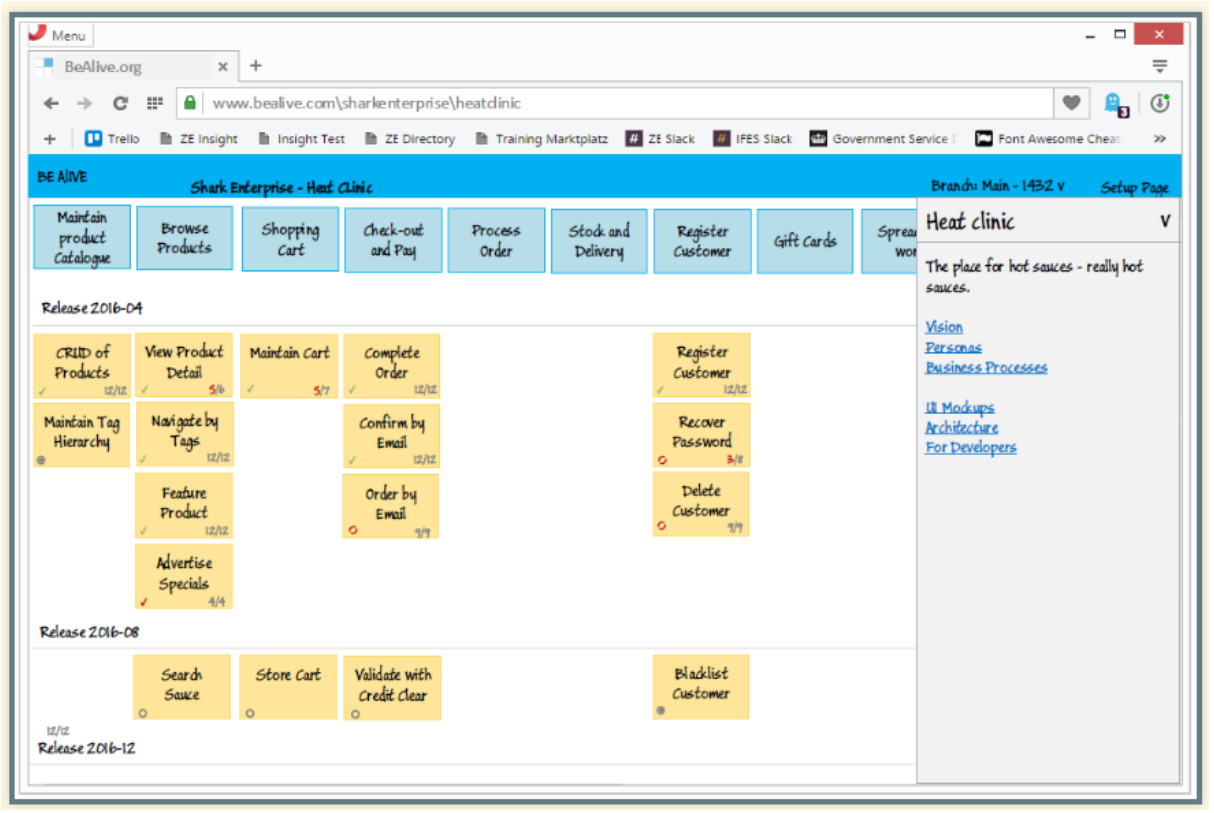


Abbildung 22: Idee für eine Map

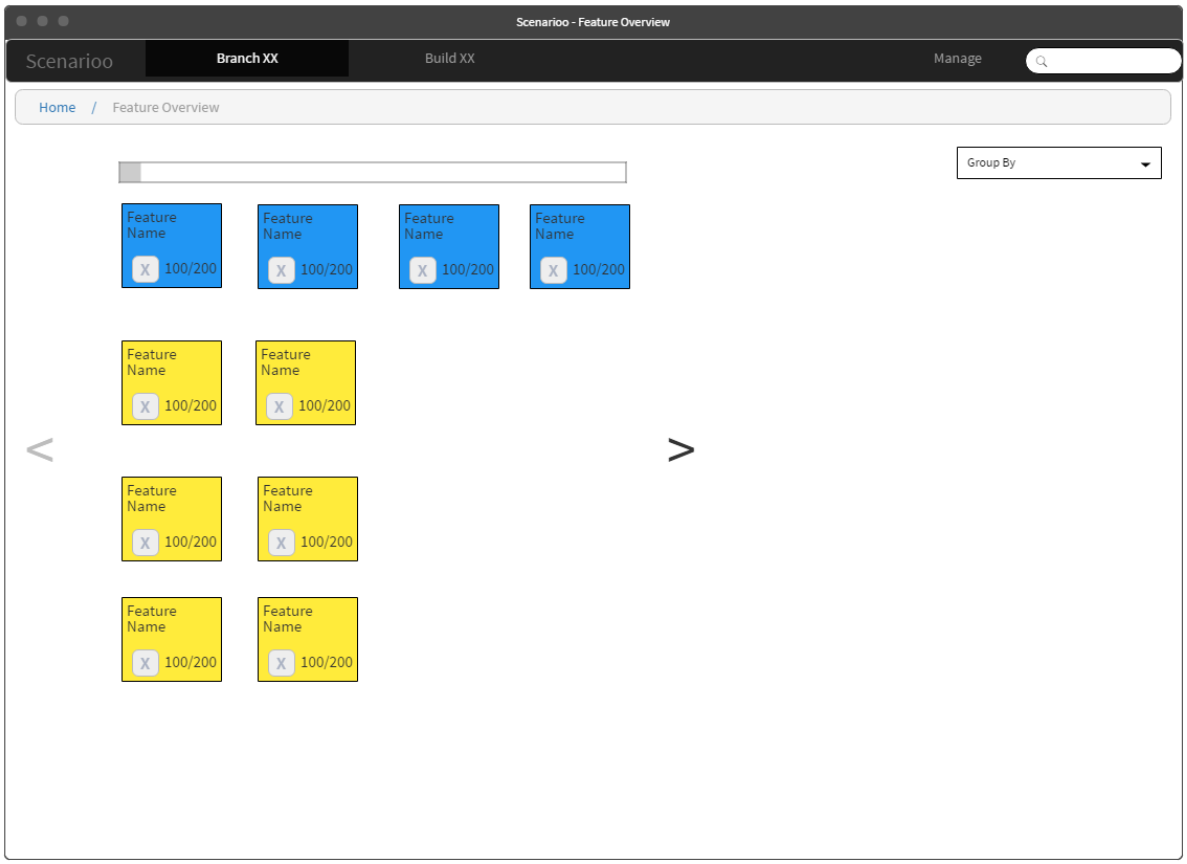


Abbildung 23: Map-Entwurf mit horizontalem Scroll

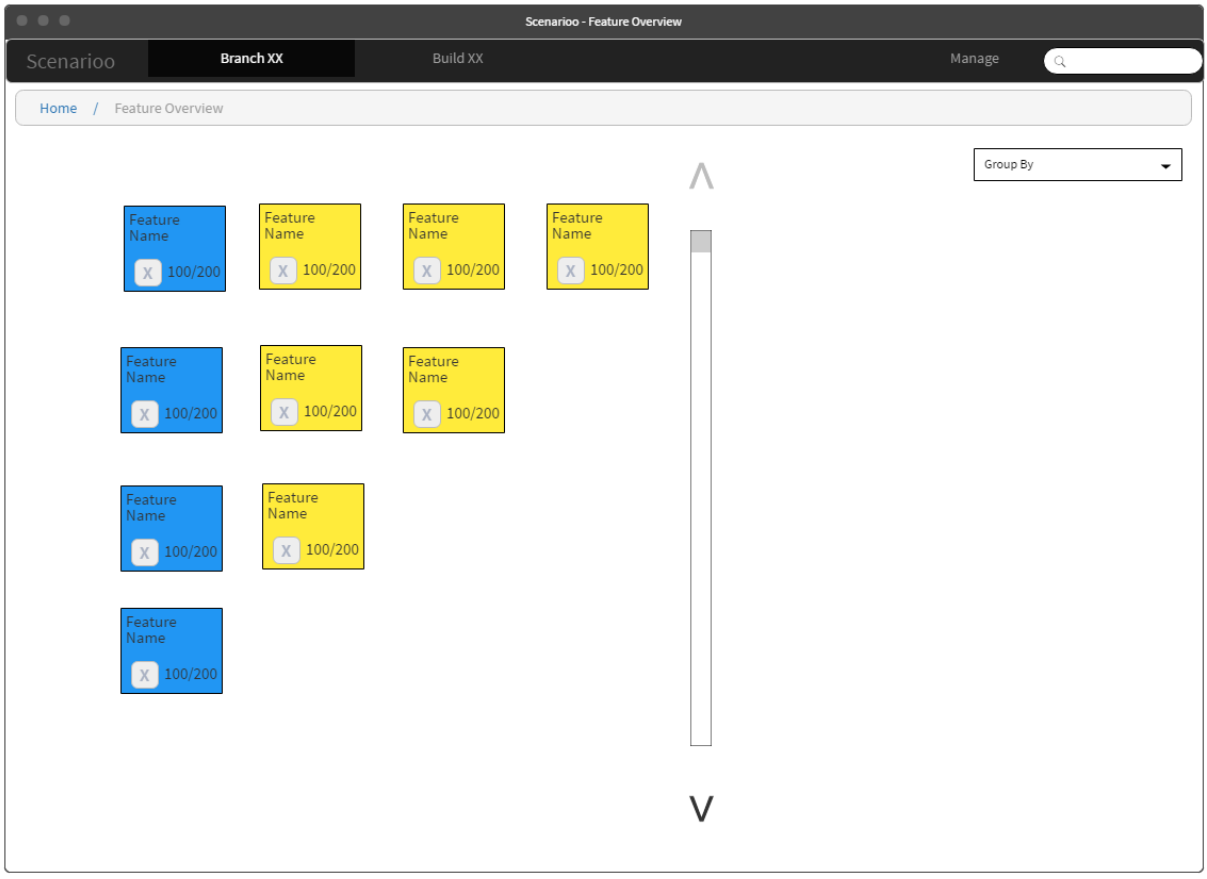


Abbildung 24: Map-Entwurf mit vertikalem Scroll

5.4.3.4 Features-View

In der Features-View sollen alle Subfeatures zum aktuellen Feature angezeigt werden, angelehnt an die alte «Use Cases-Tab-View».

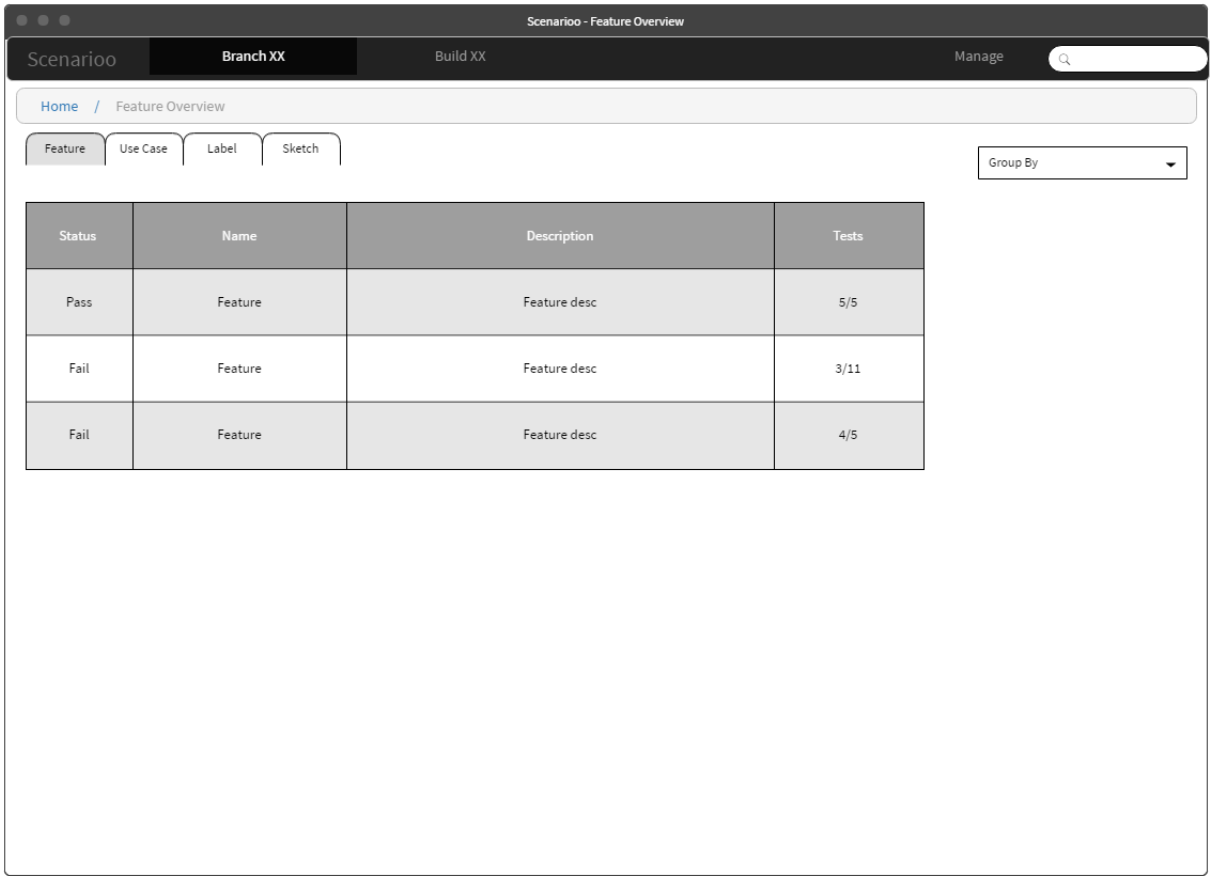


Abbildung 25: Features-View Entwurf

5.4.3.5 Scenarios-View

In der Scenarios-View sollen alle dem aktuellen Feature zugeordneten Scenarios in einer Tabelle angezeigt werden, angehlehnt an die alte «Use Case» View, welche alle Scenarios zu einem Use Case zeigte.

