Stundenplan–App für iOS

Dokumentation, Spezifikation, Konstruktion

Daniel Glaser, Stefan Scharrer, Daniel Zizer,
Sebastian Fuhrmann, Paul Forstern, Andreas Wolf, Kevin Rodd

06.07.2017

Inhaltsverzeichnis

1 Architektur

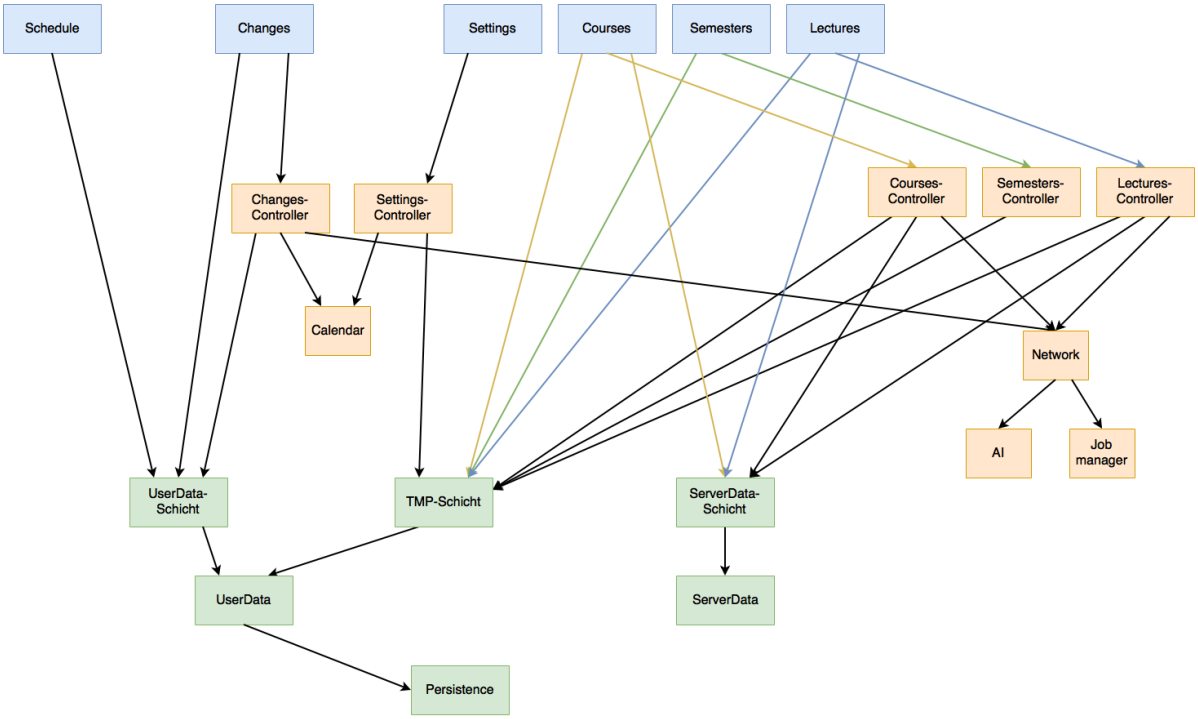

Abbildung 1.1: Ablauf des Background Fetch

In der obigen Abbildung ist die Gesamtarchitektur der Stundenplan-App zu sehen. Folgend wird das Diagramm weiter erläutert.

- Blau: View
- Orange: Controller
- Grün: Model

Die Farben der Pfeilen dienen nur der Übersichtlichkeit.

2 Daten und Zugrifsschichten

2.1 Einleitung

Benutzer und App-Daten werden in zwei verschiedenen Datentöpfen gespeichert. Auf bestimmte Inhalte soll nur durch die Zugriffsschichten zugegriffen werden. Im Folgenden werden die Daten und die zugehörigen Zugriffsschichten aufgelistet.

