HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

# Event-basierte Offline-Synchronisation

**Bachelorarbeit / Studienarbeit in Informatik**

HSR Hochschule für Technik Rapperswil (FHO)

Frühlingssemester 2019

**Autoren** Michelle Kunz und Pascal Bertschi
**Betreuer** Prof. Stefan Keller

**Datum** 12. Juni 2019

# Inhaltsverzeichnis

## Abstract

Die Umsetzung von Offline-Anwendungen stellt Herausforderungen dar. Offline-Fähigkeit und die damit entstehenden Problematiken zu lösen wird von vielen Frameworks und Datenbanken angepriesen. In Wirklichkeit müssen aber meist viele Kompromisse eingegangen werden.

Das Ziel dieser Arbeit ist es, bestehende Lösungen zu analysieren und ein neues Framework namens “Aion“ zu entwickeln. Dieses soll universell einsetzbar sein und die vielfältigsten Anforderungen erfüllen. Anhand eines Anwendungsbeispiels aus der realen Arbeitswelt, “Offliss“ genannt, wird die Funktionsweise veranschaulicht.

Die Aktionen der einzelnen Benutzer, sollen durch Synchronisationen zusammengeführt werden können. Diese sind von den Benutzern einseh- und nachvollziehbar.

Wenn zwei Benutzer einen Konflikt generieren, indem sie dieselben Attribute bearbeiten, löst Aion diesen automatisch auf. Wo nötig können die Benutzer eingreifen und die Entscheidungen korrigieren.

Wir haben uns überlegt, welche Probleme in einer Offline-Applikation auftreten können. Analysierten, ob diese mit den bestehenden Offline-Lösungen umsetzbar wären und dachten uns ein Anwendungsbeispiel in unserem Offliss dazu aus.

Im Anschluss folgte die agile Umsetzung des Framework Aion und des Anwendungsbeispiels Offliss. Nach 15 arbeitsintensiven Wochen resultierte ein funktionierendes Framework und ein anschaulicher Prototyp. Der Ansatz, das Konzept Event Sourcing in der Synchronisation zu verwenden, bewährte sich.

Offliss ist eine Issue-Verwaltung, die als Android-App umgesetzt wurde. Alle Benutzeraktionen sind offline verfügbar. Ausserdem banden wir erfolgreich externe Systeme an, wie die Autorisierung und einen E-Mailversand. Mit Offliss wurde demonstriert, dass Aion alle definierten Problemstellungen lösen kann.

## Abstract

The implementation of offline applications presents challenges. Offline capabilities and solving the resulting problems are promoted by many frameworks and databases. However, many compromises have to be made.

The goal of this work is to analyze existing solutions and develop a new framework called “Aion“. The solution should be universally applicable and meet the most diverse synchronization requirements. We would like to showcase our functionality in our demonstration application “Offliss“.

If two users generate a conflict by editing the same attributes, Aion resolves it automatically. Where necessary, the users can intervene and correct the decisions.

We have considered which problems can occur in an offline application. We analyzed whether these can be implemented with the existing offline solutions and came up with an application example in our Offliss.

This was followed by the agile implementation of the Aion framework and the Offliss application example. After 15 labour-intensive weeks a functioning framework and an impressive prototype resulted. The approach of using the Event Sourcing concept in synchronization proved to be successful.

Offliss is an issue management system that was implemented as an Android app. All user actions are available offline. In addition, we successfully integrated external systems such as authorization and e-mail dispatch. Offliss demonstrated that Aion can solve all defined problems.

# Management Summary

Wer befand sich nicht schon einmal in einem Funkloch und wurde in seiner Tätigkeit blockiert. Diese Arbeit befasst sich mit der Lösung dieser Problematik.

## Ausgangslage

Die Umsetzung von Offline-Anwendungen ist aufwendiger als man denkt. Ein Entwickler muss sich bei der Implementation mit neuen, bisher unbekannten Herausforderungen auseinandersetzen, die bei einer anderen Applikation nie beachtet werden mussten.

Viele Frameworks und Datenbanken werben mit Offline-Fähigkeit und versprechen Hilfe im Umgang mit den neu entstandenen Problematiken. Spätestens nach den ersten Entwicklungen werden einzelne Schwächen der Technologien bekannt. Es müssen viele Kompromisse eingegangen werden, um die Funktionalität, den Anforderungen entsprechend, umzusetzen. Sehr häufig leidet die Performanz, da das mobile Endgerät nicht die gleiche Leistung wie ein stationärer Rechner aufbringen kann. Des Weiteren kann auf diesen Geräten nicht die übliche Software zur Speicherung und Durchsuchung von grossen Datenmengen eingesetzt werden, weil man offline nicht mit dem Server interagieren kann und keine mobilen Versionen zur Verfügung stehen.

## Ziele

Das Ziel dieser Arbeit ist es, die Schwachstellen der bestehenden Lösungen aufzuzeigen und diese in einem neuen Framework namens “Aion“ zu lösen. Das Resultat soll das Entwickeln einer Offline-Anwendung sowohl vereinfachen als auch beschleunigen.

Anhand eines Anwendungsbeispiels aus der realen Arbeitswelt, “Offliss“ genannt, wird die Funktionsweise veranschaulicht. Die Lösung soll universell einsetzbar sein und die vielfältigsten Anforderungen erfüllen.

Die Aktionen der einzelnen Benutzer, wie das Erstellen, Ändern oder Löschen eines Datensatzes, sollen durch Synchronisationen zusammengeführt werden können. Dabei wird der Hauptteil der daraus resultierenden Konflikte im Hintergrund ohne Benutzereingriffe aufgelöst. Diese Synchronisationsentscheidungen sind von den Benutzern einseh- und nachvollziehbar. Wo nötig können sie eingreifen und die Entscheidungen korrigieren.

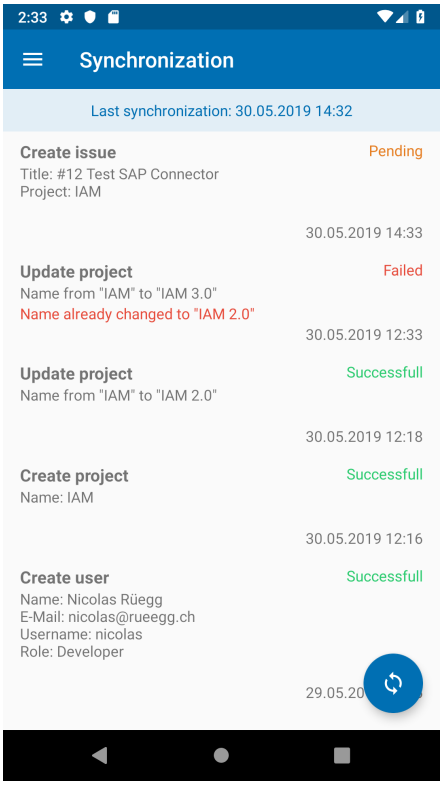


Abbildung 1: Synchroniation GUI

## Vorgehen und Technologie

Wir haben uns überlegt, welche Probleme in einer Offline-Applikation auftreten können. Anschliessend suchten wir existierende Offline-Lösungen und versuchten mit diesen unsere Problematiken zu bewältigen. Wie sich herausstellte, kann keine dieser die Herausforderungen vollständig lösen.

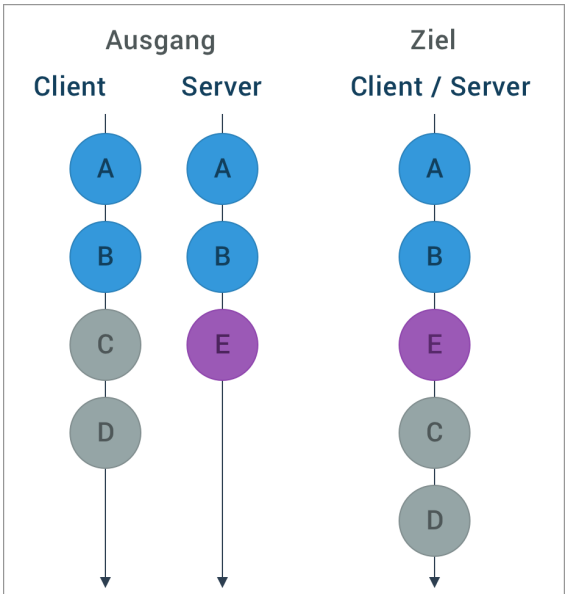

Abbildung 2: Timeline Visualisierung

Daraufhin arbeiteten wir Anforderungen anhand der Problemstellungen für ein Anwendungsbeispiel aus. Dieses diente dazu, unser Offline-Framework an einem praktischen Beispiel anzuwenden und aufzuzeigen, dass unser Lösungsansatz funktioniert.

Im Anschluss folgte die agile Umsetzung des Framework Aion und des Anwendungsbeispiels Offliss. Dafür verwendeten wir das Architektur-Pattern “Event Sourcing“, da sich Events einfach und effizient synchronisieren lassen. Wir definierten die Benutzeraktionen im Offliss und erstellten daraus die zu verarbeitenden “Events“.

Der Server bestimmt die Reihenfolge der Events in der “Timeline“ (Zeitachse). Die Endgeräte passen sich an diese an und fügen ihre Events als Letztes ein.

## Ergebnis

Nach 15 arbeitsintensiven Wochen resultierte ein funktionierendes Framework und ein anschaulicher Prototyp. Der Ansatz, lediglich die Events zu synchronisieren, bewährte sich, einfach und effizient Daten zwischen den Geräten auszutauschen. In der Applikation werden die Events zu sogenannten “Projections“ verarbeitet. Projections sind mit objektorientierten Datenbank-Views zu vergleichen. Event Sourcing ist gut test- und nachvollziehbar, sowohl für die Entwickler als auch für die Benutzer.

Die Schnittstellen von Aion sind projekt- und technologieunabhängig. Die Kommunikation mit den Umsystemen erfolgt über HTTPS. Durch Konfigurationen lässt es sich steuern, ohne dass in dessen Code Anpassungen vorgenommen werden müssen. Aion wurde in Java realisiert.

Offliss ist eine Issue-Verwaltung, die als Android-App umgesetzt wurde. Alle Benutzeraktionen, wie “Projekte hinzufügen“, “Issue bearbeiten“ etc., sind offline verfügbar. Ausserdem banden wir erfolgreich externe Systeme an, wie Autorisierung und den E-Mailversand, die wir mit Keycloak und Mailgun implementierten. Mit Offliss wurde demonstriert, dass Aion alle definierten Problemstellungen lösen kann.

## Ausblick

Wir denken, dass Aion vielen Entwickler, die sich mit Offline-Applikationen beschäftigen, eine grosse Hilfestellung bietet. Sie können sich auf die Funktionalität ihrer Anwendung konzentrieren und müssen sich nicht mit den technischen Details der Synchronisation oder ähnlichem auseinandersetzen.

Die Performanz könnte noch weiter ausgebaut werden, so dass Aion die Daten noch schneller synchronisieren kann. Zudem könnte man den benötigten Speicherplatz weiter reduzieren.

Teil I

# Technischer Bericht

# 1 Einführung

Die Offline-Synchronisation von Daten ist heutzutage trotz stark ausgebautem Mobilnetz eine wichtige Funktionalität, welche von vielen Anwendern in Auftrag gegeben oder ersehnt wird. Obwohl die Mobilnetzabdeckung stetig erweitert wird, werden im Alltag eines mobilen Anwenders immer wieder Unterbrüche vermerkt. Das Anwendererlebnis kann dadurch massgebend beeinträchtigt werden und die Bedienung verunmöglichen. Ausserdem ist der Energieverbrauch der Geräte, bei der Kommunikation mit dem Mobilnetz, hoch und resultiert in einem rapiden Akkuverbrauch.

Die Entwicklung von Offline-Anwendungen unterscheidet sich stark von Online-Anwendungen und ist mit mehr Aufwand und Planung verbunden. Bei der Entwicklung mit Offline-Datenbanksystemen muss der Entwickler umdenken, da viele Funktionalitäten von relationalen Datenbanken in diesen Fällen nicht möglich sind. Durch gute Planung können diese Limitationen frühzeitig erkannt und gelöst werden, jedoch können kleinste Anpassungen an der Anwendung zu Inkonsistenzen oder unzureichender Performanz führen.
Eine weitere Herausforderung stellt die Testbarkeit von Synchronisationsproblemen dar. Ein Fehler, welcher erst durch eine bestimmte Reihenfolge an Ereignissen eintritt, kann schwer zu entdecken sein.

Diese Arbeit ist in zwei Teile aufgeteilt. Im “Technischen Bericht“ gehen wir auf die Problemstellung, den aktuellen Stand der Technik bezüglich der Offline-Synchronisation sowie die Vision und die erarbeiteten Resultate ein. Der zweite Teil, die Software-Projektdokumentation befasst sich mit Artefakten aus dem Software-Engineering-Vorgehen.

## 1.1 Problemstellungen

In der Entwicklung von offline-fähigen System treten Probleme auf, die denen von verteilen Systemen gleichen. Bevor wir uns in die Analyse von existierenden Lösungen vertiefen, wollen wir die Herausforderungen in den nachstehenden Kapiteln kurz erläutern und anhand von Beispielen veranschaulichen.

### 1.1.1 Konfliktbehandlung

Die Konfliktbehandlung ist eines der grössten Probleme in Offline-Anwendungen, wenn mehrere Anwender die gleichen Daten bearbeiten können. Das Grundproblem liegt darin, dass wenn zwei Anwender offline den gleichen Datensatz verändern und diesen miteinander synchronisieren, ein Konflikt entsteht. Es gibt verschiedene Strategien diesen Konflikt aufzulösen:

**Letzte Synchronisation gewinnt** Der Anwender, welcher seine Änderungen dem Server zuletzt kommuniziert hat, gewinnt. Synchronisiert Anwender 1 Daten, welche Anwender 2 zuvor bereits verändert hat, werden die Änderungen von Anwender 1 behalten und die von Anwender 2 überschrieben. Diese Strategie ist einfach zu realisieren und performant, birgt jedoch Gefahren. Eine der Grössten stellt ein Anwender dar, der eine lange Zeit seine Daten nicht synchronisiert hat und somit alte Datenstände in der Datenbank besitzt. Sobald dieser seine Daten auf den Server lädt, werden alle Änderungen in der Zwischenzeit von diesem Anwender überschrieben.

**Erste Synchronisation gewinnt** Mit dieser Strategie darf der Anwender, welcher zuerst seine Änderungen dem Server kommuniziert, seine Daten behalten. Synchronisiert Anwender 1 Daten, welche Anwender 2 zuvor bereits verändert hat, gehen die Änderungen von Anwender 1 verloren und die von Anwender 2 werden festgeschrieben. Für diese Strategie muss auf allen Datensätzen eine Version geführt werden, zur Erkennung von Konflikten, auch bekannt unter Optimistic Concurrency Control.

**Zusammenführung** Änderungen von verschiedenen Anwendern werden bei dieser Strategie zusammengeführt. Dabei werden die einzelnen Datenfelder eines Datensatzes verglichen und in einer Version, mit den Änderungen von beiden Anwendern, gespeichert. Sollten die beiden Anwender unterschiedliche Datenfelder verändert haben, werden beide Änderungen gespeichert, so dass keine verloren geht. Wenn einige Datenfelder von beiden Anwendern bearbeiten wurden, muss entschieden werden welcher Anwender bevorzugt wird. Da eine Logik implementiert werden muss, welche Entscheidungen aufgrund der einzelnen Datensätze vornimmt und diese speichert, ist diese Strategie besonders aufwändig zu realisieren. Diese Logik ist spezifisch auf einen Anwendungsfall ausgerichtet und kann nicht für alle Arten von Daten gleich implementiert werden.

### 1.1.2 Berechtigung

Die Berechtigung von Daten kann für Anwendungen entscheidend sein. Es gibt Daten und Aktionen, die nur von Anwendern in einer bestimmten Rolle eingesehen oder getätigt werden können. In einer klassischen Server-Client-Architektur könnte eine Serveranwendung diese Berechtigungen prüfen und das Resultat dem Client mitteilen. Dies ist in einer Anwendung, ohne Verbindung zu einer Serverinstanz nicht möglich. Alle Aktionen, welche die Anwendung offline anbieten will, müssen auf dem Gerät implementiert und in einer lokalen Datenbank gespeichert werden. Manipulationen am Client können von Angreifern ausgenutzt und nicht vollständig verhindert werden. Deshalb sollten die Änderung der lokalen Datenbank nicht ohne Prüfung von der Serverseite übernommen werden.

### 1.1.3 Transaktionen

Einige Anwendungen haben Aktionen, welche Änderungen an verschiedenen zusammenhängenden Datensätzen durchführen und sollten deshalb in einer isolierten Transaktion ausgeführt werden. Diese Gruppierung von Änderungen in einer Transaktion müssen auch bei einer Synchronisation berücksichtigt werden, da ansonsten diese verletzt werden kann. In vielen dokumentenbasierten Datenbanken ist dieses Problem vorhanden und wird durch das Wählen einer geeigneten Datenstruktur umgangen. Eine zu wählen, die zu einer möglichst konfliktfreien Synchronisation führt sowie transaktionsunabhängig ist, kann schwierig oder gar unmöglich sein.

### 1.1.4 Einmaliges Vorkommen

Manche Daten dürfen im System nicht doppelt vorkommen, zum Beispiel E-Mailadressen von Anwendern. Um diese Anforderung einhalten zu können, muss in einer Offline-Anwendung sichergestellt werden, dass nach einer Synchronisation diese Regeln nicht verletzt sind. Werden diese verletzt, muss dagegen etwas unternommen werden.

### 1.1.5 Sequenzen

Um beispielsweise eine chronologische Bestellnummer in einer Offline-Anwendung zu führen, muss nach erfolgreicher Datensynchronisation die Laufnummer hochgezählt werden. Diese Laufnummer wird erst nach dem Synchronisationszeitpunkt ermittelt und muss bis zu diesem Zeitpunkt versteckt oder angenähert werden.

### 1.1.6 Unterbruchsfreie Bedienung

Eine weitere Schwierigkeit einer Offline-Anwendung stellt die unterbruchsfreie Bedienung der Anwendung dar. Die Datensynchronisation kann jederzeit im Hintergrund ausgelöst werden und darf den Anwender nicht in der Bedienung seiner Tätigkeit unterbrechen.

### 1.1.7 Teil-Synchronisation

Nicht alle Daten einer Datenbank müssen jederzeit auf den mobilen Geräten verfügbar sein. In vielen Fällen können auf diesen nur limitierte Datenmengen gespeichert werden. Ausserdem ist die Rechenleistung der Geräte für grosse Mengen nicht ausreichend. Durch eine Selektion der Daten, werden diese eingeschränkt und bewirken dadurch kürzere Synchronisationszeiten.

### 1.1.8 Externe Systeme

Über Schnittstellen sind Anwendungen oft mit anderen Systemen verbunden. Die Anwendung muss damit umgehen können, dass diese nicht jederzeit verfügbar sind. Alle Anwenderaktionen müssen deshalb auch ohne diese Systeme auskommen und die Kommunikation später nachholen können. Ein praktisches Beispiel wäre eine E-Mail zu versenden, sobald ein Datensatz bearbeitet wurde. Damit diese Aktion offline funktionieren kann, muss die E-Mail versendet werden können, sobald die Anwendung Verbindung zum E-Mailserver aufbauen konnte.

### 1.1.9 Daten-Migrationen

Anwendungen sollen stetig weiterentwickelt werden können. Daraus entstehen verschiedene Versionen. Diese können Datenstrukturen erweitern oder verändern, um neue Funktionalitäten anzubieten. Eine alte Version einer Anwendung kann unsynchronisierte Änderungen besitzen, welche verloren gehen können, wenn der Anwendungsserver die alte Version nicht mehr versteht und behandeln kann.

### 1.1.10 Anwendungsoberfläche

Anwendungen müssen Aktionen ausführen und die resultierenden Veränderungen darstellen. Diese können bis zum Synchronisationszeitpunkt nicht mit Sicherheit bestätigt werden. Im Falle eines Konflikts, welcher nicht aufgelöst werden kann, muss der Anwender interagieren können oder informiert werden, dass seine getätigte Aktion nicht erfolgreich war. Der Anwender sollte einen Überblick haben über ausstehende Aktionen und Änderungen, damit er erkennen kann, ob diese für andere Benutzer sichtbar sind oder er zuerst eine Datensynchronisation durchführen muss.

### 1.1.11 Performanz

Zur Performanz gibt es verschiedene Aspekte, die bei einer Offline-Anwendung wichtig sein können.

**Initiale Datensynchronisationszeit** Die erste Datensynchronisation ist die, bei welcher die meisten Daten vom Server auf den Geräten landen. Dies kann eine hohe Last auf der Serverseite erzeugen, falls dieser Fall nicht optimiert wurde. Der Anwender kann dadurch blockiert werden, weil er die Anwendung erst bedienen kann, wenn sich alle benötigten Daten auf seinem Gerät befinden.

**Synchronisationszeit** Eine Anwendung sollte möglichst oft synchronisieren, damit alle Anwender eine gleiche Datenbasis besitzen. Die Synchronisation sollte darum so kurz und mit so wenig Datenaustausch wie möglich funktionieren. Im Idealfall ist die Synchronisationszeit nicht von der Anzahl der Datensätze abhängig, sondern von den veränderten Datensätzen, welche synchronisiert werden.

**Serverauslastung** Auf dem Server muss die Auslastung verteilbar sein, damit die Synchronisationszeit bei ansteigender Nutzerzahl nicht zunimmt. Der Verarbeitungsdurchsatz synchronisierter Datensätze sollte bei steigender Anzahl Nutzer konstant bleiben. Ausserdem dürfen unter den verschiedenen Anwender einzelne nicht bevorzugt werden.

**Gerätauslastung** Während einer Synchronisation darf die Rechenleistung nur soweit genutzt werden, dass der Anwender davon nichts merkt. Dabei sollen die Kommunikation und der Rechenaufwand auf einem Minimum gehalten werden, damit der Akku des mobilen Geräts nicht beeinträchtigt wird.

## 1.2 Vorteile von Offline-Lösungen

**Unterbruchsfrei** Die Anwendung kann immer und jederzeit bedient werden, unabhängig von ihrer Verbindungsqualität.

**Schnellere Abfragen** Da sich die Daten lokal auf dem Gerät befinden, verkürzt sich die Latenz erheblich, weil keine direkte Kommunikation zum Server stattfinden muss.

**Datenmenge** Bei regelmässiger Nutzung wird die Datenmenge in der Kommunikation geringer, da nur die Veränderungen übermittelt werden.

**Dezentralisierung** Die mobilen Geräte funktionieren eigenständig und benötigen keinen stetig erreichbaren Server. Dadurch wird die Verfügbarkeit der Anwendung gesteigert, ohne die des Servers zu erhöhen.

## 1.3 Vision

Mit dieser Arbeit wollen wir aufzeigen, dass obwohl Offline-Anwendungen komplizierte Problematiken beinhalten, diese lösbar sind. Die Entwicklung dieser Anwendung muss vereinfacht werden. Der Fokus der Softwareentwickler soll bei den Kernproblemen der Business-Domäne liegen und nicht bei der Synchronisation der benötigten Daten.

Die erarbeitete Lösung soll sich nicht nur auf eine Art von Anwendung beschränken, sondern generisch in vielen Problembereichen eingesetzt werden können. Uns ist bewusst, dass unser Produkt nicht alle Bedürfnisse aller Anwender befriedigen kann. Deshalb legen wir grossen Wert auf vielfältige Konfigurationsmöglichkeiten und Erweiterbarkeit. Wir streben eine einfache, unkomplizierte Lösung an.

Unser Framework soll nicht wie andere Systeme über den Datenzustand (Projection), sondern über eine Änderungsliste synchronisiert werden. Dabei werden einzelne Ereignisse (Events) in einer nicht veränderbaren Liste gespeichert. Der aktuelle Zustand kann durch diese Reihe an Ereignissen errechnet und persistiert werden, dient aber nicht als Grundlage zur Replikation.

Das Resultat dieser Arbeit soll frei zugänglich sein. Wir vermuten, dass andere Entwickler und Unternehmen dieselben Problemstellungen wie wir haben. Darum möchten wir dieses Projekt unter einer Open Source Lizenz veröffentlichen. Bei Projekterfolg wäre die Open Source Community eine gute Bereicherung an Rückmeldungen und Unterstützung.

## 1.4 Ziele und Unterziele

**Problemstellungen** Wir möchten möglichst viele Problemstellungen in der Entwicklung von Offline-Anwendungen aufzeigen. Dies soll uns helfen, eine Denkweise für Offline-Anwendungen zu bekommen.

**Spezifikation eines Anwendungsbeispiels** Es soll eine Spezifikation eines Offline-Anwendungsbeispiels mit einem praktischen Nutzen in der realen Welt definiert werden. Zu diesem Anwendungsbeispiel werden die Anforderungen so gewählt, dass alle erarbeiteten Problemstellungen vorkommen. Die Anwendung soll in Java entwickelt und auf einem Android Gerät installiert werden können.

**Analyse** Anhand der erarbeiteten Problemstellungen, möchten wir die existierenden Lösungen analysieren. Das Fazit wird entscheiden, wie das Anwendungsbeispiel umgesetzt wird und ob eine Entwicklung eines eigenen Frameworks sinnvoll ist.

**Offline-Framework entwickeln** Stellt sich heraus, dass keine der existierenden Lösungen die definierten Probleme vollumfänglich lösen, werden wir unser eigenes Framework planen und entwickeln. Dieses wird nicht anwendungsspezifisch umgesetzt, sondern offen für Erweiterungen. Wie das Anwendungsbeispiel soll auch das Framework in Java entwickelt werden und aus einer Client- sowie Serverkomponente bestehen. Als Linux Container wird die Serverkomponente ausgeliefert. Die Entwicklung soll unter der MIT Open-Source Lizenz stattfinden.

**Dokumentation** Erfolgserlebnisse sowie Fehlschläge werden dokumentiert. Die Dokumentation soll anderen einen Einblick in die Probleme und Lösungen der Offline-Entwicklung geben.

# 2 Stand der Forschung und Technik

## 2.1 Aktuelle Situation

Es gibt heute bereits einige Datenbanken und Frameworks, die Offline-Funktionalitäten anbieten oder spezifisch für diesen Zweck entwickelt wurden. Die meisten Datenbanken basieren auf Dokumenten oder Schlüssel-Werten und besitzen nicht viele Möglichkeiten die Konfliktbehandlung zu steuern. Es werden viele Probleme dem Entwickler überlassen, welche in relationalen Datenbanken standardmässig im RDBMS gelöst werden. Dies ist nicht ohne Grund, denn viele der Herausforderungen sind schwieriger zu lösen in einem verteilten System als in einem RDBMS.

## 2.2 Forschung

Zur Offline-Synchronisation oder Datenreplikation gibt es bereits einige Forschungsergebnisse. Die wissenschaftlichen Artikel geben verschiedene Einblicke in die Thematik. Nachfolgend haben wir diejenigen aufgelistet, welche unserer Thematik am ähnlichsten sind.

Ein wissenschaftlicher Artikel [AST02] aus dem Jahre 2002, der sich mit Datensynchronisation von PDAs und mobilen Geräten beschäftigt, beschreibt fünf Synchronisationsprozesse. Diese arbeiten teilweise mit Statusfeldern auf den jeweiligen Datensätzen und nutzen diese, um einen minimalen Kommunikationsaufwand und eine geringe Auslastung zu erzielen. Der CPIsync stammt aus einem anderen Artikel [MTZ03]. Er wird als der skalierbarste beschrieben und wurde entwickelt, um zwei ähnliche Datenstände zu synchronisieren. Allerdings werden Konsistenzprobleme und Konfliktbehandlung darin nicht behandelt. Deshalb ist es nicht die optimalste Lösung.

Ein weiterer Artikel [MS12] aus dem Jahre 2012 beschreibt verschiedene Methoden zur Synchronisation von Daten. Die Beschreibungen sind jedoch sehr oberflächlich und lassen viele Fragen zur Implementation offen.

Im Jahre 2010 wurde ein Artikel [Ijt+10] bezüglich Offline-Synchronisation im Webbereich veröffentlicht. Dabei wird die E-Learning Plattform Moodle, mittels der HTML5 Offline-Funktion, um ein Quiz erweitert. Es handelt sich dabei aber mehr um ein Caching-System als um eine komplexe Datensynchronisation.

Der Artikel [BCN16] aus dem Jahr 2016 beschäftigt sich mit der Entwicklung einer mobilen Applikation für Bauern ohne Internetverbindung. Dabei entwickelten sie eine eigene Methodik, welche mit Revisionsnummern arbeitet, um Konflikte zu erkennen. Dieser Ansatz wird von CouchDB sowie Couchbase verwendet und unter Kapitel 2.5.1 und Kapitel 2.5.2 genauer erläutert.

## 2.3 Übersicht bekannter existierender Lösungen

In dem folgenden Kapitel stellen wir die bekanntesten bestehenden Lösungen vor. Diese sind unterteilt in Datenbank-, Framework- und Replikationslösungen.

### 2.3.1 Datenbanken

**CouchDB / PouchDB** CouchDB ist eine dokumentenbasierte Datenbank aus der Apache Foundation. Sie wird in Erlang entwickelt und ist seit 2005 unter der Apache-Lizenz frei zugänglich[1]. Die Datenbank gehört zu den AP Systemen aus dem CAP Theorem [GL12] und setzt auf Verfügbarkeit und Partitionstoleranz. CouchDB verspricht eine Multi-Master Lösung, die sich gut skalieren lässt, mit höchster Verfügbarkeit. Zusammen mit PouchDB[2], welches eine JavaScript Implementation von CouchDB ist, können die Daten auf Web- und mobilen Geräten gespeichert werden. Durch ein Revisionssystem werden Konflikte festgestellt und behandelt.

Abbildung 3: CouchDB Logo

**Couchbase** Couchbase ist sowohl eine dokumentenbasierte als auch eine Schlüssel-Wert-Datenbank. Das Projekt wurde damals im Jahr 2010 gestartet und ist unter der Apache-Lizenz frei zugänglich. Couchbase ist ein CP System, welches sich auf Konsistenz und Partitionstoleranz fokussiert.


Abbildung 4: Couchbase Logo

Couchbase Mobile[3] bietet mit ihrem Couchbase Sync Gateway einen sicheren Datenaustausch zwischen Couchbase Server und Couchbase Lite an. Couchbase Server ist eine Datenbank, die man auf einem oder mehreren Servern betreiben kann. Dagegen ist Couchbase Lite für mobile Geräte ausgelegt. Ausserdem führten sie mit ihrer neuen Couchbase Lite Version ein neues Websocket-basierendes Replikationsprotokoll ein, dass effizienter als der REST-basierte Vorgänger ist.

**Firebase / Firestore** Firebase ist eine dokumentenbasierte Echtzeit-Datenbank, die von Google übernommen wurde. Kurz darauf veröffentlichten sie Firestore, ein Nachfolger von Firebase. Die Datenbank ist für Echtzeitanwendungen ausgelegt und verspricht eine gute Skalierung. Firestore sowie Firebase sind Dienste, die nur via Google Cloud Platform bezogen und nicht auf einer eigenen Infrastruktur betrieben werden können. Beide Dienste sind proprietäre Systeme. Der Source Code ist nicht frei zugänglich.[4]


Abbildung 5: Firebase Logo

**MongoDB Mobile** MongoDB Mobile[5] ist eine dokumentenbasierte Datenbank, die auf Android und iOS läuft. Sie kann mit einer MongoDB Datenbank synchronisiert werden. [6] MongoDB ist ein CP System, das sich auf Konsistenz und Partitionstoleranz fokussiert. Sie wird unter der Server Side Public Lizenz entwickelt, die aktuell nicht von der Open Source Initiative anerkannt ist. Der Source Code ist aber frei zugänglich.

Abbildung 6: MongoDB Logo

[1] CouchDB: http://couchdb.apache.org/
[2] PouchDB: https://pouchdb.com
[3] Couchbase Mobile: https://blog.couchbase.com/couchbase-mobile-2-0/
[4] Firebase: https://firebase.google.com/
[5] MongoDB Mobile: https://www.mongodb.com/products/mobile?lang=de-de
[6] MongoDB Mobile Sync: https://docs.mongodb.com/stitch/mongodb/mobile-overview/#mobile-sync-beta

**Realm** Realm ist eine mobile Datenbank, welche auf Android, iOS und Windows läuft[7]. Sie wurde als eine Speicherlösung für mobile Geräte entwickelt und bietet seit 2017 eine Zweiwegsynchronisation an. Diese funktioniert jedoch nur zusammen mit der Real Plattform[8], die eine Cloud-Lösung von Realm ist. Der Source Code der Datenbankimplementation für die mobilen Plattformen ist unter der Apache-Lizenz frei zugänglich. Die Synchronisationsimplementation gehört zur Realm Plattform und ist nicht frei verfügbar.

Abbildung 7: Realm Logo

**RethinkDB / Horizon** RethinkDB wurde in einem Startup im Jahr 2009 gegründet und unter der Apache-Lizenz entwickelt.[9] RethinkDB ist eine dokumentenbasierte Datenbank mit einer speziellen Abfragesprache, welche mehr der einer relationalen als einer dokumentenbasierten Datenbank gleicht. Der Client namens Horizon wurde von der gleichen Firma entwickelt, der es ermöglicht die Datenbank in einer limitierten Form auf einem mobilen Gerät zu nutzen und Daten auszutauschen. Im Jahre 2016 wurde die Firma hinter der Datenbank aufgelöst und das Projekt einer Open Source Gemeinschaft übergeben. Seither nahm die Entwicklung von RethinkDB stark ab. Horizon wird weder weiterentwickelt noch gewartet.[10]

Abbildung 8: RethinkDB Logo

### 2.3.2 Frameworks

**turtleDB** turtleDB[11] ist ein frei zugängliches Open Source JavaScript Framework mit einer in-Browser-Datenbank, ausgelegt auf Offline-First Webapplikationen. Die Datenbank gehört zu den dokumentenbasierten Datenbanken und funktioniert mit IndexedDB.


Abbildung 9: turtleDB Logo

Seit Juli 2018 existiert das unter MIT Lizenzierte Framework. Zum Zeitpunkt unserer Arbeit ist die letzte veröffentlichte Version 1.0.0. Die letzten Arbeiten[12] wurden im September getätigt. Seither gab es keine neuen Entwicklungen mehr.

Für die Nutzung der Offline-Fähigkeiten, wird keine zusätzliche Anwendung benötigt. turtleDB übernimmt die Synchronisation und Konfliktbehandlung zwischen verschiedenen Benutzern selbst. Sie wirbt damit, grosse Datenmengen während des Offline-Betriebs im in-Browser-Speicher zu halten und macht Konflikte sicht- und nachvollziehbar.

### 2.3.3 Replikationslösungen

**SymmetricDS** SymmetricDS ist eine Datenbank- und Dateisynchronisationsapplikation, welche unter der GNU General Public License seit 2007 entwickelt wird.[13] Sie ist kompatibel mit einer grossen Auswahl an relationalen Datenbanken, inklusive SQLite, die auf mobilen Geräten eingesetzt werden kann. SymmetricDS erlaubt es, mehrere verschiedene Datenbanken mit einem Pull/Push Verfahren zu synchronisieren, welches individuell konfiguriert werden kann.


Abbildung 10: SymmetricDS Logo

[7] Realm Database: https://realm.io/products/realm-database
[8] Real Plattform: https://realm.io/products/realm-platform
[9] RethinkDB Wikipedia: https://de.wikipedia.org/wiki/RethinkDB
[10] RethinkDB Contributors: https://github.com/rethinkdb/rethinkdb/graphs/contributors
[11] turtleDB: https://turtle-db.github.io/about#introduction
[12] turtleDB GitHub: https://github.com/turtle-DB
[13] SymmetricDS Sourceforge: https://sourceforge.net/projects/symmetricds/

## 2.4 Engere Auswahl

Die oben erwähnten Lösungen werden mittels folgender Kriterien bewertet. Dabei werden die drei bestbewerteten Lösungen in der Detailanalyse analysiert.

Jedes Kriterium vergibt Punkte zwischen 1 und 3. Je höher die Punktzahl, desto besser.

**Open Source**

1. Der Source Code ist nicht frei zugänglich.
2. Der Source Code ist frei zugänglich.
3. Der Source Code ist unter einer Open Source Lizenz verfügbar.

**Dokumentation**

1. Es ist keine Dokumentation vorhanden oder nicht auffindbar.
2. Es ist eine Dokumentation vorhanden, jedoch nicht gut und klar beschrieben.
3. Es ist eine Dokumentation vorhanden und klar beschrieben wie die Offline-Funktionalität funktioniert.

**Aktives Projekt**

1. Das Projekt wird nicht weiterentwickelt.
2. Am Projekt wird nur das nötigste entwickelt.
3. Das Projekt ist in aktiver Entwicklung.

**Anwender**

1. Das Projekt hat einen weltweit tiefen Google Suchtrend, wenig Stackoverflowfragen und Anwender.
2. Das Projekt hat einen weltweit hohen Google Suchtrend, viele Stackoverflowfragen oder Anwender.
3. Das Projekt hat einen weltweit hohen Google Suchtrend, viele Stackoverflowfragen und Anwender.

**Resultat** Aufgrund der Bewertungskriterien haben wir alle Lösungen bewertet und kamen dabei auf das Resultat in Tabelle 11.

| | Open Source | Dokumentation | Aktives Projekt | Anwender | **Total** |
|---|---|---|---|---|---|
| **CouchDB / PouchDB** | 3 | 3 | 3 | 2 | **11** |
| **Couchbase** | 3 | 2 | 3 | 2 | **10** |
| Firebase / Firestore | 1 | 2 | 3 | 3 | 9 |
| MongoDB Mobile | 1 | 3 | 3 | 2 | 9 |
| Realm | 2 | 3 | 3 | 1 | 9 |
| RethinkDB / Horizon | 3 | 1 | 1 | 1 | 6 |
| turtleDB | 3 | 3 | 1 | 1 | 8 |
| **SymmetricDS** | 3 | 3 | 3 | 1 | **10** |

Abbildung 11: Bewertungsresultat der existierenden Offline-Lösungen

**Anmerkungen** CouchDB, Couchbase sowie SymmetricDS haben es, aufgrund ihrer Punktzahl, in die engere Auswahl geschafft.

Nachfolgend möchten wir nochmals kurz erklären, wieso die anderen Technologien nicht detailliert betrachtet werden.

MongoDB Mobile befindet sich noch in der Beta Phase und vieles ist nicht dokumentiert. Das Projekt könnte in der Zukunft vielversprechend aussehen, der aktuelle Stand ist jedoch nicht ideal. Der Source Code von MongoDB Mobile ist nicht einsehbar und funktioniert nur mit ihrem proprietären MongoDB Atlas Dienst.

Firebase sowie Firestore sind Datenbanken, ausgelegt auf Echtzeitsysteme und bieten limitierte Offline-Funktionalitäten an. Diese sind schlecht dokumentiert und gleichen einem Caching-System.

Die mobile Realm Datenbank verfügt über eine Open Source Lizenz und ist auf diversen Plattformen verfügbar, aber nicht die dazugehörige Real Plattform. In der Dokumentation konnten wir viele Informationen nicht nachlesen.

RethinkDB und Horizon sind nicht mehr unter aktiver Entwicklung und darum nicht ideal für neue Projekte.

turtleDB ist ein sehr kleines Projekt, von drei einzelnen Entwicklern. Der Source Code wurde innerhalb von zwei Monaten entwickelt. Doch seit August 2018 wurde nichts mehr verändert.

## 2.5 Detaillierte Analyse

Couchbase, CouchDB und SymmetricDS werden, anhand der zuvor aufgelisteten Probleme in Kapitel 1.1, nachstehend detaillierter analysiert.

### 2.5.1 Couchbase

**Konfliktbehandlung** Couchbase Lite verwendet Revisionen, um Konflikte zu detektieren. Die verwendete Technik nennt sich Multiversion Concurrency Control und wird auch von Git und Subversion eingesetzt.

Jedes Dokument schreibt beim Speichern bzw. Löschen die aktuelle und eindeutige Revision ID in ihr spezifisches Feld _rev. Eine neue Revision wird als untergeordneter Knoten an den Revisionsbaum gehängt. Wurde das Dokument in der Zwischenzeit von jemand anderem bearbeitet, ist die mitgegebene Revision ID nicht mehr aktuell. Dann existieren zwei untergeordnete Knoten, die nun im Konflikt stehen. Der "Gewinner" wird daraufhin weitergeführt, während der andere Knoten als gelöscht markiert wird.[14] [15]

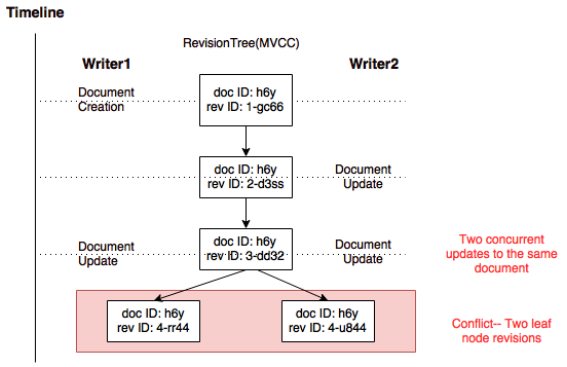


Abbildung 12: Revisionsbaum von Couchbase

In Couchbase Lite 1.x, wurden Konflikte mit Auflösungsregeln behandelt. Diese Regeln wurden jeweils in der Applikation definiert, um die Kontrolle über die zu verwendete Revision zu haben. Beliebte Methoden waren:[16]

- "Delete always win"
  Wird dasselbe Dokument von einem Benutzer gelöscht und gleichzeitig von einem anderen bearbeitet, gewinnt das Löschen.
- "N-way merge"
  Wurden Änderungen an verschiedenen Eigenschaften am gleichen Dokument vorgenommen, werden diese zusammengeführt.
- "Last update win"
  Falls Änderungen an der gleichen Eigenschaft getätigt wurden, gewinnt derjenige, welcher zuletzt die Änderung speichert.

Seit Couchbase Lite 2.0 werden Konflikte automatisch aufgelöst. Die Wahl der Revision ist deterministisch, so dass jedes Gerät dieselbe Revision auswählt. Das vereinfacht nicht nur die Handhabung, sondern spart auch Speicherplatz, weil konfliktäre Datensätze nicht länger in der Datenbank gespeichert werden.

[14] Couchbase Konflikterkennung: https://docs.couchbase.com/couchbase-lite/1.4/swift.html#revision
[15] Revisionsbaum: https://blog.couchbase.com/conflict-resolution-couchbase-mobile/
[16] Couchbase Konfliktbehandlung 1.x: https://docs.couchbase.com/sync-gateway/2.1/resolving-conflicts.html

Standardmässig gewinnt die letzte Änderung. Dieses Verhalten kann mit dem optionalen Argument "ConcurrencyControl" beim Speichern übersteuert werden. Der Konflikt wird dann manuell auf drei Arten aufgelöst: Änderungen an den Revisionen zusammenführen, Standardverhalten forcieren oder die aktuellen Änderungen verwerfen. In einem weiteren Modus können auch alle Konflikte verworfen werden.[17]

**Berechtigung** Die REST Schnittstellen von Sync Gateway bestehen aus einer öffentlichen API für die Replikation und einer Administrator-API zur Verwaltung von Benutzer und Rollen.

Mittels Rollen werden die Benutzer auf die Dokumente lesend oder schreibend berechtigt.[18]

Mit sogenannten Sync Functions, welche auf dem Server hinterlegt und in JavaScript implementiert werden, können Zugriffe auf einzelne Revisionen mit spezifischen Rollen oder Benutzer eingeschränkt werden. Diese Funktionen werden bei jedem Erstellen, Ändern und Löschen aufgerufen und können weitere Berechtigungen prüfen.[19]

**Transaktionen** Couchbase Server unterstützt Transaktionen auf einem Dokument. Transaktionen über mehrere Dokumente können nur mittels Two Phase Commit getätigt werden.[20]

Informationen bezüglich Transaktionen über mehrere Dokumente in Couchbase Lite oder Sync Gateway lassen sich keine finden. Wir gehen somit davon aus, dass diese nicht unterstützt werden.

**Einmaliges Vorkommen** Couchbase Lite unterstützt kein Unique Constraint.[21] Allerdings könnte man dies auf umständliche Weise erzwingen:[22]

- Einzigartiger Wert als ID missbrauchen
  Möchte man nur eine Eigenschaft einzigartig im System halten, kann man anstelle einer generierten ID, diese Eigenschaft als ID verwenden.
- Einzigartige Felder in andere Dokumente auslagern
  Die einzigartigen Werte werden von den restlichen Daten, in einem neuen Dokument, getrennt aufbewahrt.

**Sequenzen** Sequenzen werden durch "Counter"-Dokumente gehandhabt.[23] Ein "Counter" kann sowohl inkrementiert als auch dekrementiert und auf einen Initialwert gesetzt werden. Zusätzlich sind die Operationen atomar. [24]

**Unterbruchsfreie Bedienung** Über Parameter kann die Replikation der Daten von Sync Gateway asynchron gestartet werden. Dabei werden die Daten einer Instanz zur anderen übertragen. Die Benutzer sind in ihrer Tätigkeit nicht blockiert, da bei einem asynchronen Aufruf nicht gewartet wird, bis die Replikation beendet ist.[25]

[17] Couchbase Konfliktbehandlung 2.0: https://blog.couchbase.com/document-conflicts-couchbase-mobile/
[18] Couchbase Autorisierung: https://docs.couchbase.com/sync-gateway/2.1/authorizing-users.html
[19] https://docs.couchbase.com/sync-gateway/2.1/sync-function-api.html
[20] Couchbase Transaktion: https://blog.couchbase.com/multi-document-transactions-acid-couchbase-2/
[21] Couchbase Unique Constraint: https://forums.couchbase.com/t/cbl-2-unique-index/15659
[22] Couchbase Unique Constraint-Alternativen: https://kfalck.net/2009/06/29/enforcing-unique-usernames-on-couchdb/
[23] Sync Gateway Sequenzen: https://github.com/couchbase/sync_gateway/wiki/Sequence-Handling
[24] Couchbase Sequenzen: https://blog.couchbase.com/using-autonumber-in-couchbase/
[25] Couchbase asynchrone Replikation: https://docs.couchbase.com/sync-gateway/2.1/running-replications.html

**Teil-Synchronisation** Mit Filter Funktionen kann die Replikation auf ein Teilset der verfügbaren Dokumente beschränkt werden. Dadurch muss nicht die ganze Menge an Daten oder private Daten transferiert werden. Es gibt drei verschiedene Möglichkeiten die Replikation einzuschränken:[26]

- Gefilterte “push“ Replikation
  Bei der “push“ Replikation werden nur die eigenen Daten zu den anderen Instanzen repliziert. Dabei kann man einen Filter setzen, der nur einen Teil seiner Daten hochlädt.
- Gefilterte “pull“ Replikation
  Jedes Dokument ist mit einem Set von Kanalnamen versehen. Jeder Kanal synchronisiert nur die Daten, welche dem Benutzer zugänglich sind.

  Zusätzlich lässt sich auf die Kanalnamen filtern, um so die Daten noch mehr einschränken zu können.
- Nach Dokumenten ID gefiltert
  Es ist möglich in einer “pull“ Replikation eine Liste der zu ladenden Dokumenten IDs mit zu geben. Damit werden nur diese Dokumente synchronisiert.

**Externe Systeme** Sync Gateway kann auf zwei Arten in andere Systeme integriert werden. Eine davon ist eine sortierte Liste von Änderungen an Dokumenten. Diese kann über ein REST API oder ein Websocket geladen werden. Bei Websockets wird eine Verbindung zu der angegebenen URL geöffnet. Anschliessend muss der Client dem Server mitteilen, ab welchem Zeitpunkt er die Änderungen möchte. Danach werden diese, ab dem gewählten Zeitpunkt, mittels JSON-Nachrichten übertragen.