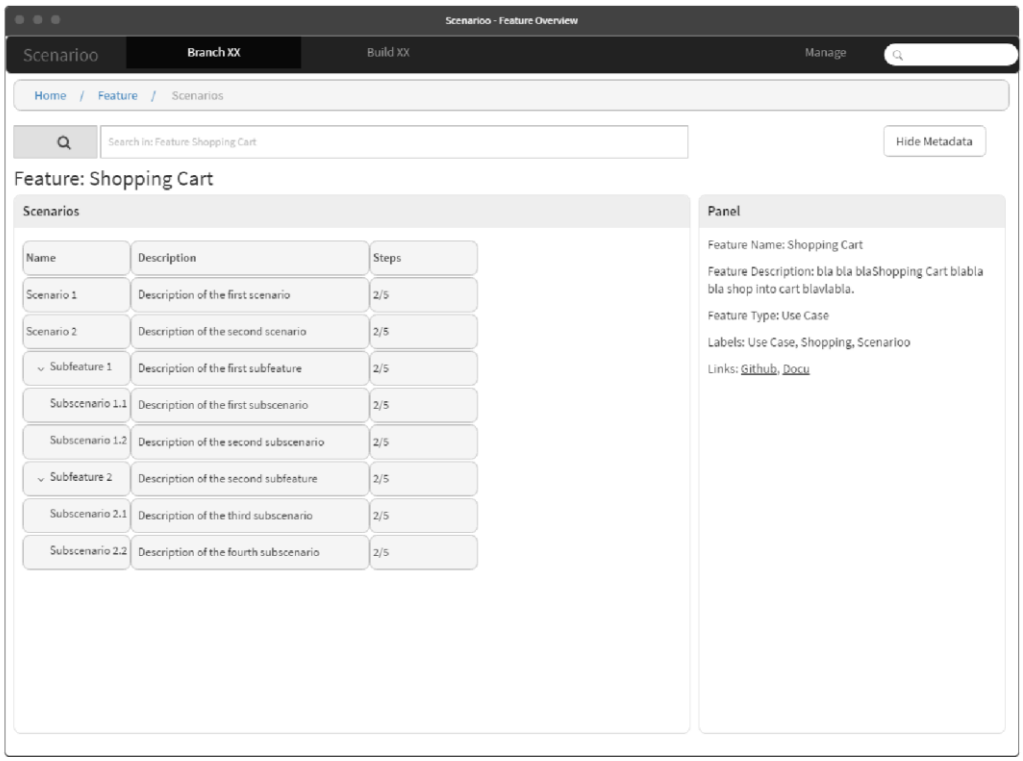


Abbildung 26: Scenarios-View Entwurf

5.4.4 Use Cases

5.4.4.1 Use Cases Diagramm

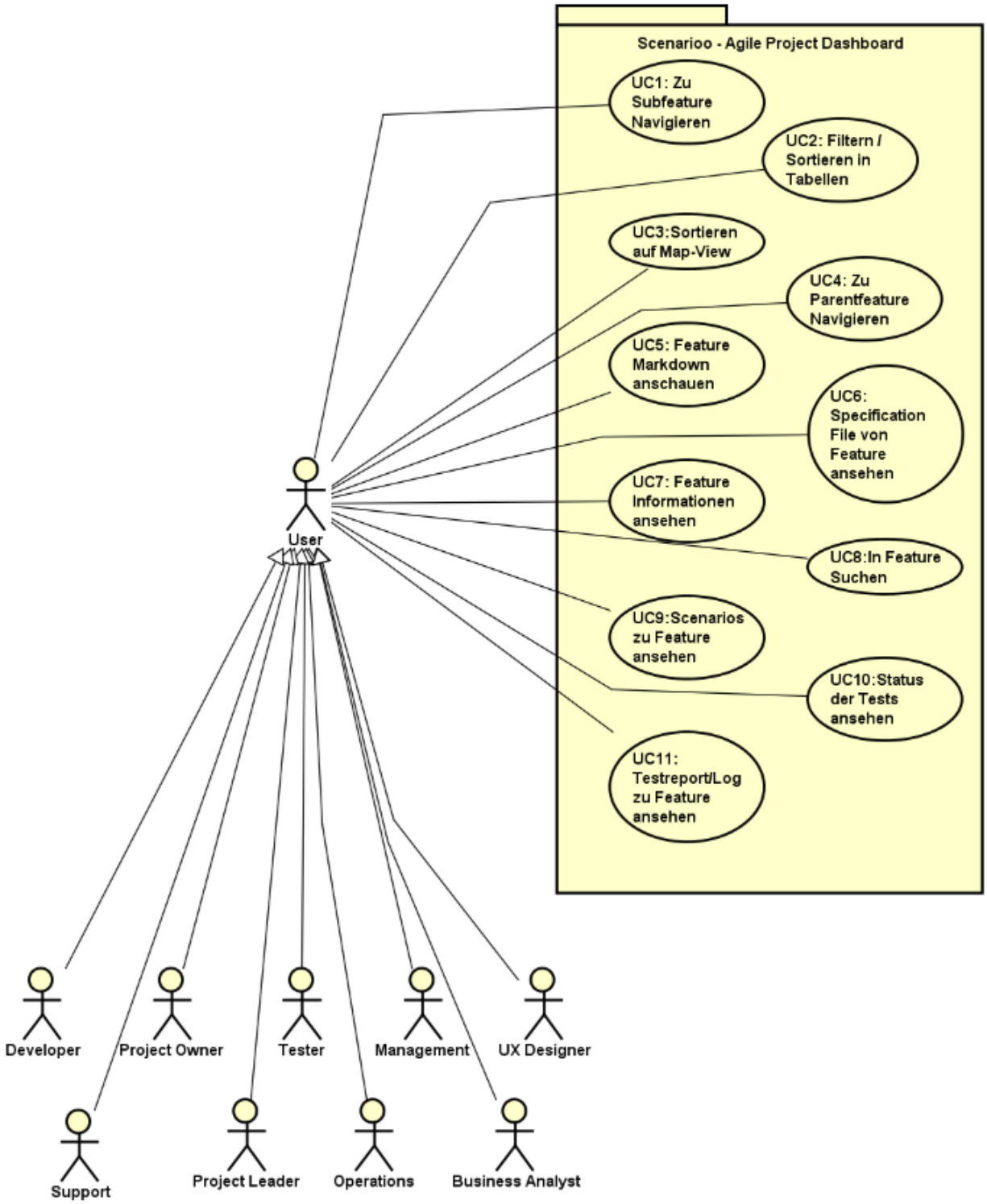


Abbildung 27: Use Case Diagramm

5.4.4.2 Use Cases Beschreibung

UC1: Zu Subfeature navigieren

Dem Nutzer soll es möglich sein ein Subfeature zum aktuell gewählten Feature auszuwählen. Dies wird dadurch zum nächsten aktuellen Feature.

Views: Auf folgenden Views soll diese Funktionalität umgesetzt werden.

- Map-View (inklusive Subsubfeatures)
- Features-View
- Specification-View
- Detail-View für aktuelles Feature
- Feature Explorer für den gesamten Feature-Tree

UC2: Filtern / Sortieren in Tabellen

Dem Nutzer soll es möglich sein, in einer Tabelle nach den gegebenen Spalten zu sortieren. Zudem soll der Benutzer den Inhalt der Tabelle über ein Suchfeld einschränken können.

Views: Auf folgenden Views soll diese Funktionalität umgesetzt werden:

- Features-View (für die Subfeatures)
- Scenario-View (für die Scenarios des aktuellen Features)

UC3: Sortieren auf Map-View

Dem Nutzer soll es auf der Map-View möglich sein, sowohl für den Backbone (Subfeatures), als auch in den Swimlanes (Subsubfeatures) die Sortierung auswählen zu können.

Views:

- Map-View

UC4: Zu Parentfeature navigieren

Dem Nutzer soll es möglich sein zum aktuellen Feature das Parentfeature auszuwählen. Dadurch wird dieses zum aktuellen Feature.

Views:

- Breadcrumbs auf allen Views
- Feature Explorer für gesamten Feature-Tree
- Detail View zum aktuellen Feature

UC5: Feature Markdown anschauen

Es soll dem Nutzer möglich sein die auf dem aktuellen Feature und den Subfeatures hinterlegten Markdown Dokumentationen anzuzeigen.

View:

- Specification-View

UC6: Specification File von Feature ansehen

Es soll dem Nutzer möglich sein die auf dem aktuellen Feature und den Subfeatures hinterlegten Spezifikations-Files anzuzeigen.

View:

- Specification-View

UC7: Feature Informationen ansehen

Es soll dem Nutzer möglich sein die Informationen zum aktuellen Feature schnell zu sehen.

View:

- Detail View für aktuelles Feature auf der Seite.

UC8: In Feature Suchen

Es soll dem Nutzer möglich sein die angezeigten Features zu filtern, nach Vorhandensein von bestimmten Informationen. Dies wird über ein Suchfeld erreicht. (UC2 Beschreibt Teil davon)

Views:

- Map-View
- Features-View
- Scenario-View
- Specification-View

UC9: Scenarios zu Feature ansehen

Der Benutzer soll die Scenarios des aktuellen Features wählen können und zur Übersicht der Steps für dieses Scenario kommen.

UC10: Status der Tests ansehen

Dem Nutzer soll angezeigt werden wie viele Tests in einem Feature erfolgreich waren, wie viele Tests «gefailed» sind und alle übrigen (Ignorierte Tests, noch nicht implementierte Tests).

UC11: Testreport/Testlog zu Feature ansehen

Testreport und Log sollen angezeigt werden können. Hier soll dem Benutzer die Zeile angezeigt werden, auf welcher im Specification-File, ein allfälliger Fehler aufgetreten ist.

5.4.5 User Stories

5.4.5.1 Story 1

Als Projektleiter möchte ich auf der Map-View alle Toplevelfeatures und deren Subfeatures sehen, um so beim aktuell zu bearbeitende Toplevelfeature und dessen Subfeatures den Status der Entwicklung ablesen zu können. Dieser Status soll in Form von den passed, failed und ignored Tests angezeigt werden.

5.4.5.2 Story 2

Als Entwickler möchte ich auf der Feature-View mir schnell einen Überblick über alle Toplevelfeatures verschaffen können und diese filtern/durchsuchen können, um so schnell zu den gewünschten Features und deren Subfeatures zu kommen (Bsp. Gruppiert nach Release-Datum, Sortiert nach Storyorder, Sortiert nach meisten offenen Tests, um zu sehen, wo für den nächsten Release noch die meiste Arbeit zu erledigen ist).

5.4.5.3 Story 3

Als Supportmitarbeiter will ich von jedem Feature, zu welchem ein Szenario zugeordnet ist, jeden einzelnen Schritt dieses Szenarios durch den entsprechenden Anwenderfall als Printscreen sehen können, damit ich einem User bei Problemen sagen kann, bei welchem der Schritte der Fehler liegt.

5.4.5.4 Story 4

Als Tester will ich die Möglichkeit haben, bei einem Feature auf einzelne Test-Logs zugreifen zu können, um nachzuschauen, welche Tests erfolgreich waren und welche nicht.

5.4.5.5 Story 5

Als BA will ich die Möglichkeit haben, bei einem Feature auf die angehängten Dokumentationsartefakte (Gherkin, Markdown, Links, usw.) zugreifen zu können, um die genaueren Details zu dem Feature hier weiter nachschlagen zu können.

5.4.6 User Centered Scenarios

5.4.6.1 Persona Roman

Roman ist ein Softwareentwickler in der Testfirma AG, im Moment arbeitet er an einem Projekt zur Erstellung eines Saucen Shops. In dem Saucen Shop werden verschiedenste Saucen Mischungen, Merchandise und Gift Cards für den Shop angeboten. Der Saucen Shop ist ein typischer Webshop, welcher als simple Website realisiert werden soll.

Einige der Hauptfeatures des Saucenshops umfassen: «Maintain product catalogue», «Browse products», «Shopping Cart», «Checkout & Pay», «Process order», «Stock and Delivery», «Register Customer» und «Gift Cards».

Scenario 1

Roman hat am Vorabend ein Feature fertig programmiert, hatte aber keine Zeit mehr dieses zu testen. Da er wusste, dass über die Nacht automatisch alle Tests ausgeführt werden, dachte er sich, er könne ja auch am Morgen nachschauen wie gut er gearbeitet hat.

Am nächsten Morgen kommt er zur Arbeit und stellt fest, dass ein Test fehlgeschlagen hat. Er öffnet Scenarioo und sieht einen Baum aller Features in seinem Projekt. Im Suchfeld tippt er «Maintain» ein und drück Enter um eine begrenzte Auswahl an Features anzuzeigen, so findet er sein Feature «Maintain Cart» schnell und einfach.

Mit der Scenario-View sieht Roman der Teststatus des Features «Maintain Cart», dieser ist auf «Failed», da bei mindestens einem der Subfeatures ein Test fehlgeschlagen ist. Er kann durch die Liste der Subfeatures scrollen und sieht von jedem einzelnen den Teststatus. Er sieht, dass beim Subfeature «Add product from product overview» ein Test fehlgeschlagen ist und kann dort das Protokoll der einzelnen Testszenarien anschauen. Mit der Übersicht aus Testresultaten, Testdefinition und Funktionsbeschreibung kann er den Fehler schnell lokalisieren, offensichtlich war es etwas später geworden gestern und seine Konzentration liess nach. Mit frischer Energie kann er nun den Fehler korrigieren, die Tests manuell auslösen und das erfolgreiche Ergebnis seinem Vorgesetzten mitteilen.

Roman navigiert zurück zu seinem Parentfeature «Maintain Cart» indem er auf der rechten Seite die Treenavigation öffnet und dort wieder «Maintain» eintippt um dann sein Feature auszuwählen und wechselt in die Map-Sicht. Dort sieht er alle Subfeatures von «Maintan Cart» und die Subsubfeatures nach Release geordnet. Anhand des Status kann er erkennen, welches Subsubfeature als nächstes benötigt wird und noch nicht richtig oder noch gar nicht implementiert wurde. So weiss er woran er als nächstes weiterarbeiten muss.

Scenario 2

Roman hat für einen neuen Sprint den Auftrag bekommen, sich nun mehr um das Feature «Register Customer» zu kümmern. Da er noch nicht weiss, um welchen Umfang es sich bei dieser Aufgabe handelt, und wie er anfangen soll dieses Features zu programmieren, geht er zuerst auf die Scenarioo Map-View, um hier sich schnell einen Überblick über das Feature und alle darunter geordneten Subfeatures zu verschaffen. Hiernach klickt er das Feature «Register Customer» an. Dann wechselt er auf die Docs-View, um die Feature Beschreibung lesen zu können.

Hier sieht er direkt die Dokumente, welche für das Feature hinterlegt wurden. Geschachtelt darunter kann der auch gleich direkt auf die Dokumentation aller Subfeatures, welche zum Feature «Register Customer» gehören. Diese sind «Register Customer Implementation», «Recover Password», «Delete Customer» und «Blacklist Customer».

5.4.7 Nicht-funktionale Anforderungen

NFA Interoperability:

- Technologien sollen von Scenarioo übernommen werden, sodass keine zu grossen Änderungen entstehen.

NFA Transparency:

- Sind in einer Ansicht keine Daten vorhanden, so soll der Nutzer entsprechend auf deren Abwesenheit aufmerksam gemacht werden durch das anzeigen eines Hilfetextes.

NFA Usability:

- Die Bedienbarkeit soll auch nach der Weiterentwicklung mit einem validen Datensatz für das MVP optimiert werden. Hierzu soll ein Usability-Tests mit möglichem Zielpublikum getestet werden, die Resultate und Entscheidungen daraus sind in der Dokumentation festzuhalten.

NFA Stability:

- Das MVP soll auch nach dem Umbau der Datenstruktur stabil mit einem validen Datensatz laufen. Dies soll über manuelle Integrationstests sichergestellt werden.

NFA Documentation

- Zum MVP und dessen Funktionen soll eine vom Kunden abgenommene Benutzeranleitung (in DE und EN) existieren.

5.5 Entwicklung

5.5.1 Verwendete Technologien

Abbildung 28: Verwendete Technologien

Die Technologien von der alten Scenarioo-Version wurden komplett übernommen. Teilweise wurden weitere Libraries ergänzt, um eine einfacheren Darstellung gewisser Elemente zu ermöglichen.

5.5.1.1 GIT

GIT wurde als Versionsverwaltungssystem für den Implementationsteil dieser Bachelorarbeit verwendet. Auch das Scenarioo Entwicklungsteam verwendet GIT für die Implementation und hat die Ressourcen auf GitHub öffentlich zur Verfügung gestellt. Auf GitHub wurde für diese Arbeit ein Fork des Scenarioo Repositorys erstellt. In diesem Fork wurde dann die Implementation der mit dem Industriepartner abgesprochenen Erweiterungen umgesetzt.

5.5.1.2 Gradle

Gradle wurde als Main-Buildtool vom Scenarioo Projekt übernommen. Für den Umbau auf die Feature-Struktur wurde die Java Writer-Library von Scenarioo mit in das Hauptrepository integriert, um so auch die korrekte Generierung der Testdaten aus dem Gradle Submodule «scenarioo-docu-generation-example» zu gewährleisten. Hierfür wurden am Gradle Buildprozess die nötigen Änderungen vorgenommen. Ansonsten wurde daran nichts weiter geändert.

5.5.1.3 Gulp

Gulp wurde als Clientseitiges Buildtool vom Scenarioo Projekt übernommen. Der Gulp Buildprozess wurde um sämtliche für die Komponenten extra zu ladenden CSS Files ergänzt, sowie um einen neuen Task, welcher alle die Weiterentwicklung betreffenden E2E Tests ausführt.

5.5.1.4 Jenkins

Die Zühlke hat einen BuildJob auf ihrem Jenkins für den «develop» Branch auf unserem Fork erstellt.

5.5.1.5 JAXB

JAXB oder auch «Java Architecture for XML Binding» wurde für die Erweiterung von Scenarioo übernommen. Diese Library dient dem lesen und dem Schreiben der XML Dokumente, welche zum Speichern des Scenarioo Datenformates verwendet werden. Dabei Stützt sich JAXB auf Annotationen, um die Serialisierung der Java-Objekte samt all ihrer Referenzen zu realisieren. Somit können ganze Objektstrukturen in XML Dateien gespeichert werden, oder aus validen XML Dateien wieder ausgelesen und zu Objekten der richtigen Klasse konvertiert werden.