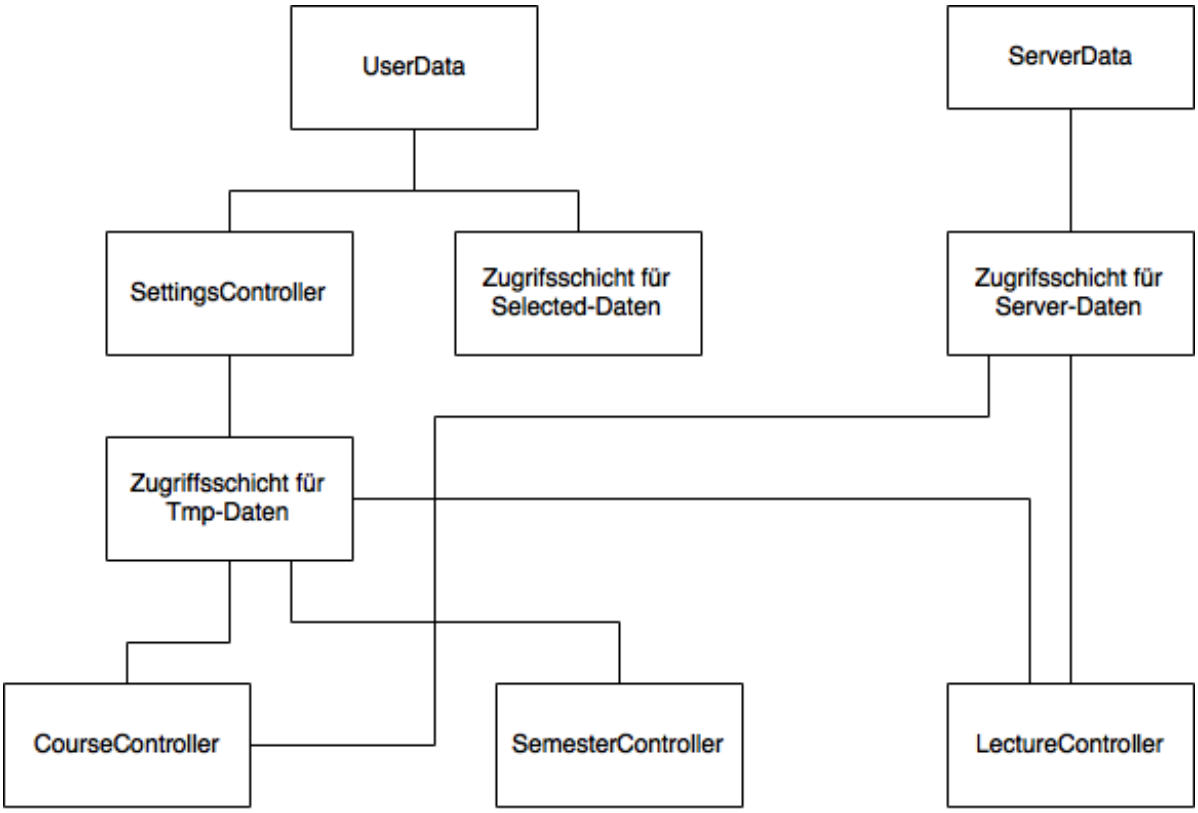

Abbildung 2.1: Daten und zugehörige Zugrifsschichten

Folgende Datentöpfe werden genutzt:

- UserData
- ServerData

Folgende Zugriffsschichten sind vorhanden:

- SelectedCourse
- SelectedSemester
- SelectedLectures
- AllChanges
- AllCourses
- AllLectures

2.2 UserData

Alle Daten mit relevanten Informationen über den Nutzer werden in UserData gespeichert. Mit den Zugriffsschichten SelectedLectures, SelectedSemesters, SelectedCourses und AllChanges kann auf die Daten zugegriffen werden, die vom User gespeichert wurden. Diese Schichten haben nur einen lesenden Zugriff und können keine Daten manipulieren.

Für das Ändern der Daten ist die Tmp Zugriffsschicht zuständig. Die Tmp Zugriffsschichten, welche TmpSelectedLectures, TmpSelectedSemesters und TmpSelectedCourses beinhalten, arbeiten mit einer eigenen Kopie von UserData. Dadurch können Veränderungen einfach wieder verworfen werden, da keine gespeicherten Daten verändert werden. Diese Zugriffsschichten werden nur in den Einstellungen verwendet, um u. a. die ausgewählten Studiengänge, Semester und Vorlesungen festzuhalten. Erst beim Bestätigen der Eingaben werden die Daten aus der Kopie in das original UserData übernommen. Die Kopie von UserData wird vom SettingsController erzeugt und verwaltet.

