Implementierungsdokumentation enCourage

Cole Bailey - Dominik Doerner - René Brandel - Tobias Röddiger

ImagineCup 2015 KIT Praxis der Softwareentwicklung Betreuer: Alexander Wachtel

Inhaltsverzeichnis

1. Einleitung

„enCourage“ ist eine moderne Notfallbenachrichtigungs-Applikation, mit der Benutzern die Möglichkeit gegeben wird, schnell und lautlos alle Personen in der näheren Umgebung über einen Notfall zu informieren.

Dieses Dokument beschreibt dabei den Fortschritt und die Änderungen der Implementierungsphase der Entwicklung dieser Applikation und baut direkt auf das vorangegangene Pflichtenheft und Entwurfsdokument auf, auf welche im Folgenden des Öfteren verwiesen wird.

Die App wurde für Windows Phone Geräte entwickelt und folgt dem Universal Apps Standard. Dementsprechend wurden alle genannten Klassen durchgehend objektorientiert in C# realisiert und nutzen für die Serverseite die von Microsoft bereitgestellten Dienste und APIs von Windows Azure.

Die Einteilung der Klassen in Pakete wurde funktionsorientiert durchgeführt und folgt den Vorgaben des Entwurfsdokumentes. Im Allgemeinen wurde eine möglichst direkte Umsetzung der Entwürfe angestrebt, auch wenn Hilfsmethoden sowie private Attribute zum Teil bewusst aus dem Entwurf ausgelassen wurden und sich deshalb in diesem Dokument unter den beschrieben Änderungen befinden. Größere Änderungen des Entwurfs, die über die innere Arbeit einer Klasse hinausgehen, waren nur an wenigen Stellen erforderlich und werden im Folgenden aufgezählt.

Dieses Dokument stellt zur Übersicht noch einmal Teile der vorangegangen Planung dar, um den Vergleich zu erleichtern, bevor im nächsten Kapitel alle Änderungen dokumentiert werden. In der Beschreibung der Klassen und der Diagramme sind dabei alle öffentlichen und privaten Attribute und Methoden der Vollständigkeit halber aufgezählt, die einzigen Auslassung betreffen die privaten Attribute hinter öffentlichen Properties sowie umbenannte Methoden (vorausgesetzt die Parameter haben sich nicht geändert).

Zur Unterscheidung der Änderungen in den Diagrammen wurden verschiedene Farben genutzt:

- Blaue Elemente markieren dabei neue oder stark geänderte Methoden/Klassen/Attribute
- Rot wurde benutzt, um entfernte Elemente zu markieren
- Alle grauen Markierungen sind dagegen direkte Umsetzungen aus dem Entwurfsdokument ohne nennenswerte Änderungen

Abbildung 1.1 – Farbmarkierungen in den Diagrammen

2. Vorangegangene Planung

Dieses Kapitel dient einer kurzen Übersicht der wichtigsten Elemente der vorangegangenen Planung, damit im Laufe des Dokuments auf diese Bezug genommen werden kann.

2.1. Kriterien (Planungsphase)

Hinweis: Der Übersicht wegen sind hier Muss- und Wunschkriterien zusammen aufgeführt. Sie sind weiterhin anhand der Kürzel voneinander unterscheidbar (KM = Muss-Kriterium, KW = Wunschkriterium).

1 – Melden eines Notfalls

[KM1010]	Eine Person kann einen Notfall sowohl über die App als auch das LiveTile melden und jederzeit wieder beenden.
[KM1020]	Nach dem Melden eines Notfalls wird der Melder kontinuierlich vom Server verfolgt.
[KM1030]	Mit dem Melden eines Notfalls wird ein Alarm ausgesendet, welcher nach einem festen Zeitlimit wieder verstummt, falls der Melder diesen nicht verlängert.
[KM1040]	Der Alarm endet automatisch, wenn der Melder den Notfall beendet oder eine gewisse Anzahl Helfer erreicht ist.
[KM1050]	Ein Notfall wird nach einem festen Zeitlimit automatisch beendet, falls der Melder das nicht tut.
[KW1010]	Der Standort des Melders kann auf Wunsch für alle verknüpften Personen oder alle Informierten stetig aktualisiert werden.
[KW1020]	Der Benutzer kann die Behörden (Polizei, Notarzt, …) über die App informieren.
[KW1030]	Falls niemand auf einen Notfall reagiert, vergrößert sich der Benachrichtigungsradius nach einer festen Zeit um eine feste Größe.
[KW1040]	Falls bereits ein Notfall innerhalb eines festen Radius gemeldet wurde, wird über ein Popup nachgefragt, ob es sich um denselben Notfall handelt.

2 – Notfalldetails

[KM2010]	Der Melder kann die Details eines Notfalls (außer GPS-Position) sowohl vor als auch nach dem Melden spezifizieren, editieren oder löschen.
[KM2020]	Informierte können alle Details eines Notfalls einsehen, einschließlich der Anzahl der Helfenden.
[KM2030]	Informierte können die Position des Melders einsehen oder in einer Navigations-App verwenden.
[KW2010]	Informierte sehen die Anzahl der verschiedenen Experten, die angegeben haben, auf dem Weg zu sein.

3 – Notfall-Benachrichtigungen erhalten

[KM3010]	Ein Alarm versendet Push-Benachrichtigungen über den Notfall an alle Personen, die sich in der näheren Umgebung des Notfalls befinden oder diese betreten.
[KM3020]	Beim Erhalt einer Benachrichtigung kann der Benutzer diese entweder per Button ignorieren oder angeben, dass er sich auf den Weg macht.

[KM3030]	Informierte können Notfälle als Missbrauch melden.
[KM3040]	Über die Push-Benachrichtigung gelangt der Benutzer direkt zu den Notfalldetails.

4 – App-Einstellungen

[KM4010]	Der Benutzer kann die App in einen Ruhemodus schalten, in dem er weder getrackt wird, noch Push-Benachrichtigungen erhält.
[KM4020]	Der Benutzer kann eine Sicherheitsfrage für privilegierte Aktionen wie das Beenden eines Notfalls einstellen, ändern oder löschen.
[KM4030]	Beim ersten Benutzen der App wird der Benutzer gefragt, die AGB sowie besondere Berechtigungen für die GPS-Ortung zu akzeptieren.
[KM4040]	Die App ist in Englisch verfügbar.
[KW4010]	Personen mit besonderen Qualifikationen (Erste Hilfe, Kampfsport, Brandschutz, Deeskalation) können diese in ihrem Profil angeben.
[KW4020]	Der Benutzer kann die (Standby-)Aktualisierungsrate der eigenen Position ändern.
[KW4030]	Der Benutzer kann den Ruhemodus für eine gewisse Zeitspanne (z.B. nachts) aktivieren.
[KW4050]	Falls bei der Verwendung der App die GPS-Funktionalität auf dem Gerät deaktiviert ist, wird der Benutzer über ein Pop-Up gefragt, diese direkt in den Einstellungen zu ändern.
[KW4060]	Die App ist zusätzlich in Deutsch und Chinesisch verfügbar.
[KW4070]	In der App ist ein Tutorial-Video verfügbar.

5 – Verknüpfte Personen

[KM5010]	Der Benutzer kann verknüpfte Personen umbenennen, löschen oder durch Eingabe des Profilcodes neu hinzufügen.
[KM5020]	Mit dem Melder verknüpfte Personen werden standortunabhängig über einen Notfall benachrichtigt.
[KW5010]	Vor dem Ablauf des Alarm-Zeitlimits eines Notfalls wird auch den verknüpften Personen die Möglichkeit gegeben, das Zeitlimit zu verlängern.

6 – LiveFeed

[KM6010]	Es gibt eine Liste mit allen aktuell nicht verstummten Notfällen in der größeren Umgebung (LiveFeed).
[KM6020]	Der Benutzer kann im LiveFeed einen Notfall auswählen und gelangt somit zu den Notfall-Details mit allen Informationen und der Möglichkeit, zu helfen.
[KW6010]	Im LiveFeed ist zusätzlich eine Kartenansicht verfügbar.

7 – Serververbindung

[KM7010]	Die Position aller aktiven Personen wird in festen Zeitabständen verfolgt.

