Object-Oriented Development of a Smart Phone Application From an Existing Web Service

Jon Borglund
Abstract

Object-Oriented Development of a Smart Phone Application From an Existing Web Service

Jon Borglund

Abstract Lissly.com is a monitoring service that monitors the presence of brands, companies and people in social media. As a complement to the web based tool, an application for smartphones is needed. This thesis presents the adaption of the web application to a smartphone implementation. This is done by creating a design document using object-oriented principles, and an implementation using the Objective-C programming language. The design document and the implementation forms a foundation for future development of the service.
# Contents

## 1 Introduction

1.1 Background ................................. 1  
1.2 Problem description ....................... 1  
1.3 Purpose ...................................... 1  
1.4 Method ...................................... 2  
1.5 Limitations ................................. 2

## 2 Method

2.1 Scrum ...................................... 2  
2.1.1 Roles ..................................... 3  
2.1.2 Backlog .................................. 3  
2.1.3 Sprint ................................... 3  
2.1.4 Daily Scrum ............................... 3  
2.1.5 Sprint review ............................. 4  
2.1.6 Burn down chart ......................... 4

## 3 Design document

3.1 Requirements ............................... 5  
3.1.1 Requirements specification .............. 5  
3.2 Use cases ................................. 5  
3.2.1 UC2 View account list .................. 5  
3.2.2 Use cases ............................... 6  
3.3 Class diagram .............................. 7  
3.3.1 Development of the class User .......... 7  
3.3.2 The class diagram ....................... 8  
3.4 User interface ............................. 8

## 4 Implementation environment

4.1 Objective-C ............................... 9  
4.1.1 Memory Management ..................... 9
II  Design Document  33

III  Scrum backlog  42
1 Introduction

1.1 Background

In the end of October 2010 Entertainitity AB released the web based tool Lissly. Lissly is a monitoring service that monitors the presence of brands, companies and people in social media. The tool targets mainly PR agencies, communications agencies and marketing departments. As a complement to the current web tool, there is a need to follow what is happening in social media through smartphones.

Lissly defines social media sources as social networks, blogs, micro-blogs, forum, video and images. Well known services are e.g. Facebook, Twitter and YouTube. The user can add keywords that are monitored. When a keyword matches text from a source, a so called mention is created with the specific post. The user can then view a mash-up of the mentions, plots with mention statistics over different time periods, and collaborate by highlighting mentions as positive or negative and optionally leaving feedback.

1.2 Problem description

The problem is to determine which functionality of the existing web tool that is important to include in a handheld device, and how the existing functionality shall be presented and adjusted to fit the new format, with both user accessibility and object-oriented design in mind.

1.3 Purpose

The purpose of the report is to describe the object-oriented development of a smartphone application from an existing web service.
1.4 Method

The method that I have used to approach the problem is object-oriented[1] principles and the creation of a design document. The document will contain use cases, class diagram and information which will describe the internal structure of the application. Both the planning and the implementation are done using an agile development method called Scrum[2].

1.5 Limitations

The aim of the design document is to be general and not to focus on a specific implementation. The implementation done here does not need to be the final version of the application.

2 Method

The software application is planned and designed using an object-oriented[1] approach and implemented for iOS on the iPhone. I have chosen parts of the Scrum agile software development method[2] for this project, but with me as the only developer. The daily consultation is held with the product owner, who also takes on the role as scrum master. The plan is to create a backlog, which will be divided into sprints and then monitored with burn down charts.

2.1 Scrum

Scrum[2] is an agile methodology for system development, which is based on iterative incremental development. This approach optimize predictability and control risk. Every implementation in Scrum is uphold by three pillars of empirical process control: transparency, inspection, and adaptation.
2.1.1 Roles

There are three main roles in a scrum team:

- The *Product owner* receives, handles and priorities extensions or changes for a product.

- The *Team* consists of developers, the actual workers.

- The *Scrum master* is like a coach for the team, secures the team’s progress and that the task at hand is understood and followed. The Scrum master help those outside the Scrum team understand which of their interactions with the team that are helpful and which are not.

2.1.2 Backlog

The backlog is a list of tasks that are derived from the product requirements. The list is ordered by priority. The development time for each task should be estimated to 4-8 hours. Tasks that are too large can be split-up and tasks that are too small can be combined with other tasks.

