Degree project

Developing Reusable iOS Components

Author: Kateryna Gridina
Supervisor: Dr. Rüdiger Lincke
External Supervisor: Dr. Rüdiger Lincke, Softwerk
Date: 2013-09-30
Course Code: 5DV00E, 30 credits
Level: Master
Department of Computer Science
Acknowledgments

I would like to thank and acknowledge all people that help me during master thesis development. Express gratitude to my supervisor Dr. Rüdiger Lincke for giving great supporting and advances. My good friends from Växjö and Kharkiv, family members and boyfriend for support and help during all master thesis development process.
Abstract

Companies often create apps in the same problem domain with similar features. Implementing these features over and over again is inefficient, time consuming, and error prone. This leads to high costs for the developing company and prices which are not competitive on the market.

To address these problems we identify reusable features and implement them in a library which is well tested and allows frequent reuse. Features include navigation patterns, lists, menus, animated components, shared components, gallery and so on.

With the help of these features programmers can easily create various applications with good design and functionality, reducing the time spent on development and allowing competitive prices for related services and products.

Key-words: Components, UIkit, Animation, QuartzCore, XML, UIViewController, XIB, UIView, Foundation.
Content

1 Introduction.........................................................................................................................1
  1.1 Problem and Motivation..............................................................................................1
  1.2 Goals and Criteria.........................................................................................................2
  1.3 Outline..........................................................................................................................4

2 Background.......................................................................................................................4
  2.1 Objective-C ..................................................................................................................5
  2.2 Cocoa Framework .........................................................................................................5
  2.3 UIKit Framework ..........................................................................................................5
  2.4 Foundation Framework ...............................................................................................5
  2.5 Core Graphics Framework .........................................................................................6
  2.6 Core Animation Framework .......................................................................................6
  2.7 Assets Library Framework ........................................................................................6
  2.8 Property List ..............................................................................................................6
  2.9 Git...............................................................................................................................7

3 Features and Requirements ..............................................................................................8
  3.1 Actors ..........................................................................................................................8
  3.2 Features ......................................................................................................................8
  3.3 Requirements ............................................................................................................9
  3.4 Features Description ..................................................................................................11
    3.4.1 Sharing Feature .....................................................................................................11
    3.4.2 Tree Feature ..........................................................................................................12
    3.4.3 Password Manager ...............................................................................................14
    3.4.4 Gesture Locker Feature .......................................................................................15
    3.4.5 Image Picker Feature ..........................................................................................15
    3.4.6 Gallery Feature ......................................................................................................17
    3.4.7 Image Editor Feature ............................................................................................19
    3.4.8 Stack Menu Feature .............................................................................................19
    3.4.9 Stripe Menu Feature ............................................................................................20
    3.4.10 Tabs Feature .........................................................................................................21
    3.4.11 Animated View Controller Feature .....................................................................21

4 Architecture ....................................................................................................................22
  4.1 Basic Programming Concepts and MVC in Cocoa .......................................................22
  4.2 Design Guidelines for MVC Applications ....................................................................23
  4.3 The iOS Architecture .................................................................................................25
    4.3.1 Cocoa Touch Layer ...............................................................................................25
    4.3.2 Media Layer ..........................................................................................................27
    4.3.3 Core Layer ............................................................................................................28
  4.4 Master Thesis Library Architecture ............................................................................30
5 Implementation .................................................................................................................. 31
  5.1 Classes Structure and Main Methods ....................................................................... 31
     5.1.1 Share Module .................................................................................................. 31
     5.1.2 Image Gallery Module .................................................................................... 33
     5.1.3 Tabs Module ..................................................................................................... 34
     5.1.4 Tree Module ...................................................................................................... 35
     5.1.5 Password module .............................................................................................. 35
6 Application example description ..................................................................................... 36
7 Conclusion and Future Work .......................................................................................... 41
   7.1 Conclusions ............................................................................................................ 41
   7.2 Future Work ........................................................................................................... 42
References ............................................................................................................................ 44
List of Tables

Table 3.1: List of features ................................................................. 8
Table 3.2: Functional requirements ................................................... 9
List of Figures

Figure 2.1: Property list file example ................................................................. 7
Figure 3.1: Sharing view example ...................................................................... 11
Figure 3.2: Flow diagram of sharing feature ................................................... 12
Figure 3.3: Tree view example .......................................................................... 13
Figure 3.4: Flow of tree building process ......................................................... 14
Figure 3.5: Password manager view example ................................................. 15
Figure 3.6: Image picker example .................................................................... 16
Figure 3.7: Flow diagram of image picking ...................................................... 17
Figure 3.8: Image gallery view example ............................................................ 18
Figure 3.9: Use-Case diagram for Gallery control ......................................... 18
Figure 3.10: Image picker view example ......................................................... 19
Figure 3.11: Stack menu view example ............................................................ 20
Figure 3.12: Stripe menu view example ........................................................... 20
Figure 3.13: Tab view example ........................................................................ 21
Figure 4.1: MVC Architecture ......................................................................... 22
Figure 4.2: Layers of iOS .................................................................................. 25
Figure 4.3: UIKit class hierarchy ...................................................................... 26
Figure 4.4: Core Animation layers interaction ............................................... 28
Figure 4.5: Cocoa Objective-C Hierarchy for Foundation (main part) .......... 29
Figure 4.6: Base architecture overview ............................................................ 30
Figure 5.1: Pocket status processing method code listing ................................... 31
Figure 5.2: UML Class diagram for Share feature ........................................... 33
Figure 5.3: SWGridView main methods listing ................................................. 33
Figure 5.4: UML class diagram of Gallery module .......................................... 34
Figure 5.5: Listing of SWTabViewControllerDelegate protocol .................. 35
Figure 6.1: Password screen ........................................................................... 36
Figure 6.2: Main application screen ............................................................... 37
Figure 6.3: Notes tree screen .......................................................................... 37
Figure 6.4: My Calendar screen ...................................................................... 38
Figure 6.5: Start Story screen .......................................................................... 38
Figure 6.6: Gallery view .................................................................................. 39
Figure 6.7: Image details screen ...................................................................... 39
Figure 6.8: Share screen .................................................................................. 40
1 Introduction

At the moment, Sweden is a very popular and fast growing tourism center. The tourism industry discovered apps as complement for existing communication channels like printed maps and guides, delivering an additional value to the users. Mobile applications become more popular year after year, more and more companies try to support not only desktop or web application, but also mobile ones. We can see tendency to create not only web application but also mobile web application – even this factor can show us, how fast mobile development grows. There are a lot of reasons, why mobile development is growing so fast. First of all mobile application do not require access to computer – user can use application from smartphone in every place and in every time. Another very important reason is easy to use – mobile application interactive, interesting, and every user can find application for its own taste. For developers mobile development also very interesting because of compatible new field in information technologies, a lot of new languages, technologies and hardware.