Die andere Variante ist der Austausch mittels Webhooks. Änderungen an den Dokumenten werden an die spezifizierte URL gesendet. Diese können je nach Bedarf gefiltert werden. [27]

**Daten-Migrationen** Da die Daten nicht strukturiert sind und auch keine Bedingungen für die einzelnen Attribute gesetzt werden können, ist es dem Entwickler überlassen die Migration durchzuführen. Couchbase bietet dabei keine Unterstützung an.

**Performanz**

**Initiale Datensynchronisationszeit** Alle Dokumente, ab einem definierten Kontrollpunkt, werden vom Sync Gateway an den Server bzw. an die mobilen Geräte synchronisiert. Beim ersten Laden gibt es noch keinen Kontrollpunkt, deshalb werden alle Dokumente geladen.

Mit dem “bulk“-Aufruf der API gibt es eine weitere Möglichkeit mehrere Dokumente auf einmal zu laden bzw. zu versenden. Dies ist eine effizientere Weise.[28]

**Synchronisationszeit** Je nach Replikationsart können die mobilen Geräte ihre Daten an den Server senden (push) oder von diesem empfangen (pull). Couchbase synchronisiert dabei nur die Änderungen an den Dokumenten. Dadurch kann die benötigte Zeit erheblich reduziert werden. Falls gewünscht, kann eine vollständige Synchronisation aller vorhandenen Dokumente ausgelöst werden.[29]

[26] Couchbase gefilterte Replikation: https://docs.couchbase.com/couchbase-lite/1.4/swift.html#replication
[27] Couchbase externe Systeme: https://docs.couchbase.com/sync-gateway/2.1/server-integration.html
[28] Couchbase Synchronisation: https://github.com/couchbase/couchbase-lite-ios/wiki/Replication
[29] Couchbase Synchronisation: https://blog.couchbase.com/data-replication-couchbase-mobile/

**Serverauslastung** Um die Serverauslastung zu reduzieren, bietet Couchbase eine horizontale und eine multidimensionale Skalierung an. Bei der Horizontalen, auch eindimensionale Skalierung genannt, kann man die Last auf mehrere Datenbankknoten aufteilen.

Hingegen, bei der multidimensionalen Skalierung kann man die Datenbank in Abfrage-, Index- und Datenknoten aufteilen. Dadurch lassen sich die Ressourcen nicht nur skalieren, sondern auch isolieren und optimal ausnutzen.[30]

**Geräteauslastung** Die Synchronisation und das Berechnen der Änderungen werden vom Replikator übernommen. Die Endsysteme erledigen lediglich das Senden und Empfangen der Dokumente. Dadurch ist die Auslastung der Geräte minimal.[31]

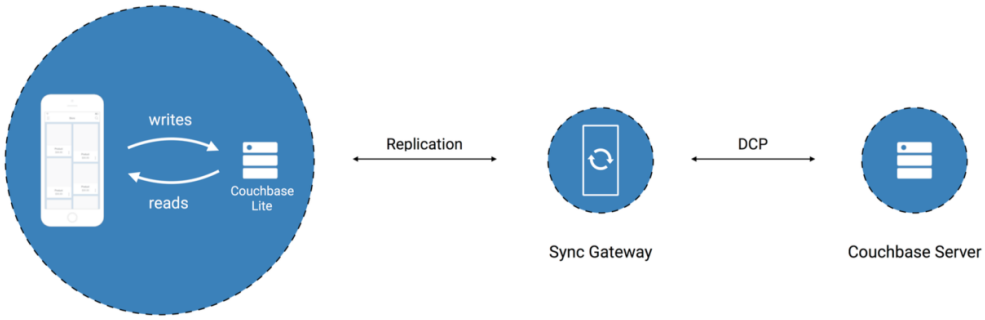


Abbildung 13: Sync Gateway in Couchbase

**Fazit** Couchbase erfüllt sehr viele unserer Kriterien für eine gute Offline-Synchronisation. Allerdings kann sie nicht mit Transaktionen über mehrere Dokumente umgehen. Dasselbe gilt für Unique Constraint und Daten-Migrationen. Viele NoSQL-Datenbanken haben diese Probleme, Couchbase ist dabei kein Einzelfall.

Als Datenbank für mobile Geräte eignet sich Couchbase sehr gut. Sie setzt effiziente Methoden für Replikationen ein und verteilt die Last auf mehrere Server.

Für die Entwicklung von Offline-Anwendungen müssen jedoch einige Einschränkungen in Kauf genommen werden.

[30] Couchbase Skalierung: https://www.couchbase.com/multi-dimensional-scalability-overview
[31] Sync Gateway: https://blog.couchbase.com/wp-content/uploads/2017/02/couchbase-mobile-stack_1920-1300x420.png

### 2.5.2 CouchDB

**Konfliktbehandlung** CouchDB verwendet für die Identifikation von Dokumenten einen Primärschlüssel sowie eine Revisionsnummer. Die Revisionsnummer wird bei einem Lesezugriff zurück- und bei allen folgenden Schreibzugriffen mitgegeben. Nach einem erfolgreichen Schreibvorgang wird eine neue Revisionsnummer erzeugt, welche auf die vorherige verlinkt.

Die Abbildung 14 veranschaulicht einen Konfliktfall. Zwei Aktoren lesen das gleiche Dokument und versuchen anschliessend nacheinander zu schreiben. Der zweite Schreibzugriff erhält dabei einen Konfliktfehler, den der Aktor behandeln muss. Dazu erhält er beide Versionen des Dokuments und kann sich für eine entscheiden.[32]

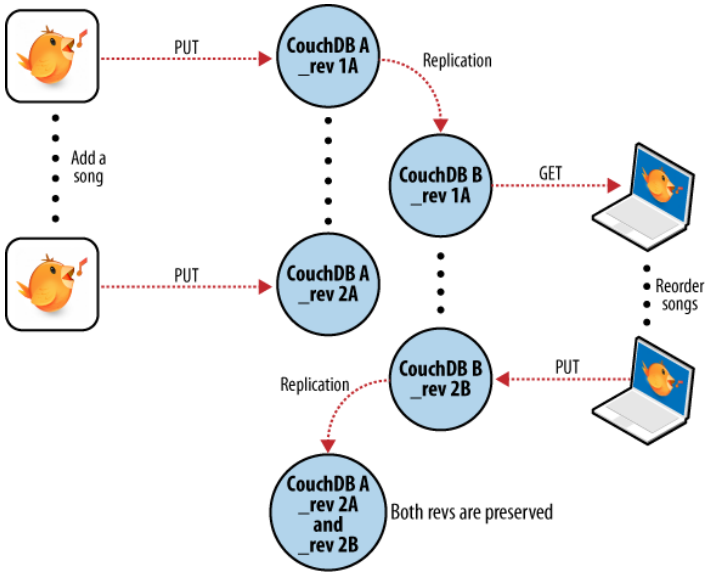


Abbildung 14: Konfliktfall von zwei parallelen Schreibzugriffen in CouchDB

Hierbei muss beachtet werden, dass bei einem Konfliktfall keine neue Version als aktuelle Version gewählt werden kann. Dadurch können Daten verloren gehen. Ein kombiniertes Dokument, welches beide Änderungen in einem neuen Dokument zusammenfasst, kann nur durch einen neuen Schreibzugriff realisiert werden. Dieser kann wiederum einen neuen Konflikt verursachen. Es ist darum wichtig, beim Design der Datenstruktur darauf zu achten, dass ein Dokument möglichst klein ist, um Konflikte und mögliche Verluste zu minimieren.

**Berechtigung** In CouchDB gibt es Datenbanken und Dokumente, wobei Datenbanken aus mehreren Dokumenten bestehen. Auf einer Datenbank können Lese- und Schreibberechtigungen vergeben werden. Bei Lesezugriffen gilt alles oder nichts. Für den Schreibzugriff kann mit Design-Dokumenten gearbeitet werden, welche Schreibzugriffe verhindern können. Die Design-Dokumente sind Dokumente mit einem "_design" Präfix im Primärschlüssel. Sie beinhalten JavaScript-Code, welcher die gewünschte Autorisierungslogik beinhaltet. Aufgrund der beschränkten Leseberechtigungsmöglichkeiten ist es nicht unüblich, dass jeder Benutzer seine eigene Datenbank besitzt.

[32] CouchDB Konfliktfall: https://docs.couchdb.org/en/stable/intro/consistency.html

Deshalb werden Daten, die nicht jeder lesen darf, in eigene Datenbanken gespeichert. Schreibzugriffe können flexibel konfiguriert oder programmiert werden, sind jedoch Aufwändig. Für jede Berechtigung muss ein entsprechendes Design-Dokument erstellt werden, dass diese, in Form von JavaScript-Code, realisiert. Diese Design-Dokumente können von allen Benutzern mit Lesezugriff gelesen und analysiert werden. Diese Funktionen müssen deshalb sicher implementiert sein. In der Vergangenheit gab es bereits mehrere Sicherheitslücken,[33] welche Angreifer trotz korrekt implementierter Design-Dokumente nutzen konnten, um sich Zugriff zu verschaffen. In einer Offline-Anwendung wird die CouchDB API für die Synchronisation der Daten benötigt und ist darum meistens direkt oder über einen Proxy angebunden.

**Transaktionen** In CouchDB gibt es keine Transaktionen, die mehrere Dokumente umfassen. Es können einzelne Dokumente geladen, gespeichert oder verändert werden, jedoch sind dies alles eigenständige Aktionen. Änderungen, welche zusammengehören, sollten sich deshalb im gleichen Dokument befinden. Ansonsten kann es, infolge von potenziellen Konflikten, zu Inkonsistenz kommen.

**Einmaliges Vorkommen** Aufgrund der Partition Toleranz gibt es in CouchDB keine Prüfung von Einzigartigkeit. Dies kann nicht sichergestellt werden, sollte ein Teil des Systems getrennt werden. Durch regelmässige Hintergrundprozesse, welche im Nachhinein die Daten korrigieren, könnte man ein einmaliges Vorkommen erzwingen. Doch selbst durch diese Prozesse, kann zu keinem Zeitpunkt davon ausgegangen werden, dass ein Wert einzigartig ist.

**Sequenzen** In CouchDB können keine Sequenzen geführt werden, da auch hier die Partition Toleranz dies verhindert. Nicht immer haben alle Datenbankinstanzen eine aktive Verbindung zueinander. Darum können sie sich auch nicht auf die nächste Sequenznummer einigen.

**Unterbruchsfreie Bedienung** Es gibt kein Locking. Darum kann selbst während einer laufenden Datensynchronisation auf der Datenbank gelesen und geschrieben werden. Der Anwender ist dadurch zu keinem Zeitpunkt blockiert. Werden während der Synchronisation neue Daten erzeugt, kommen diese in die nächste Datensynchronisation.

**Teil-Synchronisation** In CouchDB werden komplette Datenbanken synchronisiert. Falls nur eine Teilmenge synchronisiert werden soll, kann man dies mit Views realisieren. Diese erstellen mit MapReduce einen Index, welcher für die Datensynchronisation verwendet wird. Eine View wird in einem Design-Dokument definiert. Sie ist eine JavaScript Funktion, welche durch alle Dokumente iteriert und passende Dokumente in einem neuen Index zusammenfasst.

**Externe Systeme** Externe Systeme können nicht angebunden werden. Da es sich bei CouchDB um eine Datenbank handelt, gehört dies nicht zu ihrem Aufgabenbereich. Sie müssen von der Applikation selbst gehandhabt werden. CouchDB bietet keine Möglichkeiten Aktionen auszulösen, durch das Verändern von Daten, nach einer Synchronisation. Zum Beispiel muss mit einem Polling-Mechanismus gearbeitet werden, welcher regelmässig alle Datensätze mit den gewünschten Kriterien durchsucht und darauf die Externen Systeme anspricht.

**Daten-Migrationen** In CouchDB gibt es kein Schema und somit auch keine vorgesehenen Migrationen. Dies hat den Vorteil, dass Dokumente nicht zwingend migriert werden müssen, erfordert jedoch, dass die Anwendung die verschiedenen Versionen erkennt und versteht. Dies bedeuten mehr Aufwand und Eigenverantwortung für den Entwickler und muss von Anfang an bedacht werden.

[33] CouchDB CVE: https://docs.couchdb.org/en/latest/cve/index.html

#### Performanz

**Initiale Datensynchronisationszeit** Die initiale Datensynchronisation ist abhängig von der Anzahl Dokumenten in einer Datenbank. Je mehr, desto länger dauert diese. PouchDB ermöglicht es einen Daten Dump, welcher im Vorfeld erstellt wurde, einzuspielen. Damit kann eine Applikation mit Initialdaten ausgeliefert und die erste Datensynchronisation reduziert werden.

**Synchronisationszeit** Die Datensynchronisationszeit einer Datenbank steigt mit der Anzahl an Dokumenten. Der Synchronisationsmechanismus vergleicht nämlich alle Dokumente mit Revisionsnummern und Primärschlüssel mit dem jeweils anderen System. Steigt die Anzahl, müssen mehr Dokumente verglichen werden, wodurch die Zeit und auch das Datenvolumen einer Synchronisation ansteigt. Bei einer mobilen Anwendung, welche sich im mobilen Netz mit einer hohen Latenz befindet, ist dies besonders kritisch.

**Serverauslastung** CouchDB kann sehr gut skaliert werden, da aufgrund des Revisionssystems eine Multi-Master Replikation möglich ist. Die Last von Schreib- und Lesezugriffen kann auf mehreren verschiedenen Instanzen verteilt werden, welche untereinander regelmässig synchronisieren. Es gibt durch die Multi-Master Topologie auch keinen Single Point of Failure, weil jeder mit jedem synchronisieren kann.

**Geräteauslastung** Die Auslastung auf dem mobilen Gerät ist gering, denn der Server übernimmt bei der Datensynchronisation den grössten Teil der Arbeit. Der Kommunikationsaufwand steigt jedoch mit der Datenmenge an und erfordert viel Informationsaustausch.

**Fazit** CouchDB hat viele gute Eigenschaften und kann eine hervorragende Wahl als Datenbank sein. Die Multi-Master Synchronisation erlaubt eine hohe Verfügbarkeit, mit einer hohen Skalierbarkeit. Der Anwendungsbereich für CouchDB sehen wir aber mehr im Serverbereich von hochverfügbaren Systemen als auf mobilen Geräten. Für mobile Anwendungen benötigt die Synchronisation zu viel Datenvolumen, skaliert nicht für eine grosse Anzahl an Dokumenten und bietet wenig Basisfunktionalitäten an.

### 2.5.3 SymmetricDS

Bei SymmetricDS handelt es sich um eine Software, die für Replikationen von relationalen Datenbanken entwickelt wurde. Das Projekt hat dadurch viele Funktionalitäten im Bereich von Datenfilterung und Datentransformationen und ist ausgelegt auf die Verteilung von Datenbanken.[34]

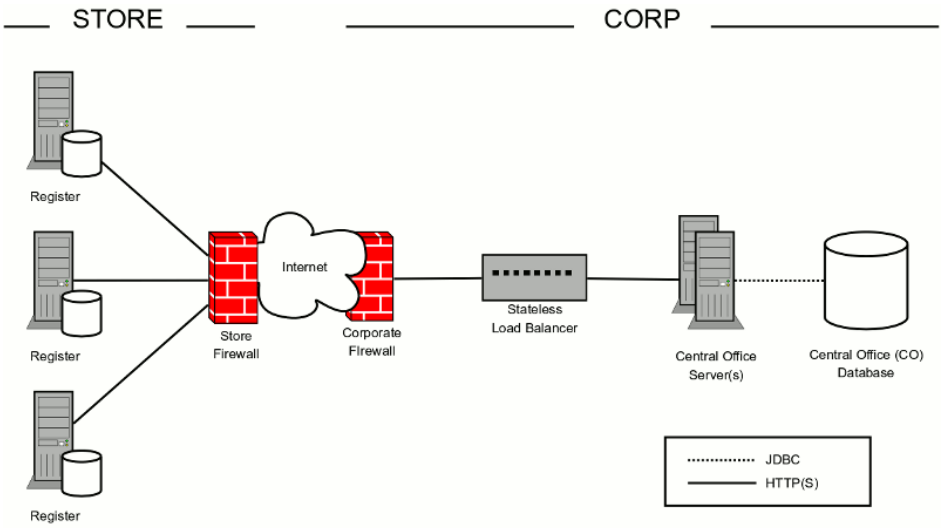


Abbildung 15: SymmetricDS Architektur

Mit der Hilfe von regelmässigen Jobs und Tabellentriggern werden die Synchronisationsprozesse angestossen, welche auf hinterlegten Konfigurationen beruhen. Die statische Konfiguration ist dabei ein Nachteil, weil diese konstant nachgeführt werden muss. Zum Beispiel benötigt jedes mobile Gerät einen Konfigurationseintrag, bevor es an einer Datensynchronisation teilnehmen kann. Obwohl das Projekt nicht für diesen Anwendungsfall ausgelegt ist, versuchen wir trotzdem alle verfügbaren Möglichkeiten anzuwenden, um einen Vergleich zu ermöglichen.

**Konfliktbehandlung** Die Konfliktbehandlung kann mit “Conflicts“ konfiguriert werden.[35] Dabei werden eine Konfliktfall-Erkennung und eine Auflösung definiert. Die Auswahlmöglichkeiten sind dabei sehr limitiert. Insgesamt stehen vier Strategien zur Auswahl. Bei allen muss entschieden werden, welche der in konfliktstehenden Versionen die richtige ist. Dabei können keine Daten zusammengeführt werden.

**Berechtigung** Es gibt in SymmetricDS kein Berechtigungskonzept. Es ist jedoch möglich mittels Synchronisationsregeln zu steuern, welche Daten eingesehen werden können. Eine Schreibberechtigung kann man nicht definieren. Der Synchronisationsprozess verwendet jeweils einen Datenbankbenutzer auf beiden Seiten der Synchronisation, der alle Rechte zur Schema- und Datenmanipulation besitzt.

**Transaktionen** Transaktionen können von den einzelnen Datenbanken auf Client und Server unterstützt werden. Jedoch werden diese bei der Synchronisation nicht berücksichtigt. Die Synchronisation funktioniert nur auf Zeilenbasis, es werden keine Zusammenhänge berücksichtigt.[35]

**Einmaliges Vorkommen** Die unterstützten relationalen Datenbanken verfügen alle über Unique Constraint. Im Konfliktfall gibt es jedoch Fehler in der Synchronisation. Diese können nicht ohne manuelle Interaktion aufgelöst werden und blockieren die Synchronisation, was wiederum zu inkonsistenten Daten führen kann.[36]

[34] SymmetricDS Architektur: https://www.symmetricds.org/doc/3.0/html-single/images/two-tier-arch.gif
[35] SymmetricDS User Guide: https://www.symmetricds.org/doc/3.9/html/user-guide.html
[36] SymmetricDS Diskussion: https://sourceforge.net/p/symmetricds/discussion/739235/thread/3a74da4f/

**Sequenzen** Wie beim einmaligen Vorkommen unterstützen die relationalen Datenbanken Sequenzen. Allerdings können diese nicht synchronisiert werden. Darum wird empfohlen mit GUID Datentypen zu arbeiten, anstelle von aufzählenden Nummern. Auch hier entstehen mögliche Konflikte bei einer Zweiwegsynchronisation.

**Unterbruchsfreie Bedienung** Die Synchronisation blockiert keine Schreib- oder Leseoperationen und somit auch den Benutzer nicht. Die Änderungen während einer Synchronisation werden vorgemerkt und beim nächsten Durchlauf aufgelöst.

**Teil-Synchronisation** SymmetricDS besitzt die Möglichkeit mittels “Router“-Konfigurationen die Datensynchronisation zu steuern. In dieser kann angegeben werden, welche Daten aus den verschiedenen Datenbanken abgeglichen werden sollen. Die Konfiguration kann auch Bedingungen beinhalten, um nur einen bestimmten Teil zu synchronisieren und bietet somit viel Flexibilität an.

**Externe Systeme** Es gibt eine Anbindung an einen E-Mail-Server, jedoch nur für Warnmeldungen von Synchronisationsproblemen. Wir konnten keine möglichen Schnittstellen aus der User Guide[35] entnehmen und vermuten deshalb, dass externe Systeme nicht angebunden werden können.

**Daten-Migrationen** SymmetricDS beschreibt, dass eine Schemamigration möglich ist.[37] Die zentrale Datenbank sollte zuerst das Schema anpassen. Danach können die anderen Datenbanken die Änderungen übernehmen. Es werden aber auch potenzielle Probleme beschrieben. Eines der schwerwiegendsten ist, dass ausstehende Änderungen bei Schemaänderungen verloren gehen können. Es wird empfohlen, diese erst durchzuführen, wenn keine offenen Änderungen vorhanden sind. Dies ist praktisch unmöglich zu realisieren, in einer Anwendung mit vielen mobilen Geräten. Ausserdem können die Geräte die Schemaänderungen erst durchführen, wenn auch die Applikationslogik dazu stimmt.

**Performanz**

**Initiale Datensynchronisationszeit** Für die erste Synchronisation gibt es einen Massenimport, welcher anstelle von einzelnen Datensätzen grössere Blöcke importiert. Dadurch kann die Auslastung verringert und Zeit eingespart werden.

**Synchronisationszeit** SymmetricDS kommuniziert während einer Synchronisation nur die vorgemerkten Änderungen und reduziert dadurch die Synchronisationszeit sowie die Netzwerkauslastung auf ein Minimum.

**Serverauslastung** Für eine gute Serverleistung gibt SymmetricDS Minimalanforderungen an.[35] Es wird von einer CPU sowie 2 GB Arbeitsspeicher für 500 MB Datentransfer pro Stunde bei 350 Clients ausgegangen. Umgerechnet sind das 140 KB pro Sekunde, was für die angegeben Ressourcen relativ wenig ist. Für unseren Anwendungsfall wäre die Serverauslastung problematisch, denn in vielen Anwendungen, kann es zu einer sehr grossen Anzahl von Geräten kommen, die alle synchronisiert werden müssen.

**Geräteauslastung** Es werden leider keine Angaben zur Auslastung der Client-Instanzen gemacht.

**Fazit** SymmetricDS dient dazu Daten verschiedener Datenbanken miteinander auszutauschen. Die Anwendung ist für den Einsatz in einer Serverumgebung ausgelegt und benötigt viele Ressourcen sowie eine ausführliche Konfiguration. Diese ist aufwändig und unflexibel, für mobile Anwendungen mit einer stark schwankenden Anzahl an Anwendern nicht ideal. Bei SymmetricDS sind alle Teilnehmer vertrauenswürdig. Deshalb besitzt sie kein zusätzliches Sicherheitskonzept, was für den Einsatz mit mobilen Anwendungen riskant ist.

[37] SymmetricDS Schema Changes: https://www.symmetricds.org/docs/how-to/sync-schema-ddl-changes

## 2.6 Schlussfazit

In diesem Kapitel wollen wir kurz zusammenfassen, wo sich die Probleme der evaluierten Lösungen befinden.

**Couchbase**

1. Änderungen, welche über mehrere Dokumente stattfinden, können nicht zusammengefasst werden, wodurch Inkonsistenzen entstehen.
2. Einmaliges Vorkommen kann nicht sichergestellt werden.
3. Versionierungen müssen selbst implementiert werden, womit die Rückwärtskompatibilität mit viel Aufwand verbunden ist.

**CouchDB**

1. Die Synchronisationszeit ist für grosse Datenmengen ungeeignet. Es müssen bei jeder Synchronisation alle Dokumente abgeglichen werden.
2. Durch Einschränkungen in der Vergabe von Leserechten müssen Dokumente in verschiedene Datenbanken aufgeteilt werden, was wiederum die Komplexität erhöht.
3. Externe Systeme können nicht integriert bzw. angestossen werden. Es muss eine eigene Lösung erarbeitet werden.
4. Sequenzen können nicht implementiert werden.
5. Dieselben Probleme die Couchbase hat, treten auch bei CouchDB auf.

**SymmetricDS**

1. Abhängig der Teilnehmerzahl, wird viel Konfigurationsaufwand erzeugt.
2. In der Konfliktlösung wird wenig Flexibilität geboten, weil das Set fest definiert ist.
3. Die Skalierung kann nur horizontal erfolgen und ist durch die Leistung der Datenbank limitiert.
4. Einmaliges Vorkommen sowie Sequenzen können verwendet werden. Allerdings muss mit manuellen Interaktionen im Konfliktfall gerechnet werden.
5. Wie bei CouchDB können keine externen Systeme angebunden werden.

CouchDB sowie Couchbase sind zwei gute Beispiele, welche aus technischer Sicht einwandfrei funktionieren. Die Herausforderung besteht darin, die Anforderungen der Anwendung und deren Benutzern in die technischen Rahmenbedingungen einzufügen. Es ist möglich mit allen analysierten Lösungen eine funktionierende Offline-Anwendung umzusetzen, jedoch würden einige Anforderungen viel Zeit und Entwicklung in Anspruch nehmen. Es fehlt in allen das Wissen der Applikationslogik, welche für Synchronisationsentscheidungen wichtig sein könnte.

Wir glauben, dass die Programmierung von Offline-Anwendungen einfacher sein könnte und möchten dies mit dieser Arbeit aufzeigen. Eines der grössten Probleme liegt darin, dass die Synchronisation nicht von der Datenbank kontrolliert werden soll und nicht in deren Verantwortung gehört. Es wird ein Synchronisations-Framework benötigt, welches mit der Anwendung zusammen agiert und die Konfliktentscheidungen flexibler abhandeln kann.

# 3 Umsetzungskonzept

In der ersten Phase haben wir, durch unsere bisherigen Erfahrungen mit Offline-Applikationen, Problemstellungen definiert, welche häufig während der Entwicklung auftreten. Aufgrund dieser haben wir die Forschungsergebnisse und existierenden Lösungen, die sich mit Offline-Funktionalitäten beschäftigen, analysiert und miteinander verglichen. Dabei stellten wir fest, dass noch keine Technologie unsere Probleme vollumfänglich lösen. Deshalb erarbeiteten wir ein eigenes Framework, mit dem Namen “Aion“.

Stream Processing oder auch Event Sourcing wird heutzutage für die Verarbeitung von grossen Datenmengen genutzt. In der Idee hinter diesem Paradigma sahen wir Potenzial in der Datensynchronisation. Darum setzten wir auch Event Sourcing in unserem Framework ein. Die Events werden durch Benutzeraktionen ausgelöst und in Aggregatoren verarbeitet. Diese werden in einer Reihe persistiert und stellen den Zustand des Systems dar.

Im Gegensatz zu einer CRUD-Applikation ist der Zustand flüchtig. Dieser nennt man auch Projection. Er wird aus den Events generiert. Ausserdem können Events nicht nachträglich bearbeitet werden, was oftmals ungewohnt erscheint.

Um die Funktionsweise des Frameworks zu demonstrieren, haben wir eine Anwendung zur Issue-Verwaltung (“Offliss“) entwickelt. Es handelte sich dabei um ein einfaches Beispiel, welches extra so gewählt wurde, um die Problemstellungen der Entwicklung mit Offline-Applikationen darzustellen.

# 4 Resultate

In diesem Kapitel werden die Resultate aus Aion unserem Offline-Synchronisations-Framework präsentiert. Dabei wird überprüft, ob die Problemstellungen aus Kapitel 1.1 erfüllt wurden.

## 4.1 Zielerreichung

**Konfliktbehandlung** Die Konfliktbehandlung mit Aion bietet grosse Flexibilität und deckt alle Varianten, von “First-Win“ über “Last-Win“ bis zu “Merge“ ab. Datensätze können gleichzeitig bearbeitet werden, sofern es sich um unterschiedliche Attribute handelt. Der Entwickler einer Applikation kann entscheiden, ob und wie ein Event die Projection verändert.

Konflikte werden dem Benutzer angezeigt und genau beschrieben. Dabei können dessen veränderte Daten durch andere Events überschrieben werden. Aus zeitlichen Gründen konnten wir nicht mehr umsetzen, dass der Anwender auf die überschriebenen Daten reagieren kann.

**Berechtigung** Die Schreibberechtigungen mit Aion können feingranular gewählt werden. Diese werden auf den einzelnen Events definiert, die der Anwender ausführt.

Allerdings können die Informationen in den Events, aufgrund der Synchronisation, von allen eingesehen werden. Die Leseberechtigung wird mittels Event-Gruppierung auf Timelines eingeschränkt. Dadurch können nur berechtigte Benutzer beispielsweise die User-spezifischen Events auf der User-Timeline ansehen.

**Transaktionen** Jeder Event wird in einer eigenen Transaktion ausgeführt. Diese werden automatisch vom System eröffnet und beendet. Somit kann bei fehlerhaften Verarbeitungen die Ausführung abgebrochen und der letzte Zustand wiederhergestellt werden. Ausnahmen bilden die Aktionen mit den externen Systemen, wie zum Beispiel das Versenden eines E-Mails. Deshalb sollte in der Implementation darauf geachtet werden, dass diese jeweils am Schluss der Verarbeitung ausgeführt werden.

**Einmaliges Vorkommen** In Aion ist es möglich Attribute so zu definieren, dass sie in den Projections einmalig vorkommen. Jedoch findet diese Prüfung erst nach der Synchronisation mit dem zentralen Server statt. Entsteht dabei ein Konflikt, weil der Wert bereits von einem anderen Datensatz in Verwendung ist, wird der getätigte Event rückgängig gemacht und der vorherige Zustand wiederhergestellt.

**Sequenzen** Numerische Sequenzen können definiert werden, so dass sie jedes Mal den nächst höheren Wert zurückliefern. Dieser wird bereits vor der Synchronisation angezeigt, kann sich danach aber ändern. Erst nach einer Synchronisation ist der Sequenzwert definitiv. Der Wert sollte bis dahin als temporär markiert oder in der Oberfläche nicht dargestellt werden, da ansonsten der Benutzer verwirrt wird.

**Unterbruchsfreie Bedienung** Alle Aktionen können lokal ausgeführt werden, ohne Interaktion mit einem Server. Die Synchronisation findet im Hintergrund statt und behindert den Benutzer nicht in seiner Tätigkeit. Allerdings läuft die Verarbeitung der Events im Main-Thread. Dadurch kann es zu kurzen Verzögerungen in der Oberfläche kommen, die man aber kaum wahrnehmen kann.

**Teil-Synchronisation** Durch die Gruppierung der Events auf verschiedene Timelines, werden nur die tatsächlich benötigten Daten synchronisiert. Im Offliss haben wir die eigenen Zeitbuchungen, welche vom eigenen Benutzer erfasst und bearbeitet werden können, von den restlichen Timelines separiert. Dadurch erhält er nur seine eigenen Zeitbuchungen auf sein Gerät.

**Externe Systeme** An Aion können externe Systeme angebunden werden, mit denen die synchronisierten Events interagieren können. Unser Anwendungsbeispiel haben wir mit dem Authentifikationssystem Keycloak und dem E-Mailserver Mailgun verbunden. Aus Sicherheits- sowie Stabilitätsgründen findet die Kommunikation nur auf dem Anwendungsserver statt.

**Daten-Migrationen** Aktuell können noch keine Daten-Migrationen durchgeführt werden. Aus Zeitgründen konnten wir diese Funktionalität nicht umsetzen. Die Idee wäre, dass der Entwickler eine neue Business-Logik implementieren könnte, die eine neue Struktur der Projections bewirkt und alle Events neu verarbeiten lassen.

**Anwendungsoberfläche** Alle Benutzeraktionen werden als Events mit Verarbeitungsstatus im Aion gespeichert. Diese werden dem Anwender angezeigt, so dass dieser einen Überblick über seine getätigten Aktionen hat. Der Status des Events ist abhängig von der Synchronisation und wird deshalb erst danach als erfolgreich oder fehlerhaft angezeigt. Vor der Synchronisation steht der Event in Bearbeitung.

**Performanz** Aus zeitlichen Gründen konnten wir die Performanz in Aion nicht messen. Trotzdem nehmen wir Stellung dazu und treffen Annahmen.

Dank Event Sourcing können wir die Events schnell abgleichen, weil wir nur die verketteten Listen vergleichen müssen.

Jedoch hat diese Methode auch Nachteile, denn zurzeit erlaubt Aion keine vertikale Skalierung. Dazu kommt, dass die Scheibgeschwindigkeit von SQLite auf die des Datenträgers beschränkt ist. Es besteht die Möglichkeit Aion vertikal zu skalieren, würde jedoch einiges an Implementationszeit benötigen.

Ausserdem ist die initiale Synchronisation, bei der allerersten Verwendung des Apps langsam, weil die Events über die Zeit zunehmen und sich gleichzeitig die Synchronisationszeit erhöht. Um dem entgegen zu wirken, müssten die Projections bereits auf dem Server vorgeneriert werden, so dass nur dieser Zustand anstelle der ganzen Eventliste übertragen werden muss. Zusätzlich könnte dadurch der Speicherplatz des mobilen Geräts verringert werden.

Die Performanz von Aion ist ausbaufähig, funktioniert jedoch für das Anwendungsbeispiel gut.

## 4.2 Vergleich zu bisherigen Lösungen

Unser Offline-Synchronisations Framework löst, im Gegensatz zu den anderen evaluierten Lösungen, fast alle Problemstellungen einwandfrei und erleichtert den Entwicklern das Implementieren der Offline-Fähigkeit. Auch wenn man Aion noch verbessern und erweitern könnte, wurden unsere Anforderungen, welche wir zu Beginn der Arbeit aufstellten, erfüllt.

Wir sehen Aion als ein Erfolg an und konnten das mit dem Offliss auch beweisen. Offline-Fähigkeit muss nicht kompliziert sein und man muss auch keine Kompromisse eingehen, es erfordert aber ein Umdenken zu einer Server-Client Anwendung.

## 4.3 Ausblick: Weiterentwicklung

Wir haben uns hauptsächlich auf den Funktionalitätsumfang von Aion fokussiert, deshalb konnten wir nicht alle Optimierungen umsetzen, die wir angedacht hatten. Für den produktiven Einsatz sind diese jedoch nicht nötig, wie man an dem Anwendungsbeispiel erkennen kann.

Teil II

# SW Projekt Dokumentation

# 5 Überblick

Unsere Arbeit haben wir in zwei Projekte aufgeteilt. Das erste Projekt ist die Entwicklung eines Offline-Frameworks. Das Zweite stellt ein Anwendungsbeispiel dar, welches das entwickelte Framework nutzt.

**Offline-Framework** Das Aion Framework vereinfacht die Programmierung und das Verwenden von Offline-Anwendungen. Es besteht aus einer mobilen und einer serverseitigen Komponente. Durch deren Zusammenspiel, werden die Synchronisationsprobleme aus Kapitel 1.1 gelöst.

**Anwendungsbeispiel** Das Anwendungsbeispiel ist für die fiktive Bank Labank entwickelt:
In der Bank werden viele Projekte agil durchgeführt. Dafür benötigen die Projektleiter ein Verwaltungssystem für ihre Arbeitspakete (Issue). Mitarbeiterzufriedenheit wird grossgeschrieben. Nebst flexiblen Arbeitszeiten können die Mitarbeiter auch von zu Hause aus arbeiten. Allerdings können sie aus Sicherheitsgründen nicht auf das interne Netz und Applikationen zugreifen.
Um trotzdem von zu Hause arbeiten zu können, nutzt Labank unteranderem die Offline-Issue-Verwaltung Offliss.

# 6 Anforderungsspezifikation

## 6.1 Use Cases (Offliss)

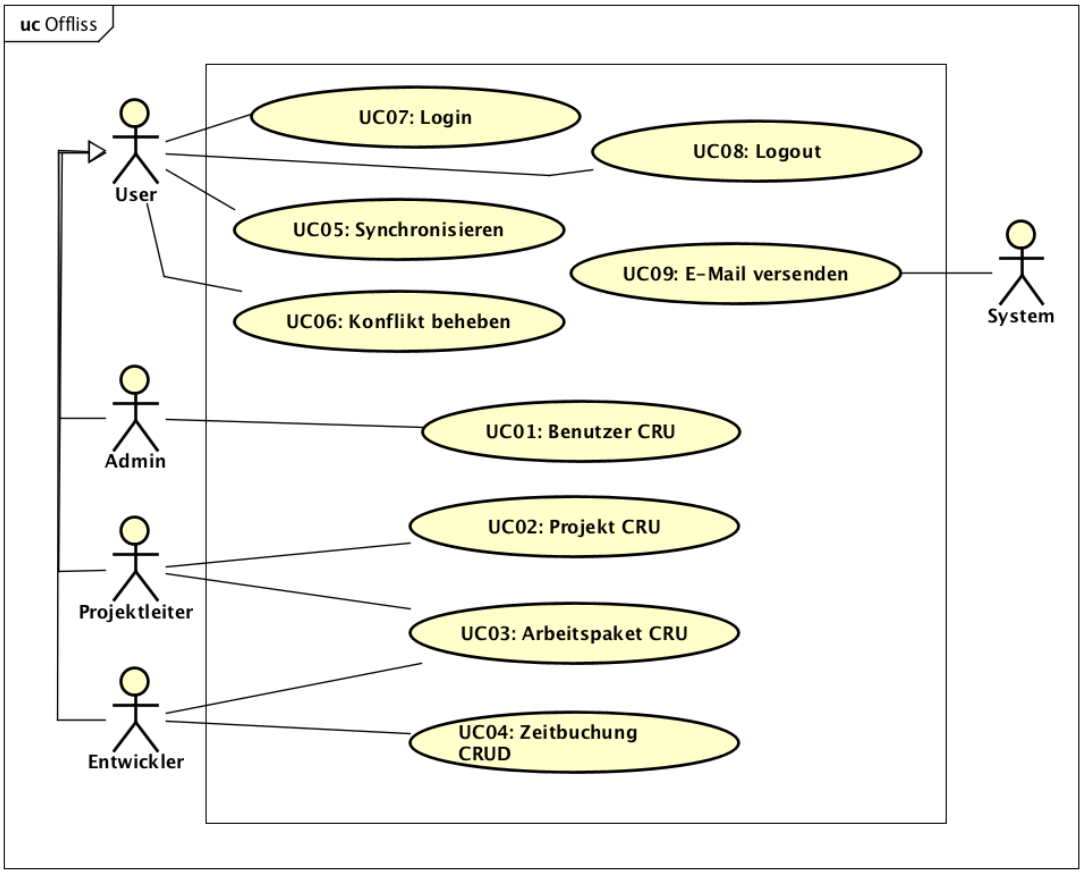


Abbildung 16: Offliss Use Case Diagramm

### 6.1.1 Aktoren

In der Tabelle 17 befinden sich alle Aktoren des Offliss. Deren Interessen werden kurz erläutert.

| Aktor | Beschreibung und Interesse |
|---|---|
| Admin | Der Administrator verwaltet die Benutzer für die Entwickler, Projektleiter und anderen Administratoren. |
| Projektleiter | Der Projektleiter verwaltet Projekte und ist verantwortlich für die Planung. Dabei verwaltet er die Aufgaben in seinem Projekt und verteilt diese an seine Entwickler. |
| Entwickler | Der Entwickler arbeitet an seinen, ihm zugeteilten Arbeitspaketen und kann auch solche erstellen. Er rapportiert für alle Arbeitspakete die aufgewendete Zeit und passt deren Status an. |
| System | Das System versendet in verschiedenen Fällen ein E-Mail an die Entwickler. |
| User | Der User existiert als solcher nicht. Die anderen Aktoren erben lediglich seine Fähigkeiten. Dadurch können sie sich am App an- und abmelden, ihre Daten synchronisieren und entstandene Konflikte beheben. |

Abbildung 17: Use-Case Aktoren

### 6.1.2 UC01: Benutzer CRU

**Aktoren**: Admin

Der Administrator verwaltet die Benutzer des Systems. Er erstellt und ändert deren Attribute. Dazu gehören der Name, die E-Mailadresse, die Rolle, der Benutzername und den Status eines Benutzers. Das Benutzerbild wird automatisch bei der Erfassung generiert, kann aber ausgetauscht werden. Er kann Benutzer nicht löschen. Dafür lassen sie sich über ihren Status inaktivieren. Passwörter werden aus dem System generiert. Der Administrator kann das Passwort weder einsehen noch verändern.

### 6.1.3 UC02: Projekt CRU

**Aktoren**: Projektleiter

Der Projektleiter verwaltet seine Projekte. Ein Projekt besteht aus einem Namen und einem Status. Der Status zeigt an, ob das Projekt noch aktiv ist.

### 6.1.4 UC03: Arbeitspaket CRU

**Aktoren**: Projektleiter, Entwickler

Ein Arbeitspaket besteht aus einem Titel, Beschreibung, Status und einer eindeutigen fortlaufenden Nummer. Es ist immer einem Projekt zugeordnet. Der Status ist entweder “Offen“ oder “Abgeschlossen“. Das Arbeitspaket kann einem, mehreren oder keinem Benutzer zugewiesen werden. Ausserdem kann es mehrere untergeordnete Arbeitspakete haben, jedoch immer nur ein übergeordnetes.

### 6.1.5 UC04: Zeitbuchung CRUD

**Aktoren**: Entwickler

Eine Zeitbuchung besteht aus einem Anfangs- und End-Zeitpunkt und ist immer einem Benutzer und einem Arbeitspaket zugeordnet. Der Startzeitpunkt muss sich jeweils vor dem End-Zeitpunkt befinden. Die Benutzer können nur ihre eigenen Zeitbuchungen verwalten.

### 6.1.6 UC05: Synchronisieren

**Aktoren**: Admin, Projektleiter, Entwickler

Verbindet sich der Benutzer, nach längerer Zeit, wieder mit dem System, müssen die ausstehenden Änderungen synchronisiert werden. Diese Aktion startet regelmässig im Hintergrund. Sie kann aber zusätzlich auf Verlangen ausgelöst werden.

### 6.1.7 UC06: Konflikt beheben

**Aktoren**: Admin, Projektleiter, Entwickler

Editieren zwei Benutzer die gleichen Daten und versuchen diese zu speichern, stehen diese Änderungen im Konflikt. Die Konflikte können teilweise automatisch aufgelöst werden. Sollte dies nicht möglich sein, muss der Benutzer die Entscheidung treffen.

#### 6.1.8 UC07: Login

**Aktoren**: Admin, Projektleiter, Entwickler

Um eine Aktion am System tätigen zu können, müssen die Benutzer sich vorher mit ihrem Benutzername und Passwort anmelden. Sind diese korrekt kann er seine Aktionen mit dem System ausführen.

#### 6.1.9 UC08: Logout

**Aktoren**: Admin, Projektleiter, Entwickler

Die Benutzer können sich nach getaner Arbeit wieder aus dem System ausloggen. Danach können keine weiteren Aktivitäten mit dem System durchgeführt werden.

#### 6.1.10 UC09: E-Mail versenden

**Aktoren**: System

Sobald ein Arbeitspaket einem Benutzer zugewiesen oder ein neuer Benutzer erstellt wird, versendet das System automatisch ein E-Mail an die betroffenen E-Mailadressen. Wenn der Status eines Arbeitspaket geändert wird, versendet das System allen zugewiesenen Benutzern eine Statusänderungsmeldung. Auch das Zurücksetzen des Passworts löst ein E-Mail aus.

### 6.2 Nicht funktionale Anforderungen (Aion und Offliss)

#### 6.2.1 Funktionalität

Aufgrund von definierbaren Rollen können die Zugriffe auf die Daten, je nach Berechtigung, eingeschränkt werden. Die dazugehörigen Aktionen, welche auf die Rollen prüfen, werden vom Entwickler der Anwendungsapplikation definiert. Dies steigert die Sicherheit des Frameworks.

Die Interoperabilität muss trotz nicht verfügbarer Anwendungsapplikation gewährleistet werden können. Die Informationen müssen, sobald die Applikation wieder verfügbar ist, ausgetauscht werden.

#### 6.2.2 Benutzbarkeit

Die Synchronisation erfolgt asynchron, so dass der Benutzer während seiner Tätigkeit nicht blockiert wird. Er kann während der Synchronisation uneingeschränkt weiter arbeiten.

2/3 aller Benutzer können intuitiv und ohne Anleitung mit Konflikten umgehen, die das Framework nicht automatisch aufgelöst hat. Die Benutzer werden über den Konflikt informiert.
Fehlermeldungen und ausstehende Offline-Aktivitäten werden verständlich dargestellt, so dass der Benutzer über den Zustand des Systems Bescheid weiss.

#### 6.2.3 Effizienz

Die Synchronisationszeit ist unabhängig von der Datenmenge. Deshalb werden nur die ausstehenden Änderungen, und nicht alle Daten, auf die Geräte bzw. den Server synchronisiert. Dadurch bleibt die Auslastung der Benutzergeräte auf einem Minimum.

### 6.2.4 Wartbarkeit

Mittels Kubernetes Helm Chart kann die Infrastruktur einfach installiert werden. Die fertige Installationskonfiguration vermindert das Fehlerrisiko, welches durch falsche Konfiguration oder Nutzung der Wartungsperson entstehen könnte.

Anwendungen, die unser Framework verwenden, sollen wart- und erweiterbar sein. Zum Beispiel können sie ihre Datenstruktur durch neue Tabellen und Spalten verändern und erweitern.

### 6.2.5 Übertragbarkeit

Die Serveranwendung soll als Docker-Image ausgeliefert werden, damit das Framework portabel auf allen Plattformen installiert werden kann. Dadurch ist der Installationsprozess standardisiert und es gibt weniger Aufwand, um das Framework auf allen Plattformen zur Verfügung zu stellen und zu testen.

# 7 Analyse

## 7.1 Programmiersprache (Offliss und Aion)

Unser Projekt besteht aus einem Android-Anwendungsbeispiel und einer Serverkomponente, die mit der Android-Anwendung kommuniziert.

Durch Android beschränkt sich unsere Auswahl der verfügbaren Programmiersprachen auf Java, Kotlin und JavaScript. Unsere Entscheidung fiel auf Java, da wir in dieser Sprache beide bereits Erfahrungen gesammelt haben und eine grosse Auswahl an Bibliotheken im Serverbereich vorhanden ist. Des Weiteren ist die Implementationslogik zwischen unserem Anwendungsbeispiel sowie unserer Serverkomponente aufgeteilt. Darum ist eine gemeinsame Programmiersprache von Vorteil.

## 7.2 Java Runtime (Offliss und Aion)

Die Java Runtime für Android ist vom Betriebssystem vorgegeben, aber im Serverteil mussten wir eine Entscheidung treffen. Wir haben uns dabei für die freie Implementation von OpenJDK[38] entschieden. Dahinter ist eine grosse Gemeinschaft von Entwickler und Firmen. Ausserdem beinhaltet es keine proprietären Code-Teile, wie das beim JDK von Oracle der Fall ist.

## 7.3 Datenbank (Offliss und Aion)

Unser Framework persistiert Daten. SQLite ist eine Open Source Datenbank, welche auf allen Plattformen läuft. Sie wird häufig eingesetzt im Zusammenhang mit Apps und ist allgemein weit verbreitet.[39] Der Funktionalitätsumfang und die Performanz sind mehr als ausreichend für unser Vorhaben. Darüber hinaus bringt eine relationale Datenbank viele Vorteile mit sich, die uns bei der Lösung der Problemstellungen aus Kapitel 1.1 helfen werden.

## 7.4 Docker (Aion)

Die Serveranwendung ist dank Docker portabel, auf verschiedenen Umgebungen einsetzbar. Durch die Verwendung von Docker ist die Reproduzierbarkeit gewährleistet und das Laufzeitverhalten auf allen Systemen gleich. Unser Framework ist besser isoliert und es müssen auf dem Betriebssystem des Servers keine Abhängigkeiten installiert werden, wie beispielsweise Java.