2.1.3 Sprint

A sprint is an iteration in the development process. Each sprint has its own sprint backlog, which contains the most important tasks from the main backlog. The time runs over a time-box, which is the limited time allocated for the sprint. During the sprint the time for the tasks at hand are re-estimated, so the total time for the sprint backlog is not necessarily constant. If there is not enough time to complete all the tasks in the sprint, the remaining ones are moved back to the main backlog.

2.1.4 Daily Scrum

Daily Scrum[2] meetings are used to forecast the next 24 hours in the development process and inspect the work since the last meeting. The scrum
master asks each member in the group three questions:

1. What have you accomplished since yesterday?

2. What will you accomplish until the next meeting?

3. Are there any obstacles in your way?

The benefits with this approach are that challenging problems can be identified early and then be discussed and solved by the team. Daily Scrum also improves communication, eliminates other meetings and promotes quick decision making [2].

2.1.5 **Sprint review**

The review is planned from the beginning of each sprint and is used to show the product owner the project progress. This is achieved by reviewing the states of each task in the sprint, and demonstrating new functionality since the last time.

2.1.6 **Burn down chart**

A burn down chart is a line chart with two lines, one with the sum of remaining approximated task hours and one with real work hours left in the time-box. Each day a new point is added on each line, as the approximated sum and the work hours decrease, the graph *burns down*. When the real work hours burn faster than the approximated, it is likely a sign of a problem in the work progress. It is possible that the remaining hours increase, because tasks can be re-estimated and new ones can be added during the sprint.

3 **Design document**

The main idea of the design document is to get an overview of the application’s basic architecture, not a detailed road map to the final product. This
means that most of the details are omitted and the document is kept on a high level.

3.1 Requirements

First I studied the Lissly web tool and then a meeting was held with the product owner from Lissly AB. The requirements of the application were gathered, then analyzed and negotiated, and finally documented.

3.1.1 Requirements specification

The resulting requirements specification consists of a document describing the functionality and some rough sketches of the user interface. The specification is mainly focused on the iPhone version of the application. The specification describes the general layout of the application, navigation principles, which views that are supposed to be included and what kind of functionality each view is supposed to provide. The specification was also altered during the Scrum iterations. The specification (in Swedish) can be seen in the appendix.

3.2 Use cases

Use cases[1] are used to model and represent units of functionality provided by the system. The use cases were mainly derived from the requirements specification.

3.2.1 UC2 View account list

Here is an example of Use Case 2, view account list. The purpose is to see the available accounts. The only actor in this case is the authenticated user. Except from the basic flow there are two alternative flows depending on the number of accounts available to the user.
Figure 1: Use case 2

Purpose: To view available accounts

Actors: User

Pre: User is authenticated

Basic flow:

1. User views accounts

Alternative flows:

1a. There is no account

1. User is notified about the problem.

1b. There is only one account

1. User is forwarded to the project list view.

3.2.2 Use cases

None of the use cases are fully dressed, because implementations might not share the same design. The use cases are more a summary of the functionality needed, used to guide the programmer. A total of 15 casual use cases can be seen in the design document[1].
3.3 Class diagram

The class diagram only covers the data model of the application. It was constructed by adding empty classes directly into the class diagram, hence it was very much like a conceptual model from the beginning, then continued to evolve with attributes, relational constraints and methods.

3.3.1 Development of the class User

Figures 2 and 3 show the development of the class User and its relation to the class Account. The class User is the root of the hierarchical model.

![Figure 2: Empty User class.](image)

In figure 2 classes and an undefined relation between them were added. By building the whole diagram this way a conceptual diagram emerged which provides a good overview of the model.

![Figure 3: User class with attributes.](image)

In figure 3, most of the attributes were added. Obvious getters and setters are omitted, because they reduce the overview of the diagram.
Figure 4: User class with attributes, methods and constraints.

Figure 4 includes the relational constraints and the public methods, i.e. the objects’ external interface.

3.3.2 The class diagram

The full class diagram can be seen in the design document, which is included in the appendix.

3.4 User interface

Human computer interaction principles are omitted in the design document. The recommendation is to follow the device’s standard approach to navigation. This is much better than applying a custom navigation or layout. The user will be able to learn the application quickly because of recognition, and the user’s mental model will be close to the system model from the beginning [3].

4 Implementation environment

iPhone applications are implemented using the Objective-C programming language[4], with the iOS software development kit, which includes the frame-
work Cocoa Touch. The platform used is the integrated development environment (IDE) Xcode 4[5] and related applications on Mac OS X Snow Leopard.