2.2. Klassendiagramm (Entwurfsphase)

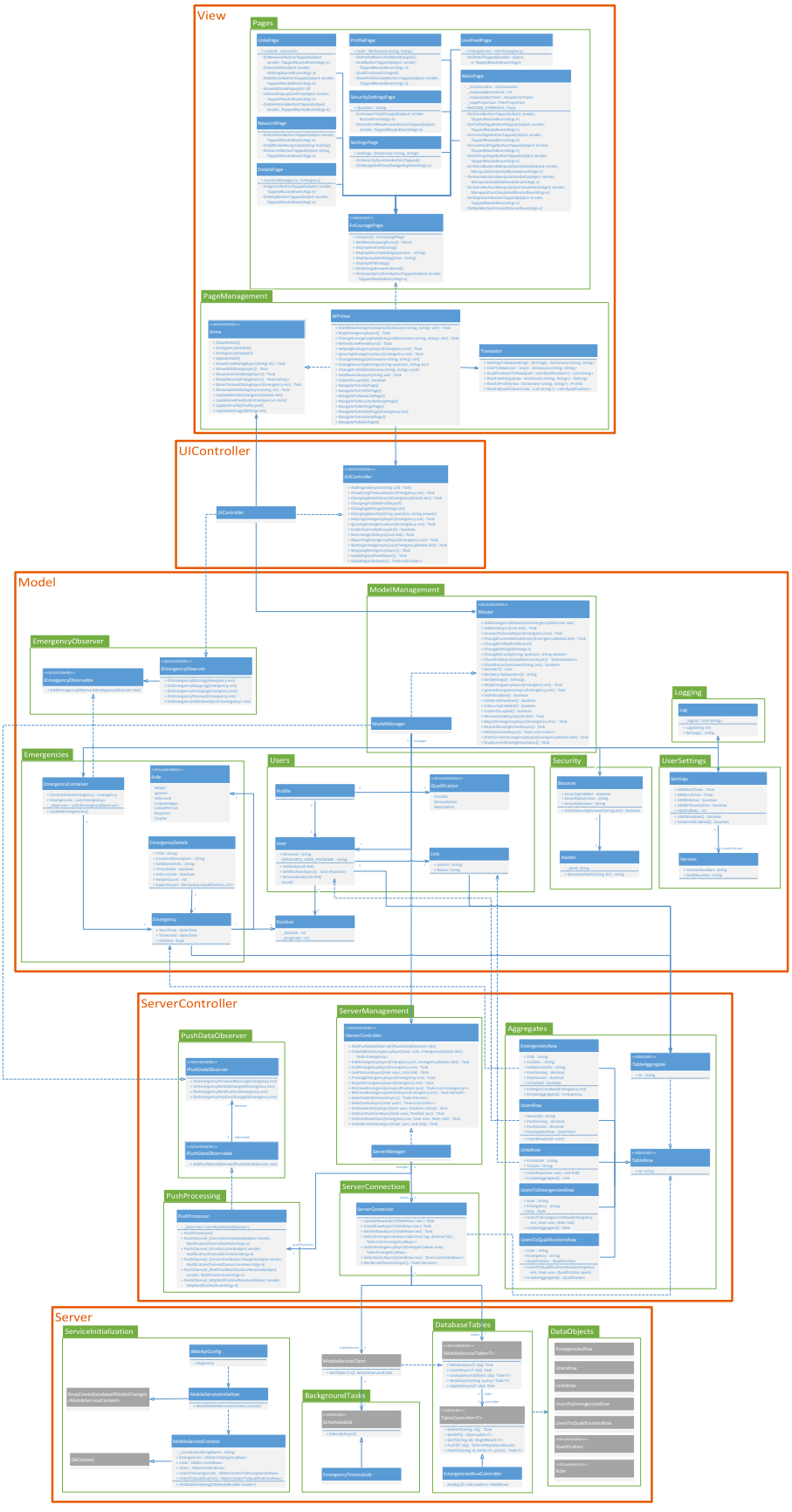


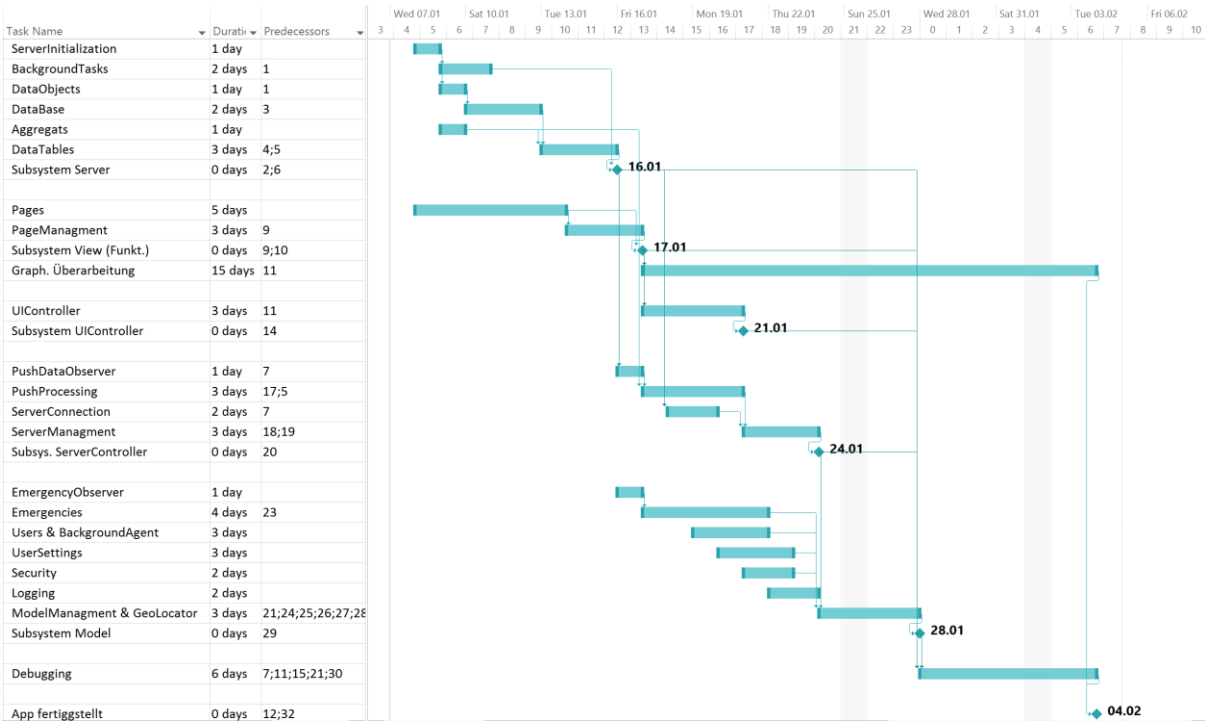
Abbildung 2.1 – Klassendiagramm der Entwurfsphase

2.3. Gantt-Diagramm (Entwurfsphase)

Die Durchführung der Implementierungsphase war als Outside-In-Implementierung angesetzt, die stückweise den Subsystemen folgen sollte. Dementsprechend sollten je zwei Team-Mitglieder an Server und View beginnend programmieren und sich danach zu den anderen Subsystemen weiterarbeiten.

Aufgrund der Wichtigkeit des Servers sollte dieser dabei möglichst schnell implementiert sein, um direkt Daten von dort verwenden zu können. Eine niedrige Fehlerquote war dabei nebensächlich, um die Entwicklung der anderen Subsysteme nicht zu weit aufzuhalten.

Um im Gegenzug aber die Qualität und Funktionstüchtigkeit nicht zu gefährden, wurde eine längere Debugging-Phase am Schluss angesetzt, zu der die Kernimplementierung abgeschlossen sein sollte, um sich auf Fehlerentfernung zu konzentrieren und um als möglicher Puffer zu dienen.



Task Name	Duratii	Predecessors
ServerInitialization	1 day	
BackgroundTasks	2 days	1
DataObjects	1 day	1
DataBase	2 days	3
Aggregats	1 day	
DataTables	3 days	4;5
Subsystem Server	0 days	2;6
Pages	5 days	
PageManagment	3 days	9
Subsystem View (Funkt.)	0 days	9;10
Graph. Überarbeitung	15 days	11
UIController	3 days	11
Subsystem UIController	0 days	14
PushDataObserver	1 day	7
PushProcessing	3 days	17;5
ServerConnection	2 days	7
ServerManagment	3 days	18;19
Subsys. ServerController	0 days	20
EmergencyObserver	1 day	
Emergencies	4 days	23
Users & BackgroundAgent	3 days	
UserSettings	3 days	
Security	2 days	
Logging	2 days	
ModelManagment & GeoLocator	3 days	21;24;25;26;27;28
Subsystem Model	0 days	29
Debugging	6 days	7;11;15;21;30
App fertiggstellt	0 days	12;32

Abbildung 2.2 - Gantt-Diagramm des geplanten Ablaufs

3. Geänderte Daten

3.1. Geänderte Entwürfe

[E01] Singleton-Entwurfsmuster für Pages (View)

In der Entwurfsphase entschieden wir uns für ein Singleton Entwurfsmuster, um die Pages anzusprechen. Der klassische Singleton beruht auf der Idee, einen privaten Konstruktor zu haben und eine statische Methode / Eigenschaft „Instance“ zu besitzen, die eine private statische Variable „*_instance*“ zurück gibt und diese initialisiert, wenn sie null ist. Ursprünglich wollten wir durch die *EnCouragePage* (die Oberklasse aller Seiten) die Instance Methode vererben lassen. Während der Implementierung fiel uns allerdings auf, dass das Betriebssystem einen öffentlichen Konstruktor erzwingt. Daraufhin wurden die Konstruktoren wieder auf öffentlich gesetzt und in den Konstruktoren die *_instance* Variable auf „*this*“ zugewiesen. Dies führt zu einer Art Pseudo-Singleton, in welchem auch immer ein valider Wert zurückgeliefert wird, allerdings kann dieser Wert auch vom Betriebssystem neu gesetzt werden.

Außerdem sollte die *Instance*-Methode vererbt werden, die generisch in der Klasse *EnCouragePage* enthalten ist, um die einzelnen Seiten über diesen Parameterwert zu identifizieren. Dies scheiterte ebenfalls, aus systemspezifischen Gründen. In Windows Universal Apps werden Seiten als eine „.xaml“ Datei und eine „.xaml.cs“ Datei gespeichert und angezeigt. Die „.xaml“ stellt statische Benutzeroberfläche Elemente dar. Unter Verwendung der Generics in der Oberklasse bereitete diese Datei Schwierigkeiten bei der Kompilierung. Deswegen mussten wir uns dazu entscheiden, in jeder Unterklasse von *enCouragePage* die Instance Methode und die private Variable *_instance* hinzuzufügen.

