HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

Hochschule für Technik Rapperswil

Bachelorarbeit

Abteilung Informatik

Solution Strategy mit der Methode 635 als Cross Plattform App

Autoren:
Elias Brunner & Oliver Dias

Betreuer:
Prof. Dr. Olaf Zimmermann

11. Juni 2019

Aufgabenstellung

Die komplette Aufgabenstellung wurde von Prof. Dr. Olaf Zimmermann erstellt und kann dem Anhang A entnommen werden.

Ausgangslage

Die papiergestützte Methode 635 ist eine Kreativitäts- und Brainwriting-Technik, für die es bisher noch keine Unterstützung in mobilen Apps gab. Eine Studienarbeit im Herbstsemester 2018/2019 konzipierte und implementierte daher einen ersten Prototyp einer SmartPhone App für die Methode 635; Cross Platform Support (Android- und iOS) wurde durch Verwendung von Xamarin erreicht. Ein Vorteil einer derartigen mobilen Anwendung ist, dass Anwender die Methode 635 nutzen können, wenn sie sich nicht ständig an einem Ort befinden.

In dieser Bachelorarbeit sollen weitere Features implementiert werden auf Basis der Testergebnisse für den bestehenden Prototypen. Im Fokus steht dabei insbesondere die Software Engineering Aktivität 'Solution Strategy' aus dem Buch „Effektive Softwarearchitekturen“ von G. Starke, die in der HSR-Vorlesung Application Architecture behandelt wird. Diese Aktivität erfordert ein hohes Mass an kollaborativen Problemlösungsprozessen auf unterschiedlichen Abstraktionsebenen (z.B. Konzepte, Technologien, und Produkte).

Ziele der Arbeit und Liefergegenstände

Die Vision der Arbeit ist es, die Papierversion für diese Methodik zu funktional und qualitativ zu überbieten. Damit die erweiterte App einen Mehrwert gegenüber der Papierversion bietet, soll es z.B. möglich sein, verschiedene Medien (Text, Video, Bilder, etc.) zu verwenden bzw. einzubinden. Hardware-Features und andere Apps sollen – wenn sinnvoll und technisch möglich – integriert werden.

Bei der Umsetzung spielen Erfolgsfaktoren wie einfache und intuitive Bedienung der App und ein unkompliziertes Reporting sowie Robustheit und Stabilität (Bsp. keine Zeit- und Datenverluste) eine wichtige Rolle. Ein wichtiger Input für die Bachelorarbeit ist das externe Feedback zu den Ergebnissen der vorangegangenen Studienarbeit, zum Beispiel Feedback zur Usability und zur Codequalität.

Weitere kritische Erfolgsfaktoren sind:

- Konfigurierbarkeit und Erweiterbarkeit (im Hinblick auf Folgearbeiten, die u.U. auch andere Brainstorming- und Innovationsmethoden wie z.B. Design Thinking unterstützen).
- Sinnvolle Ausnutzung der Smartphone-Fähigkeiten, um einen Mehrwert im Vergleich zur traditionellen, papiergestützten Methode zu erreichen.
- Validierung der Konzepte und ihrer Implementierung mit Hilfe von User Tests in mindestens einem Anwendungsbereich (Bsp. Architekturentscheidungen und -optionen).

Abstract

Die papiergestützte Methode 635 ist eine Kreativitäts- und Brainwriting-Technik, welche die Entwicklung von neuen, ungewöhnlichen Ideen für Problemlösungen in der Gruppe fördert. Nach unserem Wissensstand existiert für die Methode 635 noch keine mobile Applikation.

Aus der vorliegenden Bachelorarbeit ist eine Cross-Plattform Applikation für iOS und Android hervorgegangen. Diese Applikation erlaubt es Benutzern, die Methode 635 gemeinsam auf ihrem Smartphones oder Tablets zu verwenden. Zusätzlich zur Eingabe von Ideen via Text ist es möglich, Skizzen als Lösungsvorschlag zu zeichnen oder aus einer Liste von vordefinierten Microservice API Patterns (MAP) ein passendes Pattern auszuwählen. Erarbeitete Brainstormings können im Markdown-Format exportiert werden. Die Möglichkeit, Skizzen zu zeichnen und Patterns aus der vordefinierten Patternsprache zu verwenden, kommt gerade (aber nicht ausschliesslich) Software-Architekten zugute. So können sie diese Funktionen beispielsweise im Zuge der Entwurfsphase Solution Strategy nutzen, um kollaborativ und kreativ an einer allfälligen Lösung zu arbeiten.

Als Teil der Dokumentation beschreiben wir zudem die Herausforderungen, die während der Umsetzung aufgetreten sind. So können andere Entwickler in ähnlichen Cross-Plattform-Vorhaben von unseren Erfahrungen profitieren und die Projektrisiken verringern.

Als Ausblick für die vorliegende Arbeit wäre es denkbar, weitere Ideen-Typen zu integrieren oder den Fokus etwas zu weiten und andere Brainstorming-Methoden oder Konzepte wie z.B. ein Mitglied der World-Cafe-Methodenfamilie zu integrieren.

Inhaltsverzeichnis

1 Management Summary

1.1 Ausgangslage

Die papiergestützte Methode 635 ist eine Kreativitäts- und Brainwriting-Technik, für die es bisher noch keine Unterstützung in Form von mobilen Apps gab. Eine Studienarbeit im Herbstsemester 2018/2019 konzipierte und implementierte daher einen ersten Prototypen einer Smartphone App für die Methode 635. Durch die Verwendung von Xamarin konnte sowohl die Unterstützung für iOS wie auch für Android Smartphones ermöglicht werden.

1.2 Vorgehen

Für diese Arbeit lässt sich unser Vorgehen in drei Phasen einteilen. In der ersten Phase fokussierten wir uns primär auf Recherchearbeiten und konzeptionelle Arbeiten. Dabei führten wir eine Vorstudie durch und beschäftigten uns mit der Frage, was die Solution Strategy ist und wie sich diese in das bestehende Projekt einbetten lässt. Weiter überlegten wir uns, wie Bilder in die bereits verwendete dokumentorientierte Datenbank abgelegt werden können. Da Software-Architekten häufig Skizzen für z.B. Designentwürfe anfertigen, stellte dies ein grundlegender Bestandteil der geplanten Funktionen dar. Um die bestehende Applikation für weitere Ideen-Typen zu erweitern, investierten wir einige Zeit, ein entsprechendes Konzept dafür zu entwickeln.

In der zweiten Phase unterzogen wir sowohl das Backend wie auch die Cross-Plattform App einem umfangreichen Refactoring. Dies sollte uns helfen den bestehenden Code, welcher in der vorhergegangenen Studienarbeit geschrieben wurde, zu vereinfachen und dahingehend anzupassen, um ihn für die geplanten Funktionen verwenden zu können.

In der dritten und letzten Phase ging es nun darum, die geplanten Funktionen und Erweiterungen in Form von Code zu implementieren. Um die Machbarkeit jener Funktionen festzustellen, wurden vereinzelt Prototypen dafür geschrieben. Da sich diese wie gewünscht umsetzen liessen, konnten die Funktionen in die bestehende Applikation integriert werden.

Um sicherzustellen, dass das gesamte System wie gewünscht funktionierte, führten wir, zusätzlich zum manuellen Testen während der Entwicklung, gegen Ende der Arbeit mehrere User-Tests durch.

1.3 Ergebnisse

Aus der vorliegenden Bachelorarbeit ist eine solide und performante Cross-Plattform Applikation für Android und iOS hervorgegangen. Diese ermöglicht es Benutzern die Methode 635 auf dem eigenen Smartphone oder Tablet durchzuführen.

Die in der Vorstudie gemachten Überlegungen in Bezug auf die Solution Strategy verhalfen uns zu einem Konzept (siehe Tabelle 1), welches die Gestaltung des Lösungsfindungsprozess in unserer Applikation beschreibt. Dabei stellte sicher heraus, dass sich die Methode 635 gut für die Vorschläge und Tipps der Solution Strategy eignet.

Zusätzlich zu den bestehenden Funktionen aus der Studienarbeit schufen wir mit dieser Arbeit die Möglichkeit, Skizzen als Lösungsvorschlag zu zeichnen, aus einer Liste von Patterns ein passendes Pattern auszuwählen sowie das erarbeitete Brainstorming als

Markdown zu exportieren. Die Möglichkeit Skizzen zu zeichnen und Patterns aus der vordefinierten Patternsprache (MAP) zu verwenden, kommt gerade (aber nicht ausschliesslich) Software-Architekten zugute. So können sie diese Funktionen im Zuge der kreativen (Design-)Entwurfsphase nutzen, um kollaborativ an einer allfälligen Lösung zu arbeiten.

Dank des Code-Refactorings zu Beginn der Bachelorarbeit konnte sowohl die Performance wie auch die Stabilität deutlich verbessert werden. Aus Rückmeldungen von User-Tests konnten ausserdem Verbesserungen hinsichtlich Userführung und UI-Design durchgeführt werden.

2 Technischer Bericht

2.1 Einleitung und Übersicht

Da die vorliegende Bachelorarbeit auf der Studienarbeit "Methode 635 als Cross Plattform App mit Xamarin" [1] aufbaut, wird an einzelnen Stellen in diesem technischen Bericht auf die Studienarbeit verwiesen. Es wird davon ausgegangen, dass dem Leser Begriffe wie "Methode 635" oder "Xamarin", sowie weitere softwarebezogene Ausdrücke bereits geläufig sind. Ist dies nicht der Fall, können allfällige Wissenslücken in diesen Bereichen in der erwähnten Studienarbeit nachgelesen werden.

Der technische Bericht selbst besteht neben diesem Kapitel noch aus weiteren Unterkapiteln.

Dem Leser soll zunächst nochmals ein kurzer Überblick gegeben werden, was am Ende der Studienarbeit (Kap. 2.2) erreicht wurde und was der Stand der Entwicklung war, mit welcher diese Arbeit gestartet werden konnte. Wie schon in der Studienarbeit führten wir auch bei der Bachelorarbeit zunächst eine Vorstudie durch, in der wir grundsätzliche Analysen durchführten und Fragen klärten. Dabei ging es vor allem um die Frage, wie Dateien in einer Datenbank gespeichert werden können. All dies ist im Kapitel "Vorstudie" (Kap. 2.3) niedergeschrieben. Im Kapitel der "Anforderungsspezifikation" (Kap. 2.4) listen wir alle funktionalen sowie nicht-funktionalen Anforderungen auf, welche neu dazu kamen. Das Domain-Modell findet sich im Kapitel "Domainanalyse" (Kap. 2.5). Die "Architekturdokumentation" (Kap. 2.6) ist das nächste Kapitel und dokumentiert die logische Architektur, sowie das Deployment. Jegliche Entscheidungen, welche wir in Bezug auf die Architektur getroffen haben, werden im Kapitel "Architekturentscheide" (Kap. 2.7) begründet. Während der gesamten Umsetzung der Applikation sind natürlich auch Probleme aufgetreten. Diese haben wir im Kapitel "Herausforderungen" (Kap. 2.8.8) zusammengetragen. Die eigentliche Implementation haben wir im Kapitel "Ergebnisse" (Kap. 2.8) dokumentiert. Zum Schluss blicken wir im Kapitel "Schlussfolgerungen" (Kap. 2.9) nochmals kritisch auf unser Projekt zurück und bewerten unsere Arbeit und geben einen Ausblick, wie man die Applikation noch erweitern könnte.

2.2 Ergebnis der Studienarbeit

Aus der vorherigen Studienarbeit ist eine lauffähige Applikation hervorgegangen, welche für verschiedene Brainstormings verwendbar ist. Die einzelnen Durchführungen können zudem in Punkto Teilnehmeranzahl, Rundenanzahl und Rundenzeit konfiguriert werden. Ein durchgeführter Test ergab allerdings, dass die Applikation in Sachen Stabilität und Benutzerführung noch Potenzial für Verbesserungen aufwies.

Wegen der beschränken Zeit von 14 Wochen für die Studienarbeit konnte zudem die Eingabe von neuen Ideen nur via Text umgesetzt werden (nicht wie zunächst angedacht auch mit Bilder, Skizzen, etc.).

Das Ziel der Studienarbeit war es einen lauffähigen Prototypen zu entwickeln. Unser Fokus stand daher primär auf der Umsetzung jenes Prototypen. Kombiniert mit der beschränkten Zeit führte dies dazu, dass der Code gegen Ende des Projektes immer umfangreicher und unübersichtlicher wurde. Um diesem Umstand in einer allfälligen Folgearbeit

entgegenzuwirken, entwickelten wir noch in der Studienarbeit entsprechende Konzepte und Ideen.

Weiterführende Informationen zur Ergebnisbewertung der Studienarbeit kann im gleichnamigen Kapitel jener Arbeit [1] nachgelesen werden.

2.3 Vorstudie

Dieser Abschnitt dokumentiert die Vorarbeiten in verschiedenen Bereichen, die für das Abwickeln dieses Projektes relevant sein können. Dabei werden die entsprechenden Themen analysiert und es wird abgewägt, inwiefern diese für den Projekterfolg von Nutzen sein können. Stellt sich heraus, dass eines der analysierten Themen sinnvoll und machbar ist, wird es in das Projekt integriert.

2.3.1 Solution Strategy

In der von Gernot Starke entwickelten Vorlage zur Software Architektur Dokumentation arc42 [2] sowie in seinem Buch *Effektive Softwarearchitekturen* [3], befindet sich ein für uns besonders interessanten Abschnitt namens *Solution Strategy*. Darin wird ein Vorschlag angeboten, wie man in der Synthese der Software Architektur die Ansätze zur Lösungsfindung erarbeitet und dokumentiert. Dieser Vorschlag besteht aus mehreren Tipps zum Inhalt, zur Darstellung und zur Entwicklung dieser Ansätze. Als Beispiel beinhaltet dieser Vorschlag folgenden Tipp: “Erkläre die Lösungsstrategie so kompakt wie möglich (z.B. als Liste von Schlüsselwörtern)“. Da sich dieses Projekt intensiv mit Lösungsfindungsmethoden auseinandersetzt, bietet die Solution Strategy nützlichen Input, wie der Lösungsfindungsprozess in unserer Applikation gestaltet werden könnte.

Eine mögliche Erweiterung unserer Applikation wäre, ein zusätzliches Modul für eine Lösungsfindungsvariante anzubieten. Dabei käme beim Erstellen eines BrainstormingFindings die Auswahl zwischen zwei Typen von Lösungsfindungen: “Software Architektur Lösung“ und “Generelle Lösung“. Bei der Software Lösung wären dann die Tipps von arc42 eingearbeitet, wobei eine Abbildung gemäss Tabelle 1 vom Kontext des arc42-Templates in den Applikationskontext denkbar wäre.

2.3.1.1 Fazit

Durch die verschiedenen geplanten Erweiterungen bezüglich den Input-Varianten (siehe Use Case 8 im Kapitel 2.4.1 Funktionale Anforderungen) lässt sich diese Vorlage sinnvoll in die Applikation integrieren. Dies erfordert allerdings eine erweiterbare Grundlage, um das erwähnte Modul (“Software Architektur Lösung“) zu implementieren. Durch eine saubere Abstraktion sollte die Architektur der Applikation aber auch andere Lösungstypen unterstützen.

[1] Buch Effektive Softwarearchitekturen

arc42 & eswa[1]-Kontext	**Applikationskontext**
Lösungsansatz finden	Aufgabe von Benutzer, Kreativität ist von Benutzer gefragt
Lösungsansatz so kompakt und genau wie möglich erklären (z.B. als Liste von Keywords)	Liste von Keywords mittels Textinput
Lösungsansatz als Tabelle beschreiben	Tabellarische Darstellung mithilfe der Skizzenfunktion
Lösungsansatz im Kontext der Qualtätsattribute beschreiben	Bezugnahme auf Qualitätsattribute mittels Textinput, oder Auswahlliste mit definierten Qualitätsattribute anbieten
Konzepte, Views oder Code referenzieren	Liste von definierten (Microservices-API-)Patterns[4] als Input anbieten (z.B. *Frontend Integration*, mit Icon und kurzem Beschrieb des Patterns)
Lösungsansatz inkrementell und iterativ wachsen lassen	Durch rundenbasiertes Brainwriting gegeben.
Lösungsansatz rechtfertigen, vergleichen, entscheiden und begründen	Teil der nachträglichen Diskussion, nicht Teil der Applikation (Wertung und definitive Entscheide sollten nicht in Brainwriting einfliessen), Exportfunktion

Tabelle 1: Abbildung der Solution Strategy gemäss arc42 und eswa auf Brainstorming Applikation

2.3.2 Dateien in Datenbank speichern

Da der wesentliche Bestandteil dieser Bachelorarbeit darin besteht, die bestehende Applikation um verschiedene Medien wie Bilder, Skizzen, Videos oder Links etc. zu erweitern, mussten wir uns zuerst klar werden, wie wir dies erreichen könnten.

Da wir uns während der Studienarbeit [1] für MongoDB als NoSQL Datenbank entschieden haben, sahen wir es als sinnvoll an, wieder auf diese Technologie zu setzen, zumal wir auch gute Erfahrungen damit gemacht hatten.

Mit GridFS [5][6] fanden wir eine Spezifikation, welche es ermöglicht, verschiedenste Dateien in eine MongoDB zu speichern oder von dort wieder herzustellen. Einer der Vorteile von GridFS liegt darin, dass auch grosse Dateien (GB an Daten) in der Datenbank abgelegt werden können. Dabei zerlegt GridFS eine solche Datei in kleinere Teile sogenannte Chunks und speichert jedes dieser Chunks in einem eigenen Document ab [5].

GridFS kann daher nicht nur jegliche Art von Dateien speichern, welche die Standard-Documentgrösse von 16MB übersteigen, sondern kann diese auch wieder zur Verfügung stellen ohne dass die komplette Datei in den Memory geladen werden muss. GridFS bietet somit die Möglichkeit von Streaming [7].

Ausserdem limitiert GridFS nicht die Anzahl an Dateien, welche gespeichert werden können, wie es teilweise bei Filesystemen der Fall ist.

Zudem können mit GridFS auch nur Teile einer Datei zur Verfügung gestellt werden. So kann z.B. bei einem Video mittels Range Queries bis zur Mitte "gesprungen" werden ohne das die erste Hälfte (herunter)geladen werden muss [5].

GridFS ist laut docs.mongodb.com allerdings nicht geeignet, falls man den Inhalt einer Datei atomar updaten möchte oder muss. In solch einem Fall muss man die aktualisierte Datei erneut in die Datenbank speichern und als neue Version ablegen. Des Weiteren sollten keine Dateien, welche kleiner als 16MB sind als einzelne Chunks gespeichert werden. Stattdessen wird empfohlen, diese wie gewohnt in einem Document abzuspeichern.

Die Implementation von GridFS wurde unter der MIT Lizenz veröffentlicht.

2.3.2.1 Fazit

Mit GridFS steht uns eine Spezifikation zur Verfügung, welche nicht nur alle unsere Bedürfnisse abdeckt sondern auch die Möglichkeit von Streaming bietet, was der Performance unserer Applikation hinsichtlich der Videofunktionalität zugutekommen könnte. Der ausschlaggebende Punkt warum wir uns zum Schluss aber für GridFS entschieden haben, war die Tatsache, dass diese Spezifikation schon in unserem verwendeten Treiber implementiert ist. So konnten wir mit sehr wenig Aufwand GridFS für unser Projekt verwenden.

2.3.3 Code Review durch Senior Entwickler

Der Quellcode der gesamten Applikation, welcher im Umfang der Studienarbeit entstand, wurde durch Silvan Gehrig, ein wissenschaftlicher Mitarbeiter des Instituts für Software, geprüft. Dadurch konnten einige Mängel und Punkte zur Verbesserung identifiziert werden. Die wichtigsten Kritikpunkte sind nachfolgend aufgelistet und erläutert.

Der komplette Bericht von Silvan Gehrig kann im Anhang D nachgelesen werden.

2.3.3.1 Server

Auf dem Backend wurden folgende Punkte bemängelt:

Unsauberes Layering Controller greift direkt auf Datenbank zu.

Keine DTOs vorhanden Controller arbeiten teilweise direkt mit JSON Nodes anstatt mit DTOs.

Technologie zielgerichtet einsetzen Zum Teil zu lange Controller-Klassen, des Weiteren existieren keine Unit Tests

2.3.3.2 Client

Clientseitig gilt es folgende Punkte zu verbessern:

UI-Strings nicht in Resource Files Besser als die UI-Strings im ViewModel zu definieren ist das Verwenden von Resource Manifests.

Layering Business Services sind im ViewModel implementiert, anstelle von separaten Business Logik Klassen. Zum Teil wird direkt vom ViewModel auf den DAL zugegriffen.

Technologie zielgerichtet einsetzen Separate Projekte pro Layer, Unit Tests fehlen.

2.3.3.3 Fazit

Da zu Beginn der Construction Phase (siehe B.5) zwei Sprints für die Überarbeitung und Verbesserung des bestehenden Codes geplant sind, werden diese und weitere Punkte in dieser Zeit verbessert. Dadurch erhoffen wir uns eine ausgereiftere Plattform, für die sich zukünftige Features effizient und einfach entwickeln lassen.

2.4 Anforderungsspezifikation

In diesem Kapitel werden die Ergebnisse der Anforderungsanalyse in Form von diversen Diagrammen und Tabellen festgehalten. Ziel ist es, nötige Einschränkungen für das System zu definieren.

2.4.1 Funktionale Anforderungen

Aufgrund der Fortsetzung der Arbeit existieren im Bezug auf die Anforderungen Überlappungen bei allen Use Cases, die für den wesentlichen Ablauf der Applikation nötig sind. Im Folgenden werden die Gemeinsamkeiten und Unterschiede beschrieben und detailliert auf Neuheit eingegangen.

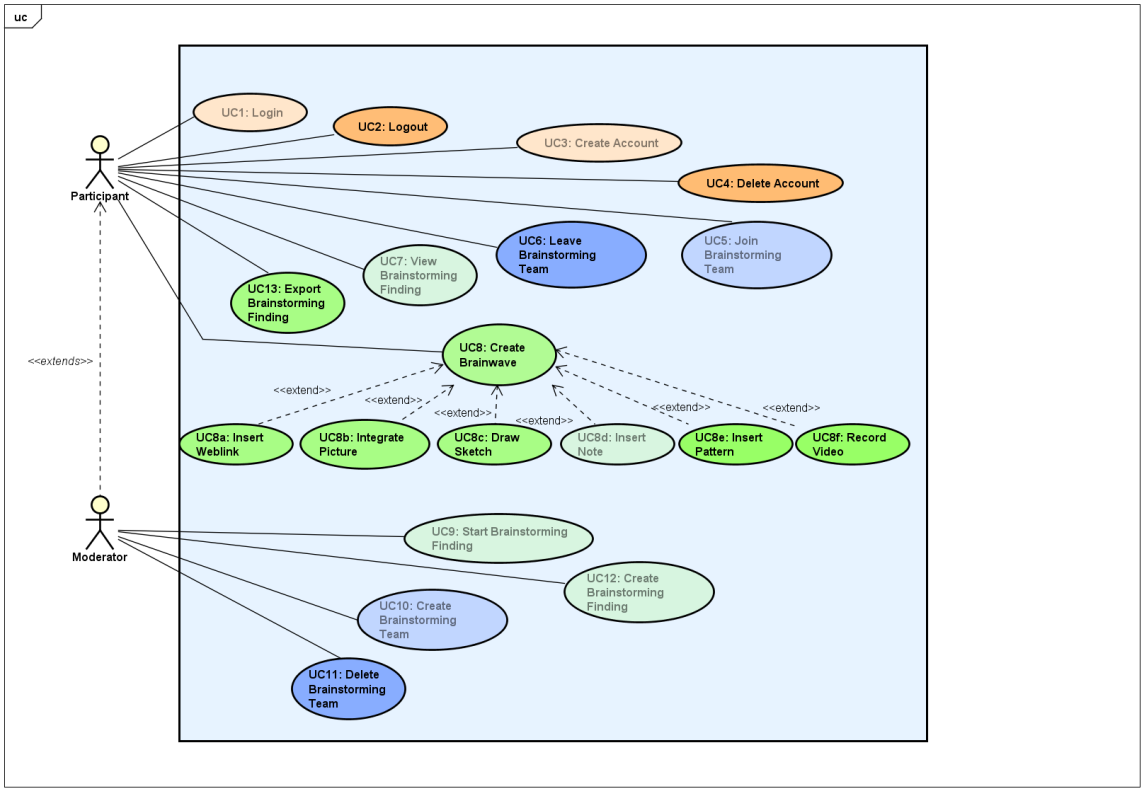

Abbildung 1: Use-Case Diagramm Methode 635

Abbildung 1 zeigt das Use-Case Diagramm für diese Arbeit. Es gibt einige Abweichungen zur Studienarbeit. Im Diagramm sind 8 Use-Cases in blasser Farbe gestaltet, bei diesen handelt es sich um bereits implementierte Features im Umfang der Studienarbeit.

Wichtig zu erwähnen ist dabei, dass UC12: Create Brainstorming Finding erst im Nachhinein hinzugekommen ist. Dies ist so, weil zu Beginn der Entwicklung wir die Annahme trafen, dass für jedes Team genau ein Brainstorming entsteht und zwar zum Zeitpunkt des Starts des Brainstorming Findings (UC9). Wie wir aber im Verlaufe der Zeit feststellten, war die Umsetzung mit mehreren Brainstorming Findings pro Team innert vertretbarer

Zeit möglich. So entstand ein zusätzlicher Use-Case, der nicht mehr implizit mit UC9: Start Brainstorming Finding ausgeführt wurde.

Weiter sind Use-Cases in satter Farbe erkennbar. Bei diesen handelt es sich um diejenigen, welche in dieser Arbeit im Fokus liegen. Komplett neu hinzugekommen sind UC8e: Insert Pattern, UC8f: Insert Video und UC13: Export BrainstormingFinding. Als zentraler Use-Case wird zudem UC8c: Draw Sketch angesehen. Im folgenden Kapitel werden nun die in dieser Arbeit relevanten Use-Cases kurz beschrieben. Im Kapitel 2.4.1.2 sind alle Sub-Use-Cases von UC8: Create Brainwave im fully-dressed Stil detailliert erläutert.

2.4.1.1 Brief Use-Cases

Die für diese Arbeit relevanten Use-Cases sind hier beschrieben.

