# Architektur und Implementierung

Das Ziel dieses Kapitels ist es, dem Leser/der Leserin das Verständnis für die architekturspezifischen Zusammenhänge sowie die jeweiligen softwareinternen Abläufe zu vermitteln.

Auf dieser Basis werden die einzelnen Architektur Komponenten, deren Beziehungen zueinander, sowie die Funktionsweise des Gesamtsystems im folgenden Verlauf erläutert und an geeigneter Stelle durch beispielhaften Abläufen von Nutzeraktionen beschrieben.

## Architekturübersicht

Aufgrund der Vorzüge einer konsequenten zentralen Verwaltung des Zustands der Applikation, welche in diesem Rahmen auch mit einer komfortablen Wartbarkeit und den strikt getrennten und auf diese Weise auch eindeutig nachverfolgbaren Datenflüssen und Operationen einhergeht, wurde eine Architektur konstruiert, welche sich an dem Redux-Architekturmuster orientiert (siehe TODO GRUNDLAGEN).

Die folgende Grafik soll in vereinfachter Form verdeutlichen, wie die verschiedenen Komponenten der gewählten Architektur hinsichtlich des Aufbaus und des Datenflusses zueinander in Beziehung stehen. Die detaillierte Beschreibung der einzelnen Komponenten folgt in Kapitel (siehe TODO).

TODO GRAFIK

### State

Der zentral gespeicherte Status der Applikation (State) trägt alle relevanten Informationen in sich, welche den internen Zustand des Systems abbilden. Der Status selbst dient also lediglich als zentrales Datenhalterelement in der Architektur.

### Container

Die Container Komponenten lesen den aktuellen Zustand der Applikation aus dem State ein und bereiten diesen unter Umständen, vor der Übergabe an die Presentation Komponenten, vor. Weiterhin lösen die Container Komponenten im Allgemeinen die Actions aus, welche auf den von dem Benutzer getätigten Interaktionen basieren.

### Presentation

Die Presentation Komponenten sind alleinig für die Visualisierung der gegebenen Informationen sowie für die Entgegennahme der Benutzereingaben zuständig.

### Action

Eine Action Komponente beinhaltet Informationen über die jeweilige auszuführende Operation und dient somit als Identifikationsmittel und Informationspaket, welche von der Middleware und/oder den Reducer Komponenten aufgegriffen und verarbeitet werden.

### Middleware

Die Middleware verarbeitet und/oder manipuliert die in den Actions übermittelten Informationen bevor diese zu den Reducer Komponenten gelangen. Weiterhin können auch in der Middleware selbst Actions ausgelöst werden Siehe (TODO IMPLEMENTIERUNG ACHIEVEMENTS).

### Reducer

Die Reducer Komponenten nehmen die eigentliche Änderung am Zustand vor, indem sie einen völlig neuen Zustand generieren und den vorherigen Zustand überschreiben. Durch die über die Actions erhaltene Informationen weiß die Reducer Komponente welche Aktion der Benutzer ausgeführt hat und folglich auch, welche Information sich an dem aktuell existierenden Zustand der Applikation ändern soll. In diesen Komponenten wird also der aktuelle Zustand in einen neuen Zustand transformiert.

## Beschreibung der Klassen Komponenten

## Model

Die Model Komponenten stellen den aktuellen internen Zustand des Systems dar und enthalten somit in erster Linie die verwendeten Objekt-Klassen.

Die folgenden Klassen sind Teil der Model Komponente:

### Achievement-Klasse

Ein Achievement besitzt die folgenden Attribute:

(TODO LISTING)

Der Titel (title) des Achievements ist der Teil, welcher dem Nutzer als Identifikationsmittel des jeweiligen Achievements angezeigt in wird.

Die Information ob ein Achievement erreicht wurde, wird durch den boolschen Wert des Attributs completed dargestellt.

Der Typ (type), welcher durch ein Enum verdeutlicht wird, stellt die Art des Achievements dar, welcher für den Nutzer vor allem durch eine spezielle Visualisierung, durch zum Beispiel ein Badge kenntlich gemacht werden soll (siehe TODO Konzept).

Im Rahmen der Prototyp Entwicklung wurde diese Visualisierungsmöglichkeit jedoch aus Zeitgründen auf die einfache Unterscheidung von unterschiedlich gefärbten Sternsymbolen reduziert, welche sich jedoch durch einfache Anpassungen der Oberfläche auch durch andere grafische Elemente austauschen lassen.