More and more tourist companies need new functional and modern apps for marketing needs which defines a well-defined application domain, where apps similar in functionality provide access to region specific data. But, creating apps in the same problem domain with similar features is not always efficient.

1.1 Problem and Motivation

Softwerk is IT company, which develops different kinds of software for Sweden customers as well as for foreign clients. Our company situated in Växjö - small Sweden town. Our company’s fields are Android, Windows Phone and iOS development, also different business and analytic applications. But huge part of development takes exactly mobile development. Softwerk has a lot of customers, starts with municipality and ends with small tourism companies, and we do our best for cover needs all of them in a best way.

Since Softwerk creates a lot of applications related to the tourism field, and many applications implement the same or similar requirements the question arose, if it is possible to implement reusable user interface components in a similar way as it is done for desktop, web, or server applications, in order to save costs and development time, increase testability and customer satisfaction.

This is a difficult question to answer, since finding the right level of abstraction, designing and implementing reusable components is not a trivial task. Thus our aim is to design a library that would include all the necessary components and has architecture, required to be used in different iOS software systems, with a minimum cost for use. We evaluate this in an experimental way, by creating app with bunch of reusable elements. Than we will calculate time for development and amount of code lines that we need, to import reusable elements to the application.
1.2 Goals and Criteria

To solve our problem the following goals need to be reached:

- The first goal is to define a component architecture which supports the composition of apps using reusable components. This goal is reached if we have a clear definition of what a component is, how components are implemented based on the iPhone Operation System Software Development Kit (iOS SDK), how components reuse utility classes from the existing iOS Framework and how to create a new app based on a set of components. The components should be able to share common configuration and reuse existing functionality.
- The second goal is to implement a number of new components. This goal is reached if we have at least 4 different components providing valuable features to an app with minimal implementation and configuration effort. I.e. the code needed for reusing a component is about less than 20% of the code comprising the component.
- The third goal is to maximize the reuse potential of written code. This goal is met if classes, which are similar for at least two components, would be available in an abstract and reusable way in the framework.
- The fourth goal is to make the components and framework usable for developers. This goal is met if the components and framework are well documented and we have an example of application showing how to use all current components.
- The fifth goal is to make standard UI elements more interactive, using animation and transformation. This goal does not connect to reusability directly, but, as we create the whole library, with the set of characteristics, this criteria becomes very important. This goal is met if there will be at least 4 features in the library that will be used/shown in a irregular way, using Core Graphics Framework.

1.3 Constraints

Constraints for this master thesis are to limit the implementation part to iOS and Objective-C. This constraint take place according to the different reasons, described below.

All applications for Apple hardware, as MACs, iPhones, iPads etc. are written in Objective-C language – C based language. There is one common framework – Cocoa, that provides all necessary environment and contains the set of other frameworks. Among them are most important, such as User Interface Kit (UIKit), Foundation, Core Graphics, Core Animation, and Assets Library. All these frameworks form the skeleton of each app. Also every application should have property list file, where the information about configuration, version, and architecture of the system is stored.

Direction of the company - is mostly mobile development. The most popular mobile platforms are Android, iOS and Windows Phone, among them Android takes almost 80% of market. In spite of this facts, we decide to chose iOS platform for development, according to the such reasons.
A large number of free software for Android speaks not only on its favor but as it may sound paradoxical, on its contrary. The fact that the openness of a platform (we are talking about open source) has attracted some serious developers on the one hand, and on the other attracts a lot of mid-level or simply amateurs. In addition there were even cases that releases from the Android Market for free software were infected with Trojans. Although for many applications claimed compatibility with tablets, often it is not implemented fully, that is simply ported from smartphones and as it is not considered that the tablet display has higher resolution.

Programs quality at Apple is more satisfactory. At first, there are more professionals among developers, at second, applications are really optimized to use the tablets on.

Talking about the price of applications in the AppStore, we note that any really high quality and demanded software simply has to be paid, on the one hand, on the other hand in the AppStore applications can often be bought at a discount.

According to these reasons we secied to create framework for iOS platform.

1.4 Outline

Master thesis has such structure: Section 2 presents all necessary information to prepare background for further research, i.e. describe technologies, development context etc. Section 3 provides us all diagrams (such as UML), define requirements and features that we have in Master thesis programming part. Section 4 is about architecture and technical properties of application. Section 5 presents implementation details, such as system structure, features, and integration. In Section 6 we discuss future development and results of work.
2 Background

In this section we provide quick review for main technologies being used during work on development part of master thesis. Small review gives general tips and common information to readers, if they are not familiar with used technologies.

2.1 Existing frameworks overview

Since the developed framework should contain a large number of different modules and components, it was almost impossible to find a similar system with the same functions. Thus, it was decided to split a large framework into subtasks, and try to find frameworks, that can solve same problems.

So, we began with the most complex component of framework - Sharing component. ShareKIT framework was selected for comparison [1]. During the review we revealed that framework is very functional with a lot of useful methods, but it has a lot of functionality, that was not necessary to solve challenges posed by Softwerk. Among them are sharing for Plurk and Linkedin, sharing audio and video, offline sharing. In the software development it is not necessary to overload the system with unusable modules, if there is an opportunity to create your own, simple variant.

Further, it was decided to add a new module - the Gallery. Was created a list of demands to the item, and we began to study existing frameworks. We found a few ones: FSIImageViewer for iOS [2], MWPhotoBrowser for iOS [3], NVGallery for iOS [4]. Each of them had various disadvantages: too simple image management system, the inability to dynamically add or delete an image (NVGallery), lack of Editor (FSImageViewer), poorly adaptable view of galleries, a great similarity with the standard iOS Gallery (MWPhotoBrowser). In order to correct the shortcomings of these systems, we decided to create our own gallery module, which would fully satisfy the requirements.

Since the system was planned with large number of UI elements, we found a similar framework HTFramework [5], which solved similar problems. It has different interesting and useful UI elements, but they are not directed to solve particular task, they were developed for very wide range of tasks. Also the disadvantages of this system are weak scalability and fragmentation of using elements. But the biggest drawback was that it does not have the required components for the tourism-based applications development. It is easy to see, that a large number of companies, especially ones that have narrow specialization in applications development, prefer to write their own frameworks. Self-constructed framework has the following advantages:

• contains precisely specific modules that are required for the specific tasks;
• easy adapts to the needs of applications;
• easy to scale and maintain;
• consumes application development time;
• high-quality documentation.
Thus we can say that, in spite of the fact that there are many different frameworks, they do not fully cover the needs of a specific type of applications, have different structure of code and of the systems, most of them are redundant.

2.2 Objective-C
Objective-C is the primary programming language you use when writing software for OS X and iOS. It is a superset of the C programming language and provides object-oriented capabilities and a dynamic runtime. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods. It also adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime [6][7].