***UC2:* Logout**	Als eingeloggter Participant will ich mich ausloggen, sodass das Startfenster wieder erscheint.
***UC6:* Leave Brainstorming Team**	Als Participant will ich ein beigetretenes Team verlassen können.
***UC8:* Create Brainwave**	Als Participant will ich während einer Brainstorming Session ein Brainwave (bestehend aus mehreren Ideen) erstellen und einreichen können.
***UC8a:* Insert Weblink**	Als Participant will ich einen Weblink in mein aktuelles Sheet einfügen können.
***UC8b:* Integrate Picture**	Als Participant will ich ein Bild in mein aktuelles Sheet integrieren können.
***UC8c:* Draw Sketch**	Als Participant will ich in mein aktuelles Sheet zeichnen können.
***UC8e:* Insert Pattern**	Als Participant will ich aus Vorlagen software-relevante Patterns in mein aktuelles Sheet einfügen können.
***UC8f:* Record Video**	Als Participant will ich ein Video aufnehmen und in mein aktuelles Sheet einfügen können.

***UC11:* Delete Brainstorming Team**	Als Moderator will ich ein Brainstorming Team löschen können.
***UC13:* Export Brainstorming Finding**	Als Participant will ich ein abgeschlossenes BrainstormingFinding exportieren können.

2.4.1.2 Fully-Dressed Use-Cases

Die fully-dressed Use-Cases folgen den im Modul *Software Engineering 1* des Bachelorstudiums Informatik der HSR empfohlenen Punkten.

Use-Case 8a: Insert Weblink	
Primary Actor	Participant
Stakeholders & Interests	Ein Participant wünscht als Teil seiner Brainwave einen Weblink zu teilen.
Preconditions	Participant existiert im System (UC3), ist eingeloggt (UC1) und einer Gruppe beigetreten (UC5). Des weiteren existiert mindestens ein BrainstormingFinding (UC12), welches schon gestartet ist (UC9).
Post Conditions/Success Guarantee	Der Weblink ist erfolgreich in der Brainwave gespeichert und kann in einer nächsten Runde von den anderen Participants angesehen werden.
Main Success Scenario/Basic Flow	1. Der Participant speichert sich den gewünschten Weblink in den Zwischenspeicher des Smartphones. 2. Der Participant fügt mittels Einfügen den Text in das Textfeld ein. 3. Der Participant speichert den Link mittels commit in seiner Brainwave ab.
Alternative Flows	-
Frequency of Occurrence	gelegentlich, erweiterte Kernfunktionalität.

Use-Case 8b: Integrate Picture	
Primary Actor	Participant
Stakeholders & Interests	Ein Participant wünscht als Teil seiner Brainwave ein Bild zu teilen.
Preconditions	Participant existiert im System (UC3), ist eingeloggt (UC1) und einer Gruppe beigetreten (UC5). Des weiteren existiert mindestens ein BrainstormingFinding (UC12), welches schon gestartet ist (U9).
Post Conditions/Success Guarantee	Das Bild ist erfolgreich in der Brainwave gespeichert und kann in einer nächsten Runde von den anderen Participants angesehen werden.
Main Success Scenario/Basic Flow	1. Der Participant klickt den Plus-Button (o.Ä) neben dem Text-Eingabefeld. 2. Das System zeigt ein Menü mit weiteren Ideentypen (unter anderem das Bild) an. 3. Der Participant klickt auf den Menüeintrag, um ein Bild aufzunehmen. 4. Das Smartphone öffnet die Kamerafunktion. 5. Der Participant macht ein Foto und speichert das Bild mit dem entsprechenden Button in seine Brainwave.
Alternative Flows	-
Frequency of Occurrence	gelegentlich, erweiterte Kernfunktionalität.

Use-Case 8c: Draw Sketch	
Primary Actor	Participant
Stakeholders & Interests	Ein Participant wünscht als Teil seiner Brainwave eine eigene Skizze zu zeichnen.

Preconditions	Participant existiert im System (UC3), ist eingeloggt (UC1) und einer Gruppe beigetreten (UC5). Des weiteren existiert mindestens ein BrainstormingFinding (UC12), welches schon gestartet ist (U9).
Post Conditions/Success Guarantee	Die gezeichnete Skizze ist erfolgreich in der Brainwave gespeichert und kann in einer nächsten Runde von den anderen Participants angesehen (aber nicht bearbeitet) werden.
Main Success Scenario/Basic Flow	1. Der Participant klickt den Plus-Button (o.Ä) neben dem Text-Eingabefeld. 2. Das System zeigt ein Menü mit weiteren Ideentypen (unter anderem die Skizze) an. 3. Der Participant klickt auf den Menüeintrag, um eine Skizze zu zeichnen. 4. Das Smartphone öffnet im App selbst eine entsprechende View, um Bilder zu zeichnen. 5. Ist der Participant zufrieden mit der Skizze, speichert diese mit dem entsprechenden Button in seine Brainwave.
Alternative Flows	-
Frequency of Occurrence	gelegentlich, erweiterte Kernfunktionalität.

Use-Case 8e: Insert Pattern	
Primary Actor	Participant
Stakeholders & Interests	Ein Participant wünscht als Teil seiner Brainwave einen bekanntes Pattern zu teilen.

Preconditions	Participant existiert im System (UC3), ist eingeloggt (UC1) und einer Gruppe beigetreten (UC5). Des weiteren existiert mindestens ein BrainstormingFinding (UC12), welches schon gestartet ist (U9) und vom Typ SoftwareLösung ist.
Post Conditions/Success Guarantee	Das Microservice-API-Pattern (Symbol und Name von MAP[4]) ist erfolgreich in der Brainwave gespeichert und kann in einer nächsten Runde von den anderen Participants angesehen und auf einer externen Webseite genauer begutachtet werden.
Main Success Scenario/Basic Flow	1. Der Participant klickt den Plus-Button (o.Ä) neben dem Text-Eingabefeld. 2. Das System zeigt ein Menü mit weiteren Ideentypen (unter anderem das Pattern) an. 3. Der Participant klickt auf den Menüeintrag, um eine Auswahl von verschiedenen Patterns zu erhalten. 4. Er wählt das passende Microservice-API-Pattern aus und speichert dieses mit dem entsprechenden Button in seine Brainwave.
Alternative Flows	-
Frequency of Occurrence	gelegentlich, erweiterte Kernfunktionalität.

Use-Case 8f: Record Video	
Primary Actor	Participant
Stakeholders & Interests	Ein Participant wünscht als Teil seiner Brainwave ein Video aufzunehmen.

Preconditions	Participant existiert im System (UC3), ist eingeloggt (UC1) und einer Gruppe beigetreten (UC5). Des weiteren existiert mindestens ein BrainstormingFinding (UC12), welches schon gestartet ist (U9). Die Länge des Videos wird auf wenige Sekunden limitiert, da dieses sonst länger werden kann als die zur Verfügung stehende Zeit.
Post Conditions/Success Guarantee	Das Video ist erfolgreich in der Brainwave gespeichert und kann in einer nächsten Runde von den anderen Participants angesehen werden.
Main Success Scenario/Basic Flow	1. Der Participant klickt den Plus-Button (o.Ä) neben dem Text-Eingabefeld. 2. Das System zeigt ein Menü mit weiteren Ideentypen (unter anderem das Video) an. 3. Der Participant klickt auf den Menüeintrag, um ein Video aufzunehmen. 4. Das Smartphone öffnet die Kamerafunktion. 5. Der Participant macht ein kurzes Video und speichert dieses mit dem entsprechenden Button in seine Brainwave.
Alternative Flows	-
Frequency of Occurrence	selten, erweiterte Kernfunktionalität.

Use-Case 13: Export BrainstormingFinding	
Primary Actor	Participant
Stakeholders & Interests	Ein Participant wünscht die ausgearbeiteten Ideen zu exportieren.

Preconditions	Participant existiert im System (UC3), ist eingeloggt (UC1) und einer Gruppe beigetreten (UC5). Des weiteren existiert mindestens ein BrainstormingFinding (UC12), welches schon abgeschlossen ist.
Post Conditions/Success Guarantee	Alle Ideen des ausgewählten BrainstormingFinding werden in einem passenden Format exportiert (vorstellbar wäre Markdown), sodass diese für andere Tätigkeiten weitergenutzt werden können.
Main Success Scenario/Basic Flow	1. Der Participant navigiert zum gewünschten BrainstormingFinding, welches er exportieren möchte. 2. Er klickt auf den Button export (o.Ä). 3. Das System exportiert alle eingereichten Ideen an einen Ort nach Wahl.
Alternative Flows	-
Frequency of Occurrence	gelegentlich, erweiterte Kernfunktionalität.

2.4.1.3 Sequenzdiagramm

Der Vollständigkeits halber haben wir in Abbildung 2 nochmals das erarbeitete Sequenzdiagramm unserer Studienarbeit [1] dargestellt. Darin ist der Prozess vom Erstellen des BrainstormingTeams bis zum Abschliessen der Brainwave modelliert.

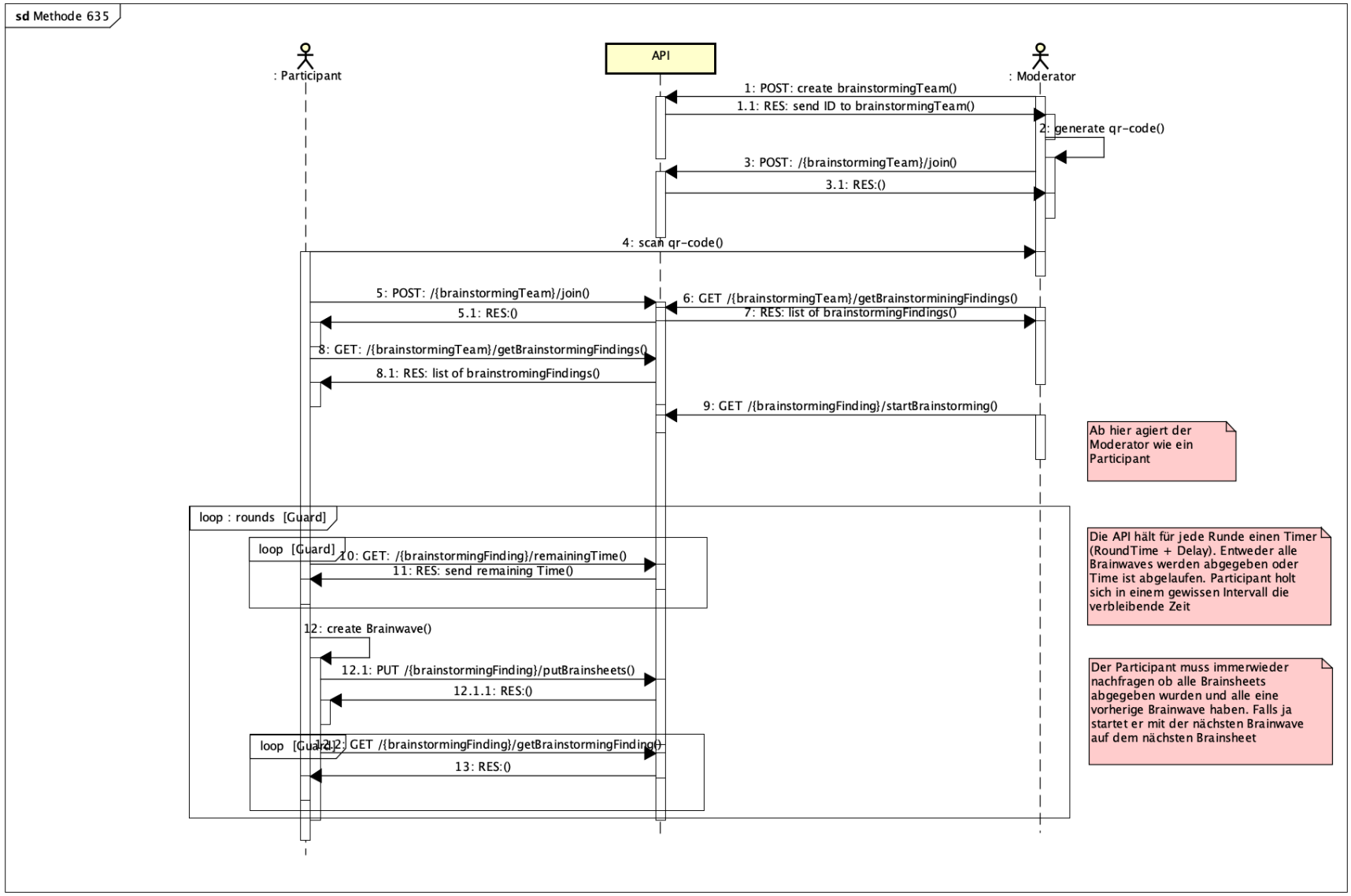

Abbildung 2: Ablauf der Kernlogik

Zusätzlich dazu haben wir in Abbildung 3 und 4, die aus unserer Sicht wichtigsten neuen Use-Cases (UC8c und UC8e) als Ablaufdiagramm gezeichnet und beschreiben, wo diese im Ablauf zum Tragen kommen.

UC8c Draw Sketch: Nachdem eine Brainwave gestartet ist (siehe 12 in Abbildung 2) und sich der Participant dazu entschlossen hat, eine Skizze zu zeichnen und damit zufrieden ist, speichert er diese in seine Brainwave. Dabei wird ein POST-Request an das API gesendet, welcher das gezeichnete Bild in der Datenbank speichert. Als Antwort wird eine ID an den Participant bzw. an die Applikation zurück geschickt. In einem nächsten Schritt wird wie bis anhin (siehe 12.1 in Abbildung 2) das gesamte Brainsheet, welches nun auch die ID in der entsprechenden SketchIdea gespeichert hat, an die API geschickt.

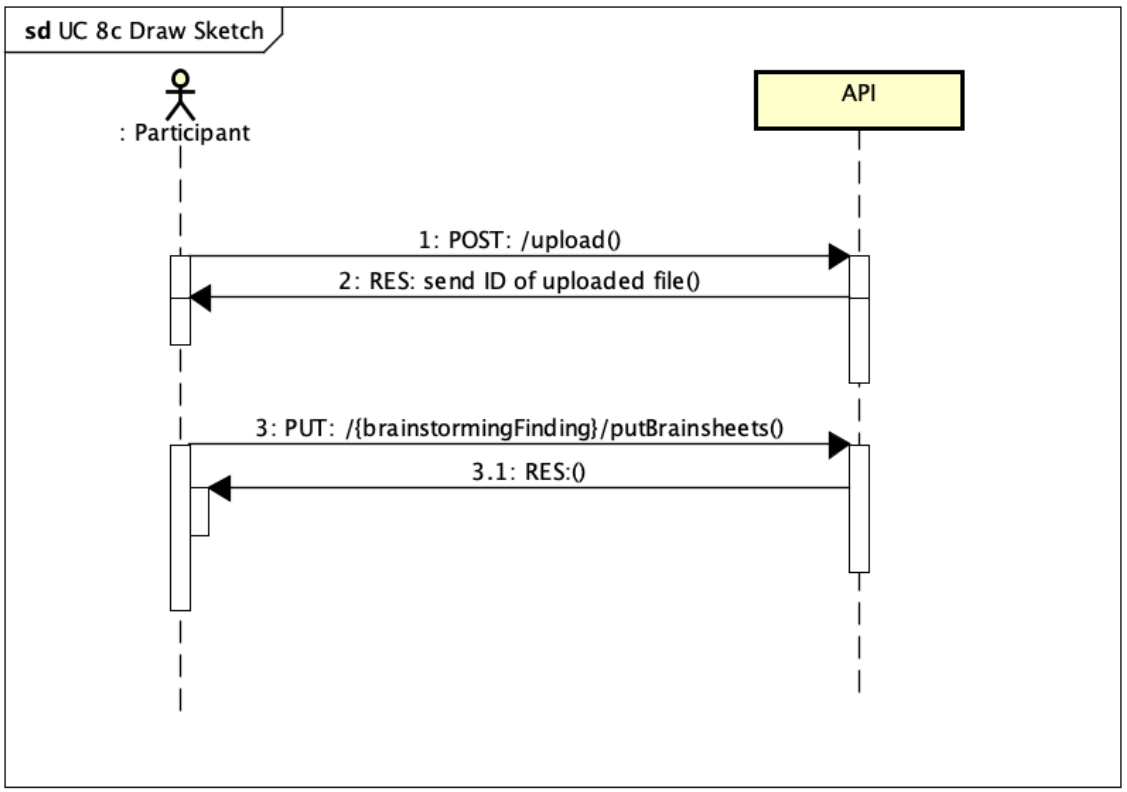

Abbildung 3: Ablauf beim Skizzieren

UC8e Insert Pattern: Nachdem eine Brainwave gestartet ist (siehe 12 in Abbildung 2) und sich der Participant dazu entschlossen hat, ein Pattern als Idee zu nutzen, klickt er auf das Plus-Zeichen neben der Texteingabe und wählt das Pattern als Ideentyp aus. Nun

wird ein GET-Request an das API gesendet, was als Antwort eine Liste aller verfügbaren Patterns zur Folge hat. Der Participant kann nun das gewünschte Pattern auswählen. Auch hier wird nun wieder, wie bis anhin (siehe 12.1 in Abbildung 2), das gesamte Brainsheet, welches nun auch das Pattern in der entsprechenden PatternIdea gespeichert hat, an die API geschickt.

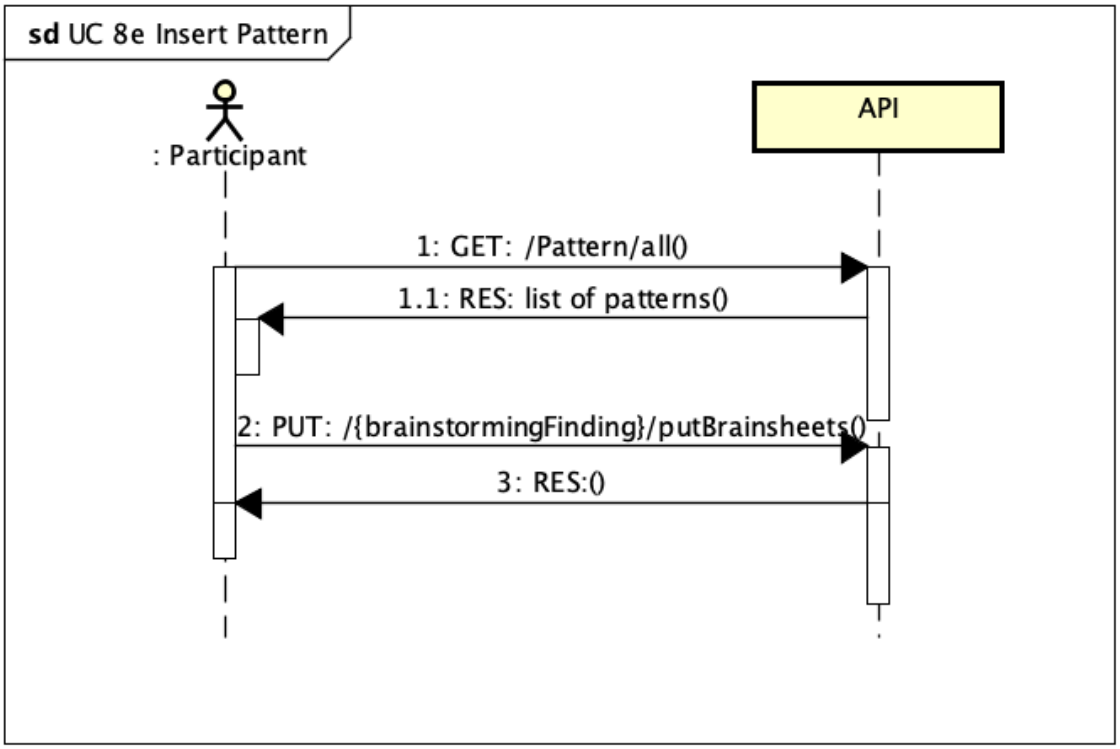

Abbildung 4: Ablauf beim Einfügen eines Patterns

2.4.2 Nicht-Funktionale Anforderungen

Wie schon in unserer Studienarbeit halten wir uns auch hier wieder an die Standards ISO 9126[8] bzw. dessen Nachfolger ISO 25010[9], welcher Gegensatz zum Standard ISO 9126 noch ausführlicher ist. Beide ISO-Normen liefern daher eine gute Checkliste für jegliche Art von nicht-funktionalen Anforderungen.

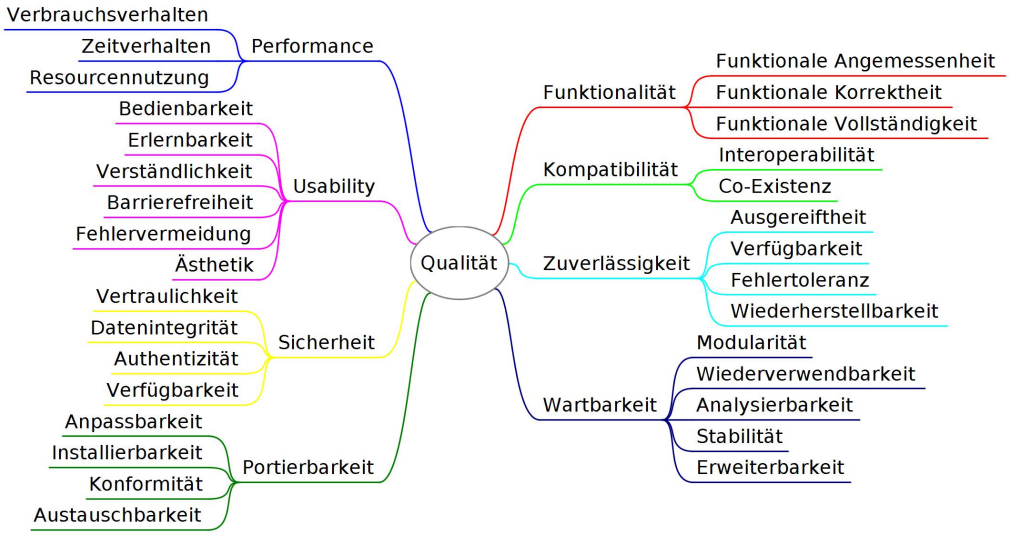

Abbildung 5: Anforderungskategorien nach ISO 25010 [10]

Im Gegensatz zur Studienarbeit wollen wir uns bei dieser Arbeit aber vor allem auf Faktoren wie Erweiterbarkeit und Modularität konzentrieren. Auch sollen Faktoren wie Zeitverhalten und Ästhetik stärker in den Vordergrund rücken.

Die bekannten nicht-funktionale Anforderungen aus der Studienarbeit bleiben allerdings weiter bestehen. Um genaue und erfüllbare nicht-funktionale Anforderungen zu definieren, müssen die SMART-Kriterien [11] erfüllt sein.

Um die SMART-Kriterien noch besser zu erfüllen, haben wir uns in dieser Arbeit dazu entschieden, sogenannte Landing Zones (gemäss HSR-Vorlesung: Application Architecture vom 25.09.2018) zu definieren. Diese erlauben es flexibler auf Ziele einzugehen und Toleranzen für akzeptable Werte festzusetzen.

Kriterium	Minimum	Gut	Übertroffen
Zeitverhalten Zeitkritische Kommunikation (Abgabe von Ideen/Brainsheets) zwischen Server und App beträgt	2 Sekunden	1 Sekunden	weniger als 1 Sekunde
Erweiterbarkeit Die Anzahl an neuen Ideenarten, welche mit der bestehenden Architektur als umsetzbar gelten, wird als ... angesehen	zu wenig (0-1)	ausreichend (1-5)	unbegrenzt (>5)
Modularität Die App ist innert ... Tagen um eine neue Ideenart erweitert	5 Tage	2 Tage	weniger als 1 Tag
Ästhetik Die Anziehungskraft gegenüber dem Endnutzer wird als ... charakterisiert	ausreichend für eine kritische Masse an Nutzern	nachhaltig	tägliche Nutzung in kritischen Projekten
Ausgereiftheit Der Grad der Ausgereiftheit oder Reife wird als ... angesehen	ungenügend für den produktiven Einsatz (Prototypen-Stadium)	genügend für den produktiven Einsatz, kann aber immer noch in Fehlerzustände gelangen	kaum Fehlerzustände und somit keine Abstürze der App

2.5 Domainanalyse

Das Domain-Modell besteht grob aus zwei Teilen: den Benutzern und der Brainstorming Methodik.

Dabei bilden mehrere *Participants* ein *BrainstormingTeam*. Diese wird von einem der *Participants*, dem *Moderator*, gegründet. Das Team hat die Möglichkeit, ein oder mehrere *BrainstormingFindings* zu erarbeiten. Dies entspricht einem gesamten Durchgang der Methode. Der *Moderator* erstellt diese und hat die Möglichkeit, die Anzahl von Ideen sowie die erste Rundenzeit zu konfigurieren. Jede weitere Runde wird um eine Minute verlängert.

Das *Brainsheet* entspricht einem physikalischem Blatt, das herumgegeben wird. In der Standardkonfiguration 635 existieren also 6 Sheets (weil 6 Teilnehmer dabei sind).

Eine *Brainwave* ist das Produkt jedes *Participants* am Ende einer Runde. Es gehört in ein *Brainsheet*, das jede Runde an den nächsten *Participant* weitergegeben wird. In der Standardkonfiguration besteht eine *Brainwave* aus 3 Ideen (6**3**5).

Die *Idea* ist ein effektiv erarbeiteter Teil einer *Brainwave*. Im Normalfall ist eine Idee simpler Text (*NoteIdea*), wobei weitere Typen von Ideen (Bild, Weblink und Zeichnung) durch das verwendete Design erdenklich sind. Speziell erwähnenswert ist der Umstand, dass im Gegensatz zur Studienarbeit, neu die Ideentypen *SketchIdea* und *PatternIdea* angedacht sind.