Weiterhin beinhaltet ein Achievement bestimmte Eigenschaften (properties), welche erfüllt werden müssen, damit das Achievement als *erreicht* bezeichnet werden kann. Ein Achievement wird also nur erreicht, wenn alle seine Properties als erfüllt gelten. TODO (WEIL NUR STING ID NÖTIG)

Die Eigenschaften (properties) werden im Rahmen des flach ausgerichteten Key-Value Stores des States nicht als Objektreferenzen, sondern als IDs in textueller Form gespeichert und identifiziert.

### Property-Klasse

Die bereits angesprochene Property-Klasse enthält die folgenden Attribute:

TODO

Der Name (name) eines Property-Instanz stellt die Bezeichnung dar, welche dem Nutzer als Identifikationsmöglichkeit und im Rahmen der Applikation als Instruktion angezeigt werden kann., weshalb diese Bezeichnung wohlüberlegt gewählt werden sollte.

Der Initialwert (initialValue) beschreibt wie der Name bereits impliziert den Startwert, welcher bei Initialisieren des Objektes angenommen wird.

Ein aktueller Wert (currentValue) stellt den aktuellen Fortschritt der jeweiligen Eigenschaft, welche es zu erfüllen gilt dar.

Das Datenfeld activationValue stellt den Zielwert dar, welcher als zu erfüllen gilt.

Die genannten Felder initialValue, currentValue und activationValue werden durch eine definierte Regel (activiationRule) in Beziehung zueinander gesetzt. Diese Regel kann verschieden ausgeprägt sein um dem Ersteller von Achievements sowie deren Kriterien zur Erfüllung eine gewisse Freiheit zu ermöglichen. Im Rahmen des Projektes wurden drei verschiedene Regeln in Form von konstanten Variablen des Datentyps String aufgestellt:

TODO LISTING

Diese Regeln werden als Werkzeug benutzt, um die Semantik der zu erfüllenden Eigenschaften auf die logische Ebene zu übertragen.

Beispielsweise kann somit überprüft werden, ob der Benutzer bereits drei Meilensteine erstellt hat.

TODO BEISPIEL LISTING

### AppContentLoader-Klasse

Die ContentLoader Klasse stellt die Schnittstelle zum gewählten Git-Repository dar und trägt in diesem Umfang die Aufgabe, die in dem Repository Ordner angelegten JSON-Dateien, welche einen Teil des Inhalts der Applikation darstellen, durch Http-Requests einzulesen und dem System zur weiteren Verarbeitung zur Verfügung zu stellen. Dies wird umgesetzt, indem diese eingelesenen Dateien in lokaler Form auf dem Endgerät gespeichert werden.

Die Klasse besitzt lediglich ein Datenfeld (httpRequest), welches die Http-Anfrage beinhaltet, auf welcher die auszuführenden Operationen der Klasseninstanz beruhen.

### JSONAppContentFile-Klasse

Eine Instanz dieser Klasse, welche von durch eine Objektinstanz der Klasse ContentLoader erzeugt wird, stellt eine aus dem Repository eingelesene JSON-Datei dar und enthält die folgenden Attribute:

TODO LISTING

Der Name (name) der im Repository vorliegenden JSON-Datei wird auch weiterhin als Dateiname in dem lokalen Speicher des Endgeräts verwendet.

Die durch den ContentBuilder ermittelte jeweilige Download Adresse (downloadUrl) der verschiedenen im Repository liegenden Dateien, dient zum Herunterladen des jeweiligen Inhaltes der JSON Dateien.

Der jeweilige Inhalt der Datei wird durch das Datenfeld jsonFileContent repräsentiert, welches durch die Methode Future<String> getJsonContent() durch Ausführung eines Http-Requests initialisiert wird.

### InformationToolContentBuilder-Klasse

Die Hilfsklasse InformationToolContentBuilder dient als Werkzeug, zur Generierung des Inhaltes der „Guide“ Sektion der Applikation siehe (TODO KONZEPT). Die Content-Objekte werden durch die jeweilige JSON-Datei, welche zuvor aus dem Git-Repository ausgelesen und gespeichert wurde, in rekursiver Form generiert. Jedes Content Objekt kann mehrere Content Objekte beinhalten, welche in diesem Rahmen als subsections bezeichnet werden. Dies geschieht so lange, bis ein Content-Objekt keine subsections mehr besitzt. Jedes dieser Content-Objekte enthält weiterhin ein Titel (title) sowie eine Beschreibung (description), welche dem Nutzer im Rahmen der „Guide“ Funktion dargestellt werden (siehe Kapitel TODO PRESENTATIOON).