2.3 Cocoa Framework
Cocoa framework is a branch of different object-oriented frameworks for development in iOS and OS X operation systems. It presents different runtime environment to make developers life easier. In iOS exists only one application environment – Cocoa, that is also one of two used environments in OS X. In OS X we can also use Carbon as alternative, but it is not native framework, that has more difficult interface – procedural, so it does not support code bases that are already existed. Well-known applications in OS X and iOS such as Notes, iMap, AppStore and a lot of others based on Cocoa framework. As development platform is used Xcode - in this environment we can create as iOS as OS X based applications. Good environment in combination with well designed framework – Cocoa make possible to create functional and well-featured applications [8].

2.4 UIKit Framework
One of most useful frameworks is UIKit – it provides all necessary UI elements and constructions to develop high-quality fast-rendering UI, that can be easily managed by provided bunch of functions and structures. This framework is a part of iOS Developer Library and can be used only during development for mobile devices [9].

2.5 Foundation Framework
The Foundation framework have some similarity with UIKit framework in definition of general classes. But logically these two frameworks are quite different – in time, when UIKit can help us to create all user interface stuff, the Foundation framework reaches all non-user-interface targets in your app.

Foundation framework present how basically object behaves in the system, how notification, localization, memory and other important features work.
Utilities, object wrappers are provided in Foundation framework – it helps to work with numeric values, collections, strings, threads, networking, files, time, date presentation and many other [10].

### 2.6 Core Graphics Framework

Core Graphics framework – is vector-base drawing framework, that is used such in iOS as in OS X development. It is a powerful engine, that provide very confident API for working with 2D objects, and provide such features as drawing by provided paths and coordinates, dynamic gradient, that display have any curve, simple rendering of different images and another graphical objects. Also Core Graphics contains a lot of functions for comfort work with PDF documents, and allows developers to make that types of transformation, that they want.

Core Graphics is very easy-to-use framework, as it created on base of Objective-C abstractions. Basically this framework is based on C language, but in case of abstractions you can see the result of your work, that simplify development process [11][12].

### 2.7 Core Animation Framework

Core Animation is a very powerful framework on iOS an OS X, that developer can use when faced with rendering or animation problems with different visual elements – images, controls, fields. Developer do not need anymore use difficult external libraries and structures – almost all work is done inside Core Animation – we just need to configure it in a right way and set bunch of parameters (for example animation interval, start and end time of animation, curve trajectory points etc). All work is done in a very productive way, with high rendering speed and frame rates, thus CPU is not overloaded and do not reflects on application work speed. [13].

### 2.8 Assets Library Framework

Assets Library Framework appears in iOS 4.0. With help of this framework you can retrieve videos and photos from devise, change them immediately, create new ones and save them to the user library, so manage multimedia files in an appropriate way. There exists one restriction – user need to approve access to his library of external application. After user allows access to photo library from yours application, interaction begins immediately [14].

