Simple Habits

Alarm Clock
Software Architecture Document

Version 1.4

Revision History

Date	Version	Description	Author
28/11/2016	1.0	Initial Creation	Benedikt Bosshammer
30/11/2016	1.2	Class Diagram added	René Penkert
20/12/2016	1.1	MVC added	Benedikt Bosshammer
05/05/2017	1.3	Deployment View Added	René Penkert
18/05/2017	1.4	Pattern Information	Benedikt Bosshammer

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

The scope of this SAD is to show the architecture of the Alarm Clock application. Affected are the class structure, the use cases and the data representation.

1.3 Definitions, Acronyms, and Abbreviations

SAD	Software Architecture Document
tbd	to be determined

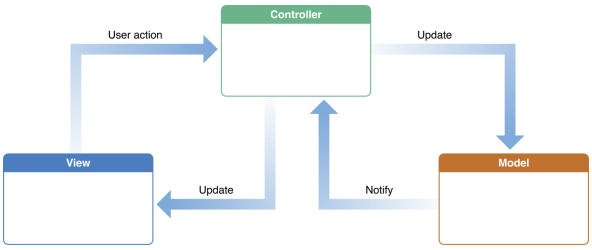
1.4 References

1.5 Overview

This document describes the software architecture to ensure that everybody understands it and new developers can be let into the project easily.
It describes how the application is structured, why it is structured like it is and what we do to ensure that our code meets a certain quality standard.

2. Architectural Representation

This project will use the MVC architecture, which Swift utilizes by default.



https://developer.apple.com/library/mac/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

3. Architectural Goals and Constraints

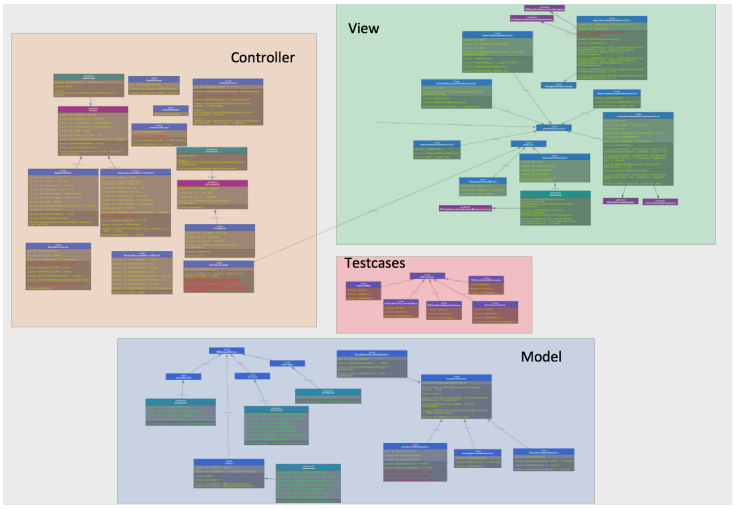
The main goal of the MVC architecture is to separate the view from the logic. Therefore the view knows nothing on its own, but gets all information from the logical part. Swift automatically creates projects based on the MVC pattern.

4. Use-Case View

n/a

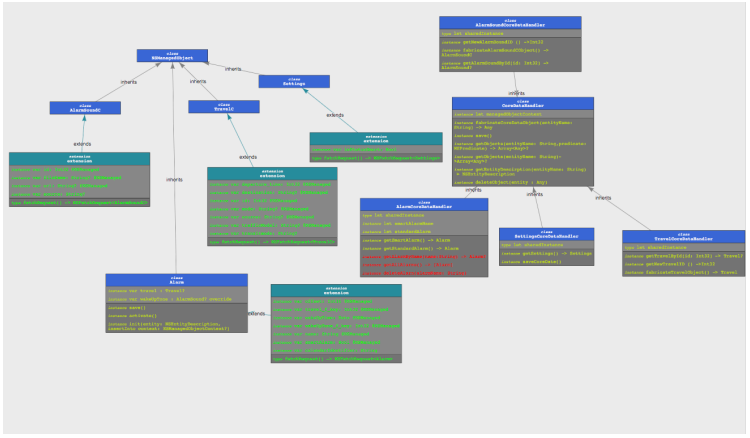
5. Logical View

See our class diagram:



