SUPERVISOR'S COMMENT

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

OPPONENTS' COMMENT

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HÔI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Họ tên sinh viên: **Nguyễn Bá Anh** Số thẻ sinh viên: **102120129**

Lớp: **12T2** Khoa: **Công nghệ thông tin** Ngành: **Công nghệ phần mềm**

1. *Tên đề tài đồ án:*

iOS cloud-based Scanning, Speaking and Translation App.

2. *Đề tài thuộc diện: ☐Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện*
3. *Các số liệu và dữ liệu ban đầu:*

……..……..……...

...……

…..……..……..……

4. *Nội dung các phần thuyết minh và tính toán:*

…...……

…...……

…...……

…...……

…...……

5. *Các bản vẽ, đồ thị (ghi rõ các loại và kích thước bản vẽ):*

…...……

…...……

…...……

…...……

6. *Họ tên người hướng dẫn:* ***Lê Thị Mỹ Hạnh***
7. *Ngày giao nhiệm vụ đồ án: ……/……/201.....*
8. *Ngày hoàn thành đồ án: ……/……/201.....*

Đà Nẵng, ngày tháng năm 2017

Trưởng Bộ môn …….. **Người hướng dẫn**

ABSTRACT

Project: **iOS cloud-based Scanning, Speaking and Translation App**

Student: **Ba Anh NGUYEN**

Student ID: **102120129** Class: **12T2**

The basis of this project was to develop an iOS application using cloud based services in order to make a useful tool for global users, includes these features: extract text from images, translate in over 100 languages, speak in 36 voices of 29 languages. The materials I used to create this piece include: The *Swift* programming language, *iOS* platform with *CocoaPods*, Google Cloud *Vision API* and *Translation API*, *Realm* mobile database. The techniques I applied in order to manipulate the project include: analyzing and mockup design, *UML* (Unified Modeling language) technique, *OOP* (Object Oriented Programming) principal, *MVC* (Model – View - Controller) architecture. My overall approach was to create a real product for real users especially travelers and language learners beside a personal purpose that may improve my skill in mobile app development.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere thanks to teachers in Information Technology Faculty, University of Technology and Science who have imparted valuable knowledge to me in the past year and facilitated me to study and complete this thesis. In particular, from the bottom of my heart I would like to thank sincerely, to **PhD. My Hanh LE** who enthusiastically guided and helped me the duration of the subject.

For the current achievement, I am deeply indebted to my beloved grandparents, parents and family members for their wholehearted encouragement and support as well as giving favorable physical and mental conditions to me during the learning process and the implementation process of this thesis.

Despite of trying to complete the project within the scope and capabilities allow, it's inevitable for this thesis not having mistaken. I am looking forward to receiving the attention and consideration from the valuable comments of teachers in order to that thesis is more completed.

Da Nang, June 2017

Sincerely,

Ba Anh NGUYEN

GUARANTY

I guarantee:

1. The contents of this senior project are performed by myself following the guidance of **PhD. My Hanh LE.**
2. All references which used this senior project thesis, are quoted with author's name, project's name, time and location to publish clearly and faithfully.
3. All invalid copies, educated statute violation or cheating will be born the full responsibility by myself.

Student,

Bá Anh NGUYỄN

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ABBREVIATION

Abbreviation	Explanation
API	Application Programming Interface
App	Application
Demo	Demonstrate / Demonstration
gRPC	Google's Remote Procedure Call
GUI / UI	Graphical User Interface / User Interface
iOS	iPhone Operating System
JSON	JavaScript Object Notation
LLDB	Low Level virtual machine Debugger
Misc.	Miscellaneous
MVC	Model – View – Controller
OCR	Optical Character Recognition
RELP	Read – Eval – Print Loop
REST	Representational State Transfer
Scan	- A saved photo inside the app after capturing and extracting texts successfully. - An action to capture a photo for the app.
URL	Uniform Resource Locator
Vs.	Versus

OVERVIEW

♦Project introduction

Nowadays we are witnessing the development of cloud computing which has been being changed our life with an incredible speed. This is the first time in history a normal developer can do something "big", the gap between an idea and the production is diminished with the help of cloud computing - where advanced technologies are distributed with scalable prices to transform your ideas to real products, no need a large and strong system as the key requirement.

This is a new playground for developers, rise 2 years ago and it becomes a hot trend now. A lot of ideas were considered impossible in the past become possible at the moment. Because of that, I decide to apply some cloud services to build an application that help user *scan* and *translate* static photos and images, it also support *synthesizing* text – a support tool on mobile devices consists of useful usages: translate sign panels, read printed books or magazine articles, addresses, numbers... etc, in top languages around the world (English, Spanish, French, Chinese, etc.). This is really nice for travellers or learners who want to get a quick translation both in text and sound: *translate more than 100 languages, speak in over 30 languages*.

In the vast of strict racing among top IT companies such as Google, Microsoft or IBM, it is not hard to choose a service provider to help us make apps easier and quicker, save us a lot of time. Therefore, I tried to make this app from an initial idea with the Swift language for iOS platform, inherit Google Cloud Platform services. Everything is new together, I hope this is a good production that can impact good values in real life.

♦Study development & Significance

This thesis sets target on developing productions – the main job that an engineer usually does after graduate. That means it requires me to design a mobile application architecture to work well with the services. In 3-month period of time, the project has to include all necessary steps to produce a product: analyzing, designing, implement, testing and maintenance. I also have a chance to study Swift 3 – a new programming language for iOS platform and now is for every platform, a short-cut of knowledge that has not happened in university till now. In a short period of time set me in self study new language and building an application I hope it can train my skills up. This is a big point in my self development to make me become a productivity engineer in the future.

CHAPTER 1: THEORIES & TECHNOLOGIES

1.1. iPhone Operating System

1.1.1. Introduction

iOS is the operating system that runs on iPad, iPhone, and iPod touch devices. The operating system manages the device hardware and provides the technologies required to implement native apps. The operating system also ships with various system apps, such as Phone, Mail, and Safari, that provide standard system services to the user.

1.1.2. iOS layered architecture

At the highest level, iOS acts as an intermediary between the underlying hardware and the apps you create. Apps do not talk to the underlying hardware directly. Instead, they communicate with the hardware through a set of well-defined system interfaces. These interfaces make it easy to write apps that work consistently on devices having different hardware capabilities.[1]

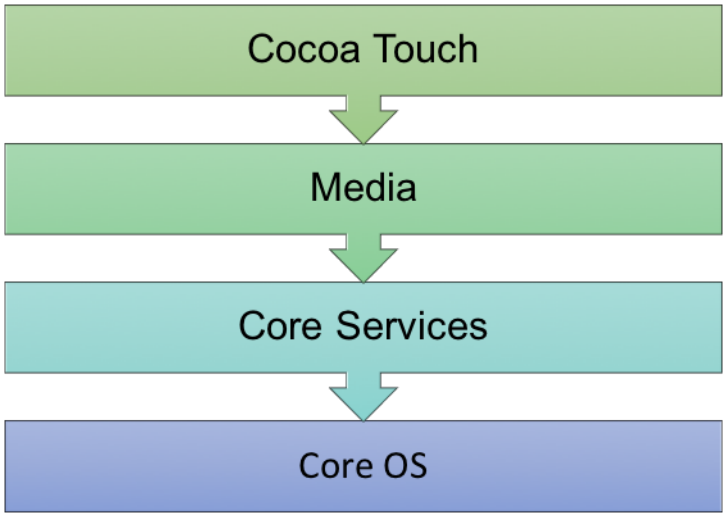


Figure 1.1: Layered architecture

The implementation of iOS technologies can be viewed as a set of layers. Lower layers contain fundamental services and technologies. Higher-level layers build upon the lower layers and provide more sophisticated services and technologies.

Apple delivers most of its system interfaces in special packages called *frameworks*. A *framework* is a directory that contains a dynamic shared library and the resources (such as header files, images, and helper apps) needed to support that library. To use frameworks, you add them to your app project from Xcode.

1.2. Swift programming language

1.2.1. Introduction

Swift is a general-purpose programming language built using a modern approach to safety, performance, and software design patterns.

The goal of the Swift project is to create the best available language for uses ranging from systems programming, to mobile and desktop apps, scaling up to cloud services. Most importantly, Swift is designed to make writing and maintaining correct programs easier for the developer. To achieve this goal, we believe that the most obvious way to write Swift code must also be: *safe*, *fast*, and *expressive*.

On December 3, 2015, the Swift language, supporting libraries, debugger, and package manager were published under *the Apache 2.0 license* with a Runtime Library Exception. Swift is now free to be ported across a wide range of platforms, devices, and use cases. [2]

1.2.2. Architectural overview

The features of Swift are designed to work together to create a language that is powerful, yet fun to use. Some additional features of Swift include:

- Closures unified with function pointers.
- Tuples and multiple return values.
- Generics.
- Fast and concise iteration over a range or collection.
- Structs that support methods, extensions, and protocols.
- Functional programming patterns, e.g., map and filter.
- Powerful error handling built-in.
- Advanced control flow with *do*, *guard*, *defer*, and *repeat* keywords.

The Swift language is managed as a collection of projects, each with its own repositories. The current list of projects includes:

- The Swift compiler command line tool.
- The standard library bundled as part of the language.
- Core libraries that provide higher-level functionality.
- The LLDB debugger which includes the Swift REPL.
- The Swift package manager for distributing and building Swift source code.
- Xcode playground support to enable playgrounds in Xcode. [3]

1.3. Cloud services

1.3.1. Google Cloud Platform

Google Cloud Platform is a cloud computing service by Google that offers hosting on the same supporting infrastructure that Google uses internally for end-user products like Google Search and YouTube. Cloud Platform provides developer products to build a range of programs from simple websites to complex applications.

Google Cloud Platform is a part of a suite of enterprise services from Google Cloud and provides a set of modular cloud-based services with a host of development tools. For example, hosting and computing, cloud storage, data storage, translations APIs and prediction APIs. This product is not free so we have to purchase for our account first before starting to work with it.

In this scope of the project, we just need 2 services of Google Cloud Platform: Vision API – OCR Feature and Translation API – Full kit included detecting language feature and translating feature.

1.3.2. Google Vision API

Google Cloud Vision API enables developers to understand the content of an image by encapsulating powerful machine learning models in an easy to use REST API. It quickly classifies images into thousands of categories (e.g., "sailboat", "lion", "Eiffel Tower"), detects individual objects and faces within images, and finds and reads printed words contained within images. You can build metadata on your image catalog, moderate offensive content, or enable new marketing scenarios through image sentiment analysis. Analyze images uploaded in the request or integrate with your image storage on Google Cloud Storage. [4]

1.3.3. Google Translation API

Cloud Translation API provides a simple programmatic interface for translating an arbitrary string into any supported language using state-of-the-art Neural Machine Translation. Translation API is highly responsive, so websites and applications can integrate with Translation API for fast, dynamic translation of source text from the source language to a target language (e.g., French to English). Language detection is also available In cases where the source language is unknown. The underlying technology pushes the boundary of Machine Translation and is updated constantly to seamlessly improve translations and introduce new languages and language pairs. [5]

1.3.4. RESTful & APIs

Representational state transfer (REST) or *RESTful* Web services are one way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.

We require this to work with the two services above although they can work well with gRPC also. But it is more applicable for a student knowledge when I apply REST into this project. This is a useful method to work with APIs.

