Hochschule für Technik Rapperswil

Studienarbeit

Abteilung Informatik

Methode 635 als Cross Plattform App mit Xamarin

Authoren:
Elias Brunner & Oliver Dias

Betreuer:
Prof. Dr. Olaf Zimmermann

22. November 2018

1 Aufgabenstellung

1.1 Ausgangslage

Gerade in der IT-Welt ist das Finden von Lösungen für neu auftretende, aber auch für bekannte Probleme ein wichtiger Bestandteil des Jobs. Durch den Einsatz von Innovationsmethoden kann der Ideenfindung auf die Sprünge geholfen werden. In diesem Bereich finden sich die verschiedensten Ansätze und Methoden; in dieser Studienarbeit soll die Methode 635 [1] verwendet werden. Methode 635 ist eine Kreativitäts- und Brainwriting-Technik, welche die Entwicklung von neuen, ungewöhnlichen Ideen für Problemlösungen in der Gruppe fördert. Die Methode wird im PQM-Modul an der HSR vorgestellt.

Es gibt nach heutigem Kenntnisstand noch keine mobile App, die die Methode 635 unterstützt. Die Motivation für die Studienarbeit besteht darin, eine Cross-Plattform App zu konzipieren und zu implementieren. Dabei sollen moderne Technologien zum Einsatz kommen, welche es den Anwendern ermöglichen, schneller und einfacher eine Lösung für ein Problem zu erarbeiten.

1.2 Ziele der Arbeit und Liefergegenstände

In der Studienarbeit soll die Methode 635 als SmartPhone App umgesetzt werden. Android- und iOS- Support soll durch Verwendung von Xamarin erreicht werden. Es wird erwartet, dass bis zum Ende des Projektes eine lauffähige und getestete Cross-Plattform Applikation umgesetzt wird, welche es Benutzern ermöglicht, die Methode 635 effektiv und effizient auf ihre Probleme anzuwenden.

Damit die App einen Mehrwert gegenüber der Papierversion bietet, soll es z.B. möglich sein, die Anzahl der Teilnehmer variabel zu bestimmen oder verschiedene Medien (Text, Video, Bilder, etc.) zu verwenden bzw. einzubinden. Die persistente Speicherung der bearbeiteten Problemstellungen soll aus Sicht des Kunden einfacher möglich sein als dies mit Papier möglich ist. Ein weiterer Vorteil einer mobilen Anwendung ist, dass Anwender die Methode 635 nutzen können auch wenn sie nicht am selben Ort sind oder die Lösungsvorschläge nicht zur selben Zeit bearbeiten.

Die Vision der Arbeit ist also, die Papierversion für diese Methodik zu funktional und qualitativ zu überbieten. Dabei 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.

Weitere kritische Erfolgsfaktoren sind:

- Konfigurierbarkeit (z.B. Anzahl Teilnehmer und Schritte) und Erweiterbarkeit (im Hinblick auf Folgearbeiten, die u.U. auch andere Brainstorming Methoden 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).

2 Abstract

Inhaltsverzeichnis

3 Management Summary

3.1 Ausgangslage

3.2 Vorgehen

3.3 Ergebnisse

4 Technischer Bericht

4.1 Einleitung und Übersicht

4.2 Was ist Xamarin?

Im Mai 2011 gründete Miguel De Icaza, welcher 2001 das Projekt Mono ins Leben gerufen hatte, die Firma Xamarin mit dem Ziel, die Entwicklung für Cross-Platform-Applikationen für Smartphones zu vereinfachen und zu beschleunigen [2].

Beim Projekt Mono handelt es sich um eine quelloffene Implementation von Microsofts .Net Framework. Die Entwicklung plattformunabhängiger Applikationen ist das Ziel des Projektes [3]. Die Entwickler achten bei der Implementation auf die Einhaltung der Standards für die Common Language Infrastrucute (CLI) und Common Language Specification (auch .Net Framework genannt)[4].

Mono wird stetig weiterentwickelt und ist fester Bestandteil der Xamarin Plattform. Softwareentwickler, welche mit der Xamarin Plattform arbeiten, können Apps für iOS, Android und WindowsPhone in C# schreiben. Der geschriebene Quellcode kann durchschnittlich zu 75 Prozent für alle Plattformen benutzt werden. Die in C# geschriebenen Applikationen werden von der Xamarin Plattform in die jeweilige native Sprache übersetzt, was gewährleistet, dass am Ende des Entwicklungsprozesses eine native Applikation für das jeweilige Betriebsystem zur Verfügung steht. Neben dem mobilen Betriebssystemen ist auch die Entwicklung für Mac und Windows möglich.[3]

Im Februar 2016 wurde Xamarin von Microsoft aufgekauft und ist seither eine Tochtergesellschaft von Microsoft mit Sitz in San Francisco [2].

4.3 Was ist die Methode 635?

Die gesamte Beschreibung der Methode 635 wurde von kreativitästechniken.info [1] übernommen.

Die Methode 635 (auch Methode 6-3-5 geschrieben) ist eine Brainwriting-Kreativitätstechnik. Der Name leitet sich aus den drei wesentlichen Eigenschaften der Methode ab: jeweils 6 Teilnehmer erhalten ein Blatt Papier, auf dem sie je 3 Ideen notieren und die Blätter dann insgesamt 5 mal weiterreichen.

Anwendungsgebiete der Methode 635

Die Methode 635 ist eine Variante des Brainwriting. Sie eignet sich besonders für die erste Phase im kreativen Prozess. Dabei werden zunächst Ideen gesammelt ohne dass eine Bewertung stattfindet. Im Idealfall können so in kurzer Zeit 108 Ideen entstehen. Die Aufforderung, bestehende Ideen aufzugreifen und weiterzuentwickeln macht die Methode 635 zu einer konstruktiven Kreativitätstechnik.

Gleichzeitig kann die Kreativität durch die strukturierte Form aber auch gebremst werden. In der Praxis entstehen daher oft etwas weniger Ideen.

Vorgehen bei der Methode 635

Zunächst erklärt der Moderator die Regeln der Methode 635, führt die Teilnehmer in das Ausgangsproblem ein und ist im Folgenden verantwortlich für die Zeitmessung. Sobald die Teilnehmer über die Ausgangsfrage oder -problem aufgeklärt sind, startet der Moderator die erste von sechs Runden. In jeder Runde werden die Teilnehmer aufgerufen, die oberste noch freie Zeile, bestehend aus 3 Kästchen, mit ihren Ideen zu füllen. Dabei sollten die Teilnehmer die Ideen der Vorgänger aufgreifen, erweitern und/oder weiterentwickeln ohne die Ideen zunächst zu bewerten. Nach einer festgelegten Zeit von beispielsweise 5 Minuten beendet der Moderator die Runde. Die Teilnehmer reichen ihr Arbeitsblatt im Uhrzeigersinn an ihren Sitznachbarn weiter und eine neue Runde beginnt. Im Idealfall sind nach 6 Runden genau 6*18 = 108 Ideen entstanden. In Realität ist die Anzahl aufgrund von doppelten oder leeren Einträgen wahrscheinlich etwas geringer. Dennoch sollten nun eine Vielzahl von Ideen vorliegen.

Nun kann eine Diskussion, Analyse und Bewertung der Ideen erfolgen.

4.4 Vorstudie

In der Vorstudie beschäftigten wir uns zuerst mit der Methode 635 an sich, um ein Gefühl dafür zu bekommen, wie es ist, diese selbst an einem konkreten Problem zu nutzen.

Weiter beschäftigten wir uns mit Xamarin. Hierbei war vor allem der Entscheid zwischen Xamarin.Forms und Xamarin native von grosser Bedeutung, da dieser im späteren Projektverlauf kaum mehr rückgängig zu machen ist.

Ein weiterer Teil der Vorstudie bestand darin eine passende Umgebung für das automatische Deployen der Xamarin Applikation zu finden.

Zum Schluss der Vorstudie beschäftigten wir uns mit der Frage, welches Backend wir für unser Projekt verwenden sollten.

Einige wichtige Punkte und Entscheide waren stark von der Vorstudie abhängig.Es war daher entscheidend, bei der Vorstudie sorgfältig zu arbeiten.

4.4.1 Erste Erfahrungen mit der Methode 635

Bevor wir uns mit den technischen Details der Umsetzung zur Cross-Plattform App auseinander gesetzt haben, spielte jeder der beiden Projektmitgliedern die Methode 635, wie in Kapitel 4.3 beschrieben, mit seiner Familie, Bekannten oder Freunden einmal durch.

Dabei wurden folgende persönliche Erfahrungen und Beobachtungen gemacht:

Diskussion gestartet

Schon nach 2-3 Runden konnte beobachtet werden, dass sich spannende Diskussionen zum gestellten Problem entwickelten. Die Teilnehmer mussten sogar angehalten werden die Diskussionen auf dem Papier weiterzuführen, da sonst die Aussagen verloren gehen.