## 7.5 Kubernetes (Aion)

Für die Orchestrierung unserer Docker-Container haben wir uns für Kubernetes[40] entschieden. Es gibt nur wenige Alternativen zu Kubernetes, zum Beispiel Docker Swarm. Im Gegensatz zu Docker Swarm hat Kubernetes in den letzten Jahren stark an Popularität gewonnen und ist auf allen Cloud-Plattformen verfügbar. Docker Swarm wird aktuell nicht aktiv weiterentwickelt. Kubernetes ist eine gute Wahl für diese Arbeit, da es Open Source ist und uns nicht auf eine Plattform einschränkt (Vendor-Lock-In). Zudem haben wir bereits viel Erfahrung damit gesammelt.

**Helm** Für das Konfigurationsmanagement von Kubernetes bietet sich nur Helm an.[41] Helm verfügt über eine Vielzahl an bereits vorkonfigurierten Paketen und nimmt uns dadurch viel Arbeit ab.

[38] OpenJDK: https://openjdk.java.net/
[39] SQLite, die meist verwendete Datenbank: https://www.sqlite.org/mostdeployed.html
[40] Kubernetes: https://kubernetes.io/
[41] Helm: https://helm.sh/

## 7.6 Schnittstellen-Spezifikation (Aion)

Bei der Schnittstellen-Beschreibung des Servers haben wir Swagger/OpenAPI [42] gewählt. Die Spezifikation der Schnittstelle wird mit dem Swagger UI[43] zugänglicher gemacht und unterstützt uns bei der Entwicklung und auch in der Fehlersuche.

## 7.7 Dependency Injection (Aion)

In unserem Projekt werden wir mit Dependency Injection arbeiten, weil dadurch die Codequalität sowie die Testbarkeit gesteigert wird. Wir haben uns für die Verwendung von Guice[44] entschieden. Guice hat eine sehr gute Dokumentation, kann durch “Extensions“ erweitert werden und besitzt viele Funktionalitäten.
Eine Alternative wäre Dagger, welcher nur wenig Funktionalitäten bietet und durch die Vorkonfiguration nicht so flexibel ist wie Guice mit Reflexion.

## 7.8 HTTP Client (Aion)

In Aion wird eine HTTP Client Bibliothek zur Kommunikation zwischen Server und dem mobilen Gerät benötigt. Die Wahl fiel auf okhttp, weil diese auch der offizielle Java Swagger-Client verwendet. Diese funktioniert auf einem Android Gerät und einer Java Applikation (Offliss Server).

## 7.9 Web Framework (Aion)

Für die Evaluation des Java Web Frameworks haben wir folgende Kriterien der bekanntesten und beliebtesten Frameworks untersucht.

**Komplexität**

1. Das Framework ist nicht einsteigerfreundlich und ist mit einer hohen Lernkurve verbunden.
2. Das Framework ist nicht einsteigerfreundlich und ist mit einer tiefen Lernkurve verbunden.
3. Das Framework ist einsteigerfreundlich und ist mit einer tiefen Lernkurve verbunden.

**Dokumentation**

1. Es ist keine Dokumentation vorhanden oder nicht auffindbar.
2. Es ist eine Dokumentation vorhanden. Es ist jedoch nicht alles klar beschrieben.
3. Es ist eine Dokumentation vorhanden und es ist klar beschrieben sowie einsteigerfreundlich.

**Testbarkeit**

1. Das Framework hat keine Testmöglichkeiten.
2. Das Framework hat Testmöglichkeiten.
3. Das Framework hat Testmöglichkeiten, eine JUnit Integration sowie eine kurze Testdurchlaufzeit.

[42] Open API: https://swagger.io/docs/specification/about/
[43] Swagger UI: https://swagger.io/tools/swagger-ui/
[44] Google Guice: https://github.com/google/guice

**Modularität**

1. Das Framework besteht aus einer Bibliothek mit vielen Abhängigkeiten.
2. Das Framework besteht aus mehreren Bibliotheken, die alle installiert werden müssen.
3. Das Framework besteht aus mehreren Bibliotheken, welche einzeln installiert werden können, je nach Bedürfnis.

**Sicherheit**

1. Das Framework hat keine integrierte Authentisierungs- / Autorisierungsmöglichkeiten.
2. Das Framework hat wenige Authentisierungs- / Autorisierungsmöglichkeiten.
3. Das Framework hat viele Authentisierungs- / Autorisierungsmöglichkeiten.

**Eigenständigkeit**

1. Das Framework benötigt einen externen Applikationsserver, wie zum Beispiel Tomcat / Glassfish.
3. Das Framework benötigt keinen externen Applikationsserver.

**Resultat** Anhand von diversen Quellen und den jeweiligen Dokumentationen kamen wir auf die folgende Auswertung in Tabelle 18. [45] [46] [47] [48] [49] [50] [51] [52]

| | Komplexität | Dokumentation | Testbarkeit | Modularität | Sicherheit | Eigenständigkeit | **Total** |
|---|---|---|---|---|---|---|---|
| Grails | 1 | 3 | 3 | 3 | 3 | 1 | 14 |
| GWT | 2 | 2 | 2 | 3 | 2 | 1 | 12 |
| Play | 2 | 3 | 3 | 2 | 3 | 3 | 16 |
| Spring | 1 | 3 | 3 | 1 | 3 | 3 | 14 |
| **Vert.x** | **2** | **3** | **3** | **3** | **3** | **3** | **17** |

Abbildung 18: Web Framework Evaluation

Wir haben uns für die Implementation von Vert.x entschieden. Vert.x gibt es bereits seit 2011 und gehört zur Eclipse Foundation. Es gibt sehr viele gute Dokumentationen und eine grosse Anwendergemeinschaft. Einer der Vorzüge von Vert.x ist der modulare Aufbau des Frameworks, welches uns erlaubt ausschliesslich die benötigten Module zu verwenden. Dadurch nimmt unsere Applikation nicht an Grösse zu und wir haben minimale Abhängigkeiten zu anderen Bibliotheken. Eine weitere gute Eigenschaft ist, dass der integrierte Webserver nicht wie die meisten anderen Frameworks mit Java Annotationen aufgebaut wird. Vert.x benötigt zudem keinen zusätzlichen Applikationsserver und kann eigenständig agieren. Zu guter Letzt, ist der Source-Code gut testbar, unteranderem durch die JUnit Integration.

[45] Best Java Webframeworks: https://javapipe.com/blog/best-java-web-frameworks
[46] Best Java Webframeworks: https://www.slant.co/topics/40/~best-java-web-frameworks
[47] GWT Dev Guide: http://www.gwtproject.org/doc/latest/DevGuide.html
[48] GWT OAuth2 Sample: https://github.com/reinert/gwt-oauth2/blob/master/samples/multi/com/google/api/gwt/oauth2/samples/multi/client/OAuth2SampleEntryPoint.java
[49] Grails Docs: http://docs.grails.org/3.1.3/
[50] Stackoverflow Question GWT: https://stackoverflow.com/questions/4030969/how-to-call-restful-services-from-gwt
[51] Grails Token Security: https://www.oodlestechnologies.com/blogs/Securing-REST-API-using-basic-Token-based-Approach-i
[52] Spring OAuth JWT: https://www.baeldung.com/spring-security-oauth-jwt

## 7.10 GUI (Offliss)

Nachfolgend werden einige GUI-Entwürfe für das Offliss App vorgestellt. Diese wurden mit Figma[53] skizziert. Ziel der Entwürfe ist es, das grundlegende Design der App zu definieren und einen Überblick über die verschiedenen Aktivitäten zu erlangen.

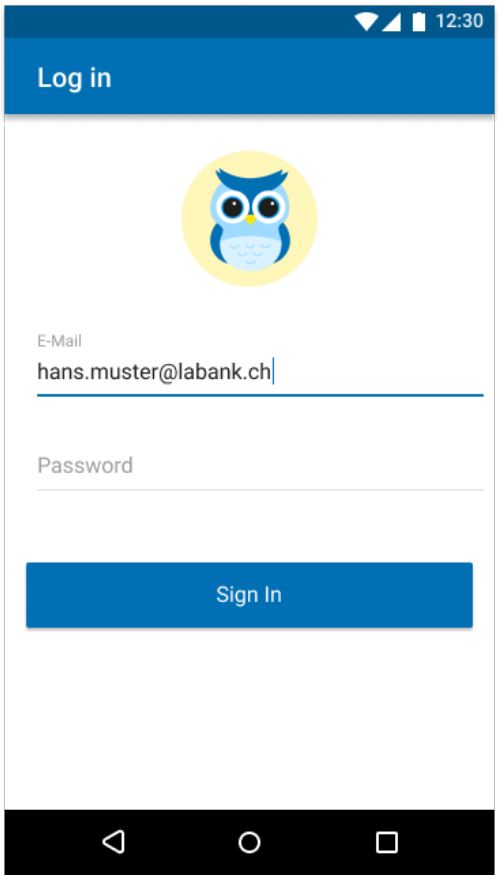


Abbildung 19: Offliss Entwurf Login

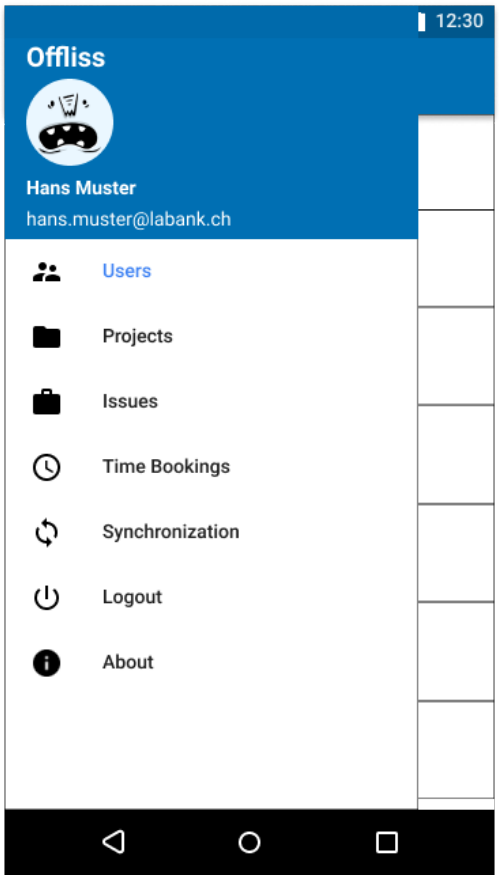


Abbildung 20: Offliss Entwurf Navigation

**Login** Jeder Benutzer muss sich vorgängig bei der App mit seiner E-Mailadresse und dem Passwort anmelden. Ohne Anmeldung können keine Aktionen im Offliss ausgeführt werden.
Das Login wird vom Administrator erstellt. Es gibt keine Möglichkeit sich selbst zu registrieren.

**Navigation** In der Navigation, welche über das “Hamburger-Menü“ ausgeklappt werden kann, sieht der Benutzer nur die Verweise, zu denen er berechtigt ist. Die aufgelisteten Punkte werden nachfolgend erläutert.

[53] Figma: https://www.figma.com

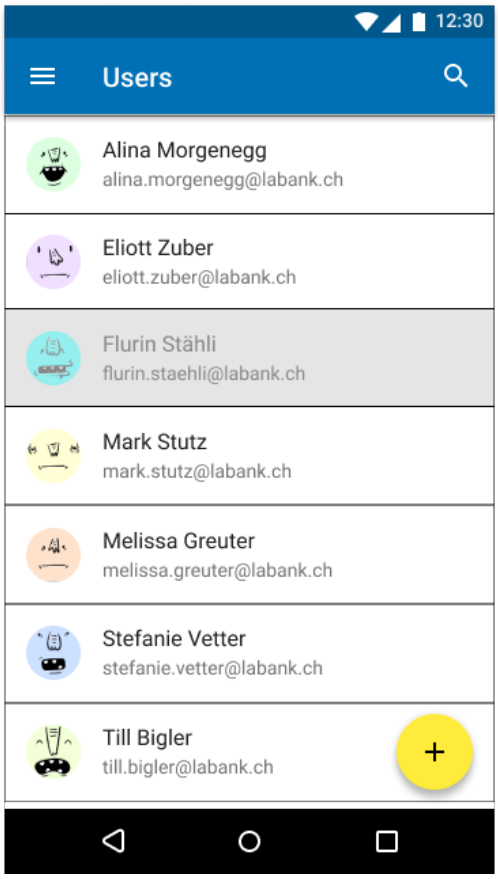


Abbildung 21: Offliss Entwurf Benutzer-Liste

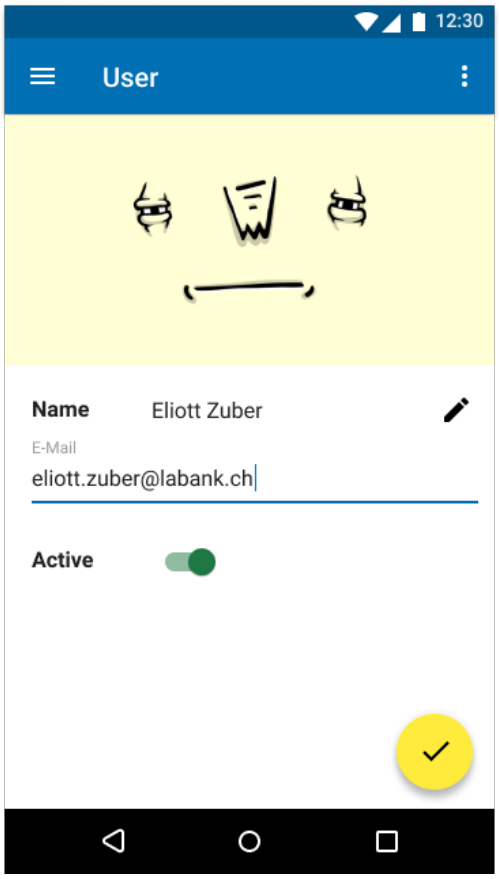


Abbildung 22: Offliss Entwurf Benutzer-Details

**Benutzer** Hinter “Users“ kann der Administrator andere Benutzer erfassen, bearbeiten und inaktivieren. Die Benutzer sind alphabetisch nach Namen sortiert. Benutzer, welche die Firma verlassen haben, bzw. nicht mehr in Projekten arbeiten, werden ausgegraut dargestellt. Diese Ansicht sieht nur der Administrator.
Durch das Anwählen eines Benutzers gelangt man auf die Benutzer-Detail-Ansicht.

**Benutzer Details** In der Detail-Ansicht werden die Daten des Benutzers detailliert angezeigt. Diese können ausserdem direkt in derselben Ansicht abgeändert werden. Die E-Mail-Adresse wird zur Anmeldung und zum Versenden von E-Mails verwendet, deshalb muss diese einzigartig sein. Ein Benutzer kann aus Gründen der Nachvollziehbarkeit nicht aus dem System entfernt werden, jedoch inaktiviert. Das Passwort kann weder gesetzt noch geändert werden. Es wird jeweils bei der Erstellung eines Benutzers generiert und an diesen versendet. Im Menü kann der Administrator ein neues Passwort generieren lassen, falls der Benutzer seines vergessen hat.
Der Avatar wird vom System automatisch generiert, kann aber durch einen neuen Avatar ersetzt werden.

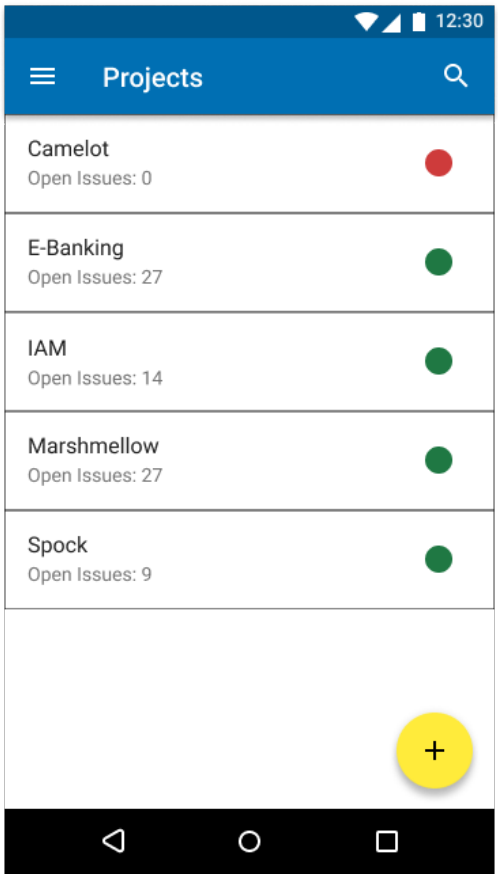


Abbildung 23: Offliss Entwurf Projekt-Liste

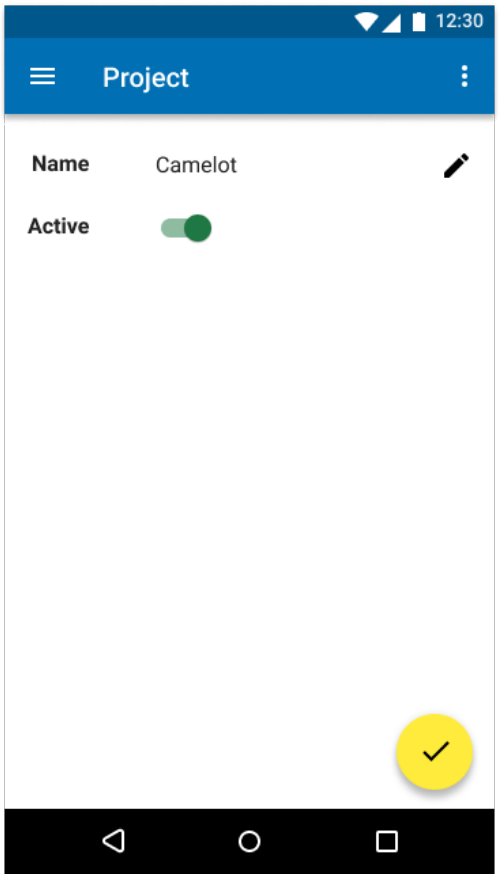


Abbildung 24: Offliss Entwurf Projekt-Details

**Projekte** Projekte können nur vom Projektleiter unter dem Navigationspunkt “Projects“ erfasst werden. Sie sind ebenfalls alphabetisch sortiert. Projekte umfassen mehrere Arbeitspakete (Issues) und haben jeweils einen Status, der anzeigt ob das Projekt noch aktiv geführt wird.
Wie bei den Benutzern, können die Projekte durch Anwählen editiert werden.

**Projekt Details** Die Projekte können direkt in der Ansicht bearbeitet werden. Wird ein Projekt inaktiviert, werden automatisch alle darunter liegenden Arbeitspakete geschlossen und ihre zugewiesenen Personen darüber informiert.

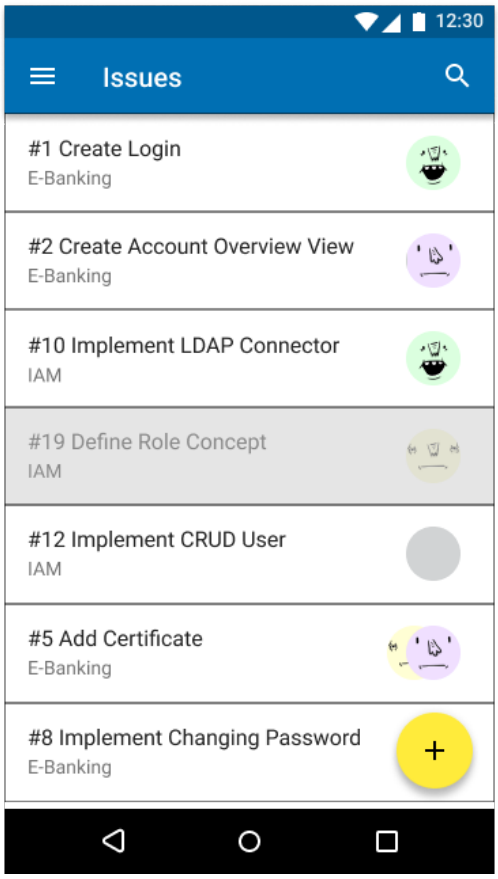


Abbildung 25: Offliss Entwurf Arbeitspaket-Liste

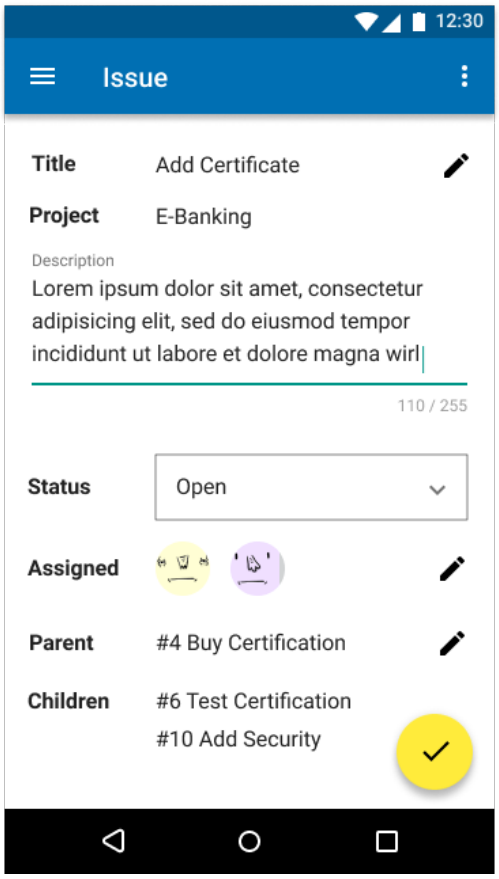


Abbildung 26: Offliss Entwurf Arbeitspaket-Details

**Arbeitspakete** Die Arbeitspakete werden unter “Issues“ angezeigt. Jedes Arbeitspaket gehört zu einem Projekt. Sie sind nach Laufnummer sortiert. Die zugeordneten Benutzer werden mit ihren Avataren dargestellt. Diejenigen, welche noch keine Zuweisung haben, werden mit einem grauen Kreis symbolisiert. Inaktive Arbeitspakete werden ausgegraut dargestellt.
Wie bei den anderen Übersichten, wird die Detail-Ansicht durch Anwählen geladen.

**Arbeitspaket Details** Die Laufnummer wird von Offliss generiert und mit jedem neuen Arbeitspaket erhöht. Der Titel und die Beschreibung können der Entwickler oder der Projektleiter selbst bestimmen. Das Projekt kann nur während der Erfassung gewählt werden. Ein Issue wird durch seinen Status abgeschlossen bzw. erneut geöffnet. Es können mehrere Benutzer gleichzeitig an einem Arbeitspaket arbeiten. Ausserdem kann ein Arbeitspaket einem anderen untergeordnet werden. Dessen untergeordnete “Kinder“ werden ebenfalls aufgelistet, können jedoch nicht bearbeitet werden.
Zudem werden alle Benutzer informiert, wenn ein übergeordnetes Arbeitspaket, bzw. dadurch auch alle seine untergeordneten Pakete, geschlossen werden.

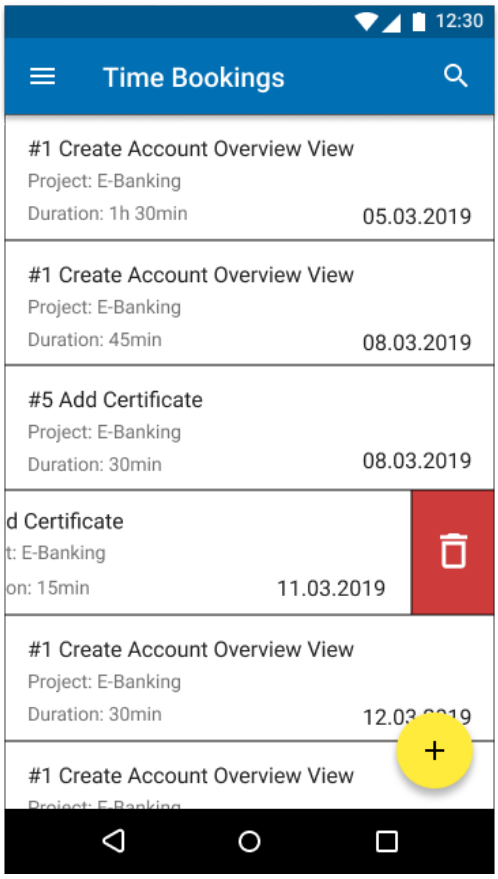


Abbildung 27: Offliss Entwurf Zeitbuchung-Liste

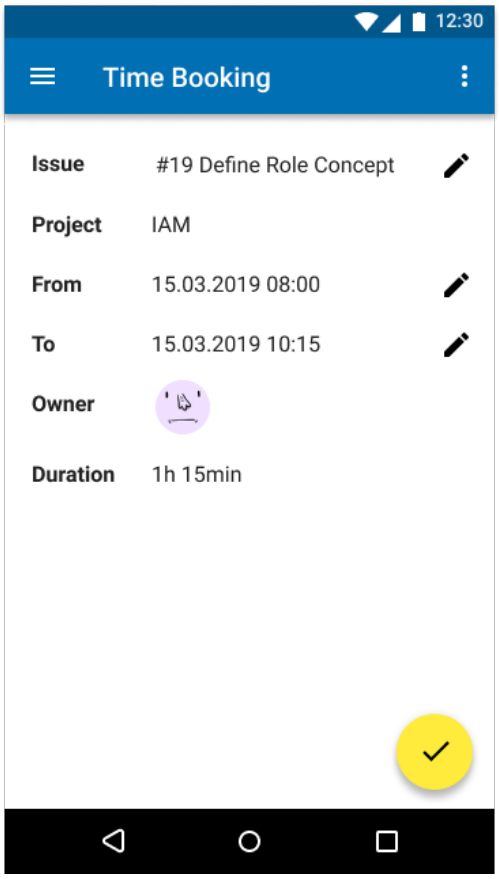


Abbildung 28: Offliss Entwurf Zeitbuchung-Detail

**Zeitbuchungen** Die aufgewendeten Arbeitsstunden eines Mitarbeiters werden mit Zeitbuchungen pro Arbeitspaket unter “Time Bookings“ festgehalten. Der Benutzer kann nur seine eigenen Zeitbuchungen erfassen, editieren und löschen. Sortiert sind die Zeitbuchungen nach Datum.
Indem man nach links wischt, werden die Zeitbuchungen gelöscht. Mit einem Klick auf einen Eintrag, gelangt man in die Detail-Ansicht einer Buchung.

**Zeitbuchung Details** Eine Zeitbuchung gehört immer zu einem Arbeitspaket. Das dazugehörige Projekt wird automatisch angezeigt. Der Benutzer kann sowohl den Start-, als auch den Endzeitpunkt eintragen. Die Dauer wird daraufhin automatisch berechnet, damit die Zeit nochmals kontrolliert werden kann.

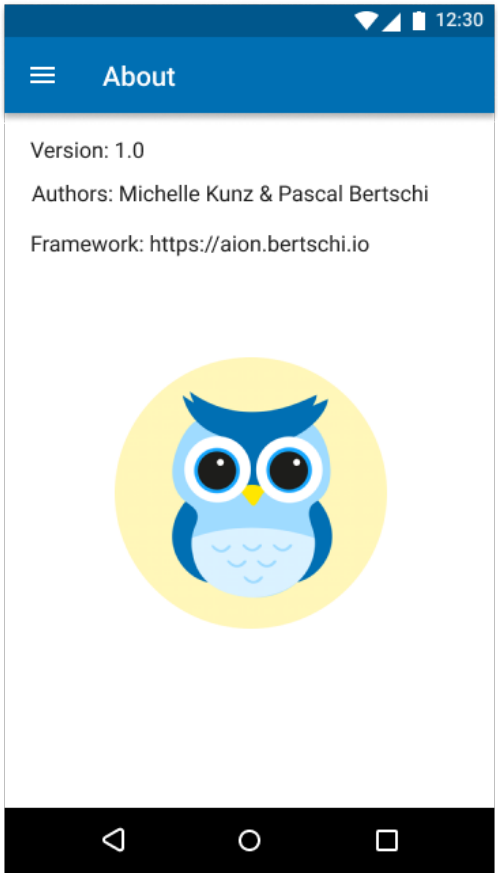


Abbildung 29: Offliss Entwurf About

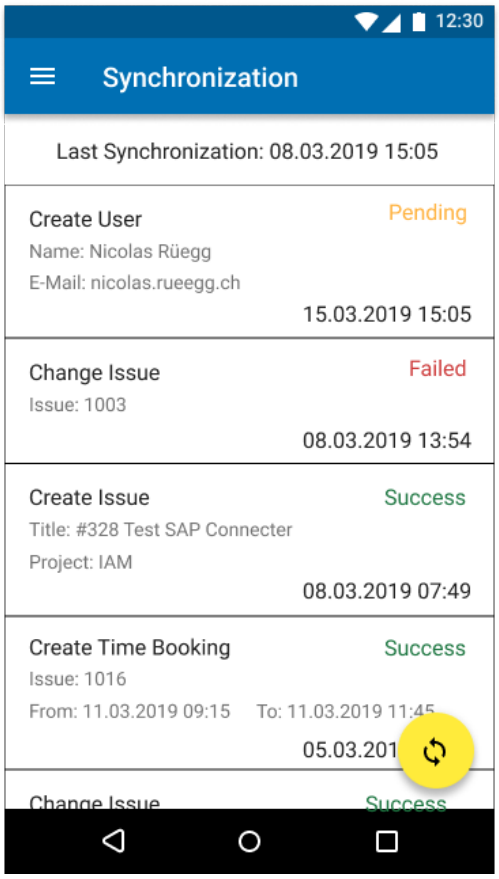


Abbildung 30: Offliss Entwurf Synchronisation

**About** Das About zeigt welche Version aktuell ausgeliefert wurde, wer die Autoren sind und welches das verwendete Framework ist.

**Synchronisation** In der Synchronisation werden die zuletzt getätigten Aktivitäten mit dem Offliss aufgelistet.
“Passed“ symbolisiert eine erfolgreiche Synchronisation.
Events mit dem Status “Pending“ sind noch in Synchronisation.
“Failed“ Events konnten nicht durchgeführt werden. Diese müssen umgehend behoben werden. Der Status bleibt jedoch auch nach erfolgreichem Abgleich bestehen.

# 8 Architektur

Die folgenden Diagramme wurden mit Asta aus unserem Java Code generiert. Asta verwendet eine abweichende Notation von UML für Kompositionen, nämlich ein Pfeil mit einem Kreuz. Aus Gründen der Flexibilität, haben wir die Notationen beibehalten. So können wir die Diagramme jederzeit aktualisieren lassen.

Das Framework Aion und das Anwendungsbeispiel Offliss wurden in folgende Packages aufgeteilt.

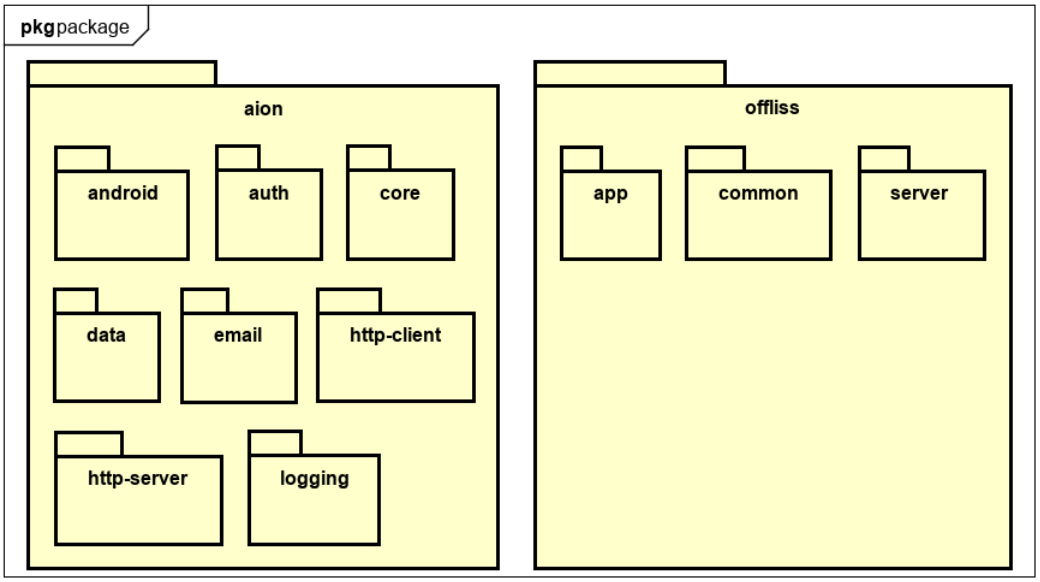


Abbildung 31: Packagediagramm

**Aion** Im Package Aion befindet sich unser Framework, welches allgemeine Funktionalitäten beinhaltet, die in diversen Projekten verwendet werden können.

**Offliss** Im Package Offliss befinden sich die Businesslogik sowie das UI und die Serveranwendungen, die nur für die Entwicklung von Offliss relevant sind.

Beide Packages beinhaltet mehrere Sub-Packages, welche als Aufteilung und Trennung der Funktionalitäten fungieren. Nachfolgend werden diese und deren Klassen in der Reihenfolge ihrer Abhängigkeit genauer erläutert.

## 8.1 Core (Aion)

Das Core Package beinhaltet die Kernlogik unseres Aion Frameworks und wird von fast allen anderen Projekten verwendet. Es definiert die Schnittstellen und grundlegenden Strukturen, welche von anderen implementiert werden. In den folgenden Unterkapiteln wird auf die einzelnen Bestandteile eingegangen.

### 8.1.1 Timeline

In einer Timeline können Events angefügt, jedoch nicht gelöscht oder verändert werden. Die Methoden in der ITimeline-Schnittstelle werden später genutzt, um durch die Liste von Events zu iterieren.

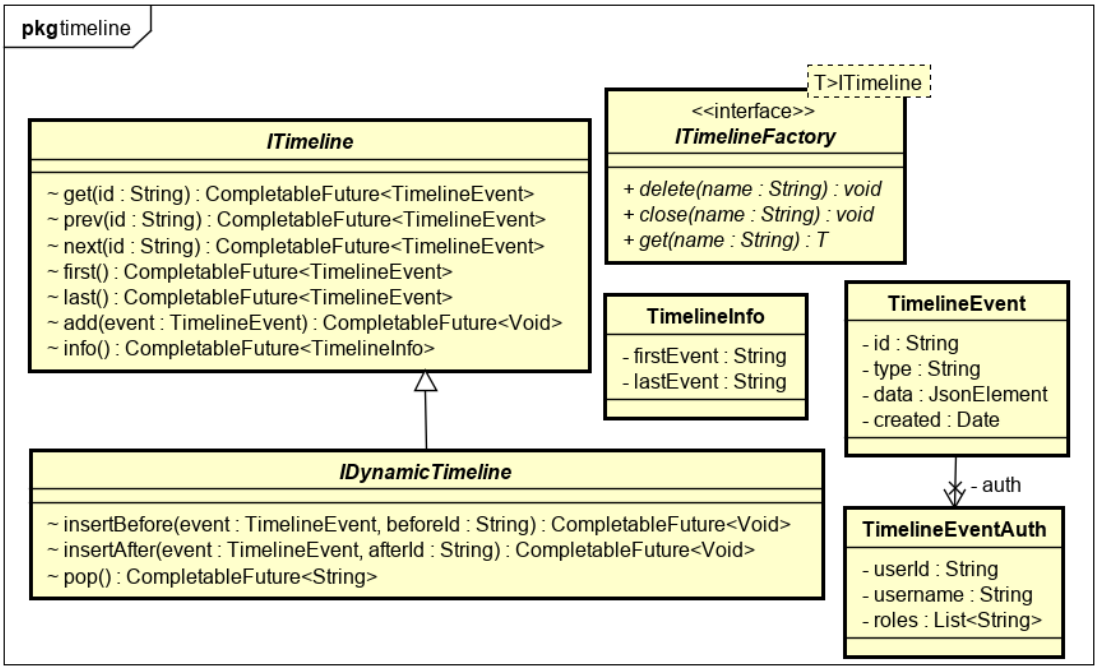


Abbildung 32: Timeline Klassendiagramm

**TimelineEvent** Ein TimelineEvent besteht aus einer ID vom Typ String. Diese muss vom Client mit einer UUID abgefüllt werden. Dadurch können sie eindeutige IDs generieren, ohne Absprache mit einem zentralen Server.

Der Type definiert um welchen Event es sich handelt und ist ebenfalls ein String, damit er serialisiert werden kann.

Die Data Eigenschaft beinhaltet die Event-Informationen und ist vom Typ JsonElement. Dieses serialisiert das Datenobjekt, wenn es an eine Datenbank oder API übermittelt wird. Damit bei der Deserialisierung der ursprüngliche Datentyp wiederhergestellt werden kann, gibt es die Methode getData, welche mit der Angabe eines Klassentyps, das Objekt konvertiert.

**TimelineEventAuth** Zu jedem Event wird der ausführende Benutzer und dessen Rollen gespeichert. Bei der Erstellung des Events wird geprüft, ob dieser tatsächlich vom angegebenen Benutzer erstellt wurde und die Rollen korrekt sind. Die Überprüfung, ob die Rolle des Benutzers diesen Event ausführen darf, wird erst später in der Verarbeitung sichergestellt.

**ITimeline** Die ITimeline-Schnittstelle stellt eine Reihe von Events dar und bietet Methoden an, welche diese abfragen oder erweitern. Über diese Funktionen wird später die Synchronisation zwischen Client und Server stattfinden. Es ist wichtig, dass Events nur erstellt werden können und nicht gelöscht, da ansonsten die Synchronisationslogik nicht funktionieren würde. Diese Schnittstelle wird von verschiedenen Klassen implementiert, welche die dahinter liegenden Technologien abstrahieren. Dies erlaubt es uns, die Kernlogik, unabhängig der Technologie, zu programmieren und ermöglicht es in Zukunft Erweiterungen mit geringen Änderungen zu implementieren.

**ITimelineFactory** Diese Schnittstelle ist für die Erstellung von ITimelines verantwortlich.

**IDynamicTimeline** Die IDynamicTimeline-Schnittstelle erweitert ITimeline um Methoden, welche Events einfügen oder entfernen. Dies wird auch für die Synchronisation auf der Clientseite verwendet.

**TimelineInfo** Die TimelineInfo enthält den ersten und letzten Event einer Timeline. Diese werden verwendet, um in der Synchronisation möglichst schnell zu erkennen, ob der Datenstand aktuell ist.

### 8.1.2 Application

Eine Applikation besteht aus einem Modul, welches definiert, was alles zur Applikation gehört. Sie kann Events aus der Timeline verarbeiten und gibt danach ein Resultat zurück. Die Applikation muss gestartet und gestoppt werden, bevor Events bearbeitet werden können.

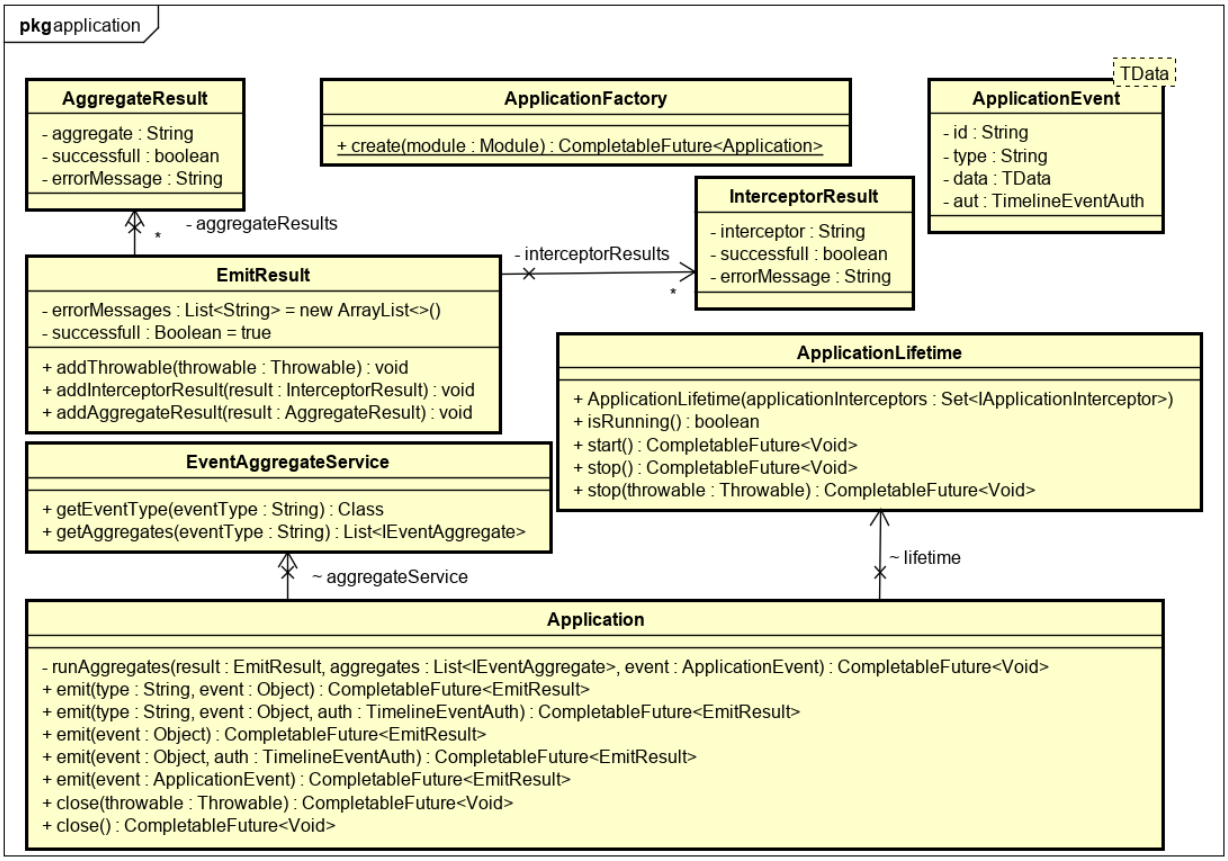


Abbildung 33: Application Klassendiagramm

**ApplicationFactory** Die ApplicationFactory erstellt aus einem Modul eine lauffähige Applikation. Die create-Methode returniert eine Applikationsinstanz, sobald diese fertig initialisiert und bereit ist.

**Application** Die Application ist für die Verarbeitung von Events verantwortlich und nutzt dazu die Event-Interceptoren sowie Event-Aggregatoren.

**ApplicationLifetime** Der ApplicationLifetime agiert mit den Applikationsinterceptoren und ist für das Starten und Stoppen der Applikation verantwortlich. Dabei werden die Interceptoren aufgerufen, welche steuern, wann die Applikation bereit ist oder abgebrochen wird.

**EventAggregateService** Der EventAggregateService entscheidet für welche Events, welche Event-Aggregatoren ausgeführt werden müssen. Dies wird aufgrund des Event-Type entschieden, welcher im TimelineEvent zu finden ist.

**ApplicationEvent** Der ApplicationEvent gleicht dem TimelineEvent, mit einem wichtigen Unterschied, nämlich der Datentyp der Data Eigenschaft. Vor der Ausführung eines TimelineEvents, wird der Datentyp ermittelt und konvertiert, dadurch müssen sich die Aggregatoren und Interceptoren nicht um die Serialisierung kümmern.

**EmitResult** Die EmitResult-Klasse repräsentiert das Resultat eines ausgeführten Events. Es enthält alle Fehlermeldungen der ausgeführten Event-Aggregatoren sowie Interceptoren und den Status der gesamten Ausführung. War die Ausführung nicht erfolgreich, werden die fehlerhaften Komponenten in den AggregateResults und InterceptorResults aufgelistet.

**AggregateResult** Die AggregateResult-Klasse repräsentiert das Resultat eines einzelnen Event-Aggregats und enthält den Namen des Aggregats sowie eine Fehlermeldung im Fehlerfall. Die Fehlermeldung und der Aggregatsname müssen serialisierbar sein und sind deshalb vom Typ String.

**InterceptorResult** Die InterceptorResult-Klasse repräsentiert das Resultat eines einzelnen Event-Interceptors und enthält ebenfalls den Namen des Interceptors sowie eine Fehlermeldung im Fehlerfall. Auch hier sind aufgrund der Serialisierung die Datentypen Strings.

### 8.1.3 Module

Das Aion Framework ist modular aufgebaut und verwendet Module, um dies zu realisieren. Ein Modul kann Events, Event-Aggregatoren, Event-Interceptoren und Applikationsinterceptoren beinhalten. Es soll dazu dienen, Funktionalitäten zusammenzufassen und diese in anderen Modulen wiederzuverwenden. Die Module können verschachtelt sein.

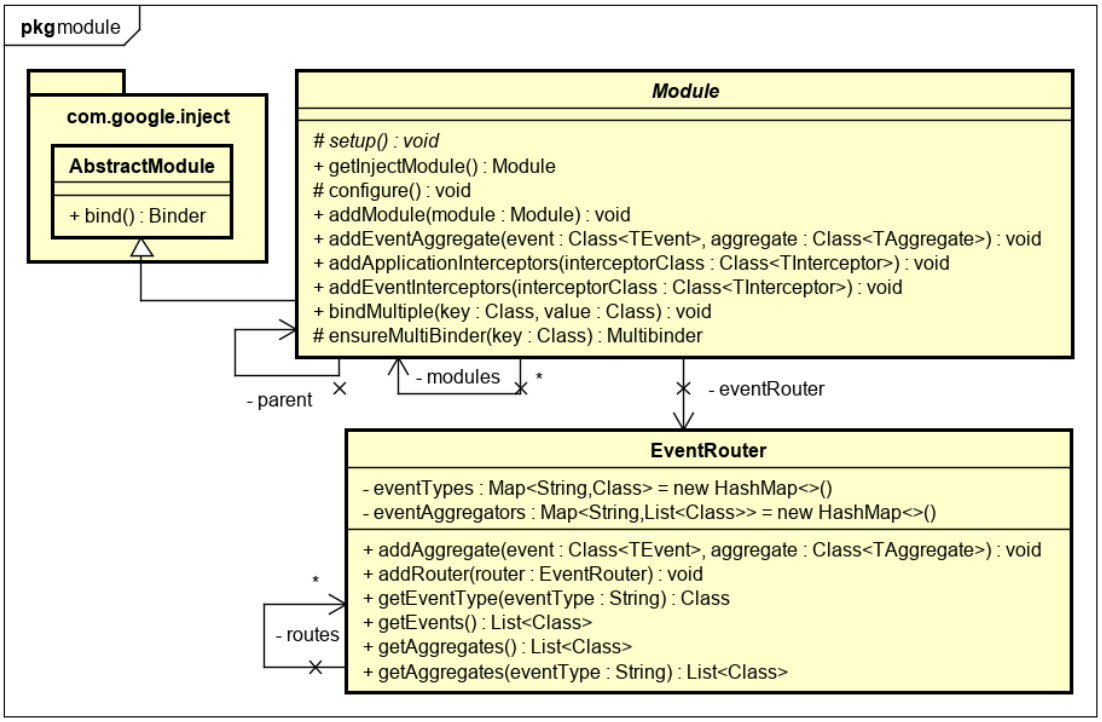


Abbildung 34: Module Klassendiagramm

**Module** Die abstrakte Klasse Module erlaubt es Events, Event-Aggregatoren, Event-Interceptoren sowie Applikationsinterceptoren zu deklarieren. Ziel des Moduls ist es, die Klassen zu registrieren, welche zur Laufzeit verwendet werden. Die setup-Methode wird jeweils einmal beim Start der Applikation aufgerufen und ist für die Registrierung dieser Klassen gedacht. Dazu werden die add-Methoden verwendet. Die AbstractModule-Klasse kommt aus dem Guice Dependency Injection Framework. Mit der Erweiterung unserer Module-Klasse, können zusätzliche Funktionen registriert werden, welche dann später von der Application genutzt werden. Ein Modul kann mehrere Sub-Module beinhalten und dann die definierten Funktionen dieser nutzen.