4.1 Objective-C

Objective-C[4] is defined as a superset of the ANSI C language. Its extensions to C are more or less based on Smalltalk. Objective-C was designed to include full object-oriented programming capabilities. Even though the language is compiled, it still employs a dynamic run-time environment which makes it very flexible. The object syntax is a bit different compared to C++ and Java, for example when calling a method on an object. This is an example of an Objective-C method call:

```object method:argument];
```

The Java equivalent to the above method call:

```object.method(argument);
```

4.1.1 Memory Management

Since Mac OS X 10.5, the Cocoa programming environment has automatic memory management, but this is not supported in iOS. The traditional Objective-C memory management mechanism is reference counting, using the methods `retain`, `release` and `autorelease` [7]. An object claims ownership of another object by increasing its reference count:

```NSString *s = [[NSString alloc] init];
/* s is allocated and has reference count 1 */
[s retain];
/* s has reference count 2 */
```

When the owner no longer needs the object it must decrease the reference count again, by releasing it:
An object is unused if its reference count is zero, and will therefore be deallocated.

An autorelease pool consists of objects that have been marked for autorelease. When a pool is drained all the objects within it are sent the message release. The pool is drained by manually sending it the message drain or automatically when it reaches the end of its scope.

More information and observations will be mentioned in section 6.

4.2 iOS Software Development Kit

The iOS is a software development kit (SDK) by Apple Inc., to develop applications for devices running iOS. The top layer in the SDK is the Cocoa Touch, which consist of frameworks, APIs and a runtime environment. When using Cocoa Touch the application will inherit appearance and behaviors of iOS, but the Cocoa Touch frameworks are optimized for touch-based interfaces.

One of the primary frameworks within Cocoa is UIKit which provides the basic tools and infrastructure to implement graphical and event-driven applications for iOS. Another important framework is the Foundation, which provides the base objects and collections used throughout Cocoa. There are also a lot of special frameworks within Cocoa, ranging from advanced mathematics to game development. Alongside the SDK Apple Inc. has guidelines, such as object-orientation patterns and code conventions.

4.3 Development environment

Mac OS X Snow Leopard includes a full suite of developer tools.
4.3.1 Xcode 4

Xcode 4[5] is the official Mac OS X IDE. It includes a code-editor, a graphical debugger and integrated Objective-C, C and C++ compilers. It is well integrated with the Cocoa frameworks and debugs code on-the-fly. It has built in support for version control, supporting both git and subversion.

4.3.2 Interface builder

Interface builder was a stand alone application prior to Xcode 4[5], but is now integrated into the IDE. With a drag-and-drop workspace it is easy to design and test a graphical interface without writing any code. The interface source files, called NIB files, consist of frozen objects. The integration with Xcode makes it possible to attach methods in the project code directly to these frozen objects with just a few clicks.

4.3.3 iOS Simulator

The iOS simulation environment makes it possible to build and run iOS applications directly on the development computer, without any external device. The iOS simulator application presents the iOS user interface, where it is possible to interact with it in several ways: via the mouse, by simulating hardware shake etc.

4.3.4 Instruments

Instruments is a tool used for debugging, analyzing and profiling. Functionality provided includes data recording, memory leak analysis and zombie detection.
5 Lissly API

The Lissly web service provides access points to its data through a web API. It is defined as a set of HTTP request messages along with a definition of the structure of response messages. The response can be either in XML or JSON.

5.1 Functions

The current available functions are:

- **login** which is used to authenticate or get information about a user.
- **get_user_tree** which is used to get information about accounts, projects and keywords for the current authenticated user.
- **overview** which is used to get statistics for a project or a single keyword from a specific time period.
- **show mentions** which is used to get mentions of a project from a specific time period.

5.2 Request example

Below is a URL to load the information about accounts and projects with a JSON response.

http://api.lissly.com/member/get_user_tree.cgi?accesskey=X4fjr0n9YT4NRBGAZCASg&userkey=QWXHQqGs&format=json

There are a three parameters in this request:

- **accesskey** which is specific to the application, this is needed in all the requests.
- **userkey** which is used to identify the current authenticated user.
• *format* which is optional and is used to override the default XML format.