5.5.1.6 highlight.js

highlight.js ist eine JavaScript Library, welche das Code-Highlighting für mehrere Programmiersprachen erlaubt, durch automatische Spracherkennung. Diese Library wird zum Anzeigen der Spezifikations-Files beim Client in der Komponente Markdown verwendet.[4]

5.5.1.7 Showdown

Showdown ist eine JavaScript Library, welche Markdown Files zu HTML konvertieren kann. Diese wurde zum Anzeigen von Markdown Files beim Client verwendet.[5]

5.5.1.8 AngularJS Material

Material Design Library zur Darstellung einiger Komponenten im Frontend.[6]

5.5.1.9 Weitere Technologien

Technologie	Wo wurde sie eingesetzt
Angular JS	Übernommen von Scenarioo. JavaScript Framework für den Client.
Bower	Package-Manager für den Client, übernommen von Scenarioo
Elasticsearch	Volltextsuche zum Indexieren der Scenarioo Details
GraphicsMagick	Zum Vergleichen der UI-Screenshots. Übernommen von Scenarioo.
Intellij-Idea	Als Entwicklungs-DIE für das Projekt.
Java	Programmiersprache für die Serverseite. Übernommen von Scenarioo.
JavaScript	Programmiersprache für die Clientseite. Übernommen von Scenarioo.
nodeJS	Für E2E Test und Gulp. Übernommen von Scenarioo
Phantom JS	Für die automatisierten E2E Tests. Übernommen von Scenarioo.
Protractor	Für die E2E Tests im Frontend. Übernommen von Scenarioo.

[4] https://highlightjs.org/ Stand 15.06.2017
[5] https://github.com/showdownjs/showdown Stand 15.06.2017
[6] https://material.angularjs.org/latest/ Stand 15.06.2017

Tomcat	Java Server Software auf der das Backend läuft. Übernommen von Scenarioo.

Tabelle 18: Verwendete Technologien

5.5.2 Implementation

5.5.2.1 Server

Auf der Serverseite wurde die Datenstruktur um die neuen Elemente Feature, DocFile, Link und FileType ergänzt. Zusätzlich wurde ein Algorithmus geschrieben, der aus der flachen Feature-Struktur die gewünschte hierarchische Baumstruktur erstellt, bevor die Daten an den Client gesendet werden.

Hierzu werden zuerst alle Features eines Builds in eine Liste geladen und ein leeres Set erstellt, um diese Features auszusortieren, welche kein eigenes Parent-Feature haben. Für jedes Feature, genannt «currentFeature» in dieser Liste wird zuerst folgender Prozess durchlaufen:

Für jede Feature-ID in der String Liste «featureNames» wird das Feature mit der entsprechenden ID in der Liste aller Features für diesen Build gesucht. Dieses Feature wird nun als Subfeature dem «currentFeature» hinzugefügt und in das Set gelegt, da dieses Feature das «currentFeature» als eigenes Parent-Feature besitzt.

Nach diesem Prozess sind nun alle Features in der Liste mit Referenzen zu ihren Subfeatures ausgestattet. Alle Features im Set haben ein eigenes Parent-Feature. Somit müssen nun aus der Liste alle Features gelöscht werden, welche im Set vorhanden sind. Dies geht glücklicherweise in Java mit einem Ein-Zeiler:

```
features.removeAll(featureSet);
```

5.5.2.2 Client

Auf der Clientseite wurde die Struktur ebenfalls um die oben genannten Objekte ergänzt. Nachfolgend werden die von uns entwickelten Ansichten, Komponenten, Kontroller und Services erklärt.

Ansichten

Durch die Erweiterungen am Domainmodell kamen neue Daten dazu, welche dem Nutzer präsentiert werden müssen. Dazu wurden vier Ansichten designed, auch in enger Absprache mit Zühlke. Zwei Ansichten wurden dann aus dem bestehenden Scenarioo übernommen und angepasst, zwei wurden neu entwickelt.

Allgemein hat jede Seite den vom bestehenden Scenarioo bekannten Header, indem Branch, Build und Comparison gewählt werden können. Darunter befindet sich die Breadcrumbs-Leiste, auf welche im Kapitel Komponenten weiter eingegangen wird. Auf der linken Seite ist der Feature Explorer, weitere Informationen dazu sind ebenfalls im Kapitel Komponenten enthalten. Jede der vier Seiten zeigt zuoberst den Namen des aktuell gewählten Features und darunter dessen Beschreibung was beides aus dem jeweiligen feature.xml gelesen wird. Die Suchleiste, die dann zu sehen ist wurde komplett übernommen und durchsucht alle Features, die unter dem aktuellen Feature liegen. Danach kommt die Ansichten-Auswahl deren vier Möglichkeiten jetzt ausführlich beschrieben werden.

Feature-View

Die Feature-View besteht aus einer Tabelle, die von der Funktionalität her vom bestehenden Scenarioo übernommen wurde, so auch die Sortierbarkeit. Neu wurden aber die Felder ‘Milestone’, ‘#Subfeatures’ und ‘#Tests’ hinzugefügt. Die ‘Name’-Spalte verfügt weiter hin über die Fähigkeit, Labels nach dem Namen anzuzeigen. Status, Beschreibung und Milestone werden direkt aus dem Feature gelesen, bei der Anzahl der Subfeatures wird zuerst überprüft, ob die Grösse des Arrays grösser als 1 ist, um dann diese auszugeben oder eine 0 um Konflikte mit leeren Arrays zu verhindern. Die Anzahl der Sceanrios kann als Attribut aus dem Feature gelesen werden. Zur Anzeige der Testübersicht wurde eine eigene Komponente entwickelt, auch diese wird im entsprechenden Kapitel erklärt.

Es gibt auf dieser Ansicht eine Abfrage, ob überhabt Subfeatures im Feature enthalten sind. Ist dies nicht der Fall, wird keine Tabelle angezeigt, dafür eine Meldung, die auf das Problem hinweist und ein Vorschlag, dass man mit der Docs-View die Dokumente des gewählten Features sehen kann oder mit der Sceanrios-View die Scenarios.

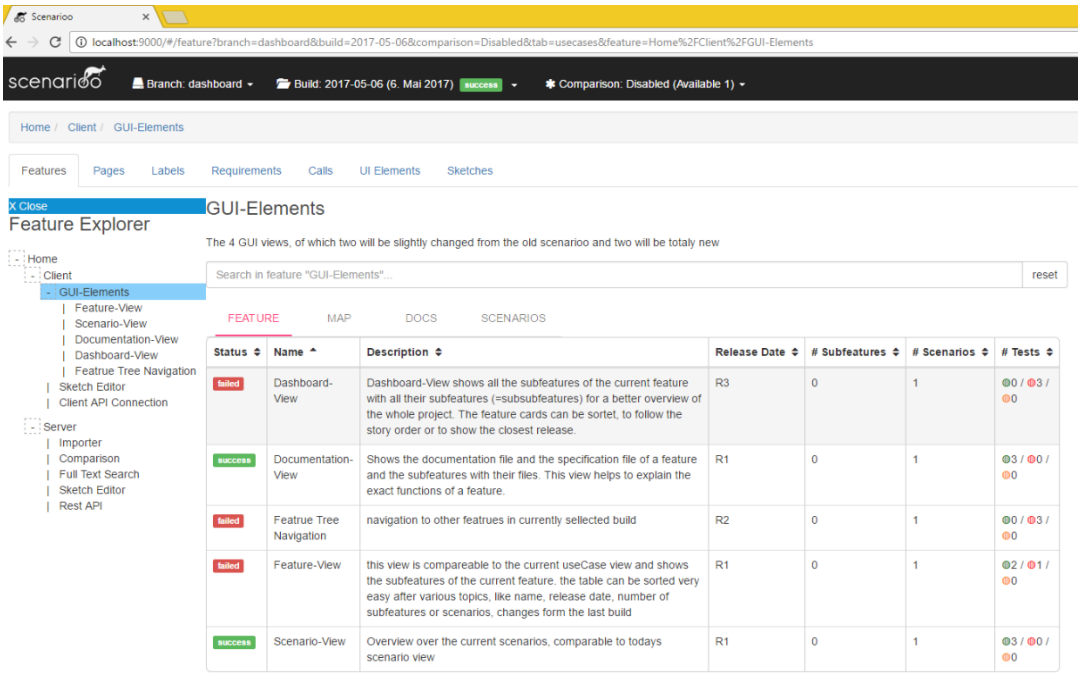


Abbildung 29: Umgesetzte Feature-View

Scenarios-View

Die Scenario-View zeigt ebenfalls eine einfache Liste, die vom bestehenden Sceanrioo übernommen wurde. Alle Attribute können direkt aus dem Feature Objekt gelesen werden. Beim Übernehmen wurde auf die Spalte Actions verzichtet, da im ersten Moment der Nutzen nicht klar war. Später beim Usability Test wurde sie nicht vermisst und es gab keinen Grund diese einzufügen. Bisher beinhaltete diese Spalte zwei Icons, eines hat dieselbe Funktion wie ein Klick auf den Rest der Zeile und öffnet die Steps des Scenarios. Das zweite Icon geht einen Schritt weiter und öffnet auch noch die erste Page im Step.

Auch hier wird überprüft ob Scenarios im Feature vorhanden sind. Ist dies nicht der Fall, wie zum Beispiel im Home Feature, wird ein Hinweis gegeben und zwei Vorschläge, welche Ansichten Daten enthalten könnten.

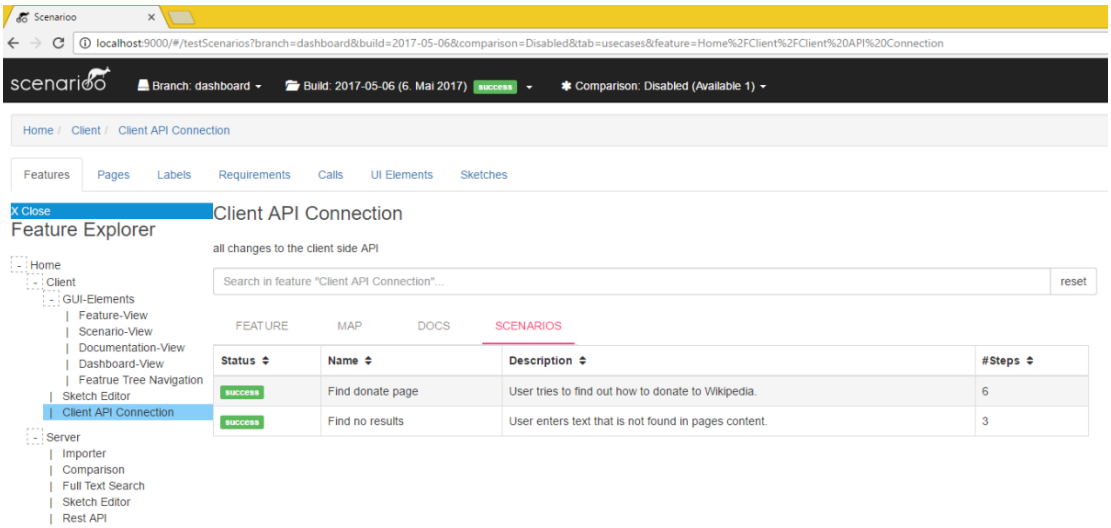


Abbildung 30: Umgesetzte Scenarios-View

Map-View

Diese Ansicht war ursprünglich eine Idee vom UX Spezialisten der Zühlke und sollte an ein Storyboard erinnern. Im Laufe der Entwicklung diese aber mehrfach umbenannt. Am Ende lag der Name bei Map-View und verglichen mit den Anderen zeigt die Map auch Subsubfeatures. So kann mehr Übersicht geschaffen werden, da die einzelnen Features wichtige Informationen über ihre Teststatus anzeigen.

Da hier nicht eine einfache Liste angezeigt wird musste eine Möglichkeit zur Sortierung geboten werden. Gleich unter der Ansichts-Auswahl kann man nun die Reihenfolge der Subfeatures (blau) mit der ‘First order’ festlegen und die der Subsubfeatures (gelb) mit der ‘Second order’. Zur Auswahl stehen je drei verschiedene Kriterien: Milestone, Order Index oder Name. Der Order Index ist eine beliebige Reihenfolge, die vom Nutzer vor dem Import definiert werden kann/muss. Die Auswahl ist als Dropdown erreichbar.

Zuerst werden die Subfeatures in einer Zeile dargestellt mit einer FeatureCard. Danach wird in allen Sub- und Subsubfeatures geprüft, welche Meilensteine vorhanden sind um dann für jeden dieser ein eigenes Feld zu kreieren. Somit entsteht eine Tabelle in welcher die Subfeatures die Spalten vorgeben und die Meilensteine die Zeilen. Dann werden alle Subsubfeatures aller Subfeatures kontrolliert und wenn Subfeature und Meilenstein stimmen, wird im entsprechenden Feld eine FeatureCard dafür angelegt. Mit einem Klick darauf kann man dieses Feature öffnen.

Da diese Ansicht wenig hilfreich ist, wenn keine Subsubfeatures vorhanden sind, gibt es auch hier eine Meldung und Links auf besser passende Ansichten.

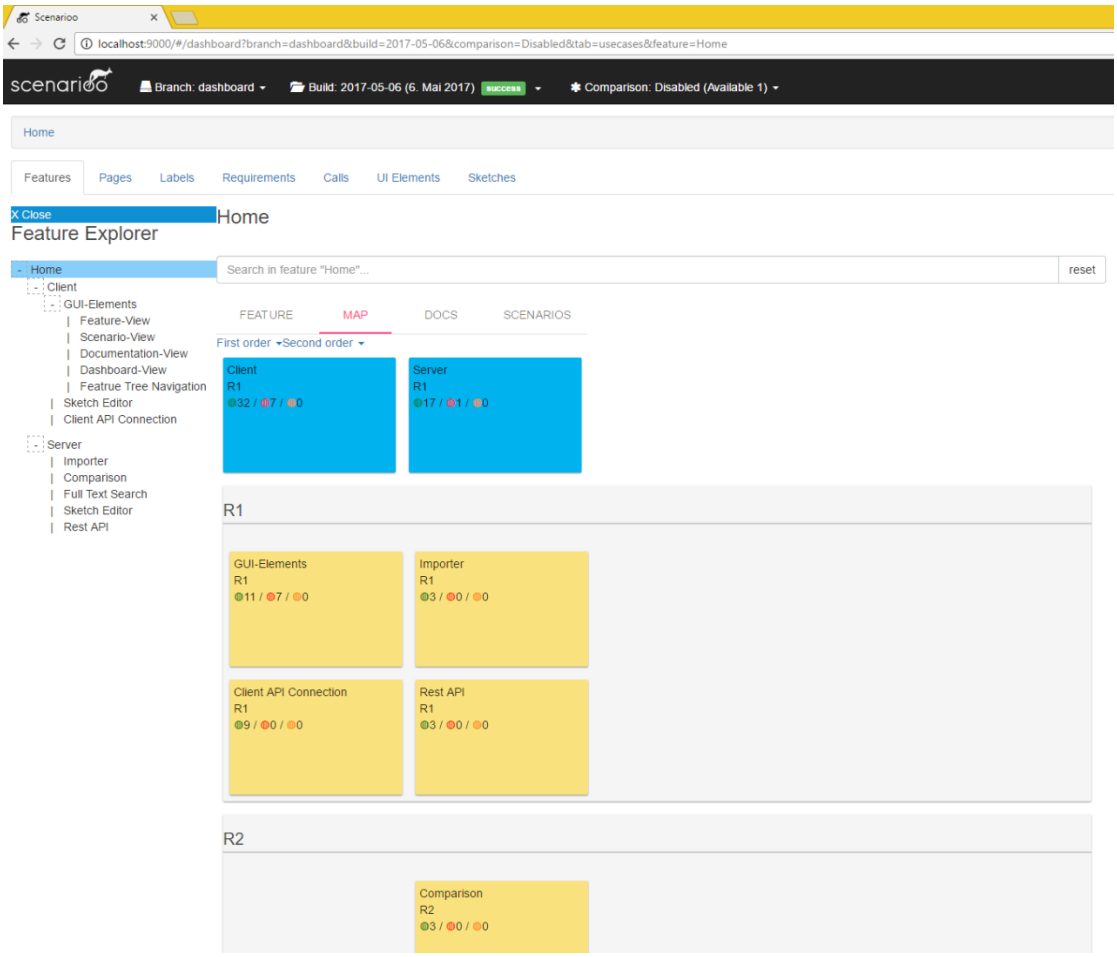


Abbildung 31: Umgesetzte Map-View

Docs-View

Diese Ansicht stellt die Dokumente dar, welche neu einem Feature angefügt werden können. Sie startete als eine Art Accordion, soll heute aber an die mobile Version von Wikipedia erinnern, bei der Kapitel ein und ausgeklappt werden können.