**EventRouter** Die EventRouter-Klasse ist verantwortlich dafür, dass die richtigen Event-Aggregatoren aufgerufen werden. Zum setup-Zeitpunkt im Modul, kann angegeben werden, welcher Event welche Aggregatoren auslöst. Diese Information wird im EventRouter festgehalten. Soll zu einem späteren Zeitpunkt ein Event behandelt werden, gibt der EventRouter die registrierten Aggregatoren dazu zurück.

### 8.1.4 Event Aggregate

Ein Aggregator ist dazu da, die Events in einer Timeline zu verarbeiten. Dazu wird durch alle Events iteriert, beim ersten beginnend. Der Aggregator kann sich auf einen bestimmten Event einschreiben und verändert dadurch beispielsweise die Projection.

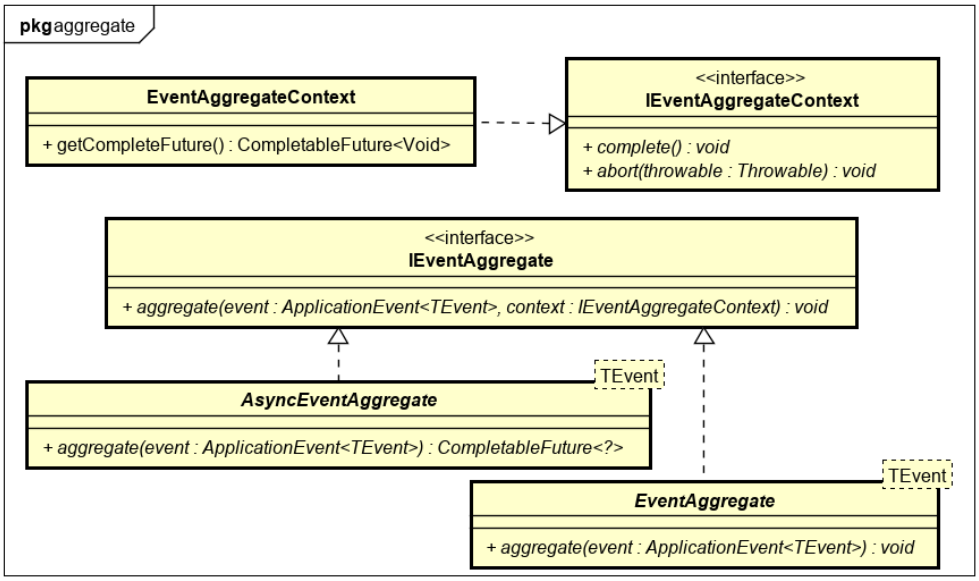


Abbildung 35: Aggregate Klassendiagramm

**IEventAggregateContext** Die IEventAggregateContext-Schnittstelle wird in IEventAggregate verwendet, um das Ausführungsresultat des Event-Aggregators zu setzen. Die Methode complete beendet den Aggregator erfolgreich, während die Methode abort der Applikation einen Fehler zurückmeldet.

**EventAggregateContext** Der EventAggregateContext implementiert die IEventAggregateContext-Schnittstelle und erlaubt dem Ersteller des Objekts, das Resultat als CompletableFuture abzufragen.

**IEventAggregate** Die Methode aggregate nimmt einen Event und einen Context entgegen. In dieser befindet sich später die Anwendungslogik für den jeweiligen Event.

**EventAggregate** Die EventAggregate-Klasse implementiert die IEventAggregate-Schnittstelle und abstrahiert den Context Parameter. Dadurch kann ein Aggregator die aggregate-Methode implementieren, ohne sich um die Rückmeldung an den Context zu kümmern. Wir wollten damit vermeiden, dass Anwender von Aion fälschlicherweise vergessen den Context abzuschliessen und die Applikation dadurch stehen bleibt. Wird eine Fehlermeldung innerhalb der aggregate-Methode geworfen, wird diese abgefangen und dem Context weitergeleitet.

**AsyncEventAggregate** Die AsyncEventAggregate-Klasse implementiert ebenfalls die IEventAggregate-Schnittstelle, erlaubt es aber, ein CompletableFuture zurückzugeben, anstelle von void. Damit sollen auch asynchrone Operationen einfacher implementiert werden können.

### 8.1.5 Event Interceptor

Event Interceptors werden vor den Aggregatoren ausgeführt und sind dazu gedacht, dessen Funktionalität zu erweitern. Sie können die Ausführung der Aggregatoren verhindern sowie die Daten des Events erweitern und modifizieren.

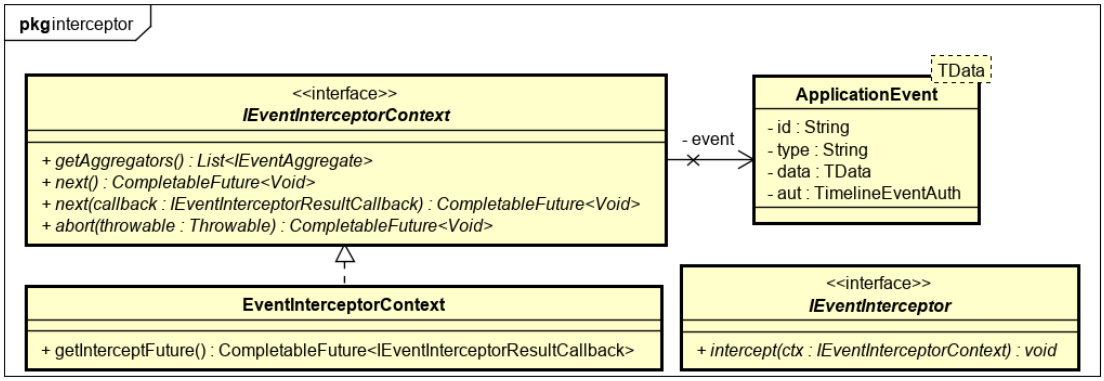


Abbildung 36: Event Interceptor Klassendiagramm

**IEventInterceptorContext** Die Schnittstelle kann die Ausführung von Event-Aggregatoren abbrechen oder starten. next, mit oder ohne Callback, signalisiert der Applikation, dass die Event-Aggregatoren ausgeführt werden können. Der Callback wird nach den Aggregatoren ausgeführt und erlaubt es dem Interceptor danach weitere Aktionen durchzuführen. Möchte der Interceptor die Ausführung verhindern, kann mit abort abgebrochen werden. Dann wird kein weiterer Event-Aggregator angestossen.

**EventInterceptorContext** Diese Klasse ermöglicht es dem Ersteller des Objekts das Resultat des Interceptors als CompletableFuture abzufragen.

### 8.1.6 Application Interceptor

Die Initialisierung einer Applikation kann durch Applikationsinterceptoren gesteuert werden. Diese werden beim Applikationsstart aufgerufen und können diesen gegebenenfalls verzögern. Beim Applikationsstopp können sie eingreifen und aufräumen.

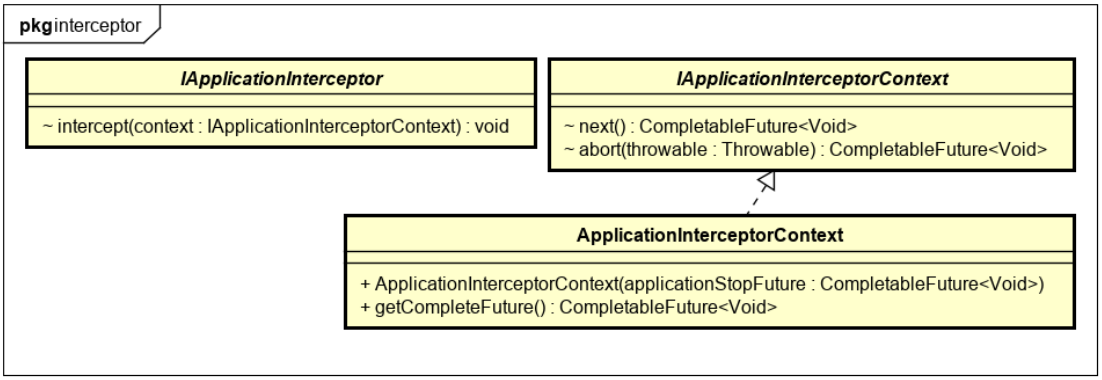


Abbildung 37: Application Interceptor Klassendiagramm

**IApplicationInterceptor** Die IApplicationInterceptor-Schnittstelle definiert die intercept-Methode, welche zum Applikationsstart aufgerufen wird.

**IApplicationInterceptorContext** Durch die IApplicationInterceptorContext-Schnittstelle kann ein Applikationsinterceptor steuern, wann die Applikation bereit ist. Solange next nicht aufgerufen wird, befindet sie sich in der Initialisierungsphase und kann keine Events verarbeiten. Sollte ein Fehler auftreten, kann mit der abort-Methode die Applikation gestoppt werden.

**ApplicationInterceptorContext** Die ApplicationInterceptorContext-Klasse implementiert die IApplicationInterceptorContext-Schnittstelle. Es erlaubt dem Ersteller des Objekts ein CompletableFuture mitzugeben, welches beim abort-Aufruf ausgelöst wird sowie ein CompletableFuture abzufragen, welches den next-Aufruf repräsentiert.

### 8.1.7 Timeline Processor

Der Timeline Processor ist für die Ausführung der Events in einer Timeline verantwortlich. Dabei wird sichergestellt, dass die Resultate persistiert und keine Events doppelt verarbeitet werden.

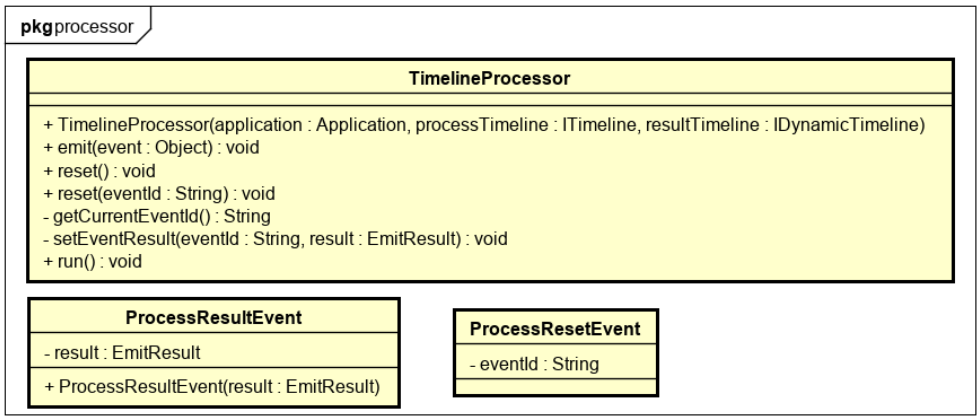


Abbildung 38: Processor Klassendiagramm

**TimelineProcessor** Die TimelineProcessor-Klasse benötigt eine Applikation, welche die Events ausführt und ein Resultat returniert. Die Events, die verarbeitet werden, kommen aus der processTimeline, welche ebenfalls mitgegeben wird. Sobald die Applikation ein Event verarbeitet, wird dessen Resultat in Form eines ProcessResultEvents in die resultTimeline geschrieben.

Die Methode run prüft ob es in der processTimeline neue Events gibt und verarbeitet diese. Der Stand der Verarbeitung kann jederzeit mit reset zum Anfang oder mit reset(eventId) zu einem bestimmten Event zurückgesetzt werden.

**ProcessResultEvent** Die Klasse ProcessResultEvent ist ein TimelineEvent und beinhaltet das Ausführungsresultat der Applikation.

**ProcessResetEvent** Die Klasse ProcessResetEvent ist ebenfalls ein TimelineEvent, wird allerdings nicht persistiert. Dieser Event wird auf der Applikation ausgeführt, wenn die reset-Methode aufgerufen wurde und erlaubt es der Applikation auf das Zurücksetzen zu reagieren. Dies wird zum Beispiel im Data Module benötigt, um den Datenstand zurückzusetzen.

### 8.1.8 Timeline Synchronizer

Der Timeline Synchronizer ist dazu da zwei Timelines abzugleichen. Normalerweise ist eine der zwei Timelines die Server Timeline und die andere die Client Timeline auf dem lokalen Gerät. Bei der Synchronisation dürfen auf der Server Timeline nur Events angefügt werden, bei der Client Timeline können Events aber auch dazwischen eingefügt werden.

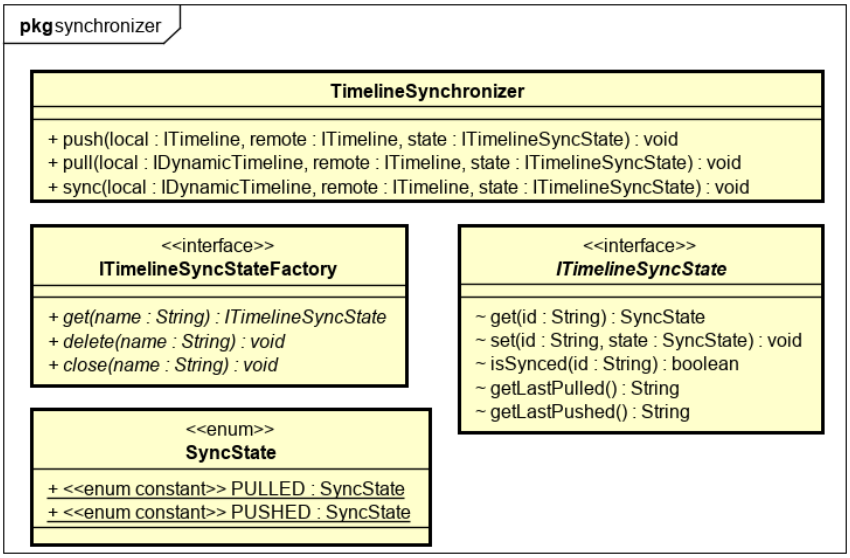


Abbildung 39: Synchronizer Klassendiagramm

**TimelineSynchronizer** Die TimelineSynchronizer-Klasse synchronisiert zwei Timelines und bietet dazu drei Möglichkeiten. Die push-Methode übermittelt nur lokale Events, welche sich nicht auf der Server Timeline befinden. Die pull-Methode übermittelt nur Events aus der Server Timeline, welche sich nicht auf dem lokalen Client befinden. Die sync-Methode führt zuerst einen push und dann einen pull aus.

**ITimelineSyncState** Die ITimelineSyncState-Schnittstelle speichert den Synchronisationsstand und wird von der TimelineSynchronizer-Klasse verwendet. Durch die Speicherung dieses Zustands, kann die Kommunikation reduziert werden. Ausserdem kann schneller ermittelt werden, was noch abzugleichen ist.

**ITimelineSyncStateFactory** Die ITimelineSyncStateFactory-Schnittstelle liefert einen ITimelineSyncState zurück und ist verantwortlich für die Erstellung von diesem.

**SyncState** Der SyncState ist entweder Pulled oder Pushed und bedeutet, dass der Event vom Server geladen oder an den Server übermittelt wurde.

### 8.1.9 Synchronisation

In den folgenden Diagrammen ist der Ablauf einer Synchronisation zusammengefasst. Diese findet zwischen zwei Timelines statt. Eine der beiden ist die Server Timeline, bei welcher nur Events zuletzt angefügt werden dürfen. Bei der anderen handelt es sich um die Client Timeline, welche sich der Server Timeline anpasst.

**Pull Synchronisation** Die Pull Synchronisation ist dazu da, Daten aus der Server Timeline auf die Client Timeline zu übertragen. Der Fortschritt wird jeweils im TimelineSyncState erfasst, damit bei einem Abbruch oder beim nächsten Durchlauf, an der letzten Stelle fortgefahren werden kann.

Zu Beginn wird die info-Methode aufgerufen, welche den letzten Event zurückgibt. Dies ist nötig, um eine endlose Synchronisation zu verhindern, wenn zum Beispiel schneller Events angefügt werden als heruntergeladen.

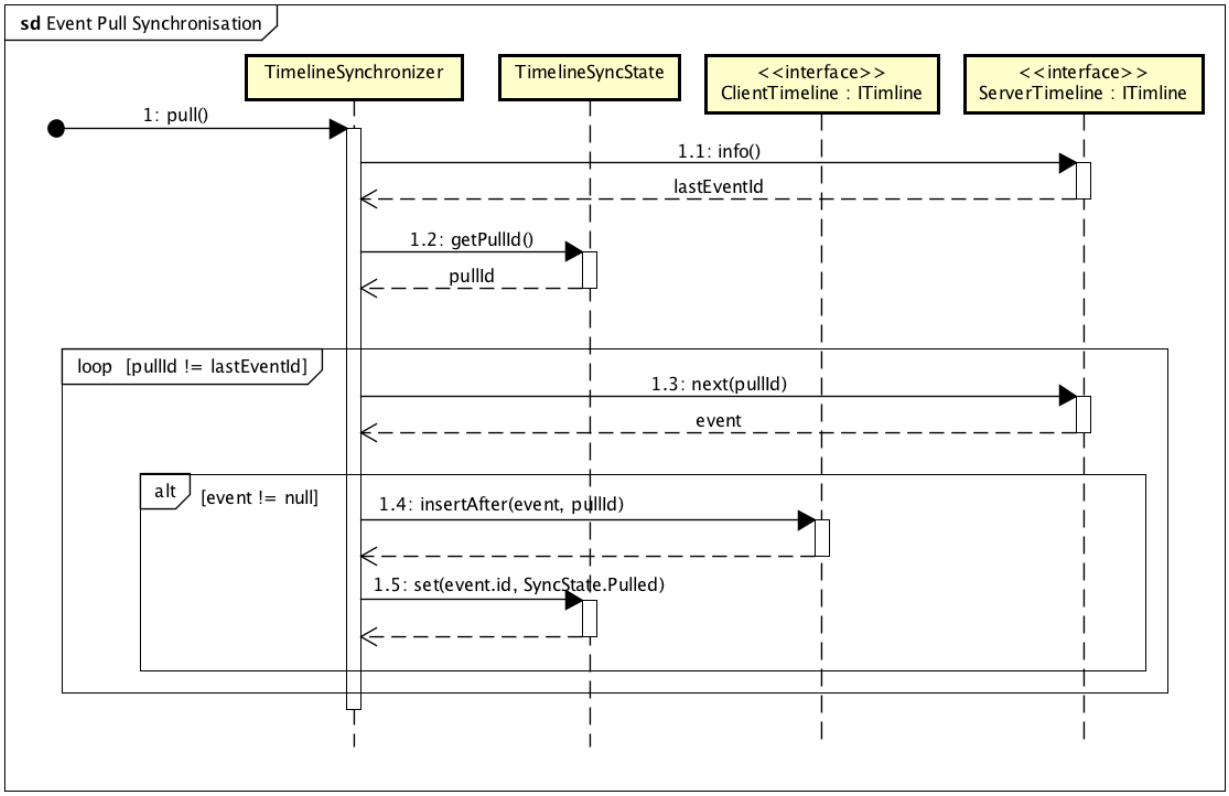


Abbildung 40: Pull Synchronisation Sequenzdiagramm

**Push Synchronisation** Die Push Synchronisation überträgt Daten aus der Client Timeline auf die Server Timeline. Der Fortschritt wird auch hier jeweils im TimelineSyncState erfasst, um doppelte Übertragungen im Fehlerfall zu verhindern.

Auch hier wird zuerst die info-Methode aufgerufen, um eine endlose Synchronisation zu vermeiden.

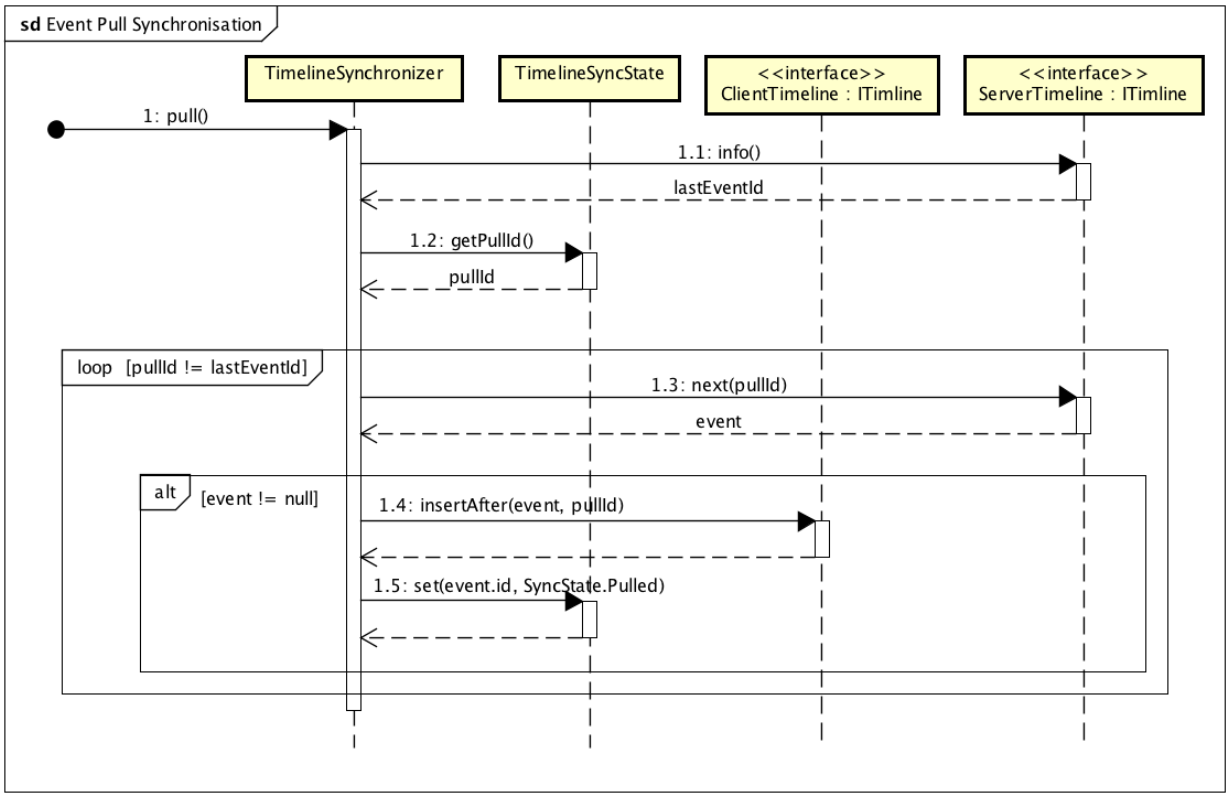


Abbildung 41: Push Synchronisation Sequenzdiagramm

## 8.2 Data (Aion)

Das Data Projekt befasst sich mit der Projection in Aion. Die Implementation haben wir mit SQLite sowie InMemory umgesetzt. Andere Datenbanktechnologien könnten analog implementiert werden.

### 8.2.1 Data

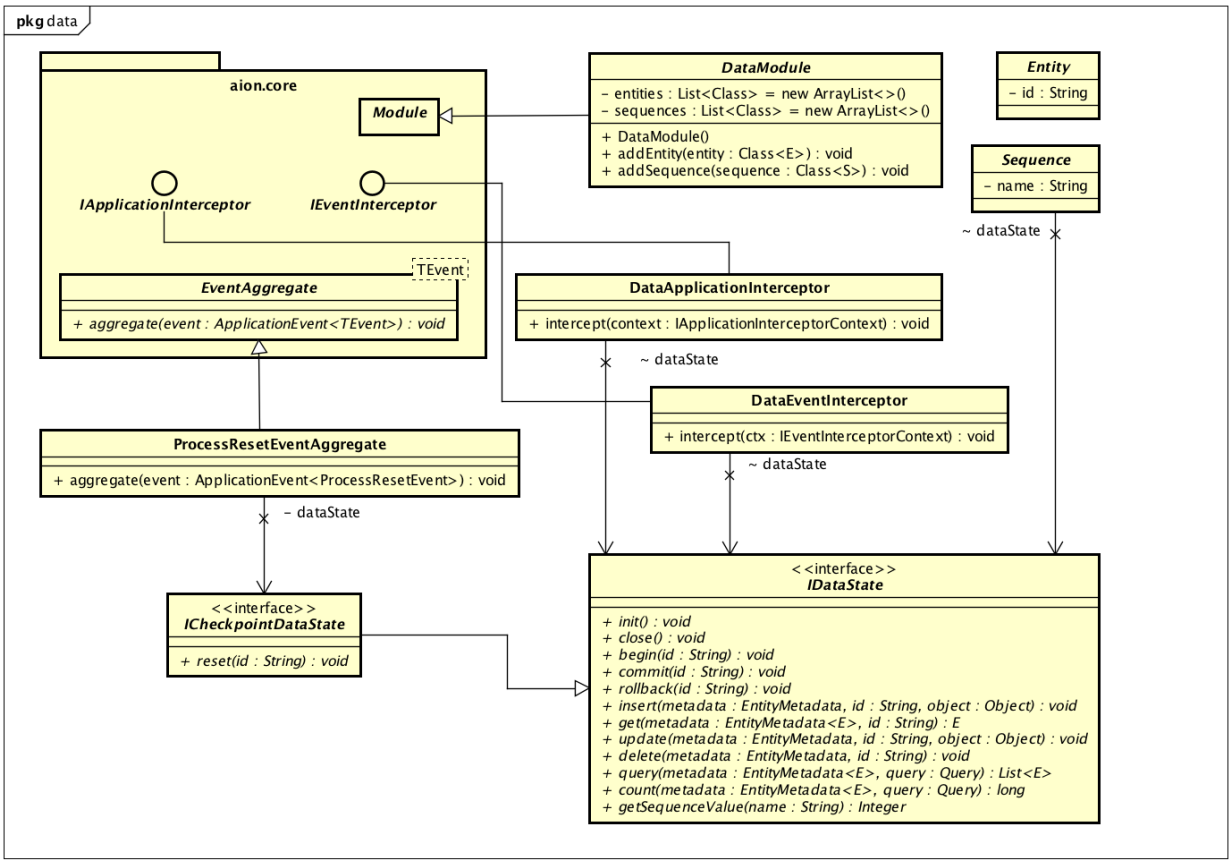


Abbildung 42: Data Module Klassendiagramm

**DataModule** Das DataModule ist die allgemeine Abstraktion, welche Anwender von Aion verwenden. Dieses Modul ist technologieunabhängig und kann daher für jede Anwendung mit der gewünschten Implementation ausgetauscht werden. Es ist eine abstrakte Klasse, welche von den spezifisch implementierten Modulen erweitert wird, wie zum Beispiel SQLiteDataModule. Das DataModule ist dazu da, Entitäten und Sequenzen zu registrieren, welche in den Aggregatoren und Interceptoren verwendet werden können.

**IDataState** Die IDataState-Schnittstelle abstrahiert die Datenbankimplementation. Alle Datenoperationen, egal ob schreibend oder lesend gehen da durch. Das Modul und ihre Event-Aggregatoren und Interceptoren sorgen dafür, dass die entsprechenden Methoden zum richtigen Zeitpunkt ausgeführt werden.

**ICheckpointDataState** Die ICheckpointDataState-Schnittstelle erweitert die IDataState um eine reset-Methode. Diese soll die Projections in den Zustand des übergebenen Events zurücksetzen.

**DataApplicationInterceptor** Beim Start- und Stoppzeitpunkt der Applikation wird der DataApplication-Interceptor aufgerufen, welcher die init-, bzw. close-Methode des DataStates ausführt. Dadurch kann beispielsweise die Datenbankverbindung aufgebaut und geschlossen werden.

**DataEventInterceptor** Der DataEventInterceptor wird vor und nach den Event-Aggregates aufgerufen. Er führt die begin-, commit- oder rollback-Methode aus. Damit sind Transaktionen möglich.

**ProcessResetEventAggregate** Der ProcessResetEventAggregate wird ausgelöst, wenn ein Process-ResetEvent gesendet wird. Dieser ruft dann die reset-Methode auf dem CheckpointDataState auf. Nur wenn der DataState die ICheckpointDataState-Schnittstelle implementiert, wird dieser Aggregator registriert.

**Entity** Die Entity-Klasse repräsentiert eine Tabelle in einer relationalen Datenbank.

**Sequence** Die Sequence-Klasse repräsentiert eine Sequenz in einer relationalen Datenbank.

### 8.2.2 ORM

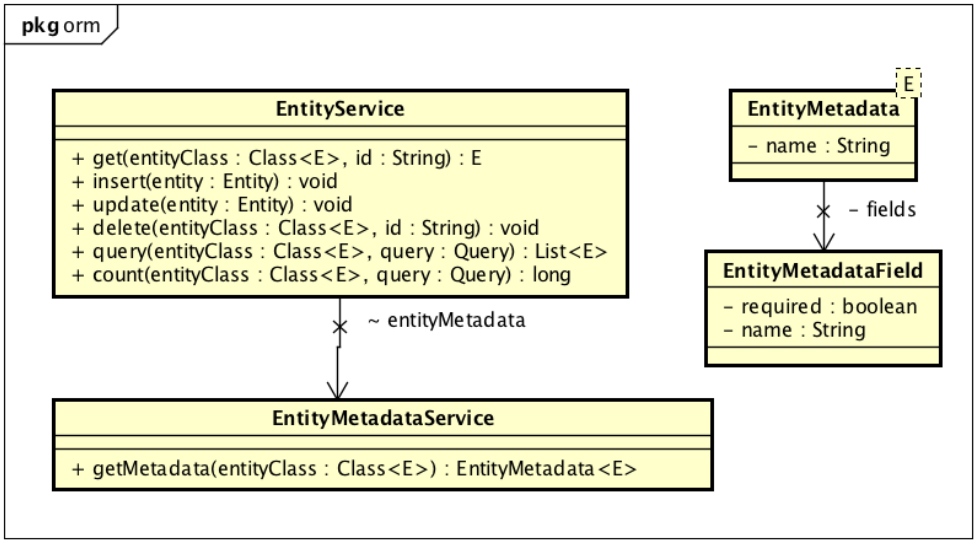


Abbildung 43: Data ORM Klassendiagramm

**EntityService** Der EntityService ist ein Service, welcher von den Event-Aggregatoren eingebunden werden kann, um die Projection abzufragen und zu verändern.

**EntityMetadataService** Der EntityMetadataService wird vom EntityService genutzt, um die SQL Abfragen zu generieren. Dieser liefert die Metadata, welche über Reflection aus den registrierten Entitäten ausgelesen werden.

**EntityMetadata** Die EntityMetadata-Klasse repräsentiert die Datenbanktabelle und deren Felder.

**EntityMetadataField** Diese Klasse repräsentiert die Tabellenspalte und deren Eigenschaften.

### 8.2.3 Query

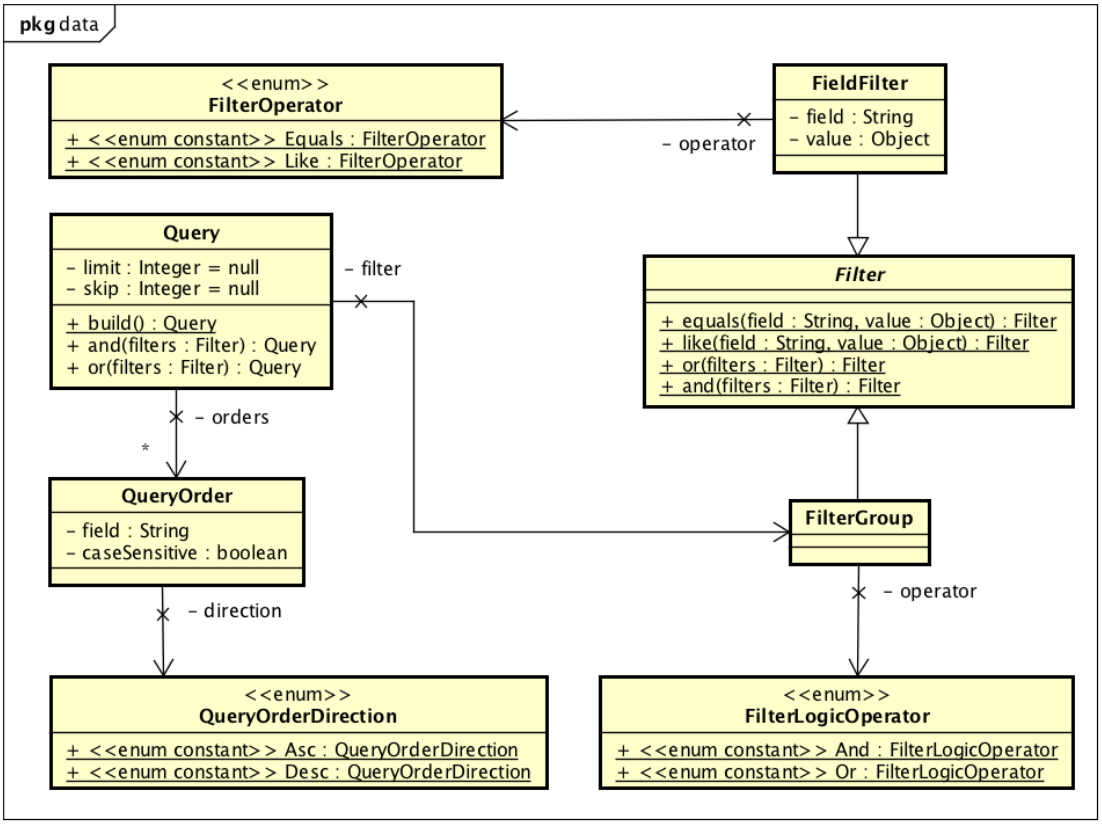


Abbildung 44: Data Query Klassendiagramm

**Query** Die Query-Klasse repräsentiert eine SQL-Abfrage. Diese wird vom ORM verwendet, um eine Liste an Entitäten zurückzuliefern. Es wurden nur die Operatoren umgesetzt, welche für die Offliss Anwendung benötigt wurden.

**QueryOrder** / **QueryOrderDirection** Mit der QueryOrder-Klasse wird die Sortierung einer Abfrage angegeben. Dazu kann das Feld, die Richtung und ob der Wert Gross- und Kleinschreibung beachten soll, mitgegeben werden.

**FilterGroup** / **FilterLogicOperator** Eine FilterGroup ist eine Liste an Filter, mit einer logischen Verknüpfung.

**FieldFilter** / **FieldOperator** Ein FieldFilter ist eine Bedingung, welche ein Feld, einen Wert und den Vergleichsoperator beinhaltet.

**Filter** Der Filter ist eine abstrakte Klasse, welche von FieldFilter und FilterGroup implementiert wird. Diese Klasse erlaubt die Verschachtelung der beiden Klassen ineinander, um alle Konditionsbedingungen abzudecken.

### 8.2.4 SQLite Connection

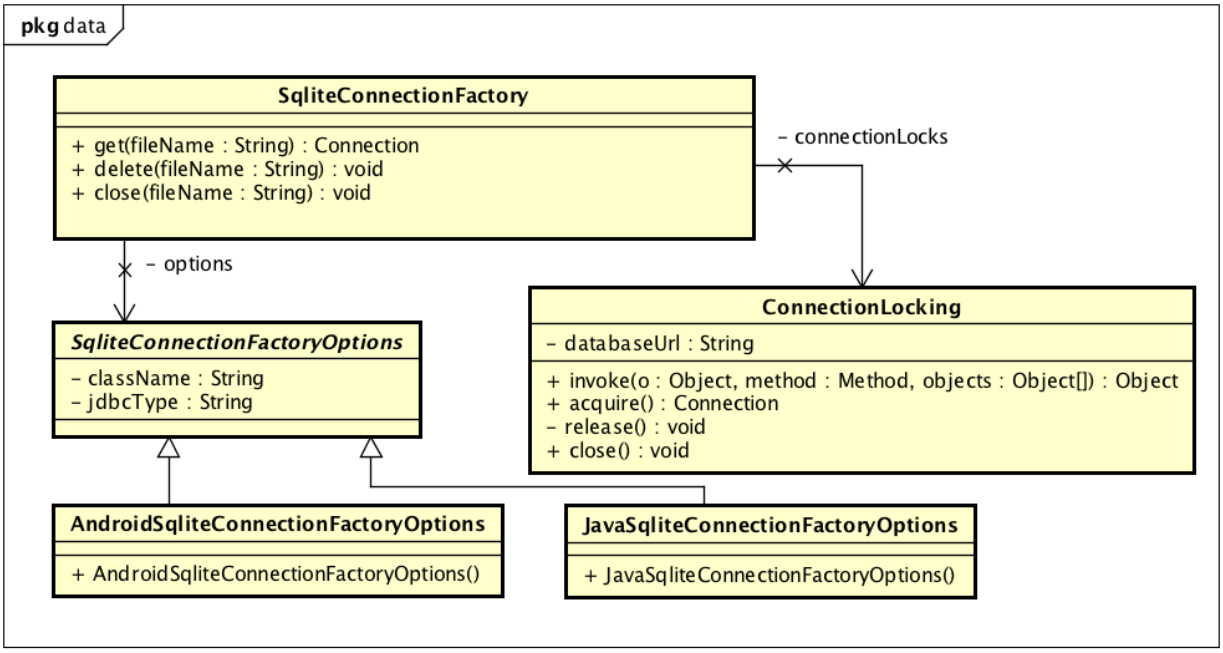


Abbildung 45: Data SQLite Connection Klassendiagramm

**SqliteConnectionFactory** Die SqliteConnectionFactory ist für die Erstellung und Terminierung von JDBC Verbindungen verantwortlich.

**SqliteConnectionFactoryOptions** Die SqliteConnectionFactoryOptions beinhaltet Informationen zum JDBC Verbindungsaufbau, welcher sich auf den verschiedenen Plattformen Java und Android unterscheiden kann.

**AndroidSqliteConnectionFactoryOptions / JavaSqliteConnectionFactoryOptions** Die AndroidSqliteConnectionFactoryOptions-Klasse beinhaltet die Optionen für die Android Plattform und die JavaSqliteConnectionFactoryOptions für die Java Plattform.

**ConnectionLocking** Die ConnectionLocking-Klasse stellt sicher, dass eine Verbindung von nur einem Thread gleichzeitig verwendet wird.

### 8.2.5 SQLite Module

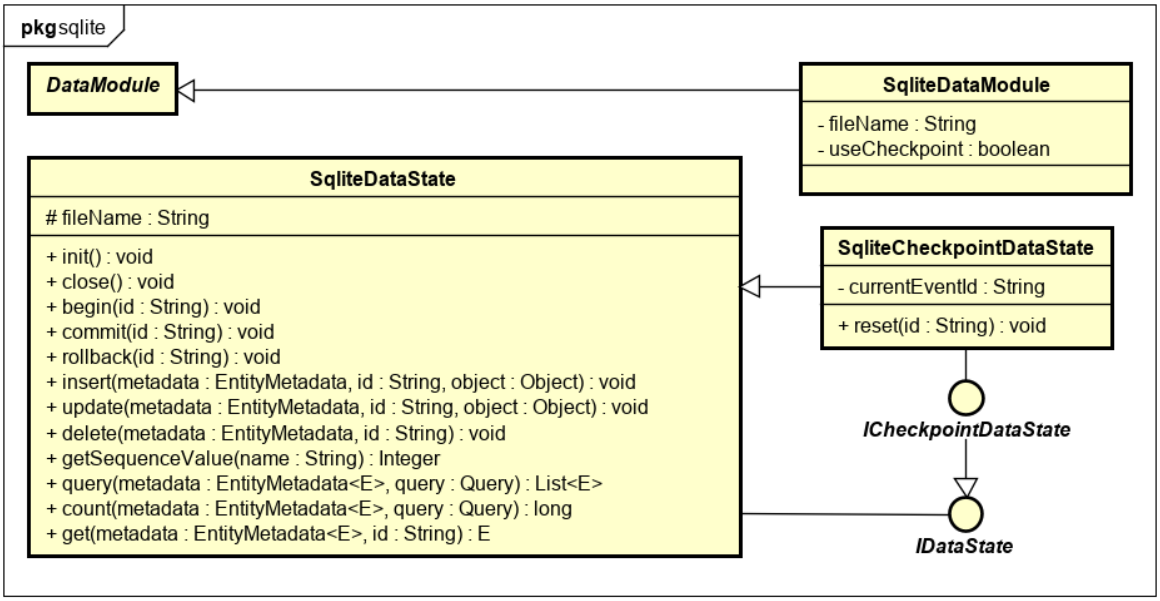


Abbildung 46: Data SQLite Module Klassendiagramm

**SqliteDataModule** Das SqliteDataModule ist die Implementation des DataModule für SQLite. Bei der Initialisierung kann angegeben werden, ob ein Checkpoint verwendet werden soll.

**SqliteDataState** Diese Klasse implementiert alle SQLite Aufrufe, die über das DataModule eingehen.

**SqliteCheckpointDataState** SqliteCheckpointDataState erweitert den SqliteDataState um die Fähigkeit, den Datenbankzustand zurückzusetzen. Dazu wird bei allen Datenmanipulationen eine Operation dazu gespeichert, welche die Veränderung wieder rückgängig macht.

### 8.2.6 SQLite Operation

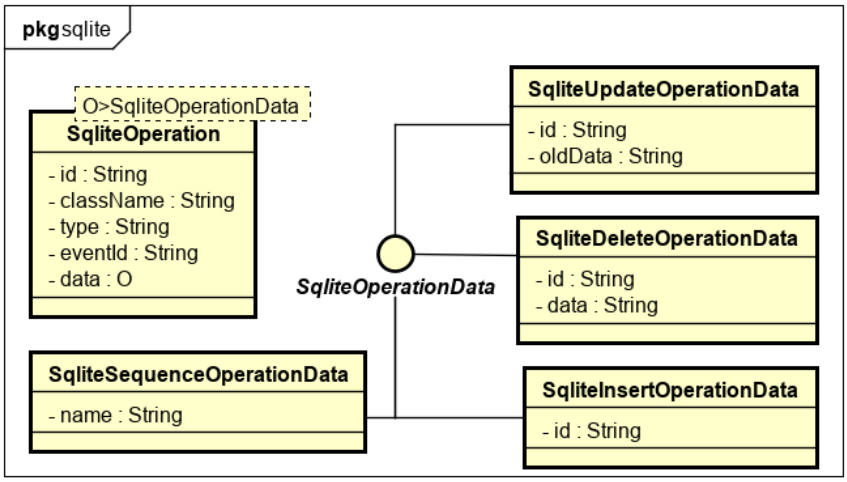


Abbildung 47: Data SQLite Operation Klassendiagramm

**SqliteOperation / SqliteOperationData** Die SqliteOperation-Klasse repräsentiert eine Datenmanipulation und beinhaltet die nötigen Informationen, um diese rückgängig zu machen.

**SqliteUpdateOperationData** Diese Klasse stellt ein Update-Statement dar. Vor dem Durchführen werden die alten Werte gespeichert, um diese in einem Fehlerfall wiederherstellen zu können.

**SqliteDeleteOperationData** Die SqliteDeleteOperationData-Klasse repräsentiert ein Delete-Statement. Auch hier werden die alten Daten für die Wiederherstellung gespeichert.

**SqliteInsertOperationData** SqliteInsertOperationData stellt ein Insert-Statement dar. Hierzu wird die ID des Datensatzes gespeichert, damit das Objekt im Fehlerfall gelöscht werden kann.

**SqliteSequenceOperationData** Die SqliteSequenceOperationData-Klasse repräsentiert die Erhöhung einer Sequenz. Dazu wird der Name der Sequenz gespeichert, damit diese bei der Wiederherstellung zurückgesetzt werden kann.

### 8.2.7 SQLite Timeline

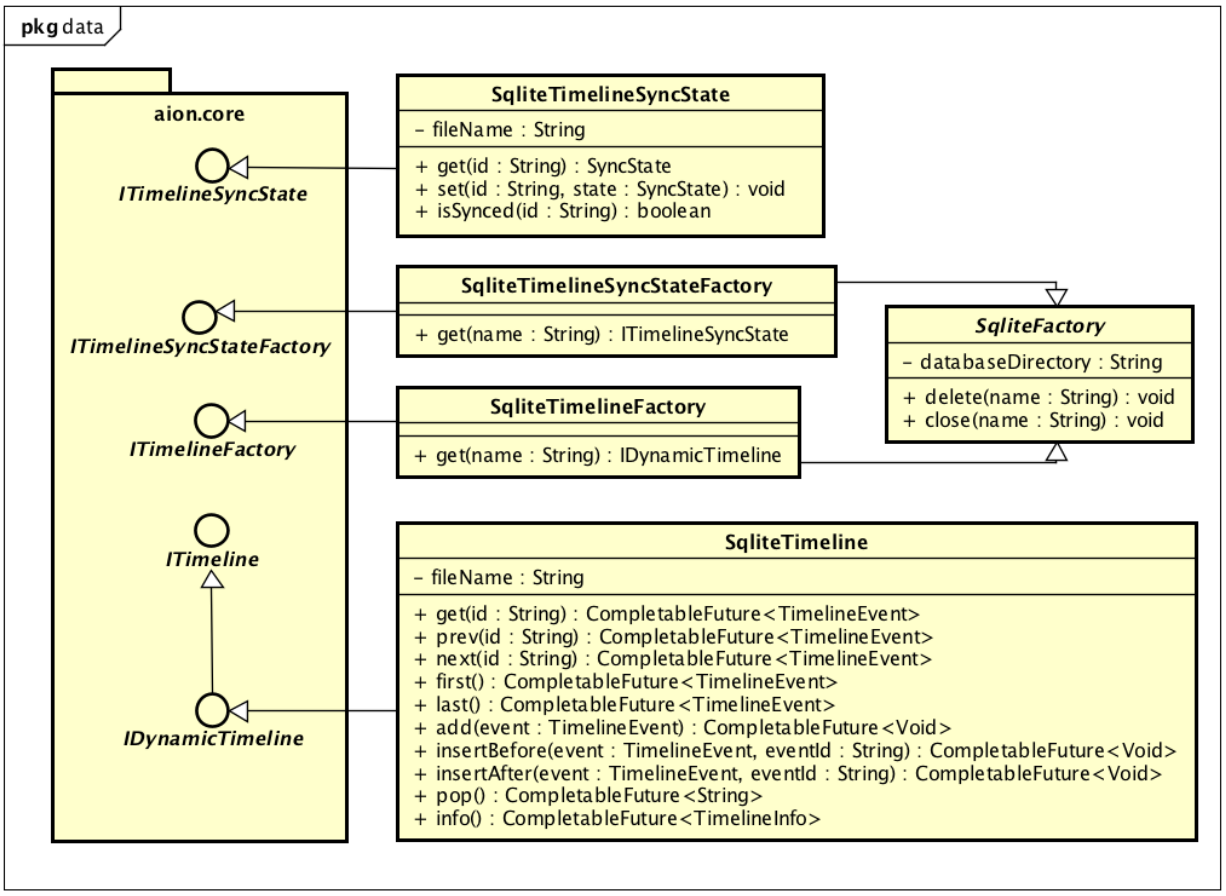


Abbildung 48: Data SQLite Timeline Klassendiagramm

**SqliteTimeline / SqliteTimelineFactory** Die SqliteTimeline-Klasse ist eine Timeline-Implementation, welche die Events in einer SQLite Datenbank speichert. Diese wird von der SqliteTimelineFactory erstellt und terminiert.

**SqliteTimelineSyncState / SqliteTimelineSyncStateFactory** Die SqliteTimelineSyncState-Klasse ist eine Implementation des TimlineSyncState, welche den Synchronisationsstatus in einer SQLite Datenbank speichert. Auch dieser wird von der SqliteTimelineSyncStateFactory verwaltet.

**SqliteFactory** Beide Factory-Klassen verwenden die SqliteFactory-Klasse, welche geteilte Funktionalitäten zum Verbindungsmanagement beinhalten.