### Content-Klasse

Eine Objektinstanz der Content Klasse stellt den Inhalt der „Guide“ Funktion dar und enthält die folgenden Attribute:

TODO ATTRIBUTE

Der Titel (title) des Content Objekts ist gleichzusetzen mit einer Überschrift für den jeweiligen Inhalt, welcher durch die Beschreibung (description) ausgedrückt wird.

Der Typ (type) enthält Informationen darüber, welche Art von Darstellung innerhalb des „Guide“ - Tools für die Benutzeroberfläche verwendet werden soll. Dies bietet den Vorteil, dass bei dem Erstellen der JSON Datei für den Inhalt dieser Funktion, keine Kenntnisse über die softwareinterne Lösung vorhanden sein muss, sondern der Ersteller nur vordefinierte Begriffe eintragen muss, welche dann softwareintern den jeweiligen Bildschirminhalt generieren.  
Im Rahmen der Prototypentwicklung sind zwei verschiedene Arten der Darstellung vorgesehen.

Die erste Möglichkeit ist das Erzeugen eines Buttons („ButtonListWidget“), welcher den Titel des Content-Objektes anzeigt und bei antippen auf eine nächste Seite führt und dort die jeweiligen subsections TODO LISTING anzeigt.

Die zweite Möglichkeit ist das Generieren eines ausklappbaren Elements („ExtensionPanelWidget“), welches den Titel des Content-Objektes anzeigt und bei Ausklappen dessen Beschreibung sichtbar wird.

### AppState-Klasse

Die Klasse AppState stellt den aktuellen Status der Applikation dar und trägt somit die meisten der zuvor beschrieben Informationen in sich. Diese Informationen werden im Rahmen der Redux-orientierten Architektur durch das Einsetzen von Key-Value Stores strukturiert und verwaltet.

Jede für den Zustand der Applikation relevante Information ist auf diese Weise an einer zentralen Stelle im System gespeichert. Durch einen lesenden Zugriff können die Oberflächenkomponenten alle benötigten aktuellen Informationen ermitteln und abbilden. Weiterhin kann der Status der Applikation durch das Ausführen von Aktionen des Benutzers, was das Auslösen der sogenannten Actions (siehe REDUX GRUNDLAGEN) mit sich trägt, verändert werden.

TODO MAPS1

Die aktuellen Meilensteine werden durch das Datenfeld currentMilestonres abgebildet. Diese Datenstruktur enthält alle von dem Benutzer angelegten Meilensteine und deren jeweiligen Unteraufgaben.

Die von dem Benutzer der Applikation zu erreichenden Achievements werden in dem Datenfeld achievements gespeichert. Dieses Datenfeld weißt einige Besonderheiten vor, auf die ich im Folgenden hinweisen möchte:

Es handelt sich bei dieser Datenstruktur im Detail um eine HashMap, welche weitere HashMaps beinhaltet.

TODO LISTING Map<String, Map<Achievement>>

Diese HashMap wird mit genau vier verschiedenen konstanten Einträgen initialisiert.

„ALL“ – enthält alle vorhandenen Achievements

„ACHIEVED“ – enthält alle Achievements, welche von dem Benutzer gesammelt wurden

„RECOGNIZED“ – enthält alle gesammelten Achievements, welche von dem Benutzer betrachtet wurden.

„NOT\_RECOGNIZED“ enthält alle gesammelten Achievements, welche von dem Benutzer noch nicht betrachtet wurden

Diese Unterscheidung ist auf die Idee zurückzuführen, die unterschiedlichen Zustände der Achievements bei der Handhabung der Benutzerinteraktion zu unterschiedlicher Zeit zu berücksichtigen. Beispielsweise sollen neu erworbene Achievements, welche von dem Benutzer noch nicht eingesehen worden sind, bis zu dem Moment in dem sie betrachtet wurden, gesondert behandelt werden.