### 2.9 Property List

Property list is a set of XML-based components, which represent property options for application. Property list normally contains serialized objects, that are, in general, settings and preferences for the application.
Figure 2.1 shows a simple example of Plist file. This structure contains information about project, such as Bundle identifier (used for creating builds for testing, connected to developer profile), Bundle Package Type – application, Bundle Version (version of the application), Device Capability (processor architecture) and so on.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleExecutable</key>
    <string>${EXECUTABLE_NAME}</string>
    <key>CFBundleIdentifier</key>
    <string>com.softwerk.master</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>1.0</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>CFBundleVersion</key>
    <string>1.0</string>
    <key>LSRequiresIPhoneOS</key>
    <true/>
    <key>UIRequiredDeviceCapabilities</key>
    <array>
      <string>armv7</string>
    </array>
    <key>UISupportedInterfaceOrientations</key>
    <array>
      <string>UIInterfaceOrientationPortrait</string>
      <string>UIInterfaceOrientationLandscapeLeft</string>
      <string>UIInterfaceOrientationLandscapeRight</string>
    </array>
  </dict>
</plist>
```

Figure 2.1: Property list file example

### 2.10 Git

Git is a version control system, which is well productive, and allows developers make local repositories working with them like with global ones. The main benefit of this system is that you can easily work with project, without disturbing remote version on server, and only after refactoring and tests, code moves on main repository [17][18].
3 Features and Requirements

This section will provide us list of entities that was discussed before, such as users, requirements and features. Entities for requirements were be presented by using UML use-case diagram.

3.1 Actors

We have one type of user in the system - developer. Developer – kind of users, that can modify, delete and create new components and have full access for the system settings.

3.2 Features

To define the architecture, i.e. to reach the first goal, we have identified the major components that need to be in the system. To do this, a list of components has been defined. The next step was the construction of the interaction between the components and program that uses them, parameters, properties, and database. Then, after there was an architecture determined, it was decided to move to implementation. Thus, all the planned components have been implemented, taking into account the architectural features and inter-component communications.

In Table 3.1 we can see structured overview of features that will be implemented in our application. Provided features helps us reach goals described in Introduction section. As we used agile technology during development, this list was been created, strictly speaking, before development process itself. That is why some of features/tasks were added during work on Master thesis.

Table 3.1: List of features

<table>
<thead>
<tr>
<th>Feature 1</th>
<th>Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Sharing feature have very wide list of functions. It allows developers to share elements in one convenient and structured way to Facebook, Twitter, Instapaper, Messages and Mail. Types of sharing elements are text, images and urls.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature 2</th>
<th>Tree Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The iOS Tree component is a UITableViewCell replacement that lets your UITableView works as a tree. It must supply UITableView’s data source. Feature should contain node data that can hold the actual data we want to display in the tree. Developers can change numbers of elements and views for them.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature 3</th>
<th>Password manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Feature that allows user to set, change or enter a simple 4-digit passcode. There developer can use sound effects; change views and number of pass digits.</td>
</tr>
</tbody>
</table>

| Feature 4 | Gesture locker |
Description: Screen gesture unlocker is the feature for unlocking screen like in Android devices. Developer can provide some set of appropriate points, and unlock screen if these points fits. Also developer can provide number of rows and cols, and view of the points on the screen.

Feature 5: Image picker
Description: Custom picker for taking photo. Photo can be stored in custom or on device gallery. Also there is an action sheet where we can get such options as delete, edit and retake photo.

Feature 6: Gallery
Description: Custom image gallery. In a gallery developer can make such view for cells as he wish: change size, position, make highlighted on sections, change background, numbers of cells and rows. All images are taken from SQLite database. It is possible to use custom gallery with device gallery at the same time by changing properties.

Feature 7: Image editor
Description: Custom image editor. Very similar to device image editor that appears after image picking. The difference is that it can be used in a custom gallery for the same purpose, while predefined editor can be used only together with device image gallery.

Feature 8: Stack menu
Description: Feature for animating menu. In this case menu should appear from the corner of the screen. Developer can change amount of elements, behave touches, change views and positions of element

Feature 9: Stripe menu
Description: Animates menu from the some border of screen. Developer can change position and view of elements.

Feature 10: Tabs menu
Description: Feature for creating custom tabs like in web browsers. It has such features: add, delete and open tabs. Developers can change container for the tab and views for them.

Feature 11: Animated viewContoller
Description: This feature is used to make application look more adjective and non usual. Controller appears not in a normal way, the animation of screen rotation shows up.

3.3 Requirements
Functional requirements basically describe a function of system or its subcomponents. Requirements can be presented as functions performed by screens, outlines of work-flows performed by the system, and other requirements the system must meet. [19]. Table 3.2 shows functional requirement of the all components

Table 3.2: Functional requirements
<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Criteria</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ARC support</td>
<td>The system should support Automatic reference count [20]</td>
</tr>
<tr>
<td>Common structure</td>
<td>The system should have common structure, so developer can easily use all components in a same way.</td>
</tr>
<tr>
<td>Common resources</td>
<td>Common resources bundle should be in a library, so all images, documents and other resources can be in the same place.</td>
</tr>
<tr>
<td>iPhone, iPad support</td>
<td>Modules can be used such in iPhone as in iPad projects without any logic transformation.</td>
</tr>
<tr>
<td>iOS 5 and latter support</td>
<td>System should support iOS5, and later operation systems.</td>
</tr>
<tr>
<td>Multi-language support</td>
<td>System should support multi-language in easy way, by using localizations inside whole system.</td>
</tr>
</tbody>
</table>

Let us look at each criterion in turn.

**ARC support.** In most systems that are written in iOS, the main problem is a memory management (memory leaks, zombies, allocations). These problems lead to crashes, brakings and instability. To avoid such errors and to simplify the work with memory, it was decided to use ARC.

**Common structure.** For any frameworks it is very necessary to have fully defined structure, and general view. For this purpose different patterns are used very often (in our case, MVC). If the system is divided into sub-components, it is easier to adapt and to scale each component.

**Common resources.** In order to create universal frameworks to be universal and widely used, regardless of the color scheme or language, all resources should be stored in one location. Thus, we can easily change all resources, which we need inside our system, without “immersion” into the system. Images must be stored in Resources bundle, and text strings in Localizable.strings files (files for storing textual strings, that are used inside the application).

**iPhone, iPad support.** To make framework universal for all iOS-based mobile devices, we should support iPad and iPhone, as they have different characteristics and functional requirements.

**iOS 5 and latter support.** The newest version of iOS is iOS7, but when we started with framework development, the latest one was iOS6. It is normal to support two latest OS (unspoken rule in “Apple” world), so we decided to support version of iOS6 too.
**Multi-language support.** Since framework must be created for travel-based apps, multy-language support becomes a very important non-functional requirement.

### 3.4 Features Description

In this section we will describe all features that we can see in Table 3.1. I start from the requirements to every feature, and continue with examples and descriptions.