In den einzelnen Pages sind inzwischen mehr Methoden als im Entwurf vorgesehen, welche hauptsächlich aus EventHandlern bestehen, die über keine umfangreiche Logik verfügen und hauptsächlich der Navigation dienen. Die restlichen neuen Methoden sind aus Refactor-Gründen ausgelagert worden, um eine bessere Übersicht der Funktionalität zu erhalten.

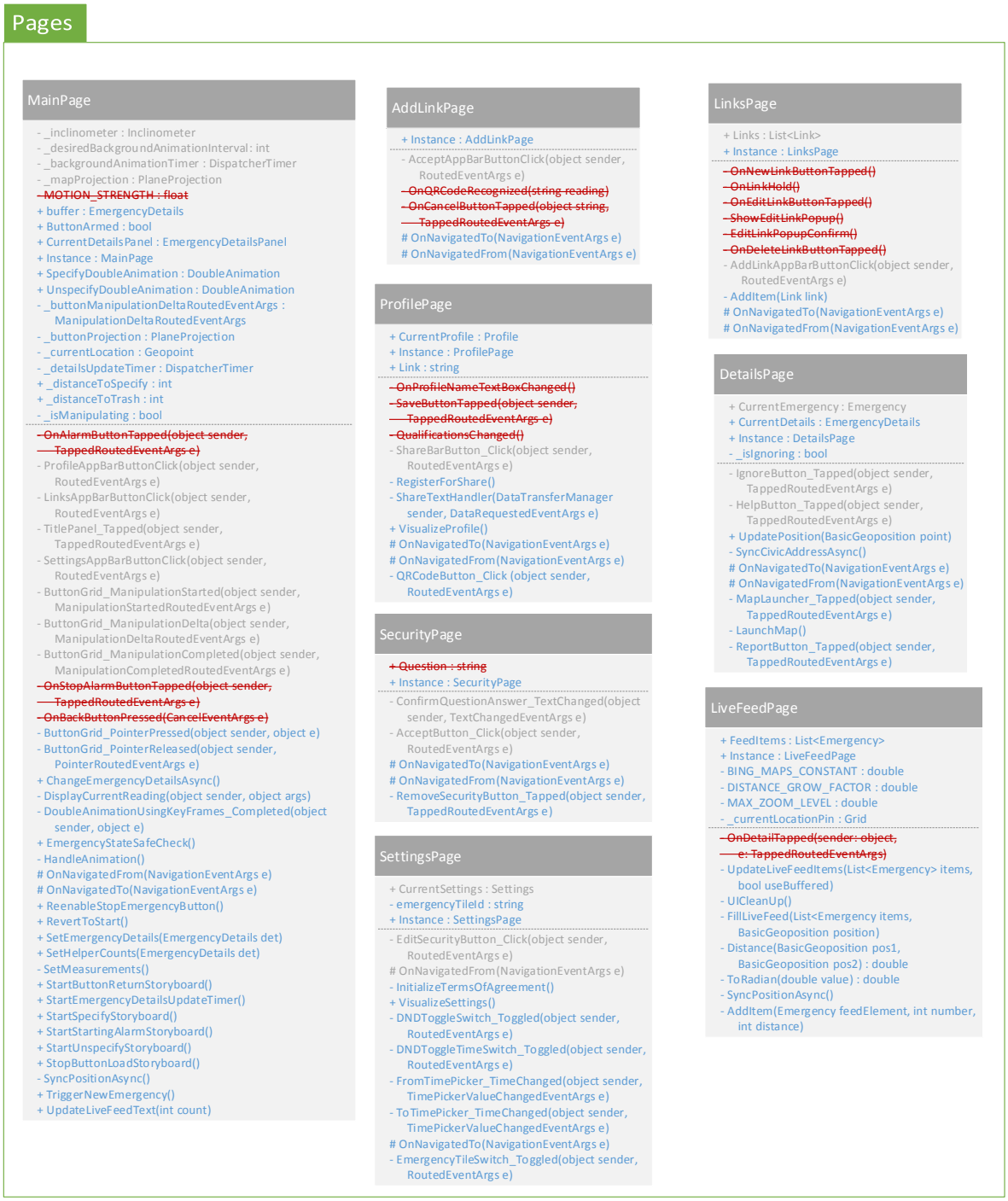


Abbildung 3.1 – Änderungen an den Seiten der UI

[E02] Zustands-Entwurfsmuster für Animationen (View)

Der AlarmButton auf der *MainPage* wird durch zwei verschiedene Manipulationstechniken in verschiedene Zustände gebracht.

So gibt es etwa EventHandler für den Fall, dass der Knopf verschoben (*ManipulationCompleted)* oder gedrückt wird (*PointerReleased)*. Bei beiden muss ausgewertet werden, wo sich der Knopf befindet und in welchen Zustand er danach überführt werden soll. Um einen guten Code-Stil beizubehalten, redundanten

Code zu vermeiden und eine komfortablere Fehlerbehebung zu gewährleisten, entschieden wir uns für diesen Entwurfsänderung.

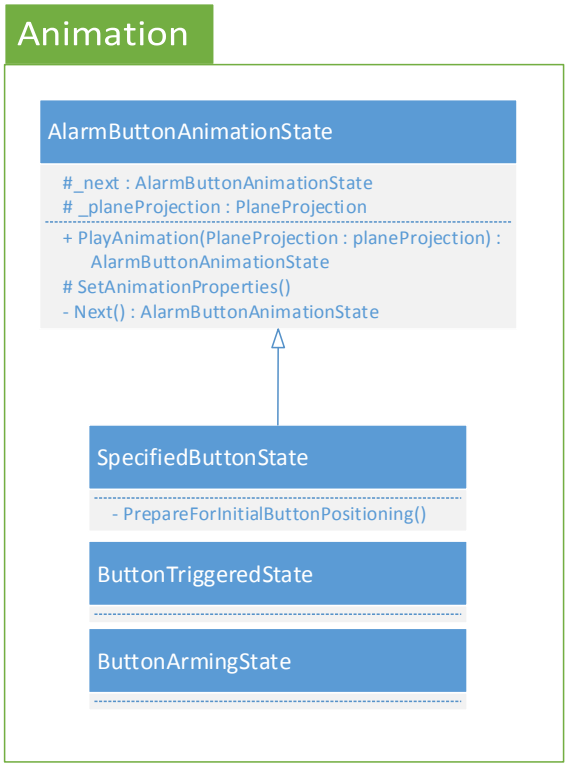


Abbildung 3.2 – Animations-Zustände der MainPage

Wir haben den Zustand des AlarmButtons in ein eigenes Zustandsentwurfsmuster ausgelagert. Den Zuständen wird jeweils die PlaneProjection (die die Position des Buttons enthält) übergeben und der Folgezustand in *SetAnimationProperties* (äquivalent zur Ausgansfunktion) bestimmt. Die *PlayAnimation* gibt den nächsten Zustand zurück, der im Konstruktor eine Eingangsfunktion besitzt.

[E03] Custom Controls (View)

Um eine bessere Übersicht über unsere Controls (z.B. HelperPanel, für die graphische Darstellung der Anzahl der Helfer) zu haben, haben wir diese in eigene Klassen ausgekapselt. Dies war unter anderem nötig, da diese mehrmals benutzt werden und es im Allgemeinen zu einen guten Code-Stil beiträgt. Deswegen haben wir auch ein neues Paket („Controls“) eingeführt, das diese UI-Elemente enthält. Diese benutzen auch Events von C#, welche abonnieren werden können, um somit eine beliebige Methode auszuführen.

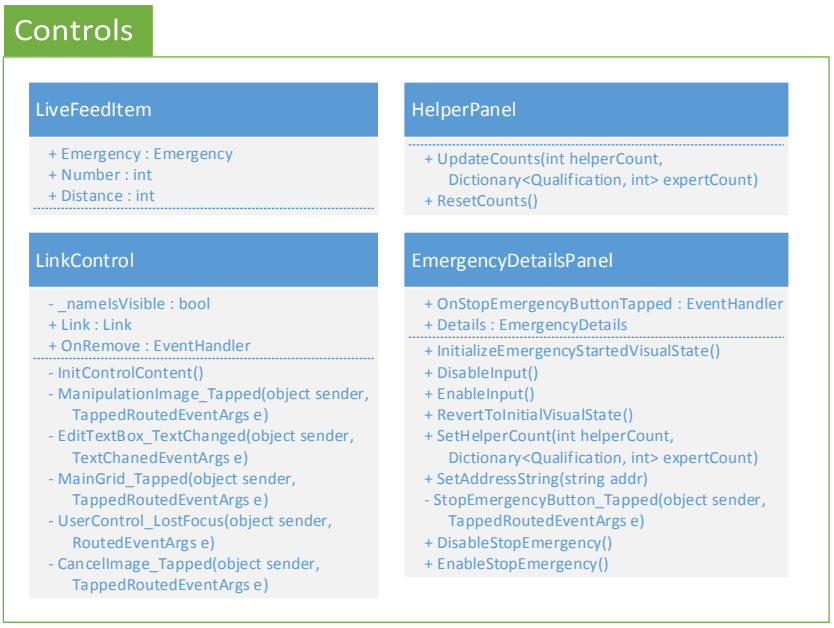


Abbildung 3.3 – Custom Controls für die View

Durch das Auskapseln der Controls entsteht auch eine Modularität in der View die nötig war, um eine leichte Fehlerbehebung zu ermöglichen.

