Bachelor Thesis

Spektrometer App

Anbindung Spektrometer an mobiles Device

Andreas Lüscher, Raphael Bolliger

Fachhochschule Nordwestschweiz
Hochschule für Technik

Dozent: Martin Gwerder
Auftraggeber: Andreas Hueni

Windisch, 6. März 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Zusammenfassung

Das Hauptziel des Projektes, ist es eine mobile Applikation zu erstellen, welche die RS3 Desktoplösung von ASD ablöst. RS3 ist eine Software, welche die Verbindung zu einem Spektralmessgerät herstellt, Messungen auslösen sowie die Resultate anzeigen kann. Das bisherige System besteht aus einem Laptop inklusive RS3 Software, welche jeweils auf ein Spektrometer abgestimmt ist. Neu soll eine mobile App ausreichen, um mehrere Spektrometer ansprechen zu können. Die App soll Forschende unterstützen, Messungen direkt vor Ort zu beurteilen und verwalten zu können.

1 Einleitung

1.1 Ziel der Arbeit

Das geologische Institut der Universität Zürich betreibt zur Forschung vier Spektroradiometer der Firma ASD Inc. aus Colorado in den USA. Zu jedem Spektrometer liefert ASD einen Notebook mit installierter Software um das Spektrometer zu bedienen und Messungen auszuführen. Ziel dieser Arbeit ist es die Software RS3 von ASD mit einer modernen Applikation für ein mobiles Device abzulösen. Das Projektteam hat sich gemeinsam mit dem Kunden dazu entschieden die Applikation für iOS Geräte zu entwickeln.

1.2 Hilfestellungen

Zur Umsetzung konnten verschiedenen Hilfestellungen in Anspruch genommen werden. ASD bietet auf der ihrer Webseite einen Download mit einem Developer Kit an. In dieser Dokumentation ist beschrieben wie interessierte Entwickler mittels eines TCP Servers der auf dem Spektrometer betrieben wird, selbst Applikationen entwickeln können. Die Dokumentation enthält ausführliche Informationen zu Verbindungsparameter, Rückgabetypen und Dateiformaten.

Weiter konnten wir auf das GitHub Repository der SPECCHIO Datenbank zurückgreifen. In dieser Applikation wurde verschiedenste Berechnungen und Manipulationen mit Spektralfiles bereits in Java ausprogrammiert.

1.3 Erreichtes

Die Applikation wurde mit den definierten Grundanforderungen vollständig umgesetzt. Der Benutzer kann, sofern das iPad mit dem Spektrometer über WLAN verbunden ist, das Spektrometer bedienen und Messungen ausführen. Es wurde speziell darauf geachtet die Messungen einfacher und für den Benutzer intuitiver zu gestalten.

1.4 Leserführung

2 Theoretischer Teil

2.1 Einleitung

2.2 Messverfahren

2.3 Berechnungsablauf

2.4 Berechnungen

2.4.1 Dark Current

2.4.2 White Reference

2.4.3 Radiance

3 Umsetzung

Dieses Kapitel beschreibt die Umsetzung des Produktes. Die Nachfolgenden Abschnitte werden Schrittweise detaillierter und beschreiben den Aufbau sowie die Designentscheidungen des Softwarecodes.

3.1 Einleitung

3.2 Anforderungen

3.3 Technologien

In diesem Abschnitt wird beschrieben, welche Technologien und Tools eingesetzt wurden, um die iOS Applikation umzusetzen. Es wurde darauf geachtet, dass bewährte und von den Herstellern vorgesehene Technologien verwendet wurden.

3.3.1 Entwicklungsumgebung

Das Projekt wurde mit XCode 8.2.1 umgesetzt. Grundsätzlich, kann jede IDE verwendet werden, welche mit Swift kompatibel ist. Um das Projekt builden zu können wird zwingend ein Apple Rechner benötigt.

3.3.2 Programmiersprache

Das Projekt wurde in der Programmiersprache Swift 3 umgesetzt und ist mit iOS 10 und höher kompatibel. Es wurde darauf geachtet, dass alle Abhängigkeiten ebenfalls in Swift umgesetzt sind. Dies erleichtert die Weiterentwicklung da kein, für ungeübte Entwickler, meist schwer lesbarer Objective C Code zum Einsatz kommt.