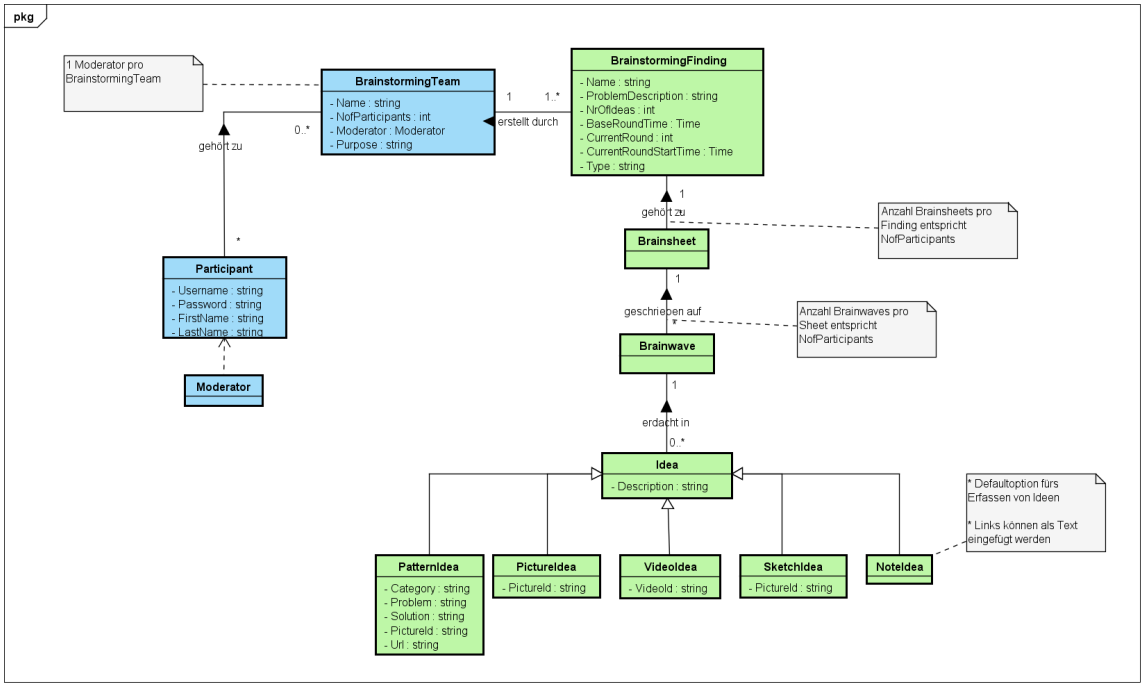

Abbildung 6: Domain Modell BrainingOutOfBox

2.6 Architekturdokumentation

In diesem Kapitel gehen wir detailliert auf die Architektur und das Deployment unseres Projektes ein. Es ist allerdings zu vermerken, dass dieses Kapitel grösstenteils aus der Studienarbeit [1] hervorgegangen ist. Es soll hier dennnoch der Vollständigkeit halber aufgeführt werden.

2.6.1 Logische Architektur

Wir teilen die Architektur des gesamten Systems in drei Schichten auf. In der Abbildung 7 sind diese als Presentation-, Businesslogic- und Persistence-Schicht zu erkennen.

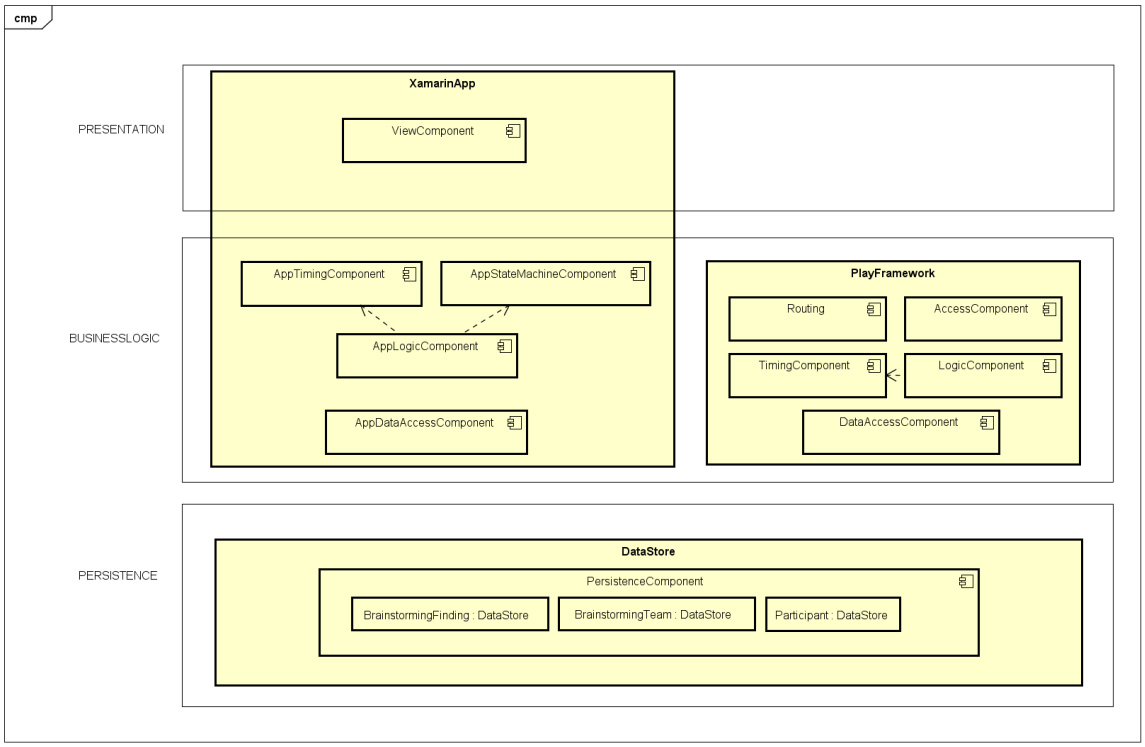

Abbildung 7: Logische Architektur BrainingOutOfBox

Die Präsentationsschicht ist die Schicht über die der Benutzer mit der Xamarin App kommuniziert. Konkreter gesagt, umfasst diese die verschiedenen View-Komponenten, welche für das Aussehen der App verantwortlich sind. Jegliche Interaktionen über die Oberfläche werden anschliessend in der Schicht der Businesslogic weiter verarbeitet. In dieser Schicht haben wir wieder unsere Xamarin App, welche selbst Logik-Komponenten wie die Timing-Komponente oder weitere App spezifische Logik-Komponenten enthält.

Die AppLogic-Komponente ist für das korrekte Verarbeiten und Weiterleiten der Eingaben an das PlayFramework verantwortlich.

Die AppTiming-Komponente ist zuständig für das Zeitmanagement während dem Brainstorming. Diese Komponente überwacht daher die noch verbleibende Zeit.

Die AppStateMachine-Komponente ist für den Runden-Wechsel-Mechanismus zuständig.

Um die Inhalte des Brainstormings sowie die Benutzerdaten an das Backend zu übermitteln, gibt es die AppDataAccess-Komponente. Im Gegensatz zur DataAccess-Komponente auf dem Backend verarbeitet diese Komponente JSON-Files.

Auf der anderen Seite haben wir das PlayFramework, welches wiederum Access-Komponenten, Routing-Komponenten, eine Timing-Komponente, Logik-Komponenten und eine DataAccess-Komponente enthält.

Die Timing-Komponente und die Logic-Komponente haben die selben Aufgaben wie ihre Gegenstücke in der Xamarin App. Auch hier verwalten diese das Zeitmanagement während den einzelnen Runden und stellen sicher, dass nach Ablauf der Zeit oder sobald alle *Brainsheets* abgegeben wurden, eine neue Runde beginnt. Des Weitern ist die Logic-Komponente zum Beispiel verantwortlich, dass ein *Participant* einer Gruppe nicht zweimal beitreten kann oder diese verlassen kann, wenn er sie schon einmal verlassen hat.

Die DataAccess-Komponente stellt sicher, dass jegliche Daten korrekt geladen oder gespeichert werden.

Die Persistence-Schicht ist für alle Datenzugriffe zuständig.

Konkret steht uns je ein DataStore für die *BrainstormingFindings*, für die *BrainstormingTeams* und für die *Participants* zur Verfügung.

Komponenten

Nachfolgend sind nochmals alle Komponenten aufgelistet und kurz beschrieben. Für eine ausführlichere Beschreibung ist der Text oberhalb zu lesen.

ViewComponent	Die View-Komponenten der Xamarin App sind für das korrekte Anzeigen der Informationen verantwortlich. Sie definieren das Aussehen der Applikation.
AppTimingComponent	Die Xamarin App hält in der logischen Schicht eine Timing-Komponente, welche dafür sorgt, dass ein *BrainstormingFinding* nach Ablauf der Zeit abgesendet wird.
AppStateMachineComponent	Die Komponente für das Evaluieren des Zustandes des Brainstormings.
AppLogicComponent	Die Logik-Komponente der Xamarin App regelt weitere Logik, wie z.B. das Hinzufügen einer Idee.
AppDataAccessComponent	Die Komponente für das Serialisieren und Deserialisieren von JSON Dateien für das und vom Backend.
AccessComponent	Die Access-Komponente auf dem PlayFramework regelt den Zugriff mittels JWT-Token. JWT-Tokens werden bei erfolgreichem Login an den Benutzer der App gesendet.

Routing Die Routing-Komponente sorgt anhand der URL für das Aufrufen der korrekten Funktion.

TimingComponent Wie die Xamarin App hält auch das PlayFramework eine Timing-Komponente, um den Zustand der Zeit verwalten zu können.

LogicComponent In der Logik-Komponente werden die eigentlichen Funktionen geschrieben. Hier ist auch die Logik für den Austausch der Blätter untergebracht.

DataAccessComponent Die DataAccess-Komponente stellt das Bindeglied zwischen dem PlayFramework und dem DataStore dar. Es ermöglicht erst den Zugriff auf die gespeicherten Daten.

PersistenceComponent Die Persistence-Komponente regelt das korrekte und dauerhafte Speichern in die einzelnen DataStores.

2.6.2 Deployment

Wie in der Abbildung 8 zu sehen ist, besteht unser System aus zwei physikalischen Geräten. Das ist zum einen der Client und zum anderen der BackendNode. Diese beinhalten jeweils sogenannte *DeploymentUnits* (DU).

Beim Client handelt es sich um das Smartphone des jeweiligen Benutzers. Auf seinem Smartphone läuft die Xamarin App, welche wiederum die appPresentationLayerDU und die appBusinessLogicLayerDU hält.

Der BackendNode ist ein Ubuntu 18.04 auf dem ein Java Runtime Environment (JRE) installiert ist. Innerhalb der JRE läuft das PlayFramework, in dem wiederum die businessLogicLayerDU läuft.

Zudem ist auf dem BackendNode ein MongoDB Service installiert, welche die persistenceLogicLayerDU beinhaltet.

Komponenten

Nachfolgend sind nochmals alle DeploymentUnits aufgelistet und kurz beschrieben.

businessLogicLayerDU Die businessLogicLayerDU enthält alle Komponenten, welche in Abbildung 7 in der Businesslogic-Schicht im PlayFramework eingezeichnet sind.

persistenceLogicLayerDU Die persistenceLogicLayerDU beinhaltet alle Komponenten der Persistence-Schicht, welche in Abbildung 7 zu sehen ist.

appPresentationLayerDU Die appPresentationLayerDU enthält alle Komponenten, welche in Abbildung 7 in der Präsentationsschicht liegen.

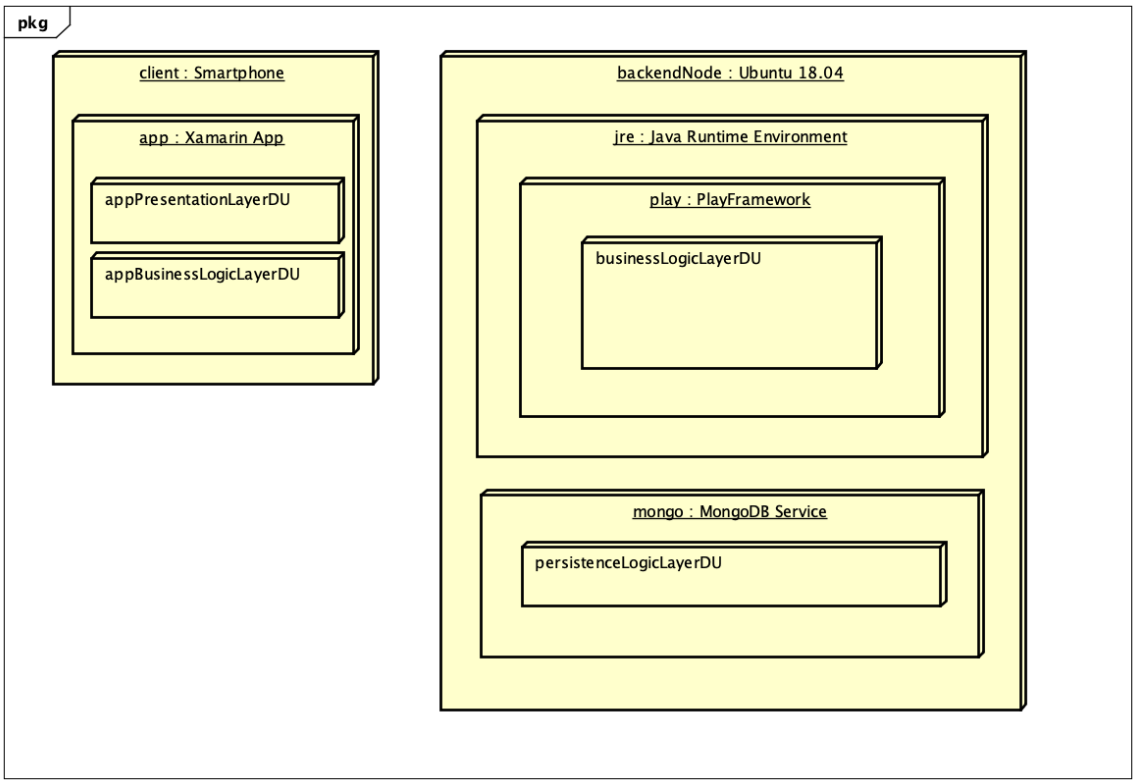

Abbildung 8: Deploymentdiagramm BrainingOutOfBox

appBusinessLogicLayerDU Die appBusinessLogicLayerDU enthält alle Komponenten, welche in Abbildung 7 in der Businesslogic-Schicht in der Xamarin App gezeichnet sind.

2.7 Architekturentscheide

Wesentliche Entscheide, welche wir während dem Projekt getroffen haben, sind hier detailliert begründet. Auch Gedanken oder Ideen, welche wir während der Analysephase hatten, dann aber wieder verworfen haben, sind hier mit Begründung aufgeschrieben.

2.7.1 Domain-Design-Entscheid

Schon sehr früh im Projekt kam die Idee auf, die BrainstormingFindings in verschiedene Arten zu unterteilen. Die Rede war von "Software Architektur Lösung" (SoftwareFinding) und "Generelle Lösung" (GeneralFinding). Siehe dazu Kapitel 2.3.1. Damit wollten wir erreichen, dass nicht jeder Ideentyp für jedes BrainstormingFinding zur Verfügung steht, zumal es auch nicht unbedingt für jede Kombination Sinn ergibt, diesen Ideentypen dafür zu verwenden. Zum Beispiel ist es wenig sinnvoll die PatternIdea bei einem GeneralFinding zu nutzen. Dies würde eher bei einem SoftwareFinding zum Einsatz kommen.

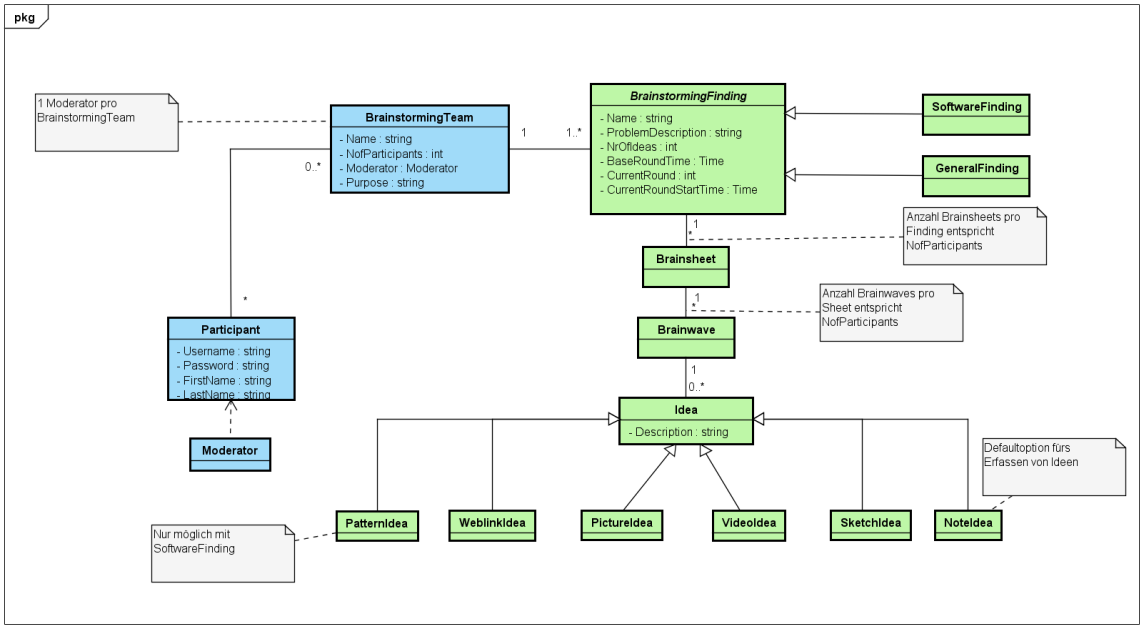

Abbildung 9: Erster Entwurf vom Domain Modell BrainingOutOfBox

Doch dabei kommt das Bedenken auf, was wenn es noch andere Patterns als nur Software-Patterns gibt, welche man zur Verfügung stellen will. Dann müssten noch mehr solcher logischen Zusammenhänge beachtet werden, was die Erweiterbarkeit der Applikation mit zunehmenden BrainstormingFinding Arten erschwert.

Je länger wir untereinander aber auch mit unserem Betreuer, Herr Zimmermann oder Silvan Gehrig darüber diskutierten, desto mehr kamen wir wieder auf unser ursprüngliches Domain-Design zurück, bei welchem es nur ein BrainstormingFinding gibt.

Dabei ist die Idee, alle Ideentypen zwar anzubieten aber die Auswahl des passenden Ideentypen komplett dem Endnutzer zu überlassen. Dies birgt allerdings die Gefahr, die Benutzung der Applikation durch eine zu grosse Auswahl an Ideentypen zu verschlechtern.

Dieses Problem kann aber durch eine durchdachte Aufteilung in der Benutzeroberfläche eingedämmt werden. Unserer Meinung nach ist mit dem verwendeten Domainmodell (siehe Abbildung 6) die Erweiterbarkeit für zukünftige Arbeiten eher gegeben, da weniger Stellen im Code bearbeitet werden müssen.

Aus diesem Grund haben wir uns für das ursprüngliche Domainmodell mit nur einem BrainstormingFinding entschieden. Die Tipps von arc42 bzw. der Solution Strategy konnten dennoch gemäss Tabelle 1 integriert werden.

2.7.2 Architekturentscheide der Studienarbeit

Nachfolgend listen wir die Entscheide auf, welche in der vorangegangenen Studienarbeit in Bezug auf die eingesetzten Technologien getroffen wurden. Es soll dem Leser nochmals verständlich machen, warum genau diese Technologien für diese Arbeit gewählt wurden. Die folgenden Kapitel sind daher der Studienarbeit [1] wortgetreu entnommen.

Xamarin.Forms oder Xamarin native

Für diesen Entscheid galt es zu evaluieren, welche User Controls für unsere Applikation die exotischsten sind. Dies, weil Xamarin.Forms eine Menge an Standard-Controls anbietet, die vom Framework selber in das jeweilige Betriebssystem konvertiert werden. Sind alle vorgesehenen Benutzerelemente in Forms enthalten, sparen wir uns die Zeit, betriebssystemspezifische Elemente zu entwickeln.

Für unser Projekt haben wir folgende Benutzerelemente als exotisch oder kritisch definiert:

- Canvas Control für Zeichnen einer Idee
- Camera Funktion für das Erkennen von Quick Response-Codes (QR-Codes)
- Verarbeitung und Generierung von QR-Codes

Nach einer Recherche stellte sich heraus, dass sich ein Canvas View von Google namens SkiaSharp eignet. Darauf lässt sich gemäss der Dokumentation zeichnen sowie definierte Formen einfügen. Dies könnte auch für eine Erweiterung spannend sein, in der Patterns in UML als Vorlage angeboten werden können.

Für die Kamera-Funktionalität steht ein NuGet-Packet (Xam.Media.Plugin) bereit, das uns diese Arbeit abnehmen wird.

Das Generieren und Lesen der QR-Codes ist an sich kein Problem von Xamarin.Forms, denn grundsätzlich müssen die von der Kamera generierten Files eingelesen und ins entsprechende QR-Code-Tool eingefügt werden. Hierfür eignet sich das NuGet ZXing.Net.

Es stellte sich relativ rasch heraus, dass die gewünschten Funktionalitäten in Xamarin.Forms in ausreichender Qualität enthalten sind und uns das individuelle Entwickeln dadurch abgenommen wird.

Backend-Technologie

Neben den Vorteilen, wie der schlanken und zustandslosen Architektur des PlayFrameworks, dem asynchronen und nicht-blockierenden Verhalten und den vielen unterstützten Bibliotheken, haben wir uns hauptsächlich dafür entschieden, weil wir in anderen Projekten schon sehr gute Erfahrungen mit dem PlayFramework gemacht haben.

Ein weiterer Grund bestand darin, dass das PlayFramework nicht nur in Scala sondern auch in Java geschrieben ist. Mit Java kennen wir uns beide gut aus und mussten uns so keine neue Programmiersprache aneignen.

Da wir uns schon relativ früh für eine MongoDB als Datenbanksystem entschieden hatten, fiel die Wahl erst recht auf das PlayFramework, als wir einen asynchronen MongoDB-Treiber für Java gefunden hatten.

MongoDB als Datenbanksystem

MongoDB (abgeleitet von humongous) ist eine dokumentorientierte, einfache, dynamische und skalierbare NoSQL Datenbank, welche von der MongoDB Inc. entwickelt wird.

Die Basis für das Speichern von Informationen bilden die sogenannten Documents. Die Datenobjekte werden in separaten Documents innerhalb einer Collection (anders als bei traditionellen relationalen Datenbanken in Spalten und Zeilen) gespeichert. Mit den Documents können zudem hierarchische Strukturen und Relationen sehr einfach gespeichert werden.

Der Vorteil einer MongoDB Datenbank liegt darin, dass zusammengehörige Informationen gemeinsam in einem Document gespeichert werden. Dies ermöglicht einen schnellen Zugriff auf die Daten mittels der MongoDB Query Language. Da MongoDB zudem ohne Schemas auskommt, ist es nicht nötig, die Datenbank offline zu nehmen, wenn man ein neues Feld einfügen möchte.

Weitere Vorteile sind laut DZone.com die hohe Performance, Verfügbarkeit (durch Replikas) und Skalierung (durch Sharding). Da alle Informationen zusammen in einem Document gespeichert sind, sind auch keine Joins zu anderen Tabellen notwendig. Auch unterstützt MongoDB Funktionen für die Speicherung von Geoinformationen.

Der Hauptgrund warum wir uns für MongoDB als Datenbanksystem entschieden haben, war die Tatsache, dass alle zusammengehörigen Informationen in einem Document abgelegt werden können. Auch die Möglichkeit hierarchische Strukturen (bei uns die *Brainsheets*, *Brainwaves* und *Ideas*) einfach zu Speichern, hat uns viel Zeit erspart.

Die Mächtigkeit von relationalen Datenbanken im Bereich einer Auswertung, ist in unserem Projekt nicht notwendig. Daher haben wir auch von Beginn an auf das Paradigma der dokumentorientieren Datenbanken gesetzt. Grund warum wir uns schlussendlich für MongoDB und nicht für eine andere dokumentorientierte Datenbank entschieden haben ist, dass MongoDB Thema während dem Studium ist und wir schon einige Erfahrung damit hatten.

Runden-Ende-Entscheid

Der Entscheid, wie eine Runde zu enden hat bzw. was passiert, wenn die Zeit in der laufenden Runde abläuft, der Participant aber noch nicht alle seine Ideen in die Brainwave gespeichert hat, hatten wir schon relativ früh in der Studienarbeit entschieden aber

nirgends dokumentiert. Dies soll hier nachgeholt und begründet werden.

Grundsätzlich haben wir uns in solch einem Fall dazu entschlossen, lediglich jene Ideen in der Datenbank zu persistieren, welche auch in der Brainwave gespeichert sind. Im Umkehrschluss bedeutet das, dass alle Ideen, welche zu diesem Zeitpunkt nicht in der Brainwave gespeichert sind (betrifft nur die letzte erfasste Idee), verloren gehen.

Der Grund warum wir uns dafür entschieden haben liegt einerseits darin, dass dies die einfachste Lösung für die geschilderte Situation darstellt. Da dies zudem nur eine Idee betreffen kann, kann auch nur maximal diese Idee 'verloren' gehen.

Zudem wird bei der originalen Version mit Papier gleich verfahren. Auch dort wird unabhängig, ob ein Participant alle Ideen ausfüllen konnte oder nicht, das Blatt weitergereicht.

2.8 Ergebnisse

Das Kapitel der Ergebnisse befasst sich mit der konkreten Umsetzung der gesamten Applikation und dokumentiert die grundlegenden Ergebnisse oder Resultate, welche aus dieser Arbeit hervorgegangen sind.

Auch hier soll nochmals erwähnt sein, dass diese Arbeit auf der Studienarbeit [1] aufbaut und grundlegende Informationen dort nachzulesen sind.

In diesem Kapitel gehen wir zunächst auf die Änderungen während der Refactoring-Phase ein und schildern was dies für die Wartung und Erweiterung des Codes bedeutet.

Anschliessend sind weitere Features dokumentiert, welche im Verlauf dieser Arbeit an der Applikation durchgeführt wurden.

Das Kapitel wird durch eine Gegenüberstellung der erreichten und geplanten Arbeit abgeschlossen.

Zur besseren Übersicht wurde der Code vereinzelt gekürzt. Dies ist durch 3 Punkte (...) gekennzeichnet.

2.8.1 Refactoring zu Beginn der Arbeit

Wie aus dem Projektplan (siehe Abbildung 15) zu entnehmen ist, haben wir uns zu Beginn unserer Arbeit dazu entschieden, zwei Sprints für das Refactoring unsere Applikation zu nutzen. Die Entscheidung überhaupt ein Refactoring durchzuführen entstand aus der Tatsache, dass wir schon gegen Ende der Studienarbeit gemerkt hatten, dass bei einer allfälligen Weiterführung der Arbeit ein Refactoring von grossem Wert sein wird. Auch haben wir diesbezüglich schon während der Studienarbeit einzelne Vorschläge für ein Refactoring beschrieben.

Das Ziel der Refactoring-Phase war es daher, den Code, welcher aus der Studienarbeit hervorgegangen ist, zu prüfen und zu verbessern. Dies sollte uns helfen, die Erweiterbarkeit für zukünftig geplante Features zu gewähren.

Auch konnten wir die in der Studienarbeit beschriebenen Vorschläge erfolgreich in der App umsetzen.

Refactoring Backend

Dieser Abschnitt ist so aufgebaut, dass anhand von zwei Beispielen aufgezeigt werden soll, wie der Code im Backend vor und nach dem Refactoring ausgesehen hat. Auch wird am Ende des Abschnitts kurz erwähnt, was dies für Verbesserungen zur Folge hatte.

