OULU UNIVERSITY OF APPLIED SCIENCES

Joonas Känsälä

FURTHER DEVELOPMENT OF SETTINGS FEATURE ON THE SPENT IOS APPLICATION

FURTHER DEVELOPMENT OF SETTINGS FEATURE ON THE SPENT IOS APPLICATION

Joonas Känsälä
Bachelor's thesis
Spring 2017
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software Development

Author: Joonas Känsälä
Title of the bachelor's thesis: Further Development of Settings Feature on the SPENT iOS Application
Supervisor: Veikko Tapaninen
Term and year of completion: Spring 2017 Number of pages: 41

This thesis was commissioned by the firm Receiptless Software Oy and the aim of this thesis was to create a new settings feature for an application called the SPENT on iOS mobile platform. The purpose of this new settings feature was to improve on the old feature that the previous version had and to add new functionalities, for example, security settings and a new user interface.

The new feature was created by using an Xcode development environment, which then had necessary components for developing the new settings feature. The components the project used were two major frameworks within the Cocoa Touch Framework: UIKit and Foundation. After the new settings feature had been created, it was tested thoroughly by testers within the firm and by an outside testing firm.

The project was a success and it was released into the App Store as a version update. Although the decisions for creating this new application were appropriate, storing user details should be done in a way that the user details will be encrypted.

Keywords: Swift Programming Language, iOS, Security

CONTENTS

VOCABULARY

ARC	Automatic Reference Counting
Framework	A structure which supports or guides the developer to build features that expands the structure into something useful.
Heap	In memory management, heap is the memory set aside for dynamic allocation.
IDE	Intelligent Development Environment
iOS	An operating system for iPhone, iPad and iPod Touch
MVC	An abbreviation for Model View Controller
Realm	A mobile database which is an alternative for SQLite and Core Data solutions
Refactoring	Process of restructuring the code without changing its external behaviour
REPL	Read-Evaluate-Print loop
SDK	Software Development Kit
SnapKit	An auto layout definitive software library for iOS and OSX
Trello	A web-based project management application
UI	An abbreviation for User Interface
UX	An abbreviation for User Experience

1 INTRODUCTION

This project was done for the company called Receiptless Software Oy. The project handled a feature which is a new settings view for an iOS application called SPENT. The phases of the project were designing the feature, implementing those designs into the development describing the testing practices and finding possible bugs during the testing phase of the development.

Receiptless Software Oy company calls themselves SPENT, which is also the name of their app. The goal of SPENT is to create an application which makes expense management easier. The CEO, who founded the startup company was unimpressed of the applications that were available on Google Play and App Store. At first the outsourcing of the product did not succeed favourably and thus SPENT, the startup company, was founded in August 2015. Now the company has increased their staff from three people into 30 people. (1.) SPENT has offices located in New York, Silicon Valley, Moscow and in Oulu, Finland (2.)

2 IOS AND FREQUENTLY USED TOOLS

2.1 History of iOS

On January 9th 2007 at the Macworld conference and expo, Steve Jobs displayed the first version of the iPhone to the world. Even though it was told that iPhone would run all the OS X applications which run on Mac OS, this was not the case at all. Later on October 17th 2007 Apple announced that the SDK was under development. On March 6th 2008 Apple released the first beta of the SDK and gave the iPhone operation system a name iPhone OS. In June 2010 Apple rebranded iPhone OS into iOS, which required them to acquire the IOS trademark from Cisco just to avoid possible lawsuit cases. (3.)

At the moment iOS has been developed up to the version 10.3.1, which was released on April 3rd 2017 with various bug fixes, security improvements and a fix for a Wi-Fi vulnerability. (3.)

2.2 Technical structure of iOS

The iOS architecture is divided into four layers. This layered structure allows the developer to create applications with ease because iOS communicates with the underlying hardware by using a set of well-defined system interfaces. These interfaces make app development very scalable between different devices with different hardware. (4.) Figure 1 below shows the architecture of iOS.

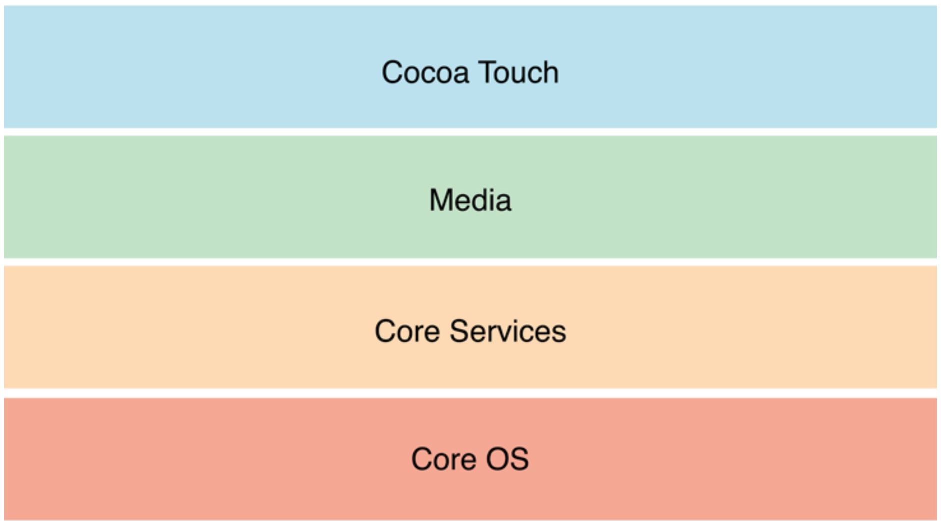


FIGURE 1. Layers of iOS (4.)

It is preferable that the developer uses higher-level frameworks over lower level frameworks. Higher-level frameworks provide object-oriented abstractions for lower-lever constructs. This reduces the amount of lines of code that the developer must write. However, if there are cases where there is not a higher-level framework application of a lower-level function, then developer may use lower-level frameworks if necessary. (4.)

The Cocoa Touch Layer is a layer which contains the important frameworks for creating iOS applications. These frameworks define the appearance of the app. These frameworks support multitasking, touch-based input, push notifications and other high-level system services. (5.)

The Media Layer contains the graphics, audio and video technologies that the developer uses to implement multimedia experiences into an application. This layer is developed in a way that makes using multimedia technologies easy to implement while building apps. (6.)

The Core Services layer is reserved for important systems services for apps. The most important frameworks in this layer are the Core Foundation and Foundation frameworks, which define the basic types that every application utilizes.

This layer also supports features, such as location, iCloud, social media and networking. (7.)

The Core OS layer contains the low-level features that most other technologies are built upon. In situations where developers must handle security or communication with an external hardware accessory, it is executed by using the frameworks in this layer. (8.)