Zuoberst finden sich zwei Knöpfe, mit einem werden alle Artikel und Subfeatures ausgeklappt, mit dem Anderen wird alles eingeklappt. Dazu hat jedes Dokument und Feature einen Status, ob es offen oder geschlossen ist. Danach folgen zuerst die Markdown Dokumentation, dann die Spezifikation, welche Testdefinitionen enthält. Jedes File hat eine Überschrift, mit der das File ausgeklappt werden kann. Dort öffnet sich dann eine FileViewer Komponente.

Darunter werden die Subfeatures aufgelistet. Ihre Überschriften beinhalten neben dem Namen auch die Testübersicht und ein ‘Select feature’ Knopf, mit dem das Feature geöffnet werden kann. Die Testübersicht soll hier helfen, Fehler schneller zu lokalisieren. Die aufklappbaren Überschriften der Features zeigen ebenfalls deren Markdown und Spezifikation in einer FileViewer Komponente.

Auch hier wird darauf hingewiesen, falls keine Subfeatures oder Dokumente vorhanden sind.

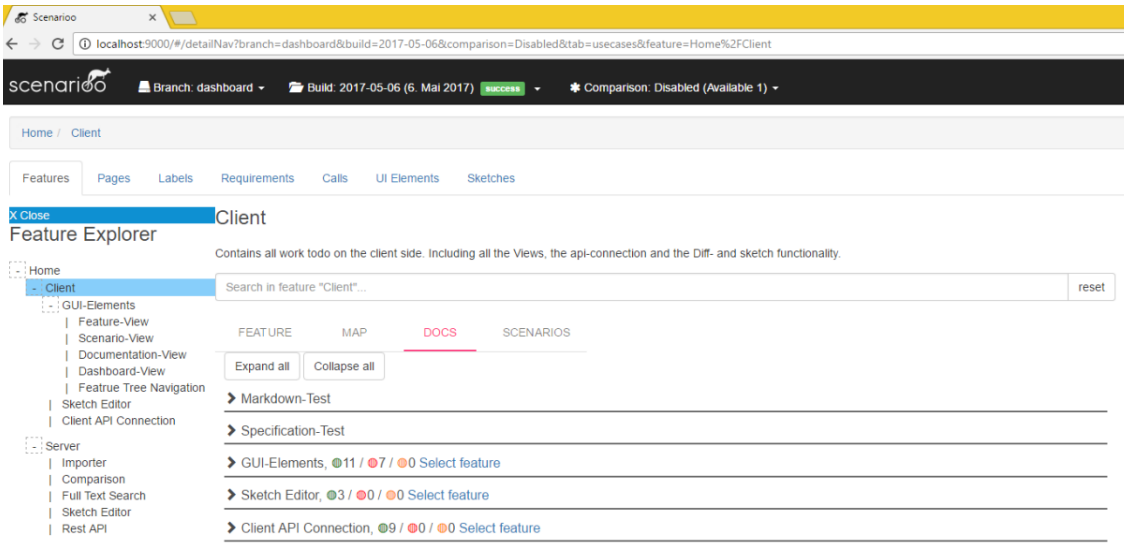


Abbildung 32: Umgesetzte Docs-View

Meta Data

Im bestehenden Scenarioo kann auf der rechten Seite eine Leiste mit zusätzlichen Informationen zum Feature eingeblendet werden. Diese Möglichkeit ist bei den Features auch vorhanden durch die Feature Detail Komponente.

Abbildung 33: Neue Meta Data Ansicht

Komponenten

Hier werden die wiederverwendbaren Komponenten erklärt.

Breadcrumbs

Die Breadcrumbs-Komponente wurde speziell für Scenarioo entwickelt und an dessen Struktur angepasst. Die bestehende Lösung war deshalb nicht sehr generisch, darum konnte sie nicht einfach erweitert werden. Mit der Änderung von Use Case auf Feature wurde aus einem Breadcrumb-Element plötzlich X-beliebig viele (maximale Tiefe des Feature-Baumes). Darum wurden das Home-Element und das Use Case-Element aus der bestehenden Breadcrumbs entfernt und durch eine Liste des Featurepfades ersetzt.

Da das erste Feature sowieso Home heisst, ist der Home-Link immer noch verfügbar und danach kommt eine Liste aller Features, die aus dem Feature Service geladen wird. Die Liste ist ein Array mit den Features von Home bis zum zuletzt gewählten Feature. Die Elemente der Liste werden direkt als Link in die Breadcrumbs-Leiste eingefügt.

Danach kommen wieder die bestehenden Breadcrumbs-Elemente von Scenario und Step. Diese zeigen immer ‘Typ: Name’, dies wäre bei den Features überflüssig, da man dann für jedes Feature ‘Feature: Name’ hätte.

Feature Card

Die Komponente Feature Card ist eine Material Design Karte welche Name, Milestone und die Testübersicht eines Features zeigt. Sie wird hauptsächlich auf der Map-View verwendet. Als mitgelieferte Attribute bekommt die Komponente das Feature, welches sie darstellen soll und die Hintergrundfarbe. Da auf der Map-View zwei verschiedene Farben von Karten angezeigt werden, wurde das so gelöst,

weil auf der Ansicht klar ist welche Karte für welches Sub-Level steht. Dies hätte auch später noch evaluiert werden können, doch auf diese Weise ist es in diesem Rahmen einfacher.

Beim Auslagern der Style-Attribute ist dann ein Problem aufgetaucht, da vom CSS nicht auf das übergebene Farbattribut zugegriffen werden konnte. Darum konnten nur die Grössenattribute ausgelagert werden und die Farbdefinition liegt im HTML.

Feature Detail

Die Feature Detail Komponente bekommt beim Aufruf ein Feature, dessen Details angezeigt werden sollen und nutzt die clickFeature-Funktion des Feature Services welche später erläutert wird.

In den Details werden folgende Attribute des Features angezeigt: Name, Milestone, Status, Beschreibung, eine Liste der Subfeatures und das Parentfeature. Sub- und Parentfeature sind Links und zeigen das entsprechende Feature an.

Auch hier wurden Grössen- und Farbattribute ausgelagert und können somit an einem zentralen Ort verwaltet werden.

Feature Explorer

Der Feature Explorer ist auf allen vier Ansichten auf der linken Seite zu sehen. Er ist standardmässig geöffnet, kann aber mit dem blauen ‘X close’ Feld geschlossen werden. Die Komponente enthält den Titel für den offenen und geschlossenen Zustand und eine NavTree-Komponente mit dem Baum über alle Features.

Übergeben wird das Home-Feature und das aktuell gewählte. Damit wird der Baum vom Anfang her aufgebaut und das aktuelle markiert. Zum Öffnen und Schliessen wurden die Methoden ‘openNav’ und ‘closeNav’ definiert. Die Eventpropagation wird unterbrochen, damit nicht jeder Klick auf ein Feature den Feature Explorer schliesst. Dies wurde anfänglich so implementiert, aber nach dem Usability Test geändert.

File Viewer

Der File Viewer wurde entwickelt um die Anzeige der Markdown- und Spezifikationsfiles auszulagern. Übergeben wird nur das Feature, zusätzlich holt sich die Komponente noch die Funktion ‘contains’ vom Feature Service.

Nun wird für beide Files dasselbe ausgeführt, nämlich zuerst eine Abfrage mit dem Feature und ‘contains’ um sicherzustellen, dass das Feature das entsprechende File enthält. Falls kein File vorhanden ist, wird auch nichts angezeigt. Falls schon, wird der Titelbalken angezeigt, mit dem Namen des Files und einem Link zur Originaldatei. Wird dieser Titelbaken in der Ansicht angeklickt, öffnet sich das eigentliche File darunter.

Markdown

Die Markdown-Komponente wird genutzt um im File Viewer das Markdown anzuzeigen. Dazu wird der Komponente ein URL zu einem File übergeben. Dieses File kann sich im Internet oder lokal auf dem Server befinden. Beim Öffnen wird der übergebene URL vorbereitet und dann geladen. Markdowns werden einfach als HTML dargestellt für die Codefiles der Spezifikation gibt es noch highlighting, damit sie besser lesbar werden.

Nav Tree

Der Navigationsbaum wird vom Feature Explorer aufgerufen und generiert rekursiv einen Baum über alle Features. Beim Aufruf werden zwei Features mitgegeben, einmal das Feature, das gerade aufgelistet werden soll und als Zweites, das aktuell gewählte Feature. Die Komponente übernimmt die Funktionen 'contains' und 'clickFeature' vom Feature Service. Zusätzlich wird die Funktion 'getBGColor' definiert, sie kontrolliert, ob das Feature welches aufgelistet werden soll gleich dem aktuell gewählten Feature ist, um ihm dann einen anderen Hintergrund zu geben.

Bei der Auflistung der Features wird jedem ein Zeichen vorangestellt. Dies ist ein senkrechter Strich, wenn keine weiteren Subfeatures vorhanden sind oder ein +/- zum ein- und ausklappen der weiteren Subfeatures. Ist letzteres der Fall, wird darunter rekursiv ein neuer Navigationsbaum geladen.

Wie schon bei der Feature Card war es auch hier nicht möglich die Hintergrundfarbe auszulagern, da sie vom Feature abhängig ist. Die restlichen Style-Attribute wurden aber ausgelagert.

Sidepanel

Das Sidepanel ist ein Container für die Feature Details. Es nimmt den auf der Ansicht gegeben Header und die Feature Details und fügt sie zusammen in einem Feld ein. Falls eine dieser beiden Informationen nicht gegeben ist, wird dafür ein Standardtext angezeigt.

Site Navigation

Die Site Navigation ist die Tab-Auswahl der Ansichten. Sie erhält beim Aufruf die aktuell gewählte Ansicht um diese zu markieren, damit der Benutzer weiss, wo er sich befindet.

Ein kleiner Service definiert die Liste der verfügbaren Ansichten, mit einem Namen, der mit dem Namen der Ansicht (Softwareintern) übereinstimmen muss, ein Kürzel, welches auf der Ansicht angezeigt wird und ein URL, der angibt, wo das HTML zu finden ist. Wird eine Ansicht gewählt, muss der Name im URL der Scenarioo-Seite angepasst werden und er muss in den Local Storage gelegt werden, für Funktionen welche im Feature Service beschrieben werden.

Test Overview

Die Testoverview wird an vielen Stellen eingesetzt und gibt einen schnellen Überblick über den Teststatus eines Features. Mitgegeben wird der Testoverview nur ein Feature. Von diesem liest es die drei Zahlen 'success', 'failed' und 'ignored' und stellt diese mit einem Farbigen Kreis dar. Die Farbe des Kreises soll die drei Namen veranschaulichen. Daneben wird dann die Zahl dargestellt, über dessen Herleitung mehr im Beschrieb des Feature Service zu finden ist.

Kontroller

Dashboardcontroller

Der Dashboardcontroller ist der Kontroller für alle vier Ansichten und implementiert diverse Methoden, damit die Darstellung in den HTML Files erleichtert wird.

Der DashboardController hat drei Scopelisteners, für das aktuell gewählte Feature, das Rootfeature und die Milestones. Diese drei müssen im Hintergrund immer aktuell gehalten werden, damit die Ansichten die richtigen Ausgaben machen.

Der Kontroller übernimmt diverse Funktionen, diese sind in der folgenden Tabelle ersichtlich:

Name	Herkunft	Funktion
clickFeature	FeatureService.js	Öffnet das Feature, welches angeklickt wurde.
equals	FeatureService.js	Prüft ob ein Objekt gleich einem anderen ist.
contains	FeatureService.js	Prüft ob ein Objekt ein bestimmtes Feld enthält.

Tabelle 19: Vom Kontroller übernommene Funktionen

Weiter werden folgende Methoden im Kontroller definiert:

Name	Beschreibung
setOrder	Sortiert die Features nach einer bestimmten Reihenfolge
isDefined	Kontrolliert, ob ein Objekt definiert ist
expandAll	Ruft für ein Feature setCollapseState mit ‘false’ auf
collapseAll	Ruft für ein Feature setCollapseState mit ‘true’ auf
setCollapse-State	Bekommt ein Feature und ein Boolean und setzt für das Feature den Collapsed-Zustand. Wiederholt dies auch rekursiv für alle darunterliegenden Features
activate	Lädt das Rootfeature, das aktuell gewählte Feature und die Label-Konfiguratio-nen
clickScenario	Bekommt die Zusammenfassung eines Scenarios und muss den Namen extrahie-ren um dann goToScenario zu benutzen
goToScenario	Navigiert aus einem Feature auf ein Scenario mit deren Namen
containsSubsub	Überprüft, ob ein gegebenes Feature Subsubfeatures enthält
getLabelStyle	Setzt die richtige Farbe für ein Label

Tabelle 20: Methoden im Kontroller

Service

Feature Service

Der Feature Service ist für das Laden und Vorbereiten der Daten verantwortlich. Er definiert Funktio-nen, welche von diversen Komponenten und Controllern verwendet werden.

Der Service hat zwei Scopelisteners, einmal für die Suche und einmal für die Comparison. Die Suche soll automatisch die Liste aktualisieren und die Comparison-Felder bei den Tabellen müssen schnell angezeigt werden, wenn diese aktiviert wurde.

Folgende Funktionen wurden definiert:

Name	Beschreibung
loadBackRefs	Setzt bei allen Features das Parentfeature

getSelectedFeatureNames	Gibt einen Array mit allen Features von Home bis zum aktuellen Feature zurück
selectFromArray	Macht aus einem Feature-Array und der Ansicht einen URL für die neu geladene Seite
loadFeatures	Lädt die Features mit einem gegebenen Branch und Build
contains	Erhält ein Feature und ein Feld und prüft, ob das Feature auf diesem Feld Daten hat
equals	Prüft ob zwei Objekte gleich sind
clickFeature	Erhält ein Feature und eine Ansicht dann wird das Feature gesetzt und die Ansicht geladen
getFeaturesByArray	Iteriert rekursiv anhand des Pfades vom Homefeature zum gewählten Feature
loadFeature	Erstellt den Pfad vom Homefeature zum gewählten Feature und sucht mit getFeaturesByArray das gewählte Feature
getFeatureString	Setzt rekursiv den Pfad von einem Feature zum Homefeature zusammen
getCurrentFeatures	Holt das gewählte Feature aus dem Local Storage
setFeature	Setzt ein neues Feature als gewähltes Feature.
getFeature	Gibt das gewählte Feature zurück
getRootFeature	Gibt das Rootfeature zurück
getMilestones	Gibt ein Array mit den Milestones zurück
setInternalAfterLoad	Erstellt das oberste Feature als Home und fügt die Projektfeatures unten an. Lädt für alle Features die Parentfeatures und Milestones und errechnet den Status vom Homefeature
calculateStatus	Ruft getStatus für das Rootfeature auf
getFeatureStatus	Evaluiert rekursiv den Status eines Features mit Hilfe der Subfeatures
GetArray	Überprüft, ob der gegebene Parameter ein Array ist, falls dies der Fall ist so wird er zurückzugeben, falls nicht wird ein leeres Array erzeugt.
getStatus	Zählt die verschiedenen Werte des «status» Attributs der Subfeatures und der Scenarios (mit Hilfe von getScenarioStatusFrom)

getScenarioStatusFrom	Zählt die verschiedenen Werte des «status» Attributs der Scenarios eines Features
isDefined	Prüft ob ein Objekt definiert wurde
loadScenariosDiffInfoInt	Ab einem gegebenen Feature werden rekursiv die DiffInfos geladen mit Hilfe von getFeatureDiffInfo
getFeatureDiffInfo	Lädt die DiffInfo eines Features und seiner Scenarien
reloadFromBranchBuild	Lädt alle Features eines Branch/Build mit Hilfe von loadFeatures
SelectedBranchAndBuildService. **callOnSelectionChange**	Ruft relaodFromBranchBuild für gegebene Branch/Build auf
unique	Filtert Array, sodass jedes Element nur einmal vorkommt
getAllMilestones	Fügt von jedem Feature die Milestones in ein Array, filtert es mit der Funktion «unique» und sortiert sie
getAllOfInArray	Sucht ab einem gegebenen Feature rekursiv alle Werte zu dem gegebenen Attribut und schreibt sie in einen Array.