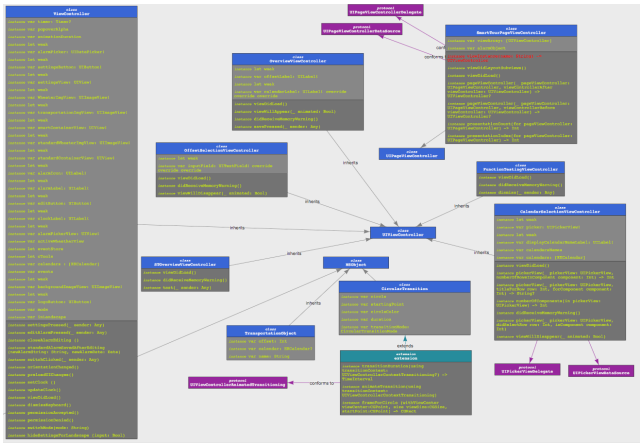
https://github.com/renpen/SimpleHabits/blob/master/SoftwareEngeneering/ClassDiagramOVWithArchitecture.png
For a detailed look at one of the sections look:

- Model:



https://github.com/renpen/SimpleHabits/blob/master/SoftwareEngeneering/ClassDiagrammModel.png

- View:

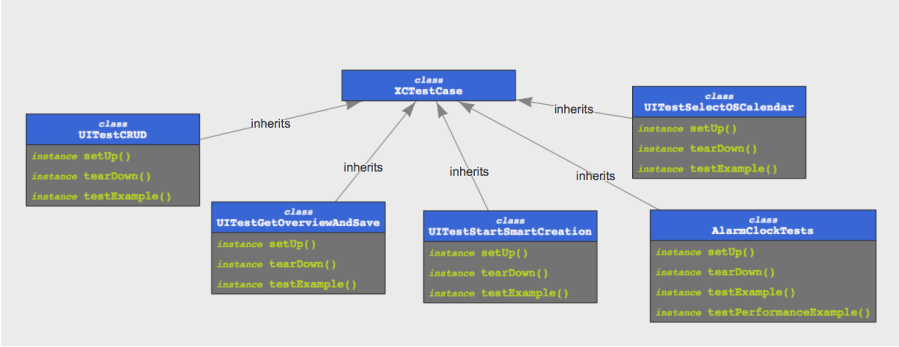


https://github.com/renpen/SimpleHabits/blob/master/SoftwareEngeneering/ClassDiagrammView.png

- Controller:

https://github.com/renpen/SimpleHabits/blob/master/SoftwareEngeneering/ClassDiagrammController.png

- Testcases:



https://github.com/renpen/SimpleHabits/blob/master/SoftwareEngeneering/ClassDiagrammTest.png

5.1 Overview

5.2 Architecturally Significant Design Packages

6. Process View

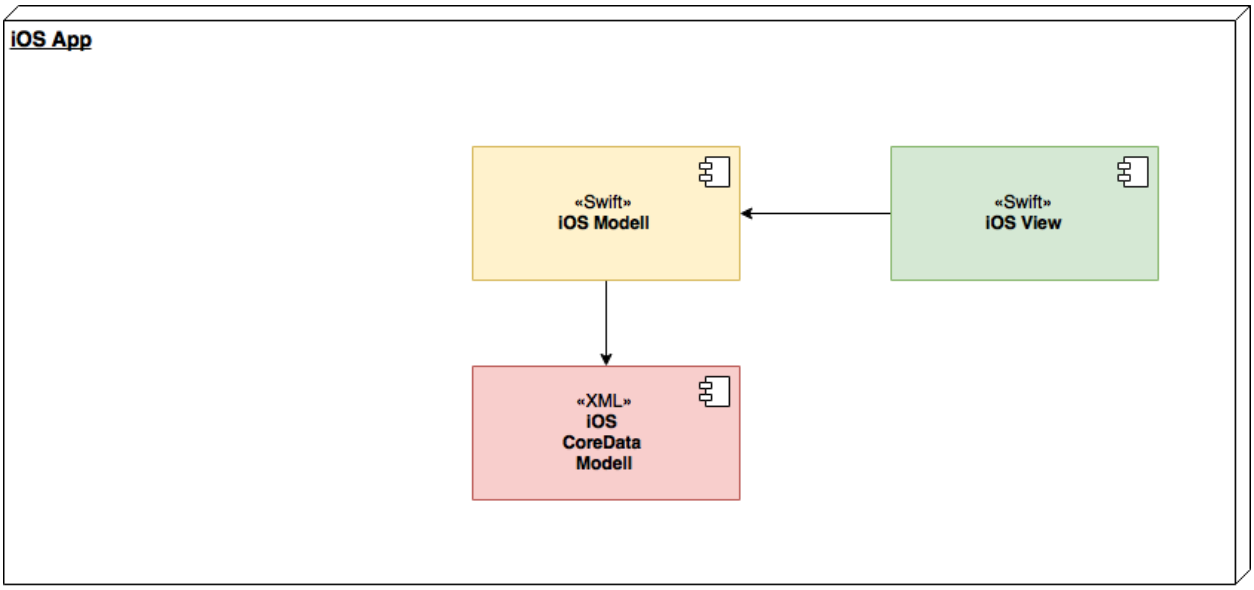
(n/a)