3.3.3 Abhängigkeiten

Externe Ressourcen, wurden vorwiegend mit CocoaPods eingebunden. CocoaPods ist ein Packet Manager, mit welchem man Packete in ein bestehendes Projekt einbinden kann. Dies geschieht über ein sogenanntes Podfile, welches zum Projektumfang gehört. Dieses File beschreibt die externen Abhängigkeiten, sowie welche Version benötigt wird.

Folgende Pods wurden für das Projekt verwendet:

- pod 'Charts/Realm'
- pod 'Zip'
- pod 'PlainPing'
- pod 'Pulsator'

- pod 'FontAwesome.swift'

Einzig die TCP Kommunikation wurde mit externem Code gelöst, welcher nicht als Pod verfügbar ist.

3.4 Architektur

3.4.1 Grobarchitektur

Um weite Teile des Sourcecodes erneut nutzen zu können, wurde das Projekt in drei Teile aufgeteilt. Diese 3 Teile werden in den folgenden Abschnitten beschrieben.

3.4.2 Core

Der Core Ordner fasst den gesamten Code zusammen, welcher Platformunabhängig ist. Dieser Code kann auch in anderen Projekten mit ähnlicher Anwendung benutzt werden.

3.4.3 Service

Der Service Layer, kapselt einige Anfragen des App Layers an den Core Teil. Dieser Layer wird oft dazwischen gelegt, um beispielsweise Parallelität zu vermeiden. So wird im Layer beispielsweise ein Synchronized Queue benötigt, um nur ein Request nach dem Anderen an ein Spektrometer zu senden.

3.4.4 iOS App

Dieser Layer beinhaltet die eigentliche iOS App mit all Ihren typischen Eigenschaften, wie das App-Delegate und die Storyboards. In diesem Ordner werden alle Views sowie deren Controller gespeichert.

3.5 Core

3.5.1 Connection

In diesem Ordner, befindet sich die TCPManager Klasse sowie die externen Socks Dateien. Das Socks Framework kümmert sich um die Kommunikation auf TCP Ebene. Eine ausführliche Dokumentation ist unter folgendem Link zu finden:
https://github.com/vapor/socks
Der TCP Manager ist für die Anfragen des Gerätes verantwortlich. Es ist die einzige Klasse, welche direkt mit dem Spektrometer kommuniziert. Er ist deshalb auch als Singleton umgesetzt. Mit der connect Funktion, kann eine Verbindung zum ASD Gerät hergestellt werden, danach können beliebig viele Kommandos mit sendCommand gesendet werden. Um die Verbindung zu schliessen, muss lediglich die disconnect Methode aufgerufen werden.

3.5.2 Input/Output

In diesem Kapitel, werden alle Klassen beschrieben, welche direkt Daten einlesen oder Ausgeben.

File Writer

Die Output Klassen, dienen dazu ein Indico File zu schreiben. Die Base Klasse kapselt dafür alle Schreibvorgänge. Von der Klasse File Writer, wird auf diese Methoden zugegriffen und stösst so das Schreiben in der Richtigen Reihenfolge an. Von aussen kann lediglich gewählt werden, welche der 3 Messmethoden geschrieben werden soll: Raw, Reflectance oder Radiance.

File Reader

Spectrometer Parser

3.5.3 Calculations

Der Ordner Calculations enthält lediglich die Klasse SpectrumCalculator welche nur statische Funktionen enthält. Diese dienen der Berechnung der Reflektanz sowie der Radianz. Ausserdem beinhaltet die Klasse noch eine Methode um eine Dark Correction auf einem übergebenen Spetrum durchzuführen.

3.5.4 Error Handling

3.6 Service

3.6.1 Instrument Store

Im Instrument Store, werden alle Objekte gespeichert, auf welche global zugegriffen werden muss und nichts mit dem Userinterface zu tun haben. Um zu garantieren, dass die Variablen nur einmal existieren, ist die Klasse als Singleton umgesetzt.

3.6.2 Command Manager

Der Command Manager übernimmt die Funktion alle Kommandos in der korrekten Reihenfolge abzuarbeiten. Der Command Manager bietet Methoden für alle gängigen Spektrometer Kommandos an. Wird eine dieser Methoden aufgerufen, übergibt der Command Manager die Anfrage an eine synchrone Queue welche nach dem First-In-First-Out Prinzip die Anfragen abarbeitet. So ist sichergestellt, dass ein Kommando erst an das Spektrometer gesendet wird, wenn alle vorherigen abgearbeitet sind.