An excerpt of the response to the query:

```json
{
  "apiversion": "1.0",
  "accounts": {
    "account": {
      "name": "Demo - Premium",
      "projects": [
        "project": {
          "keywords": [
            "keyword": {
              "lang": "sv",
              "name": "Almedalen",
              "id": "5294"
            },
            "name": "Almedalsveckan",
            "id": "1453"
          ],
          "keywords": [
            "keyword": [
              "lang": "sv",
              "name": "Håkan Juholt",
              "id": "5266"
            ]
          ]
        }
      ]
    }
  }
}
```

If the format parameter is left out, the response is in the default XML format:

```xml
<?xml version="1.0"?>
<xml>
  <apiversion>1.0</apiversion>
  <accounts>
    <account>
      <id>9</id>
      <name>Demo - Premium</name>
      <projects>
        <project>
          <id>1453</id>
        </project>
      </projects>
    </account>
  </accounts>
</xml>
```
6 Implementation

This section covers the implementation of the iOS application using Scrum.

6.1 Backlog

To set up the Scrum backlog, I needed to analyze the design document and the requirements, and derive tasks and add them to the backlog. The time-box for each task was roughly estimated, the commit order was then determined in negotiation with the product owner. The most important tasks were placed at the top of the backlog.

6.1.1 Backlog excerpt

Excerpt from the backlog:

...
<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Wrapper</td>
<td>Connection, parse JSON response and cache.</td>
</tr>
<tr>
<td>Model</td>
<td>User, Account, Keyword, Source, Mention, MentionGraphic, Project, Statistics at least empty dummy classes according to the class diagram in the design document</td>
</tr>
<tr>
<td>Log in view/controller</td>
<td>The login view with the form, the functionality to login and display error message if login failed. The login shall take user/pass and then set a apikey via the preferences.</td>
</tr>
</tbody>
</table>

...  

The entire backlog can be seen in the appendix.

6.2 Sprint 1

The initial sprint was set up by taking the top of the backlog and more thoroughly re-estimating how much time each task should take. The organization of the sprint was to cover the application’s main features. The time-box for the first sprint was set to around 50 hours.

This subsection presents examples of the implementation and the Scrum burn down chart.

6.2.1 Model

The model consists of the classes seen in the class diagram.
6.2.2 The class User

The Objective-C declaration of the class User looks like this.

User.h
1 #import <Foundation/Foundation.h>
2 #import "Account.h"

In the header file lines 1-2 are imports, the#ifimport directive is an improved version of C’s#include. The difference is that#import ensures that the header file is only included once, thus there is no need for include-guards.

User.h
3 @interface User : NSObject {
4 }

Line 3 defines the class User with the parent class NSObject.

User.h
5 @property (nonatomic, assign) NSInteger userId;
6 @property (nonatomic, copy) NSString *name;
7 @property (nonatomic, copy) NSString *apiKey;
8 @property (nonatomic, readonly) BOOL isAuthed;
9 @property (nonatomic, readonly) NSMutableArray *accounts;

Lines 5-9 are the declared properties of the class, as seen in the design document. TheNSMutableArray *accounts is like a hash map which holds references to the user’s accounts.

A property is a notation used to substitute instance variable declarations and define the rules of the accessor methods. Depending on what kind of properties that are put in parentheses, the accessor method will have different ways to handle concurrency and memory management, as explained in section 4.1.1.1. Different properties include:

atomic The property is atomic.
nonatomic The property is non-atomic.
assign The property is a reference to the value last assigned.
retain The property is a reference to the value last assigned, and the reference count is increased.
copy The property is a copy of the value last assigned.
readonly The property is read-only.

User.h

```
10  - (void) auth;
11  - (void) deauth;
12  - (void) addAccount:(Account *) account;
13  - (Account *) accountById:(NSIntege) accountId;
14  - (NSIntege) accountCount;
15  @end
```

Lines 10-14 are public instance method declarations. The “-” before each method denotes that they are instance methods, a “+” would mean that they are class methods. The +/- symbols do not imply whether they are public or private, unlike the class diagram. This can be done using the directive @public or @private above the set of declarations.

The Objective-C implementation of the class User looks like this:

User.m

```
1  #import "User.h"
2  @implementation User
3  @synthesize name, userId, apiKey, isAuthed, accounts;
```

The directive @synthesize is synthesizing the properties declared in User.h and automatically generates the setters and getters. The instance variable “name” can now be accessed through the methods name and setName. In this example: notice the absence of the “get” on the getter, which is one of Apples coding conventions[4].
Example sending messages

```swift
[userObject setName:@"John"];
userObject.name = @"John";
NSLog([user name]);
NSLog(user.name);
```

Objective-C has a boolean type `BOOL`, where `TRUE` and `FALSE` has been replaced by `YES` and `NO`. This is an attempt to make the code more conversational[4].

```swift
-(void)auth {
    isAuthed = YES;
}
```

### 6.2.3 Lissly API

The core of the application is its connection to Lissly web service.