2.3 Cocoa Touch Framework

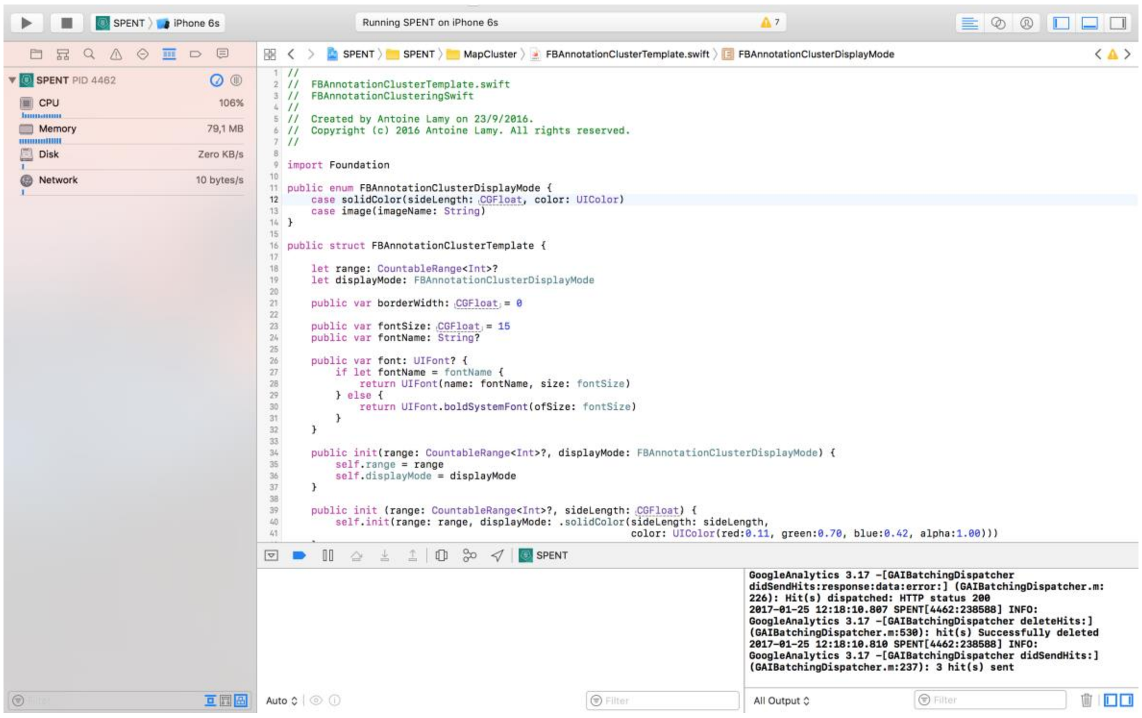
Cocoa Touch is an application framework for iOS and it includes an Objective-C runtime and two core frameworks: Foundation and UIKit. The term “Cocoa” has been used to refer generically to any class or object that is based on the Objective-C runtime and inherits from the root class, NSObject. The term “Cocoa Touch” is also used when referring to application development using any programmatic interface of the respective platforms. (9.)

The purpose of the Foundation framework is to provide a small set of basic utility classes, to make software development easier by introducing consistent conventions for such things as deallocation, to support Unicode string, object persistence and object distribution and to provide a level of OS independence, to enhance portability. (10.)

UIKit is a framework which contains a crucial infrastructure for constructing and managing iOS and tvOS applications. This framework provides the architectures needed to manage the UI, the handling event infrastructure needed for the user input responsivity and the app model needed to drive the main run loop and interact with the system. Other additional UIKit features, for example, are the foreground and background execution of the application, the PDF creation and animation. (11.)

2.4 Xcode

Xcode is an IDE (Integrated Development Environment) created for macOS. It enbles to create applications for iOS, macOS, watchOS and for tvOS. The first version of Xcode was released in 2003. The latest version of Xcode is Xcode 8.X, which is also the one that was used for this thesis. The reason why the version 8.X was used is that it was the only IDE version supporting the Swift 3.0 programming language and the iOS SDK 10.2 version. (12.) Figure 2 shows how the Xcode development environment looks like.


SPENT 〉 iPhone 6s
Running SPENT on iPhone 6s
SPENT 〉 SPENT 〉 MapCluster 〉 FBAnnotationClusterTemplate.swift 〉 FBAnnotationClusterDisplayMode
SPENT PID 4462
CPU 106%
Memory 79,1 MB
Disk Zero KB/s
Network 10 bytes/s
SPENT
Auto
Filter
All Output
Filter

```
//
//  FBAnnotationClusterTemplate.swift
//  FBAnnotationClusteringSwift
//
//  Created by Antoine Lamy on 23/9/2016.
//  Copyright (c) 2016 Antoine Lamy. All rights reserved.
//

import Foundation

public enum FBAnnotationClusterDisplayMode {
    case solidColor(sideLength: CGFloat, color: UIColor)
    case image(imageName: String)
}

public struct FBAnnotationClusterTemplate {

    let range: CountableRange<Int>?
    let displayMode: FBAnnotationClusterDisplayMode

    public var borderWidth: CGFloat = 0

    public var fontSize: CGFloat = 15
    public var fontName: String?

    public var font: UIFont? {
        if let fontName = fontName {
            return UIFont(name: fontName, size: fontSize)
        } else {
            return UIFont.boldSystemFont(ofSize: fontSize)
        }
    }

    public init(range: CountableRange<Int>?, displayMode: FBAnnotationClusterDisplayMode) {
        self.range = range
        self.displayMode = displayMode
    }

    public init (range: CountableRange<Int>?, sideLength: CGFloat) {
        self.init(range: range, displayMode: .solidColor(sideLength: sideLength,
                                                          color: UIColor(red:0.11, green:0.70, blue:0.42, alpha:1.00)))
```

```
GoogleAnalytics 3.17 -[GAIBatchingDispatcher
didSendHits:response:data:error:] (GAIBatchingDispatcher.m:
226): Hit(s) dispatched: HTTP status 200
2017-01-25 12:18:10.807 SPENT[4462:238588] INFO:
GoogleAnalytics 3.17 -[GAIBatchingDispatcher deleteHits:]
(GAIBatchingDispatcher.m:530): hit(s) Successfully deleted
2017-01-25 12:18:10.810 SPENT[4462:238588] INFO:
GoogleAnalytics 3.17 -[GAIBatchingDispatcher didSendHits:]
(GAIBatchingDispatcher.m:237): 3 hit(s) sent
```

FIGURE 2. A screenshot of Xcode IDE

2.5 Version handling

In a software development project, using a VCS (version control system) is beneficial. The source code of the project alters all the time and there are times when the code might break. Without using a VCS, it is very difficult to fix this issue. The VCS allows developers to view the version history, to do the necessary changes into the code, to make sure everything works and then to upload the changes into the repository of the VCS. The VCS has features called branching and merging. These allow developers to work on different features independently in their own branches. When the development is done, then developers can merge these branches into a branch which is deemed as the main application. (13.)

Bitbucket was used as a version handling platform in this project. Bitbucket is a web-based hosting service which allows the developers revision control over their projects by using Git as a version control system. Bitbucket has tailored itself towards helping professional developers with private proprietary code, especially since being acquired by Atlassian in 2010. (14.)

2.6 MVC

SPENT's application architecture follows the MVC based architectural pattern. The acronym MVC stands for Model, View and Controller. FIGURE 3 shows what type of interactions these layers have.

The model layer in this pattern means all the classes that handle the data, anything from persistence, model objects, parsers and networking management classes. (15.)

The view layer includes everything that is on the screen. View layer classes are easily reusable since they themselves do not have much logic behind them. (15.)

The controller layer includes classes that link the view layer components and the model layer components to each other. In the iOS application development, and in ideal case, the controller does not know the view that it is dealing with. Instead, it will communicate with an abstraction via a protocol. A classic example is the way a UITableView communicates with its data source via the UITableViewDataSource protocol. (15.)