Interessante Methode

Die Teilnehmer empfanden die Methode 635 als spannend und interessant. Durch die einzelnen Teilnehmer waren auch verschiedenste Blickwinkel auf das Problem vertreten, was wiederum zu unterschiedlichsten neuen Ideen führe.

Wertung einführen

Was allerdings als kleiner Nachteil empfunden wurde, war der Umstand, dass die Methode 635 in der original Version keine Möglichkeit für Wertungen bietet.

Als Mensch bildet man sich während des Lesens der einzelnen Ideen automatisch eine Meinung darüber und wird dadurch stark dazu verleitet,

eine wertende Bemerkung statt einer neuen oder angepassten Idee zu schreiben.

Eine Verbesserung könnte darin bestehen, eine Möglichkeit für Wertungen der Ideen einzuführen. Dies könnte so aussehen, dass man neben dem Text auch ein Label (z.B. Pro oder Contra) und die Idee, welche man bewerten will, auswählen kann.

Zielstrebige Lösungsfindung Da die zwei Durchführungen nacheinander stattgefunden haben, konnten im zweiten Versuch die Teilnehmer darauf hingewiesen werden, dass explizit keine Wertung, sondern eine Erweiterung der vorhergehenden Idee erstellt werden soll. Durch diesen Hinweis konnten am Ende des Brainstormings sehr interessante und brauchbare Ergebnisse erzielt werden.

4.4.2 Xamarin.Forms oder Xamarin native?

4.4.3 Visual Studio App Center

4.4.4 Backend-Technologie

Das Play Framework [5] ist ein, unter der Apache 2 Lizenz stehendes, Web Application Framework. Es folgt dem MVC-Pattern und erleichtert das Erstellen von Webanwendungen. Durch den Aufbau ermöglicht es dem Entwickler unter anderem auf einfache Art und Weise eine RESTful API zu schreiben.

Play basiert auf einer zustandslosen und schlanken Architektur. Da es auf Akka aufgebaut ist, nützt es eine vollständig asynchrones I/O. Ausserdem bietet Play minimalen Ressourcenverbrauch.

Sobald am Code eine Änderung vorgenommen wurde, wird der Server automatisch neugestartet, was die Produktivität des Entwicklers weiter erhöht.

Jeglicher Code, welcher nicht kompiliert werden konnte, wird zudem im Browser angezeigt und man weiss als Entwickler sofort auf welcher Zeile man nachbessern muss.

Seit der Version 2.0 des Play Frameworks ist der Framework Core in Scala geschrieben und als Build Tool wird SBT verwendet. Für das Testen stehen verschiedene Test Frameworks sowohl für Java als auch für Scala zur Verfügung. Mittels Scalatest oder JUnit können Unit Tests für beide Sprachen geschrieben werden. Es ist auch möglich Tools wie scoverage für die Code Coverage zu verwenden.

4.5 Anforderungsspezifikation

4.5.1 Funktionale Anforderungen

In den folgenden Kapiteln werden jegliche funktionalen Anforderungen an die Applikation als sogenannte Use-Cases beschrieben. Als Übersicht dient das Use-Case-Diagramm, wie in Abbildung 1 dargestellt.

4.5.1.1 Brief Use-Cases

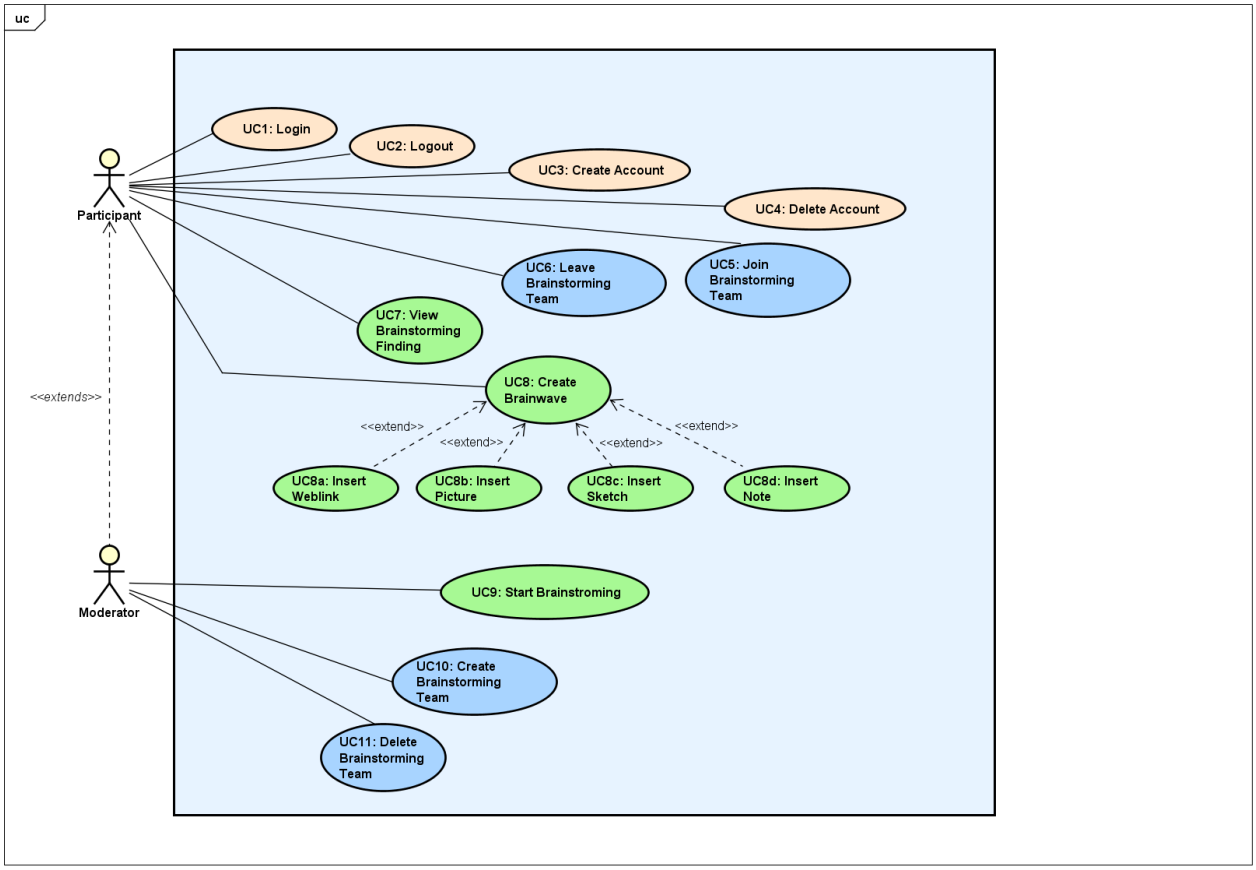

Abbildung 1: Use-Case-Diagramm

Jeder Use-Case hat den nachfolgend kurz (*briefly*) beschriebenen Funktionsumfang. Die Hauptaktivitäten UC7-9 sind am Schluss *fully-dressed* beschrieben.

***UC1:* Login** Als Participant möchte ich mich mit Benutzernamen und Passwort in das System einloggen können.

***UC2:* Logout**	Als eingeloggter Participant will ich mich ausloggen, sodass das Startfenster wieder erscheint.
***UC3:* Create Account**	Als Participant möchte ich mich registrieren können.
***UC4:* Delete Account**	Als Participant will ich meinen erstellten Account wieder löschen können.
***UC5:* Join Brainstorming Team**	Als Participant will ich einem bereits existierenden Team beitreten können.
***UC6:* Leave Brainstorming Team**	Als Participant will ich ein beigetretenes Team verlassen können.
***UC7:* View Brainstorming Finding**	Als Participant will ich, nachdem eine Brainstorming Session durchgeführt wurde, das Resultat (*Finding*) meiner Gruppe einsehen 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:* Insert Picture**	Als Participant will ich ein Bild in mein aktuelles Sheet einfügen können.
***UC8c:* Insert Sketch**	Als Participant will ich in mein aktuelles Sheet zeichnen können.

***UC8d:* Insert Note**	Als Participant will ich normalen Text in mein aktuelles Sheet einfügen können (Defaultoption).
***UC9:* Start Brainstorming**	Als Moderator will ich eine Brainstorming Session starten.
***UC10:* Create Brainstorming Team**	Als Moderator will ich ein Brainstorming Team erstellen können.
***UC11:* Delete Brainstorming Team**	Als Moderator will ich ein Brainstorming Team löschen können.

4.5.1.2 Fully-Dressed Use-Cases

Die fully-dressed Use-Cases folgen den im Modul *Software Engineering 1* empfohlenen Punkten.