The implementation handles each API method as a custom request. The requests inherit from the super class `ApiRequest`, and each subclass encapsulates the setup of the specific request and handles the response.

For best performance the JSON version of the Lissly API was chosen, because the JSON syntax is less verbose than XML[6]. Unfortunately there is no default library in Cocoa that supports JSON so a third-party library was used.

When requesting the connection to the server the `NSURLConnection` is used, which is a part of the Foundation framework.

```swift
-(void)performRequestWithString:(NSString *)url {
    NSURLRequest *request = [NSURLRequest
        requestWithURL:[NSURL URLWithString: url]
        cachePolicy:NSURLRequestReloadIgnoringCacheData
```
The URL is built and then sent from the sub class to the super class. The delegation pattern is widely used in Cocoa, and recommend by Apple[7]. The naming convention of delegation methods are “Can”, “Did”, “Will” and “Should”. The connection’s delegate is set to self, so the subclass will also capture the asynchronous call to the connection delegate method connection-DidFinishLoading, where each API implementation can handle the response. Here is the ApiLogin:

```swift
@ApiLogin.m
- (void)connectionDidFinishLoading:(NSURLConnection *)connection
{
    [self parseJSONResponse];
    if([result objectForKey:@"error"] != nil ||
        [result objectForKey:@"member_nick"] == nil) {
        [delegate loginDidFail:self withData:result];
    } else {
        [delegate loginDidSucceed:self withData:result];
    }
}
```

Each API also has its own delegation protocol. A protocol is a collection of method declarations that are made available to any class that wants to implement all or some of them. This makes it possible for unrelated classes to inherit common behaviors[4]. Example of the ApiLoginDelegate:

```swift
@ApiLogin.h
@protocol ApiLoginDelegate <NSObject>
@required
- (void)loginDidSucceed:(ApiLogin *)api withData:(
    NSDictionary *)userData;
```
6.2.4 Main navigation

According to the design document the iOS application follows the standard iPhone drill down navigation. This is accomplished by using the `UINavigationController`, which manages the navigation of hierarchical content. The navigation controller holds a stack of `UIViewController`. When a view is pushed or popped it appears or disappears on the screen.

![UINavigationController](image)

Figure 5: UINavigationController

To solve the problem that some views should be skipped in the navigation, depending on different number of accounts, a protocol for `shouldBeSkippe-
dOrPushed is created. If a view is about to push a new view, the new view is asked if it should be pushed, which then responds with YES or NO. Each view that can be skipped implements this protocol.

The view controllers in the application mainly inherit directly from the Cocoa UIViewController, which has an attribute for the UIView instance, the actual view.

The layout of the views are mostly designed in Interface Builder. The views are bridged with the controller by defining IBOutlets or IBActions. An outlet is an instance variable that points from one pointer to another. The pointer is easily assigned from Interface Builder with drag-and-drop.

Example of the LoginController:

```objective-c
17 @property (nonatomic, retain) IBOutlet UITextField *passwordField;
18 @property (nonatomic, retain) IBOutlet UITextField *userField;
19 @property (nonatomic, retain) IBOutlet UIButton *submitButton;
```

Figure 6: Interface builder, login view

6.2.5 Result of sprint 1

The burn down graph, seen in figure 7 shows that the goal of sprint 1 was not reached within the allocated time. This was because some of the tasks in the
sprint log was under-estimated, and took much longer time than expected. Extra time for contingency was added during the sprint.

![Sprint 1 burn down chart](image)

Figure 7: Sprint 1 burn down chart

The sprint was estimated to around eight working days, but it took almost seventeen, were each day is eight hour long. This is why the linear time ends at around day 7, as seen in figure 7. The estimated time is almost constant between day five and ten. After some studies some extra tasks were added to the sprint, hence the spike. In the end, the tasks that were not finished during the sprint were put back into the backlog.

After the first sprint, the basic functionality was working: the login, the main navigation, the project details menu and the load mention. The application is further described in section 7.

### 6.3 Sprint 2

This section describes parts of the implementation during sprint 2. The original approach to plan the second sprint was to implement non-critical features. Since sprint 1 took longer than expected, the time was very limited.
The decision was made to focus mainly on the functionality already added, to get a better user experience and debug the code, e.g. finding memory leaks and unsuspected behaviors.