#### 3.4.1 Sharing Feature

Very often during the travel people try to share their emotions about the trip. Travelers are increasingly using mobile devices to share their experiences. That is why the sharing function is vital in any application related to the tourism. In order to satisfy the wishes of the majority of the users, customize sharing element was created. Sharing controller allows anyone to implement sharing into their projects easily. iOS 6 will use the new activity view controller while iOS 5 will use action sheets. Except usual sharing libraries, as Facebook, Twitter, Messenger and Mail another popular frameworks have been used, such as Readability, Pocket, and Instapaper. They have been added as read-later services to share with articles.

As a user, who uses this feature in the ready application, we need to choose element for sharing, if there are images, and after that press some control, which responds for the sharing action. The example of view you can see on Figure 3.1.

![Sharing view example](image)

**Figure 3.1:** Sharing view example

Then user can see action sheet with all possible options. The flow of the sharing process described in the flow diagram on Figure 3.2. When user press “Share” button we have different scenarios: user can be registered or not, registration can be successful or failed, service can respond or can be busy. All this scenarios are shown on Figure 3.2.
3.4.2 Tree Feature

The iOS Tree component is a UITableViewCell replacement that lets your table works as a tree. The tree can contain different numbers of sub-trees, so we get effect of file system inside the application. This feature is very useful in case we need to structure information on categories and subcategories. For example, in case of travel application, user can have the list of all counties he visit as a main nodes. Inside each country node can be short informations sub nodes, such as telephone numbers of hostels, list of beautiful places, names of the best restaurants. The example of view you can see below at Figure 3.3.
The tree on the Figure 3.3 presents tree structure of notes in application. Every note can have subnotes, and direction of plain shows whether the subtree of note is shown or not. “Plus” buttons allows create new subnote.

Figure 3.3: Tree view example

This control must supply UITableView’s data source. Feature should contain a node data that can hold the actual data we want to display in the tree. Developers can change numbers of elements and views for them. The flow of tree building described in Figure 3.5.
3.4.3 Password Manager

Feature that allows user to set, change or enter a simple 4-digit pass code. In case of flexibility it is possible to develop more complex pass code. This feature is very useful in case if you want to limit access to separate part of the application, or to the whole application. It can be useful if user want to hide some part of content. The example of user interface is on Figure 3.5.

This element has possibility to interact with server, such as use only application settings. It depends on application implementation - if it has server side, and if developer connects this control to the server. There developer can use sound effects, change views and number of pass digits.
3.4.4 Gesture Locker Feature

Screen gesture unlocker is the feature for unlocking screen like in Android devices. This feature has big potential, as it reacts on finger moving. In the future it can be used for route creation or in quizzes in any case. But for now the main function of it is to get some set of points, that user touch, and react on that user touches. This feature has long been used in a such mobile OS as Android. Basically on Android devices this is used for screen unlock, like gesture commands for open some apps or execute frequently used tasks.

Developer can provide some set of appropriate points, and execute necessary function if user touches points that fit. Also developer can provide number of rows and cols, and view of the points on the screen.

3.4.5 Image Picker Feature

As we know, in iPhone devices there is native image picker [21]. It is quite easy to use it inside the application. The only problem is that native image picker work only together with native gallery. So if it is necessary to use custom gallery, developers need to create also
custom image picker. This control was implemented in such way, so you can store photos both in custom and native galleries. Developer can connect them separately, or can use both of them in a same time. If application uses both galleries, user has an option to choose where to store a photo. There is an action sheet inside the control, where user has such options as delete, edit and take new photo (Figure 3.6).

![Image picker example](imagePickerExample.png)

**Figure 3.6: Image picker example**

If user chooses edit function, he has two options: using native gallery and native editor, or custom gallery and custom editor. If user wants to delete image, he can delete it only from image place inside the picker, but not from the gallery. The flow diagram below on Figure 3.7 can show the process of image creation and management.
3.4.6 Gallery Feature

Gallery is one of the most useful elements in different applications, that related to tourism and traveling. Although there is a standard control for this purpose in iOS 6, custom gallery has more powerful interface and contains a lot of useful functions. Also Gallery controls allow developers to support iOS 5 applications. The cell control is very flexible; it allows change size, position, highlight, color, shape, grid etc. There is a great feature for changing
animation, touches behavior, deleting and editing. The example of the gallery is displayed at Figure 3.8. As there is custom image picker, it can use this gallery for their purpose.

Figure 3.8: Image gallery view example

As for the data source, gallery can take images from SQLite database. It is possible to use custom gallery with device gallery in the same time by changing properties. There are such behavior as refresh, reload, delete and add item. Use-Case for user actions you can see below at Figure 3.9.

Figure 3.9: Use-Case diagram for Gallery control
3.4.7 Image Editor Feature
Custom image editor is an independent to native image editor inside native image picker. The image editor is an independent module, so it can be used for editing images not only in standard, but also in custom galleries. This modul has different additional features: it is possible to change color theme, style, size and other parameters of view. An example of image picker is on Figure 3.10. There are zoom-in, zoom-out and cut functions. Changed image can be used on, but original one will stay without changes.

![Image picker view example](image.png)

**Figure 3.10:** Image picker view example

3.4.8 Stack Menu Feature
Stack menu feature is used for animating menu view. This animation shows menu in a non-regular way - menu buttons appear from the one point of the screen. Developer can decide from which point it appears, how much elements has, in such way should animate in, what text need to be there and how it should look like. All these properties are flexible. The example of menu looking is on Figure 3.11.
Figure 3.11: Stack menu view example

3.4.9 **Stripe Menu Feature**
As previous one it is animation feature, it Animates menu from the border of screen. Developer can change position and view of elements. Example of the menu you can see at Figure 3.12. On this figure the menu displayed near the left border of the screen and contains 5 buttons inside.

Figure 3.12: Stripe menu view example
3.4.10 Tabs Feature
Feature for creating custom tabs is the same as in web browsers. It can be very useful for creating notepads, information lists and make usual applications more interactive. Each tab contains own controller inside, so it is easy to use this feature for any purpose. It has such features: add, delete and open tabs. Developers can change container for the tab, views for them, animation and styles. The example of tabs view is on Figure 3.13. On the top of the view (Figure 3.13) we can see green tabs with “x” button in the right corner – close button. According to the pressed tab, the different controller will be loaded. Active tab marked with dark green color.

![Tab view example](image)

**Figure 3.13: Tab view example**

3.4.11 Animated View Controller Feature