Use Case 7: View Brainstorming Finding	
Primary Actor	Participant
Stakeholders & Interests	Ein Participant wünscht sich eine Übersicht von allen Ideen der Gruppenmitglieder. Er will das Gesamtresultat (*Brainstorming Finding*) einsehen und daraus etwas lernen.
Preconditions	Participant existiert im System (UC3), ist eingeloggt (UC1) und einer Gruppe beigetreten (UC5). Zudem hat die Gruppe des Participants alle Runden durchgemacht.
Post Conditions/Success Guarantee	Die Notizen aller Participants werden übersichtlich dargestellt. Das heisst, jede Ausgangsidee ist mit deren Ergänzungen von den verschiedenen Participants ersichtlich.

Main Success Scenario/Basic Flow	1. Der Participant schliesst die abschliessende Runde als Letzter ab. 2. Das System speichert seine Notizen auf dem Server. 3. Das System zeigt dem Participant eine Meldung, dass das Finding einsehbar ist. 4. Der Participant clickt auf "Resultate einsehen"(o.Ä.). 5. Das System zeigt dem Participant eine Übersicht mit allen Notizen der Teilnehmer.

Alternative Flows	Alternative Success Scenario 1: 1.a Der Participant schliesst die Runde nicht als Letzter ab. 1.b Das System speichert seine Notizen auf dem Server und zeigt dem Benutzer die verbleibende Zeit an. 1.c Der Participant aktualisiert den gezeigten Screen nach dem Ablauf der Zeit. Alternative Success Scenario 2: 4.a Der Participant clickt nicht auf "Resultate einsehen", sondern navigiert zurück auf den Homescreen. 4.b Das System zeigt dem Participant den Homescreen an. 4.c Der Participant navigiert auf seine Gruppe. 4.d Das System zeigt dem User die Gruppe an. 4.e Der Participant will sich die Resulate dieser Gruppe anzeigen lassen und clickt entsprechenden Button. 4.f Das System zeigt dem Benutzer die Übersicht an.
Frequency of Occurrence	Oft, Kernfunktionalität

Use Case 8: Create Brainwave	
Primary Actor	Participant
Stakeholders & Interests	Ein Participant will, dass seine Brainwave erfasst wird.
Preconditions	Der Participant muss existent (UC3) sowie eingeloggt (UC1) sein. Des Weiteren muss ein Brainstorming Team existieren (UC10), zu welchem er gehört. Die Brainstorming Runde muss ebenfalls gestartet worden sein (UC9).

Post Conditions/Success Guarantee	Vom Participant erfasste Notizen werden erfolgreich auf dem System gespeichert.
Main Success Scenario/Basic Flow	1. Der Participant erfasst während der aktuellen Rundenzeit Notizen. 2. Das System zeigt diese Notizen an. 3. Der Participant beendet frühzeitig die Runde und clickt auf "Brainwave abgeben"(o.Ä). 4. Das System persistiert die Notizen und zeigt dem Participant eine Bestätigung an. 5. Der Benutzer kann aktualisieren, um den nächsten Rundenstart nicht zu verpassen.
Alternative Flows	Alternative Success Scenario 1: 3.a Der Participant beendet die Runde nicht manuell. 3.b Das System zeigt dem Participant eine Meldung an, dass die Zeit der Runde abgelaufen ist. Es persistiert die bis zu dem Zeitpunkt erfassten Notizen.
Frequency of Occurrence	Sehr oft, passiert pro Brainstorming-Session mehrmals.

Use Case 9: Start Brainstorming	
Primary Actor	Moderator
Stakeholders & Interests	Ein Moderator will eine neue Brainstorming-Session starten können.

Preconditions	Der Moderator muss existent (UC3) sowie eingeloggt (UC1) sein. Des Weiteren muss ein Brainstorming Team existieren (UC10), das er erstellt hat. Zudem muss die Gruppe komplett sein (konfigurierte Anzahl an Participants sind dem Team beigetreten).
Post Conditions/Success Guarantee	Alle Participants des Teams können aktualisieren und sehen die gestartete Runde.
Main Success Scenario/Basic Flow	1. Der Moderator clickt auf "Brainstorming starten"(o.Ä) auf der Gruppe, die er erstellt hat. 2. Das System überprüft, dass alle Einstellungen korrekt sind und die korrekte Anzahl an Participants in der Gruppe sind. 3. Das System zeigt dem Moderator an, dass die Session gestartet ist. 4. Der Moderator kann wie die normalen Participants eine Brainwave (bestehend aus mehreren Ideen) erfassen.
Alternative Flows	-
Frequency of Occurrence	Oft, Kernfunktionalität.

4.5.1.3 Abuse-Cases

Um einem Missbrauch der Applikation entgegenzuwirken, sind neben den Use-Cases auch Abuse-Cases definiert. Diese helfen, mit unangebrachtem Inhalt und unangebrachter Verwendung umzugehen.

***AC1:* Unangebrachte Brainwaves** Ein Participant könnte unangebrachte [1] Inhalte in einer Gruppe hinzufügen. Um dies zu verhindern, könnte eine die Applikation um eine Funktion erweitert werden, die es dem Moderator erlaubt, Benutzer auszuschliessen.

[1] Als unangebrachte Inhalte werden Brainwaves mit rassistischen, pornographischen, sexistischen sowie unethischen Inhalten verstanden.

***AC2:* Missbrauch der Vertraulichkeit** Ein Participant könnte das Brainstorming Finding an die Konkurrenz leaken.

4.5.1.4 Sequenzdiagramm

Der Ablauf der Kernlogik ist der Abbildung 2 zu entnehmen. Darin ist der Prozess vom Erstellen des BrainstormingTeams (UC10) bis zum Abschliessen der Brainwave modelliert.

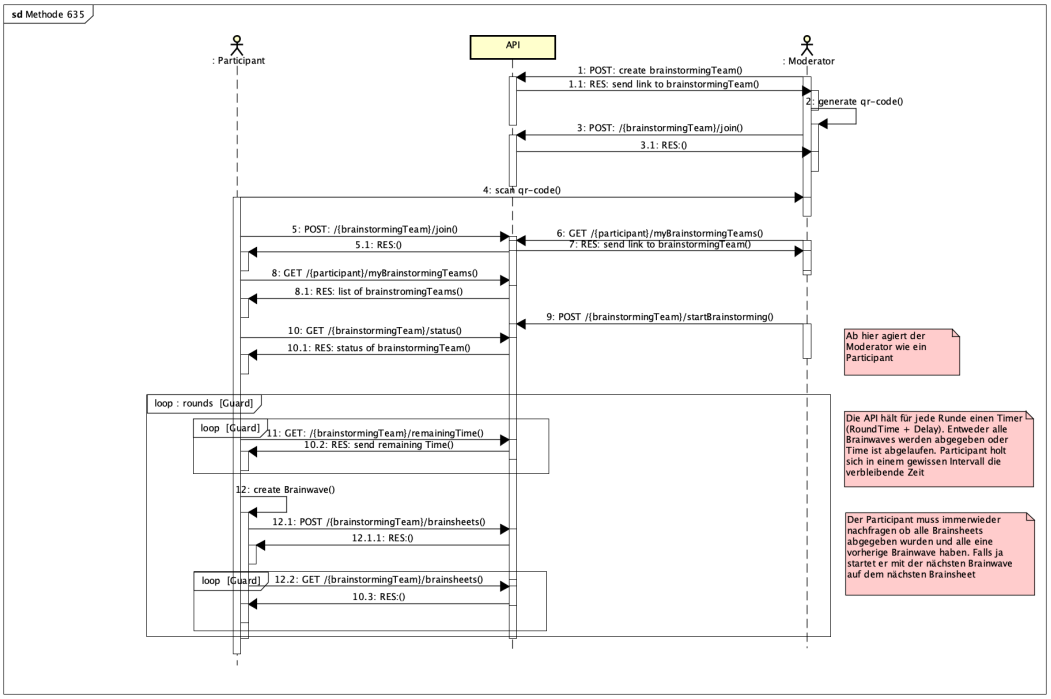

Abbildung 2: Ablauf der Kernlogik

4.5.2 Nicht-Funktionale Anforderungen

Beim Thema Nicht-Funktionale Anforderungen halten wir uns an die Standards ISO 9126[6] bzw. dessen Nachfolger ISO 25010[7]. Beide ISO-Normen sind sich sehr ähnlich und liefern eine gute Checkliste für jegliche Art von Systemanforderungen.

Diese Normen sind sehr umfangreich gestaltet. Wir werden uns daher auf die, für uns, wichtigsten Anforderungen konzentrieren. Um genaue und erfüllbare nicht-funktionale Anforderungen zu definieren, müssen die SMART-Kriterien [9] erfüllt sein.