Die zu verwaltenden Eigenschaften zur Erfüllung der Achievements werden in dem Datenfeld properties gespeichert. Dies bietet den Vorteil, dass die Properties, welche von unterschiedlichen Achievements geteilt werden können, an einer zentralen Stelle im State verwaltet werden können.

Die für den Benutzer zu erledigenden Herausforderungen werden in dem Datenfeld challenges verwaltet.

Über den bereits genannten Verwaltungsstrukturen hinaus, beinhaltet die Klasse weiterhin spezielle einzelne Informationen:

Der aktuell, im Rahmen der Bedienung der Applikation, von dem Nutzer ausgewählte Meilenstein wird in dem Datenfeld selectedMilestone gespeichert.

Die beiden Felder begin und end, bezeichnen den von dem Benutzer angegebenen Zeitraum der Bearbeitung der Bachelorarbeit.

### Notifications

Für die Implementierung der Notification wurde ein externes Plugin genutzt <https://pub.dartlang.org/packages/flutter_local_notifications>, da die grundsätzliche Möglichkeit der Handhabung von Notifications durch das Framework Flutter nicht gegeben ist.

Bei der Verwendung des Plugins sind jedoch einige zielplattformabhängige Konfigurationen nötig, welche sich unter dem angegebenen Link nachvollziehen lassen.

Bei der Umsetzung der Notifications wurde sich an den gegebenen Beispielen orientiert, welche für die eigene Umsetzung adaptiert wurden.

Die Verwendung der Notifications wird im Rahmen der prototypischen Implementierung durch spezielle Top-Level Funktionen gelöst, welche durch Aufrufen die jeweiligen Actions senden.

Weitere Informationen zum Ablauf und der Funktionsweise sind in dem Kapitel TODO MIDDLEWARE zu finden.

TODO LISTING

## Actions

Im Rahmen der auf der Anforderungsanalyse basierenden Konzeption der Softwarelösung, wurden verschieden Benutzeraktionen identifiziert, welche sich größtenteils in den sogenannten Actions wiederfinden.

Für jede dieser Actions existiert eine eigene Klasse, welche im folgenden Verlauf beispielhaft an einem für alle Actions repräsentativen Fall erläutert wird.

### EDITMILESTONEACTION

Das gewählte Beispiel bezieht sich auf die Nutzeraktion des Änderns eines Meilensteins:

LISTING EDITMILESTONEACTIOn TODO

Wie in TODO zu sehen ist, enthält die Klasse EditMilestoneAction vier verschiedene Datenfelder.

Der zu ändernde Meilenstein (milestone), sowie die jeweiligen neuen Werte newTitle, newDate und newDescription, welche die zu dem vorhandenen Meilenstein zugehörigen Werte überschreiben sollen.

Es wird deutlich, dass eine solche Klasse außer das Beinhalten der spezifischen Payload-Daten, keine tiefe Funktionalität bieten soll. Die weitere Verarbeitung der Daten wird im Kapitel TODO REDUCER verdeutlicht.

## Reducer

Die verschiedenen Reducer Funktionen werden tragen die eigentliche Aufgabe der Datenmanipulation und Verwaltung. In diesen Komponenten passieren die entscheidenden Änderungen am aktuellen Zustand der Applikation.

### App Reducer

Der App Reducer ist die Top-Funktion der Reducer, welche nach jeder durchgeführten Action einen neuen Zustand der Applikation generiert. Jedes Datenfeld des aktuellen Zustands wird in diesem Rahmen durch das Delegieren an untergeordnete Reducer Funktionen, neu generiert. Dabei wird immer der aktuelle Zustand des Systems sowie die in der Action-Klasse enthaltenen Informationen übergeben.

Im folgenden Verlauf wird dieser Ablauf an einem repräsentativen Beispiel verdeutlicht

### Milestone Reducer

TODO LISTING EDITMILESTONEACTION

Innerhalb dieser Funktion wird überprüft, um welche Action es sich handelt. Sollte es sich um keine der angegebenen Actions handeln, so wird der aktuell existierende Zustand unverändert zurückgegeben. Sollte die Action jedoch übereinstimmen, so wird anschließend die jeweils definierte Folge von Befehlen ausgeführt und das neue Ergebnis zurückgegeben.

## Container

Die Container Komponenten dienen in dieser Redux-orientierten Architektur als Schnittstelle zwischen der Benutzeroberfläche (Presentation) und dem internen Zustand der Applikation (Model). Sie stehen somit einerseits in der direkten lesenden Beziehung zu dem Store, welcher den aktuellen Status der Applikation beinhaltet und andererseits in Beziehung zu den Reducer Komponenten, welche diesen Status durch das Auslösen von Actions schreibend aktualisieren.