This feature is used to make application look more attractive and non-usual, it is only visual element. With the help of this animation controller appears not in a normal way, it appears with rotation screen animation.
4 Architecture

In this section we will discuss how MVC pattern works inside Cocoa for better understanding how our own framework must be designed. Also we will overview iOS architecture and layers to mention which classes from which layers can be used, and why they are important inside of our system. Design guidelines of MVC help us compare system design with current requirements from guidelines.

4.1 Basic Programing Concepts and MVC in Cocoa

The Model-View-Controller design pattern (MVC) is widely used – this is well-known pattern for global architecture that connected to the bunch of high-level patterns. Whole structure of project divided according to the roles in different parts of application. MVC pattern allows to create object oriented programs with better reusability, more adaptive, extensible. We can separate logic for different layers, and change this parts without direct influence on another layers. MVC pattern is a merge of different elemental patterns. MVC is used in different architectures of Cocoa Framework, among them are binding and document architecture.

The MVC pattern is represented in 3 different objects: model, view and controller objects (Figure 4.1). Pattern considers different communications between these entities and their roles in it [22].

![MVC Architecture](image)

**Figure 4.1:** MVC Architecture

There are a lot of different technologies and mechanisms in Cocoa where Model-View-Controller is a base pattern, such as:
• **Document architecture.** In this architecture the NSDocumentController plays Controller role, NSWindowController is defined as View and NSDocument is a combination of Controller and View.

• **Bindings.** MVC is central to the bindings technology of Cocoa.

• **Application scriptability.** Responsible for application state and usually requests app behavior. Represented as a scripting command.

• **Core Data.** The Core Data framework is responsible for objects persistence, their management, monitors saving state.

• **Undo.** Assessor methods play the role of Model in Undo. They implemented undo or redo, by changing data in TextView, for example. In this case TextView play ole of View and Controller.

4.2 **Design Guidelines for MVC Applications**

In general, a MVC programming pattern has common description and structure for any program language. But the development of a variety of architectures, software components, and programming languages shows that the pattern need to be improved and adapted to cover needs of a particular approach. These features have appeared in MVC for iOS also. In order to properly build application architecture, whether it is a desktop application, mobile, or, as in our case, framework, we need to follow the guidelines that are given in this section. Let us take a look at the most important ones. To create well-based and good-designed Model-View-Controller application we need to follow such rules:

• As we know, NSObject is a base class, so developers, can inherit from it all another classes for Controller part of MVC model, but it is not really necessary to do this. In Cocoa library exists a lot of different ready-made classes, that can cover our needs - such as NSArray, NSNotification, NSUserDefaults, NSTree and many others. Developer also can create its own class, that will be inherited form NSController subclasses. In another case, if application is small and has no a lot of functionality and content, can be more comfortable to write glue code for implementing Controller part by using instance from core class - NSObject. It is no difficult than to create mediate controller based on NSController sense - in this task different technologies from Cocoa can help us. For example we can use KVC, KVO and some protocols.

Because our framework has no direct inheritance from NSObject we can temporarily skip this step, but it is necessary to remember for future scalability and expansion.

• Better not to combine different roles in MVC, keep them separated, even if combining is not strictly prohibited. Separation between roles will helps us to improve extensibility, reusability and maintainability of project, where we use MVC. If you really need to merge different roles in one application, than you need to define role, which will be predominant, and than use extensions and categories to extend that class.

We have a clear division of roles in our framework. It includes classes belonging to the View as well as classes belonging to the Controller, they are the largest part of the production system. The Model part inside frameworks is a very virtual part, and almost never enforced, because it is common to use a model of the system, which uses framework. We tried to avoid
roles of the mixed type, so we can say, that we have fulfilled this recommendation in our system.

- The base idea of object-oriented development, as well as MVC - is to make as much as possible reusable classes. In ideal way, the most reusable components in MVC model should be View and Model classes. In case of specific functionality inside Controller classes, sometimes it is difficult to make them at least 50 % reusable. But well-made, full production applications should be designed in a way of high reusability.

One of the objectives of the developed framework was creating a number of reusable components. Also, because the architecture was thoroughly premeditated before development, we could avoid duplication within the code, as well as reuse most of the ready-made components of the program.

- Very important follow such interaction strategy inside application View - Controller, Controller - Model and vice versa. Developers, of course, can request changes in Model objects directly from View objects, but it is a bad-style programming. There are such reasons not to do so:
  o Loosing of advantages from median (NSController) interaction. By requesting changes directly from Model, you miss some important parts than can be done inside Controller - such as discard changes, send changes on server, perform partial requests, make operations with intermediate objects, perform type conversion and many others.
  o To get changes from Model you can use binding technology, in another case you need to inherits from View class and add observation logic there to catch changes from Model.
  o A view class should not depend on a mediating controller class.
  o A view class should not be connected to model classes. If you can not avoid that (can happens in custom classes) - minimize connection and look at first point in this list.
  o Model class should be depended only from another Model class.
  o Controller classes should be always independent from Model classes. Use any dependency only in a very critical situation.
  o Controller classes should also be independent from View classes. Use this dependency only in a very critical situations.
  o A synchronize controller class depends on roles inside MVC.

The division of responsibility for the Controller and View - is a basic rule when start to create framework&. View should not be used for other purposes except of displaying content and interacting with the Controller, because when the system is used, a developer should easily adapt view for the needs of the program. Very important to follow this rule while framework is created

- Try to make less dependency between different classes in application. High dependency in class structure leads to low code reusability. There are two more recommendations about MVC model construction: if programming problem is solved with help of MVC architecture - better use it as a core for project. Than,
Main rule is not to go in opposite direction from approach, that Cocoa offer you.

As we have set of different components inside framework, there dependencies are very weak.

Using this guideline we followed through the framework, and determine, if the architecture matches main criteria for well-designed MVC structure.

4.3 The iOS Architecture

iOS (like others operating systems) is a middle man between hardware and software. Which is good, because when you develop an application, you can rely on the platform, regardless of the concrete device. iOS consists of a few layers itself (see Figure 4.3), as well. Top layer is most clear and easy to use. But if you wanted to make something really custom, you need to go deeper. In our framework we used components from all layers, and these components we will discuss in this chapter in next subparagraphs.

![Figure 4.2: Layers of iOS](source)

### 4.3.1 Cocoa Touch Layer

Cocoa Touch is iOS specific layer. iOS developers must develop their products on top of it. It provides interface for basic services and tools of the platform. The most widely used framework is UIKit, without this one any application can't work. Because everything user see and interact with on the screen is elements of this framework. Inside Master thesis library we widely used UIKit Framework.