Ressourcennutzung Die internen Ressourcen Kamera, Dateisystem dürfen nur bei effektivem Bedarf benützt werden. Die CPU-

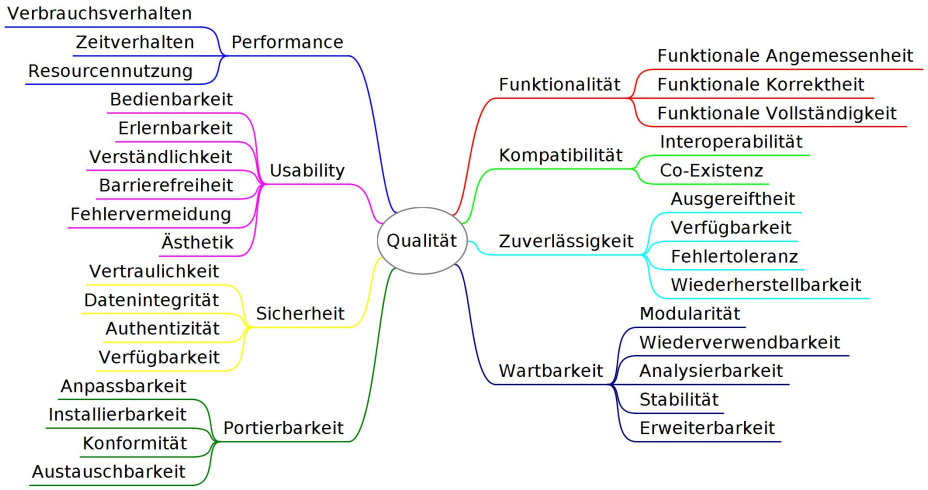

Abbildung 3: Anforderungskategorien nach ISO 25010
[8]

	Ressourcennutzung darf pro Minute im Bereich von bis zu 40% in Anspruch genommen werden.[2][3]
Bedienbarkeit	Wenn eine Aktion länger als 1-2s geht, soll dem User ein Wartesymbol angezeigt werden.
Ästhetik	Die Benutzeroberflächen der Applikation sind so gestaltet, dass die Elemente wiedererkennbar sind (Buttons haben gleichen Stil, leere Textfelder haben Platzhalter).
Vertraulichkeit	Die Daten einer Brainstorming Session können nur von der zugehörigen Gruppe eingesehen werden.
Anpassbarkeit	Die Anpassung bestehender oder Integration neuer Brainstorming-Methoden muss gewährleistet sein.
Installierbarkeit	Die Installation der Applikation auf einem Endgerät erfolgt durch das Ausführen eines *.apk oder *.ipa.[4] Dieser

[2]Referenzsystem Android: Huawei P10 mit Android Version 8.0.0 mit Hisilicon Kirin 960 CPU und 4GB RAM

[3]Referenzsystem iOS: iPhone 6 mit iOS Version 12 mit Dual-core 1.4 GHz Typhoon CPU und 1GB RAM

[4]Eine Applikation auf einem Android Smartphone hat üblicherweise die Endung *.apk. Beim iPhone bzw. beim iOS haben die einzelnen Applikationen die Endung *.ipa.

	Prozess soll unter 1 Minute geschehen.
Co-Existenz	Sollte zu einem späteren Zeitpunkt entschieden werden ein Web-Frontend zu programmieren, muss dieses co-existent mit der Xamarin Applikation existieren können.
Wiederherstellbarkeit	Im Falle eines fehlerhaften Features, muss es innerhalb eines Werktages möglich sein, die Applikation wieder auf den letzten funktionierenden Stand zurück zu holen und erneut zu deployen.
Wiederverwendbarkeit	Die Auswertung von 'Duplicated Code' in SonarQube soll im Bereich zwischen 5-10% [10] liegen. Dies deutet auf eine hohe Wiederverwendbarkeit hin, denn ansonsten müsste der Code kopiert werden.
Analysierbarkeit	Das Ausführen eines Use-Cases muss durch Analyse von Logfiles erkennbar sein.

4.6 Domainanalyse

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

Dabei bilden mehrere Participants ein *Brainstorming Team*. Diese wird von einem der Participants, dem *Moderator*, gegründet. Das Team hat die Möglichkeit, ein oder mehrere *Brainstorming Findings* zu erarbeiten. Dies entspricht einer 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.

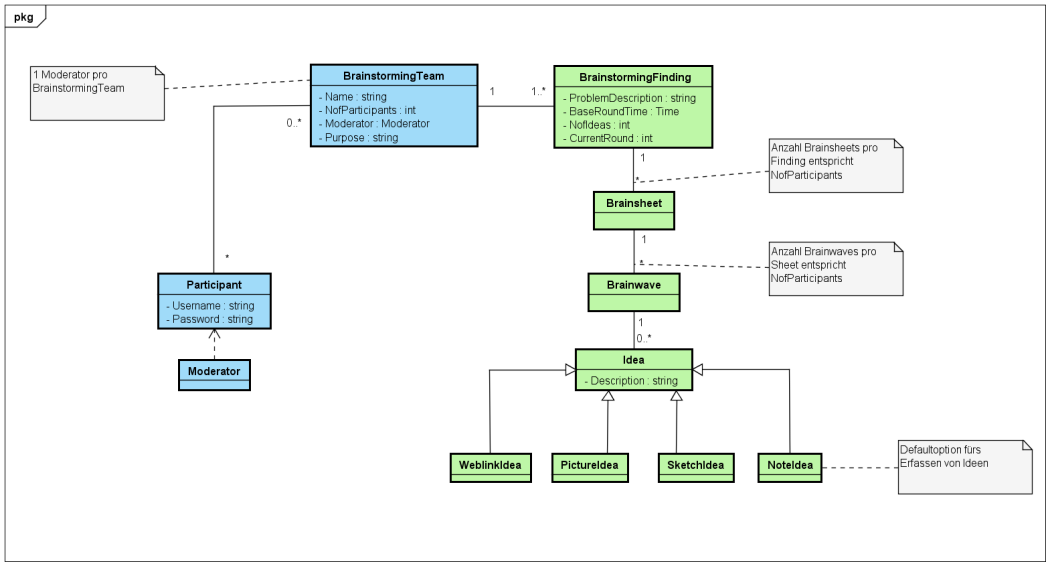

Abbildung 4: Domain Modell BrainingOutOfBox

4.7 Architekturdokumentation

4.7.1 Logische Architektur

Wir teilen die Architektur des gesamten Systems in drei Schichten auf. In der Abbildung 5 sind diese als Presentation-, Businesslogic- und Persistence-Schicht zu erkennen.

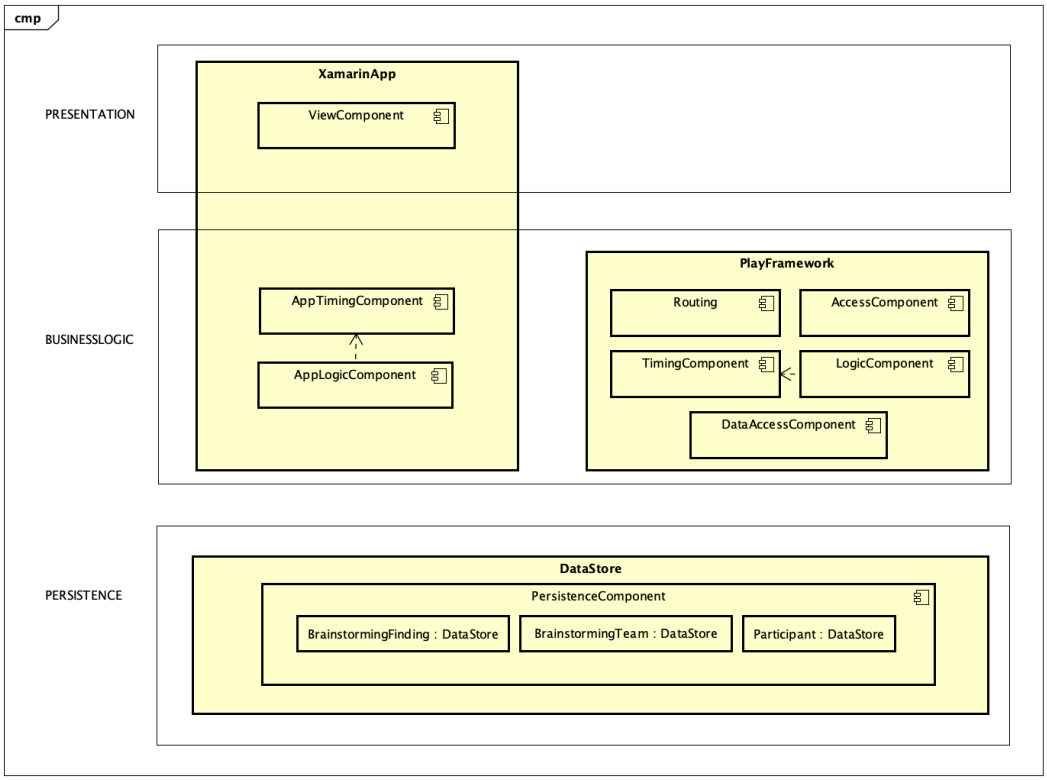

Abbildung 5: Logische Architektur BrainingOutOfBox

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 zum einen 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 den einzelnen Runden. Diese Komponente überwacht daher die noch verbleibende Zeit.