Das nachfolgende Listing 1 zeigt einen Ausschnitt aus der `ParticipantController` Klasse wie es vor dem Refactoring war. Das Listing 2 zeigt dieselbe Methode nach dem Refactoring.

```
1 public Result createParticipant(){
2
3     JsonNode body = request().body().asJson();
4
5     if (body == null ) {
```

```
6           return forbidden(Json.toJson(new ErrorMessage("Error", "
                json body is null")));
7       } else if(  body.hasNonNull("username") &&
8                   body.hasNonNull("password") &&
9                   body.hasNonNull("firstname") &&
10                  body.hasNonNull("lastname")) {
11
12      Participant participant = new Participant(body.get("username"
            ).asText(), body.get("password").asText(), body.get("
            firstname").asText(), body.get("lastname").asText());
13
14      participantCollection.insertOne(participant, new
            SingleResultCallback<Void>() {
15          @Override
16          public void onResult(Void result, Throwable t) {
17              Logger.info("Inserted Participant!");
18          }
19      });
20
21      return ok(Json.toJson(new SuccessMessage("Success", "
            Participant successfully inserted")));
22      }
23
24      return forbidden(Json.toJson(new ErrorMessage("Error", "json
            body not as expected")));
25  }
```

Listing 1: Participant erstellen vor Refactoring

```
1   @BodyParser.Of(ParticipantDTOBodyParser.class)
2   public Result createParticipant() {
3       ParticipantDTO participantDTO = request().body().as(
            ParticipantDTO.class);
4       Participant participant = modelsMapper.toParticipant(
            participantDTO);
5
6       try {
7           if (service.insertParticipant(participant)){
8               return ok(Json.toJson(new SuccessMessage("Success", "
                    Participant successfully inserted")));
9           } else {
10              Logger.info("Username already exists");
11              return badRequest(Json.toJson(new ErrorMessage("Error
                    ", "Username already exists")));
12          }
13      } catch (ExecutionException | InterruptedException e) {
14          return internalServerError(Json.toJson(new ErrorMessage("
```

```
            Error", e.getMessage())));
15      }
16  }
```

Listing 2: Participant erstellen nach Refactoring

Das Erstellen des Participants wurde komplett in den `ParticipantDTOBodyParser` ausgelagert. Dieser erstellt nun aus dem angelieferten JSON (JavaScript Object Notation) ein Data-Transfer-Object [12], welches in einem nächsten Schritt zu einem Business Object aufgelöst wird (Zeile 4).

Jegliche Überprüfungen aus Listing 1 (Zeilen 5 und 7-10) konnten so zentral in den BodyParser ausgelagert werden.

Bei den nachfolgenden Listings 3 und 4, welche einen Ausschnitt aus dem `BrainstormingFindingController` zeigen, ist der Unterschied noch stärker zu sehen. So konnte zum Beispiel die `putBraisheet` Methode von zirka 40 Zeilen auf 15 Zeilen gekürzt werden.

```
1   public Result putBrainsheet(String findingIdentifier) throws
        ExecutionException, InterruptedException {
2
3   JsonNode body = request().body().asJson();
4   JsonNode brainwaves = body.findPath("brainwaves");
5   JsonNode nrOfSheet = body.findPath("nrOfSheet");
6
7   if (body == null ) {
8       return forbidden(Json.toJson(new ErrorMessage("Error", "json
            body is null")));
9   } else if(  !brainwaves.isNull() &&
10              !nrOfSheet.isNull()){
11
12   BrainstormingFinding finding = getBrainstormingFinding(
        findingIdentifier);
13
14      if (finding == null){
15          return internalServerError(Json.toJson(new ErrorMessage("
                Error", "No brainstormingFinding found")));
16      }
17
18      Brainsheet oldBrainsheet = finding.getBrainsheets().get(
            nrOfSheet.asInt());
19      Brainsheet newBrainsheet = createBrainsheet(body);
20
21
22      findingCollection.updateOne(eq("identifier",
            findingIdentifier),pullByFilter(Filters.eq("brainsheets",
            oldBrainsheet)), new SingleResultCallback<UpdateResult>()
            {
```

```
23          @Override
24          public void onResult(final UpdateResult result, final
               Throwable t) {
25              Logger.info(result.getModifiedCount() + " Brainsheet
                   successfully deleted");
26          }
27      });
28
29      findingCollection.updateOne(eq("identifier",
           findingIdentifier),combine(pushEach("brainsheets", Arrays.
           asList(newBrainsheet), new PushOptions().position(
           newBrainsheet.getNrOfSheet())), inc("
           deliveredBrainsheetsInCurrentRound", 1)), new
           SingleResultCallback<UpdateResult>() {
30          @Override
31          public void onResult(final UpdateResult result, final
               Throwable t) {
32              Logger.info(result.getModifiedCount() + " Brainsheet
                   successfully inserted");
33          }
34      });
35
36      return ok(Json.toJson(new SuccessMessage("Success", "
           Brainsheet successfully updated")));
37  }
38
39  return forbidden(Json.toJson(new ErrorMessage("Error", "json body
       not as expected")));
40  }
```

Listing 3: PutBrainsheet vor Refactoring

```
1   @BodyParser.Of(BrainsheetDTOBodyParser.class)
2   public Result putBrainsheet(String findingIdentifier){
3       BrainsheetDTO brainsheetDTO = request().body().as(
           BrainsheetDTO.class);
4       Brainsheet newBrainsheet = modelsMapper.toBrainsheet(
           brainsheetDTO);
5
6       try {
7
8           if (service.exchangeBrainsheet(findingIdentifier,
               newBrainsheet)) {
9               return ok(Json.toJson(new SuccessMessage("Success", "
                   Brainsheet successfully updated")));
10          } else {
11              return badRequest(Json.toJson(new ErrorMessage("Error
```

```
                ", "No Brainsheet updated")));
12          }
13
14      } catch (ExecutionException | InterruptedException e) {
15          return internalServerError(Json.toJson(new ErrorMessage("
               Error", e.getMessage())));
16      }
17  }
```

Listing 4: PutBrainsheet nach Refactoring

Die gesamte Logik für den Austausch der Brainsheets wurde in die `FindingService` Klasse ausgelagert.

```
1  public boolean exchangeBrainsheet(String findingIdentifier,
      Brainsheet newBrainsheet) {
2   BrainstormingFinding finding = service.getFinding(
       findingIdentifier).get();
3   if (finding == null){
4       return false;
5   } else if (newBrainsheet.getNrOfSheet() < finding.getBrainsheets
       ().size()) {
6           service.exchangeBrainsheet(finding, newBrainsheet);
7           return true;
8       }
9   }
10  return false;
11 }
```

Listing 5: Exchange Brainsheet im Business Layer

Auch wurde die Implementation für das Einfügen in die Datenbank in eine separate Klasse (`MongoDBFindingService`) ausgelagert, um eine bessere Übersicht und ein besseres Layering zu gewähren.

```
1  public void exchangeBrainsheet(BrainstormingFinding finding,
      Brainsheet newBrainsheet){
2
3   //update Brainsheet at the correct index
4   findingCollection.updateOne(eq("identifier", finding.
       getIdentifier()),combine(set("brainsheets."+newBrainsheet.
       getNrOfSheet(), newBrainsheet), inc("
       deliveredBrainsheetsInCurrentRound", 1)), (result, t) ->
       Logger.info(result.getModifiedCount() + " new Brainsheet
       successfully updated"));
5  }
```

Listing 6: Exchange Brainsheet im Data Access Layer

Diese zwei Beispiele stellen noch lange nicht alle überarbeiteten Codestellen dar, geben aber eine gute Übersicht, wie der gesamte Code vereinfacht werden konnte.

Fazit

Gerade das Beispiel vom Austausch der Brainsheets (Listing 3) verdeutlicht, dass das Layering stark verbessert wurde. Damit konnte die Übersicht und Komplexität deutlich verbessert bzw. verringert werden.

Auch wurden im gesamten Backend Data-Transfer-Objekte (DTO) [12] eingefügt. Die Idee von DTOs ist es, alle Daten, welche über das Netzwerk gesendet werden in eben jenen Data-Transfer-Obekten zu speichern und zu übertragen. Im Unterschied zu den Business-Objects besitzen DTOs kein business-relevantes Verhalten. Sie sind ausschliesslich für die Übertragung der Daten verantwortlich. Dies war vor allem im späteren Verlauf der Arbeit von grossem Wert.

Mit Hilfe der BodyParser-Klassen konnte die komplette Überprüfung und Deserialisierung der angelieferten JSON-Daten zentral geregelt werden. Somit ist immer sichergestellt, dass das DTO korrekt (alle erwarteten Informationen sind vorhanden und das Format stimmt) erstellt wird.

Refactoring Frontend

Auf der Seite des Frontends standen folgende Punkte für das Refactoring an:

1. Performanceverbesserung
2. State Machine für die Brainstorming Logik
3. Services, allgemeines Layering
4. Einfügen von DTOs
5. Logger
6. Localisation

Zuerst wurde ein einfaches Layering vorgenommen. Alle Models wurden in ein separates Projekt ausgelagert. Zusätzlich wurden einzelne Services wie der `UiNavigationService` erstellt, welcher jegliche Navigationsschritte ausführen kann und in die ViewModels injected wird. Durch das Einführen dieses Services hat sich bereits ein grösseres Problem gelöst: die Performance Issues. Vor diesem Service rief jedes ViewModel den durch das MVVM-Framework Prism Forms bereitgestellten `NavigationService` auf. Dies führte dazu, dass bei jedem Navigationsschritt der `NavigationService` neu initialisiert wurde. Durch das Kapseln in eine eigene Klasse kann dieser als Singleton im IoC Container registriert werden, somit wird dieser nur beim Starten der Applikation initialisiert.

Nachdem die Navigation in einen Service ausgelagert wurde, musste die Implementation der Kern-Logik überarbeitet werden. Dazu wurde im Umfang der Studienarbeit bereits ein Konzept für eine State-Machine erarbeitet [1]. Diese wurde mittels Test-Driven-Design in der jetzigen Arbeit umgesetzt. Listing 7 zeigt, wie die Statemachine beim Start ermittelt, in welchem State sie sich befindet.

```
1  public void Start()
2  {
3   var currentRound = _context.CurrentFinding.CurrentRound;
4   IState evaluatedState = null;
5   if (currentRound == -1)
6   {
7    evaluatedState = new EndedState(_context, _brainstormingModel);
8   }
9   else if (currentRound == 0)
10  {
11   evaluatedState = new WaitingState(_logger,
        _brainstormingDalService, _context, _brainstormingModel);
12  }
13  else if (currentRound > 0)
14  {
15   evaluatedState = new RunningState(_logger,
        _brainstormingDalService, _context, _brainstormingModel);
16  }
17  if(currentRound < -1 || evaluatedState == null)
18  {
19   throw new ArgumentException("Invalid round or state not
        registered");
20  }
21  ChangeState(evaluatedState);
22 }
```

Listing 7: Start der Klasse StateMachine

Wie auf Zeile 4 ersichtlich wurde gegen ein Interface namens `IState` programmiert. Dieses Interface enthält eine `Init()`- und eine `CleanUp()`-Methode, sowie ein `PropertyChanged`-Event, der für das Notifizieren der Änderungen eines States zuständig ist. Durch das `IState`-Interface ist die Erweiterbarkeit gewährleistet (z.B. Einführen eines zusätzlichen 'Review' States).

Die StateMachine musste anschliessend von einem Service verwendet werden. Der `BrainstormingService` ist für die gesamte Kernlogik zuständig und hält somit eine Instanz der `StateMachine`. Auch im `BrainstormingService` wurde gegen ein Interface programmiert. Dessen Methoden sind der Abbildung 10 zu entnehmen.

Neben diesen erwähnten Services wurden auch Services im Data-Access-Layer (DAL) eingeführt. Diese sind in einem separaten Projekt und können ebenfalls mittels deren Interface in die verwendenden Klassen injiziert werden. Sie sind für den gesamten Daten-Zugriff zuständig. Für das Abbilden der Endpunkte des Backends, wurde weiter ein Konfigurationsfile eingeführt. Dieses Json-File wird geparst und vom `ConfigurationService` den Repositories zur Verfügung gestellt.

Für das Einführen von DTOs wurde in einem ersten Schritt, wie zu Beginn dieses Kapitels erwähnt, alle Models in ein separates Projekt ausgelagert. Im DAL existiert ein Ordner mit DTOs, welche für das Parsen der Objekte vom Backend verwendet werden. Es wurde danach ein Mapping-Layer eingeführt, der die Daten-Transfer-Objekte in die

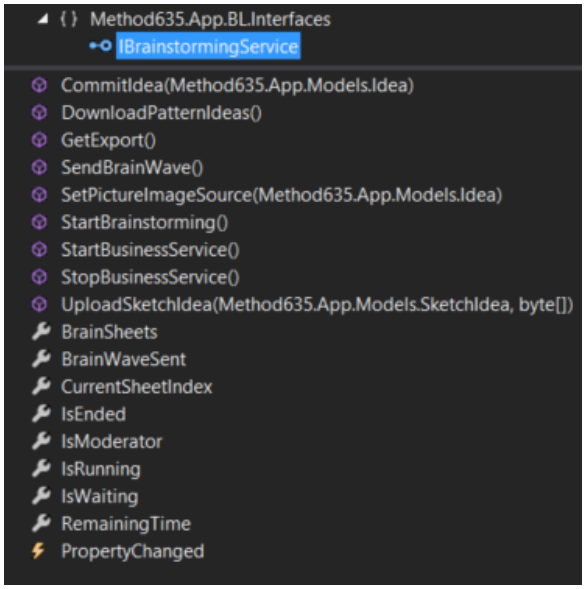

Abbildung 10: Methoden des IBrainstorming-Interfaces

Business Objekte und umgekehrt abbildet. Dazu wurde das NuGet-Packet Automapper verwendet.

Fazit

Das Auslagern und Zerstückeln der Logik in Services und das Verwenden von Interfaces erleichterte uns die Implementation der nachfolgend beschriebenen Features erheblich. Weiter konnten besonders durch die verschiedenen States in der `StateMachine` späteres Fehlverhalten viel besser eingeschränkt und korrigiert werden.

2.8.2 Implementation Skizzen Feature

Die Skizzen-Funktion soll es dem Endnutzer ermöglichen, nebst den `NoteIdeas` auch `SketchIdeas` (siehe Abbildung 6) zu erfassen. Mit den `SketchIdeas` wird dem Endnutzer die Möglichkeit gegeben, Skizzen als Teil der Brainstorming-Runden zu zeichnen.

Da es sich bei den einzelnen Skizzen um Binärdaten handelt, musste zunächst eine Möglichkeit geschaffen werden, diese Art von Daten in der Datenbank abzulegen. Dafür wurden in der Elaboration-Phase Analysen (siehe Kapitel 2.3.2) durchgeführt und ein Prototyp erstellt, um die technische Machbarkeit zu verifizieren.

Implementation Backend

In den nachfolgenden Listings wird aufgezeigt, wie die Sketch-Funktion im Backend umgesetzt wurde. Der genaue Kommunikations-Ablauf kann der Abbildung 3 in Kapitel 2.4.1.3 entnommen werden.

```
1 @BodyParser.Of(MultipartFormDataBodyParser.class)
2 public Result uploadFile(){
```

```
3       try {
4
5        final Http.MultipartFormData<File> formData = request().body
            ().asMultipartFormData();
6        final Http.MultipartFormData.FilePart<File> filePart =
            formData.getFile("name");
7        final File file = filePart.getFile();
8
9        final byte[] fileData = Files.readAllBytes(file.toPath());
10       final String fileName = file.getName();
11
12       String fileId = service.uploadFileAsStream(fileData,
            fileName);
13       return ok(Json.toJson(new SuccessMessage("Success", fileId))
            );
14
15      } catch (IOException | InterruptedException |
           ExecutionException  e) {
16           return internalServerError(Json.toJson(new ErrorMessage("
                Error", e.getMessage())));
17      }
18  }
```

Listing 8: Upload File im File Controller

Ähnlich wie bei den JSON-Daten, wird auch hier zunächst die Skizze mittels eines BodyParsers (Zeile 1) in ein File (in-Memory) umgewandelt (Zeilen 5-7), welches dann als Byte-Array dem File-Service (Zeile 12) übergeben werden kann.

```
1  public String uploadFileAsStream(byte[] stream, String fileName)
       ... {
2       return service.uploadFileAsStream(stream,fileName);
3  }
```

Listing 9: Upload File im File Service

Der File-Service nimmt das Byte-Array entgegen und gibt dieses unverändert dem Datenbank-Service weiter.

```
1  @Override
2  public String uploadFileAsStream(byte[] stream, String fileName)
       ... {
3       ByteBuffer data = ByteBuffer.wrap(stream);
4       CompletableFuture<String> future = new CompletableFuture<>();
5
6       final GridFSUploadStream uploadStream = gridFSBucket.
            openUploadStream(fileName);
7       uploadStream.write(data, (result, t) -> {
```

```
8           Logger.info("File successfully inserted; ID: " +
                uploadStream.getObjectId().toHexString());
9           future.complete(uploadStream.getObjectId().toHexString())
                ;
10
11          uploadStream.close((result1, t1) -> {
12              // stream close
13          });
14      });
15
16      return future.get();
17  }
```

Listing 10: Upload File im DB Service

Der Datenbank-Service speichert anschliessend das Bild in die Datenbank (Zeile 7) und liefert bei erfolgreicher Speicherung die ObjektID zurück (Zeile 9). Am Ende wird noch der uploadStream zur Datenbank geschlossen (Zeile 11).

Die Smartphone-Applikation speichert nun die ObjektID als Teil der `SketchIdea` und sendet das `Brainsheet` wie gewohnt nach Ablauf der Zeit an das Backend.

Das Herunterladen der gespeicherten Bilder funktioniert auf ganz ähnliche Weise. Da die ObjektID in der `SketchIdea` abgelegt ist, kann das eigentliche Bild problemlos wiedergefunden werden.

```
1   @Override
2   public byte[] downloadFileAsStream(String id) ...{
3       ObjectId fileId = new ObjectId(id);
4       final ByteBuffer dstByteBuffer = ByteBuffer.allocate(1024 *
            1024);
5       final GridFSDownloadStream downloadStream = gridFSBucket.
            openDownloadStream(fileId);
6       CompletableFuture<byte[]> future = new CompletableFuture<>();
7
8       downloadStream.read(dstByteBuffer, (result, t) -> {
9           dstByteBuffer.flip();
10          byte[] bytes = new byte[result];
11          dstByteBuffer.get(bytes);
12          Logger.info("Found file to download; Size: " + result);
13          future.complete(bytes);
14
15          downloadStream.close((result1, t1) -> {
16              // stream closed
17          });
18      });
19      return future.get();
20  }
```

Listing 11: Download File im DB Service

Hierbei wird ein downloadStream geöffnet, welcher die Daten anhand der ID aus der Datenbank liest und in ein Byte-Array schreibt (Zeile 8). Auch hier wird am Ende der Stream wieder geschlossen (Zeile 15).

Der zurückgelieferte Byte-Array wird anschliessend unverändert an den Client zurückgeschickt. Allfällige Fehler während der Ausführung sowohl beim Hochladen als auch beim Herunterladen werden im File-Controller abgefangen und als Fehler an den Client geschickt.

Implementation Frontend

Das Feature wird im Frontend durch folgende Schritte während des Brainstormings erreicht:

1. Klicken des 'More Ideas'-Buttons, der auf zusätzliche Ideentypen hindeutet
2. Auswählen des 'Sketch Idea' in der nächsten Ansicht

Nachdem die Sketch-Ansicht erscheint, hat man Optionen zur Auswahl von Stiftgrösse und -farbe und kann die Skizze speichern oder die Zeichenfläche löschen.

Abbildung 11 verdeutlicht den Navigationsablauf.

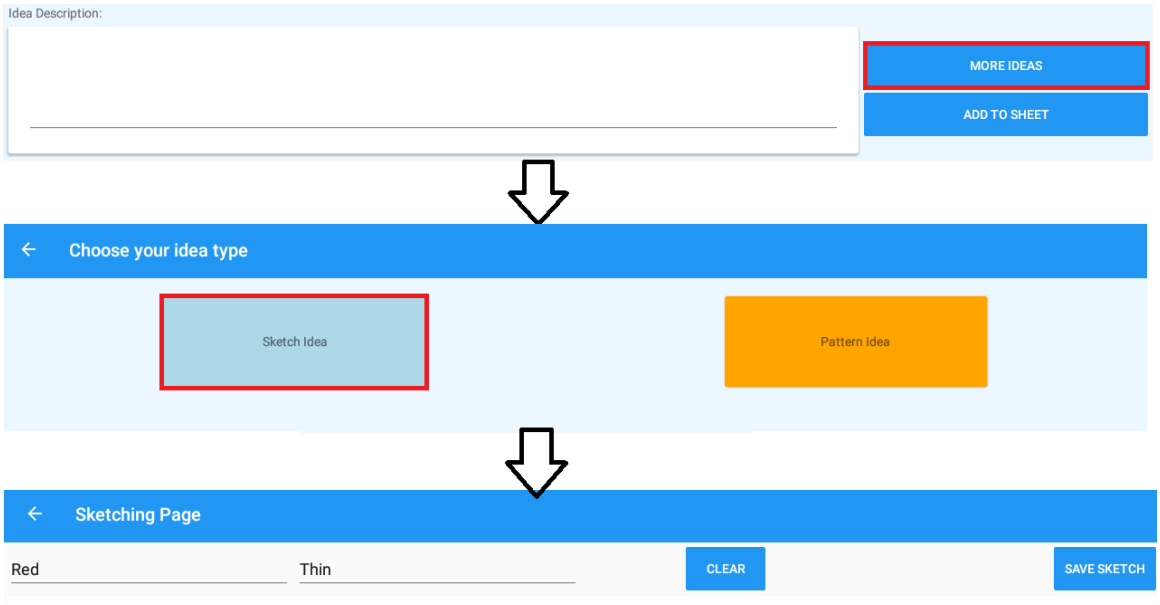

Abbildung 11: Navigationsablauf zur SketchPage

Durch das Betätigen des 'More Ideas'-Buttons wird der `UiNavigationService` aufgerufen, welcher wiederum auf die `InsertSpecialPage` navigiert. Dasselbe geschieht bei der Auswahl der 'Sketch Idea'-Kachel, nur wird dieses Mal die `SketchPage` aufgerufen.

Der Grossteil der Implementation der Sketching-Page hält sich stark an das Beispiel in der Dokumentation von Microsoft [13]. Ergänzend kam die Funktionalität, das Bild zu speichern und an das Backend zu schicken. Dazu muss ein Snapshot des Canvas' zum Zeitpunkt des Betätigen des 'Save-Image'-Buttons gemacht werden. Der Code dazu ist in Listing 12 zu finden.

```
1  private void Save_Clicked(object sender, EventArgs e)
2  {
3   var info = new SKImageInfo((int)canvasView.CanvasSize.Width, (
       int)canvasView.CanvasSize.Height);
4   var surface = SKSurface.Create(info);
5   var canvas = surface.Canvas;
6   canvas.Clear();
7
8   foreach (FingerPaintPolyline polyline in completedPolylines)
9   {
10   paint.Color = polyline.StrokeColor.ToSKColor();
11   paint.StrokeWidth = polyline.StrokeWidth;
12   canvas.DrawPath(polyline.Path, paint);
13  }
14
15  foreach (FingerPaintPolyline polyline in inProgressPolylines.
       Values)
16  {
17   paint.Color = polyline.StrokeColor.ToSKColor();
18   paint.StrokeWidth = polyline.StrokeWidth;
19   canvas.DrawPath(polyline.Path, paint);
20  }
21
22  canvas.Flush();
23
24  var snap = surface.Snapshot();
25  SketchIdea sketchIdea = new SketchIdea();
26  byte[] bytes;
27  using (var data = snap.Encode(SKEncodedImageFormat.Png, 80))
28  {
29   sketchIdea.ImageStream = data.AsStream();
30   bytes = data.ToArray();
31  }
32  _brainstormingService.UploadSketchIdea(sketchIdea, bytes);
33  _brainstormingService.CommitIdea(sketchIdea);
34  DisplayAlert(AppResources.SketchSavedTitle, AppResources.
       SketchSavedMessage, AppResources.Ok);
35 }
```

Listing 12: Save Click-Eventhandler

Zeilen 3-22 sind dafür zuständig, alle gezeichneten Linien sowie deren Farben und Dicke auf ein `SKSurface`-Objekt zu zeichnen. Darauf kann nun ein Snapshot gemacht werden. Auf dem daraus resultierenden Objekt kann nun ein png-encodierter Byte-Stream erstellt werden, der auf Zeile 32 dem `BrainstormingService` zum Upload mitgegeben wird. Die `SketchIdea` muss mitgegebenen werden, weil das Backend eine ID erstellt, unter welcher das Byte-Array abgelegt wird (siehe Zeile 9 in Listing 11). Diese ID kommt als Ant-

wort zurück und wird dem SketchIdea-Objekt gesetzt und dann commitet. Der gesamte Kommunikationsablauf kann dem Sequenz-Diagramm (Abbildung 3) im Kapitel 2.4.1.3 entnommen werden.

Ist dieser Schritt ausgeführt, muss der Benutzer, wie in der Alert-Message mitgeteilt, selber zurück zum Brainstorming navigieren. Zweimaliges Klicken auf den 'Zurück'-Pfeil oben links reicht dazu aus. Daraufhin wird sichtbar, dass die Zeichnung im entsprechenden Brainsheet platziert wurde. Dies dank dem Aufruf auf dem BrainstormingService, der die SketchIdea commitet und die Quelle des anzuzeigenden Bildes setzt. Die Methode CommitIdea() ist für das Commiten aller Ideentypen zuständig (Note-, Sketch- und Patternideas). Die Implementation dazu ist im Code-Snippet 13 aufgelistet.

```
1  public async Task CommitIdea(Idea idea)
2  {
3   try
4   {
5    SetIdea(idea);
6    commitIdeaIndex++;
7
8    if (idea is PictureIdea pictureIdea)
9    {
10    await SetPictureImageSource(pictureIdea);
11   }
12  }
13  catch (ArgumentOutOfRangeException ex)
14  {
15   _logger.Error("Invalid index access!", ex);
16  }
17 }
```

Listing 13: Commit-Idea Methode auf dem BrainstormingService

2.8.3 Implementation Pattern Feature

Nebst der Skizzen-Funktion war es unser Ziel, dem Endnutzer eine Funktion anzubieten, welche es ihm ermöglicht, aus einer Liste von verschiedensten Pattern ein passendes Pattern auszuwählen.