Das gleiche wurde auch für die Memory Timeline implementiert. In der werden die Eventdaten und der Synchronisationsstaus im Memory gespeichert.

## 8.3 Auth (Aion)

Mit Hilfe des Auth Packages werden alle Events auf ihre Berechtigung überprüft. Darf ein Benutzer den Event nicht ausführen, bricht der Interceptor die Weiterverarbeitung ab.

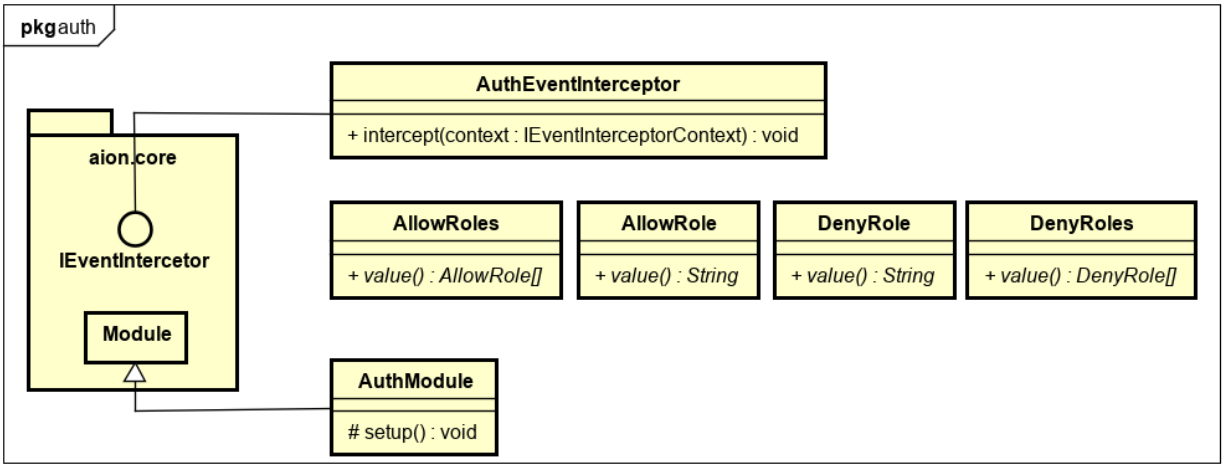


Abbildung 49: Auth Klassendiagramm

**AuthEventInterceptor** Dieser Interceptor prüft vor jeder Aggregation, ob der Benutzer eine der AllowRoles besitzt, bzw. eine der DenyRoles. Wurden keine AllowRoles definiert, dürfen alle Benutzer den Aggregator ausführen.

**AuthModule** Das Modul bindet den AuthEventInterceptor an die durchgeführten Events.

**Role-Annotation** Die Annotationen definieren, welche Rollen auf den Aggregatoren erlaubt sind. Die Rollen-Annotationen können mehrfach definiert werden.
Folgende Annotationen sind im Aion implementiert:

- AllowRole: Gibt an, welche Rollen erlaubt sind.
- DenyRole: Gibt an, welche Rollen nicht erlaubt sind.

## 8.4 Email (Aion)

Das Email Package wird für das aufbereiten und absenden von E-Mail Nachrichten auf dem Server verwendet. Bei der Implementation wurde berücksichtigt, dass es verschiedene E-Mail Transportprotokolle gibt. Darum wurde dieser Teil abstrahiert und austauschbar gemacht.

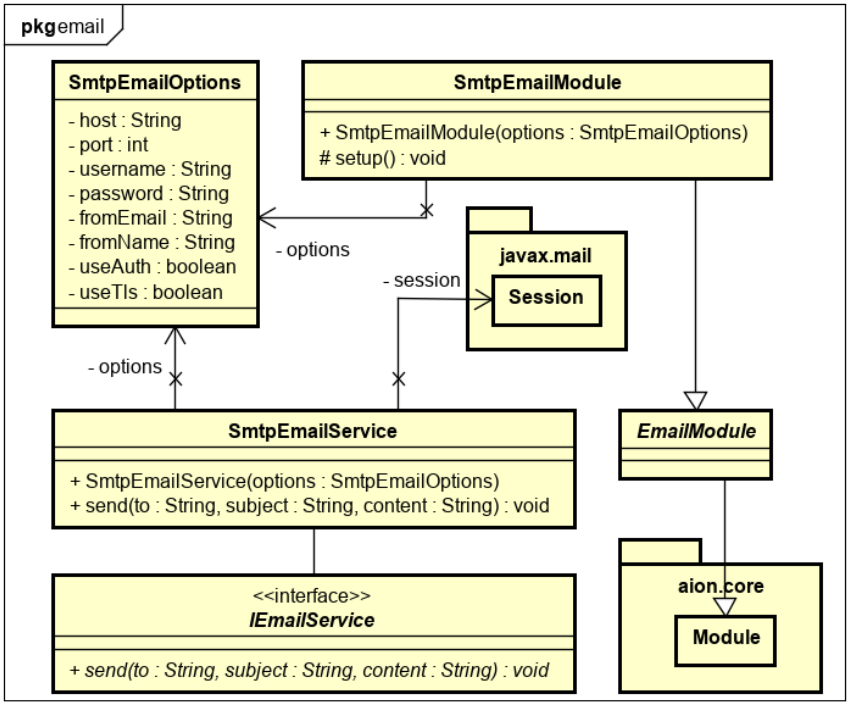


Abbildung 50: Email Klassendiagramm

Das Package ist in weitere Sub-Packages aufgeteilt. Pro E-Mailprotokoll wird ein eigenes Package mit seinen Klassen erstellt bzw. erweitert. Für unser Offliss-App erstellten wir zwei Packages: smtp und mock.

**EmailModule** Die Modulklasse erbt von der abstrakten Basisklasse Module, dadurch ist sie injectierbar und kann Events verarbeiten.

**IEmailService** Der IEmailService dient als Schnittstelle für alle Protokoll-Klassen. Die einzige Methode, die er anbietet, ist send, welche festlegt an wen was gesendet wird.

**SmtpEmailModule** Dieses Modul verbindet die Komponenten SmtpEmailOptions und SmtpEmailService und definiert dessen Ausführungsklassen.

**SmtpEmailOptions** In den SmtpEmailOptions werden die Konfigurationsdaten für SMTP verwaltet. Diese werden für den Verbindungsaufbau im SmtpEmailService verwendet.

**SmtpEmailService** Im Service wird die Verbindung aufgebaut, das E-Mail mit Inhalt befüllt und über den Transporter versendet.

## 8.5 Logging (Aion)

Das Logging-Modul ermöglicht es, die Resultate der Applikation, Event-Interceptoren und den Event-Aggregatoren in der Konsole auszugeben. Dieses Modul soll die Entwicklung und die Fehlersuche erleichtern.

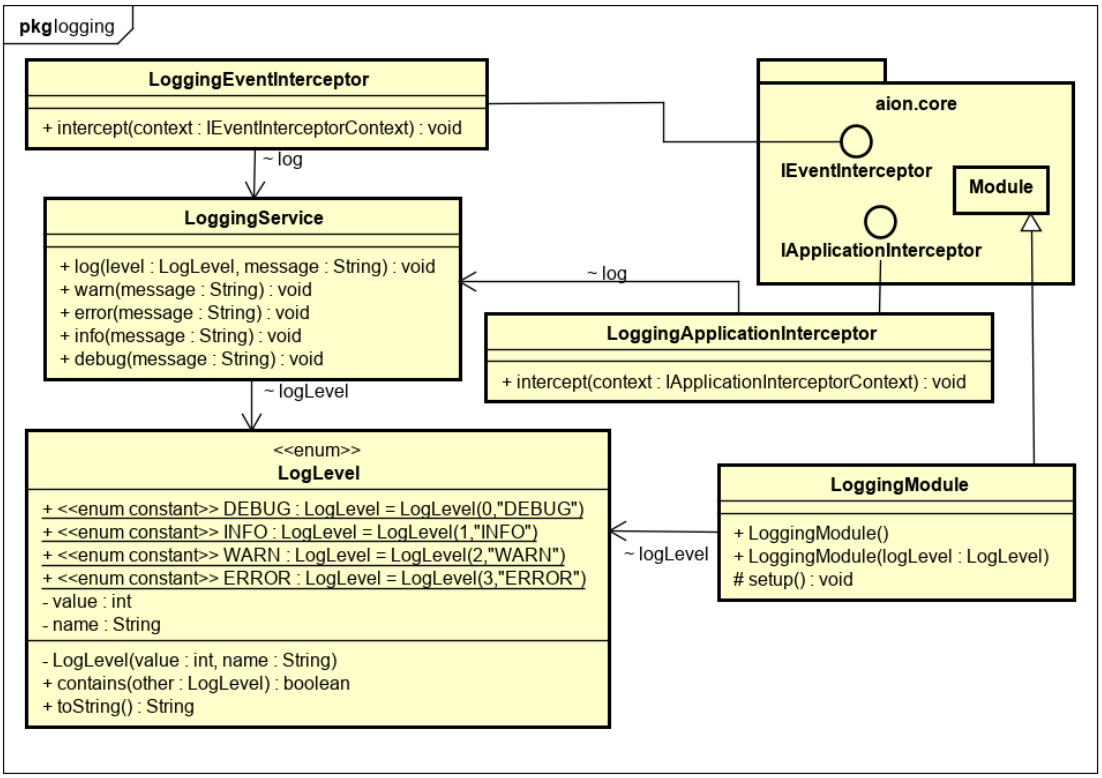


Abbildung 51: Logging Klassendiagramm

**LoggingModule** Das LoggingModule initialisiert die Interceptorenklassen mit den konkreten Implementationen.

**LoggingApplicationInterceptor** Der ApplicationInterceptor sorgt dafür, dass der Logger den Start bzw. Stopp der Applikation loggt und deren Fehler ausgibt.

**LoggingEventInterceptor** Der EventInterceptor loggt die eingehenden Events und deren Resultate in den Aggregatoren.

**LoggingService** Der Service schreibt die eingehenden Nachrichten auf die Konsole.

**LogLevel** Der LogLevel ist zur Unterscheidung der Log-Levels.
Folgende Log-Levels sind definiert:

- DEBUG: Feingranulare Informationsereignisse (nützlich, um die Applikation zu debuggen)
- INFO: Fortschrittsanzeige
- WARN: Potenziell schädliche Situationen
- ERROR: Fehlerereignisse

## 8.6 Android (Aion)

In diesem Package befindet sich Code, der in anderen App-Projekten wiederverwendet werden kann.

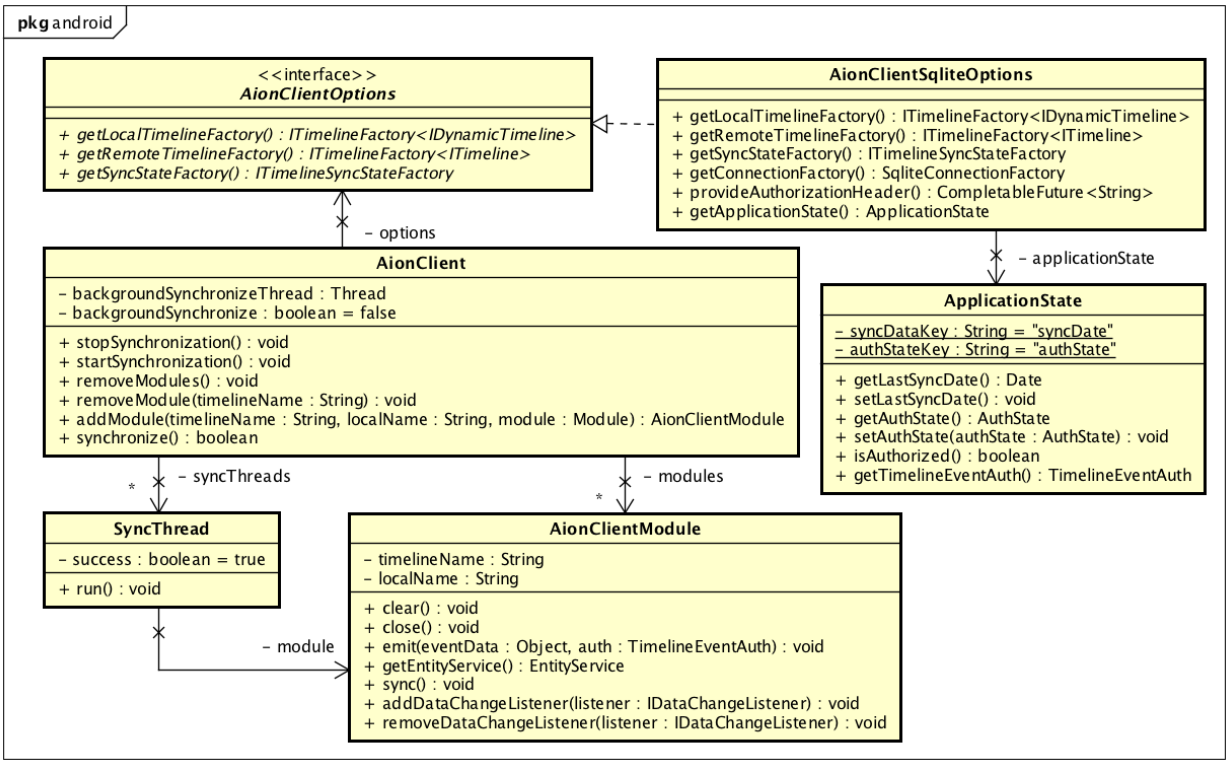


Abbildung 52: Android Klassendiagramm

**AionClient** Die AionClient-Klasse synchronisiert die einzelnen Timelines. Es können mehrere Module hinzugefügt oder entfernt werden, welche dadurch automatisch im Hintergrund synchronisieren und Daten lokal persistieren.

**AionClientModule** Dieses Modul beinhaltet einen Synchronisationsstatus und die lokale Timeline, welche zur Verarbeitung der Projection genutzt wird. Es synchronisiert die Timeline und erlaubt es, die Projection über den EntityService aus dem Data Projekt abzufragen. Zusätzlich können DataChangeListeners registriert werden, die darüber informieren, wann sich die Projection geändert hat.

**AionClientOptions** / **AionClientSqliteOptions** Die AionClientOptions-Schnittstelle definiert, welche TimelineFactories zur lokalen und entfernten Synchronisation verwendet werden sollen. Die AionClientSqliteOptions-Klasse persistiert die Daten in einer lokalen SQLite Datenbank, die wiederum mit einer anderen Timeline via HTTP synchronisiert. Dabei wird der ApplicationState verwendet, der sich um die Autorisierung der HTTP-Anfragen kümmert.

**ApplicationState** Der ApplicationState wird in der Android Applikation erzeugt und beinhaltet die Autorisierungsinformationen. Dazu wird das Datum der letzten erfolgreichen Synchronisation gespeichert. Alle Daten des ApplicationState werden in die SharedPreferences im Android abgespeichert.

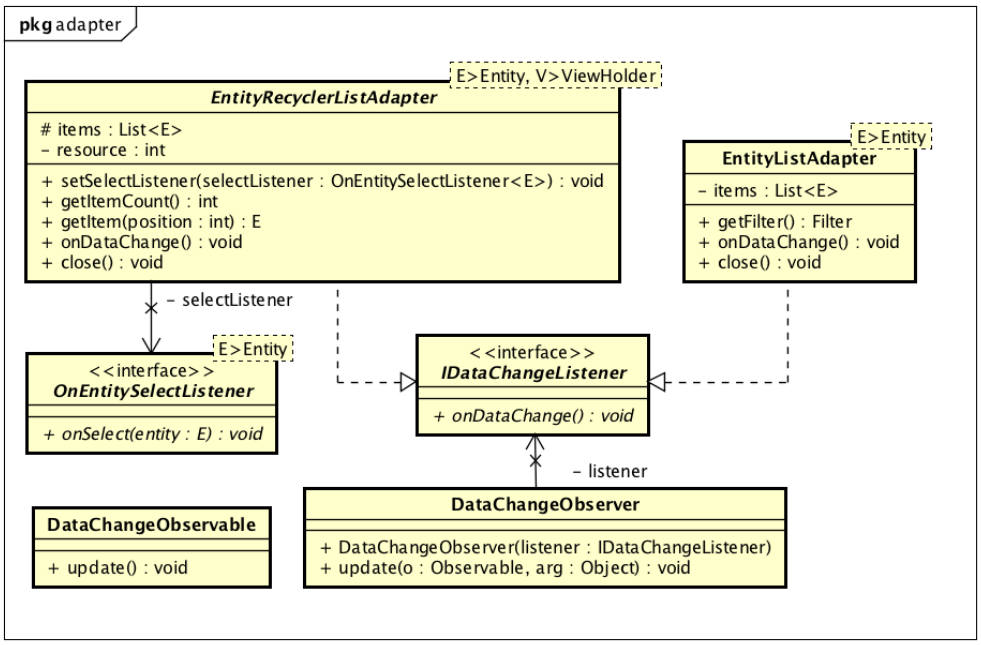


Abbildung 53: Android/Adapter und Android/Data Klassendiagramme

**EntityListAdapter** / **EntityRecylerListAdapter** Der EntityListAdapter und der EntityRecylerListAdapter erlauben es, mit geringem Aufwand, die Listeninformationen in der App darzustellen. Diese verwenden den EntityService, um auf die lokalen Daten des Gerätes zuzugreifen. Zusätzlich implementieren sie die IDataChangeListener-Schnittstelle und schreiben sich im AionClientModule ein, um immer die aktuellsten Daten darzustellen.

**OnEntitySelectListener** Mit der OnEntitySelectListener-Schnittstelle können Aktionen ausgelöst werden, sobald ein Eintrag in der Liste ausgewählt wird. Im Callback wird die angewählte Entität mitgeliefert.

**DataChangeObservable** / **DataChangeObserver** Durch den Observer und das Observable können die ListAdapter immer die aktuellsten Daten darstellen. Sobald das AionClientModule synchronisiert und Daten ändern, werden die Listen aktualisiert.

## 8.7 Http-Client (Aion)

Das Http-Client Projekt spricht eine Timeline via HTTP an. Es ist stark gekoppelt mit dem Http-Server Projekt. Die Timelinedaten auf einem entfernten Server werden gleichbehandelt wie die lokal verfügbaren.

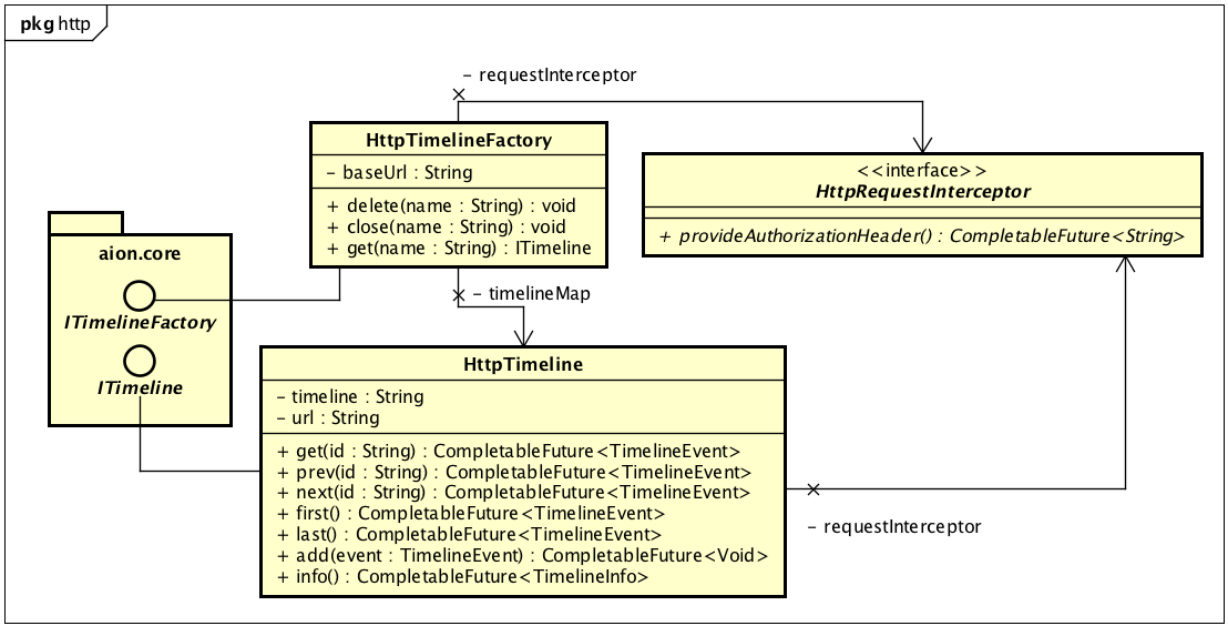


Abbildung 54: Http-Client Klassendiagramm

**HttpTimeline / HttpTimelineFactory** Die HttpTimeline-Klasse implementiert die ITimeline-Schnittstelle. Die Methoden lösen einzelne HTTP-Aufrufe aus und leiten die Informationen weiter.

**HttpRequestInterceptor** Der HttpRequestInterceptor kümmert sich um die Autorisierung des Requests. Damit ist es möglich, die Implementation frei von den Autorisierungsdetails zu halten.

## 8.8 Http-Server (Aion)

Der Http-Server ist ein eigenständiger Server, welcher mehrere Timelines via HTTP anbietet. Die dahinter liegenden Timelines können über eine Konfigurationsdatei konfiguriert werden.

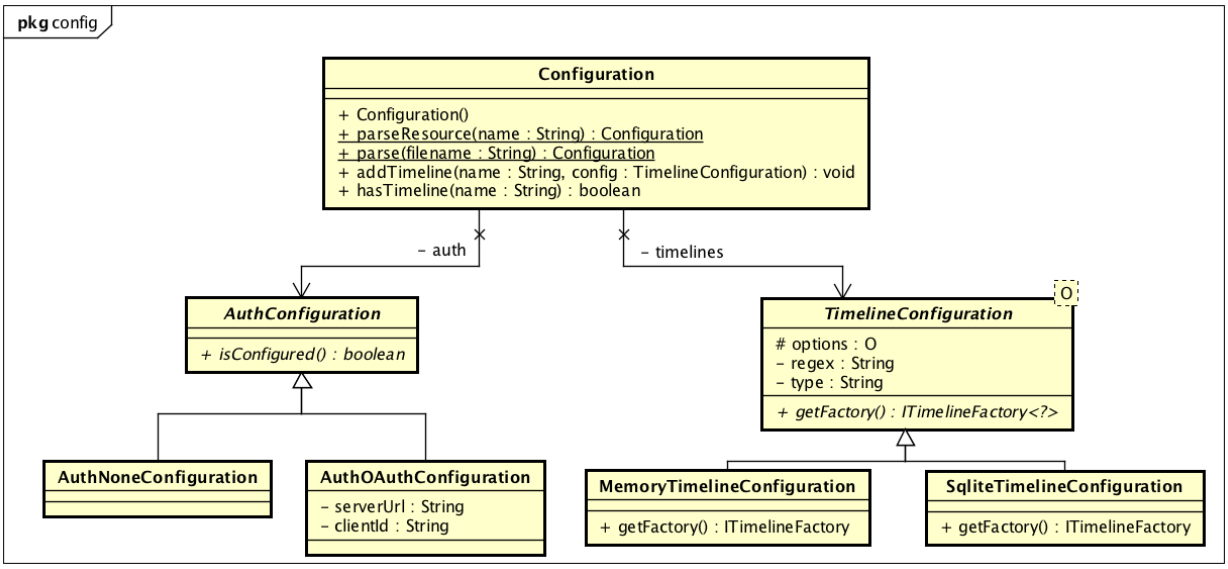


Abbildung 55: Http-Server Configuration Klassendiagramm

**Configuration** Die Configuration-Klasse dient zur Serialisierung der Konfigurationsdatei und wird vom Http-Controller bei HTTP-Aufrufen verwendet.

**AuthConfiguration / AuthNoneConfiguration / AuthOAuthConfiguration** Die abstrakte AuthConfiguration-Klasse wird durch die beiden AuthNoneConfiguration und AuthOAuthConfiguration-Klassen implementiert. Beim Auslesen der Konfigurationsdatei wird aufgrund des Typs unterschieden, welche Klasse verwendet werden soll. Durch die Verwendung der AuthNoneConfiguration-Klasse wird keine Autorisierung der HTTP-Anfragen durchgeführt. Bei der Verwendung von AuthOAuthConfiguration wird ein JWT Token bei der HTTP-Anfrage erwartet und geprüft.

**TimelineConfiguration / MemoryTimelineConfiguration / SqliteTimelineConfiguration** Die TimelineConfiguration wird von den beiden Klassen MemoryTimelineConfiguration und SqliteTimelineConfiguration erweitert. Diese stellen eine Timeline dar, welche via HTTP abgefragt werden kann. Die verschiedenen Implementationen steuern die darunterliegende Technologie und haben unterschiedliche Konfigurationsoptionen.

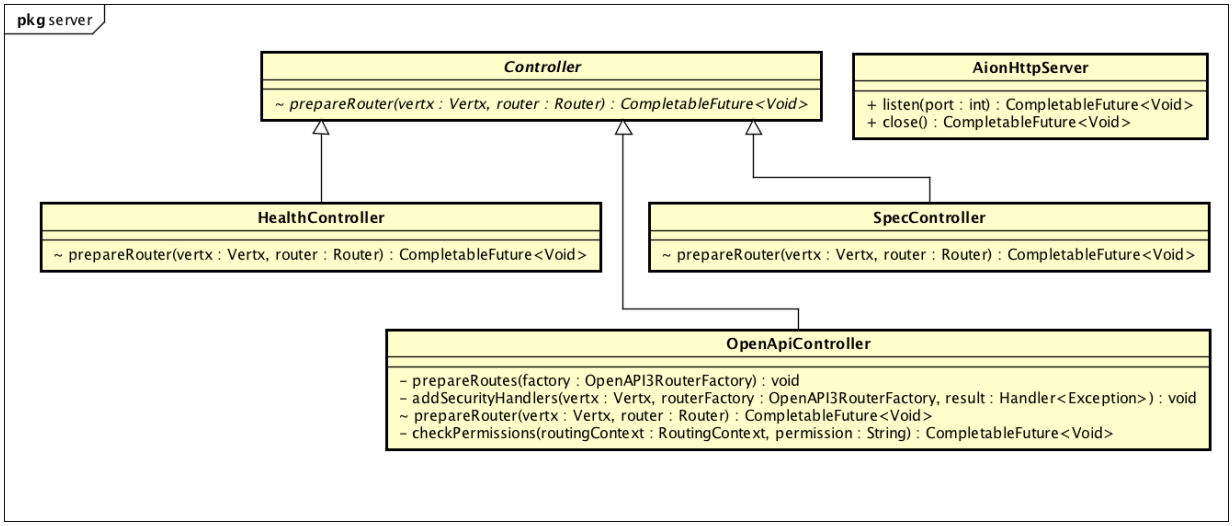


Abbildung 56: Http-Server Klassendiagramm

**Controller** Die Controller-Klasse ist die Basisklasse für alle HTTP-Ressourcen. Darin befinden sich allgemeine Funktionen, welche sich beispielsweise um die Rückgabe von Jsondaten kümmert.

**HealthController** Der HealthController fügt eine Route hinzu, welche immer mit Status 200 terminiert. Diese verwenden wir in Kubernetes, um festzustellen, ob der Server bereit ist, um HTTP-Anfragen entgegenzunehmen.

**SpecController** Der SpecController fügt eine Route hinzu, welche die OpenAPI Spezifikation returniert. Diese wird vom Swagger UI verwendet und kann zur Client Generierung genutzt werden.

**OpenApiController** Der OpenApiController implementiert alle Routen, welche sich in der OpenAPI Spezifikation befinden. Diese ermöglichen es, die Timelines anzusprechen und abzufragen. Aufgrund der Konfiguration werden die Timeline Routen aufgebaut und die angegebene Autorisierung angewendet.

**AionHttpServer** Der AionHttpServer verwendet alle Controller und startet einen HTTP-Server auf dem angegebenen Port.

## 8.9 App (Offliss)

In diesem Kapitel werden die wichtigsten Klassen des Offliss Apps erläutert.

### 8.9.1 Domain Model (Offliss)

Die untenstehende Grafik zeigt das Domain Model der Projection. Dieses wird über die erstellten Events generiert und dient zur Abfrage der Daten.

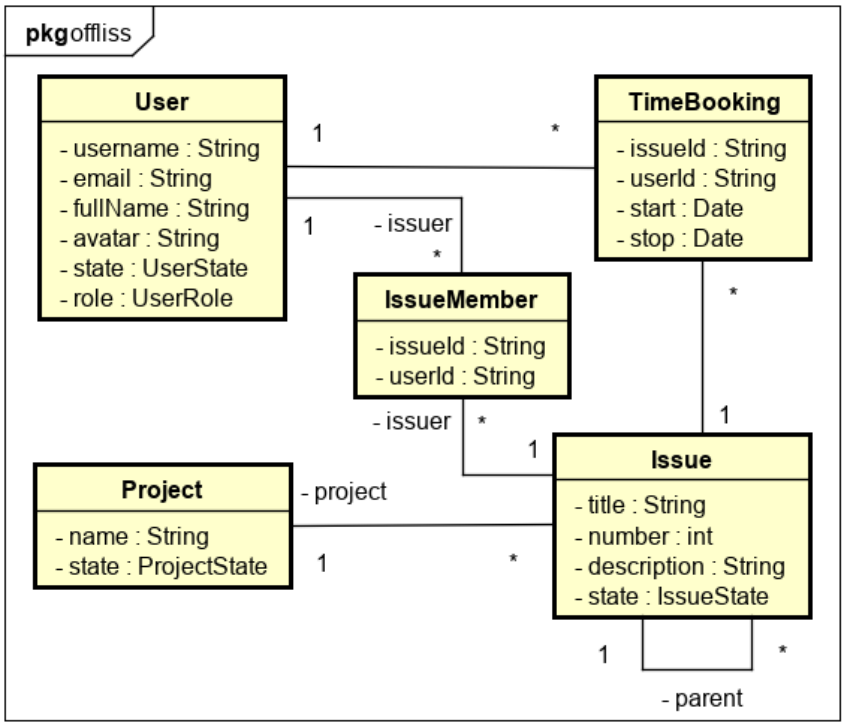


Abbildung 57: Domain Modell

Ein Issue gehört immer zu einem Projekt und kann mehreren Usern zugewiesen werden. Der User kann seine TimeBookings jeweils einem Issue zuordnen.

## 8.10 Common (Offliss)

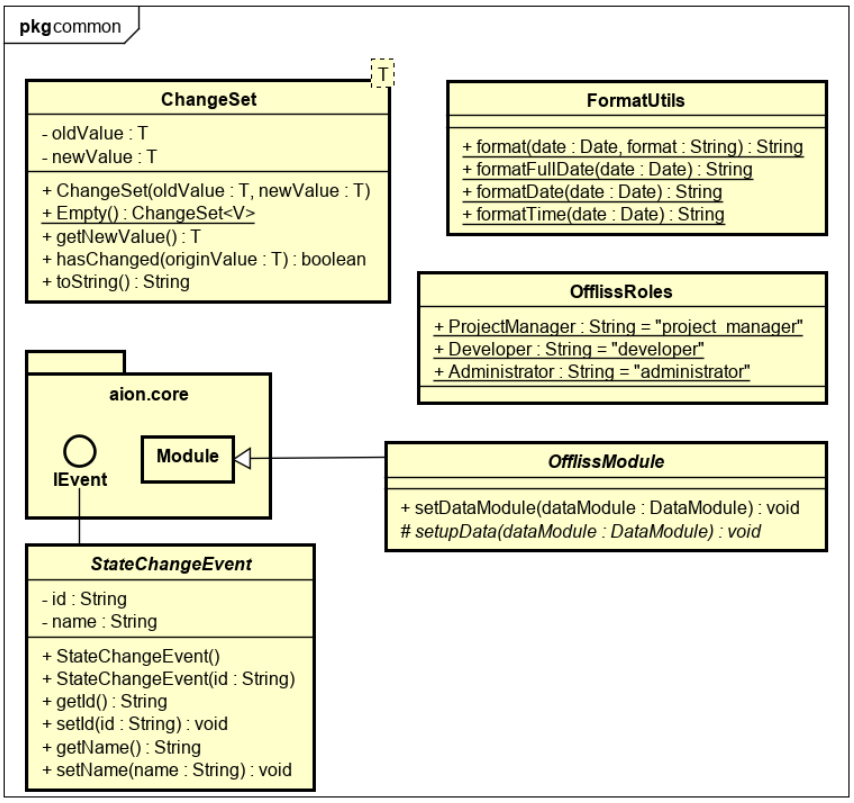


Abbildung 58: Common Klassendiagramm

**OfflissModule** Dieses Modul bildet die Basis für die Offliss-spezifischen Module, wie zum Beispiel UserModule. Dadurch werden bei allen Aggregatoren die Autorisierung geprüft, das Log beschrieben und sie bekommen Zugriff auf die Datenbank.

**OfflissRoles** Es gibt drei Rollen im Offliss: Projekt Manager, Entwickler und Administrator. Die Rollen werden hier, mit den Rollennamen im Keycloak, als globale statische Variablen definiert.

**FormatUtils** Das FormatUtils dient dazu, Datentypen in ein lesbares, verständliches Format umzuwandeln. Im Offliss sind die Umwandlungen für Datumswerte implementiert.

**StateChangeEvent** Der StateChangeEvent ist ein Offliss-spezifischer Event, der für Statusänderungen gebraucht werden kann. Alle Events, die den Status ändern, wie beispielsweise ActivateUser, erben von diesem Event.

**ChangeSet** Um nachvollziehen zu können, welche Attribute sich an den Objekten geändert haben, wird das ChangeSet verwendet. Es speichert sowohl den alten als auch den neuen Wert ab und vergleicht diese in der hasChanged-Methode. Es wird ausserdem überprüft, ob in der Zwischenzeit jemand anderes denselben Wert bereits bearbeitet hat.

### 8.10.1 User

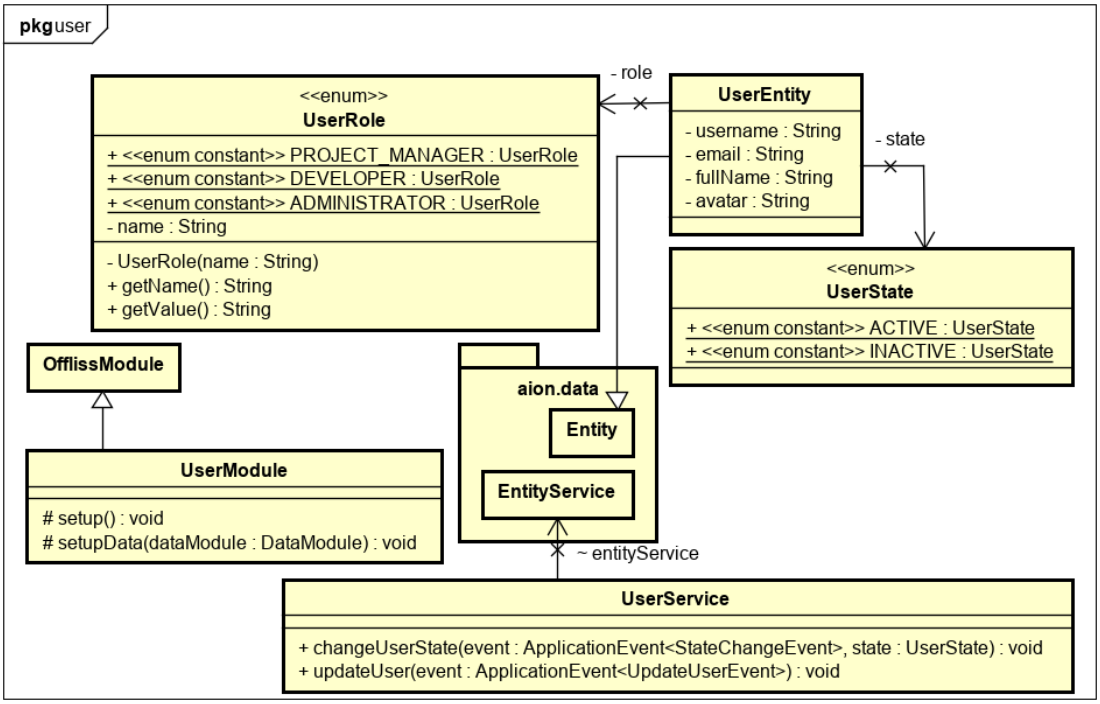


Abbildung 59: User Klassendiagramm

**UserModule** Das UserModule verwaltet alle Event-Aggregatoren für die Benutzer-Entität und initialisiert den UserService.

**UserEntity** Das User-Objekt mit all seinen Attributen wird im UserEntity gespeichert.

**UserService** Der UserService über nimmt zwei Aufgaben. Einerseits ändert er den Status des Benutzers auf ACTIVE bzw. INACTIVE und andererseits werden die Attribute des Benutzers aktualisiert.

**UserState** Im Enum UserState werden die Status des Users gespeichert.

**UserRole** Die Rollen, die ein Benutzer im Offliss annehmen kann, werden im Enum UserRole festgehalten. Die Konstanten entsprechen den Rollen im Keycloak. Im App wird das Attribut Name für das Anzeigen des Rollennamens des Benutzers verwendet.

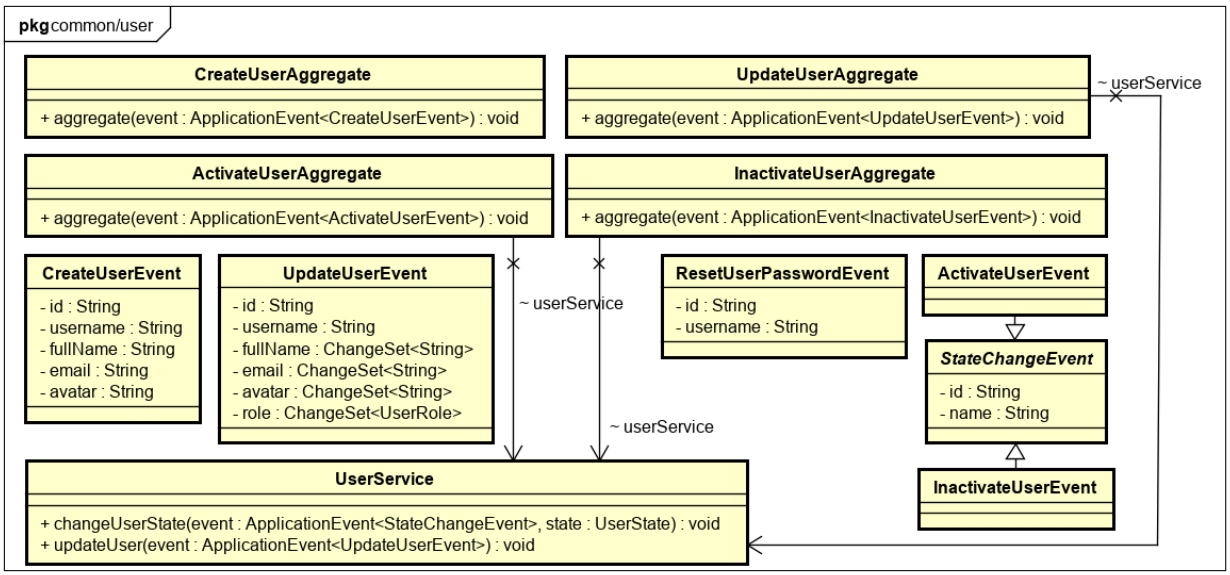


Abbildung 60: User Aggregatoren und Event Klassendiagramme

Alle User-spezifischen Aggregatoren können nur von der Offliss-Rolle Administrator durchgeführt werden. Allen anderen Rollen ist die Ausführung nicht erlaubt.

**CreateUserAggregate** Dieser Aggregator nimmt einen CreateUserEvent entgegen und erstellt einen neuen aktiven Benutzer. Während des Speichervorgangs wird überprüft, ob der Username und die E-Mailadresse bereits vergeben wurden. Dann wird das Einfügen gegebenenfalls abgebrochen.

**UpdateUserAggregate** Das UpdateUserAggregate speichert die geänderten Daten des Benutzers nur, wenn die Attribute geändert haben und die alten Werte mit den zurzeit in der Datenbank vorhandenen übereinstimmen. Auch hier wird während des Speichervorgangs geprüft, ob der Username und die E-Mailadresse eindeutig sind.

**ActivateUserAggregate** Das ActivateUserAggregate delegiert die Arbeit an den UserService. In diesem wird der Status auf ACTIVE geändert, egal welcher Status aktuell gesetzt ist.

**InactivateUserAggregate** Auch das Inaktivieren setzt den Status INACTIVE über den UserService auf dem veränderten Benutzer, unabhängig des derzeitigen Status.

**CreateUserEvent** Dieser Event enthält alle Attribute der UserEntity, bis auf den Status. Ein neuer Benutzer wird immer aktiv angelegt.

**UpdateUserEvent** Der UpdateUserEvent enthält alle Attribute der UserEntity als ChangeSets mit den alten und neuen Werten. Die ID enthält die ID des Benutzers, welcher aktualisiert werden soll.

**ActivateUserEvent** Dieser Event erbt vom Offliss-spezifischen StateChangeEvent und beinhaltet selbst keine weiteren Attribute, nebst der geerbten ID und dem Namen des Objekts.

**InactivateUserEvent** Dasselbe wie beim ActivateUserEvent gilt für den InactivateUserEvent.

**ResetUserPasswordEvent** Der ResetUserPasswordEvent wird immer ausgelöst, wenn der Administrator das Passwort für einen Benutzer ändern lässt. Der Aggregator dieses Events befindet sich im Server-Package und wird später beschrieben.

### 8.10.2 Project

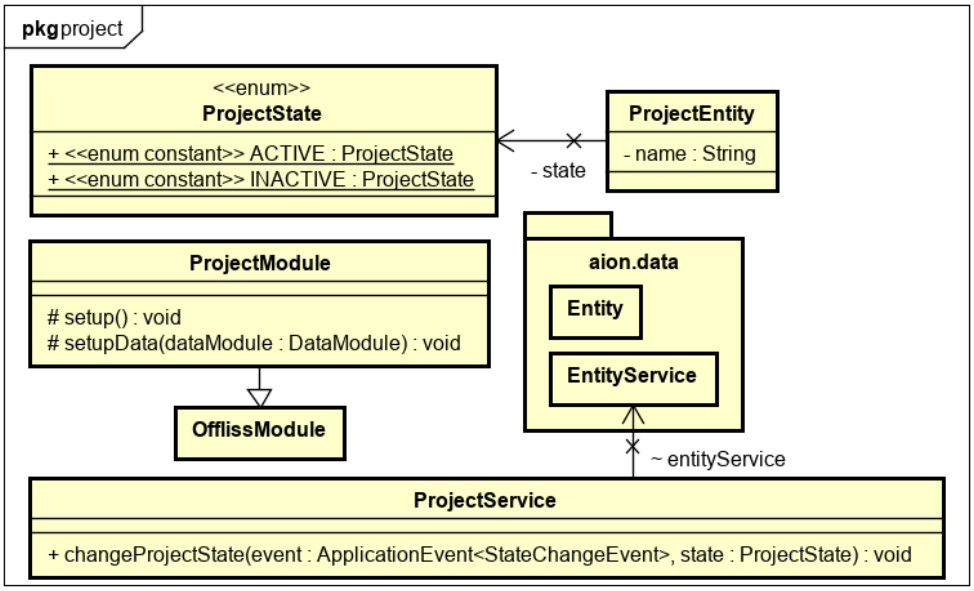


Abbildung 61: Project Klassendiagramm

**ProjectModule** Das ProjectModule verwaltet alle Event-Aggregatoren für die Projekt-Entität und initialisiert den ProjectService.

**ProjectEntity** Das Projekt-Objekt mit all seinen Attributen wird im ProjectEntity gespeichert.

**ProjectService** Im ProjectService wird die Statusaktualisierung des Projekts durchgeführt. Sowohl der Open- wie auch der CloseProjectEvent werden durch diesen verarbeitet.

**ProjectState** Im Enum ProjectState werden die Status des Projekts gespeichert.

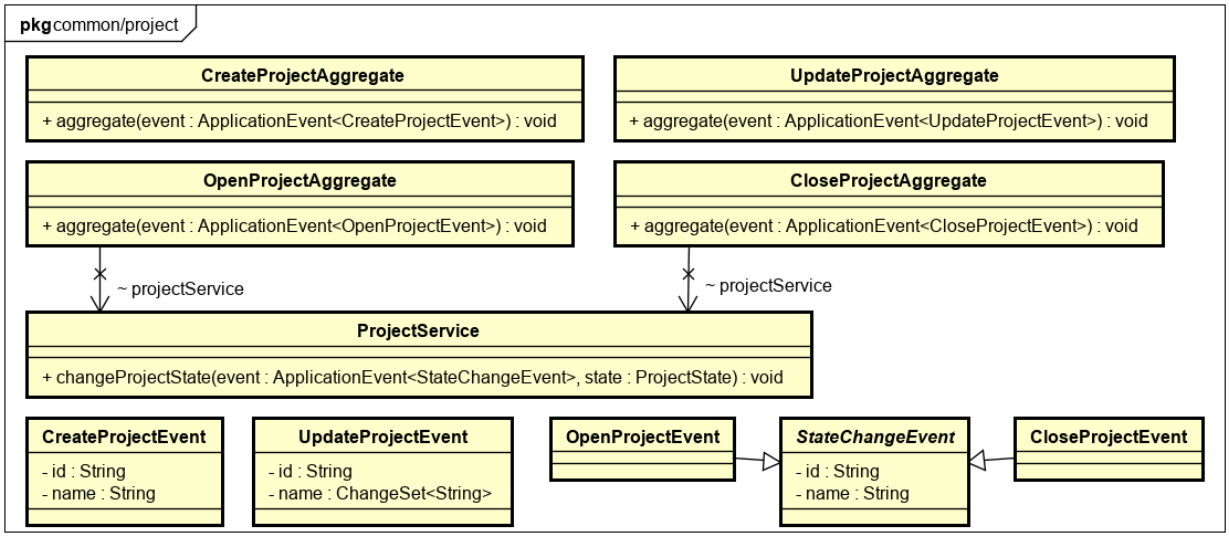


Abbildung 62: Project Aggregatoren und Event Klassendiagramme

Alle Projekt-spezifischen Aggregatoren können nur von der Offliss-Rolle Projekt Manager durchgeführt werden. Allen anderen Rollen ist die Ausführung nicht erlaubt.

**CreateProjectAggregate** Dieser Aggregator nimmt einen CreateProjectEvent entgegen, entnimmt dessen Namen und erstellt ein neues aktives Projekt. Während des Speichervorgangs wird überprüft, ob der Name bereits vergeben wurde. Dann wird das Einfügen in die Datenbank entsprechend abgebrochen.

**UpdateProjectAggregate** Das UpdateProjectAggregate speichert den geänderten Namen des Projekts nur, wenn der alte Projektname, mit dem zurzeit in der Datenbank vorhandenen übereinstimmt. Auch hier wird während des Speichervorgangs geprüft, ob der neue Name bereits vergeben wurde und der Vorgang, wenn nötig abgebrochen.

**OpenProjectAggregate** Das OpenProjectAggregate delegiert die Arbeit an den ProjectService. In diesem wird der Status auf ACTIVE gesetzt. Dabei ist es unerheblich, ob der Status des Projekts bereits auf ACTIVE steht oder nicht.

**CloseProjectAggregate** Auch das CloseProjectAgregate setzt den Status INACTIVE über den ProjectService auf dem veränderten Projekt. Wie beim OpenProjectAggregate wird auch hier nicht geprüft, ob der Status bereits INACTIVE ist.