[E04] Befehls-Entwurfsmuster (UIController)

Für die Realisierung der Benutzerauthentifizierung über die eingestellte Sicherheitsfrage war die Abfrage der Antwort vor dem Ausführen der eigentlichen privilegierten Methode (z.B. Notfall beenden) erforderlich.

Der Erweiterbarkeit wegen sollte diese Funktionalität theoretisch vor jedem Aufruf auf dem Model eingefügt werden können, wodurch eine Weitergabe der beabsichtigten Methode an den Sicherheitsdialog auf dem View notwendig wurde. In der Entwurfsphase war dabei eine asynchrone Rückgabe der Antwort an die aufrufende Methode geplant (denn diese Antwort muss noch zusätzlich überprüft werden).

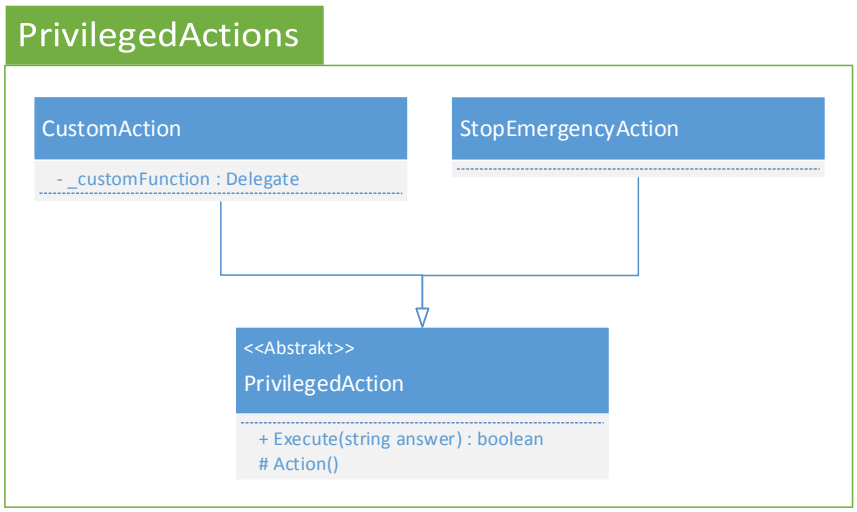


Abbildung 3.4 – Kapselung der privilegierten Aktionen als Befehle

Dies ist aufgrund der Dialog-API allerdings nicht möglich, wodurch wir nun ein neues Paket einfügten, in dem die privilegierten Aufrufe in Objekte nach dem Befehls-Entwurfsmuster gekapselt werden können, welche dem Dialog übergeben und danach von diesem aufgerufen werden. Durch die Ableitung von einer zentralen

Oberklasse (*PrivilegedAction*) ist damit auch die Überprüfung der Antwort vor dem Ausführen des Befehls möglich.

[E05] Serialisierung (Model)

Da die serialisierten Daten synchron zur Verfügung stehen müssen, ist es nicht möglich, eine Datei zu erstellen, in der die Klassen im XML-Format serialisiert werden, da die Deserialisierung dann nur asynchron möglich ist. Deswegen wurde die Klasse *UniversalSerializer* erstellt, welche ausschließlich die Local Settings des Windows Phone verwendet, um alle Daten synchron deserialisieren zu können.

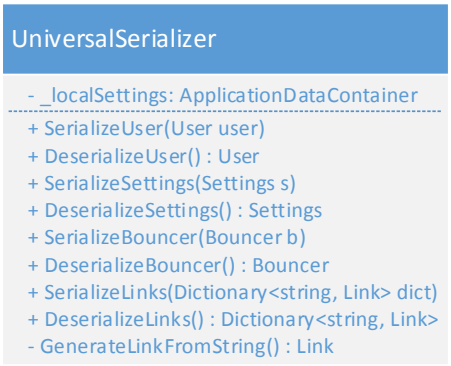


Abbildung 3.5 – UniversalSerializer als einheitliche Schnittstelle zum Hintergrundspeicher

Dadurch ist außerdem eine einheitliche Schnittstelle geschaffen, um alle persistenten Objekte auf dem Gerät des Benutzers speichern und abrufen zu können, was auch der Abarbeitung von Benachrichtigung ohne laufende Applikation oder die Realisierung des BackgroundAgents erleichtert.

[E06] Push-Notifications (Server/ServerController)

Das Versenden und Empfangen von Push Notifications wurde mit Hilfe der API von Azure Mobile Services und den dazugehörigen Notification Hubs bereitgestellt. Da das Versenden von Benachrichtigungen an mehrere Stellen der Datenbank gebraucht wurde, haben wir diese Funktionalität in einer eigenen Klasse (*UserNotifier*) gekapselt. Dort findet der Aufbau von Benachrichtigungen mit kodierten Daten *(_emergency, _userName,* …), das Setzen der Benachrichtigten (*AlsoInformLinks, _userIds,* …), und das Verschicken von spezialisierten Benachrichtigungen (*NotifyNewEmergencyAsync, NotifyDetailsChangedAsync,* ...) über den Notifcation Hub statt.

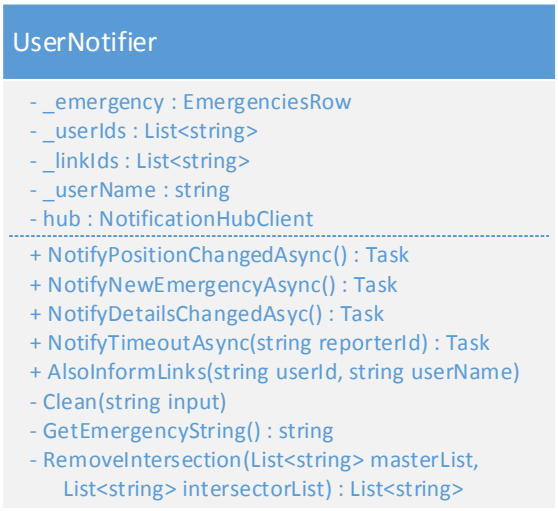


Abbildung 3.6 – UserNotifier als einheitliche Push-Schnittstelle

Das Empfangen der Benachrichtigungen wurde in der *PushProcessor* Klasse gekapselt (siehe Entwurfsdokument). Für eine konkrete Benachrichtigung einzelner Geräte über den aufgebauten PushChannel steht leider keine eindeutige Geräte-ID mehr zur Verfügung, sodass wir dies über eine Anmeldung im Notification Hub mit einem eindeutigen Tag simulieren. Um einem Benutzer auch die aktive Verfolgung aller Änderungen an den Details eines geöffneten Notfalls zu ermöglichen, verwenden wir ebenso Tags für die einzelnen Notfälle, um die Menge aller interessierten Nutzer geschlossen anzusprechen.

Diese Tags werden von den zugehörigen UserIDs und EmergencyIDs abgeleitet, was dafür sorgt, dass jedes Gerät unterscheidbar ist und sowohl gezielt als auch in der Menge angesprochen werden kann. Aus diesem Grund ist aber auch die Verwaltung dieser Tags entsprechend wichtig und umfangreich und resultierte in einigen Hilfsmethoden, die zum Großteil allerdings privat sind und nur die innere Arbeit der Klasse auftrennen.

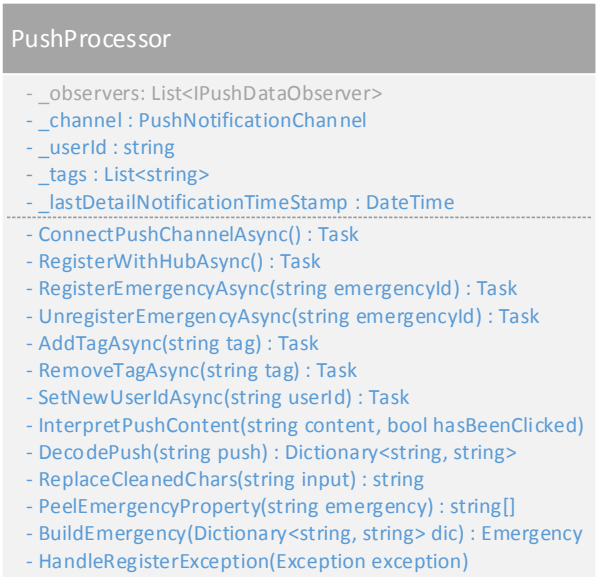


Abbildung 3.7 – PushProcessor als Benachrichtigungsempfänger

Die Methoden des *PushProcessors* werden nur aufgerufen, falls die Applikation gerade in Vordergrund läuft. Sollte die Applikation allerdings geschlossen sein und im Ruhemodus eine Benachrichtigung ankommen, sollte diese ignoriert und nicht angezeigt werden, wodurch ankommende Benachrichtigungen zuerst in der BackgroundComponent ausgewertet werden.