In UIKit framework all control classes are inherited from UIControl, where all basic functions and directives are implemented. In this framework are about 35 different visual elements, and all common functions are moved to UIControl [23].

The UIKit framework provides the classes needed to construct and manage an application’s user interface for iOS. It provides functions for work with application objects, properties for their presentation, interfaces and events, that we used during interaction with these objects, gestures and many other. Figure 4.4 illustrates the classes in this framework.

With red dots we mark all classes that have been used in our own framework. The most important and frequently used class is UIResponder. From this class all necessary UI elements inherit, so we mark it with red line. Dotted classes – are skeleton of every iOS application, as they form appearance of app, and provide interaction between all elements.
inside application. Most of the classes catch different events as different touches, swipes, moves, value changes, so we must not think about delegates.

Figure 4.3: UIKit class hierarchy
4.3.2 Media Layer

All the Media layer technologies (like audio, video and so on) are designed to improve user experience. Provided bunch of technologies allow developers to make good software for mobile devices [25]. You can find a lot of frameworks there. These are ones used in our Master thesis library:

- Quartz Core Framework
- Core Image Framework
- Assets Library Framework

Sometime developer needs to work with photos and videos in his own application in a same way, as it is in Photo application on device. For this purpose we can easily use Assets Library framework, that appears in iOS 4.0. With help of this framework you can retrieve videos and photos from devise, change them immediately, create new ones and save them to the user library, so manage multimedia files in an appropriate way. There exists one restriction – user need to approve access to his library of external application.

There is main framework, that present a lot of well-used features (for example filters) for work with images and video data – Core Image framework. It was introduced in iOS 5. This filters help to operate with images – like face detection, correction, red eye detection, and this filters are built in and easy maintainable. The advantage of using these filters is that they operate in a nondestructive manner so that your original images are never changed directly. In addition, Core Image takes advantage of the available CPU and GPU processing power to ensure that operations are fast and efficient.

Core Animation is a very powerful framework on iOS an OS X, that developer can use when faced with rendering or animation problems with different visual elements – images, controls, fields. Developer do not need anymore use difficult external libraries and structures – almost all work is done inside Core Animation – we just need to configure it in a right way and set bunch of parameters (for example animation interval, start and end time of animation, curve trajectory points etc). Another part – like drawing, rendering and other difficult features Core Animation does by itself. All work is done in a very productive way, with high rendering speed and frame rates, thus CPU is not overloaded and do not reflects on application work speed. The base of all animations and graphic effects inside master thesis framework, we developed using this library.

During development we usually use Core Animation, this usage is not very straightforward – Core Animation is behind all visual elements, methods, that we use for user interaction are based on Core Animation framework. On Figure 4.5 there is Core Animation layers interaction. AppKit and UIKit is upper layer for Core Animation. This framework take one of the most important places in Cocoa Touch and highly integrated to Cocoa workflow. Basically, main function of Core Animation, of course – is to make work with animation and user interface interaction more fine-gained and capable. CoreGraphics and OpenGL are the tools to work with graphics and are the parts of layer above hardware (physical user events, such as touches, moving, taps) that processing events coming from device.

Thus, in any case, if we need to create beautiful and interactive feature we need to use Media layer. Work with all the images, animations, shadows and graphics in general carried out through the use of Media layer features.
4.3.3 Core Layer

The Core Service layer – fundamental layer in the system, it contains all basic services and utilities for system usage. We do not need to use them directly – a lot of system functionality based on that services.

As in Media layer, Core layer has a lot of Frameworks, technologies and libraries, but let’s describe only that we use in Master thesis Library:

- ARC
- SQLite
- Core Foundation Framework
- Foundation Framework

Introduced in iOS 5, Automatic Reference Counting (ARC) one of the most powerful low-level features, created to simplify lifecycle and memory usage managing in Objective C. ARC works almost the same way as garbage collector in Java – developers do not need to remember when release, retain or deallocate object, while ARC knows lifetime of any object and automatically handles method calls.

The SQLite library lets you embed a lightweight SQL database into your application without running a separate remote database server process. From your application, you can create local database files and manage the tables and records in those files. The library is designed for general-purpose use but is still optimized to provide fast access to database records.

Foundation framework based on main paradigms that present functionality not in Objective C language. That functionality works this way to provide more useful primitive object classes, also the Foundation framework is represented as base layer in Objective C class structure. Foundation framework solves bunch of problems, such as:

- Present effective memory management policy to make development easier and more productive.
- Utility classes provided.
- Probability increase in case of providing independence level.
- Unicode characters, persistence and distribution of objects supporting.

The Foundation class hierarchy is rooted in the Foundation framework’s NSObject class (see Figure 4.6). Figure 4.6 is only one part of classes that are more frequently used.

Figure 4.5: Cocoa Objective-C Hierarchy for Foundation (main part)
4.4 Master Thesis Library Architecture

Using all above information, and huge baggage of iOS architecture structure, we can create the schema for my own Library. Figure 4.7 displays architecture and design of whole system according to the discussed requirements.

![Diagram of iOS architecture](image)

**Figure 4.6:** Base architecture overview

The system is divided in 6 main parts.
1. iOS layers - contains Cocoa Touch, Media, Core Services and Core OS layers.
3. Components – independent modules witch use Foundation frameworks. These components will be plagued into the applications and will have public functions.
4. Resources bundle - contain all external resources, such as images, documents etc.
5. Localizable strings - strings used in the system, to allow developers easily create multi-language projects.
6. Application – final application witch shows how all elements should work together.

The parts 1, 2, 3, 4, 5 were discussed before in Sections 1-4, and part 6 will be discussed at Section 6 - Application example description.
5  Implementation

This section explains the main implementation of the used algorithms. The structure of classes and main methods, features realizations and algorithms will be described.

5.1  Classes Structure and Main Methods.

Library contains 11 features, and almost each of them are presented as independent module, so it makes sense to show class structure for only most important modules. Let’s start from class structure of Share module.

5.1.1  Share Module

Share module is one of the most difficult features in the library, and there are lots of classes and methods:

- **SWMessageService** - service responsible for Message sending feature. It implements methods of MFMessageComposeViewControllerDelegate protocol
- **SWReadLaterService** - service used by Instapaper, Redability and Pocket services. Used to handle events such as connection to the server, the latest reading of messages, process logiand exit
- **SWPocketService** - class for sharing via Pocket. It is also inherits from SWReadLaterService and implement login/logout, share and status processing methods. The example of status processing listing is on Figure 5.1

![Code snippet](image)