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.

Mit der Schicht der Datenhaltung (Persistence) haben wir eine Schicht zur Verfügung, welche eine Persistence-Komponente hält.

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.

AppLogicComponent Die Logik-Komponente der Xamarin App regelt weitere Logik, wie z.B. den Zugriff auf das PlayFramework.

AccessComponent Die Access-Komponente auf dem PlayFramework regelt den Zugriff mittels JWT-Token[11]. JWT-Tokens werden bei erfolgreichem Login an den Benutzer der App gesendet. So kann sichergestellt werden, dass nur registrierte Benutzer mit dem PlayFramework interagieren können.

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.

4.7.2 Deployment

Wie in der Abbildung 6 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).

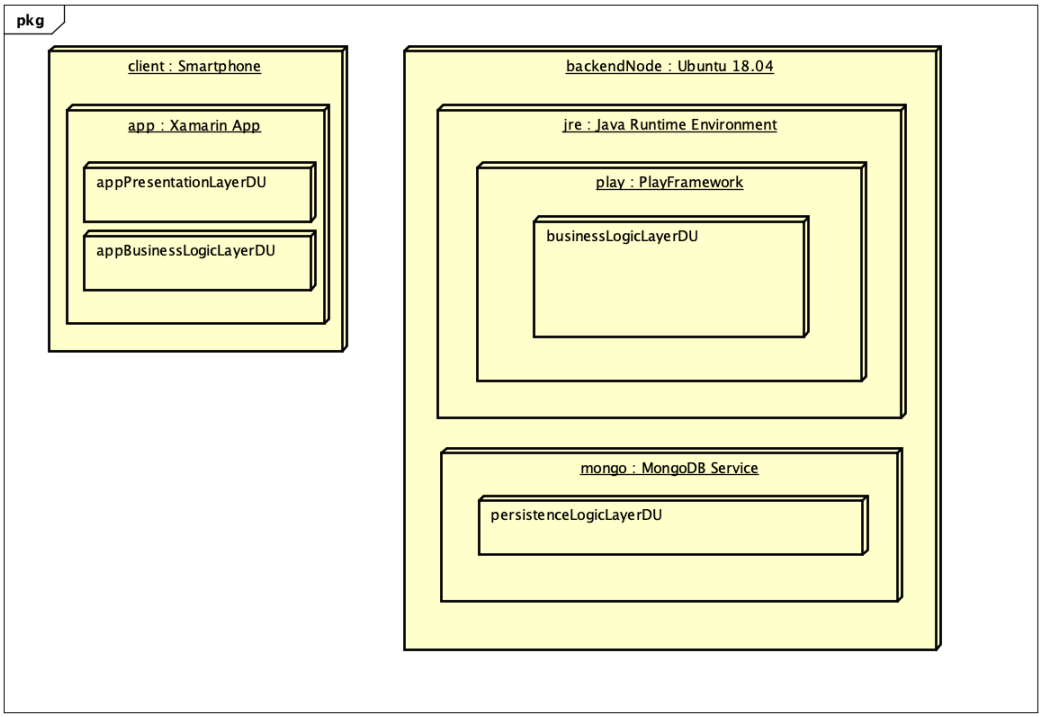

Abbildung 6: Deploymentdiagramm BrainingOutOfBox

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 5 in der Businesslogic-Schicht im PlayFramework eingezeichnet sind.

persistenceLogicLayerDU Die persistenceLogicLayerDU beinhaltet alle Komponenten der Persistence-Schicht, welche in Abbildung 5 zu sehen ist.

appPresentationLayerDU Die appPresentationLayerDU enthält alle Komponenten, welche in Abbildung 5 in der Präsentationsschicht liegen.

appBusinessLogicLayerDU Die appBusinessLogicLayerDU enthält alle Komponenten, welche in Abbildung 5 in der Businesslogic-Schicht in der Xamarin App gezeichnet sind.

4.8 Architekturentscheide

4.8.1 Erste Erfahrungen mit der Methode 635

Wie in Kapitel 4.4.1 beschrieben, tendiert der Mensch dazu, die Ideen der anderen Teilnehmer automatisch zu bewerten. Als mögliche Lösung wurde die Integration einer Bewertungsmöglichkeit beschrieben.

Wir haben uns allerdings darauf geeinigt, dass wir die originale Version, also ohne die Möglichkeit für eine Wertung, als Vorlage nehmen und diese auch so in unserer Cross-Plattform Applikation umsetzen.

Die Integration einer Bewertungsmöglichkeit wird als optionales Feature angesehen und lediglich bei genügend Restzeit im Projekt umgesetzt.

4.8.2 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 [12] 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 [13]) 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 ([14]).

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.

4.8.3 Backend-Technologie

Neben den Vorteilen, wie der schlanken und zustandslosen Architektur des Play Frameworks, 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 Play Framework gemacht haben.

Ein weiterer Grund bestand darin, dass das Play Framework 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, viel die Wahl für das Play Framework erst recht, als wir einen asynchronen MongoDB-Treiber für Java gefunden hatten.

4.8.4 MongoDB als Datenbanksystem

4.8.5 Methode 635 als Peer-to-Peer-System

Wir haben uns auch überlegt, die Cross-Plattform Applikation d.h. vor allem die Kommunikation zwischen den einzelnen Teilnehmer, als Peer-to-Peer System [15] zu konzipieren.

Prof. Thomas Bocek, Professor für verteilte Systeme an der Hochschule Rapperswil, hat uns allerdings davon abgeraten. Ein verteiltes System sei immer komplexer und komplizierter als ein Server/Client System. Für diese geringe Anzahl von Teilnehmern, welche prinzipiell nur Messages austauschen, lohnt es sich nicht ein verteiltes System zu bauen.

Daher haben wir uns für eine klassische Server/Client Architektur entschieden. Andere Varianten der Kommunikation, wie Webhooks oder Websockets werden daher nicht weiter verfolgt.

4.8.6 Kommunikation zwischen Server und App

Die Kommunikation zwischen dem zentralen Server und den Clients, also den Cross-Plattform Applikationen in unserem Fall, ist von grosser Bedeutung. Wegen der Problematik der Network Address Translation kurz NAT [16] kann diese prinzipiell auf zwei Arten erfolgen: Entweder man verwendet Websockets [17], welche eine permanente Verbindung zwischen Server und Client öffnen oder die Kommunikation beginnt ausschliesslich beim Client.

Wir haben uns für das stetige Abfragen von Informationen (Polling) entschieden, da es eine einfache Variante darstellt. Zwar werden dadurch vermeidbare Requests an den Server gesendet, da aber die Anzahl an Teilnehmer bzw. die Ressourcennutzung des Backends in einem vertretbaren Rahmen liegt, ist diese Polling-Variante völlig in Ordnung.

Zwei Beispiele für Polling in der Applikation: Wenn ein Teilnehmer seine Ideen aufschreibt, fragt die Applikation den Server im Hintergrund in regelmässigen Abständen nach der verbleibenden Zeit für diese Runde ab.

Auch nach der Abgabe der aufgeschriebenen Ideen an den Server, muss der Teilnehmer warten, bis er die Ideen bzw. das Blatt seines Nachbarn bekommt. Dafür muss die Applikation immer wieder den Server fragen, ob der Nachbar überhaupt sein Blatt bzw. seine Ideen abgegeben hat. Bevor dies nicht geschehen ist, kann der Teilnehmer auch nicht weitermachen.

Begründung in Zahlen: Bei diesen beiden Szenarien sendet die Applikation pro Sekunde einen Request an den Server. Wenn wir bei der Standardmethode von 6 Teilnehmern bleiben, wären das 6 Request pro Sekunde. Wenn wir nun noch annehmen, dass 50 gleichzeitige Ausführungen stattfinden, ergibt das 300 Requests pro Sekunde für eines der beiden Szenarien.

Laut den Release-Informationen zur Version 2.5 von Play [18] ist das PlayFramework in der Lage 60'000 Requests pro Sekunde zu verarbeiten.

Nehmen wir nun diese 60'000 Requests als Basis für unsere Berechnung, entsprechen unsere 300 Requests gerade einmal 0.5% der möglichen 60'000 Requests. Für beide Szenarien entsprechend das Doppelte, also 1%.

$$\frac{300 Requests}{1 Second} = \frac{1 Request}{1 Second} * 6 Participants * 50 Brainstormings$$

$$0.5\% = \frac{\frac{300 Requests}{1 Second} * 100}{\frac{60000 Requests}{1 Second}} = \frac{300 * 100}{60000}$$

Fazit: Diese kleine Rechnung verdeutlich schon ziemlich stark, dass die Variante mit dem Polling das PlayFramework in keinster Weise an dessen Limit bringt. Selbst wenn die Anzahl an Teilnehmern oder gleichzeitigen Ausführungen erhöht wird, ist das PlayFramework im Stande die eingehenden Requests noch zu verarbeiten. Auch andere gleichzeitige Aufrufe an das PlayFramework können dann noch ausgeführt werden.