Im Folgenden wird der Inhalt von UserData und die zugehörigen Selected und Tmp Zugriffsschichten beschrieben.

In UserData sind folgende Sachen enthalten:

- gewählte Season (Sommer- oder Wintersemester)
- gewählte Studiengänge
- gewählte Semester
- gewählte Vorlesungen
- Stundenplanänderungen für gewählte Vorlesungen

Selected-Zugrifsschichten:

- SelectedCourse: Ist für alle vom User selektierten Studiengänge zuständig
- SelectedSemester: Ist für alle vom User selektierten Semester zuständig
- SelectedLectures: Ist für alle vom User selektierten Vorlesungen zuständig
- AllChanges: Ist für Änderungen von gewählte Vorlesungen zuständig

Tmp-Zugrifsschichten:

- TmpCourse: Ist für alle vom User selektierten Studiengänge zuständig
- TmpSemester: Ist für alle vom User selektierten Semester zuständig
- TmpLectures: Ist für alle vom User selektierten Vorlesungen zuständig
- AllChanges: Ist für Änderungen von gewählte Vorlesungen zuständig

2.3 ServerData

Alle Daten, die vom Server geladen wurden, werden in ServerData gespeichert. Durch die Zugriffsschichten AllLectures und AllCourses können die Daten abgefragt werden.

Folgend wird über die in ServerData enthaltenen Daten und die zugehörigen Zugriffsschichten informiert.

In ServerData sind folgende Sachen enthalten:

- Alle Studiengänge + Semester
- Alle Vorlesungen für ausgewählte Studiengänge + Semester

Zugriffsschichten:

- AllCourses: Ist für alle geladenen Studiengänge + Semester zuständig
- AllLectures: Ist für alle geladenen Vorlesungen für Studiengänge + Semester zuständig

3 Hintergrundaktualisierung

3.1 Übersicht

Die App soll im Hintergrund regelmäßig prüfen, ob neue Stundenplanänderungen für den Benutzer entstanden sind. Daher wurde der Background Fetch implementiert. In diesem Sequenzdiagramm wird der Ablauf der application(performFetchWithCompletionHandler) Methode dargestellt.