6.3.1 Project details

The project details view should, when the application is finished, have a list of mentions, statistics, and a plot. Most of the mention list was implemented during sprint 1. When adding the statistics table it became obvious that the code of ProjectDetailsController started to grow out of proportion and needed to be re-factored.

ProjectDetailsController implements the UITableViewDataSource protocol, when using sections in a table there are protocol methods to tell the sections apart.

```
ProjectDetailsController.m
75 - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
    tableView
76 {
    return 2;
77 }
78 }
79 -(NSInteger)tableView:(UITableView*)tableView
    numberOfRowsInSection:(NSInteger)section
80 {
    return !section ? [viewController.project keywordCount] + 1 : [
        mentions count] + hasMoreMentions;
82 }
```

To solve this and make the code cleaner, a new protocol SectionController was added which defines the same section methods as the UITableViewDataSource protocol. In ProjectDetailsController each section’s controller is added to an array:

```
ProjectDetailsController.m
50 - (void)viewDidLoad
```
By introducing the new protocol it is easy to delegate the method calls to the specific section controller:

**ProjectDetailsController.m**

```objective-c
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
```

### 6.3.2 Result of sprint 2

As seen on the burn down for sprint 2 in figure 8, most of the hours were burnt.
Figure 8: Sprint 2 burn down chart

The tasks in sprint 2 was better estimated than sprint 1. The tasks that remained after the time-box were the error handling and some cosmetic changes.

7 The application

This section describes the application features after two sprints. The four major views shown in figure 9, the screen dumps are taken from the application using the demo account available on Lissly.com

The first screen shot to the left in figure 9 is the start view, which appears immediately when the application is launched. It checks if the user is authenticated, and if not, the modal login view in figure 10 is presented. If the user is authenticated the hierarchical user model is loaded with e.g. User, Accounts, Projects and Keywords.

The second view from the left in 9 is the project list view which presents the projects available in the account. There is also a table view for accounts,
but since there is only one account in the demo-user it is skipped. Unfortunately in this example the current account is named “Accounts”.

Figure 9: Views

The third screen shot to the left in figure 9 is the option view for the project details, where it is possible to select either all or a specific keyword. Further down the screen it is also possible to select source in the same manner. By tapping the date it is possible to change the time period with the date picker seen in figure 10. This functionality is also available in the fourth view.
The fourth view is the project details view. On top, the statistics table is presented and below that, the list of mentions. The mentions can be either text or graphic, the latter is presented with a thumbnail. The external source of each mention can be loaded by tapping the mention. In the end of the mention list there is a button “load more”, to load more mentions if available. The application loads 20 mentions each time.

7.1 Differences between requirements and application

Most of the functionality from the requirements document were implemented. Features that were not implemented:

- The settings view, where it should be possible to select a default project, hide images etc.
- The plots in the the detail view.
- The ability to highlight and comment mentions.

Features that changed during the implementation:
• The project view, which were suppose to have the date period picker directly instead of the modal date picker view.

• The statistics handler was not implemented according to the class diagram, instead it is handled with a hash map.

8 Conclusions and further work

The object-orientation should probably have been more thoroughly designed before starting the implementation. The class diagram was very helpful to get a good overview, but in reality it just scratched the surface of the design. The Cocoa framework is well designed, and this I have learned during the implementation.

It would have been better with shorter sprints, with less tasks, but the length of the first sprint was a necessity in order to deliver something useful to the product owner. Both burn down charts showed early on that there were some obstacles during the implementation. The main obstacles were to learn OS X, Objective-C and the Cocoa framework.

The application works, but before it is used in production it needs to be further tested. Some parts of the backlog must be implemented, like error notifications when the network connection is disrupted.
References


Part I

Requirements specification
Lisslys iPhone applikation

Utseende och navigation
Design som standard iPhone applikationer. Menyer scrollar ner och byter sida genom att dra knapparna i sidled. Uppe till vänster ska en liten bakåtknapp finnas, som alltid ska leda till föregående sida. Överst på skärmens ska en liten version av Lisslys logga synas.

Sidor

Inställningar / Settings
Under settings för logga ut, göm grafer, välj standard projekt, standard period.

Log in
Input för login. Länk till demokonto.

Kontolista
Visar en meny med länkar till konton (projektlistor). Kontosidan visas endast om antal konton > 1.