Sollte es dennoch jemals zu einer zu starken Ressourcennutzung durch das Polling kommen, könnte der Algorithmus so angepasst werden, dass dieser nicht jede Sekunde das Backend z.B. nach der verbleibenden Zeit abfragt sondern bei viel verbleibender Zeit nur alle 30 Sekunden und bei wenig verbleibender Zeit jede Sekunde.

Auf diese Weise kann die Anzahl an Requests drastisch reduziert werden.

4.9 Herausforderungen

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

4.9.1 HTTPS REST Schnittstelle in CI/CD aufsetzen

Während dem Entwickeln des Prototyps in der Evaluation stellten wir fest, dass das Abfragen unserer Backend-Schnittstelle auf Android wie gewünscht funktionierte, jedoch warf die Applikation beim Ausführen auf iOS eine Exception. Der Grund dafür war, dass iOS keine Verbindungen zu unverschlüsselten Webseiten mehr zulässt. Daher war der Fall klar, dass dies noch aufgesetzt werden muss.

Nach kurzen Recherchen haben wir für das Erstellen und Validieren des Zertifikates auf Let's Encrypt [19] gewählt, gerade deshalb weil eine ausführliche Dokumentation und eine rasche Generierung möglich ist.

Auch mit dem verwendeten Play Framework sollte das Aufsetzen der HTTPS Site kein Problem darstellen, es sollte mit einem Parameter beim Start der Applikation gut möglich sein.

Nachdem das Zertifikat installiert wurde und die Applikation lokal erfolgreich lief, galt es, das Gesamte noch in den Build-Prozess einzubauen. Dabei kam das Problem auf, dass der Pfad zum Keystore, der das Let's Encrypt-Zertifikat beinhaltet, nicht gültig war. Nach mehrmaligem, gründlichem Überprüfen des Pfades und neu Builden, wurden immer noch Exceptions geworfen.

Wir haben keinen Anhaltspunkt, was der Fehler sein könnte, denn werden die genau gleichen Befehle auf dem Backend-Server direkt ausgeführt, funktioniert die Applikation einwandfrei auf HTTPS. Sobald es aber über den Build-Server läuft, findet er den Pfad zum Keystore nicht mehr.

Als Work-Around haben wir eine `CustomSslEngineProvider` geschrieben, der den Pfad zum Keystore direkt im Code beinhaltet. Darauf hat Puppet keinen Einfluss und die Applikation funktioniert wie gewünscht.

4.10 Ergebnisse

Das Kapitel der Ergebnisse befasst sich mit der konkreten Umsetzung der Xamarin Applikation und dem PlayFramework. Wir haben uns dabei ganz konkret für die nachfolgenden Code-Beispiele entschieden. Der Grund dafür ist, dass diese entweder den typischen Aufbau einer Methode zeigen oder wie das Listing 4, eine Kernlogik darstellen. Bei den Ergebnissen der Xamarin App zeigen wir zusätzlich noch eine kurze Retrospektive auf geplantes Design und effektiver Umsetzung des GUIs.

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

4.10.1 Implementierung des PlayFrameworks

Die Umsetzung des Backends basiert auf dem PlayFramework. Die Gründe für diesen Entscheid können in Kapitel 4.4 nachgelesen werden.

Um die Daten permanent zu speichern, besitzt das PlayFramework eine Anbindung an eine MongoDB Instanz. Die Anbindung wurde mit dem MongoDB Async Driver für Java [20] realisiert. Die Implementation ist daher auch in Java geschrieben. Für die Dokumentation des Backends verwendeten wir Swagger [21].

ParticipantController.java

Die nachfolgenden Listings zeigen einen Ausschnitt aus der `ParticipantController.java` Klasse. Diese befindet sich in der LogicComponent (Abbildung 5) vom PlayFramework.

```
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

Bei der Methode für das Erstellen von Participants, prüft das Framework zuerst, ob ein HTML-Body existiert. Ist dies nicht der Fall, sendet es ein HTTP-Response Status-Code (Zeile 24) zurück.

Existiert ein HTML-Body, wird dieser auf die Existenz der Felder *username*, *password*, *firstname* und *lastname* geprüft (Zeile 7-10). Daraus erstellt das Framework als nächstes ein `participant` (Zeile 12), welcher mittels `insertOne` in die `participantCollection` gespeichert wird (Zeile 14).

Zuletzt sendet das PlayFramework eine Antwort mit dem HTTP-Status Code 200 und einer Nachricht an den Absender.

```
1   public Result login() throws
       UnsupportedEncodingException, ExecutionException,
       InterruptedException {
2   ...
3   if (body.hasNonNull("username") && body.hasNonNull("
       password")) {
4       CompletableFuture<Participant> future = new
           CompletableFuture<>();
5
6       participantCollection.find(and(
7       eq("username", body.get("username").asText()),
8       eq("password", body.get("password").asText()))).
           first(new SingleResultCallback<Participant>() {
9           @Override
10          public void onResult(Participant participant,
               Throwable t) {
```

```
11              if (participant != null) {
12                  Logger.info("Found participant");
13                  future.complete(participant);
14              } else {
15                  future.complete(null);
16              }
17          }
18      });
19
20      if (future.get()!= null){
21          ObjectNode result = Json.newObject();
22          result.putPOJO("participant", future.get());
23          result.put("access_token", getSignedToken(7l));
24          return ok(result);
25      } else {...}
26  } else {...}
27  }
```

Listing 2: Login

Für das Login eines Participants schaut auch hier zuerst das Framework im HTML-Body nach der Existenz der Felder *username* und *password* (Zeile 3). Im nächsten Schritt durchsucht es die Datenbank nach einem Participant mit den angegebenen Werten (Zeile 6-8).

Da wir einen asynchronen Treiber verwenden, benötigen wir ein `CompletableFuture`, um das Resultat der Abfrage darin abzuspeichern und um mittels `future.get()` darauf zugreifen zu können (Zeile 20).

Am Ende wird dem Resultat neben dem gefundenen *participant* noch ein *JWT-Token* angefügt (Zeile 21-24).

TeamController.java

Das Listing 3 zeigt einen Ausschnitt aus der `TeamController.java` Klasse. Auch diese befindet sich in der LogicComponent (Abbildung 5) vom PlayFramework.

```
1  public Result joinBrainstormingTeam(String
       teamIdentifier) throws ExecutionException,
       InterruptedException {
2  ...
3  if (brainstormingTeam!= null && brainstormingTeam.
       getNrOfParticipants() > brainstormingTeam.
       getCurrentNrOfParticipants() && brainstormingTeam.
       joinTeam(participant)) {
4
```

```
5       teamCollection.updateOne(eq("identifier",
            teamIdentifier),combine(set("participants",
            brainstormingTeam.getParticipants()), inc("
            currentNrOfParticipants", 1)), new
            SingleResultCallback<UpdateResult>() {
6             @Override
7             public void onResult(final UpdateResult result,
                  final Throwable t) {
8                 Logger.info(result.getModifiedCount() + "
                      Team successfully updated");
9             }
10      });
11
12      return ok(Json.toJson(new SuccessMessage("Success",
            "Participant successfully added to the
            brainstormingTeam")));
13
14  } else {
15      ...
16  }
17  ...
18  }
```

Listing 3: Einem Team beitreten

Dieses Beispiel soll zeigen, wie mit dem MongoDB Async Driver ein Eintrag mittels `updateOne` aktualisiert werden kann.

Wie auch schon bei der `find` Methode, kennzeichnet der erste Parameter das Dokument, welches man aktualisieren möchte. Der zweite Parameter steht für die Felder und deren neuen Werte und der dritte und letzte Parameter ist wieder der `SingleResultCallback` und beschreibt wie das Resultat weiter prozessiert wird. All dies ist auf Zeile 5-8 zu finden.

FindingController.java

Das Listing 4 zeigt einen Ausschnitt aus der `FindingController.java` Klasse. Wie auch schon die vorherigen Klassen, befindet sich auch diese in der LogicComponent (Abbildung 5) vom PlayFramework.