Wir erweiterten unsere Applikation daher um eine PatternIdea. Die verwendeten Pattern stammen ausschliesslich von der Webseite Microservice API Patterns. Dabei ist es aber durchaus auch denkbar Pattern von anderen Quellen zu nehmen oder sich gar auf nicht-software Pattern zu konzentrieren.

Implementation Backend

In den nachfolgenden Listings wird aufgezeigt, wie die Pattern-Funktion im Backend umgesetzt wurde. Der genaue Kommunikations-Ablauf kann der Abbildung 4 im Kapitel 2.4.1.3 entnommen werden.

```
1  public Result getAllPatternIdeas(){
2      CompletableFuture<Queue<PatternIdea>> future = service.
           getAllPatternIdeas();
3      ArrayList<JsonNode> list = new ArrayList<>();
4
5      try {
6
7          for (PatternIdea patternIdea : future.get()) {
8              PatternIdeaDTO patternIdeaDTO = modelsMapper.
                   toPatternIdeaDTO(patternIdea);
9              JsonNode node = Json.toJson(patternIdeaDTO);
10             ((ObjectNode)node).put("type", "patternIdea");
11             list.add(node);
12         }
13
14         return ok(Json.toJson(list));
15
16     } catch (InterruptedException | ExecutionException e) {
17         return internalServerError(Json.toJson(new ErrorMessage("
               Error", e.getMessage())));
18     }
19 }
```

Listing 14: Alle Pattern holen im Pattern Controller

Um alle verfügbaren Patterns abzurufen wird auf dem Pattern-Service (Zeile 2) die Methode *getAllPatternIdeas* aufgerufen. Anschliessend werden die gefundenen Patterns lediglich in ein DTO umgewandelt (Zeile 8) und fast unverändert als JSON an den Client gesendet (Zeile 14).

Der Endnutzer kann nun aus einer Liste der verfügbaren Patterns das gewünschte Pattern auswählen. Dieses wird, wie schon die `SketchIdea` oder `NoteIdea`, als Teil des `Brainsheets` an das Backend geschickt.

Wie in allen Controller-Klassen werden auch hier allfällige Fehler während der Ausführung abgefangen und an den Client gesendet (Zeile 16/17).

```
1  public CompletableFuture<Queue<PatternIdea>> getAllPatternIdeas()
       {
2          return service.getAllPatternIdeas();
3  }
```

Listing 15: Alle Pattern holen im Pattern Service

Der Pattern-Service leitet den Aufruf seinerseits direkt an den Datenbank-Service weiter.

```
1  @Override
2  public CompletableFuture<Queue<PatternIdea>> getAllPatternIdeas()
       {
3      CompletableFuture<Queue<PatternIdea>> future = new
           CompletableFuture<>();
4      Queue<PatternIdea> queue = new ConcurrentLinkedQueue<>();
5
6      patternIdeaCollection.find().sort(Sorts.ascending("category")
           ).forEach(
7              finding -> queue.add(finding), (result, t) -> {
8                  Logger.info("Get all available patterns");
9                  future.complete(queue);
10             });
11
12     return future;
13 }
```

Listing 16: Alle Pattern holen im Pattern DB Service

Der DB-Service wiederum findet alle Patterns und gibt diese nach Kategorie sortiert (Zeile 6-12) als Resultat zurück.

Implementation Frontend

Die Navigation zum Einfügen eines Patterns verläuft analog zum Einfügen einer Skizze, jedoch muss anstelle der hellblauen die orange Kachel 'Pattern Idea' ausgewählt werden. Diese Aktion führt zur Ansicht gemäss Abbildung 12.

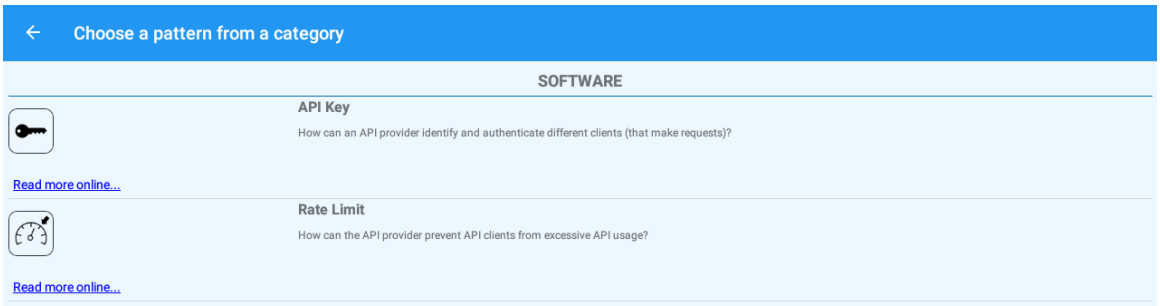

Abbildung 12: Pattern Übersicht

Der angezeigte Inhalt wird vom Backend gesendet. Die Aufgabe des Frontends ist es, die Patterns gruppiert anzuzeigen sowie den Link klickbar darzustellen. Das Icon muss zudem angezeigt werden.

Das wird erreicht, indem die `ListView` von Xamarin Forms und deren erweiterte Group-Funktion verwendet wird. Dies erlaubt es, eine Liste von Patterns gemäss ihrer Kategorie (z.B. 'Software' oder 'Psychologie') zu gruppieren.

Listing 2.8.3 zeigt, wie die zuständigen Properties im ViewModel gesetzt werden. Das `GroupDisplayBinding`-Property der `ListView` ist mit einem Binding auf das `Grouped-`

Patterns-Property verbunden. So dient ein PatternIdeaModel für jedes Item in der Liste als Informationsquelle.

```
private void SetPatternList()
{
 var patterns = _brainstormingService.DownloadPatternIdeas();
 var groupedPatterns = patterns.GroupBy(p => p.Category);
 foreach(var group in groupedPatterns)
 {
  var patternList = new GroupedPatternList();
  var patternIdeaModels = group.Select(p => new PatternIdeaModel(
     p)).ToList();
  patternIdeaModels.ForEach((p)=>
  {
   _brainstormingService.SetPictureImageSource(p);
  });
  patternList.AddRange(patternIdeaModels);
  patternList.Category = group.Key;
  GroupedPatterns.Add(patternList);
 }
}
```

Wichtig zu erwähnen: GroupedPatterns ist vom Typ List<GroupedPatternList> und GroupedPatternList wiederum erbt von List<PatternIdeaModel>. Die Relation kann ebenfalls dem Klassendiagramm in Abbildung 13 entnommen werden.

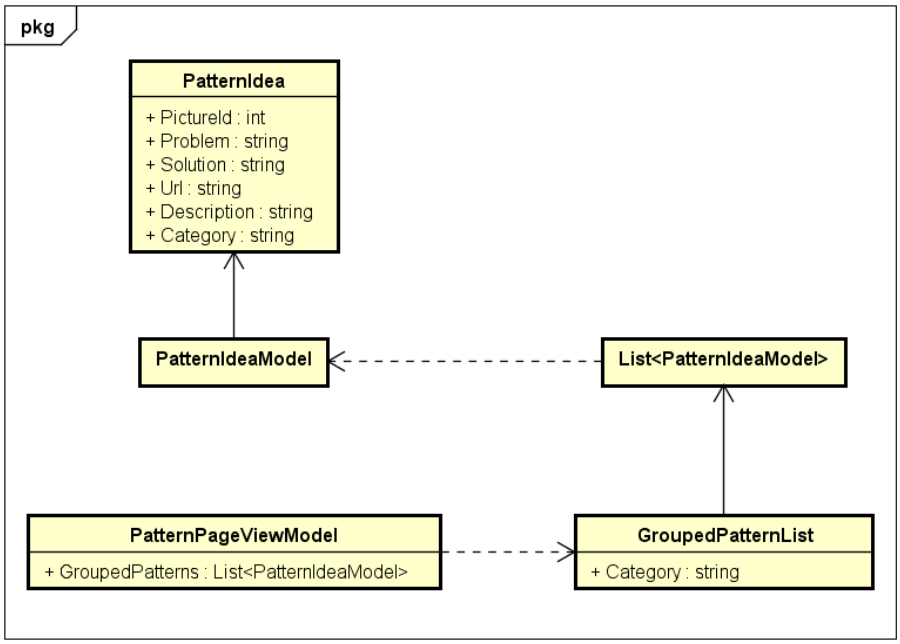

Abbildung 13: Klassendiagramm der Pattern-Page ListView

Hinzu kommt nun, dass bei einem Tap auf ein Item in der Liste das ausgewählte Pattern in das aktuelle Brainsheet eingfügt wird. Dazu wurde ein `TapGestureRecognizer` auf dem Grid der ViewCell hinzugefügt. Im dahinterliegenden Command wird wie schon bei der Skizze die `CommitIdea()`-Methode des Brainstorming Services mit dem geklickten Pattern als Parameter aufgerufen.

Zuletzt wird dem Benutzer noch eine Toast-Notification angezeigt, mit der Bestätigung, dass das Pattern dem Sheet hinzugefügt wurde und mit der Aufforderung, dass er von selbst zum Brainstorming zurücknavigieren soll.

2.8.4 Implementation Export Feature

Die letzte Funktion, welche wir in unserer Bachelorarbeit verwirklichen wollten, war die Möglichkeit, die erarbeiteten Lösungsvorschläge in geeigneter Form zu exportieren.

Nach einer kleineren Analysephase und einer Beratungsrunde mit unserem Betreuer fiel die Wahl nach der geeigneten Form auf Markdown [14]. Der Vorteil von Markdown sahen wir vor allem darin, dass man Markdown schnell und einfach z.B. auf GitHub hochladen kann. Da man als Entwickler und Software-Architekten tendenziell viel mit GitHub oder ähnlichen Tools/Webseiten arbeitet, empfanden wir diese Umsetzung am einfachsten und intuitivsten von der Handhabung.

Implementation Backend

Die Möglichkeit des Exports wird hauptsächlich durch das Backend bereit gestellt. Dabei wurden die Business-Objekte um eine Serialize-Methode erweitert, welche beschreibt, wie die Objekte im Markdown aussehen sollten bzw. welche Informationen wie (Text, BoldText, ItalicText, Header, etc.) dargestellt werden sollten.

In den nachfolgenden Listings wird aufgezeigt, wie die Export-Funktion und die Serialize-Methode im Speziellen für das `BrainstormingFinding` im Backend umgesetzt wurde. Die Umsetzung für die anderen Business-Objekte ist entsprechend gleich aufgebaut.

```
1  public class BrainstormingFinding implements MarkdownCascadable {
2  ...
3   public String getPredecessor() {
4       return new Heading(getName(),1).toString() + "\n";
5   }
6
7   ...
8   public String getSuccessor() {
9       StringBuilder successor = new StringBuilder();
10
11      successor.append(new BoldText("Basic Information").toString
            ()).append("\n")
12      .append(new Text("Description: ").toString())
13      .append(getProblemDescription()).append("\n")
14              ...
15      return successor.toString();
```

```
16  }
17
18  ...
19  public String serialize() throws MarkdownSerializationException
        {
20
21      if (getName().equals("") || getNrOfIdeas() == 0 || getType()
            .equals("") || getBrainstormingTeam().equals("")) {
22          throw new MarkdownSerializationException("name is null
                or description");
23      }
24
25      StringBuilder brainsheets = new StringBuilder();
26
27      for (Brainsheet brainsheet: getBrainsheets()){
28          brainsheets.append(brainsheet.serialize());
29      }
30
31      return getPredecessor() + getSuccessor() + brainsheets.
            toString();
32  }
33  }
```

Listing 17: Serialize-Methode von BrainstormingFinding

Im *FindingController* wird nun die `exportBrainstorming`-Methode auf dem Business-Service aufgerufen (Zeile 3).

```
1  public Result exportBrainstorming(...){
2   try {
3    String result = service.exportBrainstorming(findingIdentifier);
4
5       if (result != null) {
6           return ok(result);
7       } else {
8           return badRequest(...);
9       }
10
11  } catch (InterruptedException | ... e) {
12      return internalServerError(Json.toJson(new ErrorMessage("
            Error", e.getMessage())));
13  }
14  }
```

Listing 18: Export-Methode im FindingController

Der Business-Service wiederum holt das *BrainstormingFinding* (Zeile 2) und das *BrainstormingTeam* (Zeile 5), setzt statt der Team-ID den Namen des Teams (Zeile 6) und führt am Ende die erwähnte Serialize-Methode aus (Zeile 7).

```
public String exportBrainstorming(String findingIdentifier) ...{
 BrainstormingFinding finding = service.getFinding(
    findingIdentifier).get();

 if (finding != null) {
     BrainstormingTeam team = teamService.getTeam(finding.
        getBrainstormingTeam()).get();
     finding.setBrainstormingTeam(team.getName());
     return finding.serialize();
 }

 return null;
}
```

Listing 19: Export-Methode im FindingService

Die Tatsache, dass wir neben unseren Business-Objekten DTOs in unsere Applikation integrierten, konnte in dieser Funktion gut ausgenutzt werden. So wurde die Serialize-Methode lediglich in den Business-Objekten implementiert, da sie einen Bestandteil der Business-Logik darstellt und weniger essenziell für die Kommunikation ist.

Implementation Frontend

Der Export auf dem Frontend ist simpel gehalten; der String muss beim Betätigen des 'Export'-Buttons vom Backend geholt werden und ins Clipboard gespeichert werden. Da für das Verwalten des Clipboards unterschiedliche Implementationen für iOS und Android notwendig sind, musste für jede Plattform ein separater Service erstellt werden. Dazu kann der dafür vorgesehene Dependency Service von Xamarin Forms verwendet werden. Listing 20 zeigt die Implementation dieses Services auf der iOS-Plattform.

```
using Method635.App.BL.Interfaces;
using Method635.App.Forms.iOS.Clipboard;
using UIKit;
using Xamarin.Forms;
[assembly: Dependency(typeof(ClipboardService))]
namespace Method635.App.Forms.iOS.Clipboard
{
 public class ClipboardService : IClipboardService
 {
   public void CopyToClipboard(string content)
   {
    UIPasteboard clipboard = UIPasteboard.General;
    clipboard.String = content;
   }
  }
 }
```

Listing 20: Clipboard Service auf iOS

Besonders zu erwähnen ist die Zeile 5 (analog bei Android-Implementation), welche den `ClipboardService` beim Dependency Service von Xamarin Forms registriert.

Da schon ein IoC Container durch das MVVM-Framework Prism Forms verwendet wird, macht es Sinn, nur einen Dependency Service der Applikationslogik mitzugeben. Deshalb wurden alle benötigten Services, die im Xamarin Forms Dependency Service registriert wurden, ebenfalls beim Prism IoC-Container registriert. So werden alle Services nur von einer Quelle aufgelöst.

Im Applikationssetup (`App.xaml.cs`) wird somit der ClipboardService folgendermassen registriert:

```
1 var clipboardService = DependencyService.Get<IClipboardService>()
    ;
2 containerRegistry.RegisterInstance(typeof(IClipboardService),
    clipboardService);
```

So kann in jedem registrierten ViewModel ein Konstruktorparameter vom Typ `IClipboardService` eingeführt werden, dass dann automatisch die für die aktuelle Plattform korrekte Implementation des ClipboardServices enthält. Die einzige Methode auf dem Interface muss dann noch für das Speichern ins Clipboard aufgerufen werden.

2.8.5 Verwendete Bibliotheken im Backend

Tabelle 8 zeigt die Bibliotheken auf, die im Backend verwendet werden.

Bibliothek	Version	Repository	Lizenz
swagger-play2	1.6.0	mvnrepository.com	Apache 2.0
java-jwt	3.2.0	mvnrepository.com	MIT
mongodb-driver-async	3.8.0	mvnrepository.com	MIT
modelmapper	2.3.2	mvnrepository.com	Apache 2.0
markdowngenerator	1.3.1.1	mvnrepository.com	Apache 2.0

Tabelle 8: Verwendete Bibliotheken Backend

2.8.6 Verwendete Bibliotheken im Frontend

In Tabelle 9 sind die verwendeten Libraries und Frameworks des Frontends aufgelistet.

Bibliothek	Version	Repository	Lizenz
CarouselView.FormsPlugin	5.2.0	github.com	MIT
Microsoft.AppCenter	1.10.0	AppCenter	MIT
NUnit	3.12.0	nunit.org	MIT
NUnit3TestAdapter	3.13.0	github.com	MIT
Prism.Forms	7.1.0.431	github.com	MIT
Xamarin.Forms	3.5.0.274416	Microsoft Docs	MIT
XamlStyler.Console	3.0.0	github.com	Apache 2.0
ZXing.Net.Mobile	2.4.1	github.com	Apache 2.0
ZXing.Net.Mobile.Forms	2.4.1	github.com	Apache 2.0
AutoMapper	8.1.1	github.com	MIT
Moq	4.11.0	github.com	BSD 3
Newtonsoft.Json	12.0.2	github.com	MIT
NLog	4.6.4	github.com	BSD 3
SkiaSharp	1.68.0	github.com	MIT

Tabelle 9: Direkt verwendete Bibliotheken Frontend

2.8.7 Vergleich Soll/Ist

Wie aus dem Projektplan (siehe Kapitel B.5) zu entnehmen ist, nahmen wir uns für die Zeit während der Construction vor, ein Refactoring durchzuführen und die Skizzenfunktion (Use-Case 8c), die Patternfunktion (Use-Case 8e) sowie die Exportfunktion (Use-Case 13) zu implementieren.

Obwohl das Refactoring etwas mehr Zeit als zunächst gedacht benötigte, konnten wir die gesetzten Ziele allesamt erreichen. Die restlichen drei Use-Cases aus dem Kapitel 2.4.1.2 waren für uns schon von Beginn weg niedriger priorisiert. Dies aus dem Grund, da diese keine businesskritische Funktion darstellten bzw. durch die anderen Use-Cases teilweise abgedeckt werden konnten (Links können z.B. auch als normaler Text eingegeben werden). Weitere nicht implementierten Features sind: Use-Case 2: Logout auf Back- und Frontend sowie Use-Case 4: Delete Account und Use-Case 6: Leave Brainstorming Team auf dem Frontend. Da wir uns auf die oben erwähnten Arbeiten konzentrieren, war es auch nie ein definiertes Ziel dieser Arbeit, diese umzusetzen.

Die Ziele aus der Aufgabenstellung (siehe Anhang A) wurden wie folgt erreicht. Die Erweiterbarkeit ist durch die Applikation gegeben. Wie im Anhang F beschrieben, ist es ohne grosse Aufwände möglich, die bestehende Applikation um weitere Ideen-Typen zu erweitern. Auch steht der Integration von anderen Brainstorming- und Innovationsmethoden nichts im Wege.

Auch die Ausnutzung der Smartphone-Fähigkeiten bzw. die Verwendung der verschiedenen Medien wurde erweitert. Dank des Touchscreen ist es z.B. möglich Skizzen zu

zeichnen. Die Patternfunktion ermöglicht es zudem ein vordefiniertes Pattern als weiteres, neues Medium zu nutzen.

Das Ziel der einfachen und intuitiven Bedienung der App, einem unkomplizierten Reporting sowie der Robustheit und der Stabilität der App konnten dank den durchgeführten User-Tests validiert und beurteilt werden. Die Resultate dazu finden sich in den Kapitel 2.9.1 und G.5.

2.8.8 Herausforderungen

Hier sind besonders erwähnenswerte Herausforderungen und Hürden beschrieben, die im Verlaufe des Projektes aufkamen. Dies soll anderen Software Ingenieuren oder Interessierten helfen, aus unseren Schwierigkeiten zu lernen.

Navigationsprobleme mit Prism Forms

Die grösste Herausforderung im Frontend war die Navigation mit Prism Forms und Xamarin Forms. Ein zentraler Unterschied liegt in der absolution und relativen Navigation. Bei der absoluten Navigation wird der gesamte Navigationsstack bei jedem Aufruf neu aufgebaut, was zur Folge hat, dass alle beteiligten ViewModels neu erstellt (Konstruktoraufruf) werden und darauf navigiert wird (Aufruf von `OnNavigatedTo` und `OnNavigatedFrom`). Dies im Unterschied zur relativen Navigation, wo jeweils die zu navigierende Page auf den Stack gelegt wird und nur die Methoden auf dem zuständigen ViewModel aufgerufen werden.

Das Problem war nun, dass wir uns im Kontext einer `TabbedPage` bewegen und die Navigationsschritte bei einem Klick auf einem Element in einem Tab auf ein anderes Tab wechseln müssen. Leider funktioniert in diesem Zusammenhang nur die absolute Navigation, weil wir innerhalb einer Page mit den verschiedenen Tabs navigieren wollen und die Navigation von Prism Forms dazu implementiert wurde, zwischen Pages zu navigieren. Dies hat zur Folge, dass jeder Tabwechsel mit absoluter Navigation entwickelt werden musste. Der gesamte Code muss also berücksichtigen, dass die ViewModels mehrmals initialisiert werden und somit Instanzen überschrieben und verändert werden können. Dies ist ein erheblicher Mehraufwand, der immer wieder zu Fehlverhalten und viel zusätzlichem Code führte.

Im späteren Projektverlauf wurde festgestellt, dass der Navigationservice von Prism Forms um eine Methode namens `SelectTab()` erweitert wurde [15], was die beschriebene Problematik lösen könnte. Jedoch ist diese Extensionmethode noch nicht im stabilen Release von Prism Forms und daher noch nicht nutzbar. In einer Folgearbeit ist es sehr empfehlenswert, diese Funktionalität zu testen und zu verwenden.

2.9 Schlussfolgerungen

Im Kapitel der Schlussfolgerungen wollen wir nochmals auf unser Projekt und dessen Verlauf schauen und unsere Ergebnisse kritisch bewerten. Dabei wollen wir aufzeigen, was wir in dieser Zeit erreicht haben, aber auch an welchen Stellen es noch Verbesserungspotenzial gibt. Ausserdem soll das Kapitel 2.9.3 veranschaulichen, was das weitere Vorgehen für dieses Projekt sein könnte.

2.9.1 Ergebnisbewertung

Als der Teil nicht-funktionalen Anforderungen im Kapitel 2.4.2 haben wir sogenannte Landing Zones für die in diesem Projekt wichtigen SMART-Kriterien definiert. Wir wollen hier nochmals speziell Bezug nehmen auf diese Ziele und kritisch beurteilen, wie gut diese in unserem Projekt umgesetzt wurden.

Mehrere User-Tests haben gezeigt, dass das Zeitverhalten bzw. die zeitkritische Kommunikation zwischen Server und App in einem akzeptablen Rahmen von geschätzt einer Sekunde liegt. Dies kommt aber auch auf die Netzwerkkonnektivität an.

Wir beurteilen dieses Kriterium als 'gut' erfüllt (vgl. Kapitel 2.4.2).

In Punkto Erweiterbarkeit sind wir der Meinung, dass sich noch viele weitere Ideenarten für eine mögliche Integration eignen würden. Im Kapitel 2.9.3 haben wir dafür drei konkrete Ideenarten aufgeschrieben. Zudem gibt es Ideenarten, wie z.B. die PictureIdea oder VideoIdea, welche wir zwar zu Beginn dieser Arbeit definiert hatten aber mangels Zeit nicht umsetzen konnten.

Das Ziel der Erweiterbarkeit sehen wir daher als 'übertroffen' an (vgl. Kapitel 2.4.2).

Dank der Anleitung in Anhang F sind wir zudem der Ansicht, dass sich die Applikation innert wenigen Tagen um eine neue Ideenart erweitern liesse. In Zahlen ausgedrückt, sollte sich solch ein Vorhaben schätzungsweise innert zwei Tagen realisieren lassen.

Das Ziel der Modularität ist somit 'gut' erreicht (vgl. Kapitel 2.4.2).

Die Frage der Ästhetik ist als Kriterium eher schwierig zu beurteilen, zumal jede Person wahrscheinlich etwas andere Vorstellungen diesbezüglich hat. Dennoch gehen wir davon aus, dass es für die Mehrheit der Endnutzer eine nachhaltige Anziehungskraft hätte. An einem der zwei durchgeführten User-Test wurde sogar explizit erwähnt, dass die App ansprechend aussieht. Die User-Tests haben aber auch aufgezeigt, dass die App in Sachen Orientierung und die Usability noch Potenzial für Verbesserungen aufweist.

Die Ästhetik der App beurteilen wir insgesamt als 'gut' (vgl. Kapitel 2.4.2).

Den Grad der Ausgereiftheit ist unserer Meinung nach als 'genügend für den produktiven Einsatz' zu beurteilen. Auch hier haben User-Tests gezeigt, dass es, trotz der massiven Verbesserungen zur Studienarbeit, immer noch vorkommen kann, dass die Applikation aus unbekannten Gründen abstürzt. In solch einem Fall ist es aber ohne Probleme möglich, sich wieder einzuloggen und weiterzufahren. Dem produktiven Einsatz steht somit nichts im Wege.

Das Ziel der Ausgereiftheit ist somit auch als 'gut' zu beurteilen (vgl. Kapitel 2.4.2).

Kriterium	Minimum	Gut	Übertroffen
Zeitverhalten Zeitkritische Kommunikation (Abgabe von Ideen/Brainsheets) zwischen Server und App beträgt	2 Sekunden	1 Sekunden	weniger als 1 Sekunde
Erweiterbarkeit Die Anzahl an neuen Ideenarten, welche mit der bestehenden Architektur als umsetzbar gelten, wird als ... angesehen	zu wenig (0-1)	ausreichend (1-5)	unbegrenzt (>5)
Modularität Die App ist innert ... Tagen um eine neue Ideenart erweitert	5 Tage	2 Tage	weniger als 1 Tag
Ästhetik Die Anziehungskraft gegenüber dem Endnutzer wird als ... charakterisiert	ausreichend für eine kritische Masse an Nutzern	nachhaltig	tägliche Nutzung in kritischen Projekten
Ausgereiftheit Der Grad der Ausgereiftheit oder Reife wird als ... angesehen	ungenügend für den produktiven Einsatz (Prototypen-Stadium)	genügend für den produktiven Einsatz, kann aber immer noch in Fehlerzustände gelangen	kaum Fehlerzustände und somit keine Abstürze der App

2.9.2 Bekannte Probleme

Dieses Kapitel dient dazu Fehlverhalten in unserem System zu dokumentieren.

Falscher Zustand nach erstmaligem Auswählen eines beendeten Brainstormings

Wird nach einem beendeten Brainstorming zurück in die Übersicht aller Brainstormings gewechselt und das durchgeführte ausgewählt, wird der Zustand der Statemachine falsch eruiert (`RunningState` anstatt `EndedState`). Dies führt dazu, dass der Timer weiterläuft und schlussendlich ins Negative zählt. Dies hängt damit zusammen, dass der `BrainstormingService` durch das mehrmalige Aufrufen der ViewModels aufgrund des Navigationsproblems (siehe Kapitel 2.8.8) im Kontextobjekt veraltete Werte enthält, welche in der Berechnung der Statemachine in einem falschen Zustand resultieren.