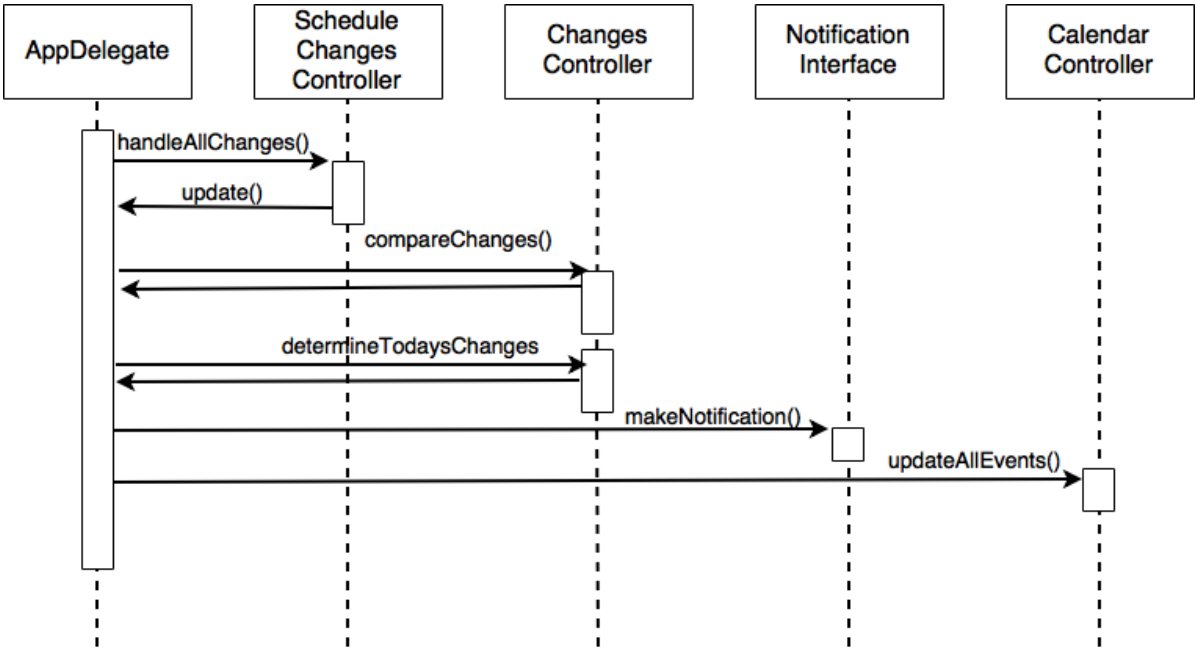

Abbildung 3.1: Ablauf des Background Fetch

Zu Beginn werden durch den ScheduleChangesController die aktuellen Stundenplanänderungen vom Server geladen. Anschließend werden mit dem ChangesController die neu dazugekommenen Änderungen ermittelt, da die aktuelle Liste mit Stundenplanänderungen mit der vorher zuletzt geladenen Liste verglichen wird. Zudem ermittelt der ChangesController die Änderungen, welche am aktuellen Tag stattfinden. Als Nächstes wird über das NotificationInterface eine lokale Notification für den User erstellt, die ihn über die neu entstandenen Stundenplanänderungen informiert. Zuletzt wird über den CalendarController der iOS Kalender aktualisiert.

4 Calendar Interface

4.1 Einleitung

Im folgenden Diagramm ist die Architektur und Integration des Calendar Interface in die Stundenplan App dargestellt.

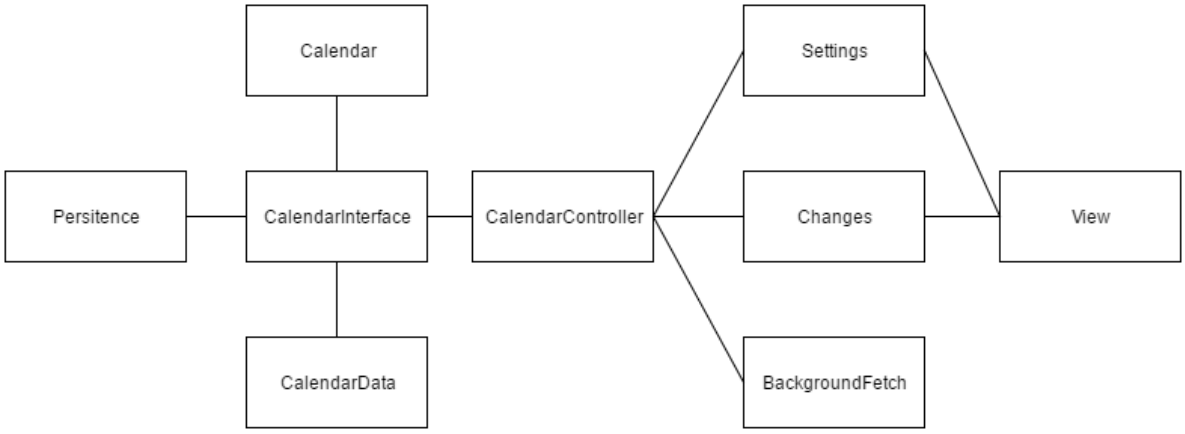

Abbildung 4.1: CalendarInterface Diagramm

Das Calendar Interface besteht aus folgenden drei Klassen:

- CalendarController
- CalendarInterface
- CalendarData

Im Folgenden wird auf die Klassen genauer eingegangen.

4.2 CalendarController

Der CalendarController ist der Controller für das Calendar Interface. Über ihn wird auf das Interface zugegriffen.

Im Folgenden wird auf die wichtigsten Methoden eingegangen:

- createCalendar() -> EKAuthorizationStatus
- removeCalendar()
- createAllEvents(lectures : [Lecture])
- updateAllEvents(changes : [ChangedLecture])
- removeAllEvents(lectures : [Lecture])
- CalendarRoutine() -> Bool

Vor dem Ausführen der Aufgabe der jeweiligen Methode wird überprüft, ob der Benutzer die Berechtigung auf den Kalender zuzugreifen gewährt hat.