```
1   private void startWatcherForBrainstormingFinding(String
        identifier){
2
3   ScheduledExecutorService executor = Executors.
        newSingleThreadScheduledExecutor();
```

```
4
5  TimerTask task = new TimerTask() {
6      @Override
7      public void run() {
8          try {
9          ...
10         if (finding.getCurrentRound() > finding.
               getBrainsheets().size()){
11             lastRound(identifier);
12             executor.shutdown();
13         }
14
15         if (endDateTime.plusSeconds(30).isBeforeNow() ||
16         finding.getDeliveredBrainsheetsInCurrentRound()
              >= finding.getBrainsheets().size()){
17         nextRound(identifier);
18         }
19             cancel();
20
21         } catch (ExecutionException e) {
22             e.printStackTrace();
23         } catch (InterruptedException e) {
24             e.printStackTrace();
25         }
26     }
27 };
28
29 executor.scheduleAtFixedRate(task, 1000L, 5000L,
       TimeUnit.MILLISECONDS);
30 }
```

Listing 4: Watcher für BrainstormingFinding

Um den Zustand über die verbleibende Zeit oder die bereits eingereichten *Brainsheets* überwachen zu können, setzen wir einen `ScheduledExecutorService` [22] ein. Dieser erlaubt uns alle 5000ms bzw. alle 5s den `TimerTask` auf Zeile 5 auszuführen.

Der `TimerTask` prüft zuerst, ob die aktuelle Runde schon die letzte Runde ist (Zeile 10). Ist dies der Fall, führt er die Methode *lastRound(identifier)* aus und beendet den executor, sodass keine neuen `TimerTask` Objekte gestartet werden. In diesem Zustand ist das gesamte *BrainstormingFinding* ausgefüllt.

Ist dies nicht der Fall, prüft er als nächstes, ob die Endzeit der aktuellen Runde plus 30s noch vor der aktuellen Uhrzeit liegt (Zeile 15). Die Bedingung auf Zeile

16 prüft, ob alle *Brainsheets*, welche für die Abgabe erwartet werden schon abgegeben wurden. Unabhängig welche dieser zwei Bedingungen (Zeile 15 oder 16) zuerst eintrifft, es wird anschliessend immer die Methode *nextRound(identifier)* ausgeführt.

Sollte keine der Bedingungen (Zeile 10, 15 oder 16) zutreffen, so beendet sich der `TimerTask` mittels *cancel* selbst.

Da der executor aber nach 5s den nächsten `TimerTask` startet, ist so für die gesamte Dauer, für die das *BrainstormingFinding* läuft, ein 'Watcher' für den korrekten Ablauf zuständig. Die Methode *startWatcherForBrainstormingFinding(String identifier)* wird beim Start eines *BrainstormingFinding* ausgeführt.