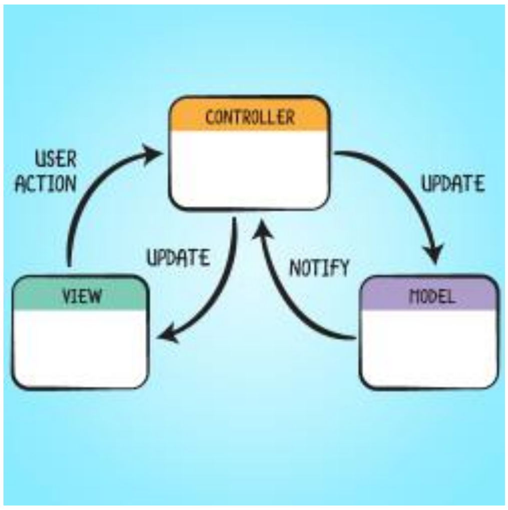


*FIGURE 3: MVC architecture (*15.*)*

The reason why the application architecture follows the MVC framework is that the idea behind the MVC architecture makes it easy to differentiate data management, views and controllers into different classes. If one logic is modified, such as a model-based class, then it will not break the logic in another class, which is either a view or a controller. This also means that the code is easy to maintain and a parallel development of the project is possible too. (16.)

3 SWIFT 3.0

Swift is a powerful and intuitive programming language for macOS, iOS, watchOS and tvOS. Swift code is safe by design, yet it also produces a software that runs lightning-fast. Swift eliminates entire classes of unsafe code. Variables are always initialized before use, arrays and integers are checked for an overflow, and memory is managed automatically. Another safety measure is that Swift objects by default cannot be nil. In fact, the Swift compiler will stop the user from trying to make or use a nil object with a compile-time error. However, if there is a case where the nil object is being used, Swift uses a feature called optionals. This means that when an object returns a nil value, then instead of using nil as a value, a default value can be used instead. This makes the compiling of the code even more safe. (17.)

3.1.1 History

The development of Swift was started in July 2010 by Chris Lattner, with the eventual collaboration of many other programmers working for Apple. Swift took language ideas, e.g. from Objective-C, Rust, Haskell, Ruby, Python and C# programming languages. On June 2nd 2014, at the Apple Worldwide Developers Conference (WWDC), a beta version of the programming language was released to registered Apple developers. During WWDC, The Swift Programming Language manual was released on the iBooks Store and on the official website for free. (18.)

During WWDC 2016, Apple announced an iPad exclusive app, named Swift Playgrounds, intended to teach people how to code in Swift. The application presents the feedback of the code structure in a 3D video game-like manner and tells the developer in what order certain lines of code should be placed. (18.)

Today, the latest stable release of Swift is the version 3.1.1, which was released on April 21st 2017 and is still open-source software under the Apache Licence 2.0. (18.)

3.1.2 Features

Swift is an alternative to the Objective-C language. The syntax of Swift language is presented in a simpler manner. The majority of people may think that Swift is easier to read compared to Objective-C. (19.) This chapter presents the features that are relevant for the project. These features are access control, optionals and chaining, memory management and debugging.

Swift provides five different access control levels for handling classes, structures and enumerations, and to properties, methods, initializers and subscripts belonging to those types. These access control levels are open, public, private, fileprivate and internal. Open, which is only for classes and their methods, indicates that the class can be subclassed outside of the module. Public means that it is accessible from a different module. Private means that a symbol is only accessible in the immediate scope. Fileprivate means that it is only accessible within the file and internal indicates that it is only accessible within a containing module. (20.)

One of the more important features in the Swift language are optionals. They are represented in the Swift by ? operator. To access the value inside, only if the value is not nil, it must be unwrapped to expose the instance inside. For unwrapping, the developer has to use the ! operator. However, if the value turns out to be nil, then the null-pointer error occurs. For these cases there is the ? operator. The ? operator checks whether the instance within the value is nil or not. If the instance value is nil, then the ? operator will not unwrap the value. Swift 2 introduced a new keyword called guard. Guard is good for situations where a value must meet a certain condition in order to be executed. (18.)

Swift uses the ARC system to manage the memory. In Swift references to objects are strong, unless they are declared weak or unowned. In Swift, it is required that values are handled in a way that they receive a nil value. These values must be handled by unwrapping them with a conditional statement, thus allowing a safe usage of the value. A strong reference to an object cannot be of type optional as the object will be kept in the heap until the reference is deallocated. A weak reference is of type optional as the object can be deallocated and the reference can be set to nil. Unowned references are neither strong or optional type. The object, which has unowned reference points, is not deallocated as long as the reference itself remains allocated. (21.)

For debugging Swift uses REPL and gives it interactive properties which are similar compared to Python. This makes the Swift system to debug cleanly and run within the development environment. REPL is further enhanced to an alternative development environment called playgrounds. Playgrounds is an interactive view which debugs the code on the fly. (18.)

4 OLD SETTINGS VIEW

SPENT is an expense management application mainly meant for people who travel a lot and have a lot of expenses to manage. The way it works is that the user first links their bank account and card to receive the expenses for a preview, then the user can swipe the expenses into different folders. These folders can either be for the user's own expenses or for business expenses. Folders can be created e.g. for budgeting traveling expenses and such. If services or products are bought from retailers that are in cooperation with SPENT, the user gets cash back from the purchases the user has made. The user can also scan receipts or create receipts if necessary. (22.)

SPENT's version 1.5.4 settings had a simplistic design and its functionality was limited. The main settings view appeared from the left side of the screen and did not cover the screen completely. This meant that there were two view controllers active on the screen. Figure 4 shows how the old main settings view was executed before.

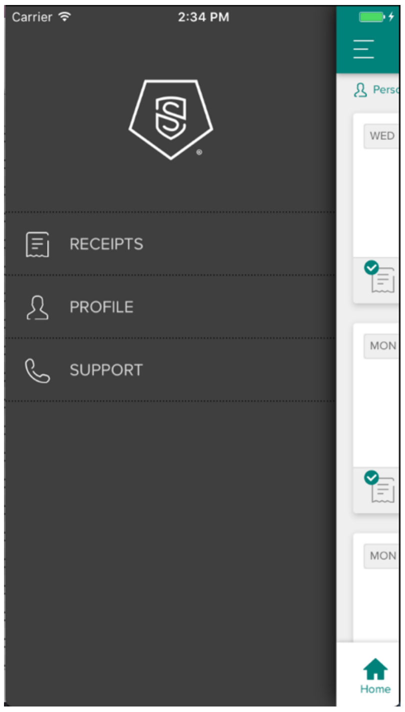


FIGURE 4. Old main settings view

The view hierarchy was portrayed in Xcode IDE. One can assume that there are too many UI components active in the application at the same time when the user has opened settings view. In figure 5, there are three different active view controllers: UITimelineViewController, UISettingsViewController and UIGalleryViewController.

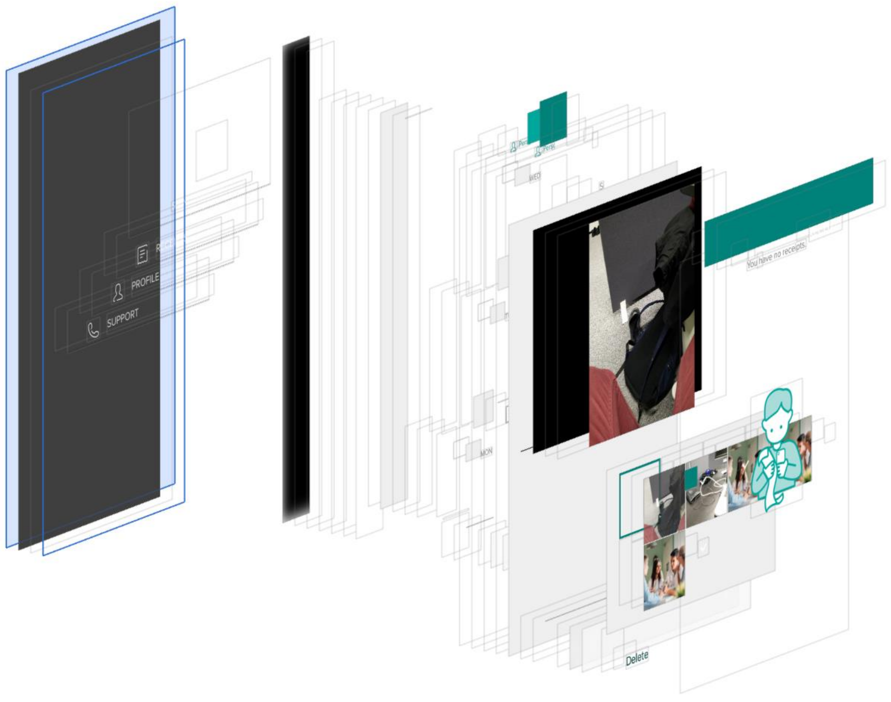


FIGURE 5. View hierarchy.

4.1 Functionality

The functionality of the old settings view was simple. There were three different list buttons which would take the user to the profile view, receipt view and support view. In the main view (UITimelineViewController), there was a button on the top left part of the screen. It opened the settings view which covered about 75% of the screen. This was executed by using a third-party library called iOS-Slide-Menu.

4.2 UI

The main view of old settings view was executed by using iOS-Slide-Menu, which is a custom-made navigation controller created by a third-party developer. Within the navigation controller, mainly normal UIKit components were used, for example UIKit view classes: UITableView, UIImageView and UILabel. The support view and the profile view used these same classes in their views, but the UI was modified to fit into the SPENT design template. In the receipt view, however, the collection view class was also used (Figure 6). The collection view shows all the receipt images that the user has taken. When the user chooses an image, the image will be portrayed in the image view which is above the collection view. The user may also delete images by pressing the delete button below the collection view.

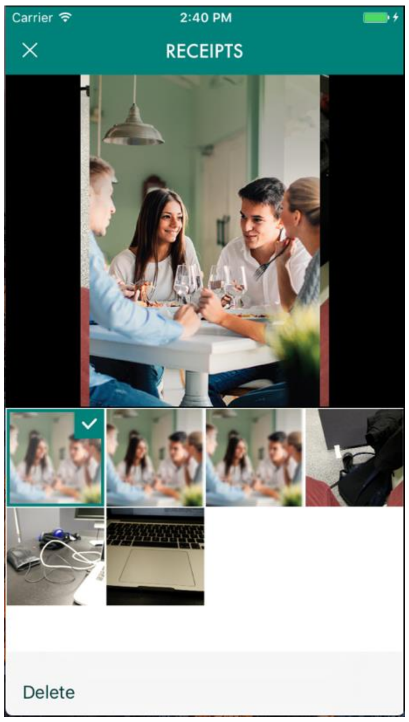


FIGURE 6. Receipt view

4.3 Limitations

The settings view does the bare minimum that it is required to do. The user can change the profile information in the profile view, change the passcode and password. In the support view, the user can find frequently asked questions, contact customer service, check terms and conditions and delete the account.

Here are the problems that the old settings feature had:

- The user could not deactivate the passcode prompt.
- There was no option to change the security settings.
    - The user could not turn the automatic screen-lock on or off.
    - The user could not change the duration of the automatic screen-lock.
    - The user could not enable or disable the passcode prompt when the screen went to sleep mode.
    - The user should be able to enable or disable the Touch ID if the device has that feature.
        - By using the Touch ID, the user should be able to bypass the passcode screen.
- The view that lists and presents the open-source licenses was missing completely.

5 NEW SETTINGS VIEW

The new settings view was developed for the release of SPENT version 1.6.0. The new design and functionality were developed according to the graphic designs and requirements were figured out by the people responsible for the product development.

5.1 Concept and design

For the concept creation and design, Confluence created by Atlassian was used. Confluence is a tool for the team collaboration. A user can create a new page in Confluence for a specific feature and document requirements, instructions and specifications, add wireframes or add a fully drawn version of the user interface instead.

5.1.1 Goals

The requirements for the new settings view were determined as follows:

1. The main settings view.
    a. The user can see the first and the last name.
    b. The user can manage user details in the profile view.
    c. The user can access the receipt view to manage all the images of the receipts.
    d. The user can see which expenses, reports or folders have been shared to other users.
    e. The user has access to the support view.
    f. The user has access to the security settings view.
    g. The user can leave a rating of the app in the App Store.
    h. The user can sign out from the account.
    i. The user can see the version number of the application.

2. Changes in the profile view.
    a. The profile deletion button changes the location from the support view to the profile view.
3. Shared links view.
    a. Lists all the expenses, reports and folders that are being shared with other users.
4. Security settings view.
    a. The user must be able to change the passcode and turn it on or off.
    b. The user must be able to disable the automatic screen-lock and, if the user enables the automatic screen-lock, the user must be able to change the duration before the screen will lock itself.
    c. When The user enables the sleep-lock, the application must be able to lock itself when screen turns off. If The user disables the sleep-lock function, the screen will not lock itself.
    d. If The user enables the Touch ID function, the user can bypass the auto-lock by using the Touch ID functionality. However, if the user does not have an iPhone with a Touch ID button, the option for Touch ID does not show at all.
    e. Status labels should show what happens when the user has enabled or disabled some functions in the settings. For example, if the user decides to set the passcode on, to set the automatic screen-lock to 30 minutes and to enable the sleep-lock, the status label should say: “A passcode will be required after 30 minutes of inactivity or when the device screen awakes form sleep mode” and “You will always be required to input your passcode after the app restarts.”
5. Changes for the support view
    a. Remove the delete button from the view and add a button which directs users to the list of open-source licenses.

5.1.2 New design

The UX designer of the company designed the user flow for the settings view alongside with the head designer. The head designer then designed the look of the settings view to start out with. The first view, which is represented to the user, is the main settings view. This is the view that has gone through the most changes in the new version of the new settings views (Figure 7).

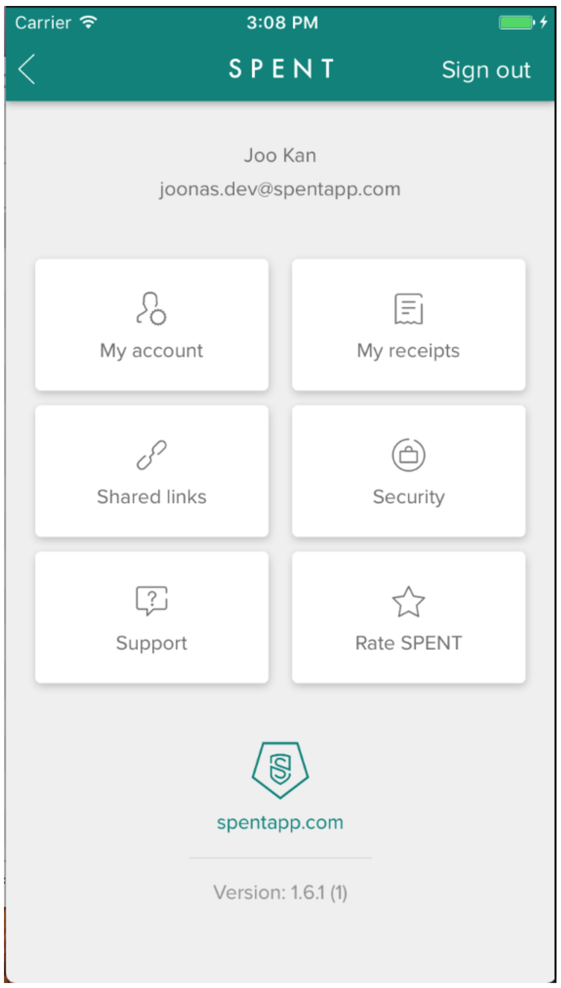
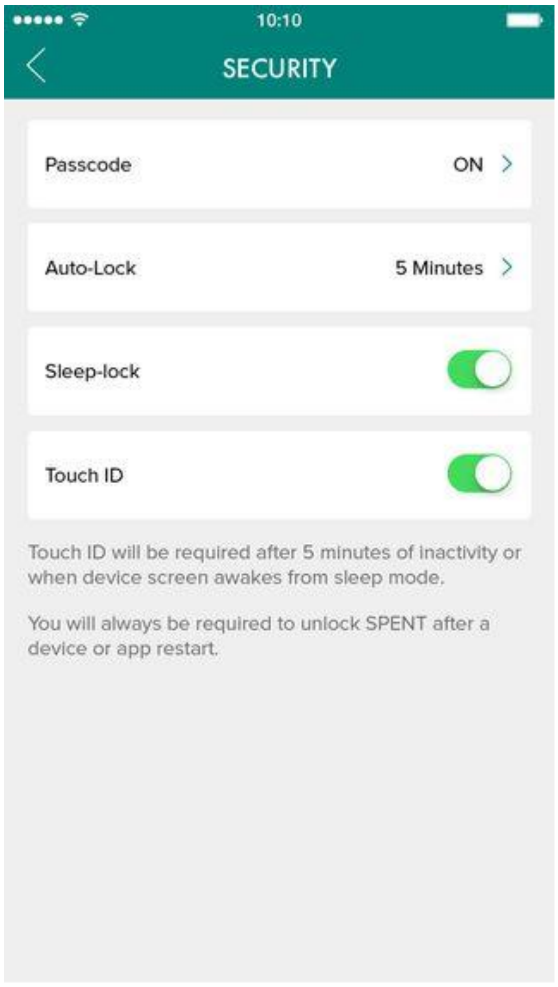


FIGURE 7. New main settings view

As mentioned in the last chapter, the design portion of this project was quite straight-forward. Another view that had to be created from scratch was the view for security settings (Figure 8).



Passcode ON

Auto-Lock 5 Minutes

Sleep-lock ✓

Touch ID ✓

Touch ID will be required after 5 minutes of inactivity or when device screen awakes from sleep mode.

You will always be required to unlock SPENT after a device or app restart.

FIGURE 8. Security settings view

5.2 Implementation

The implementation process was straight-forward. At first, it seemed to be a good idea to disable the iOS-Slide-Menu navigation controller, which was attached to the top left button on the home view. After disabling the link, the attachment of the settings view controller was created for this thesis. Then, the rest of the required view controllers were added to the main settings view and after that the next step would be to work on the UI. After the basic structure of the UI had been created, then the UI elements were connected to all the necessary model classes. After that, the functionalities for the settings were created. For example, security settings should always remember the state they are in when the user goes back to the home view to swipe expenses.

5.2.1 New UI

Developing the new UI of the new settings feature can be divided into the following steps:

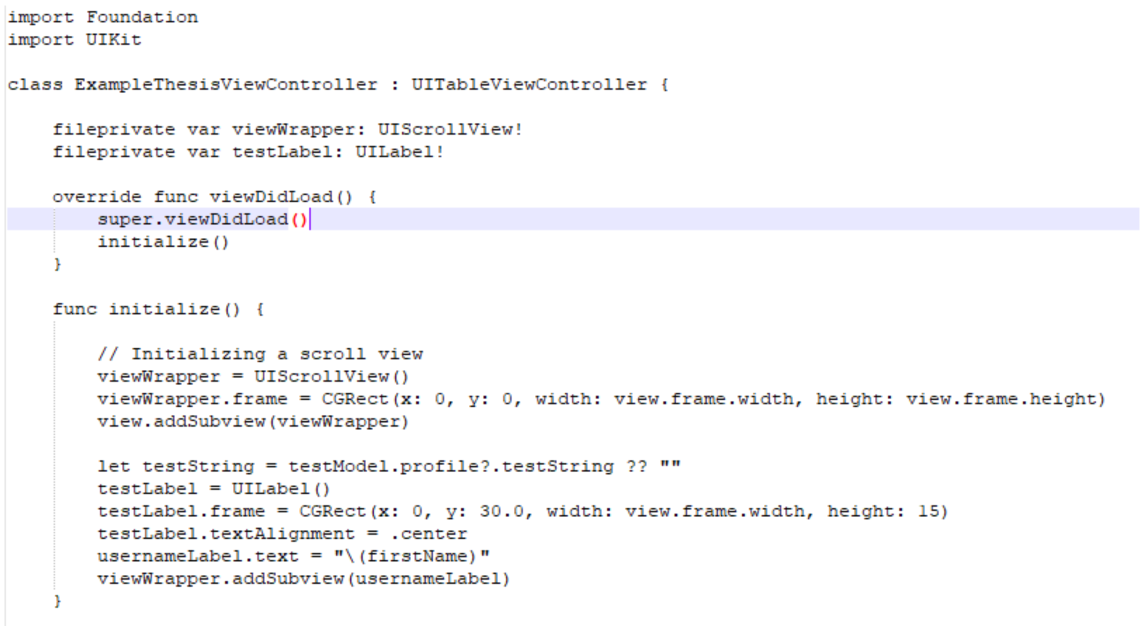
1. Unlinking the old settings view controller class of the top left navigation button.
2. Linking the new settings view controller class to the previously mentioned navigation button.
3. Creating the basic structure of the main settings view.
4. Linking all the necessary view controllers to the new settings view.
5. Adding the minor changes to the old view controllers.
6. Creating the basic view for the security settings view.
7. Creating the open-source licenses list.
8. Adding finishing touches to the UI.

The SPENT application has a file filled with localizable strings. However, now there are no different languages available for the application since the application is only available for downloading in the USA market.

The first thing to do was to disable the left navigation controller from the old settings view and to link it to the new settings view controller. Since the creation of new settings feature meant that the iOS-Slide-Menu library was not necessary anymore, it would have been more beneficial to remove the library altogether and refactor the code. But this would have slowed down the development process even further.

After this procedure, a view controller was created. Its purpose was to manage the new settings view. The way how the settings view was designed was that it uses UILabels, and UImageView and a UICollectionView with six collection view cells. The UI was developed by coding, which means that Storyboard components were not utilized while developing the UI. This meant that the UI had to be created by using Swift for every separate view controller. Basing on experience, the danger for this practice is that there is a high chance of writing messy code because the amount of view objects can become quite huge and their initializations and implementations might end up in lines upon lines of code. In the end, the written code was readable and accepted by colleagues with more experience in the software development.

Assigning the UI elements to the view controller followed a basic structure: a class, which inherits a UIViewController class, has a viewDidLoad() -function which has a custom function added into it. The purpose for this custom function is to initialize UI elements and give them locations, sizes and constraints. Figure 9 represents a piece of test code, which introduces how view controllers were developed in this project:



```
import Foundation
import UIKit

class ExampleThesisViewController : UITableViewController {

    fileprivate var viewWrapper: UIScrollView!
    fileprivate var testLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        initialize()
    }

    func initialize() {

        // Initializing a scroll view
        viewWrapper = UIScrollView()
        viewWrapper.frame = CGRect(x: 0, y: 0, width: view.frame.width, height: view.frame.height)
        view.addSubview(viewWrapper)

        let testString = testModel.profile?.testString ?? ""
        testLabel = UILabel()
        testLabel.frame = CGRect(x: 0, y: 30.0, width: view.frame.width, height: 15)
        testLabel.textAlignment = .center
        usernameLabel.text = "\(firstName)"
        viewWrapper.addSubview(usernameLabel)
    }
```

FIGURE 9. Example code of a view controller written in Swift 3.0

One of the most important components in the view was the collection view. The collection view shows six cells: profile, receipts, shared links, security, support and for app rating. To use the collection view class properly, UICollectionViewDelegate and UICollectionViewDataSource classes were required to be inherited. This enables various collectionView()-functions that alter the size, the contents, what actions are executed when user presses a cell and the number of cells displayed in the view. The easiest way to determine what contents would be displayed within the cells was to create a new separate class which determines the dimensions and the constraints of the UILabel and the UIImageView. After this procedure, the class was implemented in the view controller and it worked as it was supposed to.

After adding all the necessary UI elements into their place in the settings view, the next thing was to add all the view controllers, which were linked to the old settings view, to the new settings view. In the collectionView()-function, which has a didSelectRow parameter included in it, cases were added for different indexes. For example, if the index value is four, then the user will be directed to the settings view and if the index value is six, then the user will be directed to the App Store page of the application. Previously developed view controllers, which were used in the older version of the settings view, were the profile view controller, the support view controller and the gallery view controller for receipt snapshots.

After determining the contents and the amount of collection view cells, the cells and UI components lacked constrains. Constraints are required for ensuring that the UI does not break or do anything else which might be unthinkable. In the custom collection view cell class, the constraints of the UI components in the collection view cell were created by using SnapKit. The constraints for the collection view were made by using the screens size as parameters for views width and height of the menu.

The next task was to alter the contents of the old view controllers. The change passcode button from the profile view was removed and the delete account button was added to the view. In the support view the delete account button was removed and the open-source licenses button was added instead. The sign out button from the profile view was removed and it was added onto the top right side of the main settings view.

After linking the view controllers to the new settings view, the previously developed view controllers were decided to be altered by removing the buttons that were assigned to different locations in the new designs. The change passcode button was removed from the profile view and the delete button was moved from the support view to the profile view.

The next step in the settings view was to create the security settings view. The older SPENT version did not have any means to protect the application other than changing the user’s passcode. The structure of the UI consisted of an UITableView, which listed the custom cells. For these custom cells, an alternative class, which inherits an UIView class, was created. In this class, the dimensions and locations for the UIlabels, UIImageViews and UISwitches were determined. In the security settings view controller, certain components that were needed, were set as enabled or visible and the components that had no purpose for certain parts were disabled or hidden.

5.2.2 Functionality

For determining how to develop the functional side of the settings feature, it was listed out in a similar manner as it was sorted out in the beginning of the chapter 4. Here is the list of functions that were necessary for the improved feature:

1. Displaying the first name and the last name of the user in the main view and changing it dynamically when the user has changed the name in the profile view.
2. Directing the user to SPENT’s home page when the user taps on the SPENT logo.
3. When the user taps on an open-license link, which is in the table view component, creating a way to display the correct license file in a new view controller.
4. Directing the user to the application page on App Store when the user taps on the “Rate SPENT” button.
5. Adding the sign out functionality into the sign out button.
6. Adding the delete account functionality into the delete account button.

7. Security settings:
    a. Adding ON/OFF functionality for the passcode.
    b. Adding the Auto-lock functionality for the sleep-lock.
    c. Adding the Sleep-lock functionality for the application.
    d. Adding the Touch ID locking mechanism for the application and disabling it if the user does not have an iPhone with the Touch ID option.
    e. Changing the string values in the label components depending on the states of the passcode, sleep-lock, auto-lock and Touch ID.

The dynamic change of the first name and last name labels was executed by creating variables that received their value from a model class receiving and changing the user information from a Realm database, which then receives its contents from the backend. In addition to this, a NotificationCenter-class was required. The purpose of this class is to post and observe data when the code is executed. When the user decides to change the values in the first name and last name fields in the profile view controller, this sends a que to an observer in the settings view controller, which then obtains new values from the Realm database for the first name and last name values.

The way to direct the user to SPENT's home page was to first create an object of the UITapGestureRecognizer class and then to add the object into the spentLogo class which is an object created out of an UIImageView class. For the gesture recognizer action, the author created a function, which opens SPENT's home page on the Safari mobile browser.

For open-licences, two separate view controllers were created: one that lists all the licenses and one that displays the contents of those licenses. For the view controller that displays all the licenses, an UITableView object was created, and for the other view controller, an UIWebView object was used. UIWebView has an ability to view webpages and for these licenses, raw text pages were found in their respective repositories. (23.) To create a proper user experience, whenever the user selects a row on the list, the row gets highlighted.

For the “Rate SPENT” -button, a function was created. This function opens SPENT’s app store page where the user can rate the app. This procedure functions in the same manner as opening SPENT’s website launch does.

The sign out functionality for the button was easy to handle since it was already created. An object was created from a class, which has a function, that executes the logout action and which was added into the sign out navigation button’s navigation action.

The delete button, which was created before in the profile view controller, had a function which deletes the user profile. When the button was moved to the profile view controller, the delete profile function was also relocated in the profile view controller as well.

The security settings view was the task that required more work. The older version of the application only had a view where the user could change their passcode. The user had no way to turn off the passcode and had no functionalities for the auto-lock, sleep-mode or touch ID. Because of the strict project deadline, the new settings feature from this point onwards was developed by an experienced iOS developer.

For the security settings, ensuring that the details that were set in the security settings were stored somewhere when exiting from the security settings menu. For this purpose, the userDefaults-class was used. This class allows the user to store information locally without using any kinds of databases. Only a value and an ID are needed to distinguish the set data from each other.

An important feature added into this application was a timer, which starts to count down the duration when there are no actions happening on the screen. In the end, the developers created a class which handles whether there should be a passcode visible or not. It also sets the auto-lock value and handles whether the Touch ID is being enabled or not.

A co-worker implemented a timer, which runs on the background of the application. First a Timer object was created. It then calls a function that starts to count from 0 every time the user stops to touch the screen. The value can be changed in the security settings view. The timer object gets its value for the duration from the list of user default values. The user can also toggle the timer on or off. If the user turns off the timer, then the timer's default value gets multiplied by -1, which was enough for not launching the timer. Encapsulating the timer with a Boolean if-statement was not necessary because of this possibility.

Instead of directing the user to another view controller while setting a new value for the timer, the last developer had created a custom class for handling dialog pop ups. For the dialog pop up function, a custom class was used. This class was created by previous developer for prompting "ok" or "cancel" dialogues. This class also functions in a way that it is scalable. The way it scaled was to create an object of the custom UIDialogViewController class. Then it adds those dialogue messages, for example, "5 minutes" or "10 minutes", into the dialogue popup window. Every time the UIDialogViewController class' addAction function is called, the function adds the dialogue option into popup and does not overwrite the previous function of the similar name. Instead, it stacks up.

The touch ID gets handled in the security settings view controller by a function, which enables the Touch ID feature. This is then encapsulated by a clause that checks if the device supports the Touch ID feature. If not, then the Touch ID row will not be shown at all. The function which enables the Touch ID feature, gets its value from userDefaults.

The labels, which portray the state of security settings on the bottom part of the screen, were executed by taking information from the state of UI components in the security settings. Using these states as flags for changing the contents in the labels worked appropriately. The text contents were derived from a file filled with localizable strings.

6 TESTING

Testing was a significant part of the development process of the settings view. The firm had two parties reserved for testing: inner testers and outside testers. In inner testing, there were developers and a designated tester performing various testing tasks on the new feature. There were also outside testers performing their own testing tasks on the new feature. The following chapters will describe several different methods of testing that were used for testing the new feature.

6.1 Testing practices

Since the company employed their own dedicated tester in January 2017, no automated testing was implemented in the testing routines. This meant that those routines had to be performed manually. This did not justify leaving out automated testing, but at the time the company lacked resources and tools in the testing department. Figure 10 shows the testing methods SPENT uses in the product testing:

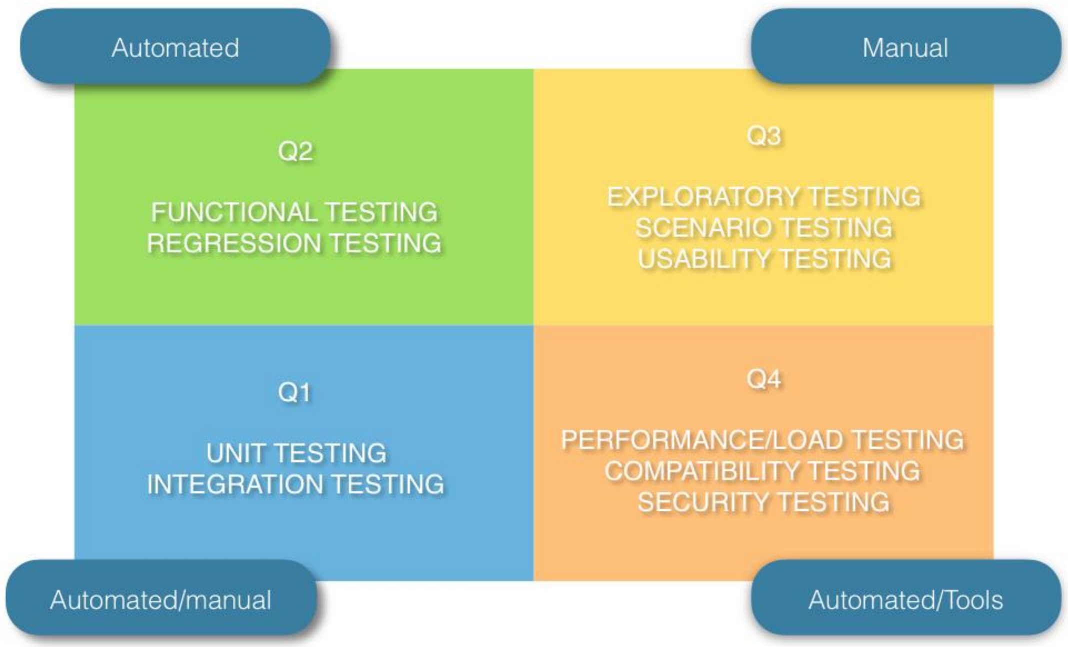


FIGURE 10. Testing methods. Q1 – Q4 mean what testing processes are meant to be done first. (24.)

Here are the explanations for the different testing methods. It is necessary to point out that every type of testing which is categorized to be automated, was executed manually.

In the first quarter, program developers usually perform unit testing and integration testing. In unit testing the developer ensures that the code has been coded correctly. This includes the new lines of code and the old, possibly refactored code. The unit testing happens immediately when coding has been finished and the application in the client device has been decided to build. (24.)

Integration testing means that the developers test that all the software components in the application work together as intended. Testing is executed when the developer has finished developing the new component for the application, for example the new settings view works as a great example of a software component. (24.)

In the second quarter, the functional testing was executed by developers and internal testers and the regression testing was executed by internal testers. Functional testing is a practice where testers ensure that all the features function as intended. At this phase, if the tester has any automated testing tools, a partial test (also known as a smoke test) or a complete test will be executed. Since the internal tester was recruited in January 2017, which is also the month when the author worked on the settings view, the tester did not have enough time to create sufficient scripts for the automation testing tools. Regression testing means that the tester tests if the application behaves in a way it is supposed to after changing the code structure or adding new features into the application. (24.)

In the third quarter, all the testing methods were executed manually. External testers and inner testers performed exploratory testing the purpose of which was to find bugs in a way that automated testing or other means of testing could not find. Exploratory testing went through all the parts in the application, but it is often directed to a specific feature for the release. In exploratory testing testers tried to check how the application recovers from crashes and how it handles errors. Testers put invalid data into text fields. They performed odd movements on the UI and studied how application performs when there is no Internet connection or when the application moves between the foreground and the background. (24.)

Scenario testing was important since it means that the application is tested in a way where testers verify if the application functions and behaves as it should from a casual user’s point of view. It was performed by an internal tester. The scope of scenario testing was more flexible compared to aforementioned testing cases, but it was still considered as a legitimate way of testing. (24.)

Usability testing means that the tester goes through UI elements and points out what is wrong with them or may propose a change so that the usability would feel more fluid. The tester also points all the UI bugs that might occur during testing. This method of testing was performed by developers and inner testers. (24.)

In the fourth quarter the product then will be tested to find out how it handles huge loads of content, how it performs other mobile devices and how secure the application is. In performance or load testing, the tester tries to find out how the application performs when dealing with a huge amount of information. In this case, since there was no automatic testing available, the tester created multiple pieces of expenses to be loaded for the home view and for home and business folder views. The aim of this kind of testing is to find out how the application behaves under a lot of stress, whether the application still work fluidly even if it has many networking cycles and loading is happening on the background. (24.)

In compatibility testing testers try to find out how the application performs on different devices. In case of SPENT’s application, testing needed to point out the incompatibility issues on different devices, such as differences in the UI or performance. (24.)

In security testing testers test out the security of the application, such as login and sign up functionalities, passcode security, both digit inputs and Touch ID. (24.)

6.2 Bug management

The bugs found during testing were marked into a Trello board, which was reserved for iOS developers. Trello is a web-based task management tool, where a team can create different boards and where team members can add lists to sort out different tasks. These lists can symbol the status of a task. One list can be set for tasks that are “in progress” or “to be tested” and there can be a list for tasks that have been done. Figure 11 below shows the development cycle of SPENT’s development team:

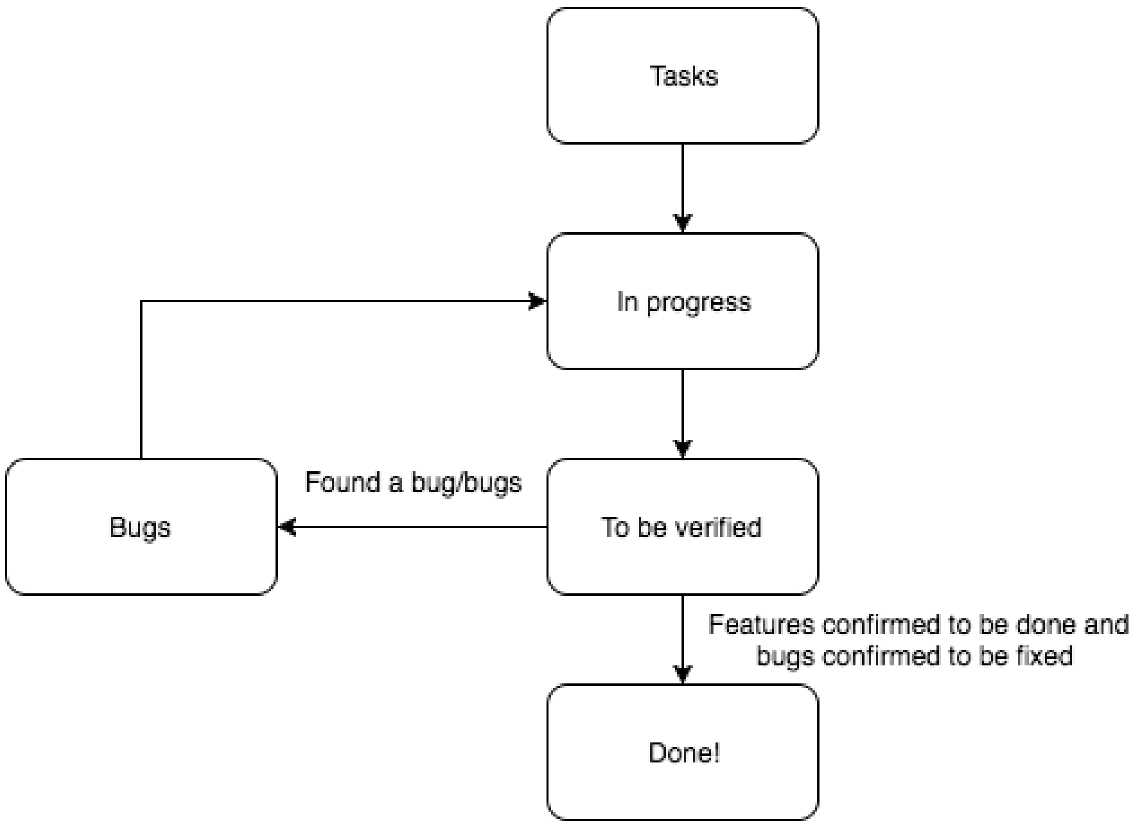

Tasks
In progress
Found a bug/bugs
Bugs
To be verified
Features confirmed to be done and bugs confirmed to be fixed
Done!

FIGURE 11. Development process of the SPENT iOS application.

7 CONCLUSION

The development of the new settings feature was successful and it feels that I learned a lot of how to develop an application feature in a startup environment. SPENT had different teams for different stages of development: a concepting team, a designing team, a programming team and, in the future, a testing team. I participated in the programming team, more specifically in the iOS development. I learned how to develop a feature by using the MVC architecture as a basis for the mobile software development. I learned how to create controllers and different views by using Cocoa Touch and Foundation frameworks. I received a lot of help from my co-workers when I was running out of time and could not deliver the finished code on time.

In the beginning, it was quite hard for me to review the code that had been created at the start of the SPENT development. There were changes in the iOS development team and programmers had their own personal sense of what type of code would be good. Alongside of this project, a lot of refactoring was done and things were executed differently. This made merging the settings feature branch to the main development branch a pain. Luckily, the merging was a success. It was my first time to merge branches that were as extensive as these two branches were.

Even though I achieved the aim of finishing the feature, instead of learning everything necessary, I found myself more curious of how to implement new features and my own personal aim was to create or utilize the model classes more often. This project, however, did not require me to utilize any other third-party frameworks extensively. These third-party frameworks that I used to some extent were Realm and Snapkit.

After receiving commentsfor using the userDefaults class instead of using the key chain functionalities for storing local data, I realized that I should not have implemented the data storing functionality in that way. The reason is that userDefaults stores data in a plain text file and does not encrypt that data at all, whereas storing critical bits of data into the key chain means that it gets encrypted (25.)

After reviewing and polishing the feature I had created, the new settings view was accepted and then it was implemented into the application itself by merging the settings development branch into the development branch and then to the master branch. I am quite happy about my input on the project. It feels good to see your own touch in the final product. Unfortunately, this application is not available in Finland and because of that I cannot download the application on my phone and see if the feature is still in use.

This project was a good way to see what it is like to work in a startup company. A small team developed a product, which has a lot of promise on the American soil. It requires a big amount of effort to develop the product even further and to keep up with the competition. A huge appreciation for software development as a practice is necessary.

REFERENCES

1. Meet Erno: Founder and CEO. 2017. Blog post. Date of retrieval: 25.4.2017. https://spentapp.com/blog/meet-erno-founder-ceo/.
2. Our Offices. 2017. SPENT Website. Date of retrieval: 25.4.2017. https://spentapp.com/about-us/.
3. iOS. 2017. Wikipedia, the free encyclopedia. Date of retrieval: 19.1.2017. https://en.wikipedia.org/wiki/iOS.
4. About the iOS Technologies. 2014. Apple documentation. Date of retrieval: 11.3.2017. https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html.
5. Cocoa Touch Layer. 2014. Apple documentation. Date of retrieval: 18.5.2017. https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html#//apple_ref/doc/uid/TP40007898-CH3-SW1.
6. Media Layer. 2014. Apple documentation. Date of retrieval https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/MediaLayer/MediaLayer.html#//apple_ref/doc/uid/TP40007898-CH9-SW4. Date of retrieval: 18.5.2017.
7. Core Services Layer. 2014. Apple documentation. Date of retrieval: 18.5.2017. https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreServicesLayer/CoreServicesLayer.html#//apple_ref/doc/uid/TP40007898-CH10-SW5.
8. Core OS Layer. 2014. Apple documentation. Date of retrieval: 18.5.2017. https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreOSLayer/CoreOSLayer.html#//apple_ref/doc/uid/TP40007898-CH11-SW1.

9. Cocoa (Touch). 2015. Apple documentation. Date of retrieval: 25.1.2017. https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html.
10. Foundation. 2017. Apple documentation. Date of retrieval: 17.5.2017. https://developer.apple.com/reference/foundation.
11. UIKit. 2017. Apple documentation. Date of retrieval: 18.5.2017. https://developer.apple.com/reference/uikit.
12. Xcode. 2017. Wikipedia, the free encyclopedia. Date of retrieval: 11.3.2017. https://en.wikipedia.org/wiki/Xcode.
13. What is version control. Tutorial. Date of retrieval: 18.5.2017. https://www.atlassian.com/git/tutorials/what-is-version-control.
14. Bitbucket. 2017. Wikipedia, the free encyclopedia. Date of retrieval: 19.5.2017. https://en.wikipedia.org/wiki/iOS.
15. Model-View-Controller (MVC) in iOS: A Modern Approach. 2016. Blog post. Date of retrieval: 11.3.2017. https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach.
16. Advantages of MVC Architecture. 2009. Blog post. Date of retrieval: 27.4.2017. http://javabynataraj.blogspot.fi/2009/05/14-advantages-of-mvc-arch.html.
17. Swift. 2017. Apple documentation. Date of retrieval: 14.2.2017. https://developer.apple.com/reference/foundation.
18. Swift (programming language). 2017. Wikipedia, the free encyclopedia. Date of retrieval: 27.4.2017. https://en.wikipedia.org/wiki/Swift_(programming_language).
19. Swift vs. Objective-C: 10 reasons the future favors Swift. 2015. Online article. Date of retrieval: 11.5.2017. http://www.infoworld.com/article/2920333/mobile-development/swift-vs-objective-c-10-reasons-the-future-favors-swift.html.
20. Access Control. 2017. Apple documentation. Date of retrieval: 15.5.2017. https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/AccessControl.html.

21. Automatic Reference Counting. 2016. Wikipedia, the free encyclopedia. Date of retrieval: 15.5.2017. https://en.wikipedia.org/wiki/Automatic_Reference_Counting.
22. How it works. 2017. SPENT website. Date of retrieval: 26.4.2017. https://spentapp.com/how-it-works/.
23. UIWebView. 2017. Apple API Reference. Date of retrieval 19.5.2017. https://developer.apple.com/reference/uikit/uiwebview.
24. Pirnes, Marko 2017. Test strategy. SPENT documentation. Oulu, Finland.
25. Are You Storing Sensitive Data in NSUserDefaults? Stop Doing That! 2014. Blog post. Date of retrieval: 11.5.2017. https://www.andyibanez.com/nsuserdefaults-not-for-sensitive-data/.