4.2.1 createCalendar() -> EKAuthorizationStatus

Erzeugt den Kalender falls die Berechtigung vorhanden ist. Falls er erzeugt wurde werden die Events mit createAllEvents in den Kalender geschrieben. Als Rückgabewert gibt er den Berechtigungsstatus zurück. Es gibt drei Rückgabewerte:

- authorized

  Bedeutet, dass der Nutzer die Berechtigung erteilt hat und der Kalender angelegt wurde oder bereits angelegt war.

- notDetermined

  Bedeutet, dass die Berechtigung gerade abgefragt wird.

- denied

  Bedeutet, dass die Berechtigung verweigert wurden.

4.2.2 removeCalendar()

Löscht den Kalender und damit alle Einträge die darin vorhanden sind mit Hilfe des KalendarInterface.

4.2.3 createAllEvents(lectures : [Lecture])

Erzeugt aus den Vorlesungen die Events, die anschließend mit Hilfe des CalendarInterface in den Kalender geschrieben werden. Falls der Kalender noch nicht erzeugt wurde, wird er angelegt. Die EventID's werden mittels der saveIDs-Methode des CalendarInterface persistent gespeichert.

4.2.4 updateAllEvents(changes : [ChangedLecture])

Ermittelt welche Art von Änderung vorliegt und passt mit Hilfe des CalendarInterface die entsprechend Events im Kalender an. Die EventID's werden mittels der saveIDs-Methode des CalendarInterface persistent gespeichert.

4.2.5 removeAllEvents(lectures : [Lecture])

Holt sich für die übergebenden Vorlesungen die EventID's und löscht die entsprechenden Events mit Hilfe des CalendarInterface. Die Änderungen an den EventID's werden mittels der saveIDs-Methode des CalendarInterface persistent gespeichert.

4.2.6 CalendarRoutine() -> Bool

Aktualisiert die Vorlesungen im Kalender. Dazu holt sie sich die abgewählten und neu ausgewählten Vorlesungen und übergibt sie der removeAllEvents() oder createAllEvents()-Methode. Der Rückgabewert gibt an, ob der Benutzer die Berechtigung erteilt hat, in den Kalendar zu schreiben.

4.3 CalendarInterface

Dies ist die Klasse die direkt auf den Kalender zugreift.

Im Folgenden wird auf die wichtigsten Methoden eingegangen:

- createCalenderIfNeeded() -> Bool
- removeCalendar() -> Bool
- createEvent(p_event : EKEvent, key : String, isChanges : Bool)
- updateEvent(eventID : String, updatedEvent : EKEvent, key : String, lectureToChange : Bool)
- removeEvent(p_eventId: String, p_withNotes: Bool?=false) -> Bool
- saveIDs()

4.3.1 createCalenderIfNeeded() -> Bool

Erzeugt den Kalender falls er nicht bereits vorhanden ist. Gibt einen Boolean-Wert zurück, ob der Kalender angelegt wurde.

4.3.2 removeCalendar() -> Bool

Löscht den Kalender und damit alle Einträge, die er beinhaltet falls dieser vorhanden ist. Gibt einen Boolean-Wert zurück, ob das Löschen erfolgreich war.

4.3.3 createEvent(p_event : EKEvent, key : String, isChanges : Bool)

Schreibt das übergebende Event in den Kalender. Unterscheidet dabei, ob es sich um eine Änderung oder eine normale Vorlesung handelt und fügt die EventID dem entsprechenden Dictonary hinzu.

4.3.4 updateEvent(eventID : String, updatedEvent : EKEvent, key : String, lectureToChange : Bool)

Das zu der EventID dazugehörige Event wird mit den entsprechend Werten des übergebenen Event angepasst. Durch den lectureToChange Boolean-Wert wird unterschieden, ob aus einer Vorlesung eine Änderung wird oder ob aus einer Änderung wieder eine Vorlesung wird. Dabei werden die EventsIDs in das entsprechende Dictonary verschoben.

4.3.5 removeEvent(p_eventId: String, p_withNotes: Bool?=false) -> Bool

Löscht das übergebene Event. Gibt einen Boolean-Wert zurück, ob das Löschen erfolgreich war.

4.3.6 saveIDs()

Speichert die EventIDs der Vorlesungen und Änderungen in CalendarData persistent.

4.4 CalendarData

Die CaledenarData-Klasse dient dazu, die EventIDs der Vorlesungen und Änderungen getrennt in Dictonarys zu speichern. Die Dictonarys bestehen aus einer Zuordnung von einer ID einer Vorlesung zu mehreren EventIDs.

5 Date Extension

Die DateExtension enthält Methoden zur Erweiterung der Date Klasse.

Im Folgenden werden die wichtigsten Methoden der Klasse aufgelistet.

- checkSemester() -> String
- startSemester(semester : String) -> Date
- endSemester(semester : String) -> Date
- startLecture(startDate: Date, weekdayString : String, semester : String) -> Date
- endLecture (endDate: Date, weekdayString : String, semester : String) -> Date
- calendarweekToDate(day: String, cw: Int, date: Date) -> Date
- combineDateAndTime(date: Date, time: Date) -> Date
- changeStartFromTwoWeekLecture()

5.1 checkSemester() -> String

Überprüft und gibt danach einen String zurück, ob das momentane Semester ein Sommersemester oder Wintersemester ist.

5.2 startSemester(semester : String) -> Date

Gibt das Startdatum des aktuellen Semesters zurück. Wenn das Startdatum im Sommersemester (15.03) oder im Wintersemester (01.10) ein Freitag, Samstag oder Sonntag ist, beginnt die Vorlesung am nächstfolgenden Montag.

5.3 endSemester(semester : String) -> Date

Gibt das Enddatum des aktuellen Semesters zurück. Wenn das Enddatum im Sommersemester (10.07) oder im Wintersemester (25.01) ein Samstag, Sonntag oder Montag ist endet, die Vorlesung am vorausgehenden Freitag.

5.4 startLecture(startDate: Date, weekdayString : String, semester : String) -> Date

Gibt das Startdatum der Vorlesung zurück. Das Startdatum wird mittels des Starts des Semesters und dem Wochentag der Vorlesung ermittelt.

5.5 endLecture (endDate: Date, weekdayString : String, semester : String) -> Date

Gibt das Enddatum der Vorlesung zurück. Das Enddatum wird mittels des Endes des Semesters und dem Wochentag der Vorlesung ermittelt.

5.6 calendarweekToDate(day: String, cw: Int, date: Date) -> Date

Ermittelt aus einer Kalenderwoche und dem dazugehörigen Wochentag ein Datum.

5.7 combineDateAndTime(date: Date, time: Date) -> Date

Kombiniert das Datum und die Uhrzeit von zwei verschiedenen Daten.

5.8 changeStartFromTwoWeekLecture(startDate: Date, semester: String) -> Date

Ändert das Startdatum einer Vorlesung die alle zwei Wochen stattfindet.

6 Artificial Intelligence

Die Artificial Intelligence enthält Methoden, die das Kommentar der jeweiligen Vorlesung überprüft. Falls es im Kommentar enthalten ist, erkennt sie ob die Vorlesung ein Blocktermin ist, alle zwei Wochen stattfindet, eine Aufzählung von Terminen enthält, nicht parsbar ist, wann die Vorlesung beginnt oder wann sie endet. Die Artificial Intelligence wird direkt aufgerufen, nachdem die Vorlesung in der Klasse JsonLecture vom Server abgefragt werden.

Im Folgenden werden die wichtigsten Methoden der Klasse aufgelistet.

- iterationOfLecture(comment: String, start: Date, end: Date) -> iterationState
- containsEnumeration(comment: String) -> Bool
- checkPeriod(comment: String) -> (String, String)
- checkStart(comment: String) -> String
- checkEnd(comment: String) -> String
- getNextCalendarWeekOrDate(comment: String, keyword: String, length: Int) -> String
- getPreviousCalendarWeekOrDate(comment: String, keyword: String, length: Int) -> String

6.1 iterationOfLecture(comment: String, start: Date, end: Date) -> iterationState

Gibt die verschiedenen Arten der Wiederholung einer Vorlesung zurück. Es gibt fünf verschiedene Rückgabewerte.

- iterationState.notParsable

  Bedeutet, dass das Kommentar Wörter enthält die das Programm nicht parsen kann.

- iterationState.calendarWeeks

  Bedeutet, dass das Kommentar eine Aufzählung von Terminen enthält.

- iterationState.individualDate

  Bedeutet, dass die Vorlesung ein Einzeltermin ist.

- iterationState.twoWeeks

  Bedeutet, dass die Vorlesung alle zwei Wochen stattfindet.

- iterationState.weekly

  Bedeutet, dass die Vorlesung jede Wochen stattfindet.

6.2 containsEnumeration(comment: String) -> Bool

Falls das Kommentar eine Aufzählung von Terminen enthält gibt die Methode diese zurück.

6.3 checkPeriod(comment: String) -> (String, String)

Durchsucht das Kommentar nach einer Zeitspanne. Ein Beispiel so einer Zeitspanne ist "KW 12 bis 26", dabei wird die erste Kalenderwoche als Starttermin und die zweite als Endtermin der Vorlesung genommen. Vorerst sucht es jedoch nach Zeitspannen die nichts mit den Vorlesungsterminen zu tun haben, beispielsweise Önline-Anmeldung 09.03. - 16.03.", und entfernt diese temporär aus dem Kommentar.

6.4 checkStart(comment: String) -> String

Durchsucht das Kommentar nach einem neuen Startdatum der Vorlesung.

6.5 checkEnd(comment: String) -> String

Durchsucht das Kommentar nach einem neuen Enddatum der Vorlesung.

6.6 getNextCalendarWeekOrDate(comment: String, keyword: String, length: Int) -> String

Gibt das nächste Datum oder die nächste Kalenderwoche nach dem angegebenen Keyword zurück. Length gibt dabei die Länge an wie viel Zeichen die Methode nach dem Keyword suchen soll, bevor sie die Suche nach einem Datum oder einer Kalenderwoche abbricht.

6.7 getPreviousCalendarWeekOrDate(comment: String, keyword: String, length: Int) -> String

Gibt das vorrausgehende Datum oder die vorrausgehende Kalenderwoche nach dem angegebenen Keyword zurück. Length gibt dabei an, welche Anzahl an Zeichen die Methode vor dem Keyword suchen soll, bevor sie die Suche nach einem Datum oder einer Kalenderwoche abbricht.

7 Jobmanager

7.1 Einleitung

Im folgenden Dokument, wird auf die Funktionsweise des JobManagers in der iOS Stundenplan App der Hochschule Hof näher eingegangen.
Innerhalb der Applikation verwaltet der JobManager alle Jobanfragen und delegiert die Aufgaben asynchron an die jeweils unterliegende Schicht.

Jede Instanz des JobMangers verwaltet seine eigene Warteschlange (Queue). In dieser Warteschlange können sich n Workitems befinden. Jedes Workitem stellt eine Netzwerkanfrage (einen Job) dar. Ergebnisse werden in der Warteschlange gesammelt und sobald alle Ergebnisse vorliegen, werden diese an den Auftraggeber weitergegeben.
Eine Abbruchfunktion für noch offene Jobs lässt den Vorgang jederzeit unterbrechen.

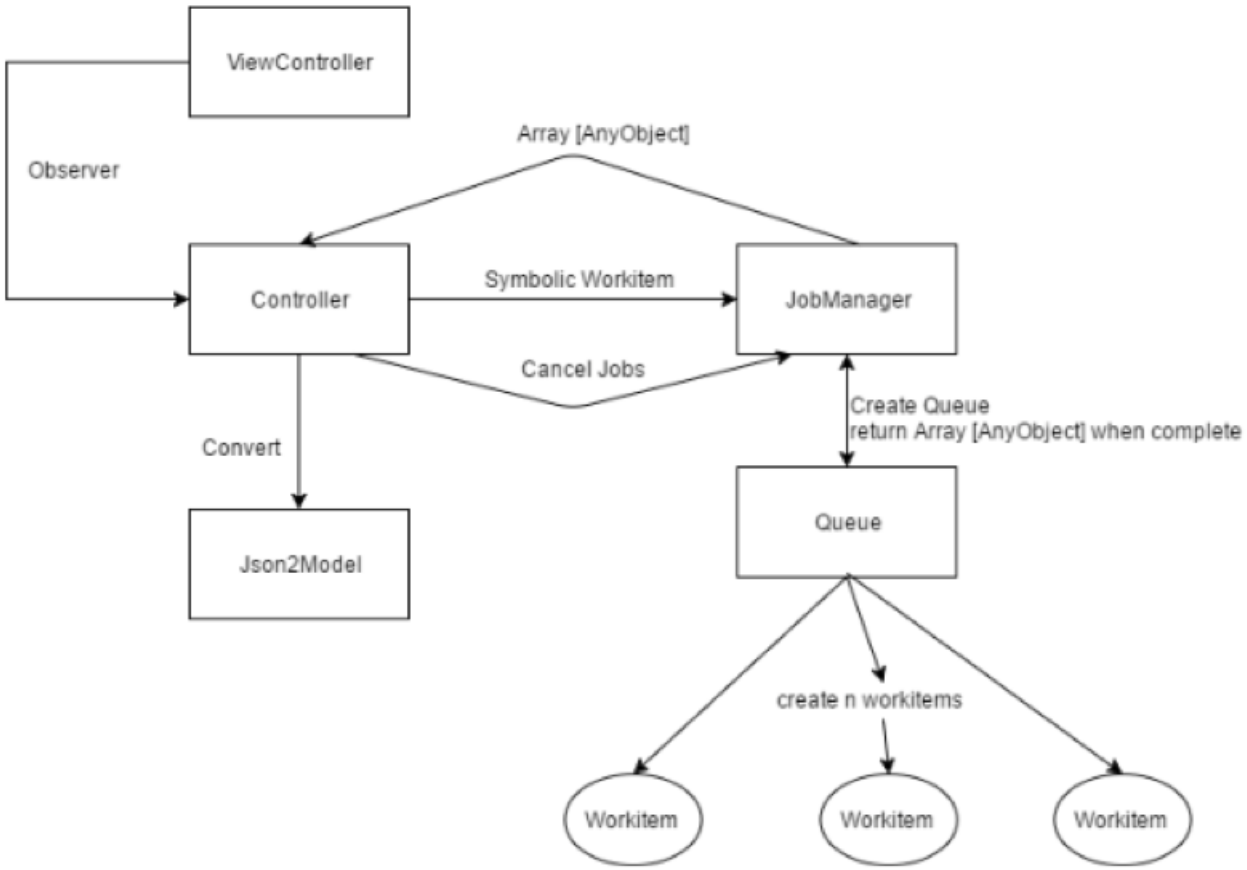

Abbildung 7.1: Architektur und Integration des JobManagers in die Stundenplan App

7.2 Ablauf

Der Ablauf des JobManagers und die Verwendung für den Benutzer lässt sich in folgende Unterpunkte unterteilen:

7.2.1 Erstellen des JobManagers

Der JobManager wird ohne speziellen Konstruktor erstellt. Dazu reicht ein
var myJobManager : JobManager = JobManager().

7.2.2 Observer anmelden

Ebenso ist das Anmelden des Observers schnell erledigt. Hierzu muss der jeweilige Controller lediglich das *DataObserverProtocol* implementieren und mittels

self.myJobManager.addNewObserver(o: self)

sich beim JobManager als Observer anmelden.

7.2.3 Anfrage

Wurde der JobManager erstellt und man hat sich als Observer angemeldet, so kann man eine neue Jobanfrage an den JobManager stellen. Eine Anfrage könnte so aussehen:

myJobManager.NetworkJob(url: urlString, username: myUsername, password: myPassword)

Hier ist zu beachten, dass das einfügen eines Usernames und eines Passwortes optionale Parameter sind.

7.2.4 Ergebnis

Sind alle Jobs erfolgreich beendet worden, so sendet der JobManager die Daten als *AnyObject* an die update(o: AnyObject) Methode des Oberservers.

7.2.5 Aktuelle Version

In der aktuellen Version ist es möglich den JobManger mehrmals zu verwenden. Eine neue Instanziierung ist nicht notwendig, wird jedoch für einen sauberen Ablauf empfohlen.