[E07] WebAPI (Server)

Im Server verlassen wir uns zu großen Teilen auf Unterklassen der vordefinierten TableController-Klasse zur einfachen Interaktion des Clients mit den Datenbanktabellen. Während der Implementierung wurde allerdings deutlich, dass diese sich weniger als gedacht erweitern lassen, um zusätzliche Funktionalitäten wie das Verlängern eines Zeitlimits oder das Melden eines Notfalles zu definieren. Auch wenn die Anpassung der Methoden mit verschiedenen Parametern funktioniert, sind die Methoden nur für CRUD-Operationen (Create, Read, Update, Delete) vorgesehen und des Stiles wegen wollten wir das nicht ändern.

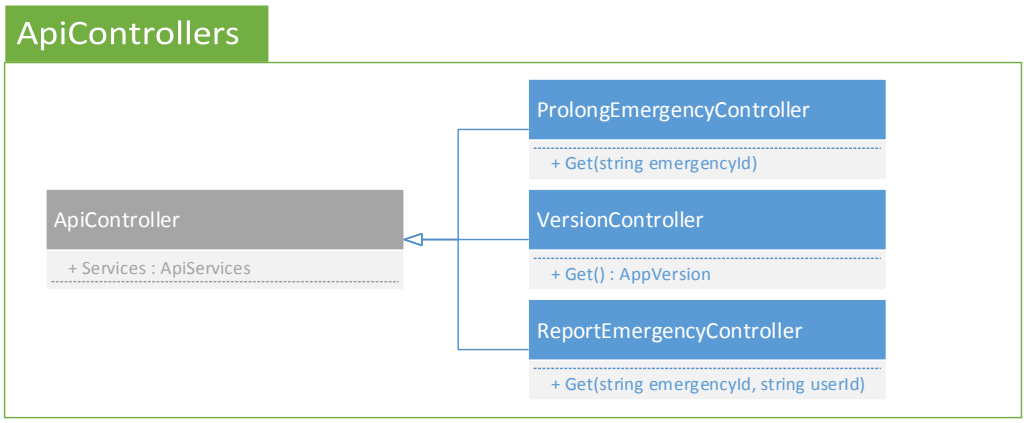


Abbildung 3.8 – Benötigte ApiController

Dadurch war eine größere Menge an eigens erstellten ApiController-Klassen erforderlich, um diese Funktionalitäten umzusetzen, da auch hier jede Funktion in einer eigenen Klasse gekapselt werden sollte. Daraus folgend muss auch im Client jede Funktion in eigenen Methoden aufgerufen werden.

Abbildung 3.9 – Ausschnitt der neuen API-Methoden im ServerConnector

Der vorangegangen Planung folgend haben wir allerdings versucht, die Aufrufe an die eigentliche API von Azure möglichst uniform zu halten, sodass auch der *ServerConnector* im Subsystem ServerController alle diese Aufrufe schlussendlich in einer Methode bündeln kann (*CallApiAsync*).

3.2. Geänderte Klassen

Hinweis: Klassen, die in den oben genannten geänderten Entwürfen beschrieben wurden, sind hier nicht mit aufgezählt.
Ebenso sind leicht umbenannte Methoden und Attribute, die immer noch dieselben Parameter/Typen haben, nicht markiert oder weiter beschrieben (sofern die Änderung sofort ersichtlich ist).

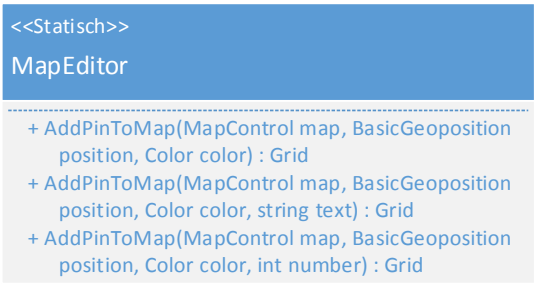


Abbildung 3.10 – Geänderte Klasse MapEditor

MapEditor

Art der Änderung: Neue Methoden

Paket: View.Pages.Imaging

Beschreibung: Der *MapEditor* dient der Darstellung visueller Signale auf einer Karte. Im Rahmen der Implementierungsphase unterstützt diese Klasse runde Pins (mit benutzerdefinierte Farbe) mithilfe einer *BasicGeoposition* auf einer Karte zu zeichnen und entweder darüber Texte anzuzeigen oder eine Nummer in den Kreisen zu setzen. Diese Klasse wird an mehreren Stellen im View verwendet, da Karten ein fundamentaler Bestandteil der UI sind.

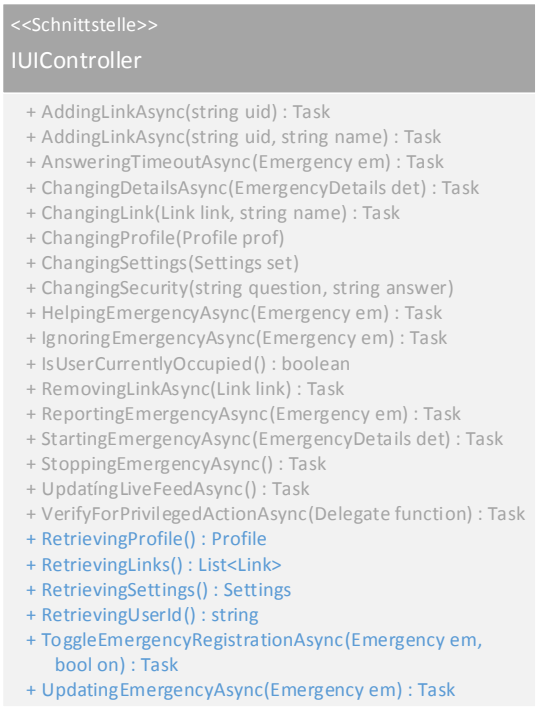


Abbildung 3.11 – Geänderte Schnittstelle IUIController

IUIController

Art der Änderung: Neue Methoden

Paket: UIController

Beschreibung: Da Daten nicht nach dem Push-Prinzip an den View weitergeben werden können, sondern von jeder Seite angefordert werden (siehe [E01] Singleton-Entwurfsmuster), sind für diesen Zweck weitere Methoden nötig (*UpdatingEmergencyAsync, Retrieve…*).

Ebenso ist eine Methode (*ToggleEmergency-RegistrationAsync*) zum An- und Abmelden von Notfällen (siehe [E06] Push-Notifications) erforderlich.

Abbildung 3.12 – Geänderte Schnittstelle IEmergencyObserver

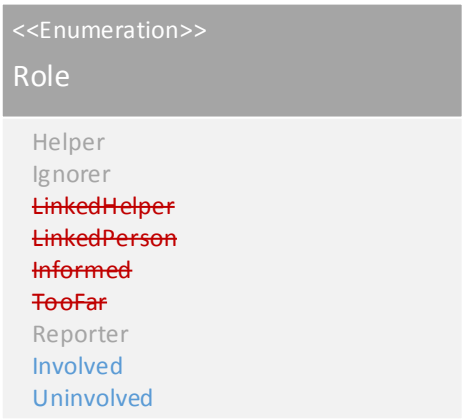


Abbildung 3.13 – Geänderte Enumeration Role

IEmergencyObserver

Art der Änderung: Neue Methoden

Paket: Model.EmergencyObserver

Beschreibung: Die *OnEmergencyNotificationReceived*-Methode ermöglicht es, einen neuen Notfall über den UIController an die View weiterzuleiten.

Der Parameter *navigatedTo* speichert dabei, ob der Benutzer direkt zur Detailseite weitergeleitet (z.B. nach einem Drücken auf die Toast-Notification) werden soll.

Role (Model)

Art der Änderung: Geänderte Werte

Paket: Model.Emergencies

Beschreibung: Die Unterscheidung zwischen verknüpften Personen und andere Benachrichtigten findet nicht mehr in der Rolle statt, da sich die Verwendungen nur wenig unterscheiden und schlecht erweiterbar sind. Die *Linked*- Rollen fallen deswegen aus dieser Enumeration weg.

Die Verwendung von *Informed* und *TooFar* für aktive und inaktive Beobachter eines Notfalls war weiterhin wichtig, diese wurden allerdings zu *Involved* und *Uninvolved* umbenannt, um Verwirrung zu vermeiden.

Die Funktionalität dahinter wurde zudem ausgedehnt, sodass mit diesen Rollen auch eine An- und Abmeldung von den Notfallbenachrichtigungen für einen konkreten Notfall verbunden ist (siehe [E06] Push Notifications).

Abbildung 3.14 – Geänderte Klasse Emergency

Emergency

Art der Änderung: Neue Methoden und Attribute

Paket: Model.Emergencies

Beschreibung: Das Attribut *LinkedReporter* wurde dem Emergency hinzugefügt, da eine Person, die mit dem Gerät verknüpfte wurde, wissen muss, von wem der Notfall stammt über den sie eine Benachrichtigung erhält. Da die Links nur einseitig sind ist es notwendig diesen Namen mitzusenden, da der Empfänger durch die ID des Melders nicht auf diesen schließen kann. Ist ein Emergency nicht von einer verknüpften Person, so ist *LinkedReporter* einfach ein leerer String.

Die *Update*-Methode wurde hinzugefügt, um einen Notfall und dessen Attribute im *EmergencyContainer* bzw. *ModelManager* einfacher zu aktualisieren, wenn Daten vom Server eintreffen, in denen einige Werte fehlen können (z.B. die Rolle des Benutzers).