```
1 public Result startBrainstorming(String findingIdentifer
    ) throws ExecutionException, InterruptedException {
2         startWatcherForBrainstormingFinding(
              findingIdentifer);
3         return nextRound(findingIdentifer);
4 }
```

4.10.2 Implementierung der Xamarin App

4.11 Schlussfolgerungen

4.11.1 Ergebnisbewertung

4.11.2 Ausblick

Literatur

[1] Kreativitaetstechniken, "635-methode." https://kreativitaetstechniken.info/6-3-5-methode/, Oktober 2011. Accessed on 2018-09-18.

[2] Wikipedia, "Xamarin." https://de.wikipedia.org/wiki/Xamarin, November 2018. Accessed on 2018-11-22.

[3] CCVossel, "Xamarin - was ist das eigentlich?." https://ccvossel.de/2016/07/xamarin/, November 2018. Accessed on 2018-11-22.

[4] S. Clark, "What is xamarin? what is its nedd?." https://www.quora.com/What-is-Xamarin-What-is-its-need, November 2018. Accessed on 2018-11-22.

[5] Lightbend, "Playframework." https://www.playframework.com/, August 2017. Accessed on 2018-10-09.

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

[7] 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.

[8] 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.

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

[10] SolidSourceIt, "Does source code duplication matter?." https://solidsourceit.wordpress.com/2012/08/03/does-source-code-duplication-matter/, August 2012. Accessed on 2018-10-06.

[11] Auth0, "Jwt." https://jwt.io, November 2018. Accessed on 2018-11-01.

[12] mono, "mono/skiasharp." https://github.com/mono/SkiaSharp, Oktober 2018. Accessed on 2018-10-25.

[13] jamesmontemagno, "Xam-media-plugin." https://www.nuget.org/packages/Xam.Plugin.Media/, Oktober 2018. Accessed on 2018-10-25.

[14] M. Jahn, "Zxing.net." https://www.nuget.org/packages/ZXing.Net/, October 2018. Accessed on 2018-10-25.

[15] ITWissen, "Peer-to-peer-netz." https://www.itwissen.info/Peer-to-Peer-Netz-peer-to-peer-network-P2P.html, Januar 2014. Accessed on 2018-10-16.

[16] A. Donner, "Was ist nat (network address translation)?." https://www.ip-insider.de/was-ist-nat-network-address-translation-a-663954/, August 2017. Accessed on 2018-10-06.

[17] Wikipedia, "Websocket." https://de.wikipedia.org/wiki/WebSocket, Juli 2018. Accessed on 2018-10-16.

[18] Play, "What's new in play 2.5." https://www.playframework.com/documentation/2.6.x/Highlights25, November 2018. Accessed on 2018-11-15.

[19] ISRG, "Let's encrypt." https://letsencrypt.org/, Oktober 2018. Accessed on 2018-10-30.

[20] MongoDB, "Mongodb async java driver." http://mongodb.github.io/mongo-java-driver/3.8/driver-async/, November 2018. Accessed on 2018-11-13.

[21] Swagger, "Swagger-play2." https://github.com/swagger-api/swagger-play/tree/master/play-2.6/swagger-play2, November 2018. Accessed on 2018-11-13.

[22] E. Paraschiv, "Java timer." https://www.baeldung.com/java-timer-and-timertask, November 2018. Accessed on 2018-11-13.

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

[24] M. Kuhrmann, "Rational unified process." http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Vorgehensmodell/Rational-Unified-Process-RUP/index.html, Juli 2018. Accessed on 2018-10-16.

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

A Projektplan

Zweck dieses Dokuments

Dieses Dokument beschreibt den Projektplan des Projekts ≪Methode 635 als Cross Plattform App mit Xamarin≫. 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 17. September 2018 bis zum 21. Dezember 2018. Das Projekt findet im Rahmen des Moduls ≪Studienarbeit≫ im Herbstsemester 2018 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.

A.1 Projektziel

Die Motivation dieser Studienarbeit besteht darin, eine Cross-Platform App zu programmieren, welche die Methode 635 [1] als mobile App für Android und iOS umsetzt. Dabei sollen moderne Technologien zum Einsatz kommen, welche es den Anwendern ermöglichen schneller und einfacher eine Lösung für ein Problem zu erarbeiten.

Mehr zum Thema Projektziel ist dem Kapitel 1 zu entnehmen.

Einschränkungen

Das Projekt ist auf die Dauer des Herbstsemester 2018 begrenzt (bis 21. Dezember 2018). Zudem sollte das Projekt mit ungefähr 240 Arbeitsstunden (gesamthaft 480 Stunden) realisiert werden können. Bleibt am Ende Zeit übrig, werden optionale Features implementiert.

A.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 7 zu entnehmen.

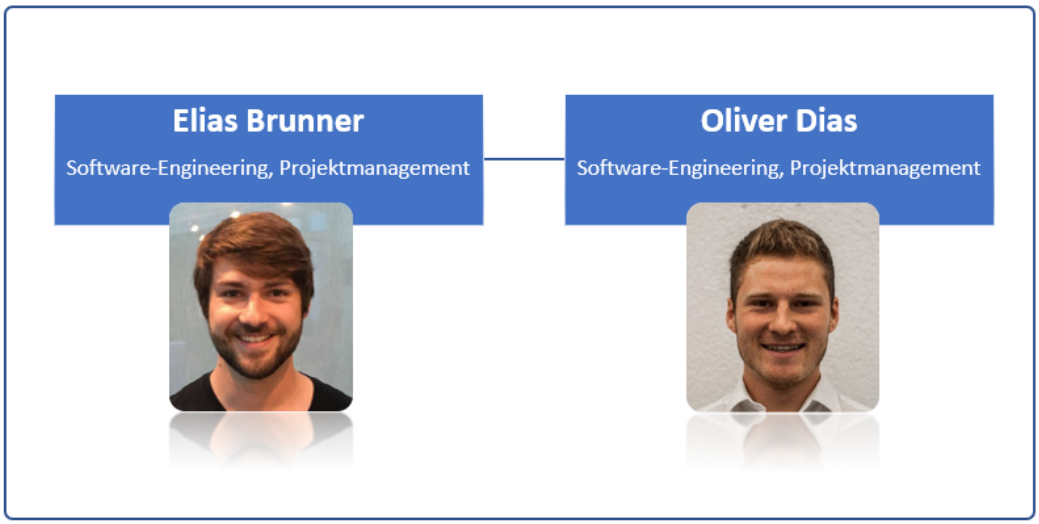

Abbildung 7: Methode 635 Organisation

Ansprechspartner

Im Projekt «Methode 635 als Cross Plattform App mit Xamarin» ist folgender Ansprechpartner vorhanden.

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

A.3 Projektmanagement

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

Da wir mit einer agilen Entwicklung arbeiten, wird die Planung während des Projekts laufend aktualisiert und den aktuellen Gegebenheiten angepasst.

Ein grober Projektablauf ist in Abbildung 8 ersichtlich.

Zur Verwaltung des Codes nutzen wir öffentliche Github Repositories. Die Kommunikation abseits der HSR Anwesenheit erfolgt über einen Whats-App Chat oder alternativ über einen Slack Channel.

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.
Vorstudie	In der Vorstudie wird analysiert, welche Backend-Technologien zur Verfügung stehen oder was der Unterschied zwischen Xamarin.Forms und Xamarin native ist.
Architekturdokumentation	Hier wird dokumentiert wie die Architektur des gesamten Systems aufgebaut ist.
Architekturentscheide	Hier wird erklärt 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 das ganze 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 Donnerstagmorgen 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 A.6) berücksichtigt, die aktuell gehalten wird und beim Planen in Betracht gezogen wird.

Qualitätsmassnahmen

Das Endprodukt dieses Projekts soll von möglichst hoher Qualität sein. Wie in Tabelle 4 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 4: Massnahmen

Arbeitspakete

Die gesamte Arbeit ist in Arbeitspakete unterteilt, die auf Jira getrackt sind. 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 5: Story Points Komplexität

Das Unterteilen in drei Punkte pro Zeile ermöglicht ein genaueres Abschätzen innerhalb der Komplexitätskategorie. Story Points können nicht direkt in zeitlichen Aufwand umgerechnet werden. Für den Aufwand 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 Planungsmeeting an, welches dieser Kategorie zugeordnet ist.

Dokumentation Arbeiten an der Dokumentation des Projektes.

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.

A.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.

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. [23]

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.

A.5 Zeitliche Planung

Phasen

In unserem Projekt verwenden wir die vier Phasen von Rational Unified Process [24]. Innerhalb der einzelnen Phasen wenden wir allerdings das Konzept von SCRUM [25] 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 die Sprints Elaboration-1 und Elaboration-2. Hier werden die Anforderungen an die Applikation definiert, die Vorstudie durchgeführt und weitere konzeptionelle Arbeiten durchgeführt. 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 3.5 Sprints eingerechnet. Dafür stehen die Sprints Construction-1, Construction-2, Construction-3 und Construction-4 zur Verfügung.

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 32 Stunden für die Studienarbeit zur Verfügung. In unserem Projekt sind folgende Sprints geplant:

Inception-1

- Projektplan erstellen
- Jira einrichten
- einzelne Risikos einschätzen

Elaboration-1

- Projektplan überarbeiten
- NFAs sowie FAs definieren
- UCs definieren
- Methode 635 durchspielen
- GUI Entwürfe designen

Elaboration-2

- Sequenzdiagramm definieren
- Architekturdiagramm definieren
- Domain Model definieren
- Proof-of-Concept (POC) erarbeiten

Construction-1

- Aus POC echte Applikation entwickeln
- Use-Cases umsetzen
- Oberflächen mit Xamarin.native umsetzen

Construction-2

- Core-Funktionalitäten in PlayFramework umgesetzt
- Use-Cases umsetzen
- Oberflächen mit Xamarin.native umsetzen

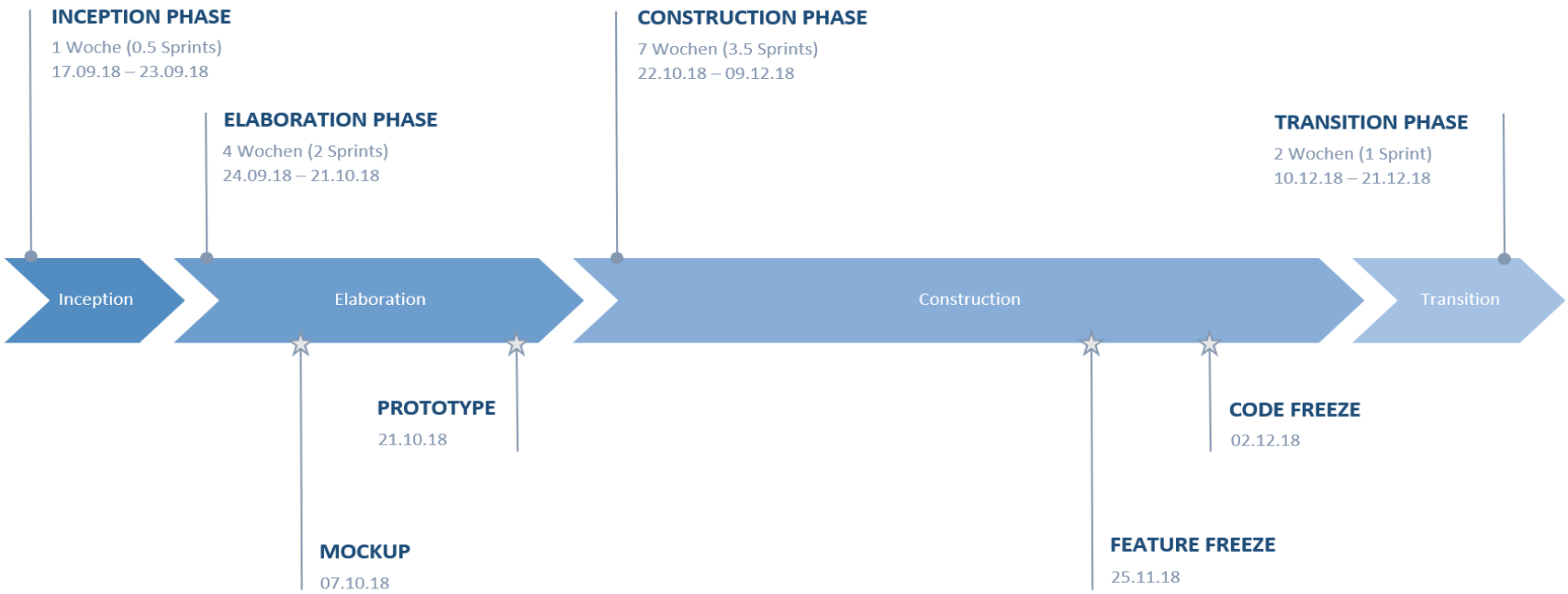

Abbildung 8: Projektplan

A.6 Risikotabelle

Risikomanagement

Projekt:	Methode 635 als Cross Plattform App mit Xamarin
Erstellt am:	21/09/2018
Autor:	Elias Brunner
Gewichteter Schaden:	46,95

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Anforderungen	Stark wechselnde Anforderungen	20	20%	4	Genaue Definition während der Elaboration und gute Planung der Iterationen	Änderungen einschätzen, überprüfen und eventuell Vornehmen
R2	Managment Tool	Arbeiten mit Jira zeitaufwändiger als erwartet da noch nie verwendet	20	10%	2	Früh genug mit Jira vertraut machen	Zusätzliche interne Schulung und erfahrene Kollegen fragen
R3	Kommunikation	Informationen werden nicht gut genug den Teammitgliedern mitgeteilt	15	15%	2,25	Frühzeitig abklären und genau mitteilen, wer was übernimmt	Zusätzliche Meetings um Ungenauigkeiten abzuklären
R4	Komplexität	Die Komplexität wurde unterschätzt. Der effektive Zeitaufwand übersteigt die Planung um ein Vielfaches	50	25%	12,5	Genaue Abschätzung der Komplexität mittels Story Points und der benötigten Zeit	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 existieren	16	20%	3,2	Internetanalyse. Gibt es bereits Projekte, die die angedachte Kombination bereits so einsetzen?	Inkompatible Komponenten ersetzen
R6	Entwicklungsumgebung	Kompatibilität der Entwicklungsumgebung oder Kentnisse derselben nicht aussreichend	10	10%	1	Auswahl der Umgebung für alle Mitglieder in Ordnung	Aushilfe bei Problemen, zusätzlich informieren
R7	Qualität	Code Guidelines, Qualitätsmanagment werden nicht eingehalten	20	10%	2	Guidelines einhalten, Tools für Überprüfung verwenden. Kontrolle bei Code-Reviews	Mehr Reviews und Gespräch mit Entwicklern
R8	Dokumentation	Erstellte Arbeiten werden nicht gut genug dokumentiert	20	20%	4	Alles verständlich dokumentieren und kommunizieren	Sensibilisierung der Mitglieder für Dokumentation

R9	Architektur skaliert nicht	Bei vielen 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
R10	Know-How	Fehlendes Know-How in den gewählten Programmiersprachen oder Arbeitsumgebungen	30	20%	6	Know-How vertiefen über verwendete Sprachen und Umgebungen	Know-How aufbauen
R11	Schwierige Umsetzung der Wireframes	Die in der Evaluations Phase erstellten Wireframes lassen sich mit Mobiletechnologien 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
Summe			251		46,95		

B Eigenständigkeitserklärung