**CreateProjectEvent** Dieser Event enthält die ID und den Namen des anzulegenden Projekts.

**UpdateProjectEvent** Der UpdateProjectEvent enthält die ID des zu verändernden Projekts und das ChangeSet mit dem alten Projektnamen und dem neuen.

**OpenProjectEvent** Dieser Event erbt vom Offliss-spezifischen StateChangeEvent und beinhaltet selbst keine weiteren Attribute, nebst der geerbten ID und dem Namen des Objekts.

**CloseProjectEvent** Dasselbe wie beim OpenProjectEvent gilt für den CloseProjectEvent.

### 8.10.3 Issue

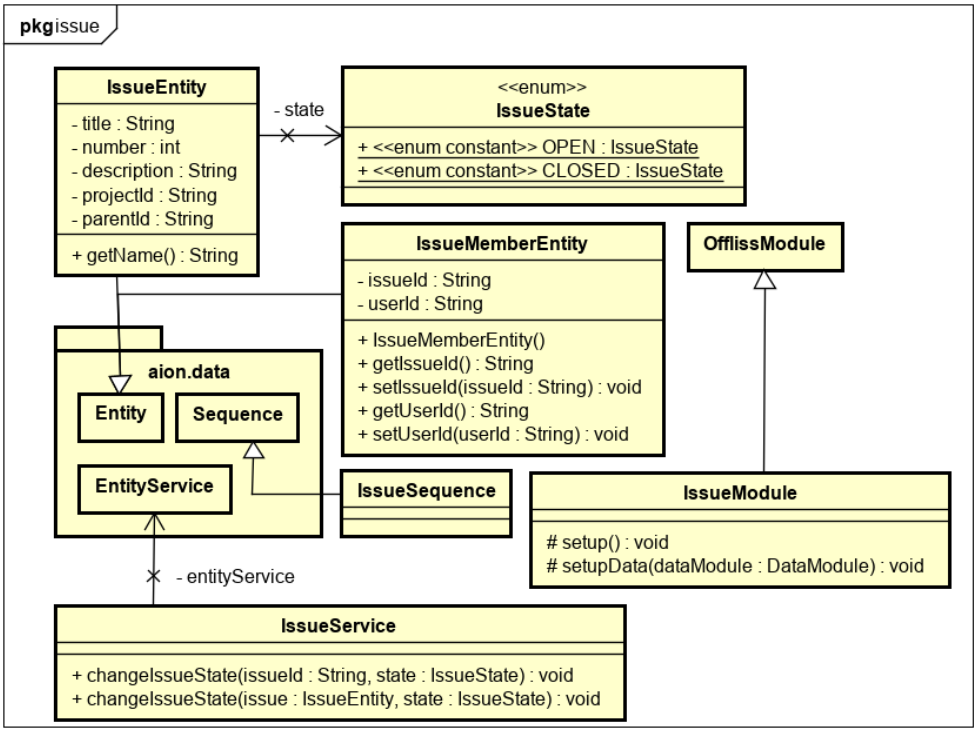


Abbildung 63: Issue Klassendiagramm

**IssueModule** Das IssueModule verwaltet alle Event-Aggregatoren für die Arbeitspaket-Entität und initialisiert den IssueService.

**IssueEntity** Das Arbeitspaket-Objekt mit all seinen Attributen wird im IssueEntity gespeichert.

**IssueMemberEntity** Die Zuweisung von Benutzer zu Arbeitspaket wird in dieser Entität gespeichert.

**IssueService** Im IssueService wird die Statusaktualisierung des Arbeitspakets durchgeführt. Sowohl der Open- wie auch der Close- und der CloseAllIssueEvent werden durch diesen verarbeitet.

**IssueState** Im Enum ProjectState werden die Status des Arbeitspakets gespeichert.

**IssueSequence** Die IssueSequence-Klasse macht nichts Eigenes. Sie erbt lediglich von Sequence im data Package.

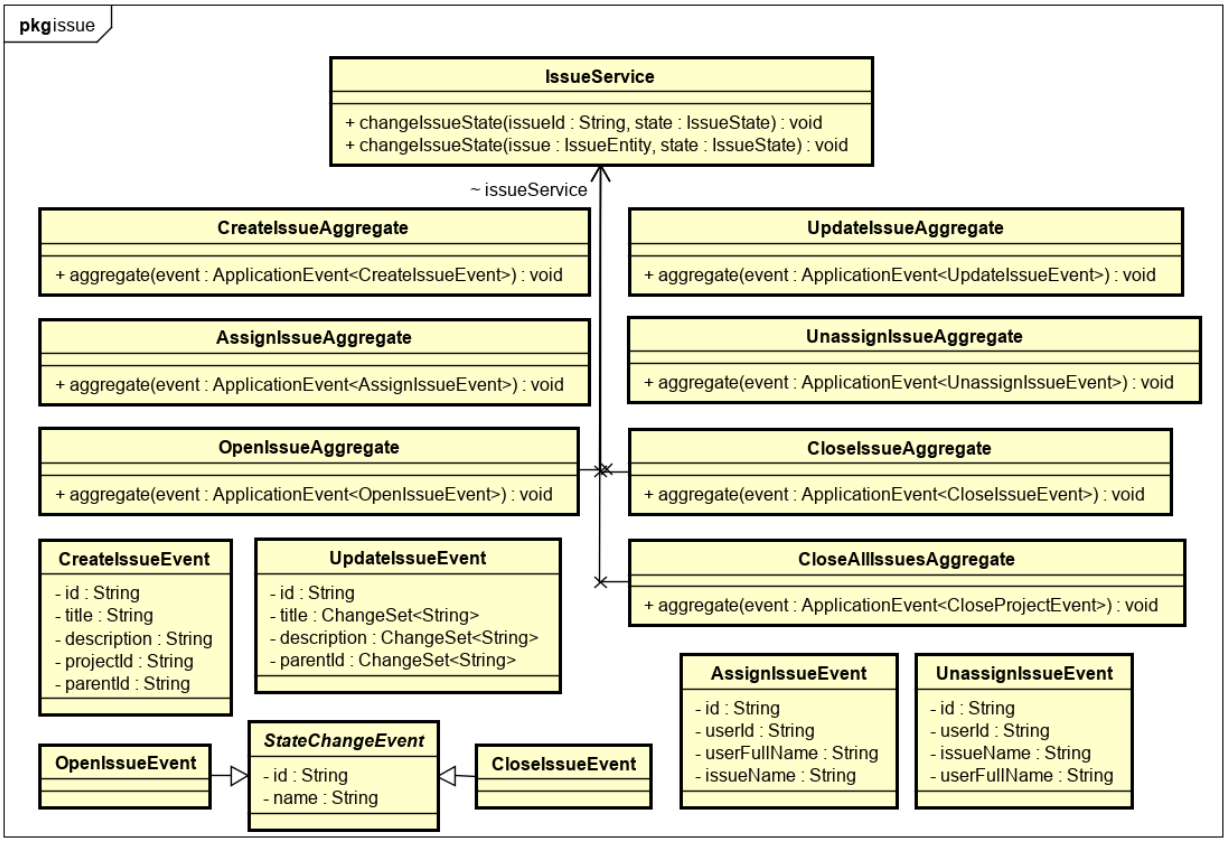


Abbildung 64: Issue Aggregatoren und Event Klassendiagramme

Alle Issue-spezifischen Aggregatoren können nur von den Offliss-Rollen Projekt Manager und Entwickler durchgeführt werden. Der Benutzer muss mindestens eine der beiden Rollen besitzen. Allen anderen Rollen ist die Ausführung nicht erlaubt.

**CreateIssueAggregate** Dieser Aggregator nimmt einen CreateIssueEvent entgegen, entnimmt dessen Attribute und erstellt ein neues aktives Arbeitspaket. Während des Speichervorgangs wird überprüft, ob der Titel mit der Arbeitspaketnummer eindeutig ist. Sonst wird das Einfügen in die Datenbank entsprechend abgebrochen.

**UpdateIssueAggregate** Das UpdateIssueAggregate speichert die geänderten Attribute des Arbeitspakets nur, wenn die Attribute geändert haben und die alten Werte mit den zurzeit in der Datenbank vorhandenen übereinstimmen. Auch hier wird während des Speichervorgangs geprüft, ob der Titel zusammen mit der Nummer bereits vergeben wurde und der Vorgang, wenn nötig abgebrochen.

**OpenIssueAggregate** Das OpenIssueAggregate delegiert die Arbeit an den IssueService. In diesem wird der Status auf OPEN gesetzt. Dabei ist es unerheblich, ob der Status des Arbeitspakets bereits auf OPEN steht oder nicht.

**CloseIssueAggregate** Auch das CloseProjectAgregate setzt den Status CLOSE über den IssueService auf dem veränderten Arbeitspaket. Wie beim OpenIssueAggregate wird auch hier nicht geprüft, ob der Status bereits CLOSE ist oder nicht.

**CloseAllIssue** Wird ein übergeordnetes Arbeitspaket geschlossen, gilt das auch für alle darunterliegenden Arbeitspakete. Dazu wird für alle dieselbe Methode wie bei CloseIssueAggregate aufgerufen.

**AssignIssue** Bei der Zuweisung eines Benutzers zu einem Arbeitspaket wird zuerst geprüft, ob dieser nicht bereits zugefügt wurde. Ist dies nicht der Fall wird eine neue IssueMemberEntity, mit der ID des Arbeitspakets und die des Benutzers, erstellt und gespeichert.

**UnassignIssue** Bei einem UnassignEvent wird die erstellte IssueMemberEntity, mit der entsprechenden ID des Arbeitspakets und des Benutzers, gelöscht.

**CreateIssueEvent** Dieser Event enthält alle Attribute der IssueEntity, bis auf den Status. Der Status eines neuen Arbeitspakets ist immer OPEN.

**UpdateIssueEvent** Der UpdateIssueEvent enthält die ID des zu verändernden Arbeitspakets, den Titel, die Beschreibung und die ID des übergeordneten Arbeitspakets als ChangeSets mit den alten und neuen Werten.

**OpenIssueEvent** Dieser Event erbt vom Offliss-spezifischen StateChangeEvent und beinhaltet selbst keine weiteren Attribute, nebst der geerbten ID und dem Namen des Objekts.

**CloseIssueEvent** Dasselbe wie beim OpenIssueEvent gilt für den CloseIssueEvent.

**AssignIssueEvent** Im AssignIssueEvent werden die IDs von dem betroffenen Issue und dem betroffenen User mitgegeben.

**UnassignIssueEvent** Der UnassignIssueEvent enthält dieselben Informationen wie der AssignIssueEvent.

### 8.10.4 TimeBooking

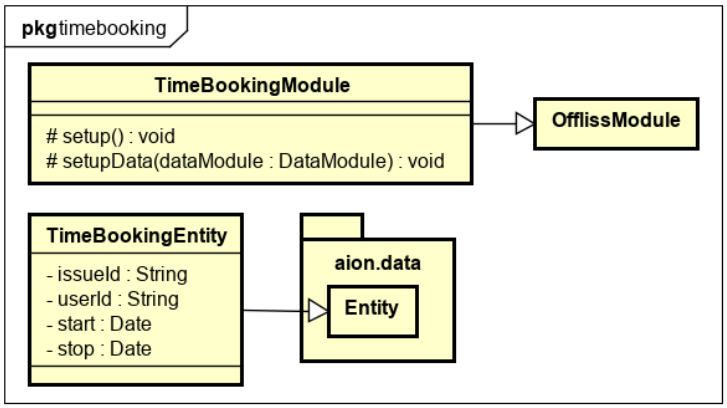


Abbildung 65: TimeBooking Klassendiagramm

**TimeBookingModule** Wie die anderen Module verwaltet auch dieses Modul alle Event-Aggregatoren für die Zeiterfassung-Entität.

**TimeBookingEntity** Die Zeiterfassung mit all ihren Attributen wird im TimeBookingEntity gespeichert.

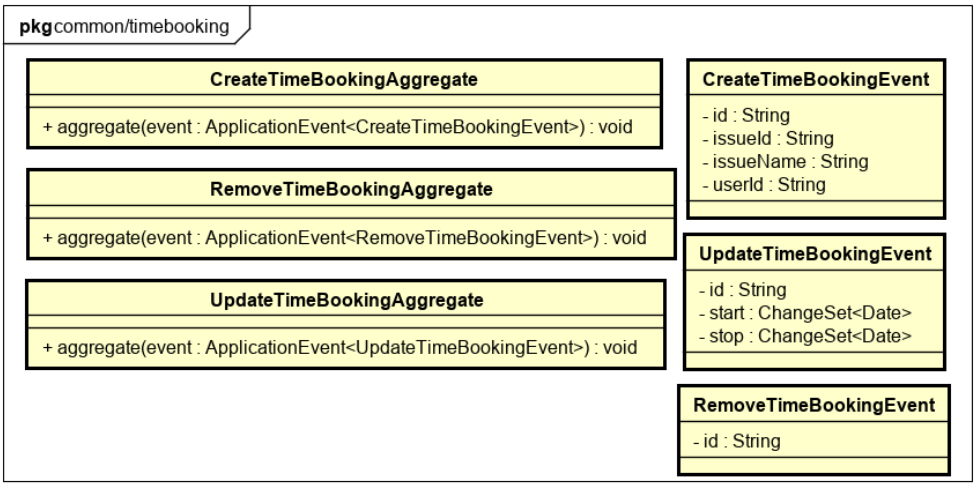


Abbildung 66: TimeBooking Aggregatoren und Event Klassendiagramme

Alle TimeBooking-spezifischen Aggregatoren können nur von der Offliss-Rolle Entwickler durchgeführt werden. Allen anderen Rollen ist die Ausführung nicht erlaubt. Ausserdem kann jeder Benutzer nur seine eigenen Zeiterfassungen sehen und bearbeiten.

**CreateTimeBookingAggregate** Dieses Aggregate nimmt einen CreateTimeBookingEvent entgegen, entnimmt dessen Attribute und erstellt eine neue Zeiterfassung.

**UpdateTimeBookingAggregate** Das UpdateTimeBookingAggregate speichert die geänderten Daten der Zeitbuchung nur, wenn die Attribute geändert haben.

**RemoveTimeBookingAggregate** Durch den RemoveTimeBookingEvent wird die übergebene Zeiterfassung aus dem System gelöscht.

**CreateTimeBookingEvent** Dieser Event enthält alle Attribute der Zeiterfassung.

**UpdateTimeBookingEvent** Aktualisieren kann man nach der Erfassung nur noch die Zeitangaben. Deshalb fehlt im UpdateTimeBookingEvent die Information über das Arbeitspaket.

**RemoveTimeBookingEvent** In diesem Event wird lediglich die ID, der zu entfernenden Zeiterfassung übergeben.

### 8.10.5 Activities und Fragments

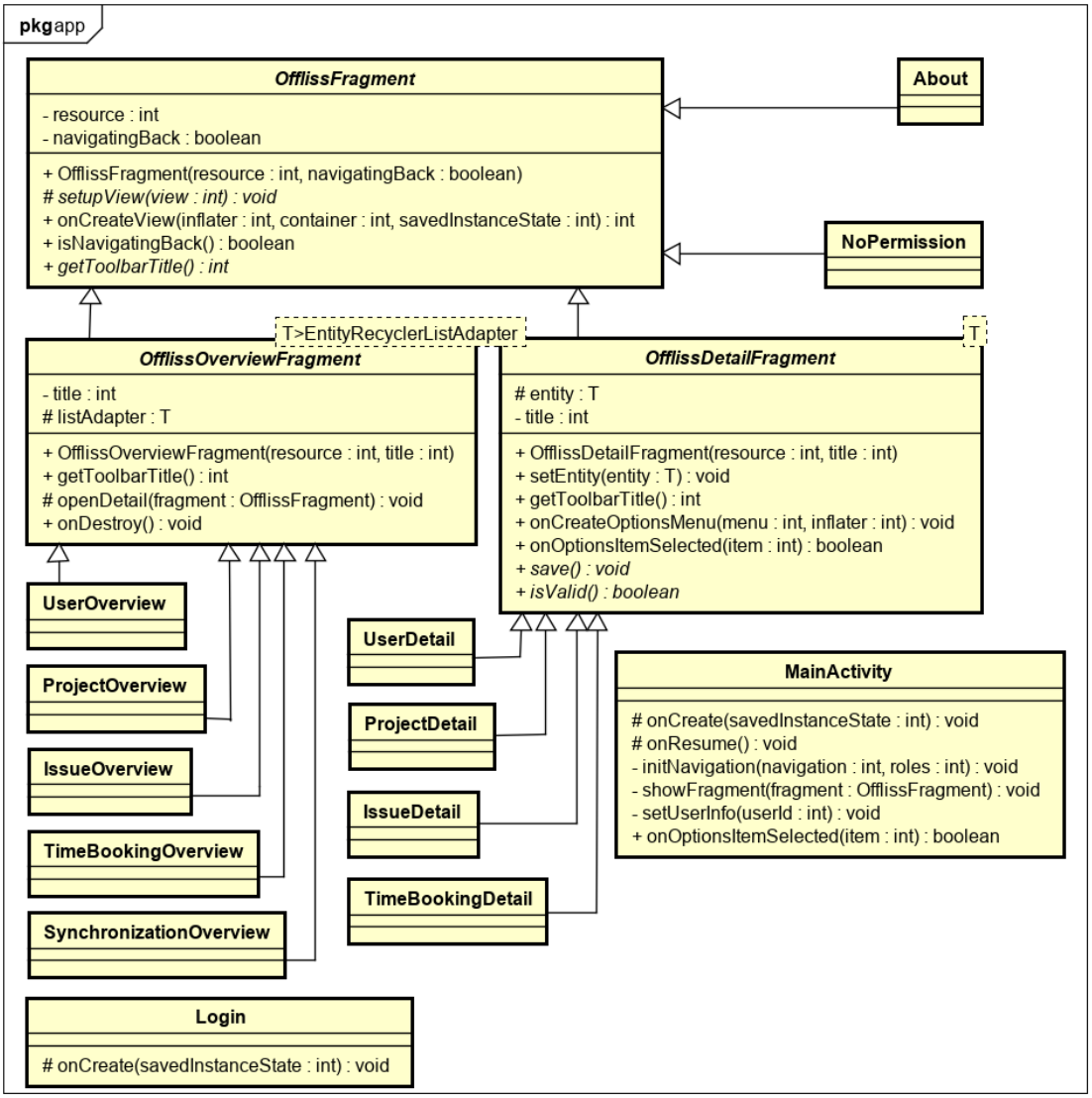


Abbildung 67: Activity und Fragment Klassendiagramme

**MainActivity** Beim App-Start wird die MainActivity aufgerufen. Diese sorgt dafür, dass die Navigation aufgebaut und mit den der Benutzer-Rolle entsprechenden Navigationslinks ausgestattet ist. Die MainActivity ist immer präsent und ersetzt lediglich ihren Inhalt durch die einzelnen Seiten (Fragments). Bei jedem Wechsel wird erneut geprüft, ob der Benutzer diese Seite noch immer sehen darf. Es könnte nämlich sein, dass in der Zwischenzeit seine Benutzer-Rolle geändert hat. Ausserdem prüft sie den Autorisierungsstatus und leitet den Benutzer, wenn nötig zum Login weiter.

**Login** Das Login wurde als Activity implementiert, damit man hin und her wechseln kann zwischen Main und Login. Das Login prüft den Benutzernamen und das Passwort. Zusätzlich wird überprüft, ob der Benutzer beim Login mit dem Internet verbunden ist. Konnte das Login nicht durchgeführt werden können wir dem Benutzer somit eine genaue Fehlermeldung bieten.

**OfflissFragment** Alle unsere Seiten (Fragments) haben eine Toolbar mit einem Titel und einem Menü-Knopf. Bei den Detail Ansichten ist der Knopf ein Pfeil zurück und bei den anderen das Hamburger-Menü Symbol, welches die Navigation öffnet. Alle unsere Fragments erben direkt oder indirekt von diesem OfflissFragment.

**OfflissOverviewFragment** Alle Overview-Fragments sind gleich aufgebaut. Sie besitzen ein Entity-RecyclerListAdapter in welchem die Entitäten angezeigt werden. Wählt man einen aus, gelangt man in die Detail-Ansicht. Über den Floating Action Button mit dem Plus kann man eine neue Entität in der Detail-Ansicht erstellen. Enthält die Liste keine Einträge erscheint ein Text, der darauf hinweist, dass zurzeit keine Entitäten erfasst sind.

**OfflissDetailFragment** Die Details enthalten mehr Logik als die Overviews. Die ausgewählte Entität wird in die Ansicht übergeben. Bei neuen Objekten wird null mitgegeben. Dementsprechend wird der Toolbar-Titel auf Create oder Edit angepasst. Ausserdem wird ein Save Button eingeblendet, um die Entität zu speichern. Beim Anwählen wird zuerst geprüft, ob die Entität vollständig ausgefüllt wurde. Dann wird diese mit den entsprechenden Events gespeichert und zum Schluss das Fragment abgebaut, so dass die Overview-Ansicht wieder erscheint. Die Validierungs- und Save-Methoden werden von den einzelnen Fragments mit Logik bestückt, da diese je nach Attribut und Objekt sich unterscheiden.

**About** Das About ist ein spezielles Fragment, da es lediglich Informationen anzeigt. Es erbt deshalb direkt vom OfflissFragment.

**NoPermission** Ist ein Benutzer berechtigt sich anzumelden, besitzt aber keine der erforderlichen Offliss-Rollen für die Navigation, wird das NoPermission Fragment eingeblendet. Mehr als ein Informationstext und das Logout sieht der Benutzer nicht.

Die anderen Fragments werden nicht weiter erläutert, da diese jeweils gleich aufgebaut sind wie das OfflissOverviewFragment bzw. OfflissDetailFragment.

### 8.10.6 List Adapter

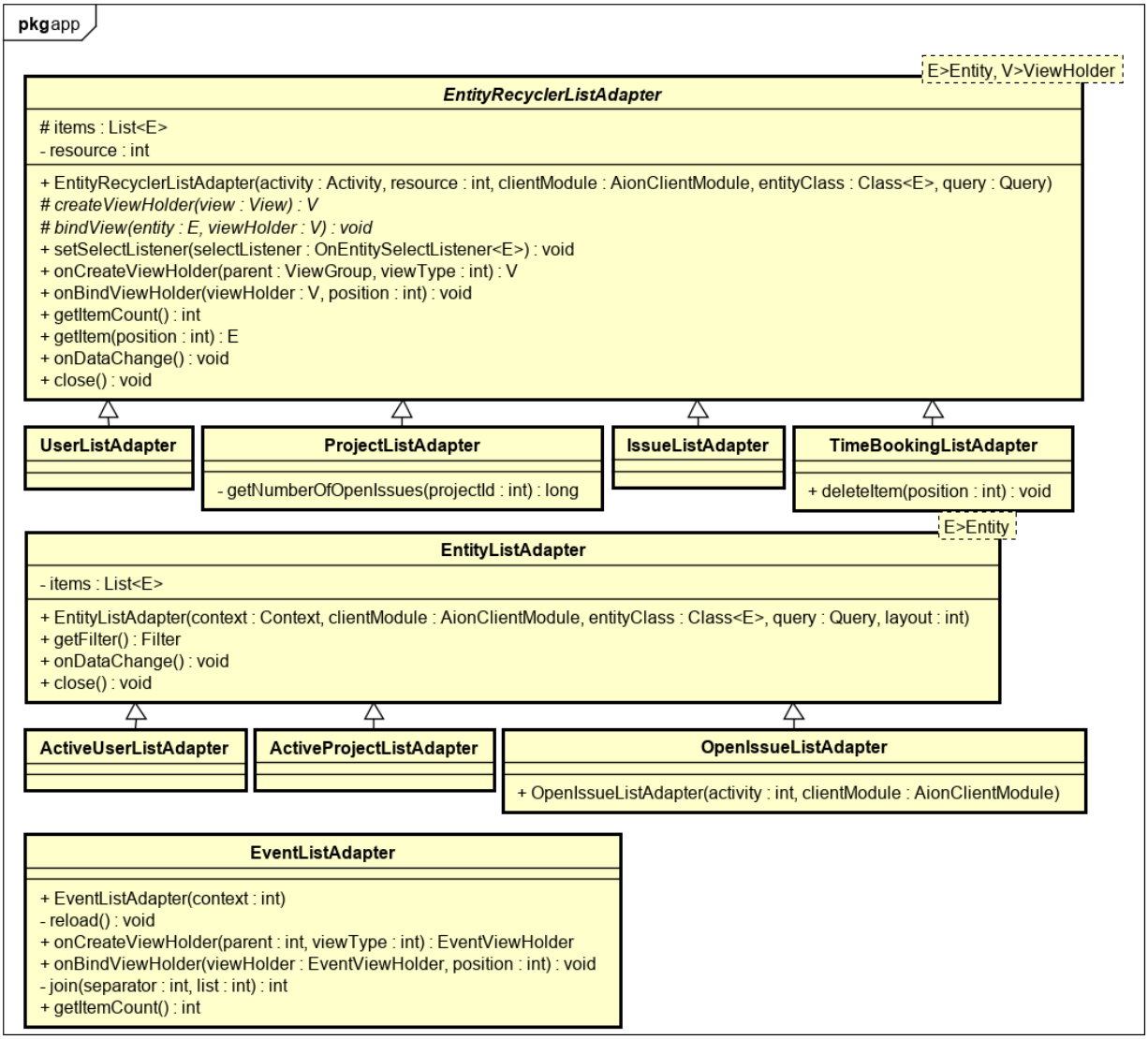


Abbildung 68: List Adapter Klassendiagramme

Die Klassen EntityRecyclerListAdapter und EntityListAdapter werden im Kapitel 8.6 genauer erläutert.

**UserListAdapter** Der UserListAdapter zeigt alle erfassten Benutzer mit ihrem Avatar, Namen, E-Mail, Benutzername und der Offliss-Rolle an. Gibt es keinen Avatar, wird das Bild mit einem grau ausgefüllt Kreis dargestellt.

**ProjectListAdapter** Der ProjectListAdapter zeigt alle erfassten Projekte mit ihrem Namen, dem Status als rundes grünes oder rotes Bild und die offenen Arbeitspakete unter dem Projekt an. Für jedes Projekt wird die Anzahl offener Arbeitspakete separat nachgeladen.

**IssueListAdapter** Im IssueListAdapter werden alle erfassten Arbeitspakete mit ihrem Namen (Titel und Nummer), zugehörigen Projekt und den zugeordneten Benutzer angezeigt. Sind keine Benutzer zugeordnet wird das mit einem schwarzen Avatar in einem grauen Kreis dargestellt.

**TimeBookingListAdapter** Das Arbeitspaket, das dazugehörige Projekt, die Dauer und das Startdatum aller erfassten Zeitbuchungen werden im TimeBookingListAdapter dargestellt. Hier gibt es eine zusätzliche deleteItem-Methode. Diese sorgt dafür, dass man durch wischen eine Zeitbuchung aus der Liste entfernen kann.

**ActiveUserListAdapter** Im ActiveUserListAdapter werden nur diejenigen Benutzer mit ihrem Namen dargestellt, welche aktiv sind. Diese Liste wird in der Auswahl der zu bearbeitenden Benutzer im Arbeitspaket verwendet.

**ActiveProjectListAdapter** Alle aktiven Projekte mit ihrem Namen werden im ActiveProjectListAdapter angezeigt. Diese Liste wird bei der Projektauswahl im Arbeitspaket benötigt.

**OpenIssueListAdapter** Dieser ListAdapter enthält alle offenen Arbeitspakete mit ihrem Namen (Titel und Nummer) für die Auswahl der Arbeitspakete in der Zeiterfassung.

**EventListAdapter** Dieser Adapter zeigt die durchgeführten Events und ihre Status an. Es werden der Eventname, Beschreibung, Status, Erstelldatum und gegebenenfalls der Fehler dargestellt. Die Daten werden immer wieder neu geladen, so dass immer der aktuelle Status auf den Events dargestellt wird.

## 8.11 Server (Offliss)

### 8.11.1 Server

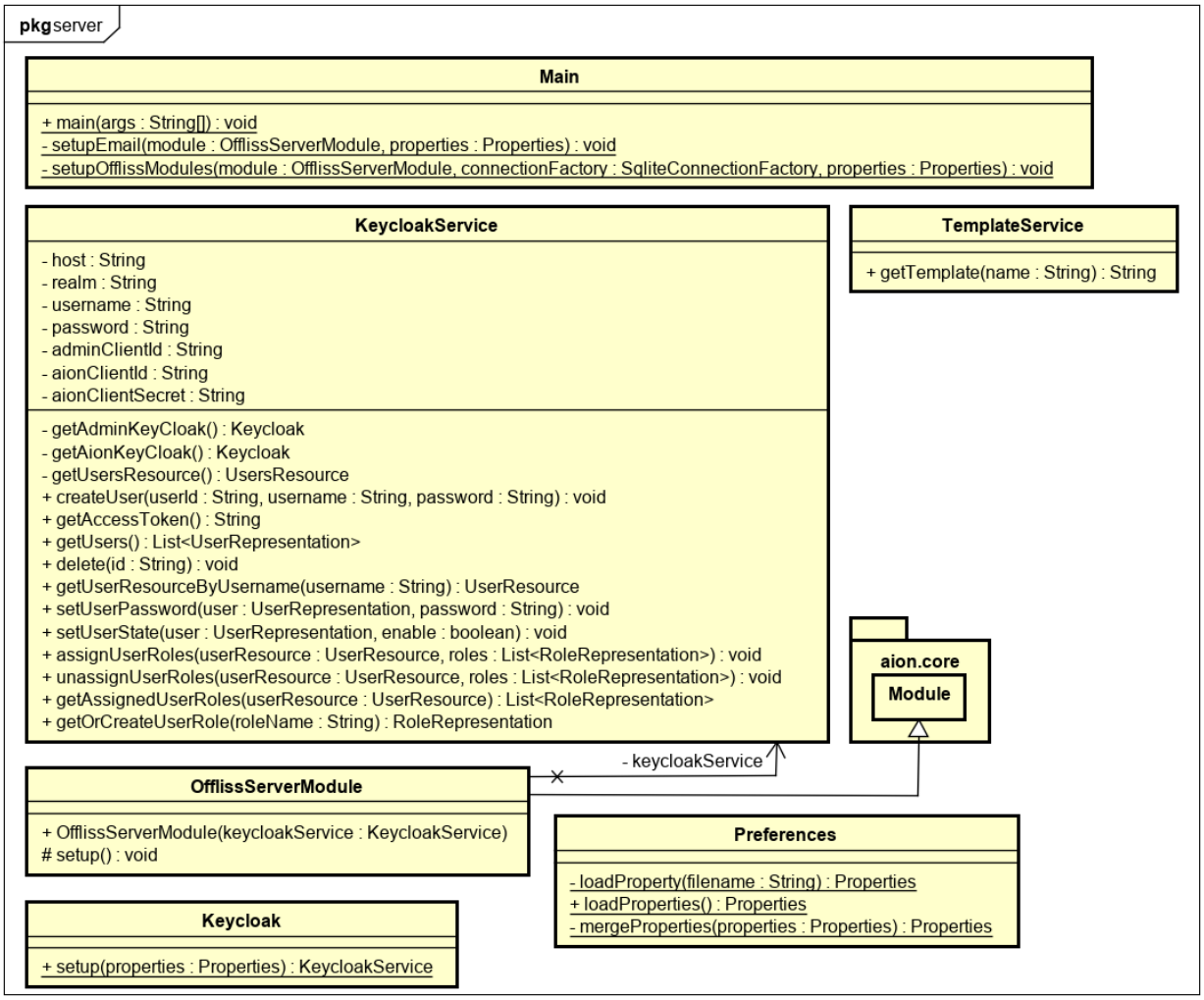


Abbildung 69: Server Klassendiagramme

**Main** Hier werden Konfigurationen geladen und Verbindungen zu den Timelines hergestellt. Es lädt die Business-Logik von Offliss und verarbeitet diese kontinuierlich in einer Endlosschlaufe.

**KeycloakService** Der KeycloakService ist die Verbindung zur Keycloak-Instanz. Mit diesem werden Benutzer erstellt, gelöscht, aktiviert, deaktiviert, Passwortänderungen und Rollenzuweisungen getätigt.

**Keycloak** Keycloak erstellt einen neuen KeycloakService und befüllt diesen mit den Verbindungsdaten.

**Preferences** Im Preferences werden die Konfigurationsdateien geladen und zusammengefügt.

**TemplateService** Der Service liest das entsprechende E-Mail-Template aus und gibt dieses zurück.

**OfflissServerModule** Dieses Modul bindet und initialisiert alle Event-Aggregatoren und Services.

### 8.11.2 Issue

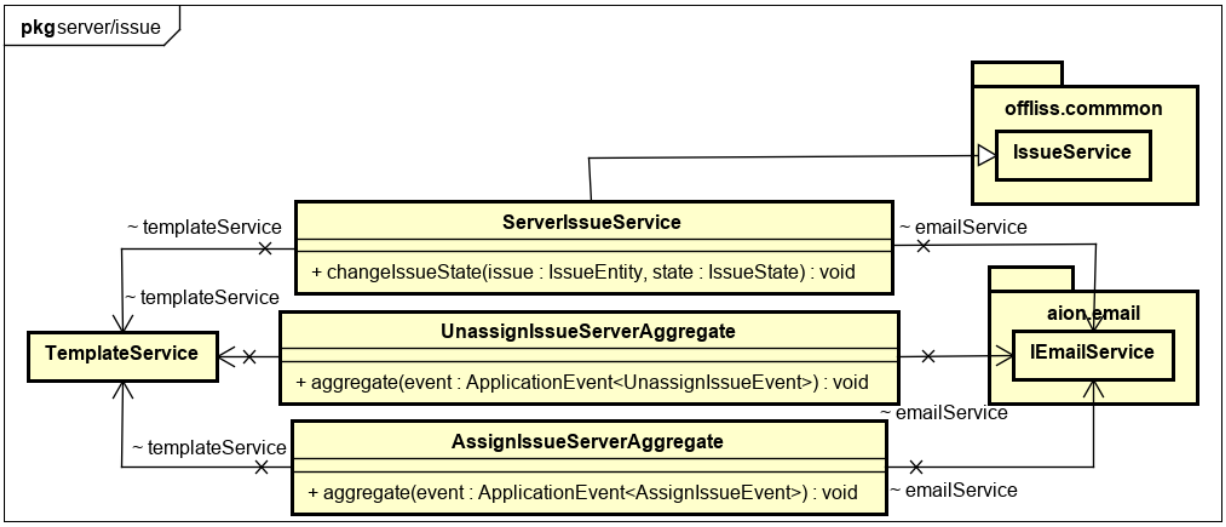


Abbildung 70: Server/Issue Klassendiagramme

**ServerIssueService** Der ServerIssueService erweitert die ChangeIssueState-Methode so, dass sie im CLOSE-Fall ein E-Mail an alle assigned Benutzer sendet und diese über den neuen Status informiert.

**UnassignIssueServerAggregate** Wird ein Benutzer von einem Arbeitspaket entfernt, erhält er ein E-Mail, indem er über den Entzug informiert wird.

**AssignIssueServerAggregate** Dasselbe gilt für das Zuweisen eines Benutzers an ein Arbeitspaket. Auch hier wird ein E-Mail zur Information ausgelöst.

### 8.11.3 User

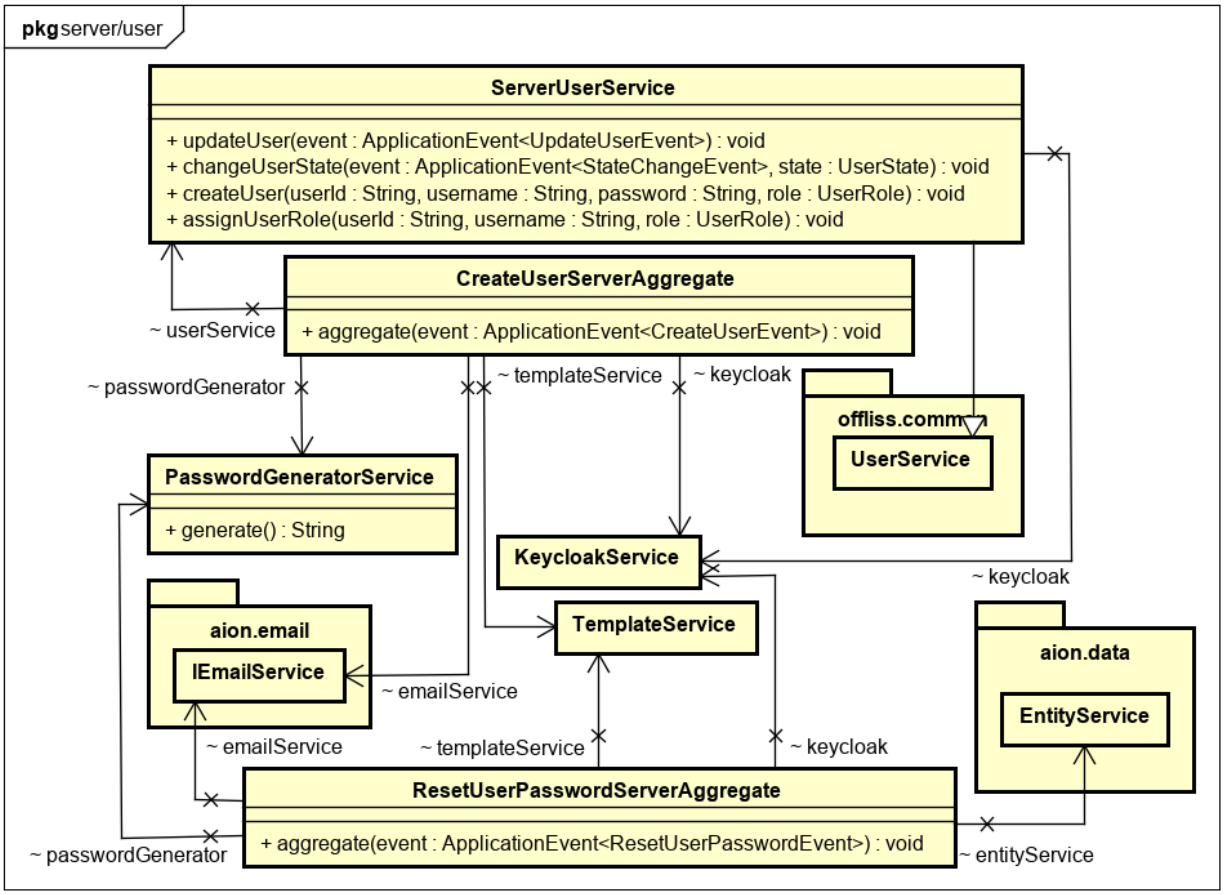


Abbildung 71: Server/User Klassendiagramme

**CreateUserServerAggregate** Ein neuer Benutzer muss im Keycloak erfasst werden, damit dieser sich am Offliss anmelden kann. Dazu wird ein neues Passwort generiert und ihm, über seine E-Mailadresse in einem E-Mail, mitgeteilt. Nachdem der Benutzer erstellt wurde, werden ihm seine Rollen zugewiesen.

**ResetUserPasswordServerAggregate** Beim Passwort zurücksetzen wird ein neues Passwort generiert und dieses per E-Mail dem Benutzer mitgeteilt.

**PasswordGeneratorService** Dieser Service generiert ein acht-stelliges neues Passwort mit jeweils zwei Gross- und Kleinbuchstaben sowie zwei Zahlen.

**ServerUserService** Der ServerUserService übernimmt die Aufgabe die Keycloak-Rollen, anhand der Offliss-Rollen, zu setzen und aktiviert bzw. deaktiviert die Benutzer.

# 9 Implementation

In den folgenden Unterkapiteln gehen wir auf die Implementationsdetails ein. Dabei wollen wir auf die wichtigsten Details und die dahinterliegenden Entscheidungen eingehen.

## 9.1 Modulstruktur

Die Implementation der einzelnen Packages wurde in separierten Java Modulen vorgenommen, welche sich gegenseitig referenzieren. Dadurch ist die Funktionalität in viele kleine Module unterteilt, kann einzeln kompiliert und installiert werden. Das Ziel ist es, die Abhängigkeiten für den Anwender zu reduzieren, so dass er nur die benötigten Module installiert. Dies machte die Entwicklung aufwändiger, da wir die einzelnen Funktionen klar strukturieren und trennen mussten. Aufgrund der Aufteilung werden die Schnittstellen besser geplant und umgesetzt. Ausserdem können andere Entwickler mit geringerem Aufwand Erweiterungen und Integrationen an unserem Framework vornehmen.

Um die Beziehungen der einzelnen Module zu zeigen, haben wir den resultierenden Abhängigkeitsbaum grafisch aufgearbeitet.

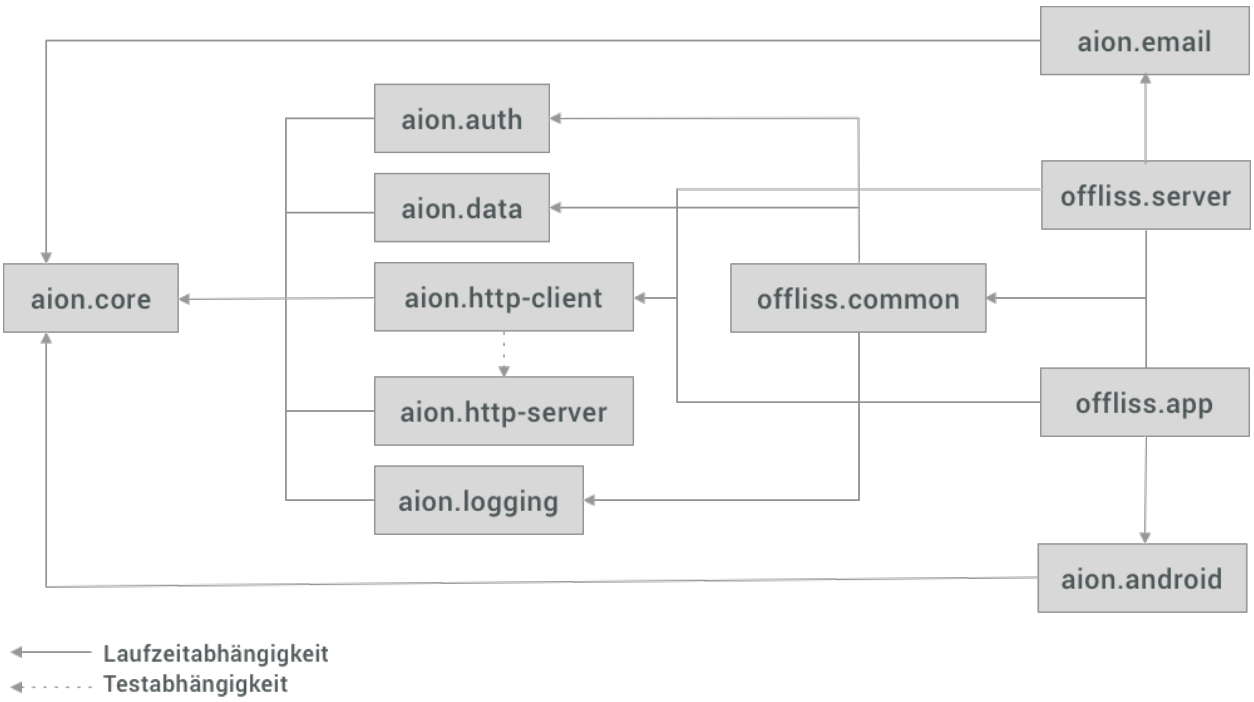


Abbildung 72: Modul-Abhängigkeitsdiagramm

## 9.2 Timeline

Die Idee hinter der Timeline war, dass eine Reihe an Events sich besser synchronisieren lassen als ein Datenzustand. Einen Datenstand abzugleichen ist aufwändiger als nur die einzelnen Events durchzugehen, welche zu diesem führen. Dieses Verfahren wird bei Datenreplikationen bereits eingesetzt, ist jedoch nicht für externe Teilnehmer einsehbar.[54]

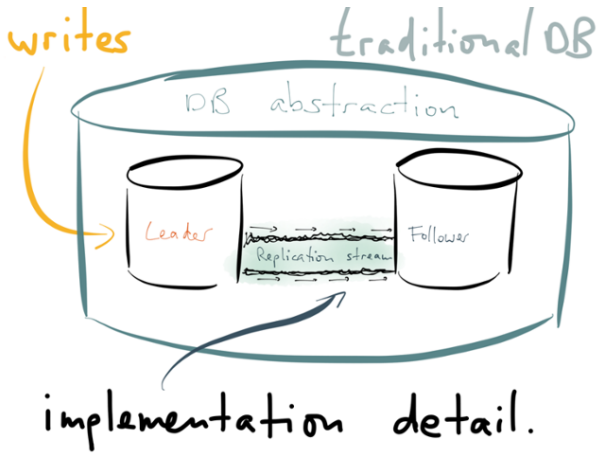


Abbildung 73: Datenreplikation (turn the database inside out)

Die Events in einer Timeline können aus mehreren Datenoperationen bestehen und bilden eine Projection. Sie können nicht parallel verarbeitet werden, da sie immer in der gleichen Reihenfolge abgespielt werden müssen. Sollte dies zu Performanz Problemen führen, müssen die Events auf verschiedene Timelines aufgeteilt werden.

## 9.3 Timeline Synchronisation

Das Ziel der Timeline Synchronisation ist es, auf Client- sowie Serverseite dieselbe Reihenfolge an Events vorzufinden. Die Client- und Server-Timeline können laufend durch neue Events erweitert werden, ohne Absprachen untereinander. Sobald die Synchronisation beendet, sollten beide die gleiche Abfolge aufweisen. Damit dies funktionieren kann, sind die Events unveränderbar und können auf der Serverseite nur angefügt werden. Somit gibt der Server vor, welches die korrekte Reihenfolge ist, während der Client seine allenfalls korrigieren muss. Dadurch ist der Aufwand auf der Serverseite gering. Gleichzeitig stellen synchronisierende Clients keinen Mehraufwand dar.

Bei einer vollständigen Synchronisation werden in einem ersten Schritt alle lokalen Events, welche sich nicht bereits auf dem Server befinden, an den Server übermittelt (push). In einem zweiten Schritt werden alle Events, welche sich noch nicht auf dem Client befinden, heruntergeladen (pull). Die Reihenfolge push vor pull ist bewusst gewählt, denn dadurch kann die Synchronisationslogik vereinfacht und die Synchronisationszeit verkürzt werden. Es verhindert, dass man in einem System, in welchem die Schreibkapazität sehr gross ist, endlos die Events herunterlädt. Die Änderungen werden zum Server gesendet, sobald eine Verbindung besteht. Dann holen wir die Informationen über den, zu diesem Zeitpunkt, letzten Event und laden alles zwischen der letzten Synchronisation und diesem herunter.

[54] Turn the database inside out: https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

## 9.4 Timeline Verarbeitung

Die Events in einer Timeline können nicht ohne weiteres für Datenabfragen verwendet werden. Um zum Beispiel festzustellen, welche Projekte aktiv sind, müssten zuerst alle Events verarbeitet werden, damit ein Resultat zurück geliefert werden kann. Dies ist sehr Ressourcenaufwändig. Darum wird aus den Events eine Projection generiert, welche für Abfragen genutzt werden kann. Der TimelineProcessor in Aion erzeugt diese zusammen mit der Business Logik. Durch die Synchronisation kann es jedoch vorkommen, dass Events dazwischen eingefügt werden. Der TimelineProcessor muss diese berücksichtigen, weil die Reihenfolge entscheidend sein kann. Sobald ein Event eingefügt wird, muss der Processor an diese Stelle zurückspringen und die Projections neu berechnen. Aus diesem Grund haben wir den CheckpointState entwickelt, welcher alle Datenmanipulationen speichert und diese zurücksetzen kann.