Im folgenden Verlauf wird dieser Ablauf an einem repräsentativen Beispiel verdeutlicht

TODO LISTING EDITMILESTONE

Um die für die Presentation Komponente AddEditMilestoneScreen relevanten Informationen zu beziehen, wird die Hilfsklasse \_ViewModel verwendet. Hier werden einerseits die Informationen aus dem Store geladen und weiterhin auch die getätigten Actions gesendet. Letzteres funktioniert durch das übermitteln von Callback Funktionen an die Presentation Komponente.

## Presentation

Die Presentation Komponenten stellen die Schnittstelle zwischen System und dem Benutzer dar. In dieser Hinsicht ist die Aufgabe dieser Komponenten die über die Nutzereigaben erhaltenen Informationen an die jeweiligen Container Komponenten zur Weiterverarbeitung zu vermittelt und den aktuellen Systemzustand für den Anwender geeignet darzustellen. Eine Presentation Komponente erhält alle nötigen Informationen von der jeweiligen übergeordneten Container Komponente und soll nicht direkt mit dem Store interagieren. Ziel ist es, die Darstellung von der direkten Systeminteraktion scharf zu trennen, um Austauschbarkeit und Änderungen an der Benutzeroberfläche zu erleichtern.

Im folgenden Verlauf wird dieser Ablauf an einem repräsentativen Beispiel verdeutlicht

TODO LISTING SUBMIT EDIT

Wie in TODO zu sehen wird die, von der Container Komponente übermittelte Callback Funktion addEditMilestone bei Bestätigen der vom Nutzer eingegebenen Inhalte ausgeführt.

TODO OBERFLÄCHE HIER BESCHREIBEN??? IN KONZEPT NUR HERANGEHENSWEISE/ZIELE?

## Middleware

Im Folgenden wird die eingesetzte Middleware beschrieben, welche im Rahmen der prototypischen Entwicklung der Applikation für die Verwaltung von Push Notifications und das Lokale Speichern und Laden des aktuellen Zustands bei Schließen und Öffnen der Applikation.

### Notification Middleware

TODO LISTING NOTIFICATIONMIDDLEWARE

Wie in TODO zu sehen ist, wird auch hier wie in den Reducer Komponenten innerhalb dieser Middleware eine Abfrage zu der Art der ausgelösten Actions getätigt. Sollte die Bedingung, dass es sich um eine SendAchievementNotificationAction() nicht bestätigen, so wird die eingegangene Action weiter an die Reducer Komponenten geleitet. Falls sich die Bedingung jedoch als wahr herausstellt, so wird der darauffolgende Ablauf von Befehlen ausgeführt, welcher sich hier auf das Auslösen einer Notification bezieht.

Eine Besonderheit innerhalb dieser Komponente ist die Verwendung eines Completers, welcher einen Teil der Action Payload darstellt.

Hinweis: Ein Completer ist eine von Dart gegebene Klasse, welche es in diesem Fall ermöglicht, eine zukünftig ausgeführte Funktion so zu behandeln, als wären der Rückgabewert bereits bekannt. TODO https://api.dartlang.org/stable/2.0.0/dart-async/Completer-class.html

In diesem Rahmen wird nach dem Ausführen von Actions, das Senden einer Notification durch das aufrufen der Top-Level Funktion checkAchievementNotification() „vorbereitet“.

Sollten folglich die Bedingungen für das Ausführen dieser Notification zutreffen, so wird wie in TODO zu sehen ist, die „versprochene“ Aktion durch action.completer.complete() ausgeführt und in diesem Fall die Notification angezeigt.

TODO checkAchievementNotification() LISTING

Um zu gewährleisten, dass neue Achievements registriert werden, wird vor Ausführung der SendAchievementNotificationAction in TODO ABBILDUNG die CheckForAchieveAction() ausgelößt, welche das Aktualisieren der aktuellen Achievements nach sich zieht.

### Persistor Middleware

Um das Speichern und Laden des jeweiligen Systemzustands bei Schließen und Öffnen der Applikation zu gewährleisten, wurde das für diese Absicht vorhandene externes Paket redux\_persist\_flutter verwendet.