Security

Obwohl das Thema Security kein eigentliches Problem darstellt, soll an dieser Stelle erwähnt werden, dass wir kein spezielles Augenmerk

auf dieses Thema geworfen haben. Bei einer allfälligen Folgearbeit wäre es daher sinnvoll, dieses Thema stärker in den Fokus zu rücken und notwendige Anpassungen an der bestehenden Applikation vorzunehmen. Auch wäre es ratsam sich Gedanken für den Zugriff von Skizzen zu machen, welche möglicherweise als intern oder schützenswert zu klassifizieren sind.

2.9.3 Ausblick

Da die Ausbaumöglichkeiten dieses Projektes sehr vielfältig sind, empfinden wir es als sehr lohnenswert diese Applikation zu erweitern.

Dazu sehen wir drei mögliche Arten oder Wege, wie die bestehende Applikation verbessert bzw. erweitert werden könnte. Diese wären weitere Ideen-Typen zu integrieren, die erarbeiteten Ideen stärker miteinander zu referenzieren oder die Methode 635 mit der World-Cafe Methode zu kombinieren.

Weitere Ideen-Typen

Die einfachste und naheliegenste Erweiterung könnte darin bestehen, die Applikation um weitere Ideen-Typen zu erweitern. Die gesamte Applikation ist so programmiert, dass dies mit möglichst wenig Aufwand machbar ist (siehe Anhang F).

Dabei könnten wir uns vorstellen, dass sich Ideen-Typen wie QualityIdea (angelehnt an nicht-funktionale Anforderungen), RequirementIdea (angelehnt an funktionale Anforderungen) oder CodeIdea gut umsetzen liessen.

Ideen stärker referenzieren

Bei verschiedensten Durchführungen der Methode 635 auf dem Papier, kam immer wieder die Frage auf, wie ich den anderen Teilnehmern mitteilen kann, auf welche Idee ich gerade Bezug nehme. Dies ist in der vorliegenden Applikation nicht möglich bzw. nur indem man sich darauf einigt, dass die Reihenfolge der Ideen ausschlaggebend ist.

Eine weitere Verbesserung würde demnach darin bestehen, dem Endnutzer die Möglichkeit zu bieten seine Ideen stärker mit den vorliegenden Ideen zu 'verknüpfen' oder zu referenzieren. Dies würde auch das Problem der oben geschilderten Unsicherheit lösen.

Methode World-Cafe

Als weitere mögliche Erweiterung ist es vorstellbar, die Methode 'World-Cafe' [16] in die bestehende Applikation zu integrieren. Im Gegensatz zur Methode 635 sind hier nicht einzelne Teilnehmer, welche ihre Ideen zu einem bestimmten Problem schildern, sondern ganze Gruppen.

Das Konzept von World-Cafe sieht daher vor, dass man sich in einer Gruppe zu einer bestimmten Fragestellung austauscht. Ist eine vordefinierte Zeitspanne abgelaufen, wechseln die Teilnehmer zu einer anderen Gruppe, um dort wieder eine andere oder auch die gleiche Fragestellung zu diskutieren.

Um die World-Cafe Methode in die bestehende Applikation zu integrieren, ist es an dieser Stelle aber ratsam von der ursprünglichen Version abzuweichen und die Gruppen

immer gleich zu belassen.

Das hätte zur Folge, dass mehrere Teams (statt mehrere Participants) gemeinsam ein BrainstormingFinding erarbeiten. Der Umstand, dass mehrere Teams ein BrainstormingFinding lösen, führt dazu, dass innerhalb eines Teams schneller verschiedenste Ideen entstehen, als wenn ein Participant alleine Ideen/Lösungsansätze entwickelt.

Dies würde der gesamten Applikation nochmals einen Mehrwert bieten.

Literatur

[1] E. B. Oliver Dias, "Methode 635 als cross plattform app mit xamarin." https://eprints.hsr.ch/740/1/Cross-Platform-App.pdf, December 2018. Accessed on 2019-03-05.

[2] P. H. . R. D. M. Gernot Starke, "arc42 template for architecture communication and documentation." https://docs.arc42.org/section-4/, December 2018. Accessed on 2019-02-26.

[3] G. Starke, *Effektive Softwarearchitekturen*. München: Hanser Fachbuch, 2018.

[4] U. Z. D. L. . C. P. Olaf Zimmermann, Mirko Stocker, "Microservice api patterns." https://www.microservice-api-patterns.org, February 2019. Accessed on 2019-02-26.

[5] M. Inc., "Gridfs." https://docs.mongodb.com/manual/core/gridfs/, March 2019. Accessed on 2019-03-05.

[6] M. Inc., "Gridfs." http://mongodb.github.io/mongo-java-driver/3.8/driver-async/tutorials/gridfs/, March 2019. Accessed on 2019-03-05.

[7] C. Menge, "Storing (small) images in mongodb." http://menge.io/2015/03/24/storing-small-images-in-mongodb/#chunking, March 2015. Accessed on 2019-03-05.

[8] Wikipedia, "Iso 9126." https://de.wikipedia.org/wiki/ISO/IEC9126, Mai 2018. Accessed on 2018-09-25.

[9] Johner-Institut, "Iso 25010 und iso 9126." https://www.johner-institut.de/blog/iec-62304-medizinische-software/iso-9126-und-iso-25010/, August 2015. Accessed on 2018-09-25.

[10] M. Kops, "Qualitaet, funktionale und nichtfunktionale anforderungen in der software-entwicklung." https://blog.seibert-media.net/blog/2018/05/14/qualitaet-funktionale-und-nichtfunktionale-anforderungen-in-der-software-entwicklung/, Mai 2018. Accessed on 2018-09-25.

[11] Wikipedia, "Smart criteria." https://en.wikipedia.org/wiki/SMART_criteria, August 2018. Accessed on 2018-09-27.

[12] Tutorialspoint, "Design pattern transfer object pattern." https://www.tutorialspoint.com/design_pattern/transfer_object_pattern, March 2019. Accessed on 2019-03-22.

[13] Microsoft, "Finger painting in skiasharp - xamarin." , June 2019. Accessed on 2019-06-04.

[14] J. Gruber, "Markdown." https://daringfireball.net/projects/markdown/, May 2019. Accessed on 2019-05-02.

[15] B. L. . D. Siegel, “Inavigationserviceextensions.” , June 2019. Accessed on 2019-06-06.

[16] T. W. Cafe, “World cafe method.” http://www.theworldcafe.com/key-concepts-resources/world-cafe-method/, March 2019. Accessed on 2019-03-21.

[17] Microsoft, “Framework design guidelines.” https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/, Dezember 2017. Accessed on 2018-09-18.

[18] M. Kuhrmann, “Rational unified process.” http://www.enzyklopaedie-der-wirtschaftsinformatik.de, Juli 2018. Accessed on 2018-10-16.

[19] S. Master, “Scrum - auf einer seite erklaert.” https://scrum-master.de/WasistScrum/ScrumaufeinerSeiteerklaert. Accessed on 2018-10-16.

[20] P. M. Institute, “Project management - how much is enough?.” , June 2019. Accessed on 2019-06-11.

[21] Apple, “Beta testing made simple with testflight.” https://developer.apple.com/testflight/, May 2019. Accessed on 2019-05-14.

[22] Wikipedia, “Testflight.” https://en.wikipedia.org/wiki/TestFlight, May 2019. Accessed on 2019-05-14.

Abbildungsverzeichnis

Listings

A Aufgabenstellung

Aufgabenstellung Bachelorarbeit
Elias Brunner, Oliver Dias-Lalcaca

Solution Strategy mit der Methode 635 als Cross-Plattform App

1. Auftraggeber und Betreuer

Diese Bachelorarbeit wird in Zusammenarbeit mit dem Institut für Software (IFS) durchgeführt.

Betreuer HSR:

Prof. Dr. Olaf Zimmermann, HSR FHO, Partner Institut für Software, ozimmerm@hsr.ch

2. Ausgangslage

Die papiergestützte Methode 635 ist eine Kreativitäts- und Brainwriting-Technik, für die es bisher noch keine Unterstützung in mobilen Apps gab. Eine Studienarbeit im Herbstsemester 2018/2019 konzipierte und implementierte daher einen ersten Prototyp einer SmartPhone App für die Methode 635; Cross Platform Support (Android- und iOS) wurde durch Verwendung von Xamarin erreicht. Ein Vorteil einer derartigen mobilen Anwendung ist, dass Anwender die Methode 635 nutzen können, wenn sie sich nicht ständig an einem Ort befinden.

In dieser Bachelorarbeit sollen weitere Features implementiert werden auf Basis der Testergebnisse für den bestehenden Prototypen. Im Fokus steht dabei insbesondere die Software Engineering Aktivität "Solution Strategy" aus dem Buch „Effektive Softwarearchitekturen" von G. Starke, die in der HSR-Vorlesung Application Architecture behandelt wird. Diese Aktivität erfordert ein hohes Mass an kollaborativen Problemlösungsprozessen auf unterschiedlichen Abstraktionsebenen (z.B. Konzepte, Technologien, und Produkte). Zum Beispiel können Ideenfindungssessions zu folgenden Themen unterstützt werden:

- Welche Muster eignen sich, um die Softwarelösung anforderungsgerecht zu strukturieren?
- Mit welchen Technologien und Produkten können die Qualitätsziele erreicht werden?
- Welche Notationen sollen zur Ergebnissicherung eingesetzt werden?

Derartige Solution Strategy Aktivitäten kennt man auch in anderen fachlichen Domänen, in denen so genannte Wicked Problems gelöst werden müssen. Auf Konfigurier- und Erweiterbarkeit der Methode-635-App soll daher geachtet werden.

3. Ziele der Arbeit und Liefergegenstände

Die Vision der Arbeit ist es, die Papierversion für diese Methodik zu funktional und qualitativ zu überbieten. Damit die erweiterte App einen Mehrwert gegenüber der Papierversion bietet, soll es z.B. möglich sein, verschiedene Medien (Text, Video, Bilder, etc.) zu verwenden bzw. einzubinden. Hardware-Features und andere Apps sollen – wenn sinnvoll und technisch möglich – integriert werden.

Die übergeordneten Projektaktivitäten und Liefergegenstände sind:

1. Lauffähiger erweiterter Prototyp, der idealerweise ausserhalb der Entwicklungsumgebungen und Testumgebungen verfügbar ist (Bsp. App Store).
2. Demo-Konfiguration und Beispielinhalte, die die Verwendung der App im Szenario Software (Architecture) Solution Strategy demonstrieren, z.B. Nutzung der Microservices API Patterns von https://microservice-api-patterns.org/.
3. Installationsanweisungen und Benutzerdokumentation
4. Technischer Bericht

Bei der Umsetzung spielen Erfolgsfaktoren wie einfache und intuitive Bedienung der App und ein unkompliziertes Reporting sowie Robustheit und Stabilität (Bsp. keine Zeit- und Datenverluste) eine wichtige Rolle. Ein wichtiger Input für die Bachelorarbeit ist das externe Feedback zu den Ergebnissen der vorangegangenen Studienarbeit, zum Beispiel Feedback zur Usability und zur Codequalität.

Weitere kritische Erfolgsfaktoren sind:

- Konfigurierbarkeit und Erweiterbarkeit (im Hinblick auf Folgearbeiten, die u.U. auch andere Brainstorming- und Innovationsmethoden wie z.B. Design Thinking unterstützen).
- Sinnvolle Ausnutzung der Smartphone-Fähigkeiten, um einen Mehrwert im Vergleich zur traditionellen, papiergestützten Methode zu erreichen.
- Validierung der Konzepte und ihrer Implementierung mit Hilfe von User Tests in mindestens einem Anwendungsbereich (Bsp. Architekturentscheidungen und -optionen).

4. Unterstützung

Die erwartete und effektiv erhaltene Unterstützung wird durch die Studenten in Sitzungsprotokollen definiert und im SA-Bericht dokumentiert.

5. Zur Durchführung

Mit dem HSR-Betreuer finden in der Regel wöchentlich Besprechungen statt (Treffen an der HSR oder Telefon- bzw. Webkonferenz). Zusätzliche Besprechungen sind nach Bedarf zu veranlassen.

Alle Besprechungen, bei denen eine Vorbereitung durch den Betreuer nötig ist, sind von den Studenten mit einer Traktandenliste vorzubereiten. Beschlüsse sind in einem Protokoll zu dokumentieren.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. Arbeitszeiten sind zu dokumentieren.

Die Spezifikation der Anforderungen geschieht durch die Studenten in Absprache mit dem Betreuer. Bei Disputen entscheidet der Betreuer in Rücksprache mit den Studenten über die definitiv für die Bachelorarbeit relevanten Anforderungen.

Vorstudie, Anforderungsdokumentation und Architekturdokumentation sollten im Laufe des Projektes mittels Milestone mit dem Auftraggeber und dem Betreuer in einem stabilen Zustand abgenommen werden. Zu den abgegebenen Arbeitsresultaten wird ein vorläufiges Feedback

abgegeben. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

Die Rechte an den Ergebnissen der Bachelorarbeit werden in einer separaten Vereinbarung definiert (keine Einschränkungen).

6. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Bei der Projektdokumentation und Ihrer Abgabe sind die „Allgemeine Informationen zu Studien- und Bachelorarbeiten" sowie die „Anleitung: Dokumentation Studien- und Bachelorarbeite" inklusive Anhängen zu beachten.

7. Termine (Quelle: HSR-Intranet)

18.02.2019	Beginn der Bachelorarbeit, Ausgabe der Aufgabenstellung durch die Betreuer
noch nicht bekannt	Fotoshooting. Genauere Angaben erteilt die Kommunikationsstelle rechtzeitig.
07.06.2019	Die Studierenden geben den Abstract für die Diplomarbeitsbroschüre zur Kontrolle an ihren Betreuer frei. Die Studierenden senden per Email das A0-Poster zur Prüfung an ihren Betreuer.
12.06.2019	Abgabe Kurzbeschreibung (Abstract für Diplomarbeitsbroschüre) an das Abteilungssekretariat (seitens Betreuer).
14.06.2019	Abgabe des Berichtes an den Betreuer bis 12.00 Uhr. Fertigstellung des A0-Posters bis 12.00 Uhr. Fertigstellung und Weitergabe des A0 Posters per Email bis 12.00 Uhr an das Studiengangsekretariat.
spätestens in der Beratungswoche	Mündliche BA-Prüfung

Allfällige weitere Termine sind am Sekretariat der Abteilung Informatik zu erfragen und sollten entsprechend in einem Sitzungsprotokoll dokumentiert werden.

8. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS-Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Siehe auch Modulbeschreibung der Bachelorarbeiten, http://studien.hsr.ch/allModules/24809_M_BAI14.html.

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/6
2. Berichte (Abstract, Management Summary, technische u. persönliche Berichte) sowie Gliederung, Darstellung und Sprache der gesamten Dokumentation.	1/6
3. Inhalt*)	3/6
4. Mündliche Prüfung zur Bachelorarbeit	1/6

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit festgelegt.

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Bachelorarbeiten.

Rapperswil, den 22. 02. 2019

Prof. Dr. Olaf Zimmermann
Institut für Software
Hochschule für Technik Rapperswil

B Projektplan

Zweck dieses Dokuments

Dieses Dokument beschreibt den Projektplan des Projekts «Solution Strategy mit der Methode 635 als Cross Plattform App». Es beinhaltet die Planung, die Organisation sowie weitere Aspekte und liefert damit eine Übersicht über das Projekt. Es dient daher als Grundlage für den Verlauf des Projekts.

Gültigkeitsbereich

Der Gültigkeitsbereich erstreckt sich über die gesamte Dauer des Projekts. Der Zeitraum geht vom 18. Februar 2019 bis zum 14. Juni 2019. Das Projekt findet im Rahmen des Moduls «Bachelorarbeit» im Frühlingssemester 2019 statt.

Verweise

An dieser Stelle ist noch zu anzumerken, dass einzelne Textstellen von eigenen, älteren Projektplänen verwendet wurden. Dabei handelt es sich um Projektpläne von Engineering-Projekten oder Studienarbeiten.

Im Speziellen zu erwähnen ist die Tatsache, dass dieser Bachelorarbeit eine Studienarbeit («Methode 635 als Cross Plattform App mit Xamarin») vorherging. Es kann daher sein, dass in einigen Textstellen auf diese Studienarbeit verwiesen wird.

B.1 Projektziel

Die papiergestützte Methode 635 ist eine Kreativitäts- und Brainwriting-Technik, für die es bisher noch keine Unterstützung in mobilen Apps gab. Eine Studienarbeit im Herbstsemester 2018/2019 konzipierte und implementierte daher einen ersten Prototyp einer SmartPhone App für die Methode 635; Cross Platform Support (Android- und iOS) wurde durch Verwendung von Xamarin erreicht. Ein Vorteil einer derartigen mobilen Anwendung ist, dass Anwender die Methode 635 nutzen können, wenn sie sich nicht ständig an einem Ort befinden.

In dieser Bachelorarbeit sollen weitere Features implementiert werden auf Basis der Testergebnisse für den bestehenden Prototypen. Im Fokus steht dabei insbesondere die Software Engineering Aktivität SSolution Strategyäus dem Buch „Effektive Softwarearchitekturen“ von G. Starke, die in der HSR-Vorlesung Application Architecture behandelt wird.

Mehr zum Thema Projektziel ist dem Anhang A zu entnehmen.

Einschränkungen

Dieses Projekt ist auf die Dauer des Frühlingssemesters 2019 begrenzt (bis 14. Juni 2019). Der Gesamtaufwand für diese Arbeit sollte 360 Arbeitsstunden pro Person (gesamthaft 720 Stunden) nicht übersteigen. Bleibt am Ende Zeit übrig, werden optionale Features implementiert.

B.2 Projektorganisation

In unserem Projekt arbeiten wir in einer flachen Organisationsstruktur, wobei die wesentlichen Entscheide im ganzen Projektteam und/oder mit dem Dozenten an den wöchentlichen Besprechungen getroffen werden. An den Besprechungen getroffene Entscheidungen werden in Protokollen dokumentiert. Die Projektmitglieder sind innerhalb des Teams gleichgestellt.

Organisationsstruktur

Die Projektmitglieder sowie deren Verantwortung sind der Abbildung 14 zu entnehmen.

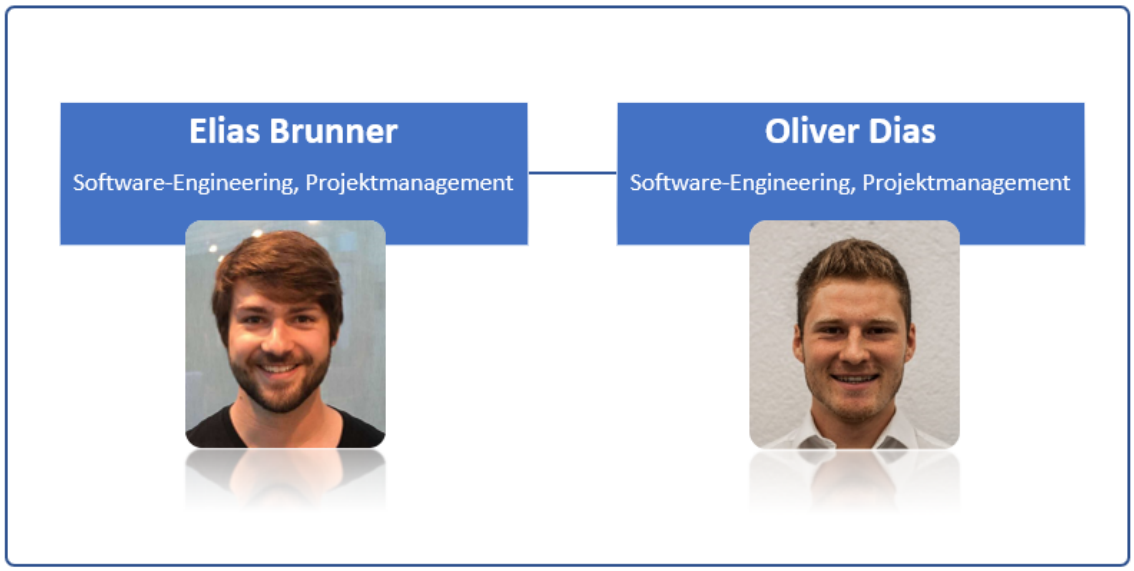

Abbildung 14: Methode 635 Organisation

Ansprechspartner

Da diese Arbeit eine Weiterführung der Studienarbeit darstellt, bleibt der bisherige Ansprechpartner bestehen.

Betreuer Prof. Dr. Olaf Zimmermann ist der betreuende Dozent für diese Bachelorarbeit. Er ist neben der Betreuung auch für die Bewertung des Projekts verantwortlich.

B.3 Projektmanagement

Eine ausführliche Iterationsplanung mit den dazugehörigen Meilensteinen befindet sich auf Jira. Die aufgewendeten Zeiten für ein Issue werden ebenfalls dort erfasst.

Dokumentenplan

Wie das Projekt selbst, werden auch sämtliche Dokumente ständig überarbeitet und angepasst. Grundsätzlich umfasst der Dokumentenplan aber folgende Bestandteile:

Projektplan	Der Projektplan umfasst den groben Ablauf des Projektes und legt unter anderem das Vorgehen bei der Entwicklung fest oder wie mit Risiken umgegangen wird.
Anforderungsspezifikation	In der Anforderungsspezifikation werden funktionale sowie nicht-funktionale Anforderungen definiert.
Architekturdokumentation	Hier wird dokumentiert wie die Architektur des gesamten Systems aufgebaut ist.
Architekturentscheide	Hier wird aufgezeigt warum wir uns für diese Architektur entschieden haben.
Installationsanleitung	In der Installationsanleitung wird gezeigt, wie die Applikation auf dem eigenen Smartphone installiert werden kann.

Besprechungen

Das Projektteam trifft sich einmal in der Woche, um sich über den aktuellen Stand des Projekts auszutauschen, Fragen zu klären, Probleme anzugehen oder die nächsten Schritte zu planen.

Diese wöchentlichen Besprechungen finden, falls nicht anders vorgesehen, jeden Dienstagmorgen um 09:00 Uhr statt.

Über jede Besprechung wird Protokoll geführt. Dies mit dem Ziel, die Entscheidungen festzuhalten und Missverständnisse zu vermeiden.

Umgang mit Risiken

Um auch auf unbekannte Risiken vorbereitet zu sein, ist am Ende des Projektes eine Reserve eingeplant. Zudem haben sich beide Teilnehmer bereit erklärt ihr Engagement punktuell zu erhöhen, falls die Situation dies erfordert. Diese Erhöhung sollte jedoch nur Phasenweise erfolgen und in einer folgenden Phase kompensiert werden.

Die häufigsten Risiken wurden mit einer Risikotabelle (Unterkapitel B.6) berücksichtigt, die aktuell gehalten wird und beim Planen miteinbezogen gezogen wird.

Qualitätsmassnahmen

Das Endprodukt dieses Projekts soll von möglichst hoher Qualität sein. Wie in Tabelle 10 zu sehen ist, treffen wir folgende Massnahmen, um diese Qualität zu erreichen.

Massnahme	Zeitraum	Ziel
Meeting im Team und mit Betreuer	Jede Woche	Projektstand aufzeigen, allfällige Probleme möglichst früh erkennen.
Code Reviews	Bei jedem Pull Request	Die Qualität des Codes wird durch die Einhaltung der Code Style Guidelines verbessert.

Tabelle 10: Massnahmen

Arbeitspakete

Die gesamte Arbeit ist in Arbeitspakete unterteilt, die auf Jira verwaltet werden. Dabei sind relevante Informationen wie die Komplexität, Dauer, der damit verbundene Epic und die Unterteilung in Arbeitskategorie erfasst.

Für die Abschätzung der Komplexität der Arbeiten verwenden wir **Story Points**. Dabei einigen wir uns auf folgendes Schema:

Story Points	Bedeutung
1-3	Niedrige Komplexität
4-6	Mittlere Komplexität
7-9	Komplexe bis sehr komplexe Arbeit

Tabelle 11: Story Points Komplexität

Das Unterteilen in drei Punkte pro Komplexitätsgrad ermöglicht ein genaueres Abschätzen innerhalb des Komplexitätsgrades. Story Points können nicht direkt in zeitlichen Aufwand umgerechnet werden. Für den benötigten Zeitaufwand existiert ein separates Feld.

Um die Pakete logisch unterteilen zu können, existieren Arbeitskategorien. Diese werden mit Labels auf den Tickets markiert und können folgende Werte annehmen:

ProjektManagement Alle Aufgaben, die im Zusammenhang mit Projektmanagement stehen, zum Beispiel das Risikomanagement.

Planung Planungsaufgaben. Zum Beispiel steht jede Woche ein Teammeeting an, welches dieser Kategorie zugeordnet ist.

Dokumentation Arbeiten, welche für Dokumentation des Projektes gemacht werden.

Infrastruktur Diejenigen Arbeitspakete, die für die Entwicklung und für den Betrieb des Projekts notwendig sind. Ein Beispiel dafür kann das Einrichten eines Codequalitätstools sein.

Entwicklung Arbeitspakete welche mit der Programmierung der Applikation in Zusammenhang stehen.

Testing Arbeitspakete wie zum Beispiel das Schreiben von Testfällen kann dieser Arbeitskategorie zugewiesen werden.

Design Hierzu zählen Arbeitspakete wie das Ausarbeiten von Benutzeroberflächen.

Analyse Typische Analyseaufgaben ist zum Beispiel das Recherchieren für bestimmte Frameworks, Bibliotheken, etc.

Eingesetzte Werkzeuge

Um ein gutes Arbeiten zu ermöglichen, stehen viele Tools zur Verfügung, die im Folgenden beschrieben sind. Die primäre Entwicklungsumgebung ist Visual Studio und Visual Studio for Mac.

B.4 Entwicklung

Der Entwicklungscode wird in öffentlichen Github Repositories unter der Organisation **BrainingOutOfBox** gehalten. Für alle einzelnen Teile des Projekts gibt es ein eigenes Repository.

Doc Dieses Repository enthält alle relevanten Dateien, welche für die Dokumentation von Relevanz sind.

App Dieses Repository enthält den gesamten Code für die Xamarin Applikation.

API Dieses Repository enthält den gesamten Code für das Play-Backend.

Vorgehen bei der Entwicklung

Jedes Teammitglied verfügt über eine lokale Kopie der Repositories von Github. Für jede Aufgabe/Issue wird ein eigener Branch erstellt. Darin werden die Änderungen für diese Aufgabe vorgenommen. Die Änderungen sollen mit sinnvollen und präzisen Commit-Notizen festgehalten werden. Um ein Tracking der Änderung möglichst effizient zu gestalten, gilt es möglichst früh, möglichst viel zu commiten.