Ein anderer Lösungsansatz hätte sich mit Rückwärtsaktionen, für alle Events in der Business-Logik, beschäftigt. Dies fanden wir allerdings umständlich für den Entwickler und auch fehleranfällig.

## 9.5 Data OR-Mapper

Weil wir die Kontrolle über alle Datenzugriffe benötigen, um die Datenmanipulationen abzufangen, entwickelten wir einen minimalen OR-Mapper. Diese brauchen wir auch über das Transaktionsverhalten, um sicherzustellen, dass die Verarbeitung eines Events in einer einzigen Transaktion stattfindet. Dadurch kommt es im Fehlerfall zu keinen Inkonsistenzen und die Verarbeitung kann zu jedem beliebigen Zeitpunkt unterbrochen werden.

Unser entwickelter OR-Mapper ist nicht ausführlich und es fehlt ihm an vielen üblichen OR-Mapper-Funktionalitäten. Wir haben aus zeitlichen Gründen nur diejenigen umgesetzt, welche wir zwingend für unsere Offliss-Anwendung benötigten.

## 9.6 Transaktionen

In Aion müssen an mehreren Stellen Daten persistiert werden. Dies sind Event Informationen, Projections und Synchronisationsfortschritte. Für diese Schreibaktionen werden Datenbanktransaktionen verwendet, um sicherzustellen, dass Aion jederzeit unterbrochen werden kann und keine Dateninkonsistenzen entstehen.

## 9.7 SQLite

Die Persistenz auf Server und Client wurde mit SQLite umgesetzt, sodass die Implementation auf allen möglichen Geräten funktioniert. Die Anbindung wurde mit JDBC entwickelt, was auf der Serverseite mit einer JVM kein Problem darstellte. Für Android fanden wir ein Projekt, welches auch mit JDBC funktionierte, jedoch wies dieses einige Abweichungen zum Server auf. Wir realisierten die Anwendungen so, dass der gleiche Code auf dem Server sowie dem Client verwendet werden kann. Aufgrund von Problemen bei der Verwendung mehrerer Threads, die auf dieselbe Datenbank zugreifen, implementierten wir ein Locking, welches die gleichzeitige Nutzung der Connection verhindert. Da dies in JDBC nicht standardmässig umgesetzt ist, entwickelten wir das selbst.

## 9.8 Aion Http-Server

Der Aion Http-Server ist eine eigenständige Anwendung, die wir als Docker-Container ausliefern und verwenden. Die Applikation wird durch eine Konfigurationsdatei gesteuert, welche die projektspezifischen Informationen beinhaltet. Diese Datei ist in yaml spezifiziert. Der Server soll wiederverwendbar und offen für andere Projekte sein.

### 9.8.1 Konfiguration

Die Konfiguration erlaubt es, mehrere Timelines zu spezifizieren. In dem folgenden Beispiel aus dem Offliss, verwenden wir vier Timelines. Dabei nutzen alle Timelines SQLite. Eine davon ist mit einem Regex konfiguriert, welcher die Timeline separiert. In Offliss wurden so die Zeitbuchungen pro Benutzer aufgeteilt, dadurch muss die Konfiguration nicht erweitert werden, sollte ein neuer Benutzer dazu kommen.

```
auth:
  type: oauth
  serverUrl: "https://auth.ebos.bertschi.io/auth/realms/master"
  clientId: "f02943d2-6e25-440b-b1bb-a49831c9e959"

timelines:
  user:
    type: sqlite
    options:
      directory: database/user

  project:
    type: sqlite
    options:
      directory: database/project

  issue:
    type: sqlite
    options:
      directory: database/issue

  timebooking:
    type: sqlite
    options:
      directory: database/timebooking
    regex: ^timebooking[-][0-9A-Fa-f]{12}$
```

### 9.8.2 Autorisierung

Ist eine Autorisierung konfiguriert, muss der Client bei jeder Abfrage seine Authentisierung mitsenden. In unserer Anwendung haben wir uns für OAuth mit JWT entschieden, da JWT sehr gut offline verwendet werden kann. In jeder Anfrage muss dadurch ein JWT Access Token enthalten sein, das Schreib- oder Leseberechtigungen hat. Darin können Rechte und Metainformationen abgelegt werden, welche über eine Signatur sicher und ohne Kommunikation prüfbar sind. Im folgenden Beispiel ist ein Ausschnitt eines JWT Access Tokens zu sehen, welcher definiert, dass der Aufrufer Schreib- und Leserechte auf seine eigene Timebooking- und Leserechte auf der User-Timeline besitzt.

```
{
  realm_access": {
    "roles": [
      "user_read",
      "timebooking-abcedf_read",
      "timebooking-abcedf_write"
    ]
  },
}
```

Der Http-Server prüft diese Rechte zu Beginn der Anfrage und weisst unautorisierte Zugriffe zurück. In jedem Timeline Event wird der ausführende Benutzer sowie seine derzeitigen Rollen erfasst. Diese werden später in der Verarbeitung der Timelines verwendet, um sicherzustellen, dass einzelne Events nur von befugten Personen ausgeführt werden können.

Lese- und Schreibrechte können nur auf einer Timeline, und nicht auf Events definiert werden. Berechtigungen auf Events können allerdings durch das AuthModule sichergestellt werden.

Keycloak implementiert OAuth 2.0 sowie OpenID, bietet eine Weboberfläche und eine API, um Benutzer und Konfigurationen zu verwalten.

In der Implementation nutzten wir den Passwort Grant und verzichteten auf den üblichen OAuth-Ablauf. So konnten wir das Login direkt in der App umsetzen, ohne den kompletten OAuth/Ablauf zu realisieren.

## 9.9 Logging

Während der Entwicklung unserer Offliss Anwendung, stiessen wir des Öfteren auf das Problem, dass wir den Status der Events nicht feststellen konnten. Das Debuggen war Zeitaufwändig und nicht immer möglich auf dem Server. Wir haben darum im Laufe der Entwicklung ein Logging Modul geschrieben, welches uns die Verarbeitung und Resultate der Events auf der Konsole ausgibt.

Das LogModule verwendet einen Interceptor, welcher vor die Aggregationen geschaltet wird und so nützliche Informationen ausgibt, wie zum Beispiel Event Informationen und Fehlermeldungen der Aggregatoren.

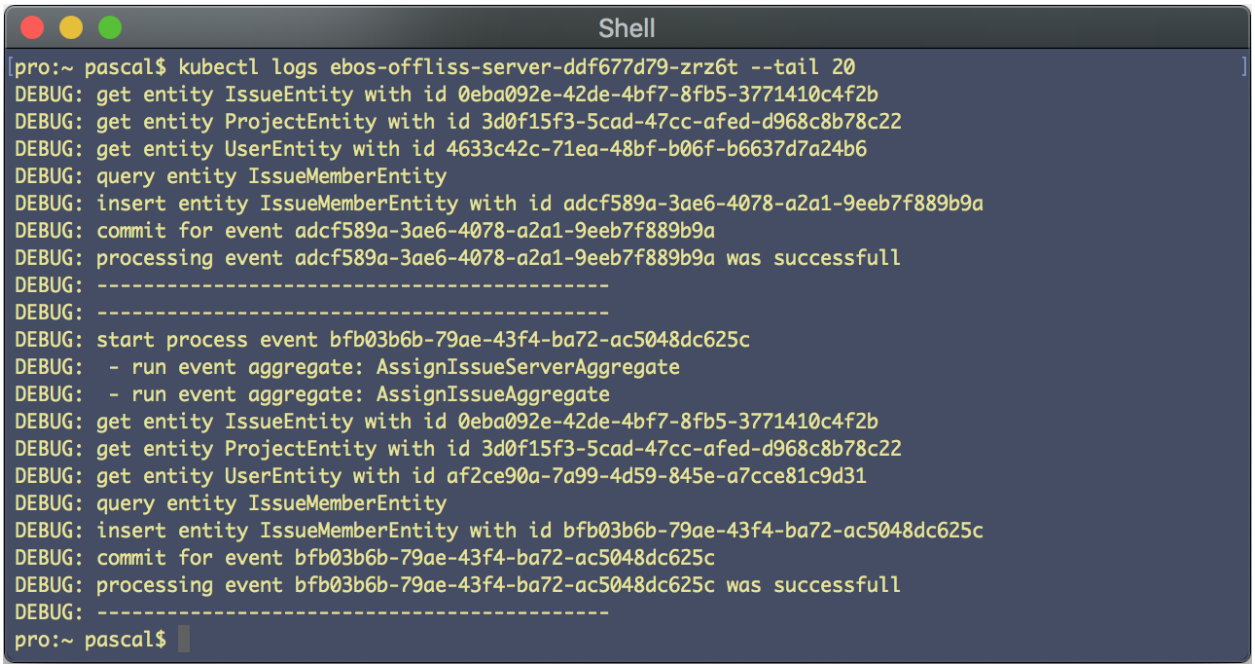


Abbildung 74: LogModule Console Output

Es wurde bewusst nicht eine bestehende Logger-Bibliothek verwendet, um die Komplexität gering zu halten. Bei der Implementation wurde aber darauf geachtet, dass die Ausgabe auf der Konsole durch eine Logger-Bibliothek ersetzt werden kann, welche diese in anderer Form abspeichern könnte.

## 9.10 E-Mail Hosting

Für den Versand von unseren Offliss E-Mails haben wir Mailgun verwenden. Mailgun ist ein Dienstleister, welcher sich um den Versand von E-Mails kümmert. Dadurch mussten wir keinen eigenen SMTP Server hosten. Die Kosten von Mailgun sind relativ gering bzw. bis zu 10'000 E-Mails gratis.

## 9.11 Offliss App

Für jede Entität gibt es eine Listenansicht und eine detailliertere Ansicht zur Eingabe von Daten.

Für die Offliss App wurden eigene Komponenten verwendet, so dass sie dem Material Design für Android Apps entsprechen.

### 9.11.1 Login

Die Login-Ansicht erscheint als erstes, wenn die App neu installiert wird. Man autorisiert sich mit seinem Benutzernamen und Passwort, die dann zur Überprüfung, via OAuth, an Keycloak weitergeleitet werden. Ursprünglich wollten wir die E-Mailadresse als Benutzername verwenden, aber Keycloak erlaubt keine Anpassungen an diesem. Deshalb fügten wir ein neues Attribut auf dem User hinzu, so dass sich die E-Mailadresse, bei einem Namenswechsel, anpassen lässt.

Nach erfolgreichem Login bleibt man angemeldet. Stimmen die Anmeldeinformationen nicht, wird der Benutzer daraufhin gewiesen. Für die Anmeldung ist zwingend eine Verbindung zum Keycloak Service nötig. Befindet sich der Benutzer offline und möchte sich anmelden, wird er mit einer entsprechenden Meldung informiert, dass das nicht geht.

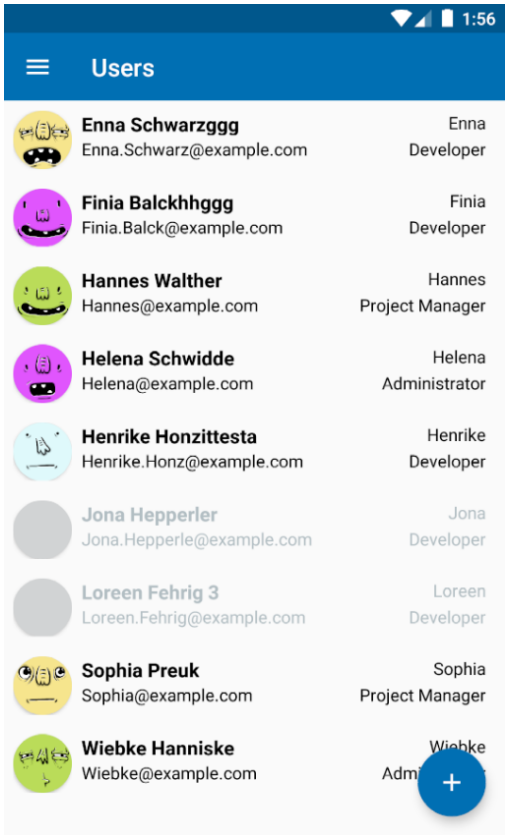


Abbildung 75: Offliss Userliste


Abbildung 76: Offliss Issue-Eingabe

### 9.11.2 Overview Ansicht (User)

Alle Listen beinhalten die wichtigsten Informationen über die Entitäten. Im Beispiel der Benutzerübersicht wurden alle Informationen dargestellt.
Viele Entitäten können aus Gründen der Nachvollziehbarkeit nicht entfernt, sondern nur inaktiviert werden. In der Liste werden diese dann ausgegraut oder mit einem roten Status markiert.

### 9.11.3 Detail Ansicht (Issue)

Wird ein Eintrag angewählt gelangt man zur Eingabemaske. Es gibt einige Entitäten, bei denen man gewisse Attribute nur einmal setzen kann. Beispielsweise gehören der Benutzername des Users oder das Projekt und die Laufnummer der Issues dazu.

### 9.11.4 Synchronization

In der Synchronisation werden alle eigenen fehlgeschlagenen und laufenden Events dargestellt. Das Datum zeigt, wann zuletzt eine Synchronisation durchlief. Erfolgreich durchgeführte Events werden aus Platzgründen nicht dargestellt. Die Synchronisation erfolgt alle 10 Sekunden, solange die App geöffnet ist. Deshalb sieht man seine ausstehenden Events in den meisten Fällen nur, wenn man offline arbeitet.

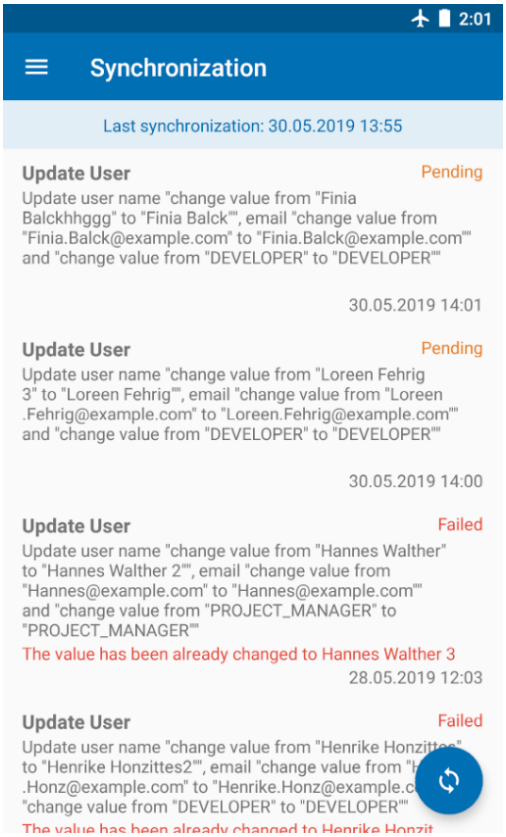

Abbildung 77: Offliss Synchronisation

### 9.11.5 Projection

In der Offliss Anwendung wollten wir die Synchronisation verkürzen und nur einen Teil der Daten synchronisieren. Deshalb haben wir die Events in vier Timelines aufgeteilt. Dadurch können diese voneinander unabhängig synchronisiert und verarbeitet werden. Allerdings wurden durch die Trennung der Timelines die Daten in verschiedene Datenbanken aufgeteilt. Daraus resultierte, dass wir die Daten nicht in einer einzelnen SQL-Abfrage abrufen konnten. Wir mussten darum die Daten der verschiedenen Projections jeweils einzeln nachladen, um alle benötigten Informationen im App darstellen zu können.

## 9.12 Offliss Common

Das Offliss Common Modul enthält die komplette Business Logik der Offliss Anwendung. Dazu gehören alle Events und deren Aggregates, welche die Projections errechnen und Konfliktentscheidungen beinhalten. Diese werden in der Android App sowie auf dem Server verwendet. Die Event Aggregates sind leicht zu testen und werden mit den In-Memory Implementationen in den Unit Tests geprüft.

Aus Sicherheitsgründen haben wir absichtlich keine Passwörter in den Event-Daten gespeichert. Diese würden ansonsten in der Timeline persistiert und für alle einsehbar sein.

## 9.13 Offliss Server

Der Offliss Server ist zuständig für das Versenden von E-Mails sowie die Erstellung der Benutzer im OAuth Server. Die Anwendung lädt regelmässig die Events aus den verschiedenen Timelines und tätigt die Aktionen, welche nicht auf dem App ausgeführt werden können. Das Erstellen von Benutzer und das Versenden von E-Mails wird, aus Sicherheitsgründen, nur auf dem Server ausgeführt. Der Server verwendet dazu die gleiche Business Logik wie in der App eingesetzt wird.

## 9.14 Offliss Timelines

Für unsere Offliss Anwendung mussten wir uns Gedanken machen wie die Timeline aussehen soll. Die Grundlage dazu war die Definitionen der Events, welche wir im Kapitel 8.10 dokumentiert haben.

Wir erstellten anstelle einer einzigen grossen Timeline mehrere kleine. Dies hat den Vorteil, dass wir später parallel und weniger Daten synchronisieren können.

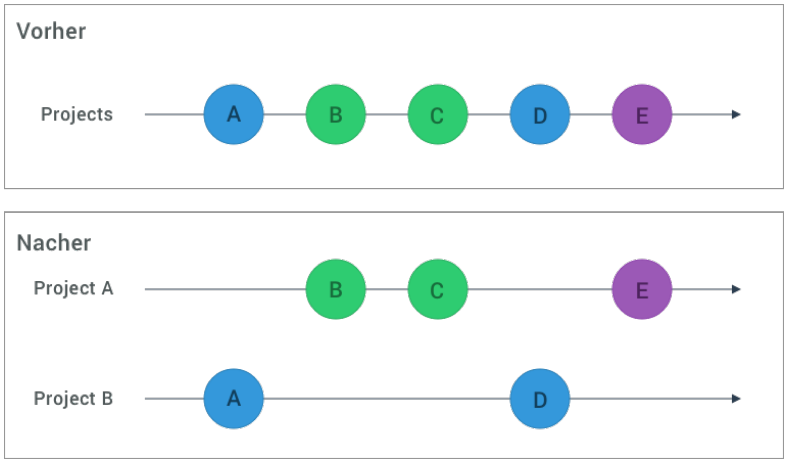


Abbildung 78: Timeline Aufteilung

**User Timeline** Die User Timeline beinhaltet die Events CreateUser, ActivateUser, InactivateUser, UpdateUser sowie ResetUserPassword. Hier werden die Benutzerinformationen gespeichert, welche für alle Anwender einsehbar sein müssen. Der Entwickler und der Projektleiter können die Events nur lesen, während der Administrator diese erstellen kann.

**Project Timeline** Die Project Timeline beinhaltet die Events CreateProject, OpenProject, CloseProject und UpdateProject. Hier befinden sich alle Projektinformationen aller Projekte. Der Entwickler kann die Events lesen und der Projektleiter erstellen.

**Issue Timeline** Die Issue Timeline wird pro Projekt geführt und beinhaltet die Events CreateIssue, UpdateIssue, AssignIssue, UnassignIssue, CloseIssue und OpenIssue. Die Entwickler und Projektleiter können hier lesen und schreiben.

**TimeBooking Timeline** Die TimeBooking Timeline wird pro Benutzer geführt und beinhaltet die Events CreateTimeBooking, UpdateTimeBooking und RemoveTimeBooking. Jeder Entwickler kann nur in seine eigene Timeline schreiben und aus dieser lesen.

## 9.15 Swagger UI

Der Http-Server bietet eine URL an, welche die OpenAPI Spezifikation retourniert. Damit kann zusammen mit dem Swagger UI eine grafische Oberfläche für die API dargestellt werden. Diese nutzten wir in der Entwicklung relativ häufig zur Prüfung und Visualisierung der einzelnen Events.

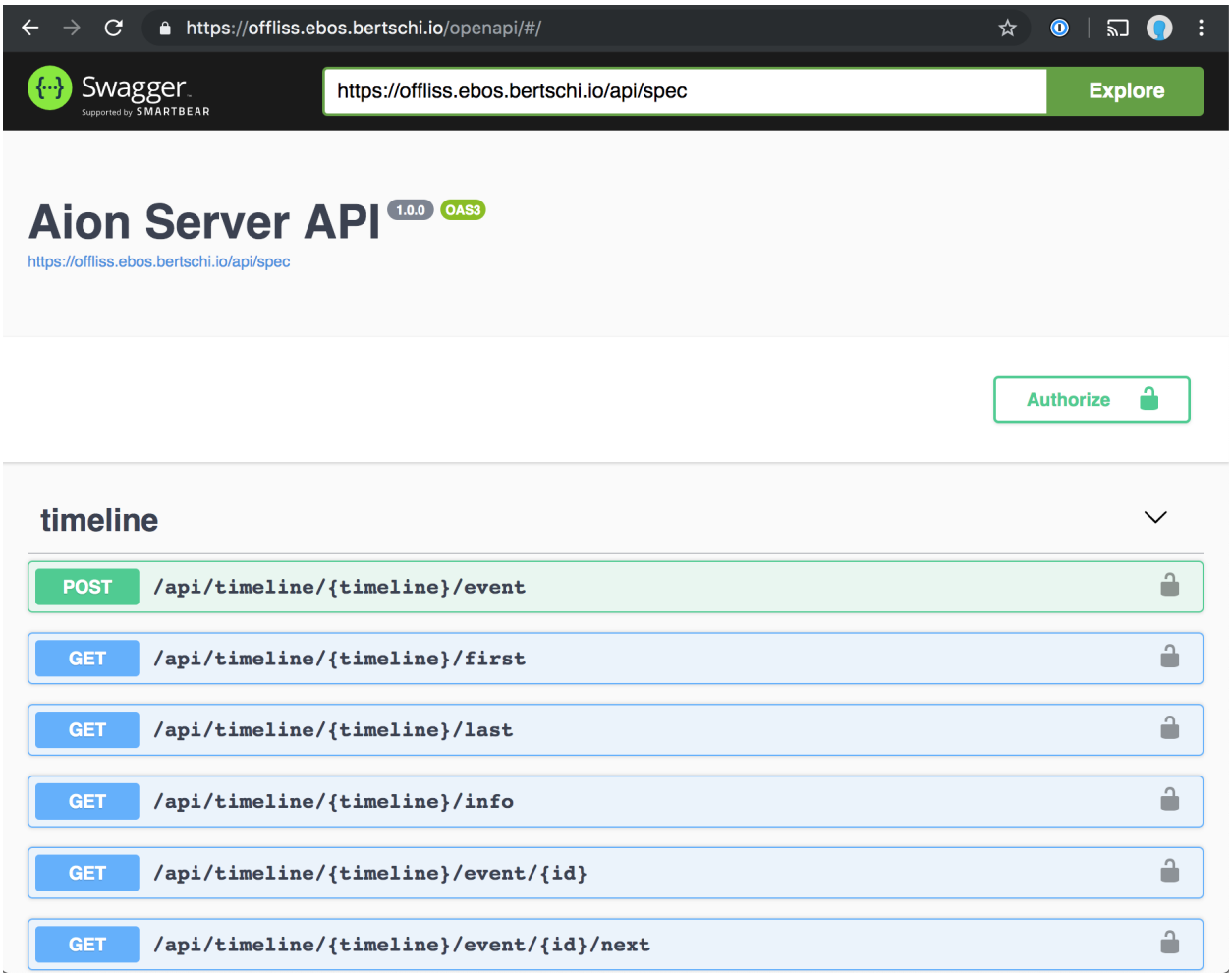


Abbildung 79: Swagger UI

Die Implementation des Http-Clients wollten wir zuerst, anhand der OpenAPI Spezifikation, generieren. Dann mussten wir jedoch feststellen, dass die generierten Datentypen Namenskonflikte verursachten. Die gemeinsame Codebasis in unserem Projekt konnten wir nicht mit dem generierten Code zusammenführen. Wir verzichteten darum auf die Generierung.

## 9.16 Unit Testing

Viele Codeabschnitte in unseren Modulen sind mit Unit Tests abgedeckt. Ausnahmen bilden die Module Android, App und der Offliss Server. Diese wurden manuell durch unsere definierten Testfälle in Abschnitt 13 getestet. Für Aion erreichten wir eine durchschnittliche Testcoverage von 82%.

| Files | ≡ | ● | ● | ● | Complexity | Coverage |
|---|---|---|---|---|---|---|
| auth/src/main/java/ch/hsr/ebos/aion/auth | 33 | 32 | 1 | 0 | 93.33% | 96.97% |
| core/src/main/java/ch/hsr/ebos/aion/core | 486 | 402 | 17 | 67 | 78.70% | 82.72% |
| data/src/main/java/ch/hsr/ebos/aion/data | 1,237 | 1,072 | 45 | 120 | 80.51% | 86.66% |
| email/src/main/java/ch/hsr/ebos/aion/email | 87 | 62 | 3 | 22 | 71.88% | 71.26% |
| http-client/src/main/java/ch/hsr/ebos/aion/http | 114 | 81 | 8 | 25 | 80.00% | 71.05% |
| http-server/src/main/java/ch/hsr/ebos/aion/server | 324 | 247 | 14 | 63 | 71.64% | 76.23% |
| logging/src/main/java/ch/hsr/ebos/aion/logging | 56 | 42 | 3 | 11 | 69.23% | 75.00% |
| **Folder Totals** (7 files) | 2,337 | 1,938 | 91 | 308 | 78.26% | 82.93% |

Abbildung 80: Code Coverage Aion

Im Offliss wurde nur die Business Logik mit Unit Tests geprüft, wobei wir auf 66% Testabdeckung kamen. Dieser tiefe Wert resultiert aus den vielen nicht verwendeten getter- und setter-Methoden in den Events, die einen grossen Teil der Zeilenanzahl ausmachen.

| Files | ≡ | ● | ● | ● | Complexity | Coverage |
|---|---|---|---|---|---|---|
| common/src/main/java/ch/hsr/ebos/offliss/common | 588 | 392 | 3 | 193 | 59.93% | 66.67% |
| server/src/main/java/ch/hsr/ebos/offliss/server | 389 | 0 | 0 | 389 | 0.00% | 0.00% |
| **Folder Totals** (2 files) | 977 | 392 | 3 | 582 | 46.23% | 40.12% |

Abbildung 81: Code Coverage Offliss

# 10 Infrastruktur

In der Abbildung 82 ist das Deployment Diagramm unserer Infrastruktur für unsere Offliss Anwendung. Wir haben uns für die Verwendung von Kubernetes entschieden und auf der Google Cloud Platform (GCP) einen solchen Cluster erstellt. Kubernetes ist eine Orchestrierungsplattform für Container, Open Source, plattformunabhängig und hat eine grosse Community.

Wir wollten uns auf unser Projekt fokussieren und nicht Zeit mit Installationen und Wartungen vergeuden. Die Wahl der Hosting-Plattform fiel auf GCP, weil diese, gemäss unserer Erfahrung, die stabilste und günstigste ist.

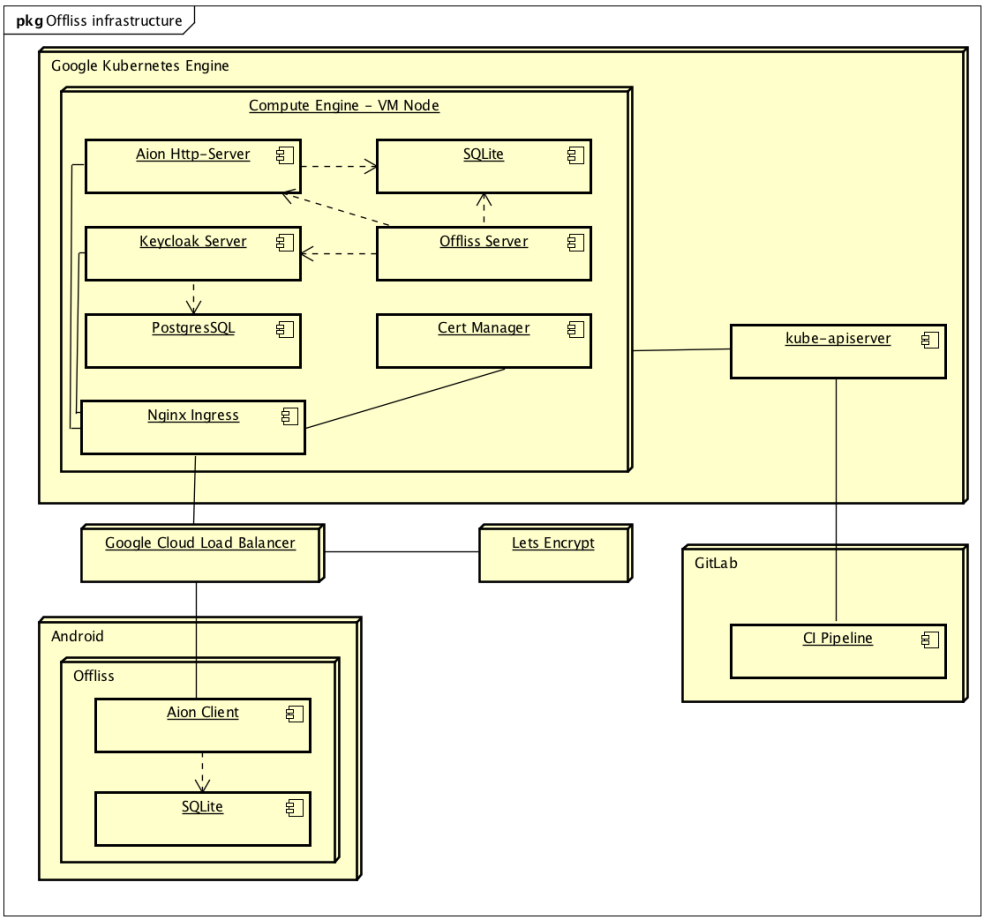


Abbildung 82: Offliss Deployment Diagramm

Der Cluster besteht nur aus einer Maschine. Auf dieser läuft ein NGINX Reverse Proxy. Hinter diesem befinden sich unser Aion Http-Server und der Keycloak-Server. Der Offliss-Server greift intern auf den Aion Http-Server zu und speichert wie er seine Daten in SQLite. Unser Cluster ist über einen Load Balancer mit dem Internet verbunden. Für eine gesicherte Verbindung haben wir von Lets Encrypt ein HTTPS Zertifikat ausstellen lassen.

Die Android Anwendung Offliss ist über den Aion Client mit unserem Server über HTTPS verbunden.

Unsere CI Pipeline auf GitLab kommuniziert mit dem kube-apiserver und verwaltet unsere Infrastruktur.

## 10.1 Continuous Deployment

Der Source Code ist in zwei GitLab Repositories aufgeteilt. Das Proof Of Concept Repository beinhaltet unsere Offliss Anwendung sowie unseren Aion Server. Im Infrastructure Repository sind die Kubernetes Konfigurationen, die unsere komplette Infrastruktur beschreiben.

Unsere CI Pipeline im Proof Of Concept wird durch einen Commit auf dem master-Branch ausgelöst, welcher ein Build und die Unit Tests durchführt. Im Erfolgsfall erstellt sie ein Docker-Image des Aion Server mit dem Tag latest. Darauf wird, durch einen Trigger, die Pipeline im Infrastructure Repository ausgelöst, welches mit dem kube-apiserver kommuniziert, um die Version des Containers auszutauschen. Somit wiederspiegelt der Codestand auf dem master-Branch immer, was auf der Server-Infrastruktur läuft. Die Passwörter der Infrastruktur werden in der CI Pipeline mit geheimen Umgebungsvariablen eingefügt. Dadurch können wir unsere Infrastrukturdefinitionen veröffentlichen, ohne an Sicherheit zu verlieren.

Es können auch einzelne Commits mit Git-Tags versehen werden, die wiederum ein versioniertes Docker-Image erstellen. Diese sind für einen offiziellen Release angedacht. Die Tags müssen gemäss der Semver Spezifikation[55] gesetzt werden.

Für die Offliss Anwendung wird auf dem master-Branch ebenfalls ein Build sowie Unit Tests durchgeführt. Das daraus resultierende apk wird als GitLab-Artefakt hochgeladen. Dieses Artefakt kann jederzeit heruntergeladen und installiert werden und ist im README[56] verlinkt.

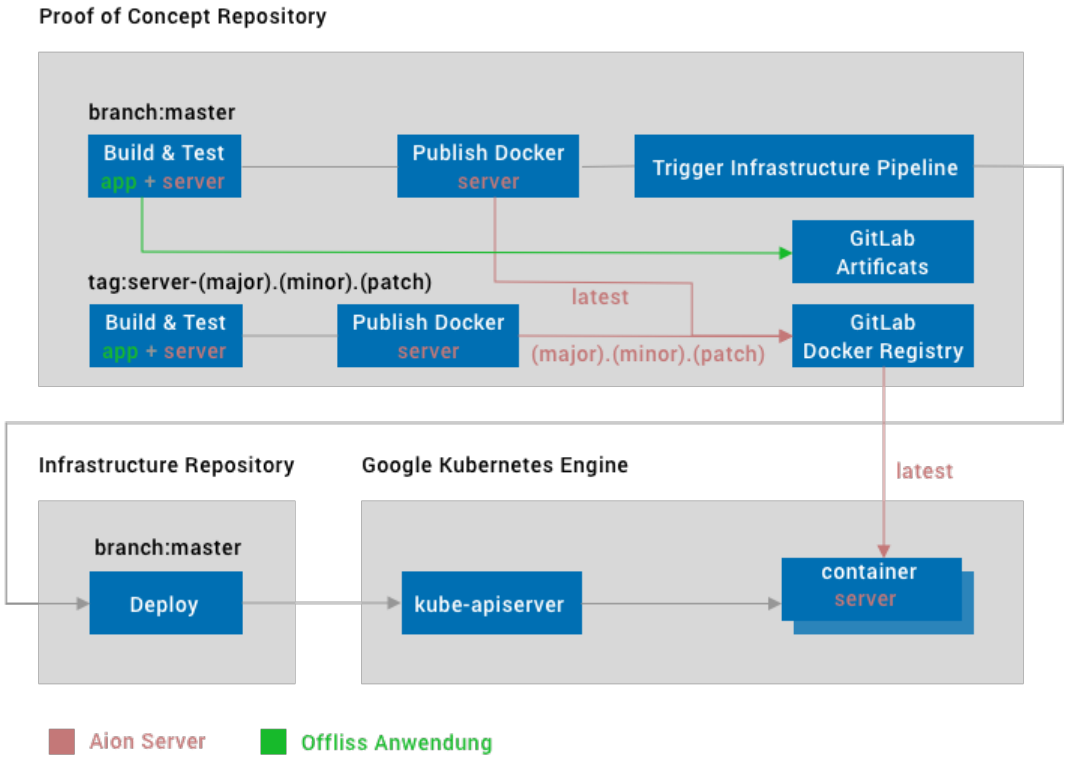


Abbildung 83: Continuous Deployment

[55] Semver: https://semver.org/
[56] POC README: https://gitlab.com/hsr-ebs/poc/tree/master

## 10.2 Log-System

Für die Fehleranalyse haben wir ein Logsystem eingerichtet. Mit filebeat[57] werden alle Logs aller Container an eine Elasticsearch[58] Instanz weitergeleitet. Mit Kibana[59] können diese Informationen ausgewertet und visualisiert werden.

Wir erhoffen uns dadurch schnell und einfach an die Fehlerquellen unserer Applikation zu gelangen.

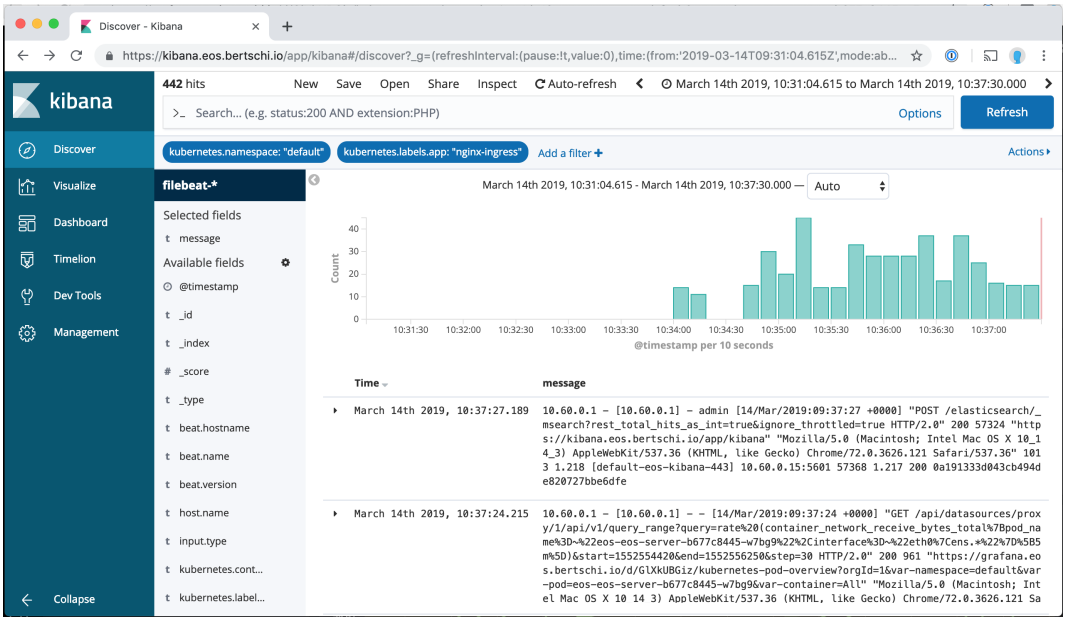


Abbildung 84: Kibana Dashboard

[57] Filebeat: https://www.elastic.co/products/beats/filebeat
[58] Elasticsearch: https://www.elastic.co/products/elasticsearch
[59] Kibana: https://www.elastic.co/products/kibana

## 10.3 Monitoringsystem

Für die Performanz-Analyse haben wir ein Monitoringsystem eingerichtet. Wir haben uns dabei für Prometheus[60] und Grafana[61] entschieden, da beides übliche Tools im Umfeld von Kubernetes sind. Eine Prometheus Instanz speichert alle CPU-, Arbeitsspeicher- und Netzwerk-Metriken, welche von einem Node Exporter[62] gesammelt werden. Die zusammengetragenen Metriken können mit Grafana visualisiert und somit auch Auswertungen von einzelnen Containern gemacht werden.

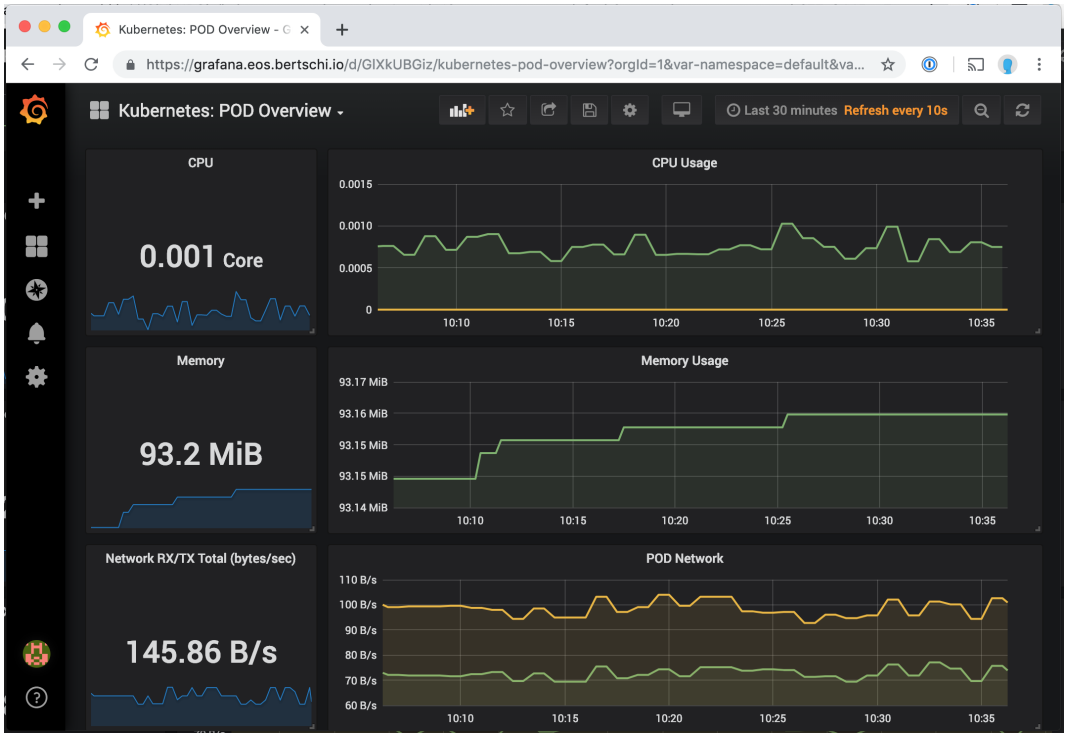


Abbildung 85: Grafana Dashboard

[60] Prometheus: https://prometheus.io/
[61] Grafana: https://grafana.com/
[62] Node Exporter: https://github.com/prometheus/node_exporter

# 11 Resultate und Weiterentwicklung

## 11.1 Resultate

Durch unsere Offline-Synchronisation haben wir erreicht, dass unsere Offliss Android App jederzeit funktioniert. Selbst wenn unsere Serverinfrastruktur für eine Stunde gewartet wird, können alle Benutzer wie gewohnt weiterarbeiten. Es müssen in der Entwicklung und in der Infrastruktur keine Ausfallmechanismen implementiert werden, um diese Ausfallsicherheit zu erreichen.

Die Programmierung der Events sowie Event Aggregatoren war sehr einfach. Auch die Business Logik ist dadurch sehr klar strukturiert und verständlich. Gleichermassen unkompliziert war das Testen durch die schlanke und flexible Logik und resultierte in einer hohen Testabdeckung.

Sehr viele Problemstellungen aus dem Kapitel 1.1 konnten wir mit dem Aion lösen. Beispielsweise können mit unserem Framework die gewohnten Datenbankfunktionalitäten angewendet werden, wie Transaktionen, Sequenzen und Constraints. Durch die Berechtigungsüberprüfung können alle Sicherheitsbedürfnisse abgedeckt werden. Mit unserem Framework lassen sich externe Systeme anbinden, wie der Mailversand. Die Teil-Synchronisation der Zeiterfassungen und die unterbruchsfreie Bedienung ermöglichen der Applikation eine gute Performanz und Stabilität.
Mit dem Aion haben wir gezeigt, dass man durch das Implementieren von Offline-Fähigkeit keine Einschränkungen in Kauf nehmen muss.

Durch den Einsatz von Event Sourcing bieten sich noch mehr Vorteile. Möchte man den Datenbestand sichern, muss man lediglich die Events aufbewahren. Dabei müssen nur die Events kopiert werden, welche seit dem letzten Stand neu dazugekommen sind. Deshalb ist das Sichern sehr speichereffizient.

Dadurch, dass alle Aktionen an der Anwendung festgehalten werden, können Benutzerfehler einfach nachvollzogen und reproduziert werden. Ausserdem lassen sich logische Fehler schnell beheben. Man muss die Events lediglich neu berechnen und die Defekte sind eliminiert.

Die Modularität hatte sowohl Vorteile als auch Nachteile. Die Flexibilität ist klar ein Vorteil gegenüber anderen Architekturen. Dafür erhöhte sich die Komplexität. Da alle Module einzeln importiert und konfiguriert werden mussten, erschwerte das die Verwendung von Aion.

Im Vergleich zu einer normalen Server-Client Applikation, war die Umsetzung nicht immer einfach. Darum empfehlen wir Aion nur an den Stellen einzusetzen, an welchen die Offline-Fähigkeit benötigt wird. Gerade weil unser Framework so flexibel ist, stellt dies kein Problem dar.

## 11.2 Weiterentwicklung

Wie in allen Projekten könnte man mit mehr Zeit noch viele praktische Funktionalitäten einbauen. Im Folgenden möchten wir einige Weiterentwicklungen aufzeigen.

### 11.2.1 Timeline Erweiterungen

Die von uns implementierte Timeline könnte um nützliche Funktionen für die Entwicklung erweitert werden. Aktuell ist das Löschen einer Timeline nicht einfach. Es müssen manuell die SQLite Dateien gelöscht und der Server neu gestartet werden.

Diese Funktion sollte nicht in einer produktiven Umgebung ausgeführt werden, weil dann der gesamte Datenstand verloren gehen würde. Doch während der Entwicklungsphase, können sich die Eventdaten ständig verändern.

### 11.2.2 Synchronisation Verbesserungen

Aktuell werden alle Events einzeln übermittelt. Um die Synchronisationszeit zu verkürzen, könnten die Events batchweise übermittelt werden.

Ausserdem könnten durch Snapshots der Projections die initiale Synchronisation beschleunigt werden. Dadurch müssten nicht alle Events auf den Client transferiert werden und die Projection wäre bereits berechnet, bis zu einem gewissen Zeitpunkt. Diese Snapshots könnten täglich erstellt werden und neuen Clients helfen, möglichst rasch Daten auf ein neues Gerät zu synchronisieren.

Die Abfrage, ob ein Client die aktuellsten Daten besitzt, könnte man so optimieren, dass weniger Daten kommuniziert werden.

Den Speicherplatz könnte man reduzieren, da sich aktuell immer alle Eventinformationen auf dem lokalen Gerät befinden. Diese könnten entfernt werden, sobald alle Events synchronisiert und die Projections erstellt wurden.

### 11.2.3 Aion Http-Server UI

Die Visualisierung der Event Timelines konnte wir zeitlich bedingt nicht mehr umsetzen. In der Entwicklung und für die Fehlersuche wäre diese Funktionalität sehr hilfreich.

### 11.2.4 Schema Migration

Das von uns entwickelte Datenmodul kann noch nicht mit Migrationen der Datentabellen umgehen. Damit die App in Zukunft auch erweitert werden kann, müsste eine Schemamigration der Projectiontabellen möglich sein.

# 12 Projektmanagement

Zur Planung und Strukturierung der Arbeitspakete und Features wird GitLab eingesetzt. GitLab bietet mit ihrer Issue-Verwaltung und deren Board-Ansicht die Möglichkeit die agile Methode Scrum umzusetzen. Die Zeiterfassung wird separat in einer Time Tracking Applikation namens Paymo erfasst. Die Abstufung wird 15 Minuten genau erfasst.

## 12.1 Entwicklung

Die Entwicklung des Source Codes wird in GitLab Repositories festgehalten. Der master-Branch wird für alle Entwickler gesperrt und kann nur mit einem Merge Request aus einem Feature-Branch erweitert werden. Die Merge Requests werden vom jeweils anderen Teammitglied überprüft und kommentiert. Dadurch werden der Wissensaustausch gefördert und die Fehlerquote gesenkt. Der Source Code wird mit der GitLab-CI automatisch gebuildet, getestet und publiziert, wodurch die Entwicklerproduktivität massiv gesteigert und repetitive Arbeiten automatisiert werden.

Der Source Code wird in Java entwickelt. Dabei wird die Google Java Style Guide[63] eingehalten.

**Definition of Done** Ein Arbeitspaket muss vor dem Abschluss folgende Kriterien erfüllen:

- Code ist nach Java Programming Style Guide formatiert
- Imports sind organisiert
- Unit Test zu Arbeitspaket wurden erstellt
- Alle Unit Tests sind erfolgreich
- Dokumentation wurde entsprechend erstellt oder erweitert
- Merge Request wurde von anderem Teammitglied überprüft und akzeptiert
- Build war erfolgreich
- Stakeholder hat das Arbeitspaket gesehen und sein Einverständnis gegeben.