Projektlista
Visar projekten och dess nyckelordsstatistik för senaste veckan, eller default i tabellform. Ska ha tänd och släck knappar. Projektlistan visas endast om antal projekt > 1.

Projekt
Länkar till 7 dagar, just nu eller anpassat datum. Sen finns länkar till varje nyckelord och länkar till varje källa.

Projektdetaljer
Visar valt datumintervall, två grafer både för källa och nyckelord, båda med möjlighet att endast visa ett nyckelord i listan via en dropdownmenu. Statistikerna för varje nyckelord visas under graferna. Mentions visas här i en lista som har favicon och data. Typerna på mentions är text, bild eller video. Video representeras av en bild från videoen. Varje mention har en knapp “+”, som tar fram fyra knappar. En knapp för “Go to source” som
öppnar originalkällan i den externa webbläsaren, en knapp för att bookmarka inlägget och +/- för positiv/negativ feedback. I de tre senare fallen visas även en input för kommentar. Under mention-listan finns en knapp "more" som laddar in nya mentions under den gamla listan.
Part II

Design Document
1 Use cases

Common alternative flows are not listed and should be handled globally, i.e. none responsive server, user isn’t connected to the Internet etc.

1.1 UC1 Log-in

Purpose: To gain access to the system.

Actors: User
Basic flow:
1. User start the application the log-in form is shown.
2. User inputs log-in credentials.

Alternative Flows:
1a. Wrong credentials
1. User is prompted to input correct credentials.

1.2 UC2 View account list
Purpose: To view available accounts
Actors: User
Pre: User is authenticated

Basic flow:
1. User views accounts

Alternative flows:
1a. There is no account
1. User is notified about the problem.
1b. There is only one account
1. User is forwarded to the project list view.

1.3 UC3 View project list
Purpose: To view available projects
Actors: User
Pre: User is authenticated

Basic flow:
1. User views projects and keydata summary.

Alternative flows:
1a. There is no project
1. User is notified about the problem.
1b. There is only one project
1. User is forwarded to the project view.
1.4 UC4 View project overview
Purpose: To view specific project overview
Actors: User
Pre: User is authenticated
Basic flow:
1. User views project graph with keys and sources.

1.5 UC5 View project details
Purpose: To view specific project details
Actors: User
Pre: User is authenticated. User is viewing project overview (UC4).
Basic flow:
1. User select time range. Options are right now, last 7 days or custom date.
2. User select specific keyword or source.
Alternative flows:
2a Show all
1. User selects show all keywords or all sources.

1.6 UC6 View mention external source
Purpose: To view mention external source
Actors: User
Pre: User is authenticated. User is viewing project details (UC5).
Basic flow:
1. User clicks the plus icon on the specific mention to show available options.
2. User select the option view external source.
1.7 UC7 Mark mention as positive feedback
Purpose: To mark mention as positive feedback.
Actors: User
Pre: User is authenticated. User is viewing project details (UC5).
Basic flow:
1. User clicks the plus icon on the specific mention to show available options.
2. User select the option to mark it as positive feedback.
3. User enters an optional comment in the provided form.

1.8 UC8 Mark mention as negative feedback
Almost identical to UC7.

1.9 UC9 Mark mention as favorite
Almost identical to UC7.

1.10 UC10 View more mentions in project details
Purpose: To view more mentions in the result list.
Actors: User
Pre: User is authenticated. User is viewing project details (UC5).
Basic flow:
1. User scrolls down to the end of the mentions list
2. User press the view more button.
Alternative flows:
2a There are no more mentions in the result
   1. The view more button is not available.

1.11 UC11 Change default view
Purpose: To set the default start view
Actors: User
Pre: User is authenticated. User is viewing settings (UC13).
Basic flow:
1. User selects new default view from a list.
1.12  **UC12 Log-out**
Purpose: To log-out from the system
Actors: User
Pre: User is authenticated. User is viewing settings (UC13).
Basic flow:
1. User selects to log-out
2. User is deauthenticated and redirected to the log-in form.

1.13  **UC13 View settings**
Purpose: To view settings
Actors: User
Pre: User is authenticated.
Basic flow:
1. User presses the button settings.
2. User is forwarded to the settings view.

1.14  **UC14 Access demo account**
Purpose: To gain access to the demo account
Actors: User
Pre: User is unauthenticated. User is viewing log-in screen.
Basic flow:
1. User presses the button “Try the demo”.
2. User is authenticated as the demo user.