1.4. Open Source & CocoaPods

Open-source software (OSS) is computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. Open-source software may be developed in a collaborative public manner. According to scientists who studied it, open-source software is a prominent example of open collaboration.

CocoaPods manages library dependencies for Xcode projects. The dependencies for your projects are specified in a single text file called a *Podfile*. CocoaPods will resolve dependencies between libraries, fetch the resulting source code, then link it together in an Xcode workspace to build your project. Ultimately the goal is to improve discoverability of, and engagement in, third party open-source libraries by creating a more centralized ecosystem.

CocoaPods supports almost every way to get source code: *git*, *svn*, *bzr*, *http* and *hg*. It is acceptable to use own private code repository to manage dependencies. It only requires a *git* repo, no server necessary.

CHAPTER 2: ANALYSIS & DESIGN

2.1. Analyzing

2.1.1. Introduction

At the first time mention to development of iOS, we should focus on iPhone first. The application has to work properly on iPhone with iOS 9.0 and above.

As the topic named already, the app contains at least 3 main features equivalent to scanning feature, translation feature and speaking feature. It also needs some miscellaneous features to make the app work smoother and more accurate.

2.1.2. Scanning Feature – Convert scanned texts to digital texts

This feature provides the app to use camera to take photo or pick a photo from device library, process it and extract text from it if possible. It saves captured photo for later uses, too. This feature applies Google Cloud Vision API to process image online, and Realm mobile database to store and persist photos offline. Of course it includes cropping feature to help users easier to choose a cropping-zone inside the captured image.

Vision API will automatically detect the language in the photo. If the photo contains texts that have more than one language, a common language in that list will be picked.

2.1.3. Translating Feature – Translate texts from language to language

This feature is the next feature after step extracting texts above. By applying Google Cloud Translation API, it can translate more than 100 languages text-to-text.

My job is to make an adaptive interface to work with Translation API, included: language picker, make languages names and their locales fitting with Apple language structure, and solve conflict between language code, region code behind the locale to provide accurate flags that represent for countries and regions, dialect and script system differences (left-to-right, right-to-left oriental, Chinese simplified vs. traditional character, etc.).

This is one of the most complicated feature of the application, because there are lots of problems come from territories and cultures between countries, like China, Hong Kong and Taiwan for an example. Or some conflictions between Latin, Arabic and Sanskrit writing systems that happening in Middle East and South Asia.

Especially there are many languages in a country or a language is the main official language of many countries. For examples: English is used in the US, UK, Ireland, India, Australia and so on; Spanish is official language of more than 25 countries, etc. The same for French, Chinese, …This required me to do a small research of linguistic field before starting to develop this feature. Additional information can be found in the appendix of this thesis.

Translation API can automatically detect the source locale of the text, but it is better if users specify a language for it before translating the text.

2.1.4. Synthesizing Feature – Speaking texts

By applying Apple synthesizing feature which is built-in with iOS from version 7.0, it is easy to make the app can speak texts with an acceptable result. Using some trick to highlight current word of speech feature. Because this feature is included in iOS, it can work offline.

Currently Apple supports 29 languages with 36 dialects worldwide. Top common languages are supported already. [6]

2.1.5. Dialect voice selection

Because the complexity of territory, a language may be spoken in many countries with a difference in speaking dialect, so it is better to give user a choice to choose a dialect that they love using.

At this time, these languages are chosen to make change: *English, Spanish, French, Italian, Dutch* and *Chinese Mandarin*.

2.1.6. App configurations and Settings

To improved user experience, the application has to remember some values happened in using time, like the last language users chose, configuration. User can enhance experience by enable quick mode and gray photo, for a better recognized result.

This feature will be applied under the operating system *user defaults* which provided by iOS, developer can store value to the system through a pair of key-value that is very easy to work with.

2.2. Wire framing

2.2.1. Sidebar Design

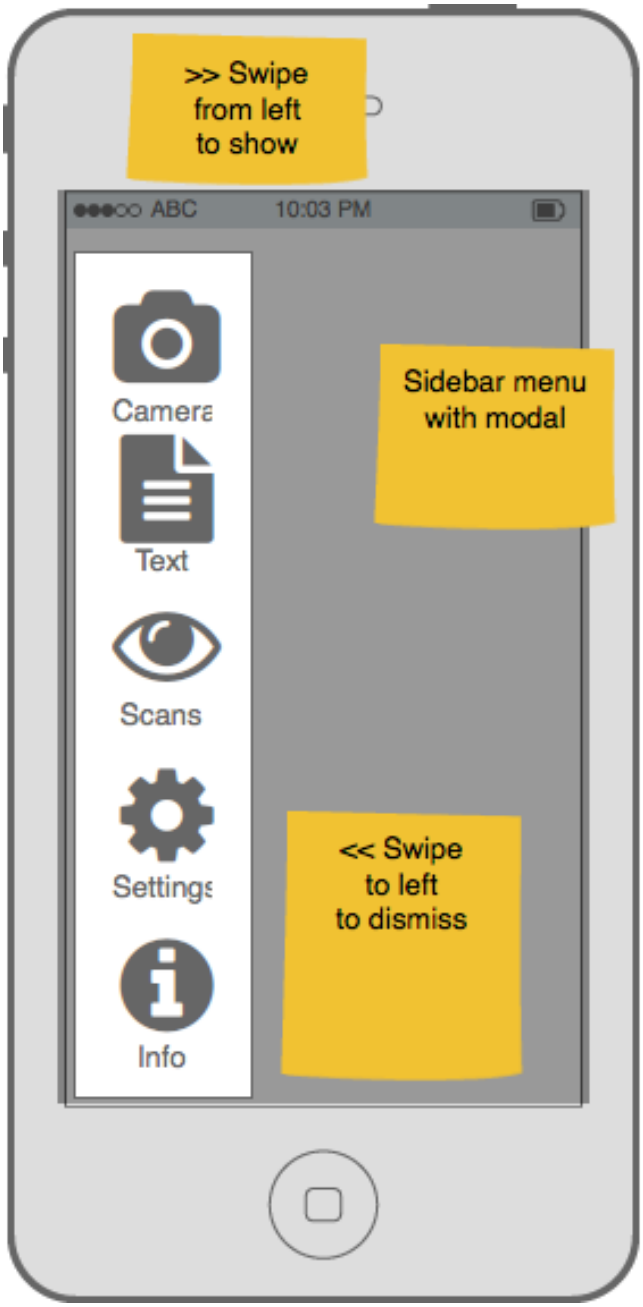


Figure 2.1: Camera View Design

Sidebar is the main navigator of the app. It manipulates navigation between views and features. Inside the sidebar is where other views anchor. The app only has sidebar from the left, which can be toggled by swipe from screen edge or a button.

By default, sidebar always float up to root view of each navigation. That means users can toggle sidebar when they are at the root view.

2.2.2. Camera View Design

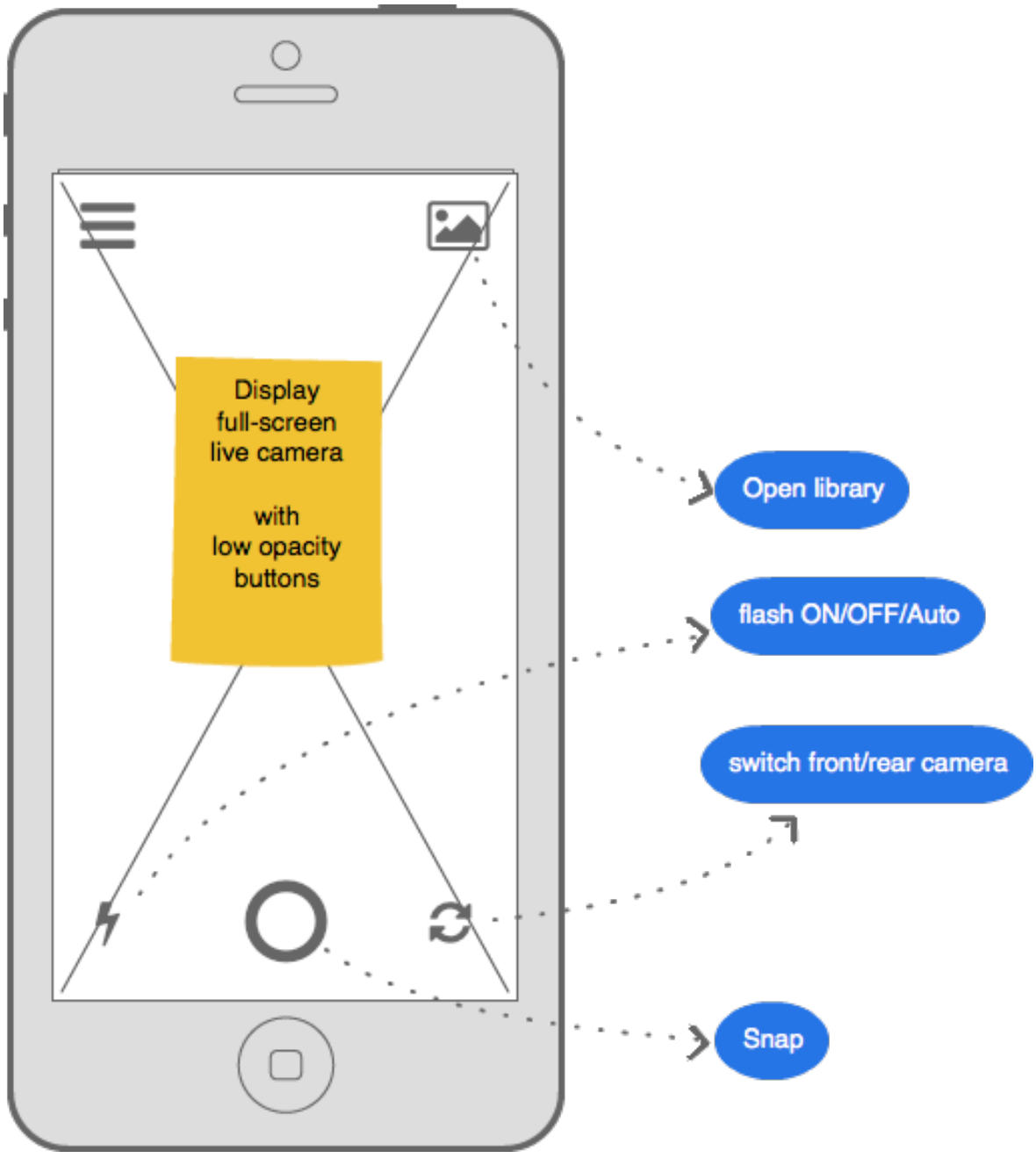


Figure 2.2: Camera View Mockup

At the time application finish launching, camera view is the first thing user can see and easily understand the usage of it. It provides basic camera functions like snap button, pick photo from library, flash toggle and camera selector. User can tap inside the live view to quickly take a focus for a sharper photo.

After snap button was pressed, crop view will work and its mockup can be found in the appendix at the end of this report.

2.2.3. Translation Text View Design

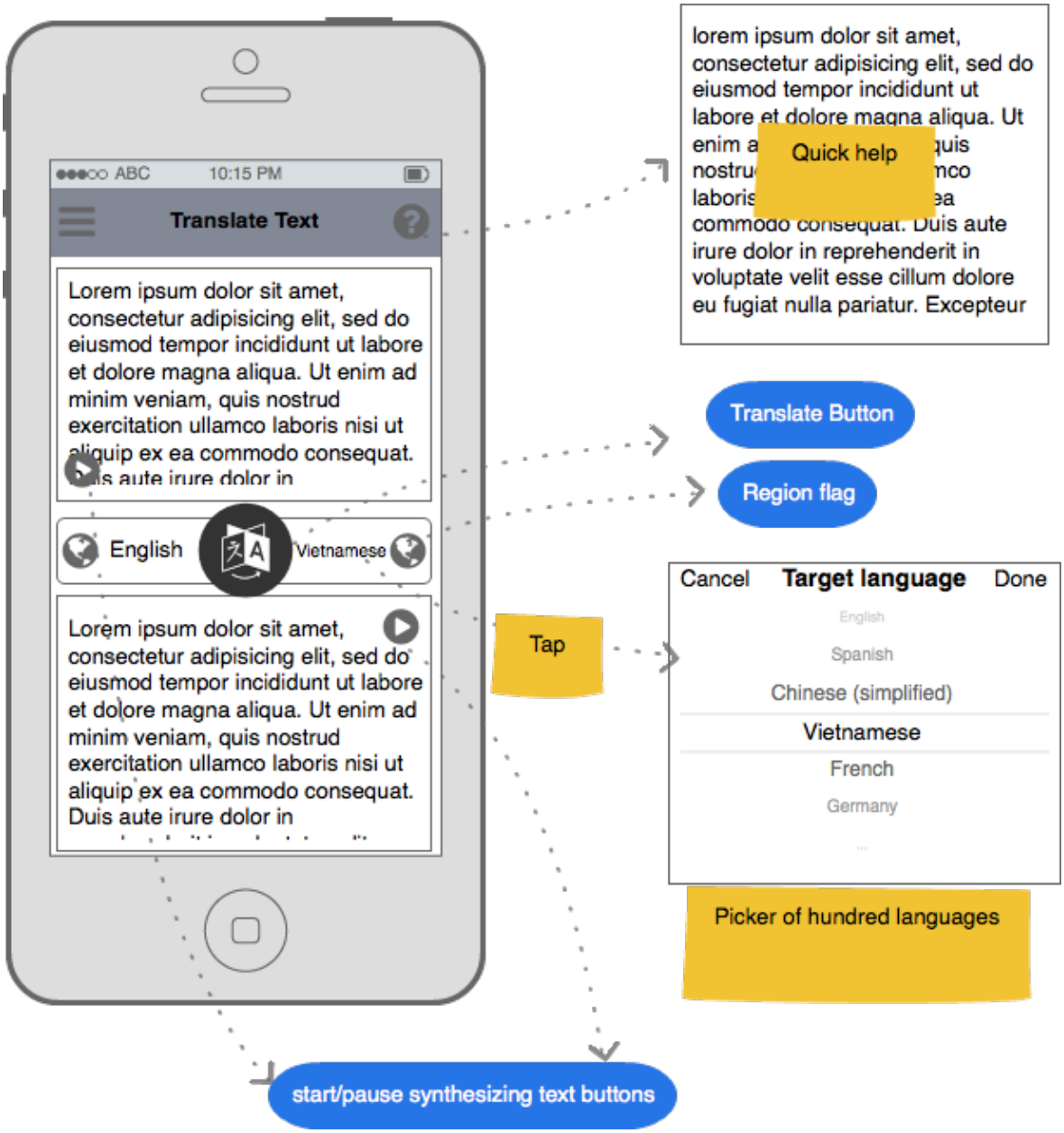


Figure 2.3: Text Translation View Mockup

This is a combination of a simple translator and a synthesizer. It receives extracted text from Google Cloud Vision response. Or user can directly access it and edit text manually. The screen is divided into 3 main parts: source text zone, target text zone and control buttons in the middle. When a language button is pressed, a picker shows a list of all languages that Google Translate supports, it has to be designed to adapt well with Apple Locale, avoid all conflicts and differences between Apple and Google.

In the text zone if the current language supports speaking text, a Play button shows up for user to start speak the text. Otherwise it should be hidden.

2.2.4. Saved Scans View Design

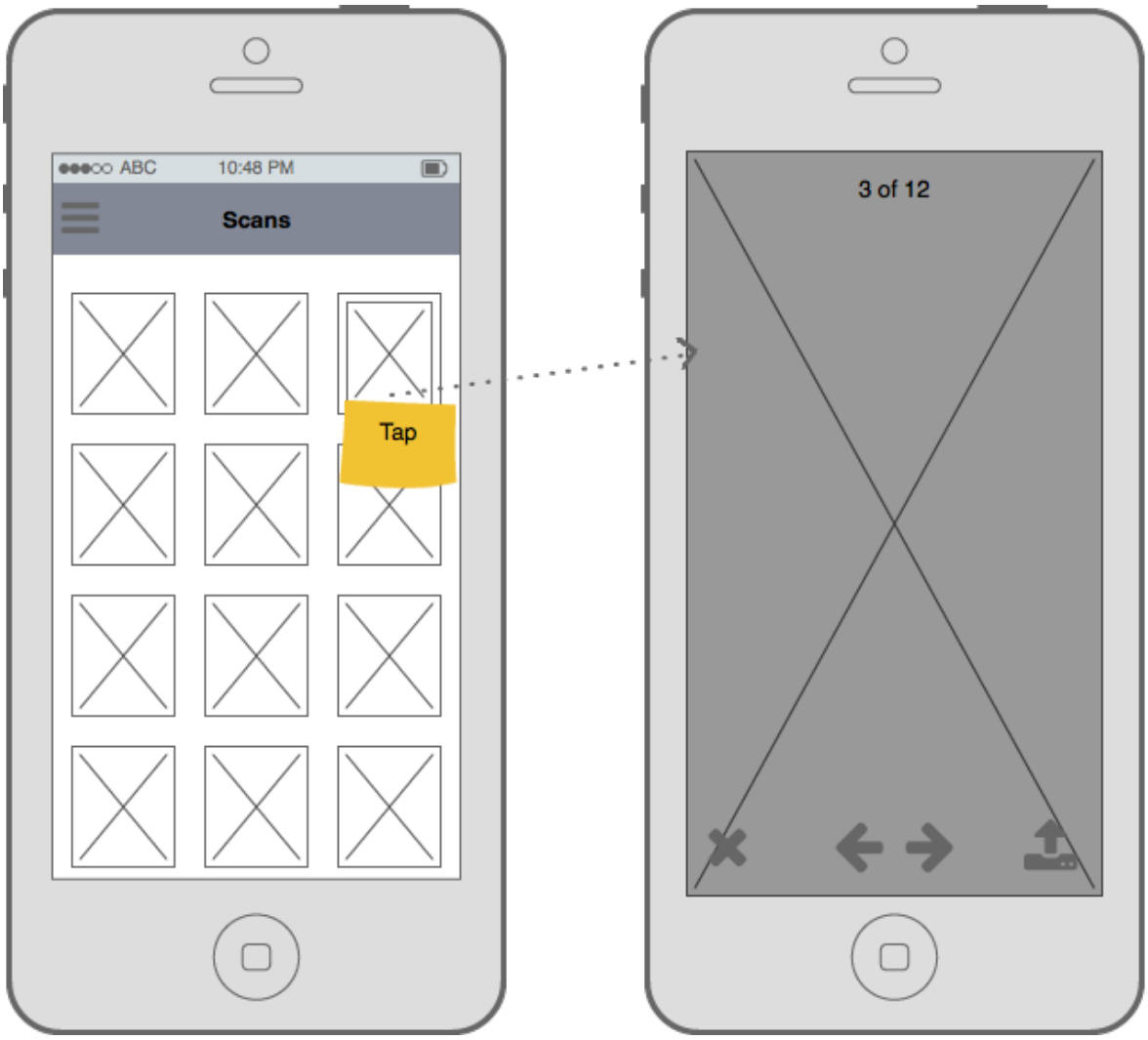


Figure 2.4: Scan and Grid View Design

This design includes a grid view of scans and a detail view for a specific scan we choose. It has to make sure that *lazy-loading* is applied because when the number of scan is numerous, it can slow down the app performance without *this* key-feature.

2.2.5. Settings View Design

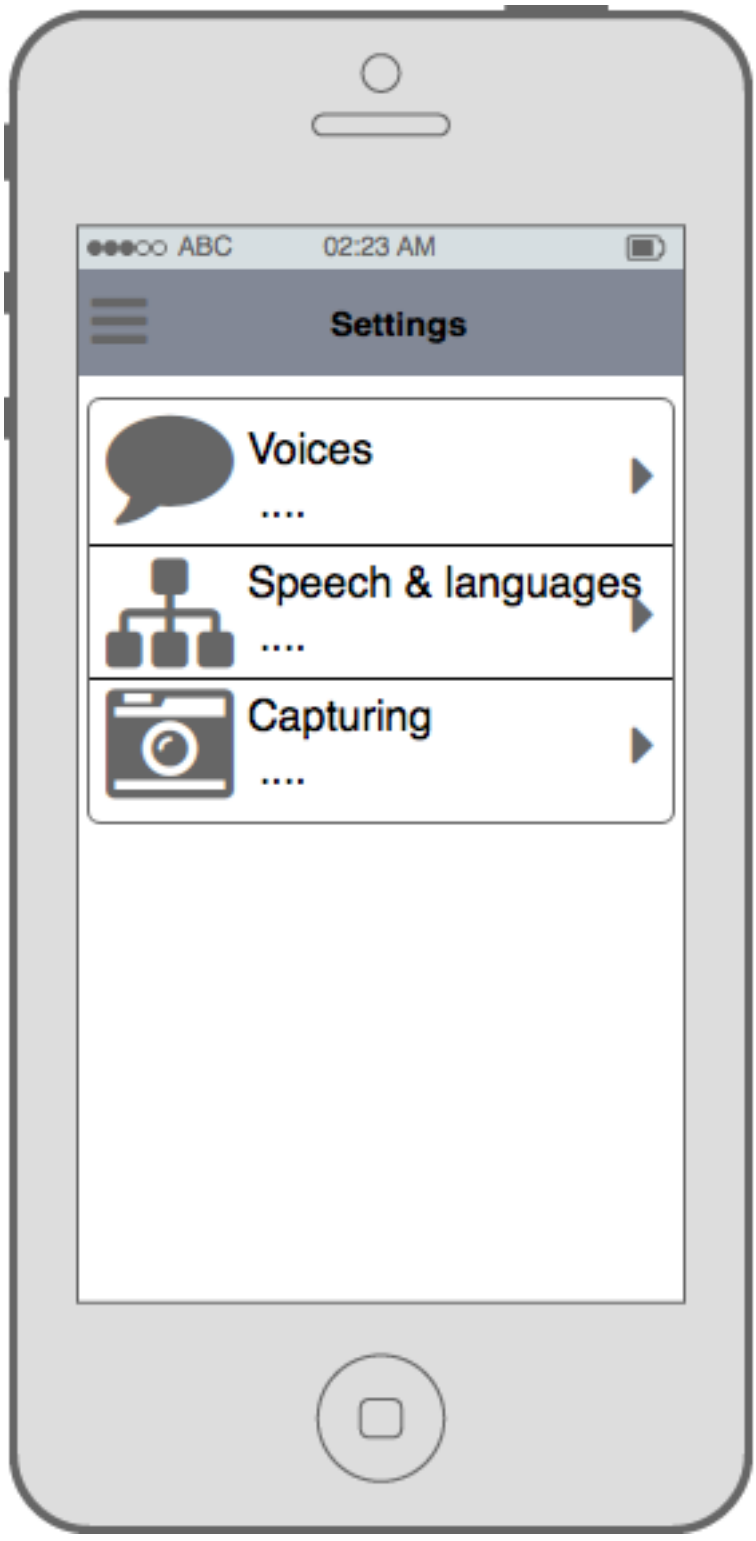


Figure 2.5: Settings Overall View Design

App settings makes user experience better by saving some common values, and for some features need such as Voice, Speech, Capturing.

2.2.6. Other screens

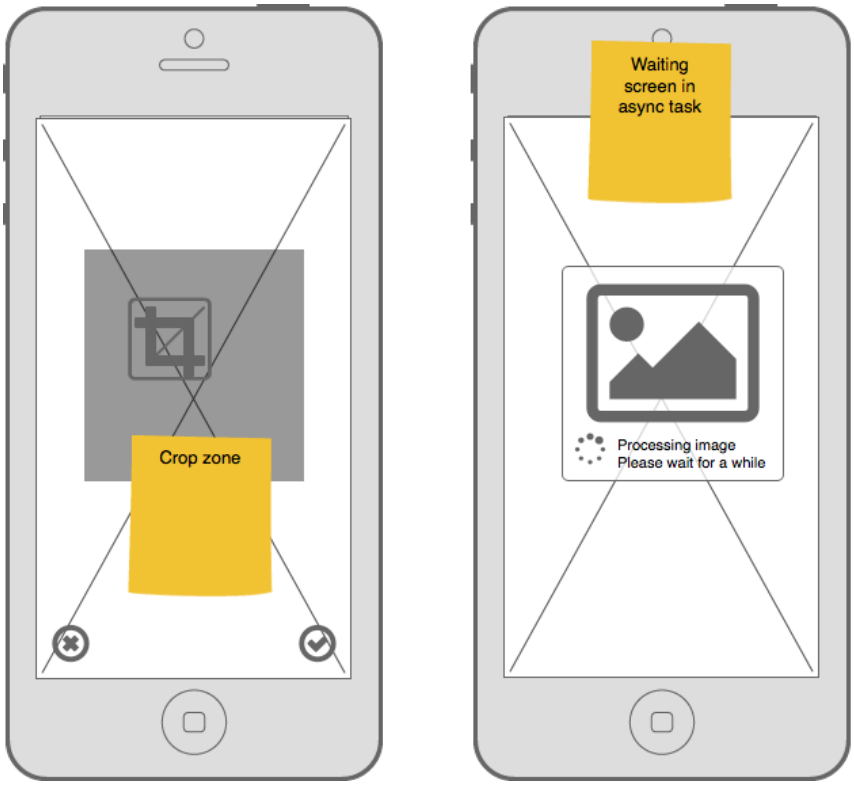


Figure 2.6: Cropping View Design

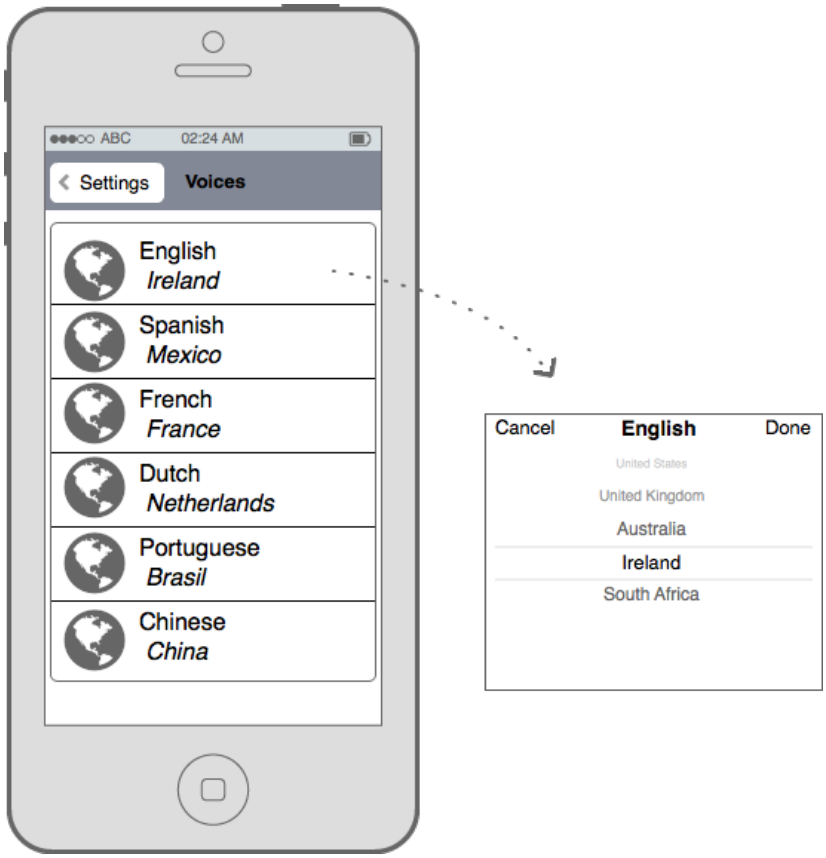


Figure 2.7: Voice Selection View Design

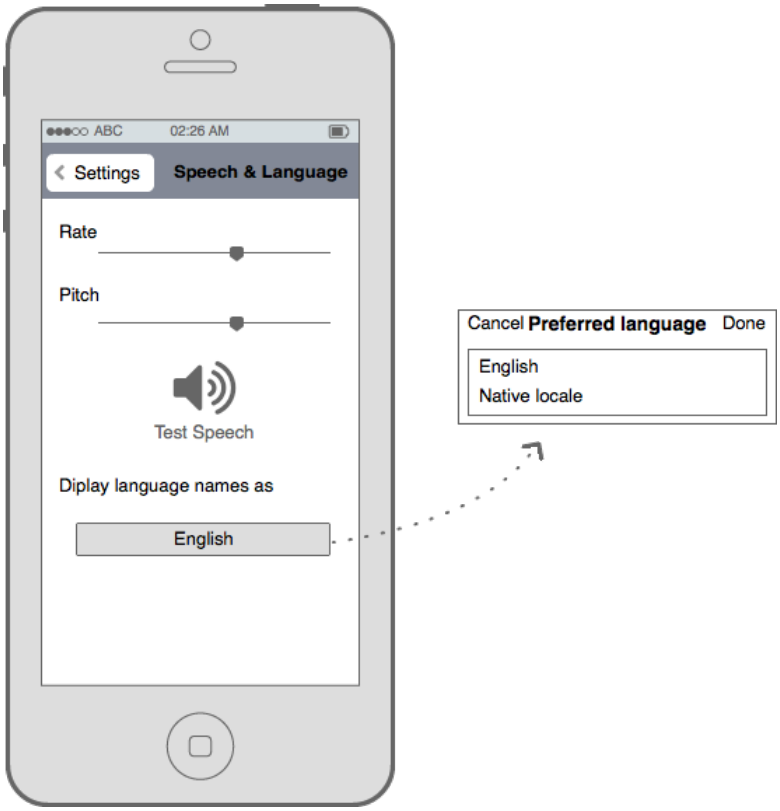


Figure 2.8: Speech & Language Options View Design

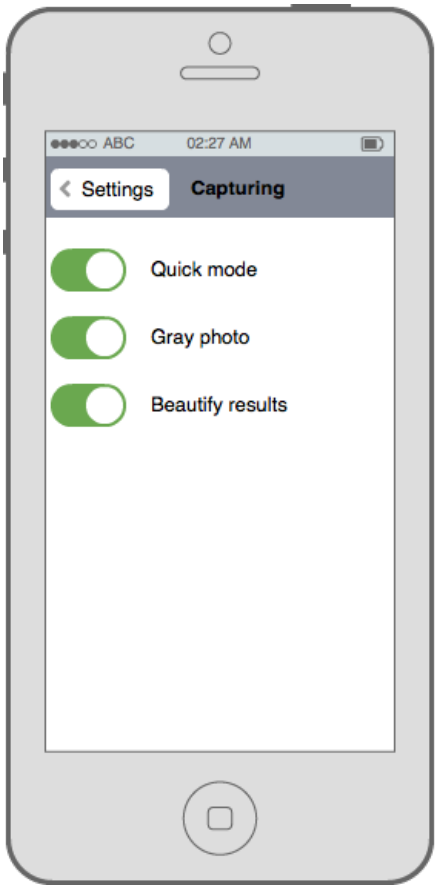


Figure 2.9: Capturing Options View Design

2.2.7. *Assets and Icons*

- Icons used in the app: Free icons from Icons8 application.
- Local stored files: *Locales.plist*
- Amharic font: Abyssinica_SIL.ttf.
- Flag Icon: named in English as Region name. For examples: flag for Vietnamese is "Vietnam.png", for Korean is "South Korean.png"… etc.

2.3. Use-cases Diagram

2.3.1. *Overall diagram*

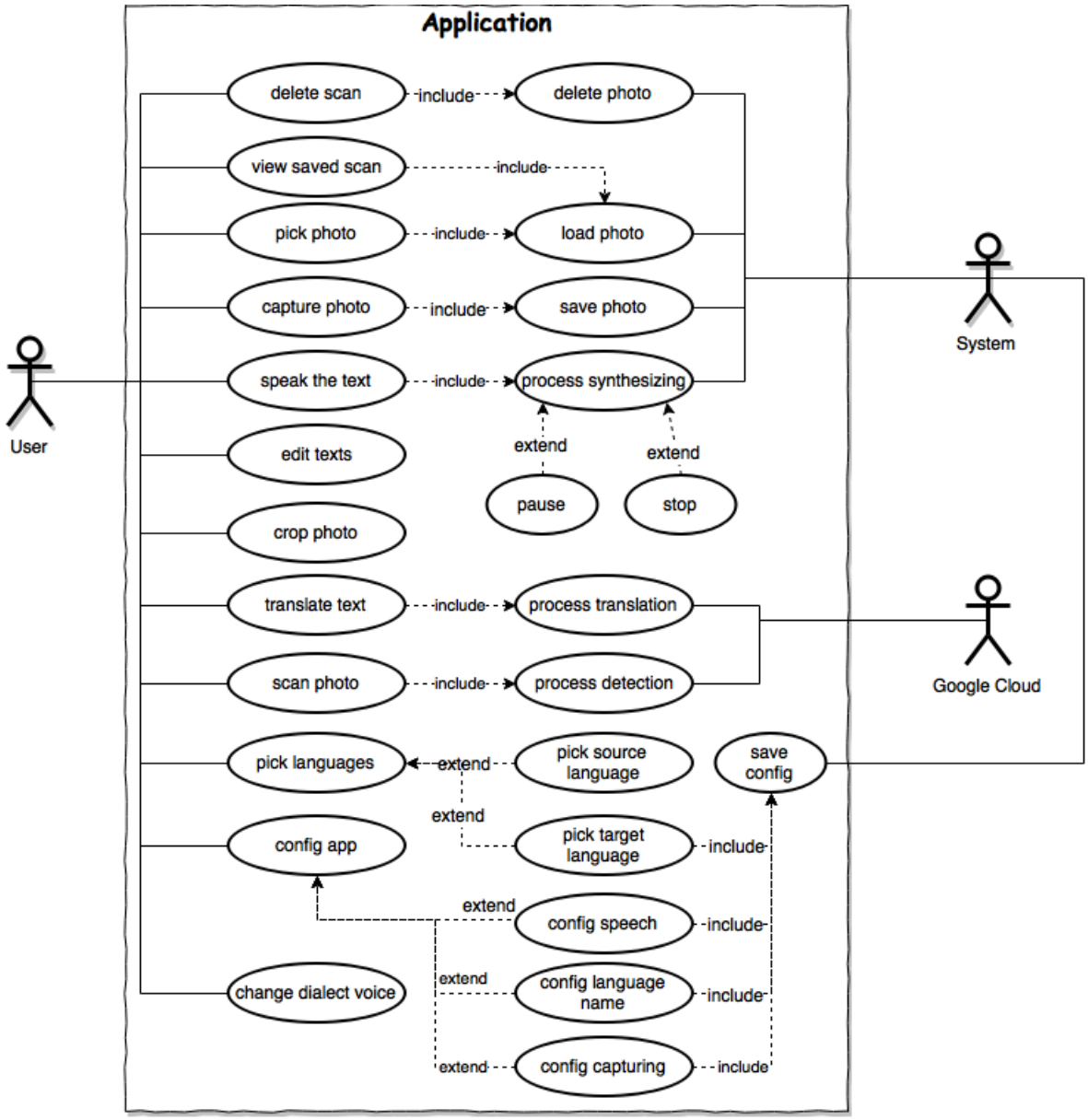


Figure 2.10: Application use-case diagram

2.3.2. Actor description

Actor	Description
User	The user who use the app
System	The operating system which provide background services, interfaces to help the app can run properly.
Google Cloud	Cloud service provider which handles Text Translation and Detection

2.3.3. Use-cases descriptions

These description tables below describe some main use-cases in the application.

Table 2.1: Use-case description – Capture photo

Use case name	Capture photo	
Actor	User	
Description	User capture a new photo from device cameras.	
Pre-conditions	User is in "Camera View" screen	
Flows of events	**Actor**	**System**
		1. Display live rear-camera view
	2. Press [Snap] button	
		3. Capture a photo and drive the photo to Cropping View
Alternative flows	- If user press [Switch] camera button, change the camera from rear to front, and reflected. - If user press [Flash] button, switch flash light mode between: ON, OFF and Auto.	

Table 2.2: Use-case description – Pick photo

Use case name	Pick photo
Actor	User
Description	User pick an existed photo from device library.
Pre-conditions	User is in "Camera View" screen

Flows of events	**Actor**	**System**
		1. Display live rear-camera view
	2. Press [Library] button	
		3. Load and show all photos in the library
	4. Pick a photo	
		5. Drive the selected the photo to Cropping View
Alternative flows	If user click [Cancel] button when showing library, dismiss the library and go back to the live-Camera View before.	

Table 2.3: Use-case description – Crop photo

Use case name	**Crop photo**	
Actor	User	
Description	User crop a photo before starting OCR.	
Pre-conditions	User is in "Cropping View" screen	
Flows of events	**Actor**	**System**
		1. Receive the photo from Camera View or Scan View, prepare cropping.
	2. Choose cropping zone, and press [Done] button	
		3. Make a new photo from cropping zone and start detecting text usecase.
Alternative flows	If user click [Cancel] button when showing Cropping View, dismiss the Cropping View and go back to the live-Camera View before.	

Table 2.4: Use-case description – Save photo

Use case name	**Save photo**
Actor	System
Description	Save the original photo to app database.
Pre-conditions	Detecting text use-case started successfully.

Flows of events	**Actor**
	1. Pause the detetecting text usecase and save the original photo to Realm
	2. After the photo saved, continue previous usecase.
Alternative flows	None

Table 2.5: Use-case description – Scan photo

Use case name	**Scan photo**		
Actor	User, Google Cloud		
Description	Let Google Cloud Vision do scanning and extract text from photo.		
Pre-conditions	User finished cropping photo after capturing from camera, picking from the library, or selecting a photo from saved scans.		
Flows of events	**User**	**Google Cloud**	**System**
			1. Resize the photo to 2MB and build detecting text request with TEXT_DETECTION feature.
	2.2. Wait for responses	2.1. Process the request.	2.3. Show indicator
			3. Extract description text from JSON response, then send the text to Translation Text View.
Alternative flows	If the Internet connection is not available, stop current operation and return to the previous screen.		

Table 2.6: Use-case description – Translate text

Use case name	**Translate Text**
Actor	User, Google Cloud
Description	Let Google Cloud Translation do scanning and extract text from photo.

Pre-conditions	User finished cropping photo after capturing from camera, picking from the library, or selecting a photo from saved scans. OR user manually open "Translation Text View"		
Flows of events	**User**	**Google Cloud**	**System**
	1.2. Edit the source text manually		1.1. Fill the source text if received text from scanning feature
	2. Select source, target languages. Press [Translate] button		
			3. Build the request with TEXT_TRANSLATION feature
	4.2. Wait for responses	4.1. Process the request	4.3. Show indicator
			4. Extract translation and detected language from JSON response, update the target text view
Alternative flows	If the Internet connection is not available, show a notification and stop current operation.		

Table 2.7: Use-case description – Speak text

Use case name	**Speak text**	
Actor	User	
Description	Synthesize text in a text view	
Pre-conditions	User is in "Translation Text View" screen, the language supports synthesizing and the buttons visible.	
Flows of events	**Actor**	**System**
	1. Press [Play] button in the text view	
		2. Speak the text and highlight speech text. Show [Stop] button
		3. Restore button states at the time before speaking

Alternative flows	If user press [Stop] button when speaking, stop synthesizing immediately. If user press [Pause] button when speaking, pause the synthesizing after finishing the current speech text, then replace button icon by "Continue" button. If user press [Continue] button, start synthesizing from the point which paused before.

Table 2.8: Use-case description – Edit Texts

Use case name	**Edit texts**	
Actor	User	
Description	Edit or modify text in source text view or target text view. To prevent user delete all things in text views without any recovery way.	
Pre-conditions	User is in "Translation Text View" screen	
Flows of events	**Actor**	**System**
	1. User modify text in a text view	
		2. Temporary save new changes
	3. User finish and conform that want to keep changes.	
		4. Commit new changes.
Alternative flows	If user does not commit changes, reset the text view to old content before editing.	

Table 2.9: Use-case description – Pick languages

Use case name	**Pick languages**	
Actor	User	
Description	Select source language or target language for the translation or synthesizing.	
Pre-conditions	User is in "Translation Text View" screen	
Flows of events	**Actor**	**System**
	1. Open the picker and pick a language	

		2. Update language name, flag, text alignment, fonts, synthesizing buttons
Alternative flows	If user changes target language, clear target text view and save it into last picked language in the app settings. If the language does not support synthesizing, hide [Play] button If the source and target language is the same, show a notification	

Table 2.10: Use-case description – Configurate application

Use case name	**Configure application**	
Actor	User	
Description	Change app settings and option	
Pre-conditions	User is in "Settings View" screen	
Flows of events	**Actor**	**System**
	1. Select a preferred option	
		2. Display options
	3. Change option values	
		4. Save new changed values.
Alternative flows	None	

Table 2.11: Use-case description – Change voices

Use case name	**Change voices**	
Actor	User	
Description	Select default voice for the language which has many voice.	
Pre-conditions	User is in "Voices Settings View" screen	
Flows of events	**Actor**	**System**
	1. Select a preferred language	
		2. Display picker
	3. Select a voice	
		4. Save new default voice.
Alternative flows	None	

Table 2.12: Use-case description – View saved scans

Use case name	**View saved scans**	
Actor	User	
Description	Display saved scan which the app processed before.	
Pre-conditions	User is in "Scans View" screen	
Flows of events	**Actor**	**System**
	1. Press [Open saved scans]	
		2. Load data from Realm and display grid view of scans
Alternative flows	None	

Table 2.13: Use-case description – Delete a scan

Use case name	**View saved scans**	
Actor	User	
Description	Display saved scan which the app processed before.	
Pre-conditions	User is in "Grid View" screen	
Flows of events	**Actor**	**System**
	1. Select a scan, press [Action], select [Delete]	
		2. Remove scan from Realm and refresh current index.
Alternative flows	None	

2.4. Featuring activity diagrams

2.4.1. Architecture

With a knowledge base at university I applied MVC model to build this application. Basically this application plays a Singleton and Factory design patterns.

Singleton is for application services like calling API, work with database locally, handling application configuration and app helpers.

Networking bases on HTTP and REST. Cloud API authentication uses an API Key to make the request become valid.

Local database engine is Realm instead of default Apple Core Data. Because Realm is a new technology and it supports many platforms with an incredible speed.

2.4.2. App navigation diagram

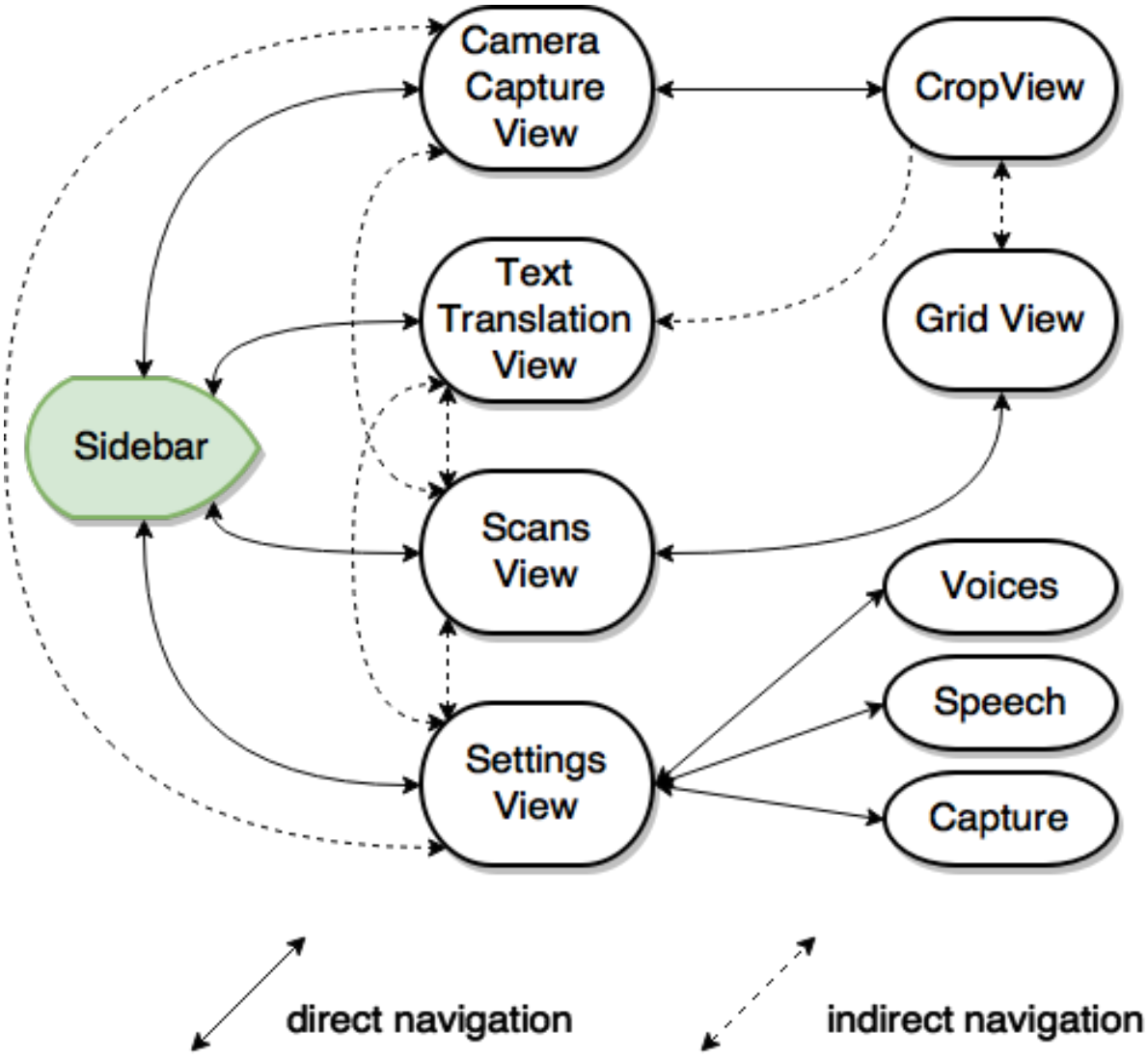


Figure 2.11: Application Navigation

2.4.3. Scanning activity diagram

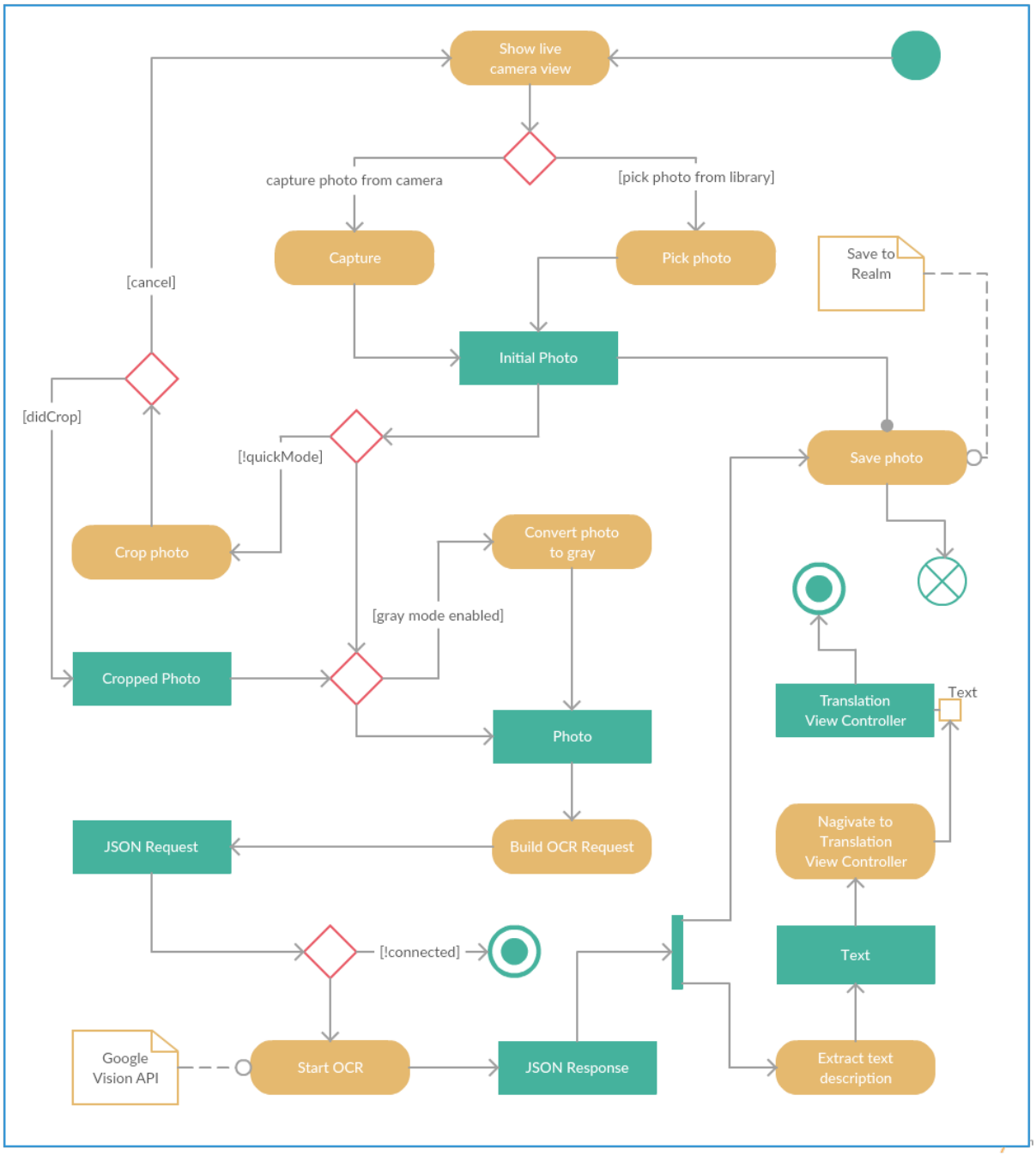


Figure 2.12: Capturing activity diagram

Scanning feature includes some sub features: picking photo from camera or library, pre-process the photo like cropping or converting color to gray, building OCR request, receiving the response and process it, prepare data for the next feature.

The OCR activity is done by Google Cloud, and the response is mapped into the models I made in this app for an easily processing. This also requires internet connection to guarantee that the flow is not broken during running.

2.4.4. Translation activity diagram

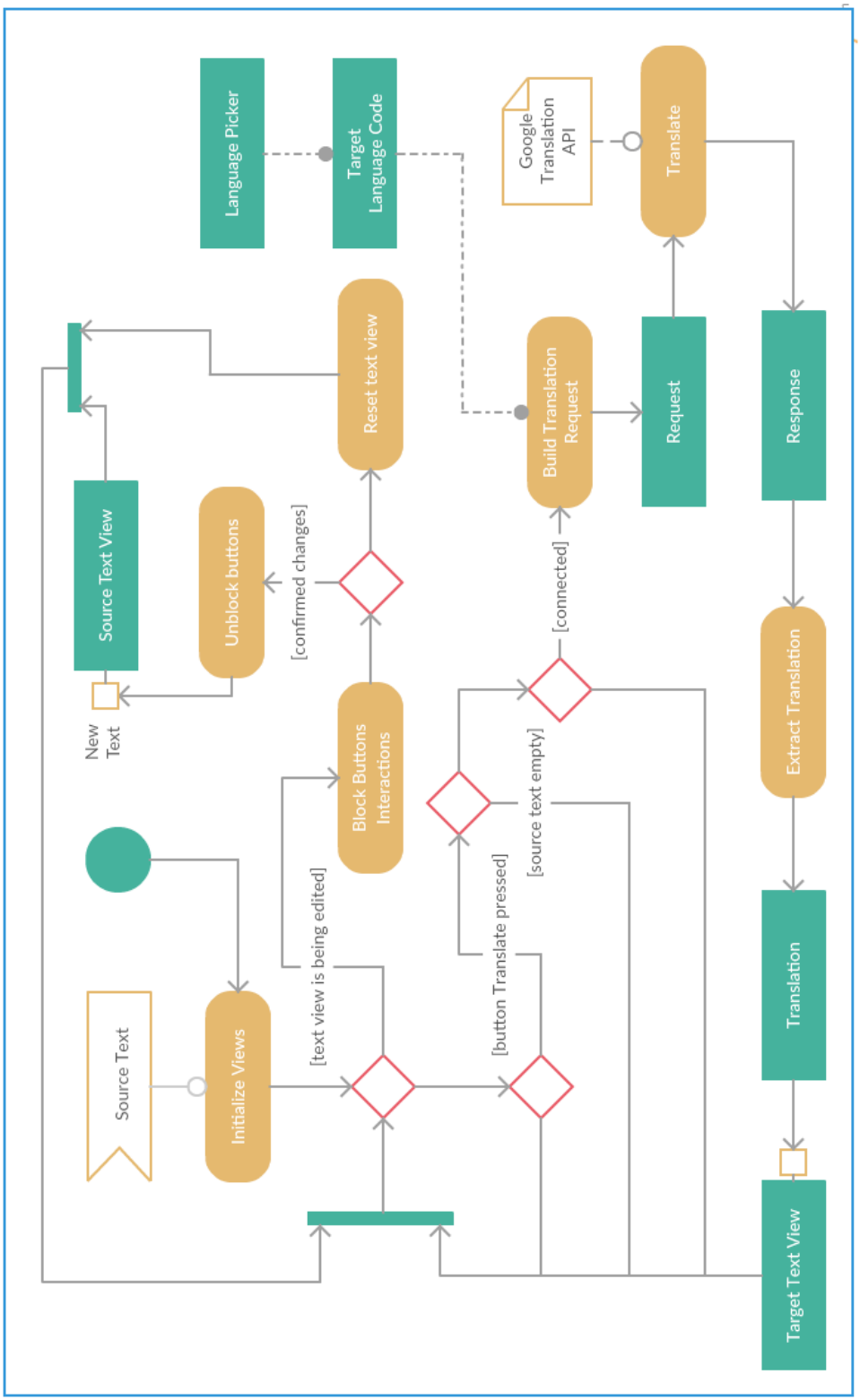


Figure 2.13: Translation activity diagram

This is a mini version of Google Translate, applied Google Translation API to translate over 100 languages. Some languages have specific context such as text

alignment, the script and font. Solving the input and parameters needs to exact because there are too many exceptions when showing the language flags, or language names in very different ways.

This feature and Synthesizing feature are integrated into one View Controller, so the result must adapt well with the other features. Please refer to appendix for more explanation about this complication.

2.4.5. Synthesizing activity diagram

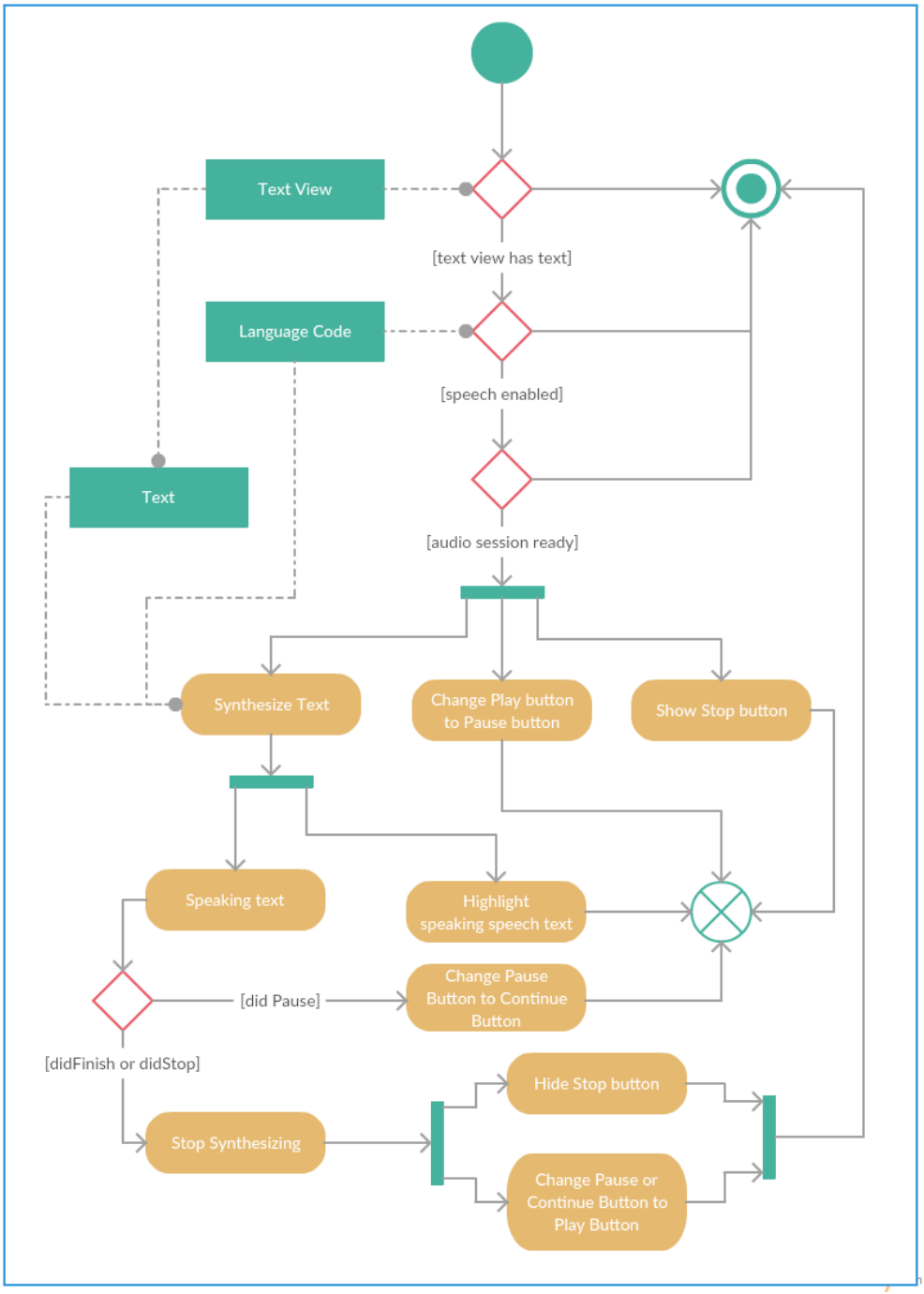


Figure 2.14: Synthesizing activity diagram

CHAPTER 3: DEPLOYMENT & TEST

3.1. Installation notes

3.1.1. Git, Bitbucket and SourceTree

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. (https://git-scm.com)

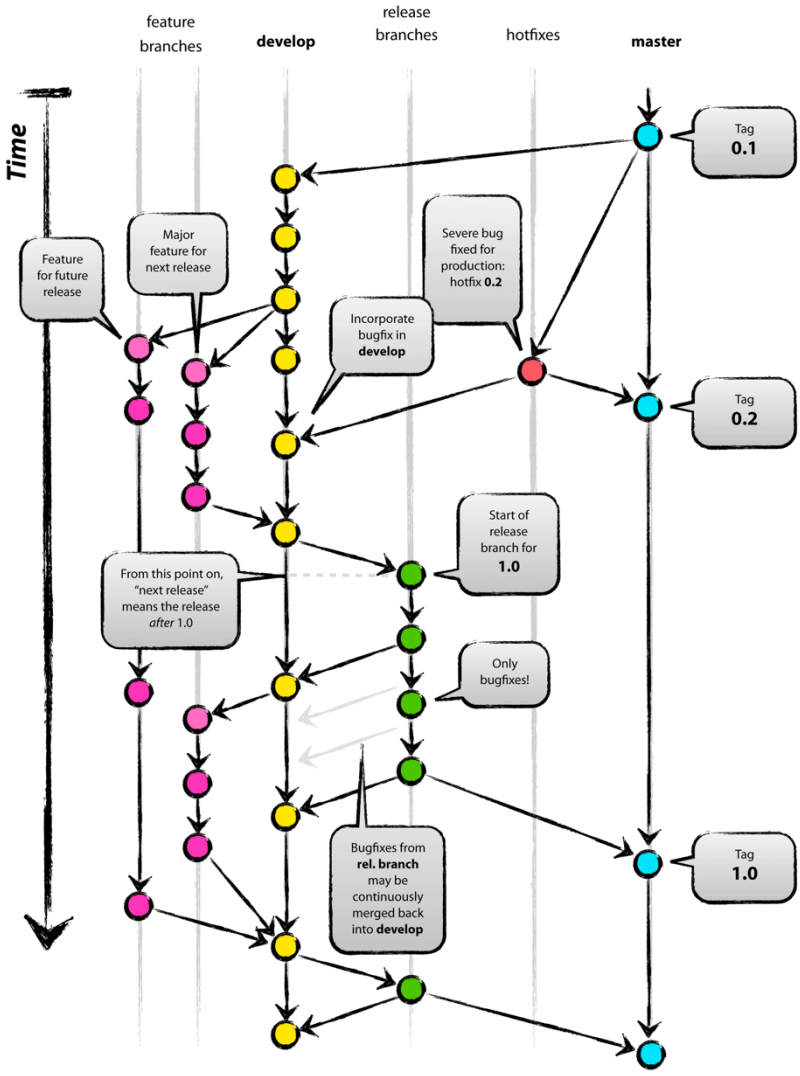


Figure 3.1: Gitflow visualization

But just Git is not enough, I applied an advanced model of Git called *"gitflow"* for an easier development. It makes the app development clear and easy to understand, reduce a lot of complex steps to focus on design and implement. Gitflow is a branching model for Git, created by Vincent Driessen. It has attracted a lot of attention because it is very well suited to collaboration and scaling the development team. [7]

Bitbucket is a web-based hosting service for source code and development projects that use either Mercurial or Git revision control systems that is owned by Atlassian. Not like similar Github, it allows developers to store private repositories. I prefer using Bitbucket because it can keep my source code privately and it supports issue tracking, too.

SourceTree is an extremely strong git client, especially for MacOS users. SourceTree simplifies how you interact with your Git and Mercurial repositories so you can focus on coding. Visualize and manage your repositories through SourceTree's simple interface. It is simple for beginners, and powerful for experts. It also works well with Gitflow.

3.1.2. OSX, Xcode and Swift version

At the *present*, OSX Sierra 10.12.4 and Xcode 8.3 with Swift 3.0 are used to make the app. To produce an iOS application, OSX is the first requirement, a computer with Mac OSX or Hackintosh installed is accepted.

Xcode is a powerful tool to develop, test and simulate the application. But some feature like camera need a real device to build and test, iPhone 5S is a minimum requirement for this. The developer has to registered to Apple to start developing iOS apps, or you can use a provisioning profile with access granted from your company instead. Otherwise the app cannot be debugged or installed to a real device for security purpose.

This application *was* written on Swift 3.0 language. It may be changed in the feature following Apple's updates. Please update code manually or using Xcode built-in tool to convert to new style of coding.

3.1.3. Enable Cloud APIs

Before taking a beta development of this application, it has to have a valid key of Google Cloud API. Please contact Google to get a paid key, or use the following key: *AIzaSyAHGtaQHM3GCFNwBi6HdBOuzDow3f_N4IQ*

Note that this key does not exist forever, and NOT to be included in the source code for security purpose, since this report was published, the key was used to get a quick beta-test only. In the future from *now*, the key may be invalid.

The application also requires internet connection alive to work properly. Please turn on Wifi or 3G/4G connection before opening the app.

3.2. Deployment

Extract source code which is attached with this report to computer.

Open Terminal, go to the folder and install dependencies by typing this command:

```
$ pod install ⏎
```

After installing dependencies, open *.xcworkspace* file in Xcode.

Connect your device to the laptop and start building the app and run by hitting shortcut ⌘ + R or select menu *Product* → *Run*.

The app will be installed on the device but it will be blocked, unlock by go to Phone *Settings* → *General* → *Device management* → *Applications and Developers*, then trust the installed app.

Build again and the app runs with no doubt.

3.3. Testing

3.3.1. Unit testing

Connect your device to the laptop and start testing the app by hitting shortcut ⌘ + U or select menu *Product* → *Test*.

Wait the app while it is being built and start testing, the result will come soon after a several steps, please be patient.

3.3.2. Performance Test Result

The app has a quick response to user interactions. The occupied memory is about 20 to 30 mega-bytes and there is no memory leak problem.

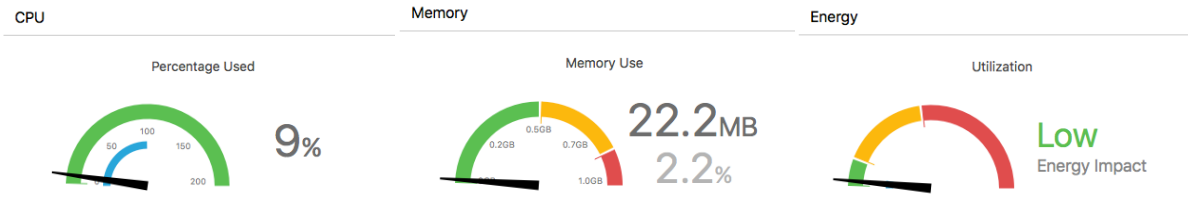


Figure 3.2: Performance debugging information

Online tasks that required internet connection base on connection quality. But it is acceptable for a regular connection.

3.4. Demonstration of main features

3.4.1. Tested Devices

This application *was* tested on these devices:

- *iPhone 5S – iOS 10.3*
- *iPhone 6 – iOS 10.3*
- *iPhone 6S – iOS 10.3*
- *iPad 4 – iOS 9.3*

3.4.2. Screen shoots

Some screenshots to demonstrate the app on iPhone.