```swift
// Pocket status processing method code listing
```

- **SWShareThis** - contains all main methods, the main class of the Share module.
• SWTwitterService - used for share via Twitter. Implement SWShareService protocol
• SWPocketService - used for share via Pocket service. Inherits from SWReadLaterService, and have methods that process login/logout, errors and put information to key-chain.
• SWRedabilityService - class that response for connection to the Redability server, share there, get messages, login and logout. Also inherits from SWReadLaterService
• SWInstapaperService - responsible for sharing via Instapaper. It used Facebook credentials, and can work through this service. But also have own login/logout methods, share and status processing methods. Inherits from SWReadLaterService
• SWEmailService - class that send e-mails using MailComposer. It implemented MFMailComposeViewControllerDelegate protocol methods.
• SWFacebookService - class that share information on Facebook. There are methods for sharing, session creations, error and status processing, closing session.

Class UML diagram for Sharing feature presented on Figure 5.2
The different Frameworks are used inside this module of Sharing, such as Social framework, Facebook SDK framework and Twitter framework.

### 5.1.2 Image Gallery Module

Image gallery is responsible for representing collection of images from some data set. There are a lot of supporting features there - edit, delete, add, shuffle, review and so on. Gallery has such main classes:

- **SWGridView** - class control all functions inside grid view of gallery, and responsible for all core actions as add, edit, delete. Also there are methods that fetch data from data source and present it in the cells. The properties inside it can allow us to change row and cells numbers, colors, animations and so on. Also it handles reloading data, changing bounds, set indexes process and define SWGridViewDataSource protocol. The listing of main methods is on Figure 5.3.

- **SWGridViewCell** - this class contain function and properties for representation and working with gallery cell. The object of SWGridViewCell class catch events, such as touch, swipe and tap, and handle this event.
• **SWGridViewData** - class for work with image data. This class can reside how much rows and columns need to be in gallery, determine number of items and send each of them unique identifier.

• **SWGridViewUpdateInfo** - class contain all necessary functions for refreshing images inside the gallery.

• **SWGridViewUpdateItem** - class contain comparator, that determine if cell was been changed or not. In case of it was been changed this class is responsible for replacing old item with new one.

• **SWImageGalleryCellChooser** - class behave choose action.

The UML class diagram is on Figure 5.4.

![UML class diagram of Gallery module](image)

**Figure 5.4:** UML class diagram of Gallery module

### 5.1.3 Tabs Module

Tabs module is very important part of the library, as it allows developers create different applications with tab navigation in a very easy way. There are such classes:
• SWTabViewController - contains methods for adding removing and selecting of view controllers and method for setting index to view controller. The SWTabViewControllerDelegate protocol is defined inside this class. Inside this protocol the most important actions behave. The list of functions is on Figure 5.5.

![Figure 5.5: Listing of SWTabViewControllerDelegate protocol](image)

- SWTabBar - class that holds all tab element and responsible for tab grouping. Also it behave touches, and decide to which tab need to send message about touch.
- SWTab - the object of tab itself. Contain some properties and methods responsible for the view and touch detection.
- SWTabsAppearance - class contain set of properties, that are used for setup view with tabs.

### 5.1.4 Tree Module
Tree module allows developers create hierarchy in the table view - it should look as file system hierarchy. The classes of this modules are:

- CWTreeViewController - this controller represent table view with non-ordinary looking. It implement table delegate methods, and get all data from the file. Inside this controller all recursive calls performs and tree view creates.
- CWTreeViewNode - object of tree node. It response for looking of the cell, also behave such actions as touches and swipes.

### 5.1.5 Password module
This module is responsible for creating password managers inside the application. The main classes are PasswordViewController that contain touches behavior and main logic, and PasswordViewControllerHelper that help to detect touched number and the cell id.
6 Application example description

In order to demonstrate the possibilities of the created library we create demonstration application. As library was created with the tourism domain in mind, we decided to create small application with features from the library. Creating this app with the help of the library took just about two days with all modifications, image searching and database configurations.

The start screen is the Password screen, where user can insert password to start using app. Sometimes it is necessary if you want to close access to your personal data, such as images and notes. The first screen of app you can see on Figure 6.1.

![Password screen](image)

**Figure 6.1:** Password screen

If user inputs right password, the first screen of actually app opened. In another case user gets an alert with message “Wrong password. Please, try again”. We do not process “Forgot password” feature because had no server interaction (all data is hard coded in database).

In Figure 6.2, we can see start button, and when we press on it, the menu appears. For showing menu in such way we use Stack menu feature. In the menu we have 4 options: Notes Tree, My Calendar, Gallery, Start Story.
When user presses on Notes tree button, new view appear like rotating cube – this feature is called “Animated View Controller Feature”. On this screen the list with user notes should appear. User can add some note and add additional marks. Everything displayed as tree, and it is very comfortable to read information. In this screen we use Tree feature. User can add nodes and sub nodes by tapping on “+” button and delete tree nodes by swiping on the cell. The controller is on Figure 6.3.
By pressing My Calendar the new view with calendar appears. When we tap on some day, the stories according to this date appear. It shows us how easy interact with elements inside app, as calendar use information from gallery. The example of this screen we can see on Figure 6.4.

![My Calendar screen](image)

**Figure 6.4:** My Calendar screen

Start Story button lead us to the screen with story creation (Figure 6.5). We can pick image and describe it, so create the story. In this screen we use Image picker feature. When user press done, the story goes to the database.

![Start Story screen](image)

**Figure 6.5:** Start Story screen
Main part of the application is Gallery. In this case we can see gallery of stories (Figure 6.6).

Figure 6.6: Gallery view

If user has not created any story, gallery will be empty; in another case will have all images inside. For creating this screen Gallery feature was used.

If we tap on some image, the image in a big size with description will appear. Here user can zoom and move image (Figure 6.7). In this screen we use Image editor feature.

Figure 6.7: Image details screen
On the right top corner the “Share” button appears, so we can share our story. The screen with sharing action sheet is on Figure 6.8. There are such options as share via Facebook, Twitter, Pocket, Mail and so on.

![Share Screen](image)

**Figure 6.8:** Share screen

Should say, that application represents only core features of all modules, but they have much wider functional. There are also about 3 features that we did not present in this application, because of their purpose, but rest of the features are not so important as described above. We can see, that using only the library, it was very easy to create useful and interactive app.
7 Conclusion and Future Work

This chapter is final one, and here we summarize all done work and tasks, explain how we meet goals and criteria that we define in first chapter, and discuss how we solve all problems that we meet during development. Also tests and future tasks for development will be discussed in this chapter.

7.1 Conclusions

We developed a reusable iOS components library, which allows plug-in of these modules to the existing project in a very easy way. This bunch of components can help Softwerk company to fix a lot of issues and problems, that was described in this thesis. Framework efficiently rises up quality, development speed and reusability of projects.

The first goal (chapter 1.2) to meet was to define a component architecture, which supports the composition of apps using reusable components. Criteria were fulfilled. As we can see from Interaction between components (Section 4.1) there is not direct interaction between final app, and frameworks. It means that we do not need to use any supported classes for inserting modules into the app. Using common localized strings and resources bundle, developer can easy adapt module to the applications requirements. As all modules have the same structure, it should not take a lot of time to understand how to use different parts of Library. We have a clear definition of what a component is, how components are implemented based on the iOS SDK, and how to create a new app based on a set of components. It is able to share common configuration and reuse existing functionality.

Implementation of the new components that satisfies Softwerks needs and solves posed problems was the second goal (chapter 1.2) to meet. Library contains 12 new components, which are providing valuable features to an app with minimal implementation and configuration effort. For testing reusability of code, the system with all features has been created. Among code that connected to the application logic there were code for using reusable modules. After LOC (Lines Of Code) counting without included modules and with them, we determined, that the code needed for reusing components is about less than 20% of the code. The formula of calculation presented below (1):

\[
\frac{f\text{LOC}}{100\%} \times r\text{LOC}
\]

where \(f\text{LOC}\) – all amount of lines of code;
\(r\text{Loc}\) – amount of code lines, that we need for reusable elements integration.

Among them are such important features as Gallery, Sharing, Image picker, Popover calendar, Image editor and so on, that are widely and often used during development process in Softwerk.

As all modules in a system have separate classes and logic, each component can be used separately without any additional separation. So we only can create the object of some type,
configure it as we need (change images, frames, languages, texts and so on) and use inside
the app. In this case we maximize the reuse potential of written code. The **third goal**
(chapter 1.2) was met because classes that are similar for at least two components are
available in an abstract and reusable way inside the app.

The **fourth goal** (chapter 1.2) is to make the components and framework usable for
developers. In case of big sizes of all components, we had to create application, which
demonstrate main possibilities of the library. In this application we used most important
functions and controls from the library, but even with help of this small parts, we created
functional and interactive application.

Documentation and comments inside the systems - it is a very important part of
development. Although documentation creation took a lot of time, it helps another developers
to create their own apps quickly in a future, and can keep a lot of development time. All
classes and main parts inside the library was widely documented and commented, so there
will not be any difficulties in the code understanding.

The **fifth goal** (chapter 1.2) is make standard UI elements more interactive, using
animation and transformation. There are about 3 direct features inside the library, that
animate existing elements, such as Stripe menu, Tabs menu, Animated view controller. These
features do not have a lot of logic inside; they are responsible only for interactive and fancy
look of existing elements. Also a lot of different modules are using Core Graphics
Framework to look more interesting and unusual. There are a lot of custom animation
features inside the modules.

Since we reached all defined goals the we can accomplish that provided system reach
delivered result. This framework will work on real environment and in real company, so we
can make a lot of tests and find and fix main errors and logical imperfections, so future
customers will accept final version.

### 7.2 Future Work

In the future works we plan to start with some interesting tasks. While one of the main tasks
of the system is improving apps quality, it is possible to add more features or improve
existing ones. This step should help us make the product more stable an rise up usability.
Developers will have a lot of sets of configuration for different applications.

To extend the library, it is important to create new features that will satisfy common
needs of customers of Softwerk. The first feature that we would like to create in future is
You-Tube integration, so developer can create multimedia apps. For example, it would be
good to create some interface for fetching videos according to the travel place, and present it
in some way inside the user application. Another important function of this module -
possibility to upload users own apps on You-Tube. There is very convenient API for this goal
- Data API protocol [24].

The media module also is very important for the Travel apps. Therefore we would like to
create audio/video module that provides all necessary functionality for creation users own
media files. It should also have interface for sharing via all famous services and stores that
files inside the app. For this purpose we would like to use AV Foundation Framework [25], because it was been created specially for creating such kind of features.

We would also like to improve existing controls. One of the most important controls in this system, we think, is Sharing control. It uses a large number of different SDK and API to implement their functionality, and architecture of it is the most difficult. On the subsequent development of this control function will be added to the sharing services such as Instagram, Evernote and Google plus. Instagram is one of the most popular services for sharing images and photo. Evernote is a service for creating annotations, notes, events and other. Connection of this service will help to make notes even while traveling. Google plus is a social network, widely used in professional circles.

Also an important task is to create a distributed database, which will be integrated with the user database, and programmer will have the same access to it as to the native database inside the application. For this purpose, a very good fit to use Core Data framework [26]. This framework has all the necessary features for sharing and merging the multiple databases in a single one. This can be achieved by exhibiting certain properties database in the parent application. For now library use SQLite framework to store all data.

It is also important to improve the gallery. Now the cells have a fairly simple interface. While the development process the ability of cells as well as gallery will be enhanced. Plans are to make more complex structure of cells, to display the content in different formats, as well as to adapt gallery to the possibility of sharing several elements in the same time.

Planned improvements can partially cover the needs of the customers, however, these improvements will improve product quality and expand its overall capacity.
References
[16] Wikipedia Property list. Available at:

[17] Git official documentation. Available at:

[18] Wikipedia Git. Available at:


[20] Apple developers portal. Available at:

[21] Apple developers portal. Available at:

[22] Apple developers portal. Available at:

[23] Apple developers portal. Available at:

[24] Google developers portal. Available at:

[25] Apple developers portal. Available at:

[26] Apple developers portal. Available at: