

# **Stundenplan–App für iOS**

## **Dokumentation, Spezifikation, Konstruktion**

Daniel Glaser, Stefan Scharrer, Daniel Zizer, Sebastian Fuhrmann

06.07.2017

# Inhaltsverzeichnis

|          |                                    |           |
|----------|------------------------------------|-----------|
| <b>1</b> | <b>Calendar Interface</b>          | <b>1</b>  |
| 1.1      | Einleitung . . . . .               | 1         |
| 1.2      | CalendarController . . . . .       | 2         |
| 1.2.1    | createCalendar . . . . .           | 2         |
| 1.2.2    | removeCalendar . . . . .           | 2         |
| 1.2.3    | createAllEvents . . . . .          | 2         |
| 1.2.4    | updateAllEvents . . . . .          | 3         |
| 1.2.5    | removeAllEvents . . . . .          | 3         |
| 1.2.6    | CalendarRoutine . . . . .          | 3         |
| 1.3      | CalendarInterface . . . . .        | 4         |
| 1.3.1    | createCalenderIfNeeded . . . . .   | 4         |
| 1.3.2    | removeCalendar . . . . .           | 4         |
| 1.3.3    | createEvent . . . . .              | 4         |
| 1.3.4    | updateEvent . . . . .              | 4         |
| 1.3.5    | removeEvent . . . . .              | 4         |
| 1.3.6    | saveIDs . . . . .                  | 4         |
| 1.4      | CalendarData . . . . .             | 5         |
| <b>2</b> | <b>Daten und Zugriffsschichten</b> | <b>6</b>  |
| 2.1      | Einleitung . . . . .               | 6         |
| 2.2      | UserData . . . . .                 | 8         |
| 2.3      | ServerData . . . . .               | 9         |
| <b>3</b> | <b>Hintergrundaktualisierung</b>   | <b>10</b> |
| 3.1      | Übersicht . . . . .                | 10        |

# 1 Calendar Interface

## 1.1 Einleitung

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

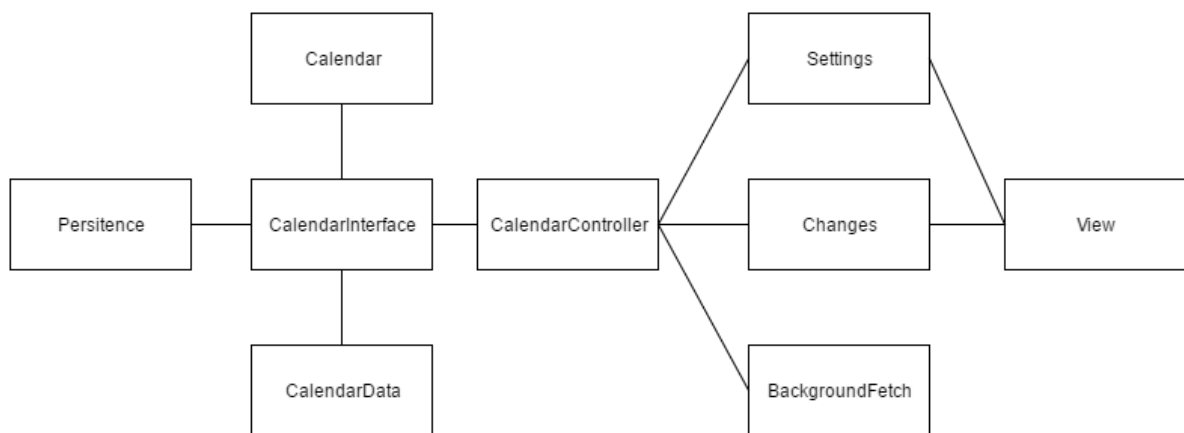


Abbildung 1.1: CalendarInterface Diagramm

Das Calendar Interface besteht aus folgenden drei Klassen:

- CalendarController
- CalendarInterface
- CalendarData

Im Folgenden wird auf die Klassen genauer eingegangen.

## 1.2 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.

### 1.2.1 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.

### 1.2.2 removeCalendar()

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

### 1.2.3 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.

### 1.2.4 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.

### 1.2.5 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.

### 1.2.6 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 Kalender zu schreiben.

## 1.3 CalendarInterface

Dies ist die Klasse die direkt auf den Kalender zugreift.

Im Folgenden wird auf die wichtigsten Methoden eingegangen:

- `createCalendarIfNeeded()` -> 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()`

### 1.3.1 createCalendarIfNeeded() -> Bool

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

### 1.3.2 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.

### 1.3.3 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 Dictionary hinzu.

### 1.3.4 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 Dictionary verschoben.

### 1.3.5 removeEvent(p\_eventId: String, p\_withNotes: Bool?=false) -> Bool

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

### 1.3.6 saveIDs()

Speichert die EventIDs der Vorlesungen und Änderungen in CalendarData persistent.

## 1.4 CalendarData

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

## 2 Daten und Zugriffsschichten

### 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.

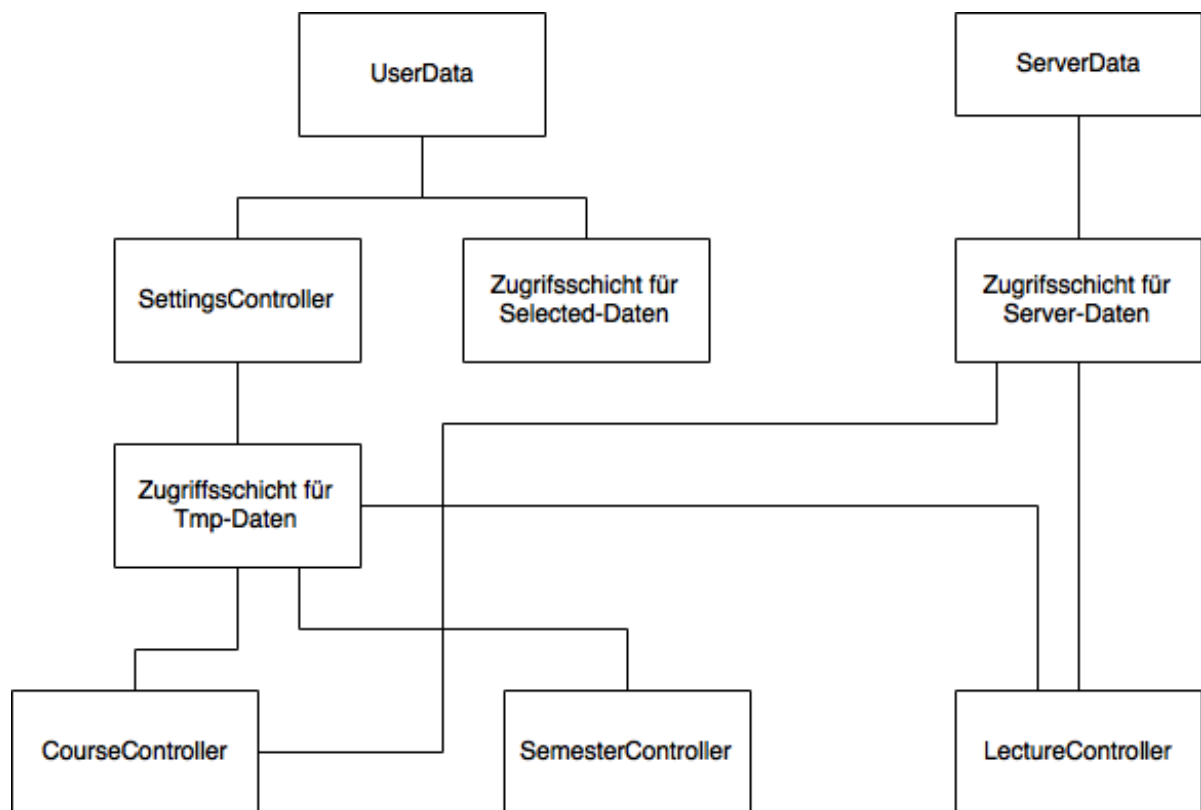


Abbildung 2.1: Daten und zugehörige Zugriffsschichten



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-Zugriffsschichten:

- 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-Zugriffsschichten:

- 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(performanceFetchWithCompletionHandler)` Methode dargestellt.

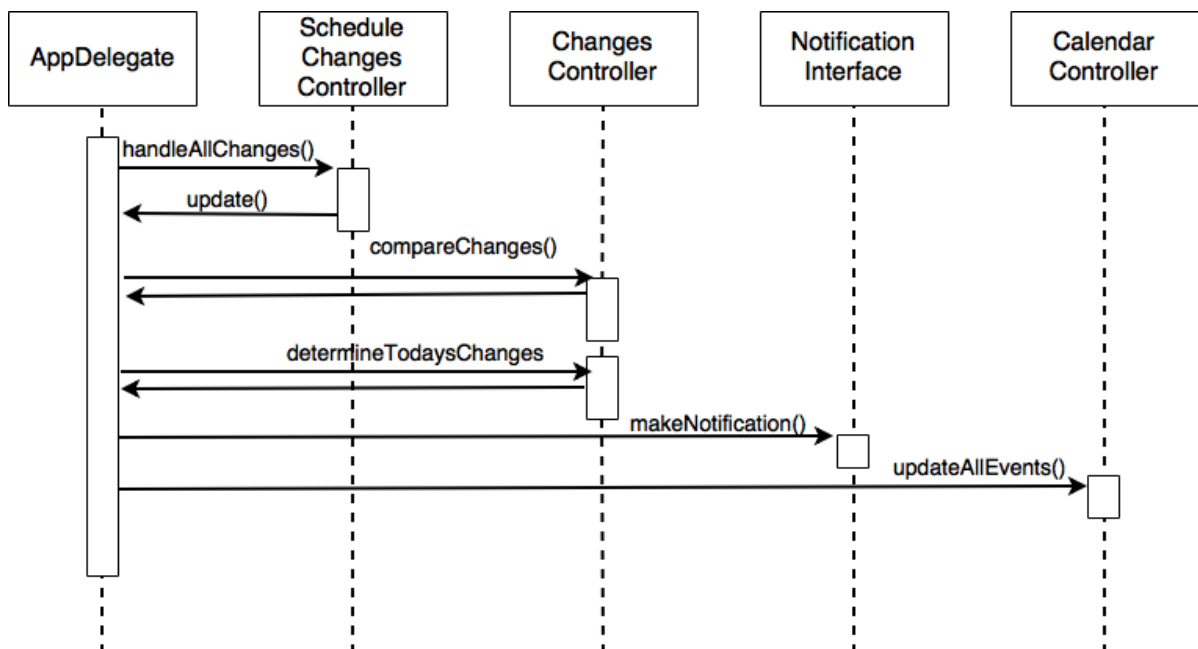


Abbildung 3.1: Ablauf des Background Fetch

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.