> "Commit early and commit often"

Dies wurde uns so in den Modulen SE1 und SE2 vermittelt.

Code Guidelines

Da Xamarin auf .Net bzw. C# aufbaut, werden die Code Guidelines von .Net verwendet. [17]

Builden und testen der App

Für das automatisierte Builden und Testen nach einem Commit wird auf Visual Studio App Center von Microsoft gesetzt. Zum einen ermöglicht es eine einfache Integration von GitHub und zum anderen bringt es alles mit, um Xamarin Apps automatisch zu builden, testen und deployen.

B.5 Zeitliche Planung

Phasen

In unserem Projekt verwenden wir die vier Phasen von Rational Unified Process (RUP) [18] mit dem Zusatz einer Testing Phase, welche in RUP nicht vorgesehen ist. Innerhalb der einzelnen Phasen wenden wir allerdings das Konzept von SCRUM [19] an.

Inception Phase In der Inception Phase geht es hauptsächlich darum, Jira einzurichten und den Projektplan zu schreiben. Diese Phase beinhaltet den Sprint Inception-1.

Elaboration Phase Die Elaboration Phase umfasst den Sprint Elaboration-1. Hier werden die Anforderungen an die Applikation definiert, den Proof-of-Concept für die Skizzenfunktion durchgeführt und weitere konzeptionelle Arbeiten erledigt. Am Ende der Phase müssen alle konzeptionellen Fragen geklärt sein.

Construction Phase In der Phase der Construction geht es an die Umsetzung der Applikation. Hierfür sind 5 Sprints eingerechnet. Dafür stehen die Sprints Construction-0, Construction-1, Construction-2, Construction-3, Construction-4 zur Verfügung.

Testing In der Testing Phase bzw. im Sprint Testing-1 wird die gesamte Applikation einem User Test unterzogen.

Transition Phase In der letzen Phase werden alle abzugebende Dokumente nochmals durchgelesen und allenfalls überarbeitet. Dies geschieht im Sprint Transition-1.

Sprints

Ein einzelner Sprint dauert zwei Wochen. So stehen pro Woche insgesamt 48 Stunden für die Bachelorarbeit zur Verfügung. In unserem Projekt sind folgende Sprints geplant:

Sprint	**Arbeiten**
Inception-1	▪ Projektplan erstellen ▪ Jira einrichten ▪ einzelne Risikos einschätzen
Elaboration-1	▪ Vorstudie zu einzelnen Themen durchführen ▪ Domainmodell überarbeiten ▪ NFAs sowie FAs überarbeiten ▪ Mockups für zusätzlich benötigte Oberflächen zeichnen ▪ PoC für Skizzenfunktion erstellen
Construction-0	▪ Refactoring Frontend ▪ Statemachine für Frontend ▪ Performance Refactoring Frontend ▪ Refactoring Backend ▪ Dokumentation schreiben
Construction-1	▪ Refactoring Frontend ▪ Performance Refactoring Frontend ▪ Refactoring Backend ▪ PoC für Exportfunktion Backend ▪ PoC für weitere Ideentypen ▪ Dokumentation schreiben
Construction-2	▪ Refactoring Frontend ▪ Skizzenfunktion Backend ▪ Skizzenfunktion Frontend ▪ Patternfunktion Backend

Construction-3	▪ Exportfunktion Backend ▪ Skizzenfunktion Frontend ▪ Patternfunktion Frontend ▪ Dokumentation schreiben
Construction-4	▪ Exportfunktion Frontend ▪ iPhone und Android deployment ausserhalb der IDE ▪ Bugfixing ▪ User Test durchführen ▪ Dokumentation schreiben
Testing-1	▪ Bugfixing ▪ User Test durchführen ▪ Dokumentation schreiben
Transition-1	▪ Endarbeiten Dokumentation

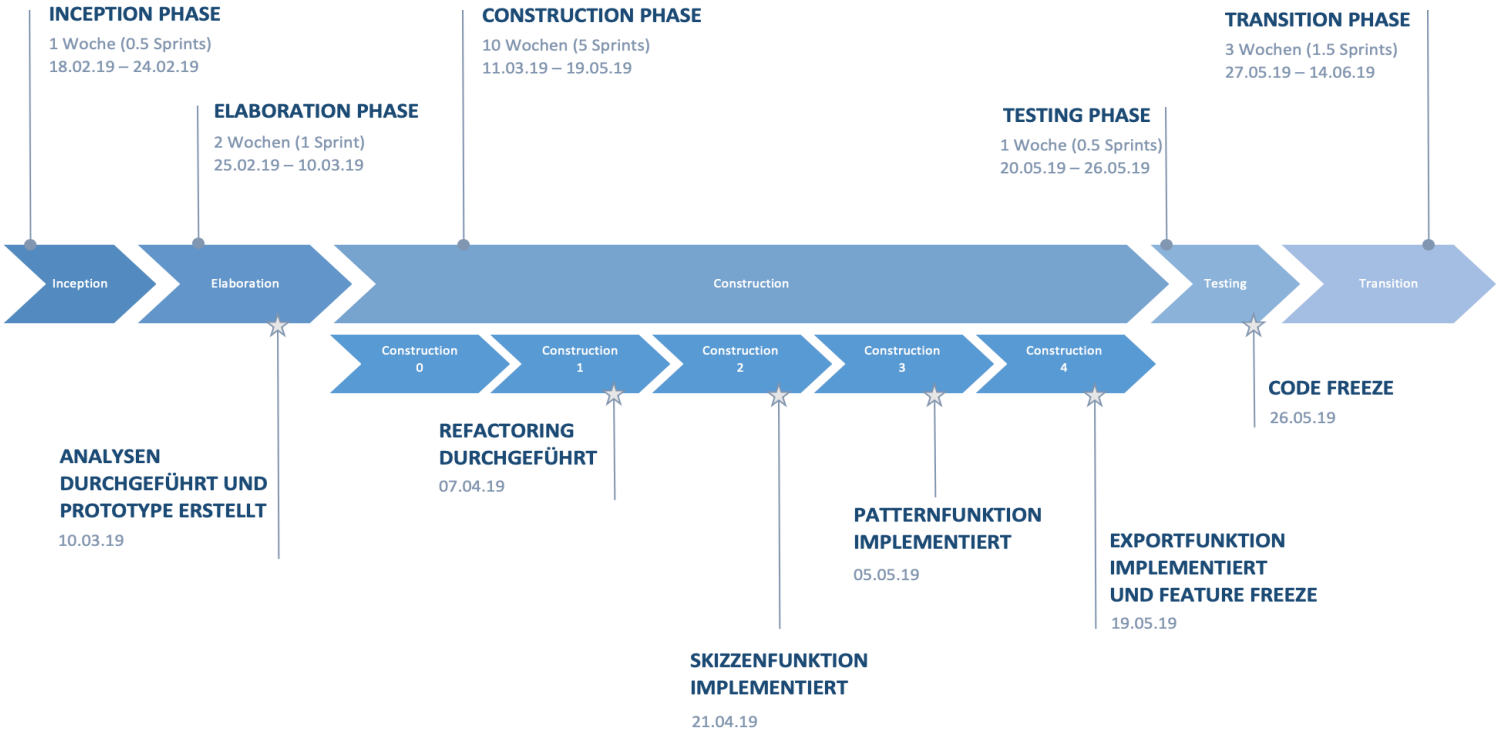

Abbildung 15: Projektplan

B.6 Risikotabelle

Risikomanagement

Projekt: Solution Strategy mit der Methode 635 als Cross-Plattform App
Erstellt am: 21/02/2019
Autor: Elias Brunner
Gewichteter Schaden: 50,5

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahr-scheinlich-keit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Anforderungen	Anforderungen werden zu ungenau Beschrieben. Es ist nicht bekannt wie lange die Umsetzung benötigt.	20	20%	4	Genaue Definition und Analyse während der Elaboration und gute Planung der Iterationen.	Änderungen einschätzen, überprüfen und eventuell vornehmen.
R2	Managment Tool	Arbeiten mit Jira zeitaufwändiger als erwartet. Einzelne Add-Ons bereiten Probleme.	20	20%	4	Früh genug mit Jira und verwendeten Add-Ons vertraut machen.	In Foren nach Lösung suchen oder anderes Add-On verwenden.
R3	Kommunikation	Informationen werden den beiden Teammitgliedern nicht genug präzise mitgeteilt.	15	15%	2,25	Frühzeitig abklären und genau mitteilen, wer was übernimmt.	Zusätzliche Meetings um Ungenauigkeiten abzuklären.
R4	Komplexität	Die Komplexität der Module und Funktionen wurde unterschätzt. Der effektive Zeitaufwand übersteigt die Planung um ein Vielfaches.	40	25%	10	Genaue Abschätzung der Komplexität mittels Story Points.	Rücksprache mit Betreuer über weiteres Vorgehen. Allenfalls Funktionalitätsumfang anpassen, verringern.
R5	Schlechtes Zusammenspiel der Komponenten (Technologie Stack)	Der angedachte Technologie Stack kann nicht wie angenommen umgesetzt werden, da inkompatible Komponenten/Packages (SkiaSharp, Bilder in DB) existieren.	20	15%	3	Internetanalyse. Gibt es bereits Projekte, die die angedachte Kombination bereits so einsetzen.	Inkompatible Komponenten ersetzen.

R6	Qualität	Code Guidelines, Qualitätsmanagment werden nicht eingehalten.	20	10%	2	Guidelines einhalten, Kontrolle bei Code-Reviews.	Mehr Reviews und Gespräch mit Entwicklern suchen.
R7	Architektur skaliert nicht	Bei viele Benutzer verhält sich das System sehr langsam und träge.	40	20%	8	Genügt Zeit in die Architekturanalyse investieren und bereits bei den ersten Prototypen mehrere Benutzer und höhere Last simulieren.	Anpassen der Architektur. Alternativ Ausbau der Hardware Infrastruktur.
R8	Continious Integration / Continious	CI/CD ist schwieriger einzurichten als erwartet.	40	10%	4	CI/CD schon von Beginn an einsetzen.	In Foren nach Lösung suchen und Dokumentation von CI/CD Anbieter lesen.
R9	Know-How und Umsetzung mit Solution Strategy	Fehlendes Know-How mit dem Thema "Solution Strategy". Die Umsetzung mit dem "Solution Strategy"-Ansatz gestaltet sich schwieriger als angenommen.	20	25%	5	Know-How im Thema vertiefen und mit Dozent angedachter Ansatz früh genug disskutieren.	Ansatz überdenken und nur auf Ansatz/Methode von "635" bleiben.
R10	Schwierige Umsetzung der Wireframes	Die in der Evaluations Phase erstellten Wireframes lassen sich mit Webtechnologien nur schwer umsetzen.	10	20%	2	Durch die Ausbildung ist den Mitgliedern relativ gut bekannt, wie ein gutes GUI auszusehen hat.	Alternative GUIs besprechen und umsetzen.
R11	Signieren der Apps	Das Signieren für die Apps gestaltet sich schwieriger als gedacht.	25	25%	6,25	Dokumentation und Tutorials lesen.	Mitarbeiter von IFS um Hilfe bitten.
Σ			**270**		**50,5**		

C Zeitauswertung

C.1 Zweck dieses Dokuments

Die Zeitauswertung zeigt auf, wann und wo wieviel Zeit aufgewendet wurde. Dabei haben wir drei unterschiedliche Auswertungen angefertigt, welche in den nachfolgenden Unterkapitel genauer erläutert werden.

C.2 Zeitliche Auswertung nach Monaten

Die erste Auswertung zeit die aufgewendete Zeit nach Monaten. Während den fünf Monaten, in denen das Projekt lief, haben wir insgesamt 724.5 Stunden aufgewendet. Dies entspricht ziemlich genau dem Richtwert von 720 Stunden. Da das Projekt am 18. Februar 2019 startete, fallen die 77 Stunden, welche im Februar geleistet wurden, etwas geringer aus als in den restlichen Monaten. In den Monaten März, April und Mai wurden gesamthaft 185 Stunden, 175 Stunden bzw. 202 Stunden investiert. Im Juni gingen die aufgewendeten Stunden wieder auf ca. 84 Stunden zurück, da das Projektende Mitte Monat (14. Juni 2019) war.

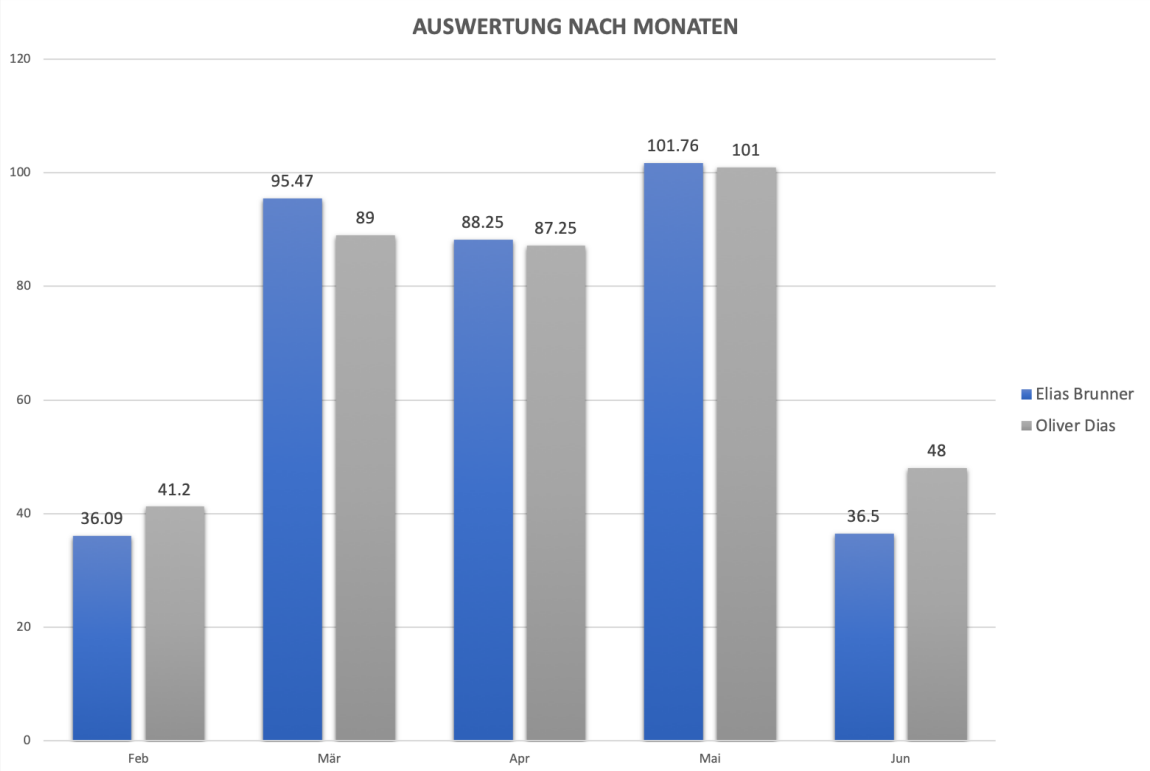

Abbildung 16: Zeitliche Auswertung nach Monaten

C.3 Zeitliche Auswertung nach Sprints

Die Zeitauswertung anhand der durchgeführten Sprints zeigt wie viel Zeit in den einzelnen Sprints investiert wurden.

Dabei ist zu beachten, dass wir einzelne Tickets teilweise über mehrere Sprints bearbeiteten. Meistens geschah dies, weil wir mit dem Ticket noch nicht fertig waren.

Die Auswertung zeigt daher nicht nur die definierten Sprints bzw. die Zeiten, die in jenem Sprint geleistet wurden, sondern auch den oben erwähnten Fall.

Der Ausreisser von 83.25 Stunden ist dadurch zu erklären, dass es sich dabei hauptsächlich um die Arbeiten des Refactorings (siehe Anhang D) handelt und wir dabei generell mehr Zeit benötigten als zunächst gedacht. Im Speziellen für das Frontend, was Oliver Dias-Lalcaca übernommen hat.

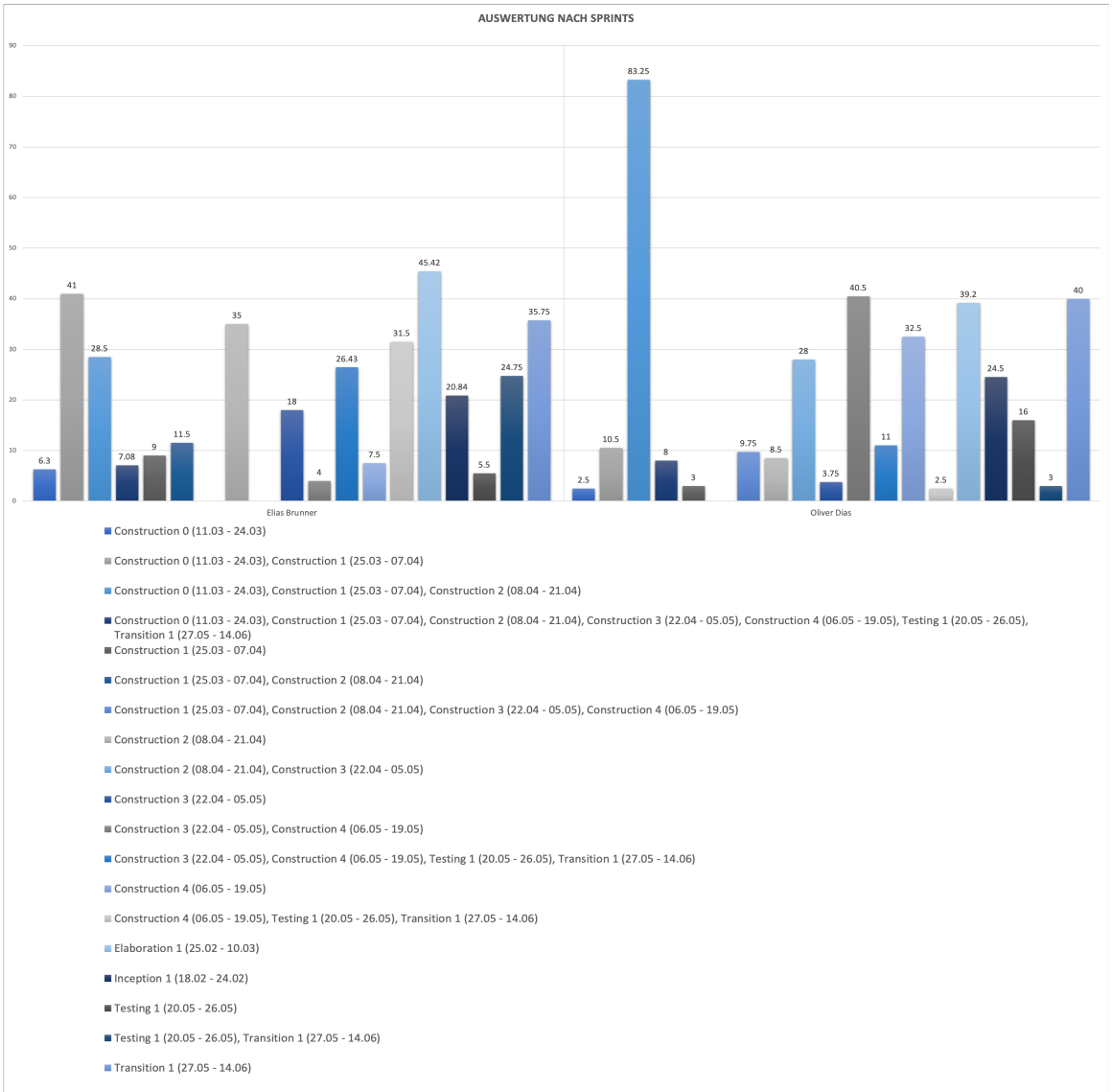

Abbildung 17: Zeitliche Auswertung nach Sprints

C.4 Zeitliche Auswertung nach Arbeitskategorien

Zuletzt interessierte uns noch wie viel Zeit wir für die einzelnen Arbeitskategorien (Labels) aufgewendet haben.

Um das Diagramm zu verstehen muss gesagt werden, dass der innere Kreis die Zeiten für Elias Brunner repräsentiert und der äussere Kreis die Zeiten für Oliver Dias-Lalcaca darstellt.

Die prozentuale Aussage gibt ein ungefähren Überblick über die investierten Zeiten für die einzelnen Arbeitskategorien. Dabei fällt auf, dass wir die meiste Zeit in die Entwicklung 34 % bzw. 64% (durchschnittlich 49%) und die Dokumentation 29% bzw. 14% (durchschnittlich 21%) investiert haben.

An dritter Stelle steht das Projekt-Management mit 15 % bzw. 10%. Laut dem Project Management Institute [20] beträgt der Zeitanteil, welcher durchschnittlich in kleineren Softwareprojekten für das Projekt-Management investiert wird zwischen 9-15%. Unser Aufwand liegt somit etwa in diesem Bereich.

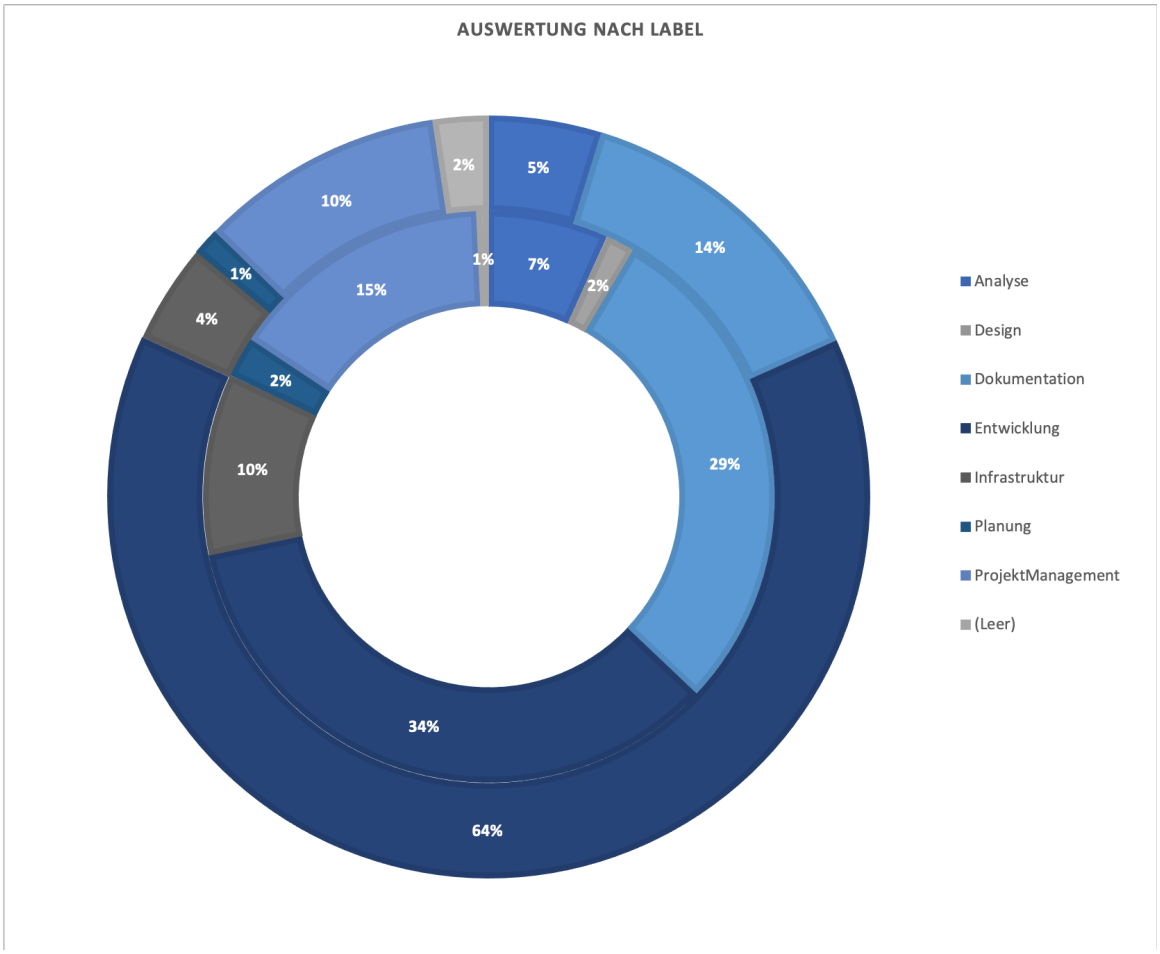

Abbildung 18: Zeitliche Auswertung nach Labels

D Code Review

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

BrainingOutOfBox

Review & Findings

Hochschule für Technik Rapperswil

IFS INSTITUTE FOR SOFTWARE

HSR Hochschule für Technik Rapperswil ■ IFS ■ Oberseestrasse 10 ■ CH-8640 Rapperswil
T +41 55 222 46 30 ■ F +41 55 222 46 29 ■ ifs@hsr.ch ■ www.ifs.hsr.ch

1 Inhalt des Reviews

Dieses Review enthält Findings zum Programmcode:

- Server / Master stand `af00957` https://github.com/BrainingOutOfBox/API.git
- Client / Master stand `9272cfc` https://github.com/BrainingOutOfBox/App.git

2 Server

2.1 MVC nicht eingehalten

Controller enthalten direkte Zugriffe auf die DB und somit auch Business Logik (konkret: kein MVC). Um die Controller nicht an eine DB-Implementation zu binden, sollten Services (und falls viel Business Logik folgt, ebenfalls Data Access Objects) eingeführt werden. Dies ermöglicht Microtesting, wartbareren und testbareren Code sowie Separation of Concerns. Die Basis dazu stellt Google Play mittels des DI Containers zu Verfügung. Ein eigentliches Layering konnte im Server-System nicht gefunden werden.

2.2 Filters zur Code-Reduktion verwenden

Server-Side könnten Filters für die Security Headers (JWT) eingesetzt werden (siehe https://www.playframework.com/documentation/2.7.x/SecurityHeaders). Dies ermöglicht es, den Code in den Controllers zu reduzieren.
Ebenfalls ist es ratsam die JSON-Serialisierung sowie das Fehlerhandling in einen Filter auslagern, da dies sehr repetitiv ist und zentral effizienter gelöst werden kann (z.B. einheitliche Statuscode- und Error-Mappings / Success Messages / …). Andere Frameworks wie Spring Boot bieten solche Features "out-of-the-box".

2.3 DTOs nicht vorhanden

Teilweise arbeiten die Controllers direkt mit den JSON-Nodes anstatt mit Data Transfer Objects (Methode putBrainsheet). Dies resultiert in sehr instabilen Schnittstellen, welche bei kleinen Feld-Namensänderungen eine Shotgun Surgery https://refactoring.guru/smells/shotgun-surgery auslösen.