Tabelle 21: Funktionen im Service

LocalStorageName Service

Im LocalStorageName Service werden die Konstanten für den Zugriff auf den localStorage komponentenübergreifend ausgelagert.

5.5.3 Schnittstellenbeschreibung

5.5.3.1 Schnittstellenbeschreibung

Beschrieben werden die serverseitigen Pfade, welche von uns erstellt oder verändert wurden. Der Requesttyp wird nicht aufgelistet, da nur GET-Requests verwendet wurden.

Nr.	Pfad und Methode	Beschreibung
1	/rest/branch/{branchName}/ build/{buildName}/usecase/ loadUseCaseSummaries(String branchName, String buildName)	Lädt die Liste aller Features von einem Build X im Branch X.
2	/rest/branch/{branchName}/ build/{buildName}/documentation)	Lädt ein Dokumentationsfile aus dem docs Ordner.

	getFile(String path, String branchName, String buildName, String referrer)	

Tabelle 22: Schnittstellenbeschreibung

5.5.3.2 Pfadbeschreibung

Beschrieben werden die Pfade, welche in der Clientapplikation aufgerufen werden können, die von uns erstellt oder verändert wurden.

Nr.	Pfad	Beschreibung
1	'/' redirectTo: '/feature'	Weiterleitung zur Feature-View
2	'/dashboard' templateUrl: 'dashboard/dashboard.html' controller: 'DashboardController'	DashboardController lädt Map-View
3	'/detailNav' templateUrl: dashboard/documentationView.html controller: 'DashboardController'	DashboardController lädt Documentation-View
4	'/testScenarios' templateUrl: dashboard / scenarioView.html controller: 'DashboardController'	DashboardController lädt Scenario-View
5	'/feature' templateUrl: dashboard / featureView.html controller: 'DashboardController'	DashboardController lädt Feature-View

Tabelle 23: Pfadbeschreibung

5.6 Qualitätssicherung

5.6.1 Testing

5.6.1.1 Integration Tests

Aktionen sind immer ein Klick auf das genannte Feature, die genannte Ansicht oder Methode. Manchmal werden nur Feature und Ansicht angegeben (Feature/Ansicht).

In dem Test werden nur drei Features benutzt:

Abbildung 34: Benutze Features

GUI-Elements besitzt selber keine Subfeatures mehr. Alle Features, ausser Home, beinhalten ein Markdown File, eine Specification und Scenarios.

Aufgrund eingeschränkter Platzverhältnissen in den Tabellen werden folgende Abkürzungen verwendet:

Meldung = Ein Hinweis, dass diese Ansicht keine Daten enthält.

Bc = Breadcrumbsleiste

ScX = Scenario X

Md = Markdown File

spec = Specification File

Ansichten

Schritt Nr.	Ausgangslage	Aktion	Soll-Ausgabe	Ist-Ausgabe	OK / NOK
1	Home/Feature	Ansicht: Map auswählen	Home/Map wird angezeigt	Home/Map wird angezeigt	OK
2	Home/Feature	Ansicht: Docs auswählen	Home/Docs wird angezeigt	Home/Docs wird angezeigt	OK
3	Home/Feature	Ansicht: Scenarios auswählen	Home/Scenarios +Meldung wird angezeigt	Home/Scenarios +Meldung wird angezeigt	OK
4	Client/Docs	Expand all klicken	Md,spec,GUI-Elements expanded	Md,spec,GUI-Elements expanded	OK
5	Client/Docs	Collapse all klicken	Md,spec,GUI-Elements collapsed	Md,spec,GUI-Elements collapsed	OK
6	Client/Docs	GUI-Elements expand Pfeil klicken	GUI-Elements expanded Md,spec collapsed	GUI-Elements expanded Md,spec collapsed	OK
7	Home/Feature	showMetaData	Metadata showing	Metadata showing	OK
8	Home/Feature	hideMetaData	MetaData hidden	MetaData hidden	OK

9	Home/Feature	Search «Client»	Features containing «Client»	Features containing «Client»	OK
10	Home/Map	Client Card anklicken	Client/Map +Meldung	Client/Map +Meldung	OK
11	Client/Map	GUI-Elements Card anklicken	GUI-Elements/Map +Meldung	GUI-Elements/Map +Meldung	OK

Tabelle 24: Integrationstests mit Ansichten

Features

Schritt Nr.	Ausgangslage	Aktion	Soll-Ausgabe	Ist-Ausgabe	OK / NOK
12	Home/Feature	Client in Tabelle anklicken	Client/Feature	Client/Feature	OK
13	Client/Feature	GUI-Elements in Tabelle anklicken	GUI-Elements/Feature +Meldung	GUI-Elements/Feature +Meldung	OK
14	Home/Docs		Meldung+ Client	Meldung+ Client	OK
15	Home/Docs	Select feature (Client)	Client/Docs	Client/Docs	OK

Tabelle 25: Integrationstests mit Features

Breadcrumbs

Schritt Nr.	Ausgangslage	Aktion	Soll-Ausgabe	Ist-Ausgabe	OK / NOK
16	Client/Feature	Bc - Home	Home/Feature	Home/Feature	OK
17	GUI-Elements/Scenarios	Sc1	Show Sc1	Show Sc1	OK
18	Soll (21)	Bc - Client	Client/Scenarios	Client/Scenarios	OK
19	Soll (21)	Bc - Home	Home/Feature	Home/Feature	OK

Tabelle 26: Integrationstests mit Breadcrumbs

Feature Explorer

Schritt Nr.	Ausgangslage	Aktion	Soll-Ausgabe	Ist-Ausgabe	OK / NOK
20	Home/Feature	Feature Explorer – Client	Client/Feature	Client/Feature	OK
21	Home/Feature	Feature Explorer – GUI-Elements	GUI-Elements/Feature	GUI-Elements/Feature	OK
22	GUI-Elements/Feature	Feature Explorer - Home	Home/Feature	Home/Feature	OK
23	Home/Feature	Feature Explorer - close	Hide Feature Explorer	Hide Featture Explorer	OK
24	Soll (17)	Feature Explorer – Text oder Pfeil	Show Feature Explorer	Show Feature Explorer	OK

Tabelle 27: Integrationstests mit Feature Explorer

5.6.1.2 E2E Integration Tests

Für die E2E Integration Tests wurde die von Scenarioo erstellte JavaScript Writer Library eingesetzt, um eine Screenshot Basierte Testdokumentation für die erstellten Ansichten zu erstellen, und die Benutzerbarkeit der Erweiterung zu gewährleisten.

Hierfür wurde ein neuer Gulp Task erstellt, welcher ausschliesslich die E2E-Integrationtests der Erweiterung ausführt.

```
154 gulp.task('test-e2e-scenarioo', function () {
155     gulp.src(['./test/protractorE2E/specs/**/*.js'])
156         .pipe(protractor({
157             configFile: './protractor-e2e-scenarioo.conf.js'
158         }))
159         .on('error', function (e) {
160             throw e;
161         });
162 });
163
164 gulp.task('test-e2e-dashboard', function () {
165     gulp.src(['./test/protractorE2E/specs/dashboard/**/*.js'])
166         .pipe(protractor({
167             configFile: './protractor-e2e-scenarioo.conf.js'
168         }))
169         .on('error', function (e) {
170             throw e;
171         });
172 });
```

Abbildung 35: Gulp-Task für E2E Tests

Dieser Task wurde angelehnt an die bereits für das alte Scenarioo implementierten E2E-Tests. Dieses etwas komplexere Testsetup wurde vom bestehenden Scenarioo übernommen.

Um die E2E Tests auszuführen, muss zuerst im Server der korrekte Testdatensatz geladen sein. Dieser Testdatensatz wird durch das Gradle Submodule «scenarioo-docu-generation-example» auf einem aktuellen Stand gehalten. Um den Datensatz zu generieren müssen also diese Tests im Submodule zuerst ausgeführt werden. Dadurch wird im Ordner «build/scenariooDocuExample» der Testdatensatz erzeugt.

```
cd scenarioo-docu-generation-example

scenarioo-docu-generation-example>gradlew test
```

Abbildung 36: Testdaten erstellen

```
Starting a Gradle Daemon (subsequent builds will be faster)
:scenarioo-server:compileJava UP-TO-DATE
:scenarioo-server:processResources UP-TO-DATE
:scenarioo-server:classes UP-TO-DATE
:scenarioo-server:jar
:scenarioo-docu-generation-example:compileJava UP-TO-DATE
:scenarioo-docu-generation-example:processResources UP-TO-DATE
:scenarioo-docu-generation-example:classes UP-TO-DATE
:scenarioo-docu-generation-example:compileTestJava
:scenarioo-docu-generation-example:processTestResources UP-TO-DATE
:scenarioo-docu-generation-example:testClasses
:scenarioo-docu-generation-example:test

BUILD SUCCESSFUL

Total time: 19.529 secs
```

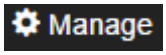
Abbildung 37: Testdaten erfolgreich erstellt

```
D:\s\scenarioo-docu-generation-example>dir build\scenarioDocuExample
 Datenträger in Laufwerk D: ist Data
 Volumeseriennummer: 005F-60DE

 Verzeichnis von D:\s\scenarioo-docu-generation-example\build\scenarioDocuExample

13.06.2017  14:54    <DIR>          .
13.06.2017  14:54    <DIR>          ..
13.06.2017  14:54    <DIR>          wikipedia-docu-example
13.06.2017  14:54    <DIR>          wikipedia-docu-example-dev
               0 Datei(en),              0 Bytes
               4 Verzeichnis(se), 16'241'180'672 Bytes frei
```

Abbildung 38: Verzeichnis der Testdaten

Hier sind die Testdatensätze generiert worden.

Nun muss dieser Ordner im Server als «Documentation Data Directory Path» konfiguriert werden.

Dazu klickt der Nutzer auf 

und dann auf den Tab «General Settings».

Hier kann der Pfad eingegeben werden. Dieser ist {{GIT_ROOT}}/scenarioo-docu-generation-example/build/scenarioDocuExample. Wobei {{GIT_ROOT}} durch den Pfad zum lokalen Repository ersetzt werden muss.

Danach kann im Tab «Builds» auf 

geklickt werden.

Hiernach muss gewartet werden, bis alle Builds wieder importiert wurden.

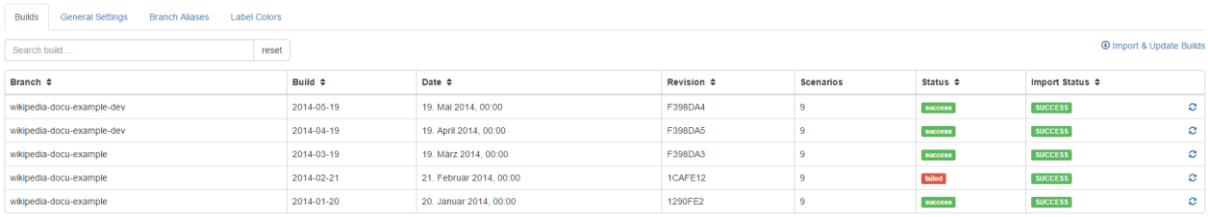


Abbildung 39: Buildübersicht

Danach kann der E2E Test gestartet werden. Hierfür werden zwei Konsolenfenster gestartet.

In einem Konsolenfenster wird ein die Clientseite nochmals mit Gulp bereitgestellt, damit der Protractor Testrunner darauf zugreifen kann.

```
D:\s\scenarioo-client>gulp serve
[15:24:17] Using gulpfile D:\s\scenarioo-client\gulpfile.js
[15:24:17] Starting 'environmentConstants'...
[15:24:17] Finished 'environmentConstants' after 995 µs
[15:24:17] Starting 'watch'...
[15:24:19] Finished 'watch' after 2.12 s
[15:24:19] Starting 'inline-templates'...
[15:24:19] Starting 'less'...
[15:24:20] Starting 'reload-files'...
[15:24:20] Finished 'reload-files' after 2.76 ms
[15:24:20] Starting 'reload-files'...
[15:24:20] Finished 'reload-files' after 997 µs
[15:24:20] Finished 'less' after 387 ms
[15:24:20] Finished 'inline-templates' after 779 ms
[15:24:20] Starting 'serve'...
[15:24:20] Finished 'serve' after 16 ms
[15:24:20] Server started http://localhost:9000
[15:24:20] LiveReload started on port 35729
[15:24:20] Starting 'reload-files'...
[15:24:20] Finished 'reload-files' after 1.03 ms
```

Abbildung 40: Clientseite mit Gulp

Im zweiten Fenster wird der Test ausgeführt. Dieser startet dann automatisch eine Browser-Instanz, mit welcher die Screenshots für Scenarioo erstellt werden.

```
D:\s\scenarioo-client>gulp test-e2e-dashboard
```

Abbildung 41: Starten der E2E Tests

```
[15:27:18] Starting 'test-e2e-dashboard'...
[15:27:18] Finished 'test-e2e-dashboard' after 7.41 ms
PROTRACTOR_BASE_URL: http://localhost:9000
BRANCH: local_dev
[15:27:18] I/direct - Using ChromeDriver directly...
[15:27:18] I/launcher - Running 1 instances of WebDriver
Reporting scenarios for scenarioo. Writing to "D:\s\scenarioo-client\scenariooDocumentation"
Started

Clear local storage for user visiting for the first time
 SUCCESS scenario "dashboard_feature_explorer - Reset" .
 SUCCESS scenario "dashboard_feature_explorer - Expand-Collapse" .
 SUCCESS scenario "dashboard_feature_explorer - Select Features in tree" .
 SUCCESS scenario "dashboard_feature_explorer - Navigation on Sub" .
SUCCESS use case "dashboard_feature_explorer": 4 passed, 0 failed, 0 pending

Clear local storage for user visiting for the first time
 SUCCESS scenario "dashboard_feature_view - First visit" .
 SUCCESS scenario "dashboard_feature_view - Navigation" .
SUCCESS use case "dashboard_feature_view": 2 passed, 0 failed, 0 pending
All done!
```

Abbildung 42: E2E-Tests erfolgreich beendet

Die Tests sind erfolgreich durchgelaufen.

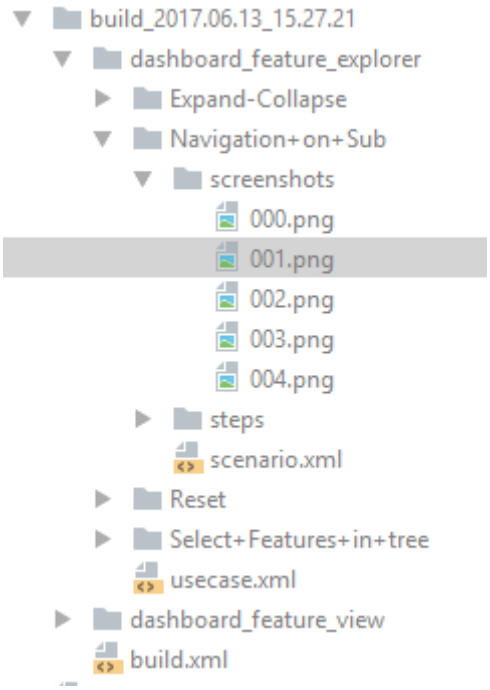


Abbildung 43: Resultate im alten Format

Nun wurde die Screenshot Dokumentation im alten Scenarioo Datenformat erstellt, da die JavaScript Writer Library in dieser Arbeit nicht verändert wurde.

5.6.1.3 Usability Test

Erkenntnisse

Am 18.05.2017 wurde bei der Zühlke ein Usability-Test durchgeführt mit drei verschiedenen Testpersonen:

Testperson 1 wusste ungefähr, was Scenarioo ist, hat es aber noch nie im Arbeitsumfeld benutzt.

Testperson 2 hatte noch nie davon gehört.

Testperson 3 sitzt in der Focus-Gruppe für Scenarioo und wusste grob über unsere Erweiterungen Bescheid.

Durch diese drei sehr unterschiedlichen Nutzer gab es ein breites Spektrum an Verbesserungsvorschlägen. Diese werden nachfolgen den Personen zugeordnet aufgelistet (Die Auflistung wird ergänzend sein, also bereits genannte Probleme werden nicht noch einmal erwähnt, da es doch diverse grundlegende Mängel gab).