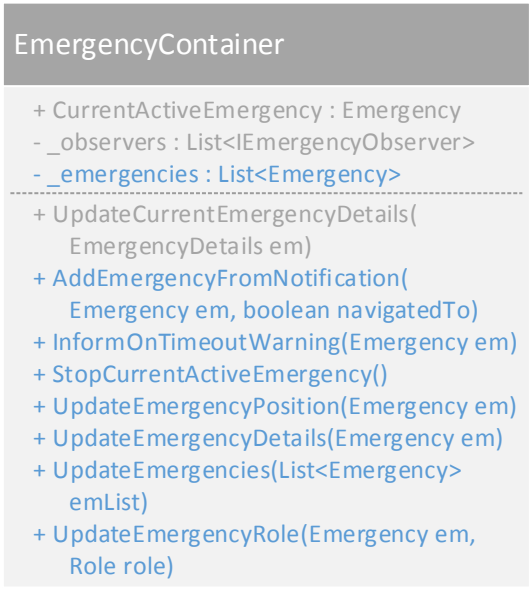


Abbildung 3.15 – Geänderte Klasse EmergencyContainer

EmergencyContainer

Art der Änderung: Neue Methoden und Attribute

Paket: Model.Emergencies

Beschreibung: Der *EmergencyContainer* nutzt nun ein Dictionary anstelle einer Liste um die Zugriffszeit auf einen Emergency zu optimieren. So können bei Aktualisierungen einfach die Emergency-Objekte über die ID aus dem Dictionary geholt werden.

Die restlichen neuen Methoden werden benötigt, da Aufrufe im Model, die Änderungen auf dem *EmergencyContainer* bewirken, zusätzlich an die Beobachter weitergegeben werden müssen, sodass die Änderung aller wichtigen Attribute und Werte durch Methodenaufrufe realisiert werden müssen (dadurch lässt sich auch Funktionalität aus dem *ModelManager* auslagern).

Abbildung 3.16 – Geänderte Klasse Profile

Profile

Art der Änderung: Neue Attribute

Paket: Model.Users

Beschreibung: Der neu hinzugefügte *DisplayName* wird beim Melden eines Notfalls für die Links im Emergency Objekt mitgesendet und repräsentiert den Namen des Nutzers der App (siehe Emergency). Dieser wird lokal auf dem Handy und nicht auf dem Server gespeichert, da die Anonymität der Nutzer weiterhin gewahrt werden soll.

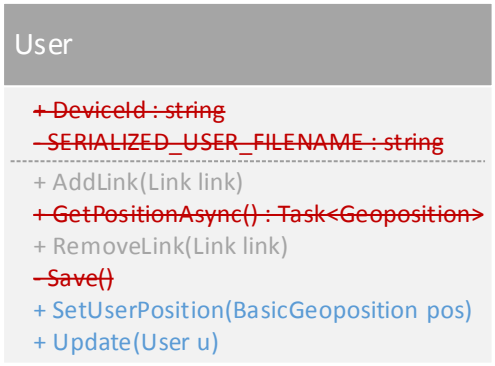


Abbildung 3.17 – Geänderte Klasse User

User

Art der Änderung: Geänderte Methoden und Attribute

Paket: Model.Users

Beschreibung: Die *DeviceId* wird nicht benötigt. Die ID erhält der User über das Erben von TableAggragte.

Die Serialisierung wurde in den *UniversalSerializer* ausgelagert weshalb die *SERIALIZED_USER_FILENAME* und die *Save*-Methode nicht mehr benötigt werden.

Die Abwicklung der Positionsaktualisierung läuft ausschließlich über den *ModelManager*; deswegen ist die *GetPositionAsync*-Methode überflüssig und es wurde dem User eine *SetUserPosition* Methode hinzugefügt, welche aufgerufen wird, sobald der Geolocator im *ModelManager* festgestellt hat, dass sich die Position geändert hat.

Die *Update*-Methode folgt dabei demselben Prinzip der gleichnamigen Methode in der Emergency-Klasse.

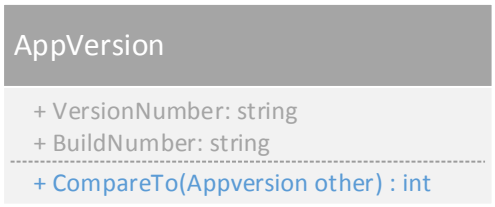


Abbildung 3.18 – Geänderte Klasse AppVersion

AppVersion (Model)

Art der Änderung: Neue Methoden

Paket: Model.UserSettings

Beschreibung: Die *CompareTo*-Methode wurde hinzugefügt um zwei verschiedene Versionen zu vergleichen. Es wird eine eigene Methode benötigt da die Version dem Format 0.0.0 entspricht.

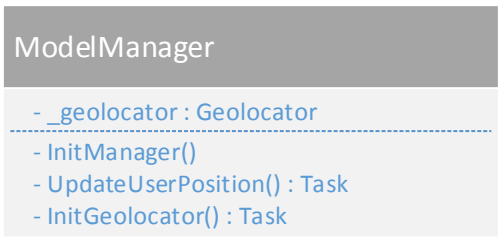


Abbildung 3.19 – Geänderte Klasse ModelManager

ModelManager

Art der Änderung: Neue Methoden und Attribute

Paket: Model

Beschreibung: Dem *ModelManager* wurde ein *_geolocator* Attribut hinzugefügt welches zur Positionsbestimmung genutzt wird.

Die *InitManager*- und *InitGeolocator*-Methoden initialisieren jeweils den *ModelManager* (Deserialisierung alter Einstellungen, Registrierung des Benutzers im Server, holen eines alten Notfalls in dem der Benutzer Reporter war und der beim letzten schließen der App noch nicht beendet wurde) und den *_geolocator* (Genauigkeit, *OnPositionChangedEvent*).

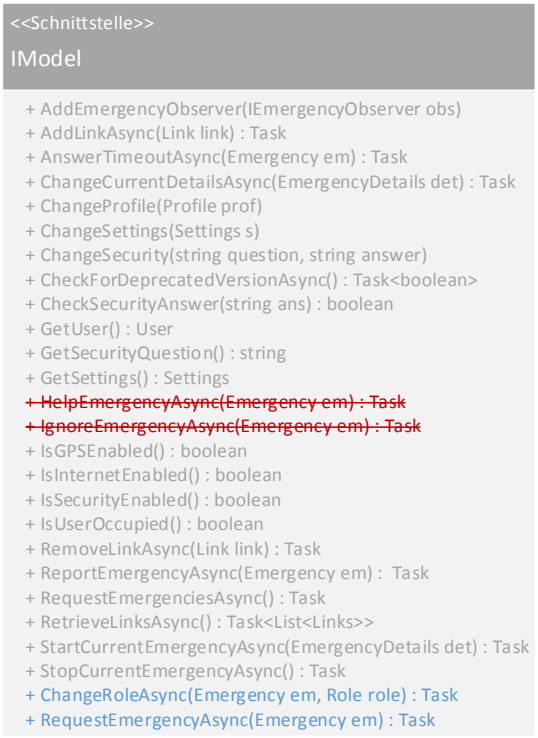


Abbildung 3.20 – Geänderte Schnittstelle IModel

IModel

Art der Änderung: Neue Methoden

Paket: Model

Beschreibung: Die Schnittstellen-Methoden *Help-EmergencyAsync* und *IgnoreEmergencyAsync* wurden durch die Methode *ChangeRoleAsync* ersetzt, welche die Funktionalität der beiden alten Methoden übernimmt und vereinheitlicht.

Die Methode *RequestEmergencyAsync* ermöglicht es, einen einzelnen Emergency vom Server anzufordern und somit zu aktualisieren.

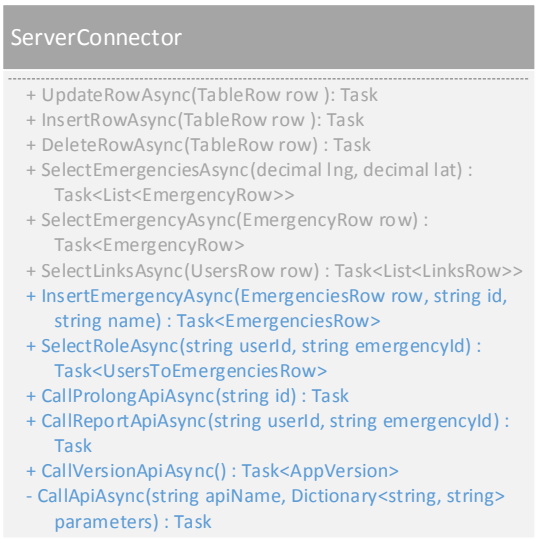


Abbildung 3.21 – Geänderte Klasse ServerConnector

ServerConnector

Art der Änderung: Neue Methoden

Paket: ServerController.ServerConnection

Beschreibung: Im Gegensatz zu allen anderen Datenbanktabellen müssen bei der Einfügung eines neuen Notfalls weitere Parameter übergeben werden (z.B. ID des Melders), wodurch eine zusätzliche Methode nötig war (*InsertEmergencyAsync*), genauso wie beim Abfragen der aktuellen Rolle (*SelectRoleAsync*) über das Benutzer/Notfall-Paar.

Die weiteren Methoden sind in [E07] beschrieben.

Abbildung 3.22 – Geänderte Klasse IServerController

IServerController

Art der Änderung: Geänderte Methoden

Paket: ServerController

Beschreibung: Durch die Entfernung des Attributs *IsActive* aus der Klasse UsersRow (Server) ist auch die Methode zum Ändern dieses Attributs nicht mehr erforderlich (*SetUserActivityAsync*).