## 12.2 Prozessmodell

Wir haben uns für eine Kombination aus Rational Unified Process (RUP) und Scrum entschieden. RUP ermöglicht es, das Projekt in mehrere Phasen zu unterteilen und koordinieren. Die Inception, welche den Projektantrag und -vision beinhaltet fand im Voraus statt und wird deshalb in diesem Dokument nicht weiter erwähnt. Die Analysen und das Prototypisieren werden in der Elaboration-Phase durchgeführt. In der Construction wird das Vorhaben iterativ mit Scrum umgesetzt. Scrum ermöglicht uns eine hohe Flexibilität durch adaptives Planen und bietet in den Meetings eine hohe Transparenz. Ausserdem können wir durch die zeitnahen Inkrementationen besser mit kurzfristigen Problemen umgehen. Der Projektabschluss wird in der Transition-Phase abgewickelt.

[63] Google Java Style Guide: https://google.github.io/styleguide/javaguide.html

## 12.3 Rollen und Verantwortlichkeiten

Die Dokumentation wird laufend während und nach der Implementation von beiden erweitert, wobei es keine klare Aufteilung gibt. Aion und Offliss haben wir in verschiedene Projekte aufgeteilt, an welchen wir einzeln arbeiten können. Unsere Aufteilung ist in den zwei folgenden Tabellen ersichtlich.

| Projekt | Core | Data | Auth | Email | Logging | Http Client/Server | Android |
|---|---|---|---|---|---|---|---|
| Pascal Bertschi | X | X | | X | X | X | X |
| Michelle Kunz | | X | X | | | | X |

Abbildung 86: Aion Aufteilung

| Projekt | Common | App | Server |
|---|---|---|---|
| Pascal Bertschi | | X | X |
| Michelle Kunz | X | X | |

Abbildung 87: Offliss Aufteilung

## 12.4 Planung

Die Planung wird bei Änderungen im Projekt laufend nachgeführt.
Details zu den einzelnen Phasen sind nachstehend genauer erläutert.

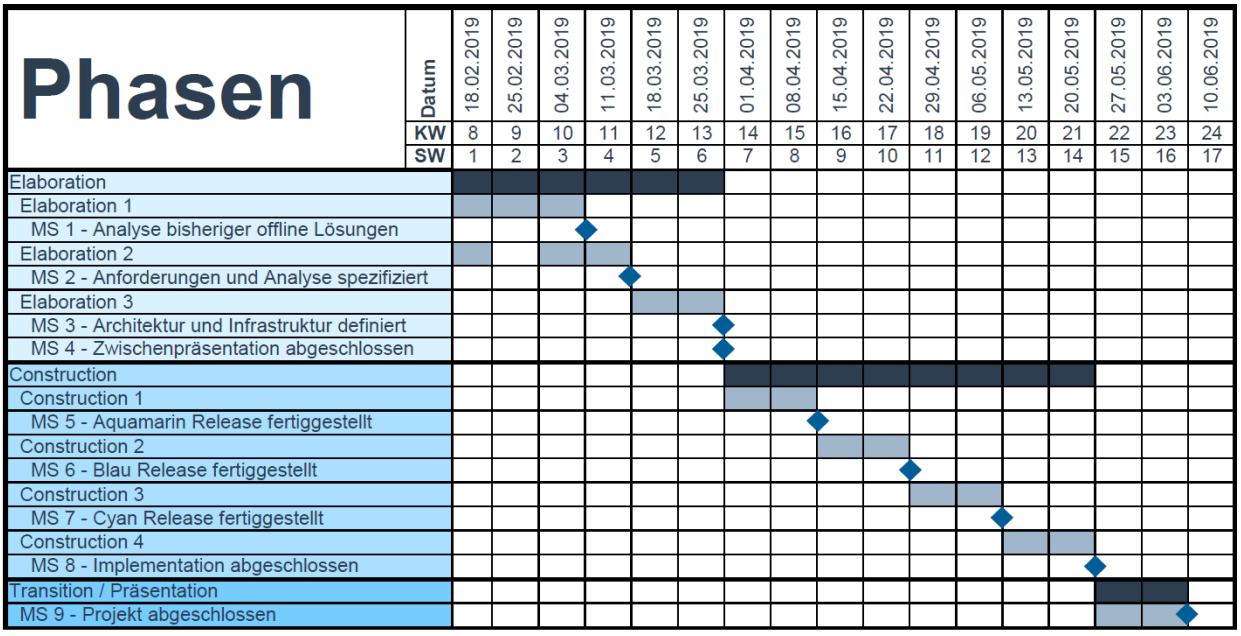


Abbildung 88: Projektplan mit den einzelnen RUP-Phasen

### 12.4.1 Elaboration

In der sechswöchigen Elaboration Phase werden die bestehenden Lösungen auf dem Markt analysiert und mit den Kriterien einer guten Offline-Synchronisationsanwendung verglichen. In der zweiten Etappe werden die Anforderungen an eine neue Lösung für den Offline-Gebrauch vertieft und festgehalten. Die beiden ersten Phasen überschneiden sich deshalb, weil erst durch die Schwachstellen der existierenden Lösungen die Anforderungen fertig spezifiziert werden können und dennoch müssen bereits erste Anforderungen bestehen, um die Analyse bestreiten zu können. Die Architektur und Infrastruktur der Tool-Chain für unsere Anwendung werden in den letzten zwei Wochen der Elaboration definiert und getestet. Diese Phase ist essenziell, da die darauffolgende Implementation stark abhängig von den resultierenden Ergebnissen ist. Die End of Elaboration stellt den Übergang zur Produktivität dar.
Gewünscht wird eine Zwischenpräsentation für die Bachelorarbeit. Diese wird ebenfalls vor der Construction Phase abgeschlossen.

### 12.4.2 Construction

Die Construction Phase dauert acht Wochen. Zu den einzelnen Meilensteinen gehören sowohl die Implementation als auch das Testing der einzelnen Releases.
Die Releases werden während zweiwöchigen Sprints realisiert. Ihre Kennung beinhaltet einerseits eine semantische Versionierung und andererseits einen Farbcode als Namen.

### 12.4.3 Transition

Während der letzten Woche werden die Dokumentation und das Abstract in den finalen Zustand gebracht und die dazugehörige Bachelor-Präsentation vorbereitet. Mit der Abgabe der Arbeit ist diese Phase beendet.

## 12.5 Stakeholder

In diesem Kapitel haben wir uns Gedanken zu den Stakeholdern von unserem Offline-Framework gemacht. Dabei konnten wir verschiedene Personen im App-Entwicklungsprozess analysieren und eine Stakeholderanalyse in Form eines Diagramms erstellen.

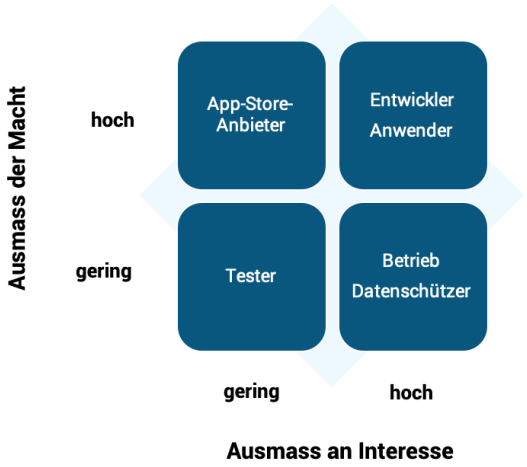


Abbildung 89: Stakeholderanalyse

**App-Store Anbieter** Der App-Store Anbieter gibt die Richtlinien vor für mobile Applikationen und hat somit einen hohen Einfluss. Es muss bei der Entwicklung darauf geachtet werden, dass alle Richtlinien eingehalten werden. Er hat das Interesse seine Nutzer seiner Plattform zu schützen und die Kontrolle darüber zu behalten.

**Entwickler** Der Entwickler trifft die Auswahl, welche Technologien verwendet werden und entscheidet somit ob unser Framework genutzt wird. Sein Interesse ist ein erfolgreiches Projekt durchzuführen und die Kunden- sowie Entwicklerzufriedenheit zu maximieren. Die Zufriedenheit besteht aus vielen Faktoren, unter anderem Dokumentation, Fehler, Stabilität und Funktionalität.

**Anwender** Der Anwender ist der Kunde des Entwicklers, welcher wiederum unser Kunde ist. Ist der Anwender zufrieden mit dem Produkt des Entwicklers, ist der Entwickler ebenfalls zufrieden. Das Interesse des Anwenders liegt darin, dass seine Bedürfnisse, die das Produkt erfüllen soll, erfüllt werden.

**Tester** Der Tester ist der erste Anwender des Produkts. Er hat nicht viel Einfluss auf das Produkt, aber ein Interesse an einer tiefen Fehlerquote.

**Betrieb** Der Betrieb ist für die Infrastruktur und Wartung des Produkts verantwortlich. Sein Interesse liegt darin, dass alles stabil läuft und er nicht jede Nacht geweckt wird, aufgrund eines Ausfalls.

**Datenschützer** Der Datenschützer will die Daten der Anwender vor Missbrauch schützen. Verstösse müssen behoben werden.

## 12.6 Risikoanalyse

Bei der Risikoanalyse geht es um die technischen Risiken, die im Laufe des Projekts eintreten könnten. Allgemeine Risiken werden nicht beschrieben, wie Hardware Ausfall, Ausfall von Git oder Git-Komponenten, Krankheit eines Teammitglieds etc. Diese projektspezifischen Risiken können den Projektverlauf erheblich beeinflussen. Es können nie alle Risiken zu Beginn identifiziert werden.

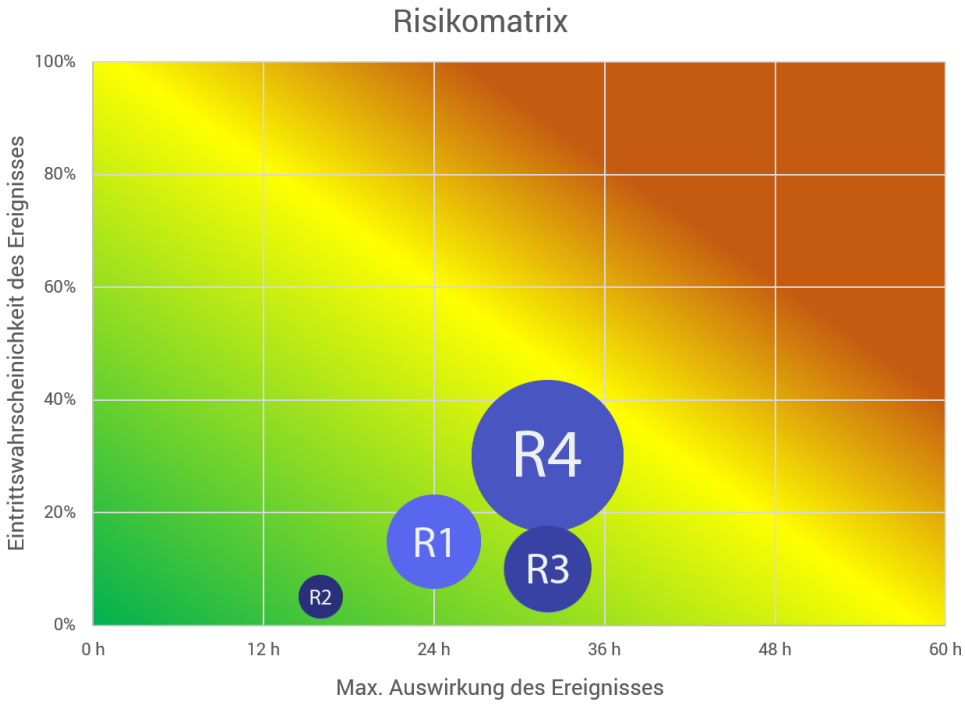


Abbildung 90: Risikomatrix

| Nr. | Titel | max. Schaden | Eintritts-wahrscheinlichkeit | Gewichteter Schaden |
|---|---|---|---|---|
| R1 | Architektur und Technologie Probleme | 24h | 15% | 3.6 |
| R2 | Lange Analyse Phase | 16h | 5% | 0.8 |
| R3 | Ungeplante Konkurrenz | 32h | 10% | 3.2 |
| R4 | Lösungskonzept funktioniert nicht | 32h | 30% | 9.6 |

Abbildung 91: Risiko Analyse

Die Risiken aus Abbildung 90 können vorgebeugt werden. Die dazugehörigen Massnahmen sind in den folgenden Abschnitten festgehalten.

### 12.6.1 Architektur und Technologie Probleme

**Beschreibung** Die gewählten Technologien, Bibliotheken und Frameworks eignen sich nicht für unsere Arbeit.
Die Architektur lässt nicht zu, dass die Bibliotheken und Frameworks miteinander arbeiten.

**Vorbeugung** Die Dokumentationen und andere Projekte mit den Technologien, Bibliotheken und Frameworks werden ausführlich studiert. Ausserdem wird in der Elaboration ein Architektur-Prototyp implementiert, um die Zusammenhänge zu testen.

**Verhalten beim Eintreten** Wir werden neue Technologien, Bibliotheken und Frameworks evaluieren müssen. Im Anschluss werden wir diese testen und einsetzen.

### 12.6.2 Lange Analyse Phase

**Beschreibung** Die Elaboration dauert länger als erwartet.
Die Anforderungen sind nicht leicht aufzunehmen.
Aufgrund schlechter Dokumentationen erweist sich die Analyse der bestehenden Lösungen, als schwer.
Der Architektur-Prototyp funktioniert nicht und andere Technologien müssen evaluiert werden.

**Vorbeugung** Wir fixieren die Phasen und Teil-Ziele mit Meilensteinen.

**Verhalten beim Eintreten** Sollte die Elaboration tatsächlich länger dauern, werden wir die Construction-Phase kürzen und auf einzelne Funktionalitäten der Offline-Synchronisation verzichten.

### 12.6.3 Ungeplante Konkurrenz

**Beschreibung** Eine neue Offline-Synchronisation, die alle Problemstellungen löst, kommt nach unserer Analysephase auf den Markt.

**Vorbeugung** Wir betreiben eine gute und gründliche Marktanalyse. Wir informieren uns nicht nur über gegenwärtige Offline-Systeme, sondern auch über Publikationen zukünftiger Systeme.

**Verhalten beim Eintreten** Sollte dieses Szenario eintreffen, müssen wir ein neues Vorgehen mit dem Betreuer definieren.

### 12.6.4 Lösungskonzept funktioniert nicht

**Beschreibung** Die Implementierte Lösung funktioniert nicht für alle Offline-Funktionalitäten.

**Vorbeugung** Die Anforderungen und Spezifikation werden gründlich definiert.

**Verhalten beim Eintreten** Wir evaluieren neue Strategien und implementieren diese.

Teil III

# Anhänge

# 13 Tests

Nach erfolgreicher Implementation werden die erfassten Anforderungen in Abschnitt 6 auf ihre Richtigkeit überprüft. Die Testfälle werden in der Testspezifikation definiert, während die Ergebnisse der Durchführung in Testresultate festgehalten werden.

## 13.1 Testspezifikationen

Jede Anforderung an das Offliss App bzw. an das Aion Framework wird in mindestens einem Testfall getestet.

### 13.1.1 TS01: Benutzer erfassen

**Use Case** UC01: Benutzer CRU

**Vorbedingung** Der Benutzer mit der entsprechenden E-Mailadresse und Benutzername existiert noch nicht und der Administrator ist synchronisiert.

**Testszenario** In der Benutzeransicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden ein Benutzername, Name, E-Mailadresse und eine Rolle für den Benutzer eingegeben. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Benutzeransicht ist ein neuer Benutzer mit dem eingegebenen Benutzername, Name, E-Mailadresse, Rolle und einem generierten Avatar erschienen.

### 13.1.2 TS02: Benutzer unvollständig erfassen

**Use Case** UC01: Benutzer CRU

**Vorbedingung** Der Benutzer mit der entsprechenden E-Mailadresse existiert noch nicht und der Administrator ist synchronisiert.

**Testszenario** In der Benutzeransicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden keine Daten für den Benutzer eingegeben. Danach wird “Save“ angewählt.

**Erwartetes Ergebnis** Dem Administrator wird angezeigt, dass der Benutzername, Name und die E-Mailadresse Pflichtfelder sind. Er kann den Benutzer nicht speichern.

### 13.1.3 TS03: Benutzer erfassen mit nicht automatisch lösbarem Konflikt

**Use Case** UC01: Benutzer CRU und UC06: Konflikt beheben

**Vorbedingung** Ein Benutzer mit der entsprechenden E-Mailadresse wurde bereits von jemand anderem erstellt und befindet sich auf dem Server. Der Administrator hat diesen Benutzer noch nicht auf sein Gerät synchronisiert.

**Testszenario** In der Benutzeransicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden ein Benutzername, Name und die vergebene E-Mailadresse für den Benutzer eingegeben. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Der Synchronisation-Event schlug fehl. Der Administrator wird in einer Fehlermeldung darauf hingewiesen, dass der Benutzer bereits existiert.

#### 13.1.4 TS04: Benutzer anzeigen

**Use Case** UC01: Benutzer CRU

**Vorbedingung** Es ist mindestens ein Benutzer erfasst.

**Testszenario** In der Navigation wird “Users“ angewählt.

**Erwartetes Ergebnis** Eine Liste von Benutzer mit Benutzernamen, Namen, E-Mailadressen, Rollen und Avataren werden dargestellt.

#### 13.1.5 TS05: Benutzer ändern

**Use Case** UC01: Benutzer CRU

**Vorbedingung** Es ist mindestens ein Benutzer erfasst und der Administrator ist synchronisiert.

**Testszenario** In der Benutzeransicht wird ein erfasster Benutzer angewählt. Der Name, die E-Mailadresse und Rolle werden geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Benutzeransicht wird der bearbeitete Benutzer mit anderem Namen, E-Mailadresse und Rolle angezeigt.

#### 13.1.6 TS06: Benutzer ändern mit automatisch lösbarem Konflikt

**Use Case** UC01: Benutzer CRU und UC06: Konflikt beheben

**Vorbedingung** Bei dem zu bearbeitenden Benutzer wurde zuvor von jemand anderem die E-Mailadresse geändert. Der Administrator hat die Änderungen dieses Benutzers noch nicht synchronisiert.

**Testszenario** In der Benutzeransicht wird der zu bearbeitende Benutzer angewählt. Der Name wird geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Übersicht wird der bearbeitete Benutzer mit anderem Namen angezeigt.

#### 13.1.7 TS07: Benutzer ändern mit manuell lösbarem Konflikt

**Use Case** UC01: Benutzer CRU und UC06: Konflikt beheben

**Vorbedingung** Die E-Mailadresse des zu bearbeitenden Benutzers wurde zuvor von jemand anderem bearbeitet. Der Administrator hat die Änderungen dieses Benutzers noch nicht synchronisiert.

**Testszenario** In der Benutzeransicht wird der zu bearbeitende Benutzer angewählt. Der Name und die E-Mailadresse werden geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Der Synchronisation-Event schlug fehl. Der Administrator wird informiert, dass der Benutzer bereits bearbeitet wurde. Die Änderungen des Benutzers wurden nicht festgeschrieben.

#### 13.1.8 TS08: Benutzer inaktivieren

**Use Case** UC01: Benutzer CRU

**Vorbedingung** Es ist mindestens ein Benutzer erfasst und der Administrator ist synchronisiert.

**Testszenario** In der Benutzeransicht wird ein erfasster Benutzer angewählt. Der Status des Benutzers wird geändert auf inaktiv. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Übersicht wird der bearbeitete Benutzer grau (inaktiviert) angezeigt.

#### 13.1.9 TS09: Benutzer inaktivieren mit automatisch lösbarem Konflikt

**Use Case** UC01: Benutzer CRU und UC06: Konflikt beheben

**Vorbedingung** Der zu bearbeitende Benutzer wurde zuvor von jemand anderem inaktiviert. Der Administrator hat diese Änderung noch nicht synchronisiert.

**Testszenario** In der Benutzeransicht wird der zu bearbeitende Benutzer ausgewählt. Der Status wird auf inaktiv gesetzt. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Benutzeransicht wird der bearbeitete Benutzer grau (inaktiviert) angezeigt. Keine Fehlermeldung wird angezeigt.

#### 13.1.10 TS10: Projekt erfassen

**Use Case** UC02: Projekt CRU

**Vorbedingung** Das Projekt mit dem entsprechenden Namen existiert noch nicht und der Projektleiter ist synchronisiert.

**Testszenario** In der Projektansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht wird der Name des Projekts eingegeben. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Übersicht ist ein neues Projekt mit dem eingegebenen Namen erschienen.

#### 13.1.11 TS11: Projekt unvollständig erfassen

**Use Case** UC02: Projekt CRU

**Vorbedingung** Das Projekt mit dem entsprechenden Namen existiert noch nicht und der Projektleiter ist synchronisiert.

**Testszenario** In der Projektansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht wird der Name nicht eingegeben. Danach wird “Save“ angewählt.

**Erwartetes Ergebnis** Dem Projektleiter wird angezeigt, dass der Name ein Pflichtfeld ist. Er kann das Projekt nicht speichern, solange der Name nicht abgefüllt ist.

#### 13.1.12 TS12: Projekt erfassen mit nicht automatisch lösbarem Konflikt

**Use Case** UC02: Projekt CRU und UC06: Konflikt beheben

**Vorbedingung** Ein Projekt mit dem entsprechenden Namen wurde bereits von jemand anderem erstellt. Der Projektleiter hat dieses Projekt noch nicht auf sein Gerät synchronisiert.

**Testszenario** In der Projektansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht wird der Name des Projekts eingegeben. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Der Synchronisation-Event schlug fehl. Der Projektleiter wird in einer Fehlermeldung darauf hingewiesen, dass das Projekt bereits existiert.

#### 13.1.13 TS13: Projekte anzeigen

**Use Case** UC02: Projekt CRU

**Vorbedingung** Es ist mindestens ein Projekt erfasst und der Projektleiter ist synchronisiert.

**Testszenario** In der Navigation wird “Projects“ angewählt.

**Erwartetes Ergebnis** Eine Liste von Projekten mit Namen, Anzahl Issues und einem grünen oder roten Status werden dargestellt.

#### 13.1.14 TS14: Projekt ändern

**Use Case** UC02: Projekt CRU

**Vorbedingung** Es ist mindestens ein Projekt erfasst und der Projektleiter ist synchronisiert.

**Testszenario** In der Projektansicht wird ein erfasstes Projekt angewählt. Der Name des Projekts wird geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Übersicht wird das bearbeitete Projekt mit neuem Namen angezeigt.

#### 13.1.15 TS15: Projekt ändern mit nicht automatisch lösbarem Konflikt

**Use Case** UC02: Projekt CRU und UC06: Konflikt beheben

**Vorbedingung** Das zu bearbeitende Projekt wurde zuvor von jemand anderem bearbeitet. Der Projektleiter hat die Änderungen dieses Projekts noch nicht synchronisiert.

**Testszenario** In der Projektansicht wird das zu bearbeitende Projekt angewählt. Der Name des Projekts wird geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Der Synchronisation-Event schlug fehl. Der Projektleiter wird mit einer Fehlermeldung darauf hingewiesen, dass das Projekt bereits bearbeitet wurde. Die Änderungen des Projekts wurden nicht festgeschrieben.

#### 13.1.16 TS16: Projekt schliessen

**Use Case** UC02: Projekt CRU und UC09: E-Mail versenden

**Vorbedingung** Es ist mindestens ein Projekt erfasst und der Projektleiter ist synchronisiert.

**Testszenario** In der Projektansicht wird ein erfasstes Projekt angewählt. Der Status des Projekts wird auf inaktiv geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Projektansicht wird das bearbeitete Projekt mit einem roten, statt grünen Status (Kreis) angezeigt. Alle aktiven Benutzer, die an offenen Arbeitspaketen dieses Projekts arbeiten, werden mit einem E-Mail über die Schliessung des Projekts informiert. Die Anzahl offener Arbeitspakete zeigt 0 an. Die Arbeitspakete in diesem Projekt haben alle den Status “Closed“.

#### 13.1.17 TS17: Projekt schliessen mit automatisch lösbarem Konflikt

**Use Case** UC02: Projekt CRU und UC06: Konflikt beheben

**Vorbedingung** Das zu bearbeitende Projekt wurde zuvor von jemand anderem inaktiviert. Der Projektleiter hat diese Änderung noch nicht synchronisiert.

**Testszenario** In der Projektansicht wird das zu bearbeitende Projekt angewählt. Der Status des Projekts wird auf inaktiv geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Projektansicht wird das bearbeitete Projekt mit einem roten, statt grünen Status (Kreis) angezeigt. Alle aktiven Benutzer, die an offenen Arbeitspaketen dieses Projekts arbeiten, wurden bereits zuvor mit einem E-Mail über die Schliessung des Projekts informiert. Die Anzahl offener Arbeitspakete zeigt 0 an. Die Arbeitspakete in diesem Projekt haben alle den Status “Closed“. Keine Fehlermeldung wird angezeigt.

#### 13.1.18 TS18: Arbeitspaket erfassen

**Use Case** UC03: Arbeitspaket CRU

**Vorbedingung** Der Benutzer ist synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden Name und Projekt des Arbeitspaketes eingegeben. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Arbeitspaketansicht ist ein neues Arbeitspaket mit dem eingegebenen Namen und Projekt erschienen. Die Nummer wird erst nachdem die Synchronisation durchgeführt wurde sichtbar. Diese ist einzigartig und fortlaufend, also 1 höher als das letzte Issue dieses Projekts.

#### 13.1.19 TS19: Arbeitspaket unvollständig erfassen

**Use Case** UC02: Projekt CRU

**Vorbedingung** Der Benutzer ist synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden der Name und das Projekt nicht eingegeben. Danach wird “Save“ angewählt.

**Erwartetes Ergebnis** Dem Benutzer wird angezeigt, dass der Name und das Projekt Pflichtfelder sind. Er kann das Arbeitspaket nicht speichern, solange diese Daten nicht abgefüllt sind.

#### 13.1.20 TS20: Arbeitspaket erfassen auf inaktives Projekt

**Use Case** UC03: Arbeitspaket CRU

**Vorbedingung** Das Projekt zu dem zu erfassenden Arbeitspaket wurde zuvor von jemand anderem inaktiviert. Der Benutzer hat diese Änderung noch nicht synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden ein Name und das inaktivierte Projekt des Arbeitspakets eingegeben. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Arbeitspaketansicht ist ein neues Arbeitspaket mit dem eingegebenen Namen und Projekt erschienen (weil Project und Issue verschiedene Timelines besitzen). Das Projekt wird inaktiv angezeigt.

#### 13.1.21 TS21: Arbeitspakete anzeigen

**Use Case** UC03: Arbeitspaket CRU

**Vorbedingung** Es ist mindestens ein Arbeitspaket erfasst und der Benutzer ist synchronisiert.

**Testszenario** In der Navigation wird “Issues“ angewählt.

**Erwartetes Ergebnis** Eine Liste von Arbeitspaketen mit Namen, Projekt und den zugeteilten Benutzern werden dargestellt.

#### 13.1.22 TS22: Arbeitspaket ändern

**Use Case** UC03: Arbeitspaket CRU

**Vorbedingung** Es ist mindestens ein Arbeitspaket erfasst und der Benutzer ist synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird ein Arbeitspaket ausgewählt. Der Name und die Beschreibung werden geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Arbeitspaketansicht wird das bearbeitete Arbeitspaket mit einem anderen Namen angezeigt.

#### 13.1.23 TS23: Arbeitspaket ändern mit automatisch lösbarem Konflikt

**Use Case** UC03: Arbeitspaket CRU und UC06: Konflikt beheben

**Vorbedingung** Bei dem zu bearbeitende Arbeitspaket wurde zuvor von jemand anderem die Beschreibung geändert. Der Benutzer hat die Änderungen dieses Arbeitspakets noch nicht synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird ein erfasstes Arbeitspaket ausgewählt. Der Name wird geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Arbeitspaketansicht wird das bearbeitete Arbeitspaket mit anderem Namen angezeigt.

#### 13.1.24 TS24: Arbeitspaket ändern mit manuell lösbarem Konflikt

**Use Case** UC03: Arbeitspaket CRU und UC06: Konflikt beheben

**Vorbedingung** Bei dem zu bearbeitende Arbeitspaket wurde zuvor von jemand anderem die Beschreibung geändert. Der Benutzer hat die Änderungen dieses Arbeitspakets noch nicht synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird ein erfasstes Arbeitspaket ausgewählt. Die Beschreibung wird geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Der Synchronisation-Event schlug fehl. Der Benutzer wird mit einer Fehlermeldung darauf hingewiesen, dass das Arbeitspaket bereits bearbeitet wurde. Die Änderungen des Arbeitspakets wurden nicht festgeschrieben.

#### 13.1.25 TS25: Arbeitspaket (ohne Kinder) schliessen

**Use Case** UC03: Arbeitspaket CRU und UC09: E-Mail versenden

**Vorbedingung** Es ist mindestens ein Arbeitspaket, welches nicht einem anderen Arbeitspaket übergeordnet ist, erfasst und der Benutzer ist synchron mit dem Server.

**Testszenario** In der Arbeitspaketansicht wird das erfasste Arbeitspaket ausgewählt. Der Status wird auf “Geschlossen“ gesetzt Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Arbeitspaketansicht wird das bearbeitete Arbeitspaket grau (inaktiviert) angezeigt. Alle aktiven Benutzer, die diesem Arbeitspaket zugeordnet sind, werden mit einem E-Mail über die Schliessung des Arbeitspakets informiert. Die Anzahl offener Arbeitspakete auf der Projektansicht ist um eins minimiert.

#### 13.1.26 TS26: Arbeitspaket mit Kindern schliessen

**Use Case** UC03: Arbeitspaket CRU und UC09: E-Mail versenden

**Vorbedingung** Es ist mindestens ein Arbeitspaket, welches einem anderen Arbeitspaket übergeordnet ist, erfasst und der Benutzer ist synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird das erfasste Arbeitspaket ausgewählt. Der Status wird auf “Geschlossen“ gesetzt Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Arbeitspaketansicht wird das bearbeitete Arbeitspaket und all seine untergeordneten Kinder grau (inaktiviert) angezeigt. Alle aktiven Benutzer, die diesen Arbeitspaketen zugeordnet sind, werden mit einem E-Mail über die Schliessung dieser Arbeitspakete informiert. Die Anzahl offener Arbeitspakete auf der Projektansicht ist um eins minimiert.

#### 13.1.27 TS27: Arbeitspaket schliessen/öffnen mit automatisch lösbarem Konflikt

**Use Case** UC03: Arbeitspaket CRU und UC06: Konflikt beheben

**Vorbedingung** Das zu bearbeitende Arbeitspaket wurde zuvor von jemand anderem geschlossen. Der Benutzer hat diese Änderung noch nicht synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird das zu bearbeitende Arbeitspaket angewählt. Der Status des Arbeitspaket wird auf “Geschlossen“ geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Der Synchronisation-Event schlug fehl. Keine Fehlermeldung wird angezeigt.

#### 13.1.28 TS28: Arbeitspaket einer Person zuweisen

**Use Case** UC03: Arbeitspaket CRU und UC09: E-Mail versenden

**Vorbedingung** Es ist mindestens ein Arbeitspaket erfasst und der Benutzer ist synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird ein Arbeitspaket ausgewählt. Der / die zugewiesenen Benutzer werden geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Arbeitspaketansicht wird das bearbeitete Arbeitspaket mit anderen zugewiesenen Personen angezeigt. Die neu und ehemalig zugewiesenen Benutzer erhalten jeweils ein E-Mail, welches sie über den Wechsel der Zuständigkeit informiert.

#### 13.1.29 TS29: Arbeitspaket einer Person zuweisen mit automatisch lösbarem Konflikt

**Use Case** UC03: Arbeitspaket CRU, UC09: E-Mail versenden und UC06: Konflikt beheben

**Vorbedingung** Das zu bearbeitende Arbeitspaket erhielt zuvor von jemand anderem einen neuen User zugeordnet. Der Benutzer hat die Änderungen dieses Arbeitspaketes noch nicht synchronisiert.

**Testszenario** In der Arbeitspaketansicht wird dieses Arbeitspaket ausgewählt. Die zugewiesenen Benutzer werden um einen weiteren Benutzer ergänzt. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** In der Arbeitspaketansicht wird das bearbeitete Arbeitspaket mit dem anderen zugewiesenen Benutzer und dem neuen als verantwortliche Personen angezeigt. Der neue und die ehemalig zugewiesenen Benutzer erhalten jeweils ein E-Mail, welches sie über den Wechsel der Zuständigkeit informiert.

#### 13.1.30 TS30: Zeitbuchung erfassen

**Use Case** UC04: Zeitbuchung CRUD

**Vorbedingung** Der Benutzer ist synchronisiert.

**Testszenario** In der Zeitbuchungsansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden das Arbeitspaket, die Start- und Endzeit ausgewählt. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Nachdem das Arbeitspaket ausgewählt wurde, wird das zugehörige Projekt in der Erfassung angezeigt. Nach der Eingabe beider Daten wird automatisch die Dauer errechnet und angezeigt. In der Zeitbuchungsansicht ist eine neue Buchung mit dem eingegebenen Arbeitspaket und Projekt sowie der errechneten Dauer und Startdatum erschienen.

#### 13.1.31 TS31: Zeitbuchung unvollständig erfassen

**Use Case** UC04: Zeitbuchung CRUD

**Vorbedingung** Der Benutzer ist synchronisiert.

**Testszenario** In der Zeitbuchungsansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden keine Daten eingegeben. Danach wird “Save“ angewählt.

**Erwartetes Ergebnis** Dem Benutzer wird angezeigt, dass das Arbeitspaket, Start- und Endzeit der Buchung Pflichtfelder sind. Er kann die Buchung nicht speichern.

#### 13.1.32 TS32: Zeitbuchung auf geschlossenem Arbeitspaket erfassen

**Use Case** UC04: Zeitbuchung CRUD

**Vorbedingung** Das Arbeitspaket zu der zu erfassenden Buchung wurde zuvor von jemand anderem inaktiviert. Der Benutzer hat diese Änderung noch nicht synchronisiert.

**Testszenario** In der Zeitbuchungsansicht wird der Floating Action Button angewählt. In der darauffolgenden Ansicht werden das Arbeitspaket, die Start und Endzeit ausgewählt. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Nachdem das Arbeitspaket ausgewählt wurde, wird das zugehörige Projekt in der Erfassung angezeigt. Nach der Eingabe beider Daten wird automatisch die Dauer errechnet und angezeigt. In der Zeitbuchungsansicht ist eine neue Buchung mit dem eingegebenen Arbeitspaket und Projekt sowie der errechneten Dauer und Startdatum erschienen.

#### 13.1.33 TS33: Zeitbuchung anzeigen

**Use Case** UC04: Zeitbuchung CRUD

**Vorbedingung** Es ist mindestens eine eigene Buchung erfasst und der Benutzer ist synchronisiert.

**Testszenario** In der Navigation wird “Time Bookings“ angewählt.

**Erwartetes Ergebnis** Eine Liste von den eigenen Buchungen mit Arbeitspaket, Projekt, Dauer und dem Startdatum werden dargestellt.

#### 13.1.34 TS34: Zeitbuchung ändern

**Use Case** UC04: Zeitbuchung CRUD

**Vorbedingung** Es ist mindestens eine eigene Buchung erfasst und der Benutzer ist synchronisiert.

**Testszenario** In der Zeitbuchungsansicht wird eine Buchung ausgewählt. Das Issue, das Start- und Enddatum werden geändert. Danach wird “Save“ angewählt und die Synchronisation abgewartet.

**Erwartetes Ergebnis** Nachdem das neue Arbeitspaket ausgewählt wurde, wird das zugehörige Projekt in der Anzeige dargestellt. Nach der Eingabe beider Daten wird automatisch die Dauer neu berechnet und angezeigt. In der Zeitbuchungsansicht ist die bearbeitete Buchung mit dem neuen Arbeitspaket und Projekt sowie der errechneten Dauer und Startdatum angezeigt.

#### 13.1.35 TS35: Zeitbuchung falsch abändern

**Use Case** UC04: Zeitbuchung CRUD

**Vorbedingung** Es ist mindestens eine eigene Buchung erfasst und der Benutzer ist synchronisiert.

**Testszenario** In der Zeitbuchungsansicht wird eine erfasste Buchung ausgewählt. Das Startdatum wird nach dem Enddatum erfasst. Danach wird “Save“ angewählt.

**Erwartetes Ergebnis** Dem Benutzer wird angezeigt, dass die Dauer negativ ist. Er kann die Buchung nicht speichern.

#### 13.1.36 TS36: Zeitbuchung löschen

**Use Case** UC04: Zeitbuchung CRUD

**Vorbedingung** Es ist mindestens eine eigene Buchung erfasst und der Benutzer ist synchronisiert.

**Testszenario** In der Zeitbuchungsansicht wird eine erfasste Buchung nach links oder rechts gewischt. Daraufhin wird die Synchronisation abgewartet.

**Erwartetes Ergebnis** Die Zeitbuchung des Benutzers ist aus der Zeitbuchungsansicht verschwunden.

#### 13.1.37 TS37: Benutzer loggt erfolgreich ein

**Use Case** UC07: Login

**Vorbedingung** Der Benutzer ist noch nicht eingeloggt.

**Testszenario** Es werden der Benutzername und das entsprechende Passwort eingegeben.

**Erwartetes Ergebnis** Der Benutzer gelangt zur Startseite und ist erfolgreich eingeloggt.

#### 13.1.38 TS38: Benutzer gibt falsches Passwort ein

**Use Case** UC07: Login

**Vorbedingung** Der Benutzer ist noch nicht eingeloggt.

**Testszenario** Es werden der Benutzername und ein falsches Passwort eingegeben.

**Erwartetes Ergebnis** Der Benutzer ist nicht eingeloggt und bleibt auf dem Login. Eine Fehlermeldung zeigt ihm an, dass entweder der Benutzername oder das Passwort falsch eingegeben wurden.

#### 13.1.39 TS39: Inaktiver Benutzer loggt ein

**Use Case** UC07: Login

**Vorbedingung** Der Benutzer ist inaktiviert und noch nicht eingeloggt.

**Testszenario** Es werden der Benutzername und das entsprechende Passwort eingegeben.

**Erwartetes Ergebnis** Der Benutzer ist nicht eingeloggt und bleibt auf dem Login. Eine Fehlermeldung zeigt ihm an, dass entweder der Benutzername oder das Passwort falsch eingegeben wurden.

#### 13.1.40 TS40: Benutzer loggt ohne Verbindung ein

**Use Case** UC07: Login

**Vorbedingung** Der Benutzer hat keine Verbindung zum Internet und ist noch nicht eingeloggt.

**Testszenario** Es werden der Benutzername und das entsprechende Passwort eingegeben.

**Erwartetes Ergebnis** Der Benutzer ist nicht eingeloggt und bleibt auf dem Login. Eine Fehlermeldung zeigt ihm an, dass er keine Verbindung zum Server hat.

#### 13.1.41 TS41: Benutzer loggt aus

**Use Case** UC08: Logout

**Vorbedingung** Der Benutzer ist eingeloggt.

**Testszenario** In der Navigation wird “Logout“ angewählt.

**Erwartetes Ergebnis** Der Benutzer wird auf das Login weitergeleitet. Durch das betätigen des “‘Back-Buttons“ gelangt der Benutzer nicht zurück ins App.

#### 13.1.42 TS42: Automatische Synchronisation erfolgreich

**Use Case** UC05: Synchronisieren

**Vorbedingung** Der Benutzer hat offene Events, die noch nicht synchronisiert sind. In der Synchronisationsanzeige werden diese als “Pending“ dargestellt.

**Testszenario** Es wird auf die Synchronisation gewartet, maximal 1 Minute.

**Erwartetes Ergebnis** Die Ausstehenden Änderungen werden auf den Server synchronisiert. Es werden keine weiteren Events angezeigt.

#### 13.1.43 TS43: Manuell ausgelöste Synchronisation erfolgreich

**Use Case** UC05: Synchronisieren

**Vorbedingung** Der Benutzer hat offene Events, die noch nicht synchronisiert sind. In der Synchronisationsanzeige werden diese als “Pending“ dargestellt.

**Testszenario** In der Synchronisationsansicht wird der Floating Action Button angewählt.

**Erwartetes Ergebnis** Die Ausstehenden Änderungen werden auf den Server synchronisiert. Es werden keine weiteren Events angezeigt.

### 13.2 Testresultate

Zu jeder Testspezifikation werden die Testresultate in der Tabelle 92 festgehalten.

| Spezifikation Nr. | Release | Datum | Testperson | Resultat |
|---|---|---|---|---|
| TS01 | app-1.0.0 | 12.05.2019 | Michelle Kunz | ✓ |
| TS02 | app-1.0.0 | 12.05.2019 | Michelle Kunz | ✓ |
| TS03 | app-1.0.0 | 12.05.2019 | Michelle Kunz | Der Event schlägt beim “online“-Benutzer fehl und wird zwei Mal in den Events dargestellt. Beim “offline“-Benutzer wird er einmal erfolgreich dargestellt. |
| TS04 | app-1.0.0 | 12.05.2019 | Michelle Kunz | ✓ |
| TS05 | app-1.0.0 | 12.05.2019 | Michelle Kunz | Nach dem Speichern wird nichts dargestellt. Der Event schlug fehl. |
| TS08 | app-1.0.0 | 12.05.2019 | Michelle Kunz | ✓ |
| TS10 | app-1.0.0 | 12.05.2019 | Michelle Kunz | ✓ |
| TS11 | app-1.0.0 | 12.05.2019 | Michelle Kunz | ✓ |
| TS12 | app-1.0.0 | 12.05.2019 | Michelle Kunz | ✓ |
| TS13 | app-1.0.0 | 12.05.2019 | Michelle Kunz | ✓ |
| TS14 | app-1.0.0 | 12.05.2019 | Michelle Kunz | Der Event schlug fehl. Das Projekt konnte nicht gefunden werden. |
| TS16 | app-1.0.0 | 12.05.2019 | Michelle Kunz | Der Event schlug fehl. Das Projekt konnte nicht gefunden werden. |
| TS03 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS05 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |

| Spezifikation Nr. | Release | Datum | Testperson | Resultat |
|---|---|---|---|---|
| TS06 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS07 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS09 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS14 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS15 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS16 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS17 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS18 | app-1.0.1 | 14.05.2019 | Michelle Kunz | Die Laufnummer wird um 2 statt 1 erhöht. Die Buttons OK und Cancel verwischen bei zu vielen User. |
| TS19 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS20 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS21 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS22 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS23 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS24 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS25 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS26 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS27 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS28 | app-1.0.1 | 14.05.2019 | Michelle Kunz | Es werden zu viel Personen angezeigt. Solche, die gar nicht ausgewählt wurden. |
| TS29 | app-1.0.1 | 14.05.2019 | Michelle Kunz | Man kann keine Benutzer einem Arbeitspaket entziehen. |
| TS30 | app-1.0.1 | 14.05.2019 | Michelle Kunz | Auf einem Benutzer ohne vorherige Zeitbuchungen, erscheint eine neu erfasste Zeitbuchung zweimal. |
| TS31 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS32 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS33 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS34 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS35 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS36 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS37 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS38 | app-1.0.1 | 14.05.2019 | Michelle Kunz | Die Fehlermeldung ist nicht verständlich. |
| TS40 | app-1.0.1 | 14.05.2019 | Michelle Kunz | Der Benutzer kann sich noch einloggen. |
| TS41 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS42 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS43 | app-1.0.1 | 14.05.2019 | Michelle Kunz | ✓ |
| TS18 | app-1.0.2 | 16.05.2019 | Michelle Kunz | ✓ |
| TS28 | app-1.0.2 | 16.05.2019 | Michelle Kunz | ✓ |
| TS29 | app-1.0.2 | 16.05.2019 | Michelle Kunz | ✓ |
| TS30 | app-1.0.2 | 16.05.2019 | Michelle Kunz | ✓ |
| TS38 | app-1.0.2 | 16.05.2019 | Michelle Kunz | ✓ |
| TS39 | app-1.0.2 | 16.05.2019 | Michelle Kunz | ✓ |
| TS40 | app-1.0.2 | 16.05.2019 | Michelle Kunz | ✓ |

Abbildung 92: Testprotokoll der getesteten Testspezifikationen

## Literatur

[AST02] Sachin Agarwal, David Starobinski und Ari Trachtenberg. „On the scalability of data synchronization protocols for PDAs and mobile devices“. In: *IEEE network* 16.4 (2002), S. 22–28.

[BCN16] Susanne Braun, Ralf Carbon und Matthias Naab. „Piloting a mobile-app ecosystem for smart farming“. In: *IEEE Software* 33.4 (2016), S. 9–14.

[GL12] Seth Gilbert und Nancy Lynch. „Perspectives on the CAP Theorem“. In: *Computer* 45.2 (2012), S. 30–36.

[Ijt+10] Royyana M Ijtihadie u. a. „Offline web application and quiz synchronization for e-learning activity for mobile browser“. In: *TENCON 2010-2010 IEEE Region 10 Conference*. IEEE. 2010, S. 2402–2405.

[MS12] Zach McCormick und Douglas C Schmidt. „Data synchronization patterns in mobile application design“. In: *Proceedings of the 19th Conference on Pattern Languages of Programs*. The Hillside Group. 2012, S. 12.

[MTZ03] Yaron Minsky, Ari Trachtenberg und Richard Zippel. „Set reconciliation with nearly optimal communication complexity“. In: *IEEE Transactions on Information Theory* 49.9 (2003), S. 2213–2218.

# Abbildungsverzeichnis

# Glossar

**Application** Eine Application ist ein Programm, welches anhand einer Timeline und deren Events Aktionen auslöst. Diese kann z. B. die Persistierung von Projections oder die Kommunikation mit Externen Systemen sein. 46

**Application Interceptor** Ein Application Interceptor steuert, ab welchem Zeitpunkt die Application bereit für die Verarbeitung von Events ist oder terminiert werden muss. Dies kann z. B. der Verbindungsaufbau zu einer Datenbank sein oder der plötzliche Verbindungsunterbruch. 50

**Dependency Injection** Ein Entwurfsmuster welche die Abhängigkeiten verschiedener Objekte auflöst und die Lebenszeit der Objekte kontrolliert. 36

**Event Aggregate** Ein Event Aggregate gehört zu einer Application und arbeitet die einzelnen Events der Timeline ab. Dabei iteriert der Aggregate durch eine Timeline und merkt sich den zuletzt verarbeiteten Event, damit keiner doppelt verarbeitet wird. 49

**Event Interceptor** Ein Event Interceptor gehört ebenfalls zu einer Application und kann Event Aggregates vor der Ausführung abhalten. Ein Beispiel dazu wäre ein Authentication Event Interceptor, welcher prüft ob der Event ausgeführt werden darf. 50

**Merge Request** Eine Methodik in GitLab für Zusammenarbeit in git. 104

**Module** Aus einem Module wird eine Application erstellt. Ein Module ist eine Zusammenfassung von Funktionalitäten und ermöglicht die Aufteilung der Anwendung in einzelne unabhängige Module. Die Module können aus mehreren Event Aggregate, Event Interceptor, Application Interceptors sowie anderen Modulen bestehen. 48

**Multiversion Concurrency Control** Ein Verfahren, welches möglichst effizient, ohne zu blockieren die Konsistenz der Datenbank bewährt. 16

**Optimistic Concurrency Control** Ein Verfahren, welches Konflikte durch gleichzeitige Schreibvorgänge erkennt. 6

**Single Point of Failure** Ein Punkt im System, welcher durch einen Ausfall das ganze System beeinträchtigt. 22

**Timeline** Die Timeline besteht aus einer geordneten Reihe an Events. 45

**Two Phase Commit** Ein Verfahren, in welchem eine Operation in zwei Phasen unterteilt wird, um die Datenkonsistenz aufrecht zu erhalten. 17

**Unique Constraint** Eine Bedingung, welche erfordert, dass alle vorhandenen Werte einzigartig sind. 17, 19, 23