Testperson 1

- Das aktuelle Feature wird markiert, ausser man ist auf dem Home-Feature. Das sorgt bereits am Anfang für Verwirrung, da man nicht weiss wo man ist.
- Der Feature Explorer ist anfangs eingeklappt und gibt sich nicht als aufklappbar zu erkennen.
- Sind keine Features oder Scenarios vorhanden werden leere Listen angezeigt, das sorgte bei allen Teilnehmern für Verwirrung. Sie wünschten sich eine direkte Weiterleitung zu einem vernünftigen Ort. (Schlecht möglich da man nicht weiss was der Benutzer sehen will.)

- Aufgrund der Hintergrundfarbe war nicht ersichtlich, dass jedes Listenelement ein Link war, das sorgte für Verwirrung, da sich die Nutzer unterschiedliches Verhalten der unterschiedlichen Zellen erhofften. Als Verbesserung könnte man die Namen der Features als Links darstellen, damit die Funktion dort klar gekennzeichnet ist,
- Eine bessere Einführung in die hierarchische Struktur wurde von der Testperson gewünscht.
- Auf der Docs-View sind diverse aufklappbare Elemente nicht als solche markiert.
- Ebenfalls auf der Docs-View waren die Subfeatures nicht als solche markiert und wurden darum auch nicht wirklich als solche erkannt.
- Über alle Features wird in den Breadcrumbs nur eines gezeigt. Dort sollte der ganze Weg durch die Feature-Hierarchie gezeigt werden, um besser nach oben navigieren zu können.
- Die Map-View muss mit Hilfe von Design klar zeigen, welche Features Subfeatures sind und welche Subsubfeatures sind.
- Bei der Anzeige der Test ist die dritte Position nicht klar. Hier sollte eventuell eine bessere Farbe als Violett verwendet werden.
- Nachdem die Struktur der Features verstanden wurde, wurde sie automatisch auf die Scenarios übertragen, was wieder für Verwirrung sorgte.
- Bei der Docs-View muss klar ersichtlich sein, welche Dokumente zu welchem Feature gehören.
- Die verwendeten Farben auf der Docs-View gaben eine falsche Message, was sich darunter verbergen könnte.
- Als Designvorschlag für die Docs-View gab er uns die Mobileseite von Wikipedia an.
- Alle Ansichten zeigen zu wenig Information zum aktuellen Feature und nur Informationen zu den Subfeatures oder Scenarios.

Testperson 2

- Das Wort Feature wurde zu oft gebraucht, da die Beispieldaten die Self-Documentation dieses Projektes waren, kam das Wort gleich noch öfter vor.
- Die Beschreibung der Features wird noch nicht angezeigt, es steht nur «<Feature-Name> desc.»
- Mit einem Hover-Element könnte man bei der Testanzeige weiter Informationen geben.
- Feature Explorer schliesst nicht konsistent: manchmal reicht klick in freien Raum, manchmal muss er über «close» geschlossen werden.
- Name der Feature-View sorgt für Verwirrung, da die anderen Views auch Features zeigen.
- Es wird geraten, den Feature Explorer immer anzuzeigen.
- Kontext erstellen indem der Name des ausgewählten Features über die Ansichtsauswahl gestellt wird. (Feature>Ansicht>Daten anstelle von Ansicht>Feature>Daten)
- Dashboard: Name wird falsch interpretiert (besser Storyboard oder Map).

Testperson 3

- Der Feature Explorer markiert gleich benannte Features.
- «Dashboard» sorgt für Verwirrung, wenn keine Subsubfeatures verfügbar sind.
- Docs-View zuerst eingeklappt.

Interessant am letzten Punkt ist, dass bei den ersten beiden Testpersonen auf der Docs-View die Markdowns und Subfeatures eingeklappt waren und beim letzten Teilnehmer ausgeklappt. Jede Person schlug jedoch das Gegenteil als Default vor.

Zusätzlich gab es noch ein paar Hinweise von den Betreuern:

- Die Beschreibung eines Features sollte unter dem Titel der Seite sein um mehr Klarheit zu schaffen.

- DOKU-View sollte umbenannt werden, da es kein englischer Begriff ist (besser DOCS oder SPECS).
- Hat man den Feature Explorer offen und die Docs-View, sind die Collapse-Befehle synchron, da die gleiche Variable verwendet wurde.

Verbesserungen

Nach dem Usability Test wurden die gefundenen Fehler und Schwächen aufgeschrieben und grob priorisiert.

Prio	Titel	Beschreibung	T-Shirt size	Done
1	Docs-Folder	Fixen Ordner für .md Files damit einfach lokale links darauf generiert werden können. Bis anhin waren hier nur statische links vom Root Folder aus möglich.	M	OK
last	Serach	Volltextsuche für alle Elemente einrichten	L	OK
	Sort	Tabellen lassen sich noch nicht sortieren mit den übernommenen Funktionen	M/L	OK
	Details	übernehmen der Detail-Anzeige von altem Scenarioo	L	NOK
	Labels	übernehmen der Label Funktionen von altem Scenarioo	M	OK
1	Wiki-Design	Docs-View an mobile-wiki design anpassen	M	OK
1	Breadcrumbs	Aktueller Pfad des Featuretree in Breadcrumbs	M/L	OK
1	Feature Explorer	Soll Anfangs immer offen sein, mit «open/close» Button, kein autohide	S	OK
1	Kontext	Durch verschieben von Featuretitel und Ansichten-Tab den Kontext verbessern	S	OK
	Design Bootstrap	Anpassen des Designs an altes Scenarioo (Bootstrap 3)	L	NOK
1	Navigation Elements	Collapsable anzeigen durch Pfeile.	S	OK
1	Rootfeature zu Home	Rename und Breadcrumb-Link anpassen	S	OK
1	Leere Views	Meldung, wenn keine Sub oder Subsub verfügbar	S	OK
	isCollapsed fixen	Gleiche Namen werden doppelt markiert	S	OK
	Tableclick Kontext abhängig	z.B. Klick auf Tests führt zu Scenarios oder Spec	M	NOK
1	Dashboard fixes	Rename, Subsub einrücken, verständlicher durch besseres Design	S	OK
1	folderName->Id	Rename	S	OK
	Feature nav in scenario/step	Feature Explorer sollte (fast) überall sichtbar sein	M	OK
	Tabs auf Home	Tabs aus dem alten Scenarioo sollten zugreifbar werden, evtl von Home aus	L	OK

Tabelle 28: Auswertung Usability Test

5.6.2 Review

5.6.2.1 Code Review

Am 25.5.2017 wurde ein Code Review durchgeführt, bei dem der im Verlauf dieses Projektes generierte Code im Client gereviewt wurde. Die gefundenen Fehler wurden in einer Liste festgehalten und dann abgearbeitet.

Nr:	Problem	Filename	Status	Kommentar
1	Style auslagern generell	App/dashboard	OK	
2	Card color entfernen	featureDetail.js	OK	
3	Component-Naming vereinheitlichen	Comp	OK	
4	Feature-Details number of md löschen	feature-details.html	OK	
5	Zu lange Zeilen und Kommentare entfernen, style auslagern, «contains» Funktion auslagern	fileviewer	OK	Style (1)
6	Module zuerst definieren, danach Controller	markdown.js	OK	
7	$rootScope Dependency entfernen, replaceLocalLinksInContainer Parameter entfernen	markdown.js	OK	
8	Rename classname md-container	markdown.js	OK	
9	«replace» Parameternamen ausschreiben, Kommentar entfernen	markdown.js	OK	
10	Methode extrahieren für «replaceFirstSlash»	markdown.js	OK	
11	style; Contoller comment löschen; view remove getColor	nav-tree	OK	
12	“clickfeature” as a service	nav-tree	OK	
13	Überflüssige Bindings entfernen	sidepanel	OK	
14	«close» Funktion entfernen style Bindings entfernen	sidepanel	OK	Style (1)
15	log entfernen, local storage key auslagern, pink zu lightskyblue, controller as: setzten	site-nav	OK	Pink fix
16	color fix	testoverview	OK	
17	$scope entfernen, Abstand zwischen Zuweisungen, log entfernen, if curly braces oder eine Zeile, style, Binding size entfernen	treeNav	OK	
18	löschen	accordion	OK	
19	isCollapsed, isExplorerCollapsed löschen	dashboardcontroller	OK	
20	log entfernen, Semicolon nach Methode, navsize löschen	dashboardcontroller	OK	
21	clickfeature in feature service, label anzeigen	dashboardcontroller	OK	
22	«load» und «activate» zusammenfügen	dashboardcontroller	OK	
23	comparison info in watch Methode löschen	dashboardcontroller	OK	
24	watch (feature) umbenennen, eq auslagern in service	dashboardcontroller	OK	

25	def zu isDefined, collapseAll 2 umbenennen	dashboardcontroller	OK	
26	contains auslagern, handleClick zu clickScenario	dashboardcontroller	OK	
27	search, filter, comments entfernen, order configurable	dashboard.html	OK	
28	data.controller, data.html, example-data.js/.zip entfernen	app/dashboard	OK	
29	header setzten	app	OK	
30	filter	documentation View	OK	
31	labels: Comment raus oder Funktion integrieren	featureview	OK	
32	Allgemein: Comments entfernen,	featureview	OK	
33	onNavigationTAblehit (sc-keyboard…)	scenario/feature view	OK	
34	rename usecase to feature, Namensgebung (Arr, loc, str)	FeatureService.js	OK	
35	localstorage key auslagern, rename getFeatureByArray (params)	FeatureService.js	OK	
36	getCurrentFEatures erstes if var entfernen	FeatureService.js	OK	
37	Comment entfernen, setInternalAfterLoad Parameter	FeatureService.js	OK	
38	def zu define rename	FeatureService.js	OK	
39	var branch und build nach oben	FeatureService.js	OK	
40	getMilestone filter entfernen	FeatureService.js	OK	
41	stati rename in status	FeatureService.js	OK	
42	getStatus refactor long methode	FeatureService.js	OK	
43	uscase replace with feature	FeatureService.js	OK	
44	loadscenariodiffinfoint refactor long methode	FeatureService.js	OK	

Tabelle 29: Auswertung Codereview

5.6.3 Code Analyse

Client

File Type	Lines of Code	Lines of Code ohne Kommentare
Cascading style sheet	775.0	719.0
Git file	1.0	1.0
HTML	2574.0	2062.0
JavaScript	8363.0	6456.0
Less	749.0	689.0
Text	1.0	
XML	459.0	458.0
Total	12922.0	10385.0
Average	1846.0	1730.83

Tabelle 30: Codeanalyse Client

Das ist die Auswertung des Clients, wobei ca. 1500 Zeilen in diesem Projekt generiert wurden.

Server

File Type	Lines of Code	Lines of Code ohne Kommentare
HTML	2.0	2.0
Java	16120.0	12373.0
Manifest	2.0	2.0
Properties	7.0	6.0
Text	2.0	
XML	250.0	250.0
Total	16383.0	12633.0
Average	2730.5	2526.6

Tabelle 31: Codeanalyse Server

Die Auswertung des Servers zeigt, dass es doch ein grosses Projekt ist.

5.7 Deployment

Die Änderungen dieser Arbeit am Scenarioo Projekt haben das Deployment nicht tangiert und es wurde deshalb nicht verändert.

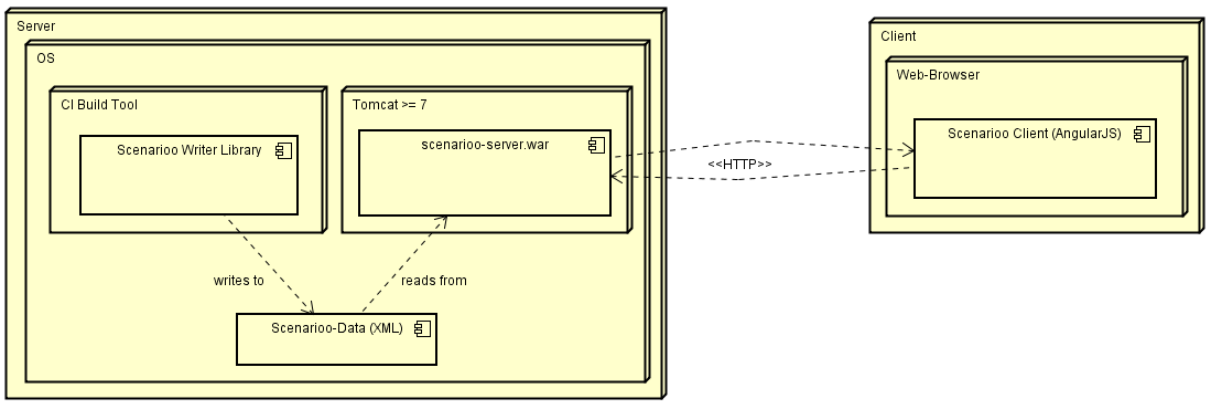


Abbildung 44:Deployment Modell

Der Server wird als .war Archivdatei deployt. Somit kann er auf allen Systemen ausgeführt werden, welche dieses Dateiformat unterstützen. Beispielsweise in einer Tomcat Instanz auf Windows, Linux oder Mac OS X.

Der Client wird mit dem Server zusammen in der Archivdatei ausgeliefert und kann aus einem Webbrowser aus aufgerufen werden.

Die Writer Libraries sind in Testing-Frameworks integriert und können so dem Buildprozess eines Projektes angefügt werden.

5.8 Schlussfolgerung & Ausblick

Für die Zühlke sollte es nun möglich sein mit ein wenig Anpassungen die Änderungen in Scenarioo zu veröffentlichen. Das Konzept konnte umgesetzt werden und die meisten Wünsche des Kunden wurden integriert. Durch die Integration der erarbeiteten Erweiterungen können die Möglichkeiten von Scenarioo vergrössert werden und es kann für noch mehr Projekte verwendet werden.

Um mit möglichst wenig Aufwand ein gutes Resultat zur Veröffentlichung zu erhalten, wäre es am besten, möglichst schnell gute Testdaten zu erstellen. Nur so können bestehende Funktionen ausgiebig getestet werden und nötige Veränderungen zur Integration aufgedeckt werden.

Ein weiterer Punkt ist die Einarbeitung weiterer Datenformate, da so nochmals ein erheblicher Mehrwert generiert werden kann.

5.9 Glossar

Scenarioo – Name der Software von Zühlke

Scenario(Szenario) – Teilschritt eines Usecases in Scenarioo, besteht aus mehreren Steps

Feature – Hauptstück der neuen Datenstruktur, beinhaltet weiter Features und Scenarios, zusätzlich je ein Markdown- und ein Specificationfile.

Feature – Ebenfalls Name der Ansicht, die die Features als einfache Liste anzeigt.

Scenarios – Name der Ansicht eines Features, die eine Liste der Scenarios des Features zeigt.

Step (Schritt) – kleinster Teilschritt eines Usecases, besteht aus einer oder mehreren Pages

Page (Seite) – Printscreen einer UI-Ansicht die bei einem Step verwendet wird.

Dokumentationsartefakte – Sind Files die einem Feature angehängt werden können, in unserem Falle sind das Specification-, Markdownfiles.

Map (anfangs dashboard) – Ansicht der Features als Kacheln, wobei ein Build oder ein Feature gewählt ist und man die Subfeatures in einer Zeile sieht und die zugehörigen Subsubfeatures darunter.

Docs (oder detailNav) – Ansicht, in der ein Build oder Feature gewählt ist und man die zugehörigen Markdwon und Specfiles sieht und darunter die Subfeatures mit den jeweiligen Files.

Label – Kennzeichnung, die an Features (resp. Alles aus der alten Datenstruktur) angehängt werden kann, damit man danach suchen kann.

Markdownfile (oder md) – Ein beliebiges Textfile im Markdown-Format, welches zusätzliche Informationen zu einem Feature enthält.

Specificationfile (oder Spec) – Enthält eine Testdefinition, evtl. schon mit Resultaten (falls vorhanden) in einem Code-Format (.java, .cs, .feature, .js).

Feature Explorer – Ist unser Hauptnavigationselement. Es befindet sich auf der linken Seite und ist standardmässig geöffnet, kann aber geschlossen werden. Es zeigt einen Baum über alle verfügbaren Features und markiert das aktuell Gewählte.