Dafür haben wir eine weitere Methode zum Aktualisierung eines Notfalls eingefügt (*Retrieve-EmergencyAsync*).

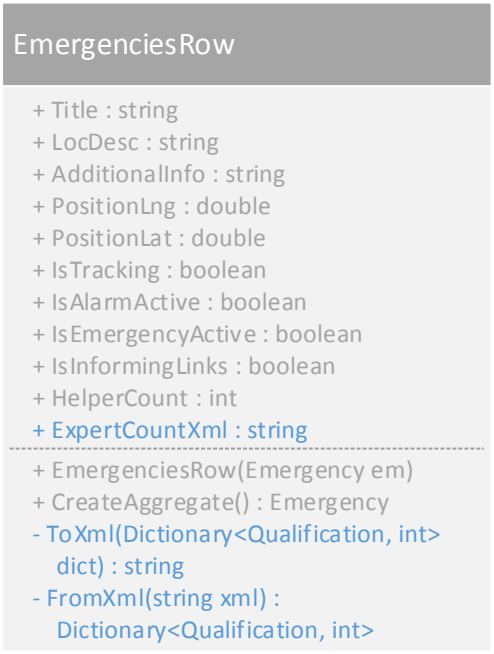


Abbildung 3.23 – Geänderte Klasse EmergenciesRow

EmergenciesRow

Art der Änderung: Geänderte Attribute

Paket: ServerController.Aggregates

Beschreibung: Sowohl das Versenden über das Netzwerk als auch das Speichern in der Datenbank ist mit einem Dictionary-Objektes nicht möglich, sodass wir das Attribut ExpertCount als serialisierten XML-String verwenden müssen (*ExpertCountXML*).

Aus diesem Grund sind auch Hilfsmethoden zur (De-)Serialisierung enthalten (*ToXml, FromXml*).

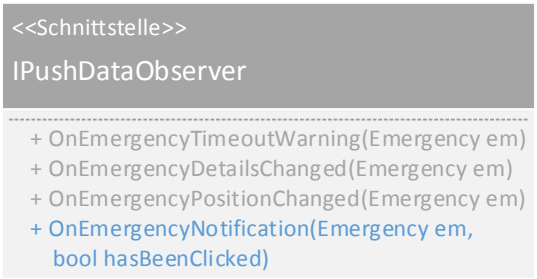


Abbildung 3.24 – Geänderte Klasse IPushDataObserver

IPushDataObserver

Art der Änderung: Geänderte Methoden

Paket: ServerController.PushDataObserver

Beschreibung: Die *OnEmergencyNotification*-Methode ermöglicht es, einen neuen Notfall an das Model weiterzuleiten, um vor der Anzeige der Benachrichtigung den Zustand des Ruhemodus abzufragen.

Im Parameter *hasBeenClicked* wird dabei gespeichert, ob der Benutzer bereits auf die Benachrichtigung geklickt hat.

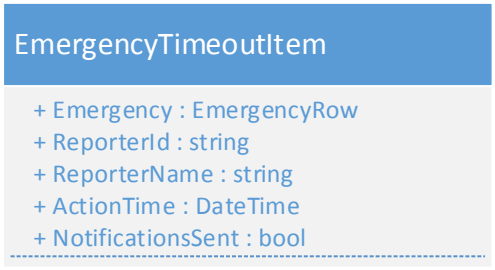


Abbildung 3.25 - Geänderte Klasse EmergencyTimeoutItem

EmergencyTimeoutItem

Art der Änderung: Neue Klasse

Paket: *Server.DataObjects*

Beschreibung: Wieder Erwarten können in Azure leider nicht mehrere geplante Aufträge gestartet werden, um für jeden Notfall je ein Zeitlimit zu realisieren.

Stattdessen speichern wir diese Aufträge nun als Datenbankobjekte und verwenden einen einzigen wiederkehrenden Auftrag, der all diese ausliest, auswertet und verarbeitet.

Abbildung 3.26 - Geänderte Klasse AppVersion

AppVersion (Server)

Art der Änderung: Neue Klasse

Paket: *Server.DataObjects*

Beschreibung: Wir haben uns für die Speicherung der Version im Server nun für die Verwendung derselben Klasse wie im Client entscheiden.

Abbildung 3.27 - Geänderte Klasse Role

Role (Server)

Art der Änderung: Entfernte Klasse

Paket: *Server.DataObjects*

Beschreibung: Da die Rolle im Server aufgrund der Beschränkung des EF (Entity Framework) 6 durchweg als string gespeichert ist, wird eine serverseitige Enumeration nicht mehr benötigt.

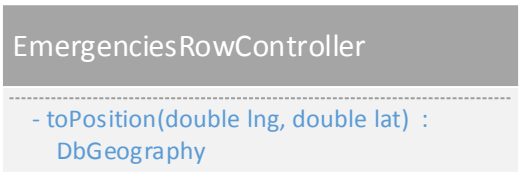


Abbildung 3.28 – Geänderte Klasse EmergenciesRowController

EmergenciesRowController

Art der Änderung: Neue Methoden

Paket: Server.Controllers.TableControllers

Beschreibung: Die recht komplexe Erstellung eines DbGeography-Objektes ist nun in einer eigenen Hilfsmethode gekapselt.

3.3. Nicht umgesetzte Kriterien

[KW1020] - *Der Benutzer kann die Behörden (Polizei, Notarzt, …) über die App informieren.*

Ein reines Informieren der Behörden wäre je nach Infrastruktur (SMS, …) kein großes Problem, aber nicht sehr gut erweiterbar. Stattdessen haben wir recht große Pläne, das Projekt im Zusammenspiel mit den Behörden auszuweiten, was aber den Rahmen des jetzigen Praktikums sprengen würde (siehe [KZ6010]).

[KW1030] - *Falls niemand auf einen Notfall reagiert, vergrößert sich der Benachrichtigungsradius nach einer festen Zeit um eine feste Größe.*

Bisher ist eine nachträgliche Benachrichtigung von Benutzern, die den Radius betreten haben, nicht implementiert, sodass auch eine Vergrößerung dieses Radius bisher nicht sinnvoll ist.

[KW1040] - *Falls bereits ein Notfall innerhalb eines festen Radius gemeldet wurde, wird über ein Popup nachgefragt, ob es sich um denselben Notfall handelt.*

Wie in der Planungsphase besprochen gibt es mit diesem Kriterium einige Fragen, die zunächst nicht geklärt werden konnten, sodass wir diese Entscheidung auf einen späteren Zeitpunkt verschoben haben.

[KW4020] - *Der Benutzer kann die (Standby-)Aktualisierungsrate der eigenen Position ändern.*

Da das Betriebssystem sowieso alle 30 Minuten die Position abfragt und die Hintergrundaufgaben abarbeitet, ist eine häufigere Aktualisierung nicht möglich und eine seltenere Aktualisierung nicht praktisch, sodass wir dieses Kriterium ausgelassen haben.

[KW4070] - *In der App ist ein Tutorial-Video verfügbar.*

Ein Video wäre zu viel Arbeit für den Umfang des Projektes und hätte auch recht wenig mit Software-Entwicklung zu tun.