2.4 Technologie zielgerichtet eingesetzt

Controller-Code teils ungünstig lange; beispielsweise könnte anstatt einer eigenen anonymen Implementation von `SingleResultCallback` mit Lambdas gearbeitet werden (sofern Java 8+ eingesetzt wird).

Anmerkung: Der Code setzt stark auf ASYNC-APIs. Dies macht den Code einiges komplexer. Wäre die synchrone Standard-API auf https://www.playframework.com/modules/mongo-1.1/home nicht bereits ausreichend?

Unit Tests fehlen.

3 Client

3.1 UI / ViewModel

UI-Strings (z.B. Status Messages) wurden vielfach im View Model definiert (`JoinTeamPageViewModel.JoinTeam`). Dafür bietet .NET Resource Manifests (Project Settings → Resources) oder alternativ Static Resources als XAML Files.

Responsibilities wie Navigation sollten in spezialisierte UI-Services ausgelagert werden (befindet sich im Moment im `MainPageViewModel`).

3.2 Model / Services

Business Services sind nicht enthalten. Es wird aus dem UI (View Model) direkt auf den Data Access Layer (http Backend Abstraktion) zugegriffen.
Beispiel: Die Login-Prozedur mit entsprechenden States, welche fürs Login gespeichert werden müssen, befinden sich somit im ViewModel. Hier könnte ein "Login"-Service diese Responsiblities übernehmen und die Funktionalität kapseln. Separation of Concerns wäre somit sichergestellt, "globale" States wie im `BrainstormingContext` gehalten würden so in den Information Experts platziert. Entsprechend ist im Moment das Zusammenspiel der Events/Properties der `BrainstormingContext` Klasse sehr intransparent implementiert.

Die Benennung der DTOs ist zu http/REST-spezifisch; der Code liest sich so, als ob das ViewModel direkt mit dem http-Backend kommunizieren würde (z.B. `LoginPageViewModel.Login()`).

3.3 Data Access Layer

Die Server-URL befindet direkt im Code (d.h. fix definiert). Dies ist unglücklich, falls die Applikation im Testbetrieb auf verschiedene Umgebungen zugreifen muss. Eine Konfiguration pro Deployment-Target würde Abhilfe schaffen.

`RestResolverBase` bietet als Basisklasse ausschliesslich statische Methoden an. Dies wäre ein typischer Anwendungsfall eines Low-Level Resource Services, welcher eine erste Abstraktion des http-Backends anbietet; die Higher-Level Resource Services würden sich diesen via DI injecten lassen. Dieselben Mime-Types wurden in der `RestResolverBase` mehrfach definiert.

Data Access Objects (z.B. `TeamRestResolver`) sollten nie direkt vom Code (View Model) instanziiert werden. Dafür bietet der DI-Container von PRISM eine entsprechende Build-Up Funktionalität, um das automatisierte Testen (Unit Tests) so einfach und flexibel wie möglich zu gestalten.

HSR Hochschule für Technik Rapperswil ■ IFS ■ Oberseestrasse 10 ■ CH-8640 Rapperswil
T +41 55 222 46 30 ■ F +41 55 222 46 29 ■ ifs@hsr.ch ■ www.ifs.hsr.ch

3.4 Technologie zielgerichtet einsetzen

Generelle Code-Hygiene sollte erhöht werden (siehe https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions):

- Statements wie `findingItems.Count > 0;` könnten vereinfacht werden zu `.Any();`
- Wo immer möglich String-Interpolation anstatt "x" + "y" verwenden.
- `this.` .. in C# weglassen, wenn es nicht zwingend gebraucht wird.
- Fields an den Beginn der Klasse platzieren.
- Tools wie Resharper können hilfreich sein, um solche Verbesserungen zu erkennen.

Der Code sollte keine Console.Logs() enthalten. Hier könnten beispielsweise Logging-Frameworks (z.B. NLog) helfen.

Jeder Layer (ViewModel / Model / Data Access) sollte in ein eigenes Projekt (oder Library) platziert werden. Somit werden zyklische Abhängigkeiten von Beginn an vermieden.

Unit Tests fehlen.

HSR Hochschule für Technik Rapperswil ■ IFS ■ Oberseestrasse 10 ■ CH-8640 Rapperswil
T +41 55 222 46 30 ■ F +41 55 222 46 29 ■ ifs@hsr.ch ■ www.ifs.hsr.ch

E Installationsanleitung

E.1 Zweck dieses Dokuments

Die Installationsanleitung bietet interessierten Personen eine Schritt-für-Schritt Anleitung, wie das gesamte System installiert wird. Da sich die Installationsanleitung nicht wesentlich von jener aus unserer Studienarbeit [1] unterscheidet, konnten viele Teile übernommen werden. Dennoch wurde diese Installationsanleitung etwas angepasst und präzisiert.

E.2 Anforderungen

Für die Installation der Cross-Plattform Applikation auf dem eigenen Smartphone wird Folgendes vorausgesetzt:

- Visual Studio IDE Community Edition (7.6.11 (build 9))
- Android + Xamarin.Forms
- iOS + Xamarin.Forms
- .NET Core + ASP.Net Core

Die Visual Studio IDE kann unter www.visualstudio.microsoft.com heruntergeladen werden.

Wem es nur darum geht, die Xamarin Applikation auszuprobieren, kann bei Kapitel E.3 weiterlesen. Wer allerdings zusätzlich noch das PlayFramework laufen lassen möchte, benötigt zusätzlich noch Folgendes:

- Java SE 1.8 oder höher
- Ein Build-Tool wie SBT oder Gradle
- Docker Community Edition

Weitere Informationen zur Installation von Java oder den Build-Tools sind unter www.playframework.com zu finden.

E.3 Installation

Bevor die eigentliche Installation jedoch beginnen kann, muss man noch über den Quellcode verfügen. Dieser kann von unseren Github Repositories heruntergeladen werden.

Installation der Cross-Plattform Applikation

Für die Installation der Xamarin Applikation auf einem Android Smartphone muss folgendermassen vorgegangen werden.

1. Öffne das Projekt mit Visual Studio.
2. Wähle das “Method635.App.Forms.Android “als Startprojekt.
3. Schliesse dein Smartphone mittels USB-Kabel an deine Entwicklungshardware an.
4. Mittels F5 oder dem Play Button kann die Applikation auf dem Smartphone gestartet werden.

Für die Installation der Xamarin Applikation auf einem iOS Smartphone muss etwas anders vorgegangen werden. Dabei sind allerdings ein Entwickler Account von Apple sowie ein Apple Mac notwendig.

1. Öffne zuerst Xcode und erstelle ein leeres Projekt. Trage dabei 'ch.brainingoutofbox-.method635' als Bundle Identifier ein.
2. Schliesse dein Smartphone mittels USB-Kabel an deine Entwicklungshardware an.
3. Signiere das leere Projekt und stelle es auf deinem iOS Gerät bereit. Dieser Vorgang erstellt auch ein sogenanntes 'Provisioning Profile'.
4. Öffne nun das Method635-Projekt mit Visual Studio.
5. Wähle das “Method635.App.Forms.iOS “als Startprojekt.
6. Navigiere zur Datei Info.plist und öffne diese.
7. Wähle nun 'Manuelle Bereitstellung'. Unter den Optionen für die Signierung wählst du nun deinen Apple Entwickler Account aus und das zuvor erstellte Provisioning Profile.
8. Mittels F5 oder dem Play Button kann die Applikation auf dem Smartphone gestartet werden.

Bemerkung: Sollte dabei der Fehler “resource fork, Finder information, or similar detritus not allowed“ auftreten, kann dieser wie auf www.stackoverflow.com beschrieben, vermieden werden. Danach muss das Projekt aber zuerst bereinigt werden.

Installation des PlayFrameworks

Für die Installation des PlayFrameworks muss zuerst eine Datenbank mit den entsprechenden Collections bereitgestellt werden. Dazu nutzen wir eine MongoDB-Instanz, welche in einem Docker-Container läuft.

1. docker run -name methode635 -p 40002:27017 -d mongo:latest
2. docker exec -it methode635 bash
3. mongo –username root –password –authenticationDatabase admin –host localhost –port 40002
4. use Methode635
5. db.createCollection("BrainstormingFinding")
6. db.createCollection("BrainstormingTeam")
7. db.createCollection("Participant")
8. db.createCollection("Pattern")

Weitere Informationen zu MongoDB als Docker-Container können unter hub.docker.com nachgelesen werden. Wer mehr über die createCollection Befehle erfahren möchte, findet unter docs.mongodb.com mehr dazu.

Um das PlayFramework zu builden und laufen zu lassen genügt es, im API Projekt-Ordner folgenden Befehl auszuführen.

1. sbt run "40000"

Weitere Hilfestellungen im Umgang mit dem Simple Build Tool (SBT) und dem PlayFramework finden sich auf playframework.com. Um die Kommunikation zwischen iOS Geräten und dem Backend aufzubauen, ist es zwingend notwendig ein SSL Zertifikat zu verwenden. Nähere Angaben dazu können auf www.playframework.com nachgelesen werden.

Bemerkung: Allenfalls muss für die Interaktion zwischen API und der Datenbank der Username, das Passwort sowie der Datenbankname geändert werden. Dies kann in der 'application.conf' Datei angepasst werden.

```
1 db {
2 # You can declare as many datasources as you want.
3   mongo.url = "localhost"
4   mongo.port = "40002"
5   mongo.database = "someDatabase"
6   mongo.username = "username"
7   mongo.password = "password"
8
9 }
```

Listing 21: Verbindung zur MongoDB

Auch muss im Quellcode der Cross-Plattform Applikation der URL auf die IP deines PC gewechselt werden. Dies kann in der 'backend-config.json' Datei geändert werden.

```
"server": {
    "hostname": "IPOfYourPC",
    "port": 40000
}
```

Listing 22: Verbindung zum API

F Anleitung für Ideen-Erweiterungen

F.1 Zweck dieses Dokuments

Dieses Dokument stellt eine Anleitung für eine allfällige Erweiterung der vorhandenen Ideen-Typen dar. Ziel dieses Dokumentes ist es demzufolge, Hilfestellungen in solch einem Fall zu bieten.

F.2 Szenario

Um diese Anleitung etwas einfacher und konkreter zu gestalten, gehen wir von einem bestimmten Szenario aus. Das Szenario sieht vor, dass man sich dazu entschieden hat die Applikation um den Ideen-Typ 'QualityIdea' zu erweitern. Bei einer 'QualityIdea' geht es dabei um nicht-funktionale Anforderungen, die sich an den ISO 25010 Standard halten (siehe Abbildung 5 in Kapitel 2.4.2). So könnte man sich vorstellen, dass ähnlich wie bei den 'PatternIdeas' eine Auswahlliste von nicht-funktionalen Anforderungen bereitgestellt wird.

Alle nachfolgenden Anpassungen beziehen sich daher auf dieses Szenario. Andere Ideen-Typen sind aber analog zu integrieren.

F.3 Anpassungen am Backend

Erster Schritt: Der erste Schritt im Backend besteht darin die entsprechenden Business-Objekte und Data-Transfer-Objekte mit all deren Attributen und Methoden anzulegen. Wichtig hierbei ist es, dass die neu erstellten Klassen von `Idea` bzw. `IdeaDTO` erben.

Zweiter Schritt: Ist dies geschafft, muss dem JSON-Serialisierer noch gekannt gegeben werden, dass ein weiterer Subtyp hinzugefügt wurde. Hierzu muss in der `IdeaDTO`-Klasse ein weiterer JsonSubTyp hinzugefügt werden (Zeile 5).

```
1 @JsonTypeInfo(...)
2 @JsonSubTypes({
3     @JsonSubTypes.Type(value = NoteIdeaDTO.class,
          name = "noteIdea"),
4     ...,
5     @JsonSubTypes.Type(value = QualityIdeaDTO.class,
          name = "qualityIdea")
6
7 })
```

Listing 23: Weiterer Ideen-Typ hinzufügen in der IdeaDTO-Klasse

Dritter Schritt: In einem letzten Schritt muss nur noch die `ModelsMapper`-Klasse etwas angepasst werden (Zeile 4/6-9 bzw. 14/16-19).

```
1  /* FROM BO TO DTO */
2  modelMapper.createTypeMap(Idea.class, IdeaDTO.class)
3      ...
4      .include(QualityIdea.class, IdeaDTO.class);
5
6  modelMapper.typeMap(QualityIdea.class, IdeaDTO.class)
       .setProvider(request -> {
7      QualityIdea idea = (QualityIdea)request.getSource
           ();
8      return new QualityIdeaDTO(idea.getDescription(),
           ...);
9  });
10
11 /* FROM DTO TO BO */
12 modelMapper.createTypeMap(IdeaDTO.class, Idea.class)
13     ...
14     .include(QualityIdeaDTO.class, Idea.class);
15
16 modelMapper.typeMap(QualityIdeaDTO.class, Idea.class)
       .setProvider(request -> {
17     QualityIdeaDTO idea = (QualityIdeaDTO)request.
           getSource();
18     return new QualityIdea(idea.getDescription(),
           ...);
19 });
```

Listing 24: Weiterer Ideen-Typ hinzufügen in der ModelsMapper-Klasse

Diese drei Schritte reichen aus, damit das Backend auch mit einer QualityIdea interagieren kann.

F.4 Anpassungen am Frontend

Das Hinzufügen von neuen Ideentypen kann im Frontend in drei Schritte ausgeführt werden:

1. Erweiterung des Datenmodells
2. Ansicht für das Einfügen des neuen Ideentyps
3. Anzeigen der eingefügten Idee in den Sheets

Die nachfolgenden Unterkapitel beschreiben die aufgelisteten Schritte.

F.4.1 Erweiterung des Datenmodells

Erster Schritt: Der erste Schritt ist analog zum Backend; die benötigte Idee muss als Business-Objekt sowie Daten-Transfer-Objekt erstellt werden. Auch hier

ist es notwendig, von der abstrakten Klasse `Idea` bzw. `IdeaDto` abzuleiten.

Zweiter Schritt: Ebenfalls muss die neue Idee dem Mapper bekannt gemacht werden. Dazu ist der Methode `MapIdeas()` in der Klasse `BrainstormingMappingProfile` das Mapping von und zum DTO zu erstellen und mit `Include<>()` der `Idea` zu registrieren. Bei Unklarheiten kann die ausführliche Dokumentation von Automapper weiterhelfen.

Dritter Schritt: Damit die JSON-Serialiserung wie gewünscht funktioniert, muss in der Klasse `IdeaConverter` das JSON-Attribut 'Type' gelesen und geschrieben werden. Dazu muss ein neuer `const string` mit Wert 'qualityIdea' initialisiert werden und im Switch-Case der `ReadJson()`-Methode, analog zu den anderen Ideen, das deserialisierte Objekt zurückgegeben werden. Zusätzlich wird in der `WriteJson()`-Methode ein weiteres `else if` benötigt, das den entsprechenden String der Konstante ins 'Type'-Attribut schreibt.

Mit diesen Schritten ist das Datenmodell erfolgreich erweitert.

F.4.2 Ansicht fürs Einfügen von neuen Ideentypen

Hierbei geht es um die Möglichkeit, mit einer neuen Page die `QualityIdea` einfügen zu können.

Erster Schritt: Es muss eine neue Prism View und ein neues ViewModel mit der gewünschten Ansicht fürs Einfügen einer `QualityIdea` erstellt werden. Wichtig ist, dass das ViewModel den `BrainstormingService` kennt. Dieser wird benötigt, um die Idee den Sheets hinzuzufügen (`CommitIdea()`).

Zweiter Schritt: In der `InsertPage` ist ein Frame mit einem `TapGestureRecognizer` zu erstellen, das im dahinterliegenden Command die Navigation auf die erstellte Page ausführt.

Somit ist es dem Benutzer möglich, eine neue Idee auszuwählen und dem Brainsheet hinzuzufügen.

F.4.3 Neue Idee im Sheet anzeigen

Erster Schritt: Jede Idee hat ihr eigenes `DataTemplate`, das das Aussehen innerhalb des Sheets definiert. Diese sind in den Content-Page-Resources der `BrainstormingPage` definiert. Hier muss ein neues Template definiert werden, das die Idee und deren Attribute wie gewünscht darstellt.

Zweiter Schritt: Um das erstellte Template auszuwählen, muss die Klasse `IdeaTemplateSelector` erweitert werden. Eine weitere Überprüfung des Typs auf `QualityIdea` reicht dabei aus, das neue Template zurückzugeben.

Hiermit sind alle benötigten Schritte erledigt und der neue Ideentyp ist integriert.

F.5 Exkurs: Andere Datenbank verwenden

Der Vollständigkeitshalber soll in diesem Kapitel auch erläutert werden, wie die momentan verwendete MongoDB Datenbank ausgewechselt werden kann, sollte dies notwendig sein.

Dies kann in zwei einfachen Schritten erreicht werden.

Erster Schritt: Zunächst muss jedes der fünf Datenbank-Interfaces für die neue Datenbankanbindung in einer entsprechenden Klasse (z.B. MySQLFindingService-Klasse) implementiert werden.

Zweiter Schritt: In einem zweiten Schritt muss im Interface nur noch mitgeteilt werden, von welcher Klasse dieses implementiert wird.

```
1 @ImplementedBy(MySQLFindingService.class)
2 public interface DBFindingInterface {...}
```

Listing 25: Andere Datenbank verwenden

G Testprotokoll

G.1 Zweck dieses Dokuments

Um sicher zu stellen, dass unsere Cross-Plattform Applikation in ihrer Kernfunktionalität richtig funktioniert, wurden mehrere systematische User-Tests anhand des nachfolgenden Testprotokolls durchgeführt.

G.2 Verweise

Die Testfälle leiten sich von den Fully-Dressed Use Cases aus Kapitel 2.4.1.2 ab. Da die Use-Cases 8c, 8e und 13 in dieser Arbeit im Fokus standen, wurde gerade auf diese Use-Cases ein spezielles Augenmerk beim Testen gelegt.

G.3 Testaufbau

Der erste User-Test wurde am 12. Mai 2019 von Elias Brunner und vier weiteren externen Testern durchgeführt. Bei den Testern handelte es sich um Familienmitglieder, welche den Test alle mit ihren eigenen iPhones (unterschiedliche iOS-Versionen) durchgeführt hatten. Der zweite User-Test wurde am 21. Mai 2019 von Oliver Dias, Elias Brunner und Prof. Dr. Zimmermann durchgeführt. Desweiteren wurde am 11. Juni 2019 ein letzter Test durchgeführt. Dabei wurden drei iOS-Geräte (iPad und iPhone) mit der Version iOS 12.3.1 und ein Android Smartphone mit der Version 9 verwendet. Alle Tests wurden komplett manuell durchgeführt.

Um unsere Applikation auf die iOS-Geräte der externen Tester zu verteilen, nutzen wir TestFlight [21]. TestFlight ist ein Online-Service von Apple, welche es Entwicklern auf einfache Weise ermöglicht, ihre Applikationen an ausgewählte interne und externe Testern zu verteilen.

Da TestFlight allerdings durch Apple aufgekauft wurde, wird das Testen für die Android-Plattform seit 2014 nicht mehr unterstützt [22].

G.4 Testfälle

Use Case 7: View Brainstorming Finding			
Test Nr	Beschreibung	Erwartetes Resultat	Beobachtetes Verhalten (Bestanden?)
1	Als Participant möchte ich als Letzter die letzte Runde abschliessen.	Das System speichert meine Notizen auf dem Server und zeigt mir eine Meldung an, dass das Finding einsehbar ist.	Ja, keine Meldung dafür automatischer Wechsel auf Endresultat.

2	Als Participant möchte ich die Resultate der Gruppe ansehen.	Das System zeigt mir eine Übersicht mit allen Notizen der Teilnehmer.	Ja.
3	Als Participant möchte ich nicht als Letzter die letzte Runde abschliessen.	Das System speichert meine Notizen auf dem Server und zeigt mir die verbleibende Zeit an.	Ja, Führung für User könnte besser sein.
4	Nach Ablauf der Zeit meldet mir das System, dass das Finding einsehbar ist.	Das System zeigt mir eine Übersicht mit allen Notizen der Teilnehmer.	Ja.
5	Als Participant möchte ich nicht direkt zu den Resultaten der Gruppe navigieren sondern zurück zum Homescreen gelangen.	Das System zeigt mir den Homescreen an. Von dort aus gelange ich aber auch zu den Notizen der Teilnehmer.	Ja.

Use Case 8: Create Brainwave			
Test Nr	Beschreibung	Erwartetes Resultat	Beobachtetes Verhalten (Bestanden?)
1	Als Participant möchte ich Ideen während der laufenden Rundenzeit zum aktuellen Problem erfassen.	Das System zeigt meine Ideen an.	Ja, bei grössern Brainwaves nicht immer klar wo die Idee eingefügt wurde.
2	Als Participant möchte ich meine Ideen frühzeitig abgeben.	Das System persistiert meine Ideen und zeigt mir eine Bestätigung an.	Nein, Speicherung erfolgt aber keine Rückmeldung an den User.
3	Als Participant möchte ich meine Ideen nicht frühzeitig abgeben.	Das System zeigt mir eine Meldung an, dass die Zeit der aktuellen Runde abgelaufen ist. Es persistiert die bis zu dem Zeitpunkt erfassten Ideen.	Nein, Speicherung erfolgt aber keine Rückmeldung an den User.

Use Case 8c: Draw Sketch			
Test Nr	Beschreibung	Erwartetes Resultat	Beobachtetes Verhalten (Bestanden?)
1	Als Participant möchte ich eine Skizze während der laufenden Rundenzeit zum aktuellen Problem erfassen.	Das System speichert meine Skizze auf dem Server und zeigt mir eine Meldung an, dass die Skizze gespeichert wurde.	Ja.

Use Case 8e: Insert Pattern			
Test Nr	Beschreibung	Erwartetes Resultat	Beobachtetes Verhalten (Bestanden?)
1	Als Participant möchte ich ein Pattern während der laufenden Rundenzeit zum aktuellen Problem auswählen.	Das System speichert das ausgewählte Pattern und zeigt mir eine Meldung an, dass das Pattern in die Brainwave eingefügt wurde.	Ja.

Use Case 13: Export Brainstorming Finding			
Test Nr	Beschreibung	Erwartetes Resultat	Beobachtetes Verhalten (Bestanden?)
1	Als Participant will ich ein abgeschlossenes Brainstorming Finding exportieren können	Das System speichert das erarbeitete Brainstorming Finding in geeigneter Form und zeigt bei erfolgreicher Speicherung eine Meldung an.	Ja, Speicherung erfolgt in die Zwischenablage.

G.5 Testauswertung

Nach einmaliger Testdurchführung am 12. Mai mit einem der Autoren und den vier externen Testern wurde folgende Rückmeldungen für potenzielle Verbesserungen entgegengenommen:

1. Der QR-Code, welcher beim Erstellen eines Teams generiert wird, sollte auch nach dem eigentlichen Erstellen auf einfache Art einsehbar sein. Sollten z.B. nicht alle Participants nach dem ersten Beitreten im Team sein, könnten diese so nachträglich eingeladen werden.

2. Bei Erstellen eines Teams ist es möglich, bei der Teamgrösse einen Buchstaben statt einer Zahl einzugeben, was zu einem Absturz der Applikation führt. Dies sollte unbedingt behoben werden.

3. Als letzte Rückmeldung wurde angemerkt, dass die Übersicht nicht immer gegeben ist. Gerade bei grösseren Teams mit 3 oder mehr Ideen pro Runde, ist nicht klar, wo die eingefügte Idee zu finden ist (speziell gegen Ende des BrainstormingFindings). Der Umstand, dass die Grösse der einzelnen Platzhalter für die Ideen grösszügig gewählt ist, führt dazu, dass man immer weit nach unten scrollen muss, um die eingefügte Idee zu sehen. Dies könnte vielleicht mit einer relativen Grössenangabe verbessert werden.

Nach einmaliger Testdurchführung des zweiten Tests am 21. Mai mit dem Betreuer und den zwei Autoren wurde folgende Rückmeldungen für potenzielle Verbesserungen entgegengenommen:

1. Speziell wenn man das erste Mal ein Brainstorming durchführt, ist nicht sofort klar, wie der genaue Ablauf aussieht, um eine Idee oder ein komplettes Sheet auszufüllen.

2. Auch ist die Übersicht und Orientierung nicht immer gegeben bzw. schwierig (siehe Rückmeldung 3 von erstem Test). Dabei wurde vorgeschlagen, den User mit hilfreichen Nachrichten oder passenderen UI-Elementen besser zu führen. Evtl. kann beim erstmaligen Start der App auch ein kleines Tutorial Abhilfe schaffen.

3. Zudem wurde angemerkt, dass es für den User hilfreich wäre, die Nummer der momentanen Runde sowie die Nummer des momentanen Blattes zu sehen. So ist klar, wie viele Runden noch zu spielen sind und welches Blatt man gerade ausfüllt.

4. Weiter wurden kleinere Verbesserungsvorschläge für bestehende Texte oder Meldungen entgegengenommen. So wurde z.B. auf Fehler in einzelnen Texten hingewiesen oder passendere Beschriftungen für Buttons vorgeschlagen.

5. Wie schon beim ersten Test vom 12. Mai wurde generell angemerkt, dass die Userführung und die Übersicht beim eigentlichen Brainstorming-Prozess das grösste Potenzial für Verbesserungen aufweisen.

In der letzten Testdurchführung am 11. Juni mit denselben Personen wurden folgende Rückmeldungen entgegengenommen:

1. Leider stürzte die Applikation auf dem iPhone 6S+ des Betreuers mehrmals ab. Dies geschah bei der Ansicht der Auswahl von Skizzenideen oder Patternideen. Da uns während der Entwicklung nur unsere Geräte zur Verfügung standen, konnte der genaue Grund nicht eruiert werden, weil das beobachtete Verhalten auf unseren Test-Devices nie vorkam. Aus zeitlichen Gründen konnten keine detailliertere Untersuchungen oder entsprechenden Verbesserungen getätigt werden.

2. Das eingearbeitete Feedback aus dem vorherigen Testbericht vom 21. Mai wurde als positiv empfunden. Die Orientierung sowie die Benutzerführung konnte mithilfe von entsprechenden Texten und UI-Elementen verbessert werden. Dies konnte durch diesen Test bestätigt werden.

Als positive Rückmeldung wurde angemerkt, dass die Applikation in Punkto Farben, Formen und Texten ansprechend aussieht.

Anders als in Studienarbeit [1], in welcher der automatische Rundenwechsel nicht optimal funktionierte, stellte dieser in den durchgeführten Tests keine Probleme mehr dar, was zusätzlich positiv zu werten ist. Auch funktionierte der Wechsel vom laufenden Zustand (Running-State) in den beendet Zustand (Ended-State) einwandfrei.