5.10 Literaturverzeichnis

- L. Racchetti, L. Tacconi and C. Fantuzzi, "Generating automatically the documentation from PLC code by D4T3 to improve the usability and life cycle management of software in automation," *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, Gothenburg, 2015, pp. 168-173.
  doi: 10.1109/CoASE.2015.7294057
  keywords: {IEC standards;programmable controllers;software management;source code (software);D4T3 documentation;Doxygen-for-TwinCAT3;IEC61131-3;PLC code;PLC-world features;automatic documentation generation;documented PLC software maintainability improvement;documented PLC software understanding improvement;documented PLC software usability improvement;documented PLC source files;object oriented features;offline reference manual;online manual;software life cycle management improvement;undocumented PLC source files;Automation;Documentation;Generators;Guidelines;Layout;Usability},
  URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7294057&isnumber=7294025
- Rodrigo Souza and Allan Oliveira. 2017. GuideAutomator: continuous delivery of end user documentation. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track* (ICSE-NIER '17). IEEE Press, Piscataway, NJ, USA, 31-34. DOI: https://doi.org/10.1109/ICSE-NIER.2017.10
- Scenarioo Documentation : http://scenarioo.org/docs/,
  2017. [Online ; last accessed June-2017]
- Einführungspräsentation: http://scenarioo.org/agile-project-dashboard,
  2017. [Online; last accessed June-2017]
- Highlightjs License: https://github.com/isagalaev/highlight.js/blob/master/LICENSE
  2017. [Online; last accessed June-2017]
- Showdownjs License: https://github.com/showdownjs/showdown/blob/master/license.txt
  2017. [Online; last accessed June-2017]
- AngularJS Material License: https://github.com/angular/material/blob/master/LICENSE
  2017. [Online; last accessed June-2017]

5.11 Abbildungsverzeichnis

5.12 Tabellenverzeichnis

A. Anhang

A.1 Projektdokumente

A.1.1 Anleitungen

A.1.1.1 Installation der Scenarioo Applikation

Im Abgebe Ordner befindet sich die aktuelle Version der Scenarioo Applikation als .war Archivdatei.

Diese kann mit einer Tomcat Installation ausgeführt werden.

Somit ist das erste was getan werden muss um die Applikation laufen zu lassen, Tomcat zu installieren.

Jede Tomcat Version ab der Version 7 sollte die Applikation ausführen können. Für diese Anleitung wurde die Version 8 auf dem Betriebssystem Ubuntu 16.04 verwendet.

Für jedes andere Betriebssystem:

https://tomcat.apache.org/index.html

Sollte hierbei helfen. Ansonsten gibt es dazu viele Anleitungen im Internet.

Installation auf Ubuntu 16.04

Auf Ubuntu kann Tomcat 8 über die Bash Befehle:

```
sudo apt-get install tomcat8 tomcat8-docs tomcat8-examples tomcat8-admin
```

installiert werden. Und über die Befehle:

```
systemctl start tomcat8
systemctl stop tomcat8
systemctl restart tomcat8
```

gesteuert werden.

Konfiguration

Um auf Tomcat über das Webinterface .war Files deployen zu können müssen in der Konfigurations-datei

/path/to/tomcat**/conf/tomcat-users.xml**

noch ein Nutzername und ein Passwort für einen Administrator-Account erstellt werden.

Unter Ubuntu 16.04 liegt diese Datei in:

/var/lib/tomcat8/conf/tomcat-users.xml

Folgendes muss hinzugefügt werden:

```
<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<user username="test" password="test" roles="manager-gui,admin-gui"/>
```

Als Nutzername und als Passwort wurde für diese Anleitung «test» verwendet.

Danach den Tomcat Server neu starten. Unter Ubuntu mit dem Befehl:

```
systemctl restart tomcat8
```

Hiernach ist das Administrationsinterface über folgende URL erreichbar:

http://localhost:8080/manager/html

Hier muss nun noch der Nutzername und das Passwort angegeben werden.

Falls der Tomcat auf einem Remote Server läuft, muss hier localhost durch den entsprechenden Domainnamen ersetzt werden.

Unter der Überschrift «Deploy» kann die Applikation nun auf den Server geladen werden.

Danach sollte Scenarioo über den angegebenen «Context Path» erreichbar sein, also Beispielsweise:

http://localhost:8080/scenarioo

Nun muss für Scenarioo sichergestellt werden, dass die Scenarioo Konfiguration gespeichert werden kann. Standartmässig wird Scenarioo versuchen im «user.home» also im Benutzer-Ordner einen neuen Ordner anzulegen mit dem Namen «.scenarioo». Sollte der Tomcat Server keine Berechtigung haben dort zu lesen und zu schreiben, so muss ein anderer Ort angegeben werden, wo Scenarioo seine Konfiguration speichern kann.

Dies kann auch im Tomcat Conf Ordner getan werden:

/path/to/tomcat**/conf/context.xml**

Unter Ubuntu 16.04 ist dies dann:

/var/lib/tomcat8/conf/context.xml

Hier muss folgender Parameter hinzugefügt werden:

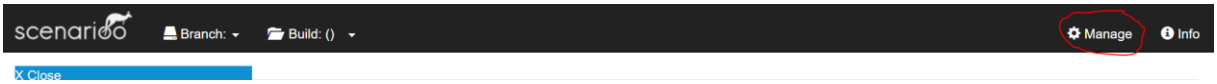
```
<Parameter name="scenariooConfigurationDirectory" value=">>>path to a directory where scenarioo can store its configuration data<<<" override="true" description="Location of scenarioo config.xml file"/>
```

Nun muss noch die Konfiguration für die Testdaten angepasst werden. Die für die Testdaten korrekte «config.xml» Datei ist im Abgabeordner zu finden. Diese muss nun entweder an den Pfad «Benutzer-Ordner -> .scenarioo» oder den Pfad kopiert werden, welcher im context.xml angegeben wurde.

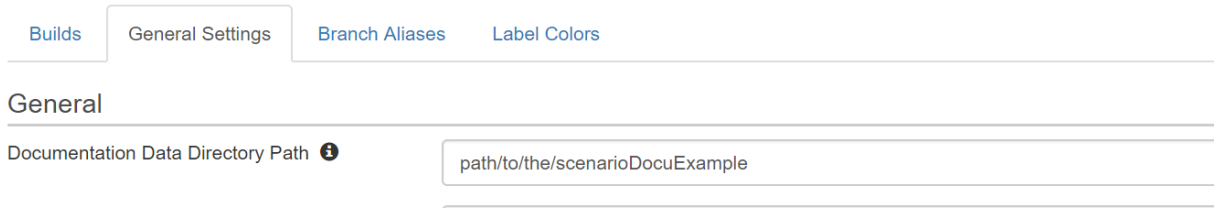
Nun kann aus der Abgabe das Testdatenverzeichnis mit dem Namen «scenarioDocuExample» an einen Ort kopiert werden, an welchem Tomcat auch Lese- und Schreibeberechtigungen hat. Danach muss dieses Verzeichnis in der Scenarioo Konfiguration angegeben werden.

Dazu kann die Scenarioo Applikation geöffnet werden und dann auf «Manage» geklickt werden

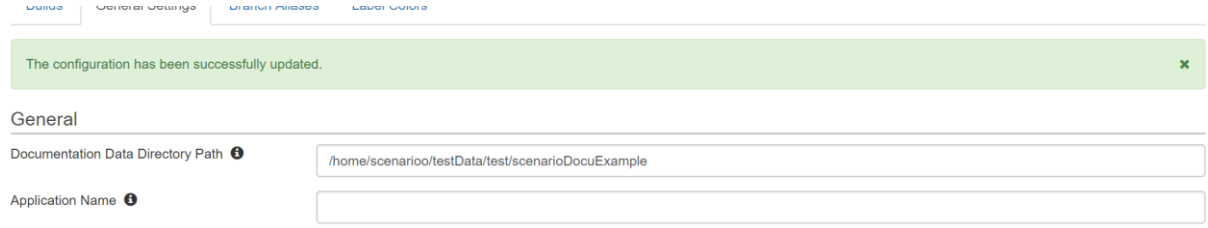
Im dieser Anleitung unter:



Danach wird «General Settings» in den Tabs ausgewählt und der «Documentation Data Directory Path» verändert.



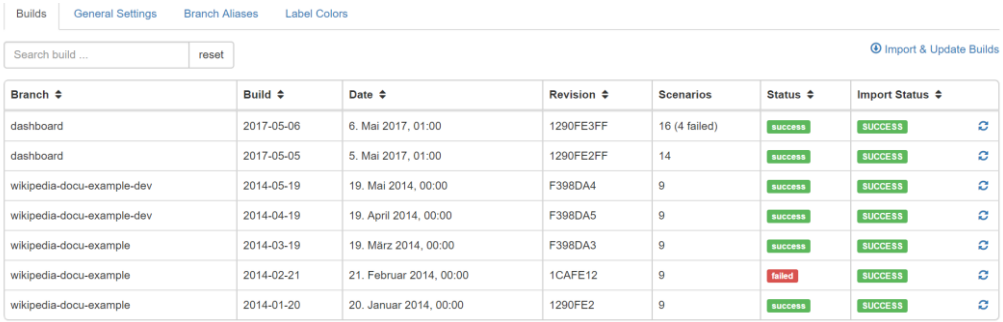
Hiernach muss am Ende der Seite auf «Save» geklickt werden.



Hiernach erscheint oben in der Seite eine Bestätigung. Ist dies nicht der Fall, so hat Tomcat sehr wahrscheinlich keine Berechtigungen, die Konfiguration zu schreiben.

Nach Erfolg kann nun auf den Tab «Builds» gewechselt werden und

geklickt werden. Nach dem Import sollte die Liste wie folgt aussehen:



Falls etwas schiefgeht oder weitere Details benötigt werden, kann die offizielle Dokumentation von Scenarioo helfen:

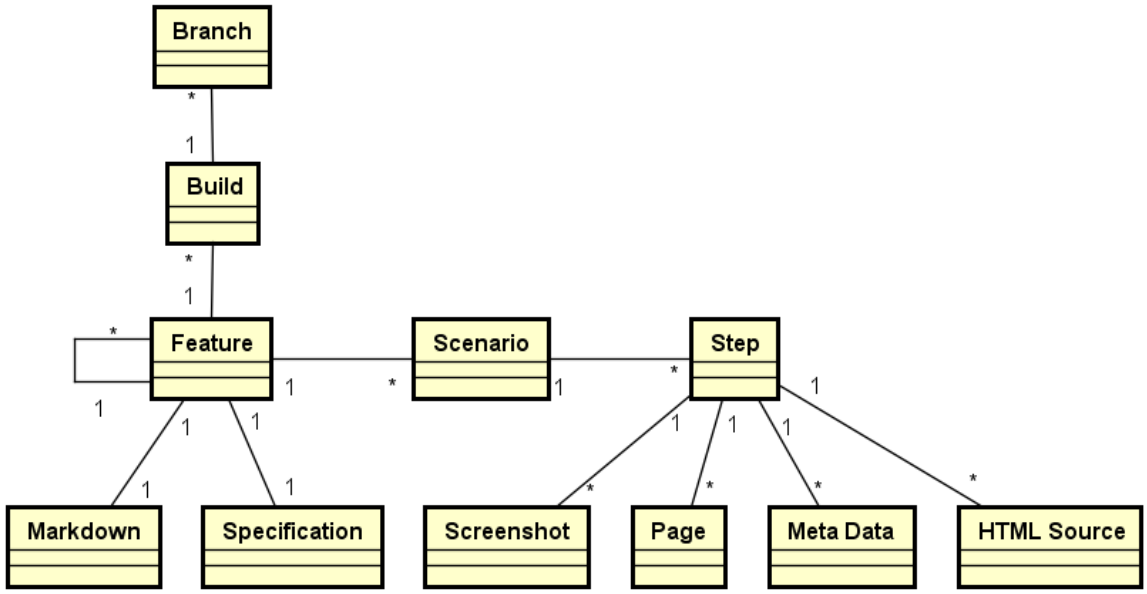
http://scenarioo.org/docs/setup/Scenarioo-Viewer-Web-Application-Setup.html

A.1.1.2 Benutzeranleitung

Installationsanleitung nach einem Release

Die Installationsanleitung kann von der bestehenden Scenarioo Applikation übernommen werden, da die Erweiterung später in das bestehende Projekt integriert wird. Somit sollte sich bei der Auslieferung der kompletten Applikation der Installationsprozess nicht ändern.

Domainmodell



Wichtig beim Domainmodell ist, dass ein Feature mehrere Features als Kinder haben kann und ein anderes Feature sein Parent sein kann. Somit entsteht eine Baumstruktur über alle Features. Dieser Baum ist immer im Feature Explorer ersichtlich.

Dies ist wichtig beim Aufbau einer neuen Projektstruktur, denn man kann Features benutzen um sein Projekt zu strukturieren. Natürlich kann man einfach alle Use Cases als Features hinterlegen, aber man kann so nicht alle Funktionen von Scenarioo nutzen. Nur wenn man gewisse Features als ‘Ordner’ zur Strukturierung erstellt kann man das volle Potenzial ausnutzen.

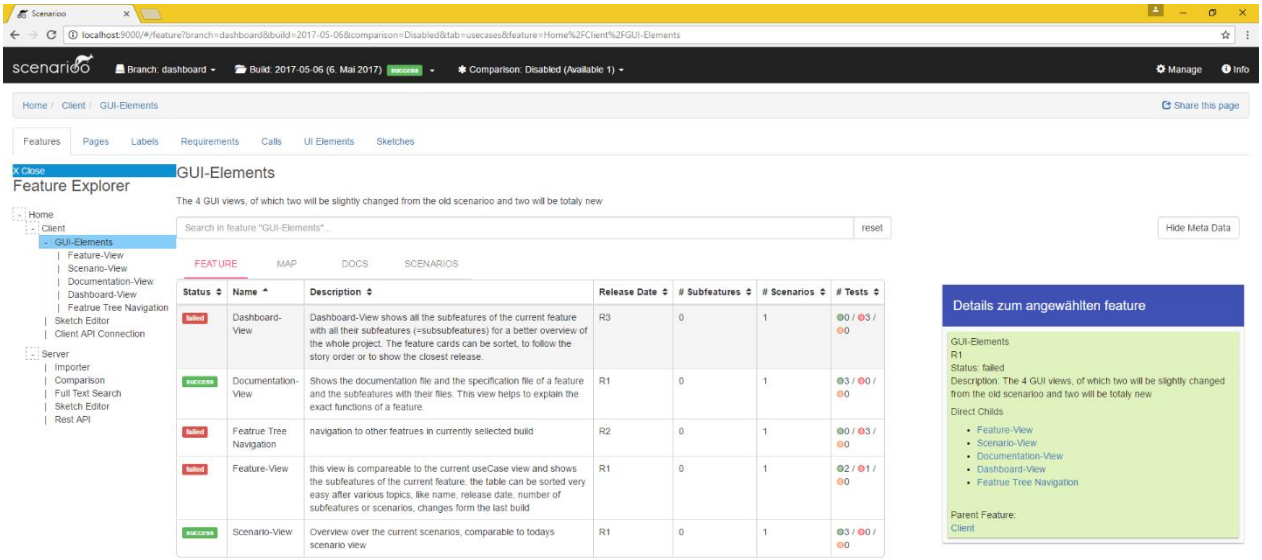
Benutzerguide

Ist Scenarioo aufgesetzt und von einem Client Computer erreichbar, kann man unter ‘’Manage > General Settings’’ den Pfad für das ‘’Documentation Data Directory’’ setzen. Danach sollte der Server neu gestartet werden, damit alle Dateien aus dem gesetzten Verzeichnis importiert werden.

Nach dem Import sieht man auf dem Client die vorhandenen Daten. Für jedes Feature hat man vier verschiedene Ansichten, welche hier kurz vorgestellt werden:

Jede Ansicht zeigt den Namen und die Beschreibung des ausgewählten Features, dieses wird auch im Feature-Explorer markiert, damit man sieht, wo man sich in der Struktur befindet. Zusätzlich können auf der rechten Seite Details zum Feature ein – und ausgeblendet werden.

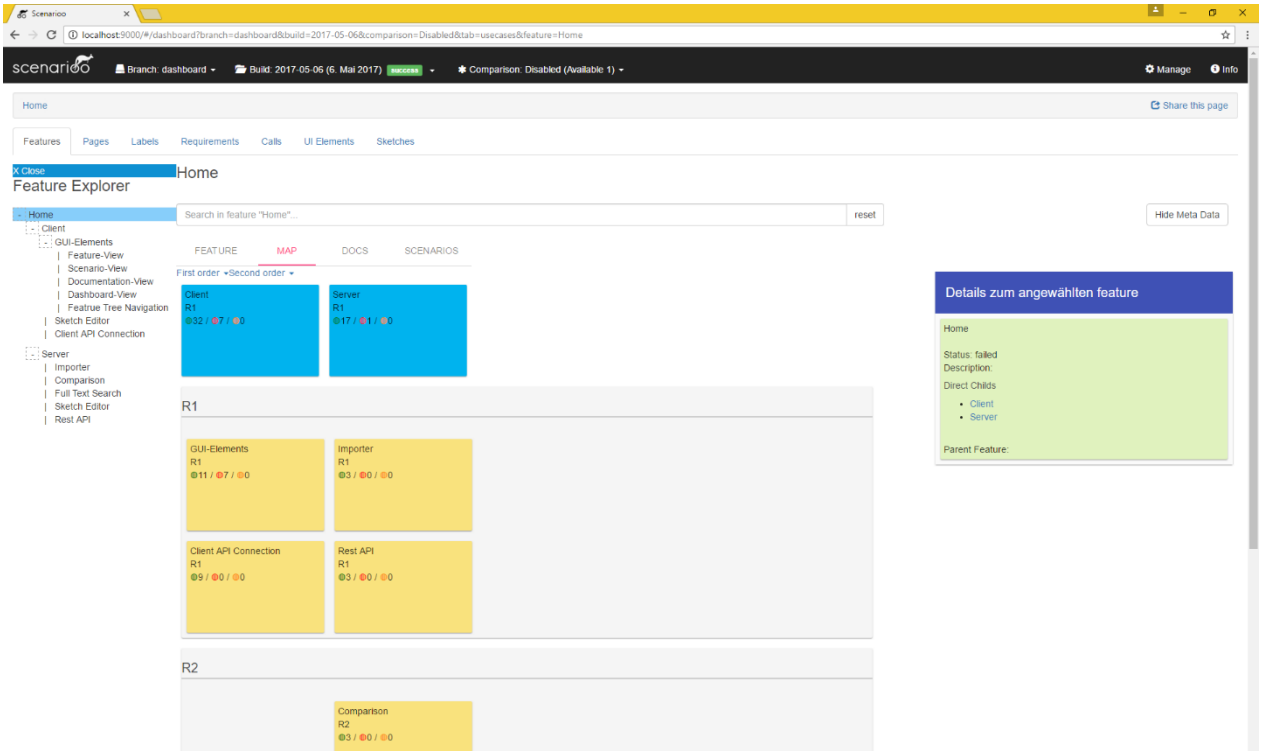
Feature-Ansicht



Die Feature-Ansicht ersetzt die bisherige Use Case Ansicht, auf der eine Liste mit den Use Cases zu sehen war, welche auf die Scenarios in den Use Cases weiterführte. Neu werden die Features angezeigt und beim Auswählen, wird man auf der selben Ansicht die Subfeatures des gewählten Features sehen. Um die Scenarios zu sehen, muss man auf die Scenario-Ansicht. Die Liste der Features zeigt den Status, der von den eigenen Tests aus dem Specification-File und den Scenarios abhängt und von den Subfeatures geerbt wird. Dabei wird immer der Worst-Case angezeigt. Weiter sieht man den Namen, die Beschreibung, den gewünschten Realease, die Anzahl der direkten Subfeatures (Subsubfeatures werden nicht gezählt), die Anzahl Scenarios (ebenfalls ohne Scenarios der Subfeatures) und Testübersicht mit erfolgreichen (grün), fehlgeschlagenen (rot) und ignorierten / nicht implementierten (gelb).

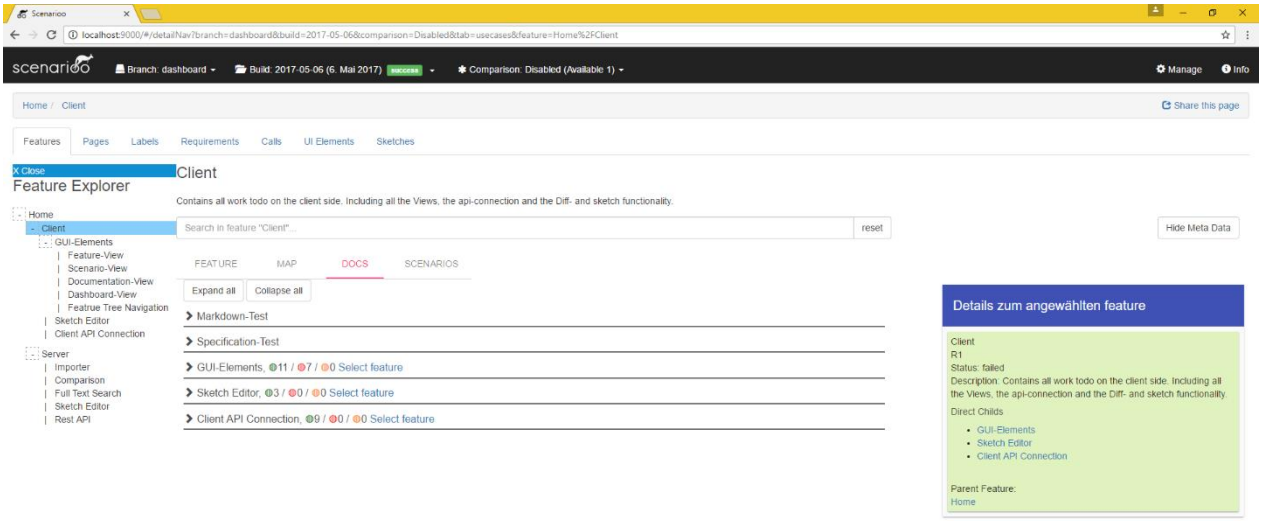
Die Liste kann nach allen Spalten sortiert werden und mit dem Suchfeld kann nach einem Feature oder einem Kind im Baum gesucht werden.

Map-Ansicht



Die Map-Ansicht soll einen Blick in die Zukunft erlauben, indem die Subsubfeatures nach Fertigstellung sortiert werden. Sie macht demnach auch nur Sinn, wenn ein Feature Subsubfeatures enthält, andernfalls ist sie eine weniger nützliche Feature-Ansicht. Mit Subsubfeatures, kann sie jedoch sehr schnell zeigen wo noch Arbeiten offen sind. Sie zeigen über den Milestone, bis wann etwas erledigt werden soll und anhand der Testübersicht sieht man wieviel noch verbessert werden muss.

Docs-Ansicht

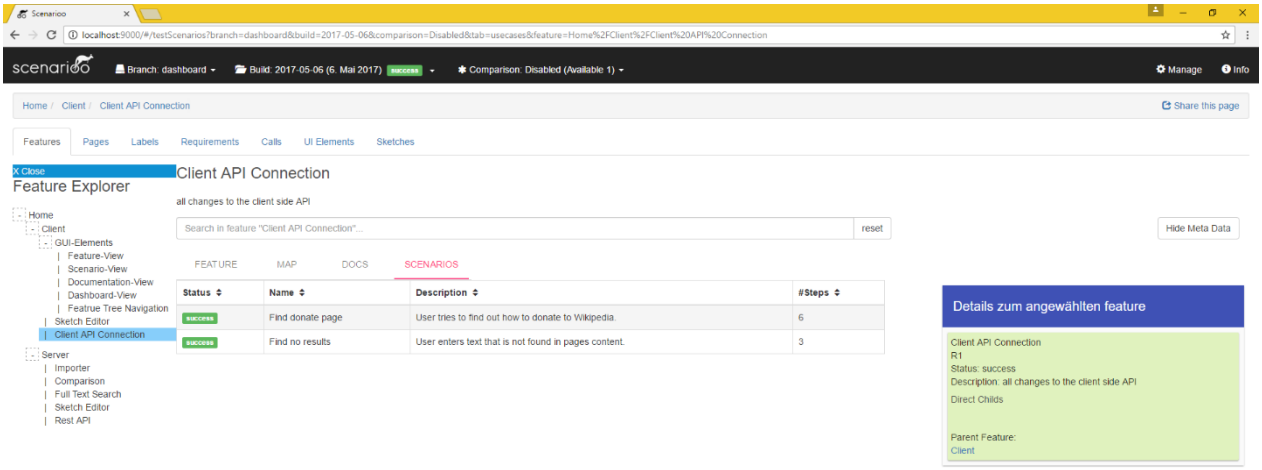


Die Docs-Ansicht soll möglichst alle erarbeiteten Artefakte zu einem Feature zeigen. Dazu ist zuoberst ein Markdown-File zu sehen, dort werden jegliche Informationen, welche im Markdown-Format gespeichert wurden, zu einem File zusammengesetzt und angezeigt. Dies kann von Diagrammen zu Screenshots bis zu einfachen Ideen zur Erweiterung beinhalten.

Danach ist das Specification-File zu sehen, dies sind Testscenarien, eventuell mit Status (wenn schon ausgeführt), die dort aus einem Code-File (*.java, *.cs, *.js, *.feature) angezeigt werden.

Darunter werden die Subfeatures aufgelistet mit ihrer Testübersicht. Sie können aufgeklappt werden und die jeweiligen Dokumente sind direkt ersichtlich. Mit ‘’Select Feature’’ kann man ein Subfeature öffnen um in dessen Subs’ zu suchen. Die Testübersicht soll dort helfen, um im Ausschlussverfahren schnell zu sehen, wo noch Probleme vorhanden sind. Mit den Tasten Expand – Collapse all können alle Dokumente und Subfeatures ein – oder ausgeklappt werden um schnell Übersicht zu gewinnen.

Scenarios-Ansicht



Die Scenario-Ansicht listet alle Scenarios eines Features auf. Zu sehen ist der Status des Scenarios, welcher von den Steps geerbt wird, der Name, die Beschreibung und die Anzahl der Steps im Scenario.

Wählt man ein Scenario aus, öffnet sich dieses und die einzelnen Steps werden angezeigt. Eine Rückwärtsnavigation ist von dort nur im direkten Featurepfad möglich.

A.1.2 Abnahmeprotokoll

Abnahmeprotokoll vom Agile Project Dashboard

Abnahme

Anforderungen

- Es wurde gemeinsam ein Anforderungskonzept erarbeitet und dokumentiert.
- Scenarioo wurde um die besprochenen Funktionen, im Sinne eines Minimum-Viable-Product erweitert.
- Die wichtigsten funktionalen Anforderungen wurden umgesetzt und manuell getestet.
- Das umgesetzte MVP wurde im Rahmen eines Usability Tests mit Benutzern getestet und die Resultate festgehalten und teilweise umgesetzt.
- Durch Reviews angesprochene Punkte wurden entsprechend geändert.
- Die nicht-funktionalen Anforderungen wurden eingehalten.

Dokumentation

- Das erarbeitete Konzept ist hinreichend Dokumentiert.
- Benutzeranleitungen sind vorhanden und verständlich.
- Das Benutzungs- und Architekturkonzept entspricht unseren Erwartungen.

Projektmanagement

- Es sind keine unangenehmen Überraschungen passiert.
- Unsere Prioritäten wurden bei der Planung entsprechend berücksichtigt.
- Wir haben regelmässig Updates über den Fortschritt der Arbeit erhalten.

Qualität

- Das entstandene MVP wurde von uns getestet und funktioniert soweit stabil.

A.1.3 Risikotabelle

				Wahrscheinlichkeit											
ID	Beschreibung	Schaden (Stunden)	Massnahme	W3	W4		W5		W6		W7		W8		W9		W10	
1	Einarbeitung dauert länger als erwartet	60	Hilfe von Zühlke oder HSR-Assistenten anfordern	100,00%	4,00	80,00%	3,20	50,00%	2,00	50,00%	2,00	50,00%	2,00	40,00%	1,60	30,00%	1,20	30,00%	1,20
2	Angular: Auftreten von Problemen mit der Technologie	30	Hilfe von HSR-Assistenten anfordern	50,00%	1,00	50,00%	1,00	70,00%	1,40	60,00%	1,20	60,00%	1,20	60,00%	1,20	50,00%	1,00	50,00%	1,00
3	Lösungsideen mit Angular nicht umsetzbar	30	Evaluation von alternativer Technologie und Abklärung mit Zühlke	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10
4	Lösungsideen mit Filesystem nicht umsetzbar	45	Evaluation von alternativer Technologie und Abklärung mit Zühlke	8,00%	0,24	8,00%	0,24	8,00%	0,24	8,00%	0,24	8,00%	0,24	8,00%	0,24	8,00%	0,24	8,00%	0,24
5	Anforderungen werden nicht richtig verstanden	30	Notfallsitzung mit allen Beteiligten	90,00%	1,80	80,00%	1,60	60,00%	1,20	70,00%	1,40	60,00%	1,20	50,00%	1,00	40,00%	0,80	40,00%	0,80
6	Personenausfall	0	Aufarbeiten	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00
7	Kommunikationsprobleme mit Zühlke (Emailantwort dauert länger)	30	Notfallsitzung mit allen Beteiligten	50,00%	1,00	50,00%	1,00	40,00%	0,80	60,00%	1,20	50,00%	1,00	50,00%	1,00	50,00%	1,00	50,00%	1,00
8	Laptopverlust	8	Projekt neu aufsetzten	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03
9	Spezifikation wir nachträglich geändert	30	Gravierende Änderung an Projektbasis umsetzten	20,00%	0,40	20,00%	0,40	25,00%	0,50	40,00%	0,80	30,00%	0,60	30,00%	0,60	30,00%	0,60	30,00%	0,60
10	Keine vernünftigen Testdaten	15	Daten selbst erfinden		0,00		0,00		0,00	90,00%	0,90	80,00%	0,80	70,00%	0,70	60,00%	0,60	60,00%	0,60
	Summe	**278**			**8,6**		**7,6**		**6,3**		**7,9**		**7,2**		**6,5**		**5,6**		**5,6**

ID	Beschreibung	Schaden (Stunden)	Massnahme	W11		W12		W13		W14		W15		W16		W17	
1	Einarbeitung dauert länger als erwartet	60	Hilfe von Zühlke oder HSR-Assistenten anfordern	30,00%	1,20	20,00%	0,80	20,00%	0,80	20,00%	0,80	10,00%	0,40	5,00%	0,20	1,00%	0,04
2	Angular: Auftreten von Problemen mit der Technologie	30	Hilfe von HSR-Assistenten anfordern	50,00%	1,00	30,00%	0,60	30,00%	0,60	30,00%	0,60	20,00%	0,40	10,00%	0,20	5,00%	0,10
3	Lösungsideen mit Angular nicht umsetzbar	30	Evaluation von alternativer Technologie und Abklärung mit Zühlke	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10	5,00%	0,10
4	Lösungsideen mit Filesystem nicht umsetzbar	45	Evaluation von alternativer Technologie und Abklärung mit Zühlke	8,00%	0,24	5,00%	0,15	5,00%	0,15	5,00%	0,15	5,00%	0,15	3,00%	0,09	1,00%	0,03
5	Anforderungen werden nicht richtig verstanden	30	Notfallsitzung mit allen Beteiligten	40,00%	0,80	30,00%	0,60	30,00%	0,60	30,00%	0,60	20,00%	0,40	10,00%	0,20	5,00%	0,10
6	Personenausfall	0	Aufarbeiten	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00	10,00%	0,00
7	Kommunikationsprobleme mit Zühlke (Emailantwort dauert länger)	30	Notfallsitzung mit allen Beteiligten	50,00%	1,00	20,00%	0,40	20,00%	0,40	20,00%	0,40	10,00%	0,20	10,00%	0,20	10,00%	0,20
8	Laptopverlust	8	Projekt neu aufsetzten	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03	5,00%	0,03
9	Spezifikation wir nachträglich geändert	30	Gravierende Änderung an Projektbasis umsetzten	30,00%	0,60	20,00%	0,40	20,00%	0,40	20,00%	0,40	10,00%	0,20	5,00%	0,10	1,00%	0,02
10	Keine vernünftigen Testdaten	15	Daten selbst erfinden	60,00%	0,60	50,00%	0,50	50,00%	0,50	50,00%	0,50	30,00%	0,30	20,00%	0,20	10,00%	0,10
	Summe	**278**			**5,6**		**3,6**		**3,6**		**3,6**		**2,2**		**1,3**		**0,7**