3.6.3 File Write Manager

Der Write Manager funktioniert ähnlich wie der Command Manager. Er übergibt write Anfragen ebenfalls einer synchronen Warteschlange und arbeitet diese ab. Dazu kümmert sich der FileWriteManager auch um die korrekte Benennung der ASD Files. Dies geschieht, indem er den höchsten geschriebenen prefix eines Files im Pfad findet und diesen dan entsprechend erhöht.

3.7 iOS App

3.7.1 App Delegate

Das App Delegate ist das eigentliche Einstiegsfile jeder iOS Applikation. Darin wird beschrieben, wie der Prozess aus verschiednen Zuständen abläuft (Stopped, Paused, Notified). Im AppDelegate werden auch Applikationsweite Designsänderungen vorgenommen, so wird beispielsweise die Schriftart für die gesamte Applikation gesetzt.
AppDelegate wird ebenfalls aufgerufen, wenn ein File erfolgreich importiert wurde. Das importieren selbst, wird vom Betriebsystem übernommen. In diesem Projekt wird auf der aktuellen View eine Meldung angezeigt, dass ein File erfolgreich importiert wurde.

3.7.2 Core Data

Core Data ist ein System, um Daten auf dem Gerät zu speichern, es dient nicht als Datenbank ersatz, kann aber für einfache Speicherungen genutzt werden. Durch die Integration in XCode, lassen sich Speicherklassen einfach modellieren und einsetzen. Core Data Einstellungen können in xcdatamodeld Klassen vorgenommen werden. Dabei können Datentypen sowie die Relationen der einzelnen Records definiert werden.

3.7.3 Views

Alle ViewController sind in der Datei Main.storyboard enthalten. Dies war eine bewusste Entscheidung, um Entwicklern einen guten Überblick über den gesamten UI Ablauf zu ermöglichen. Einzig das Design für eine Zelle der Messübersichtstabelle wurde in eine eigne XIB-Datei ausgelagert.

Die Anordnung der Controls wurde im Storyboard gelöst. Kleinere Merkmale wurden jeweils im Code angepasst, indem von bestehenden Controls abgeleitet wurde.

3.7.4 Controllers

Settings

Einstellungen, welche pro Applikation verfügbar sein müssen, werden in den sogenannten UserDefaults gespeichert. Dieser Speicher, kann sämtliche serialisierbaren Objekte speichern.

3.7.5 Components

3.7.6 View Store

Der ViewStore dient wie der InstrumentStore zum speichern weiter benötigter Variablen. Er ist ebenfalls als Singleton implementiert. Im Gegensatz zum InstrumentStore enthält der ViewStore aber nur Elemente, welche UI relevant sind. Darin wird beispielsweise gespeichert, ob gerade eine Aquire-Schleife läuft.

3.7.7 Service

Validation

Für die Validierung wurde von jedem benutzen Control eine Ableitung erstellt und und das BaseValidationControl Protokoll implementiert. Dieses Protokoll enthält ein Property isValid welches den Gültikeitszustand des Objektes enthält.

In jedem ViewController, muss nun nur noch der ValidationManager aufgerufen werden und die Hauptview übergeben werden. Dieser ValidationManager prüft nun alle Subviews, welchedas Protokoll implementieren.

Um die Validation für einen neuen ViewController hinzuzufügen, kann folgendermassen vorgangen werden: 1. Erstellen Sie einen neuen ViewController 2. Fügen Sie ein Control hinzu und leiten sie von einem bestehenden ValidationControl ab. Oder erstellen Sie eine neue Ableitung eines Controls und implementieren Sie das BaseValidationControl Protokoll. 3. Rufen Sie die validateSubViews Methode des ValidationMangers auf.

File Browser

Der FileBrowser, ist eine Abwandlung von einer externen Swift Komponente, welche unter folgendem Link zu finden ist:
https://github.com/marmelroy/FileBrowser
Diese Komponente wurde auf die eignen Bedürfnisse angepasst. Der FileBrowser besteht aus einer Basisklasse und jeweils 2 Klassen für die Datei sowie die Ordnerauswahl.