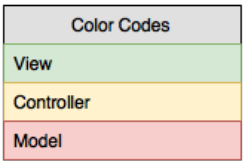
7. Deployment View

The deployment view for this application is rather simple. To run the application the user needs an iPhone or iPad with a current iOS version. He then will be able to download the application via the iOS AppStore.

For development it is also possible to install the application directly from a Mac with XCode or AppCode.

There are no hardware requirements, as all iOS devices with a current version are capable of running the application.
According to the mentioned Scenario there is only one Component in our Deployment View:





8. Implementation View

(n/a)

9. Data View

See our ER-Diagram:
https://github.com/renpen/SimpleHabits/blob/master/SoftwareEngeneering/Published/ErCoreData.pdf

The shown database diagram shows the current state. There will be more entities throughout the development of the application.

Swift uses Core Data for data persistence, which is based on sqLite (good for mobile apps).

10. Size and Performance

(n/a)

11. Quality

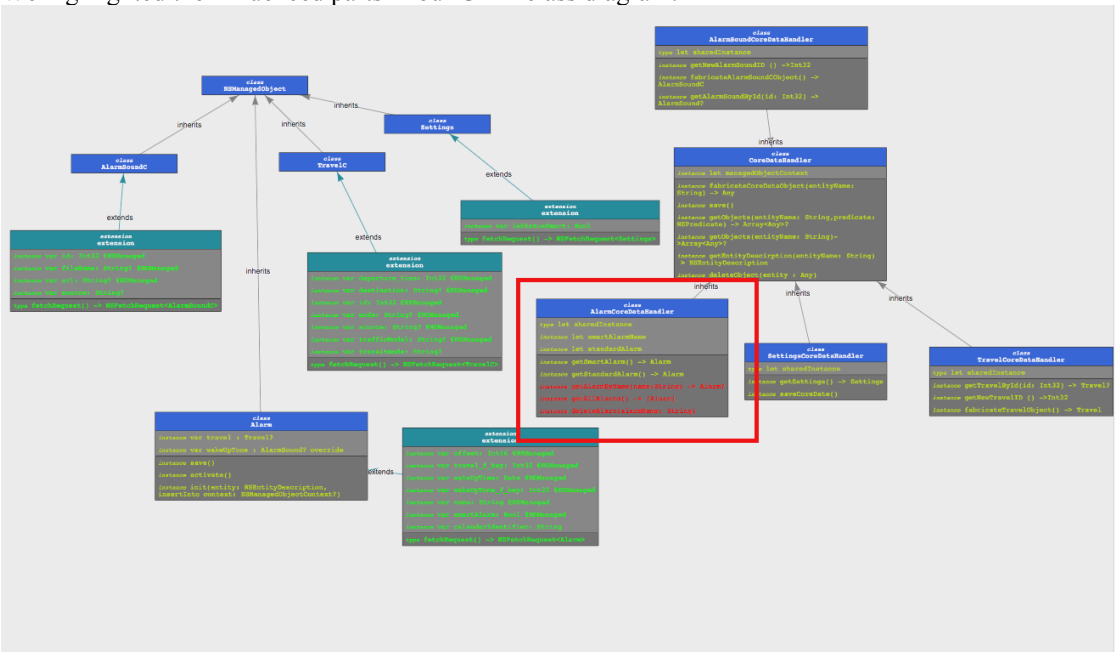
For nowadays standards the quality of our application is important to find stoutness in a great audience. Current standards are based on a flat ui design as well as clear and structured interfaces. The app should be used intuitively.

We try to build an extensible structure, so there is always the possibility for us to create new game modes or add other features. Due to Swift, the application will be available for all modern iOS devices. The migration to OSX in the end should be simple.

12. Patterns

We use design pattern to design and refactor our frontend code. Design patterns are possible solutions to general problems software developers face during software development.
For our central logic - the implementation of the alarm - we decided to use the fabric pattern to wrap the constructor and generate a Core Data Object when using the singleton managing the Core Data logic. See the implementation here.

The fabric pattern is used to generate an object by calling a method instead of the constructor.
We highlighted the influenced parts in our UML class diagram:



(source)