1.15  **UC15 Modify detailed view**
Purpose: To modify the current view
Actors: User
Pre: User is authenticated. User is viewing project details (UC5).
Basic flow:
1. User checks or unchecks keywords or sources.
2. The view reloads the graph and the mentions.
2 Class diagram

2.1 Model-View-Controller

The MVC is fairly traditional. All requests enter a controller, the controller uses the models required by the use case, and then gives the necessary models to the appropriate views. The views are rendered and the controller gives the result back to the requester.

2.2 Model

To make the class diagram as clean as possible setters and getters has been omitted, add them as you see fit.
2.3 View
2.4 Controller
Part III

Scrum backlog
<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Connection, parse data.</td>
</tr>
<tr>
<td>Model</td>
<td>User, Account, Keyword, Source, Mention, MentionGraphic, Project, Statistics at least empty dummy classes according to the class diagram in the design document</td>
</tr>
<tr>
<td>Main navigation</td>
<td>Drill down</td>
</tr>
<tr>
<td>Log in view/controller</td>
<td>The login view with the form, the functionality to login and display error message if login failed. The login shall take user/pass and then set a apikey via the preferences.</td>
</tr>
<tr>
<td>Start screen</td>
<td></td>
</tr>
<tr>
<td>Main view</td>
<td>Get a &quot;sliding button&quot; example working. Also add the previous and the setting button to view</td>
</tr>
<tr>
<td>Statistics</td>
<td>Make it possible to query the statistics class/objects with either source or keyword etc, the output is yet uncertain</td>
</tr>
<tr>
<td>Account list view/controller</td>
<td>Display a sliding menu with available accounts, forward to the project view if there is only one account</td>
</tr>
<tr>
<td>Project list view/controller</td>
<td>Display a menu with the projects. Forward to the project view if number of projects is one.</td>
</tr>
<tr>
<td>Datepicker</td>
<td>Modal date picker view.</td>
</tr>
<tr>
<td>Project view/controller</td>
<td>Menus with both keywords and sources. Calendar to select date range, last 7 days, last month or custom date</td>
</tr>
<tr>
<td>Project details view/controller</td>
<td>Show keyword statistics in the view. On top the selected time range</td>
</tr>
<tr>
<td>Mentions text view</td>
<td>Add text mentions to the project details view, with the &quot;add more&quot; button.</td>
</tr>
<tr>
<td>Mention with graphic view</td>
<td>Add the image/video to the mentions.</td>
</tr>
<tr>
<td>Demo account</td>
<td>Add a &quot;Try the demo&quot; button on the log in view, which authenticates the user as a demo user</td>
</tr>
<tr>
<td>Log out</td>
<td>Settings view, with user reset apikey</td>
</tr>
<tr>
<td>Link content</td>
<td>Link the mention to external browser</td>
</tr>
<tr>
<td>Cosmetic changes</td>
<td>Correct the design according to the &quot;meeting&quot;</td>
</tr>
<tr>
<td>Error</td>
<td>Setup error handling, view and such?</td>
</tr>
<tr>
<td>Plots</td>
<td>Add plots to project details</td>
</tr>
<tr>
<td>Settings</td>
<td>The settings view, where you can change some persistent changes such as default view and log out.</td>
</tr>
<tr>
<td>Mention comments view</td>
<td>Add the mention comments to the project detail view</td>
</tr>
<tr>
<td>Mentions submenu</td>
<td>Add a &quot;+&quot; sign to the mentions. Which show a sub-menu, with buttons for like, dislike, bookmark and external source.</td>
</tr>
<tr>
<td>Mentions like view/post</td>
<td>Make it possible to like a mention and also highlight the ones that have been liked.</td>
</tr>
<tr>
<td>Mentions dislike view/post</td>
<td>Make it possible to like a mention and also highlight the ones that have been liked.</td>
</tr>
<tr>
<td>Mentions bookmark view/post</td>
<td>Make it possible to bookmark a mention and also highlight the ones that have been liked.</td>
</tr>
<tr>
<td>Mention comments post</td>
<td>Like, dislike or comment should have the option to add a comment before sending</td>
</tr>
<tr>
<td>Project list keydata</td>
<td>Add the keydata summary to the project list view</td>
</tr>
<tr>
<td>API dummy</td>
<td>Make the API wrapper work with dummy data, for test cases.</td>
</tr>
</tbody>
</table>