3.4. Klassendiagramm (Implementierungsphase)

Abbildung 3.29 – Klassendiagramm der Implementierungsphase

4. Ergebnisse der Phase

4.1. Ablauf

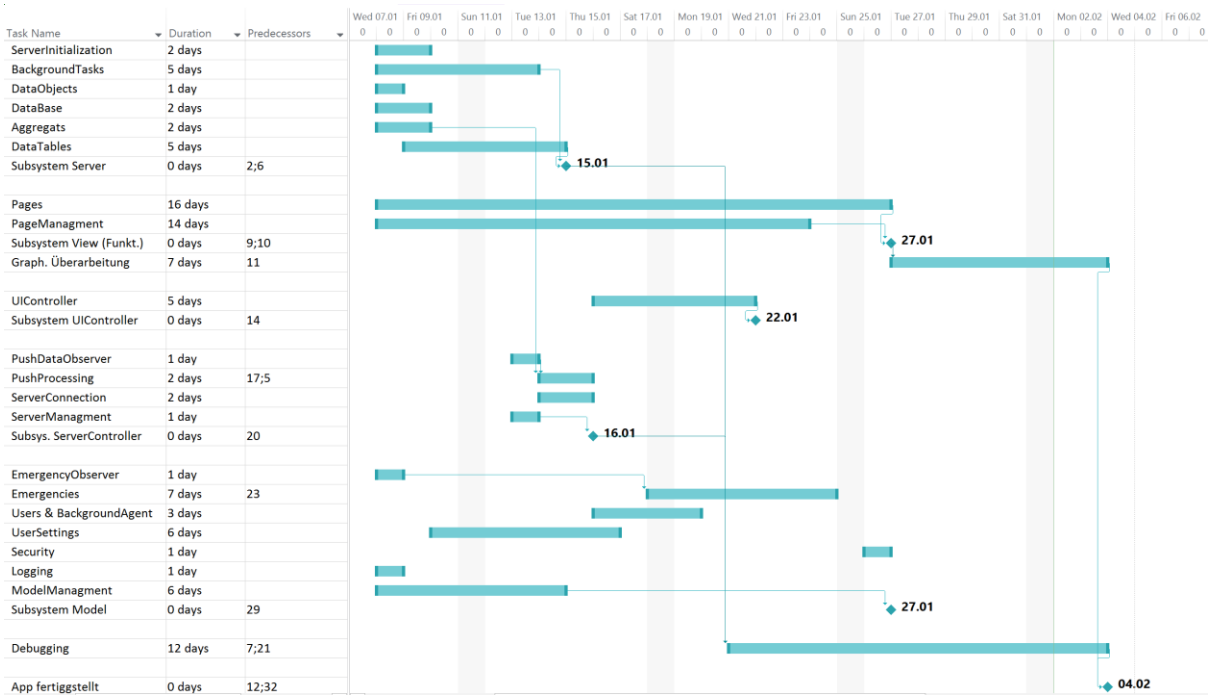


Abbildung 4.1 - Gantt-Diagramm des tatsächlichen Ablaufs

Wie geplant konzentrierten wir uns anfangs vor allem auf die Subsysteme View und Server. Gerade letzterer konnte ohne große Probleme und sogar frühzeitiger als geplant vollendet werden. Dazu gehörte aber nicht das Testen, da durch eine mangelhafte Dokumentation der Azure API einige Fragen offen blieben, sodass wir früher mit dem Debugging anfingen. Wir wollten möglichst bald die Stabilität des Servers verbessern, da die meisten Funktionen auf diesem aufbauen und zum Testen der anderen Subsysteme ein möglichst geringes Fehlerrisiko gewährleistet werden sollte.

Wie in den Änderungen beschrieben, stießen wir bei der Entwicklung des Views leider auf einige technische Probleme und nötige Änderungen, sodass sich die funktionale Implementierung weiter als geplant nach hinten hinauszog. Zudem war das Testen dieser Funktionalität stark von der Existenz des UIControllers abhängig, wobei wir uns allerdings dann eines Mock-Objektes bedienten, um die Entwicklung nicht zu sehr aufzuhalten.

Da wir die Phase damit begannen, Stummel für alle geplanten Klassen und Methoden zu erstellen, wurde eine unabhängige Entwicklung der Subsysteme voneinander und damit parallele Arbeit an fast allen Systemen möglich. Dies führte unter anderem auch dazu, dass der ServerController fast gleichzeitig mit dem Server eine Woche früher als gedacht vollständig implementiert werden konnte.

Durch die gegebene Unabhängigkeit der Grundimplementierung begannen wir die ersten Implementierungen im Model bereits zum Anfang der Phase. Die Arbeit an diesem Subsystem war zunächst nicht von hoher Bedeutung und wurde stark von den Änderungen in anderen Subsystemen beeinflusst, sodass sich einzelne Elemente hinauszögerten. Wir versuchten, die einzelnen Teile des Models soweit zu entwickeln, dass sie zum Testen des Views jeweils bereit standen, aber noch nicht zu sehr von möglichen Entwurfsänderungen beeinflussbar waren.

Mit Freude aber können wir bekunden, dass auch wenn die Implementierung nicht so linear wie angedacht verlief, wir fast alle Elemente früher als geplant beenden konnten und die App inklusive aller Muss-Kriterien seit dem 3. Februar als fertig implementiert angesehen werden kann.

4.2. Zukunftspläne

[KZ1010] – *Es gibt alternative Notfälle, eine Art Katastrophenkontrolle, die länger aktiv sind und auf einem Bereich statt einem Punkt definiert sind.*

Wir wollen den Grundgedanken der Applikation auf weitere Felder ausweiten. Dazu gehören auch andere Szenarien, in denen eine solche Funktionalität nützlich sein könnte.

[KZ1020] – *Ein Katastrophenfall kann nur von den Behörden erstellt, bearbeitet und beendet werden, um Missbrauch und Paniken zu vermeiden.*

Da Katastrophenfälle wesentlich stärker wahrgenommen, aber nur relativ selten sind, erscheint es uns sinnvoll, diese nur in die Verantwortung der Behörden zu übergeben.

[KZ1030] – *Innerhalb eines Katastrophenfalls können Aufgaben und besondere Punkte definiert werden, mit denen einzelne Benutzer interagieren (lesen, helfen, …) können.*

Zu einem Katastrophenfall gehört mehr als nur die Benachrichtigung der Benutzer. Dem Grundgedanken unserer Applikation folgend wollen wir es auch ermöglichen, zu helfen und sich zu organisieren, was innerhalb eines Gebietes mehrfach für verschiedene Fälle möglich sein soll (z.B. Essensausgabe, …).

[KZ1040] – *Falls der Alarmknopf falsch verwendet wird (z.B. durch zu kurzes Drücken), so wird der Benutzer durch eine kurz erscheinende Meldung darüber informiert.*

Die Applikation wird vermutlich (oder eher hoffentlich) selten verwendet, sodass man erwarten kann, dass Benutzer nicht zu vertraut damit sind. Neben einer möglichst intuitiven UI sollen kurze Hinweise die Bedienbarkeit unterstützen.

[KZ2010] – *Der Benutzer hat die Möglichkeit, sich die eingegebenen Spezifikationen eines Notfalls in die eingestellte Systemsprache übersetzen zu lassen.*

Mithilfe des Bing Translators wollen wir Benutzern die Möglichkeit geben, Notfälle automatisiert übersetzen zu lassen. Internationalisierung ist für uns sehr wichtig und neben den vielen bereits hinzugefügten Sprachen soll das die Verwendung zwischen Menschen verschiedener Sprachen erleichtern.

[KZ2020] – *Falls mehrere Notfälle eines Benutzers als Missbrauch gemeldet werden, werden neue Meldungen dieses Benutzers für einen längeren Zeitraum im Server ignoriert.*

Bisher sind wir uns noch nicht ganz einig über die Grenzen und Ausarbeitung der Missbrauchsfälle, halten es aber für wichtig, einen Benutzer ab einem gewissen Limit für die Zukunft einfach zu ignorieren.

[KZ5010] – *Benutzer können nicht nur ihre selbst hinzugefügten verknüpften Personen sehen, sondern auch alle Personen, die sie hinzugefügt haben.*

Eine Übersicht über die verknüpften Personen ist wichtig, um einen Überblick zu behalten und diese zu organisieren. Aus diesem Grund ist auch eine Liste der anderen Seite nützlich.

[KZ6010] – *Den Behörden wird die Möglichkeit gegeben, über ein Web-Interface einen beliebigen Kartenausschnitt anzuzeigen, auf dem sie alle enthaltenen Notfälle überwachen und beobachten können.*

Die Einbindung der Behörden in unsere Applikation kann in vielen Notfällen helfen, in denen zivile Hilfe nicht ausreicht. Auch Häufungen von gemeldeten Notfällen sowie Missbrauch der App kann so leichter überwacht werden. Dies baut direkt auf der Möglichkeit auf, die Behörden über die App zu informieren ([KW1020]).

5. Anhang

5.1. Abbildungsverzeichnis

5.2. Glossar

AKTIVE PERSON *Active User*	Person, die die Applikation momentan nicht im Ruhemodus hat.
ALARM *Alarm*	Funktion eines Notfalls, die Personen benachrichtigt und auf dem LiveFeed erscheint.
BENACHRICHTIGUNG *Notification*	Statusleistennachricht/Notification durch die Applikation.
BENUTZER *User*	Person, die die Applikation geöffnet hat und momentan verwendet.
CLIENT *Client*	Gerät des Benutzers, auf dem die Applikation installiert ist.
DETAILS *Details*	Alle Eigenschaften eines Notfalls, die ein Informierter einsehen kann. Gebündelt verfügbar in der Detailansicht eines Notfalls.
EXPERTE *Expert*	Person, die in ihrem Profil besondere Qualifikationen angegeben hat.
GRÖSSERE UMGEBUNG *Larger Area*	Alle räumlichen Punkte in einem festgelegten, größeren Radius.
HELFENDE *Responder*	Informierte, die in der Applikation ausgewählt haben, dass sie auf dem Weg sind.
INFORMIERTE *Informed User*	Melder oder benachrichtigte Personen, die den Notfall nicht ignoriert haben.
LIVEFEED *LiveFeed*	Karte/Liste mit allen aktuellen Notfällen
MELDER *Reporter*	Person, die den Notfall als Erste gemeldet hat.
NÄHERE UMGEBUNG *Nearby Area*	Alle räumlichen Punkte in einem festgelegten, kleineren Radius.
NOTFALL *Emergency*	Vorfall, bei dem Menschen, Tiere oder Eigentum ohne menschliches Eingreifen Schaden nehmen können.
PERSON *Person*	Mensch, der die Applikation auf seinem Smartphone installiert hat.
POP-UP *Pop-Up*	Dialog auf der grafischen Benutzeroberfläche, der sich (teilweise) über einen anderen Bildschirm legt und Informationen darstellt. Kann geschlossen werden.
PROFIL *Profile*	Auswahlbildschirm für gerätespezifische Informationen, die weitere Details über den Benutzer preisgeben.
RUHEMODUS *Do Not Disturb Mode*	Modus der Applikation, in der alle Benachrichtigungen und Standortdaten deaktiviert sind.
SPEZIFIKATION *Specification*	Die Angabe und Änderung der Details eines Notfalls durch den Melder.
VERKNÜPFTE PERSONEN *Linked Person*	Personen, die freiwillig mit dem Gerät des Benutzers verknüpft sind, um priorisiert behandelt zu werden.