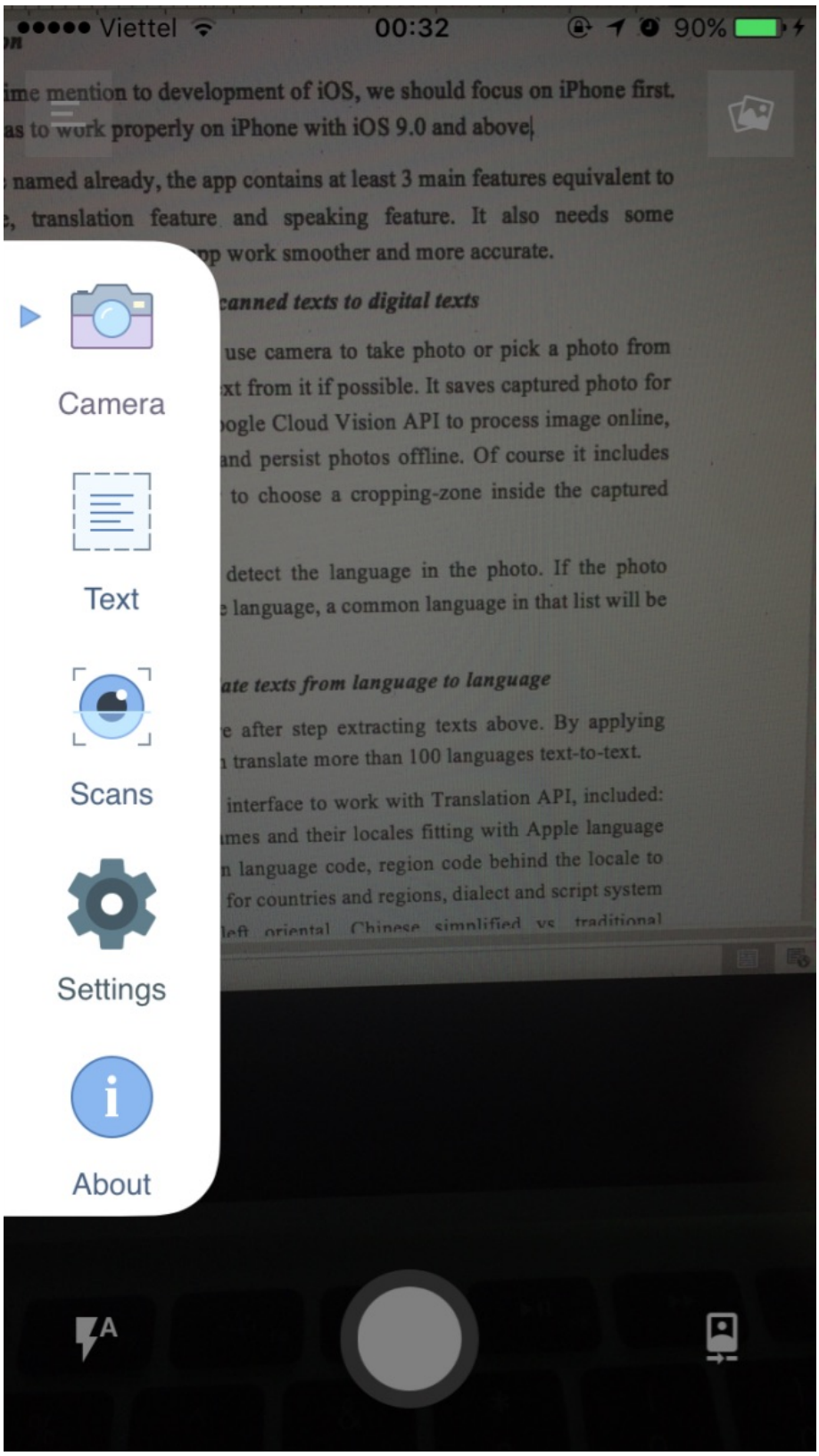


Figure 3.3: Screen shoot – Live camera view with Sidebar

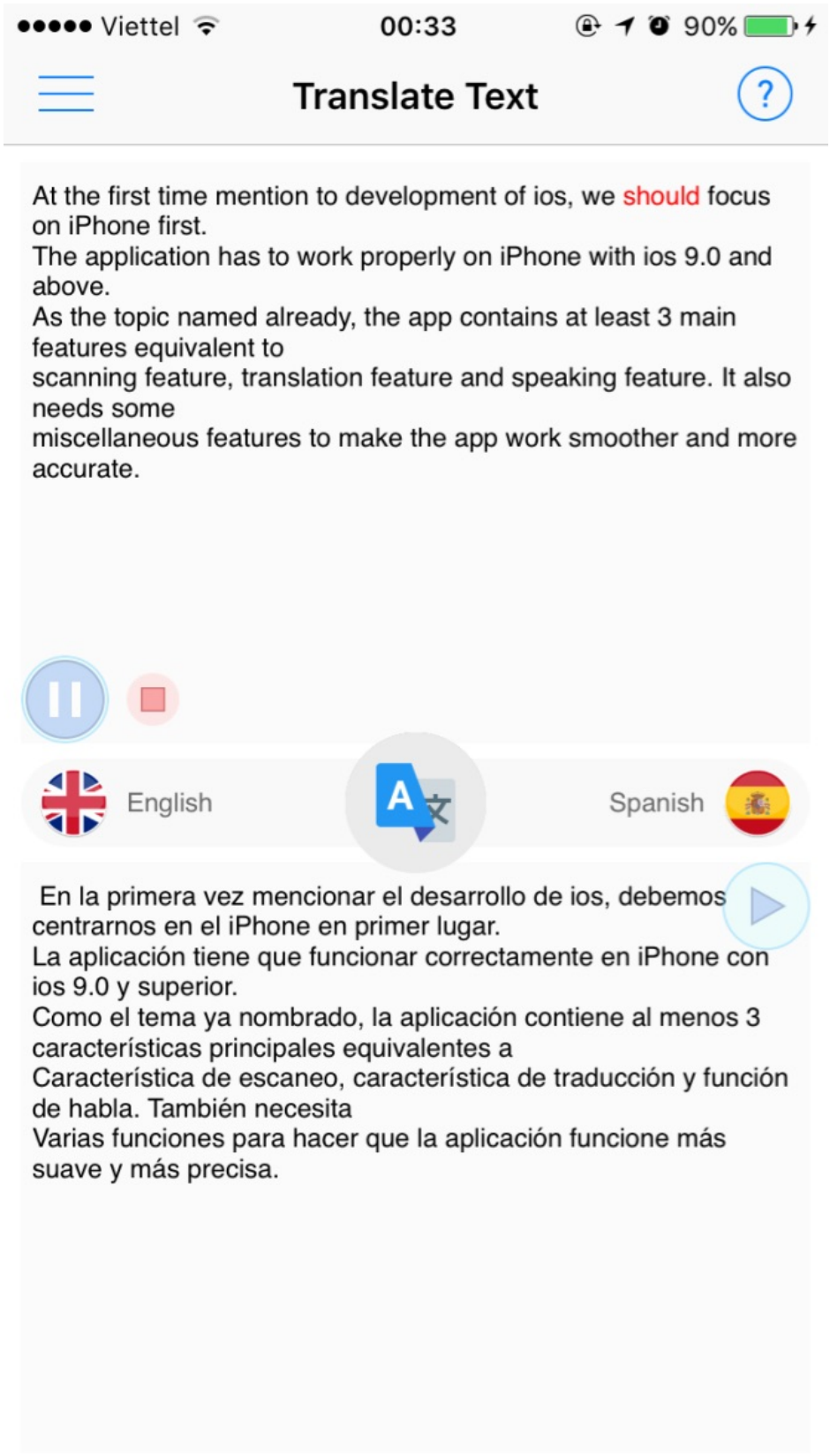


Figure 3.4: Screen shoot – Translation Text View and Synthesizing feature

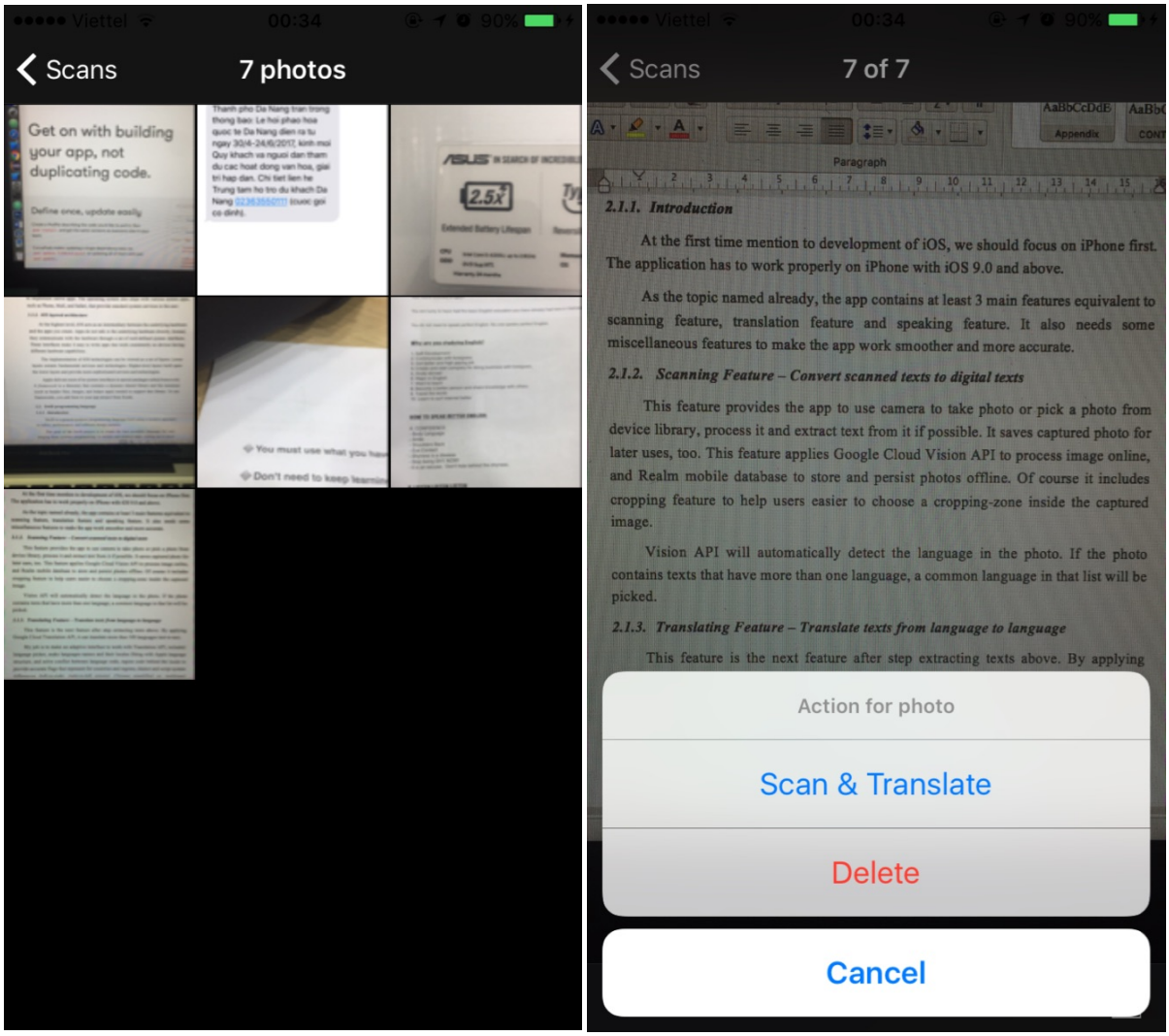
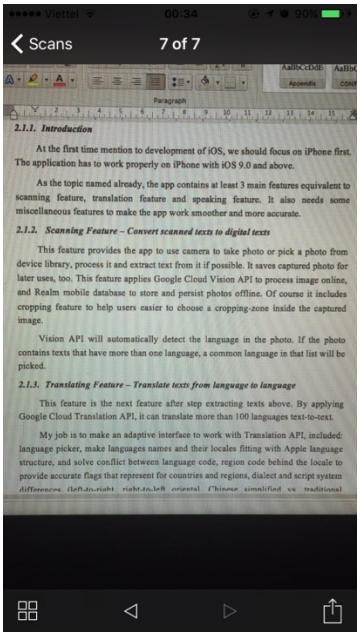


Figure 3.5: Screen shoot – Grid view and Scan view

With toolbar below:



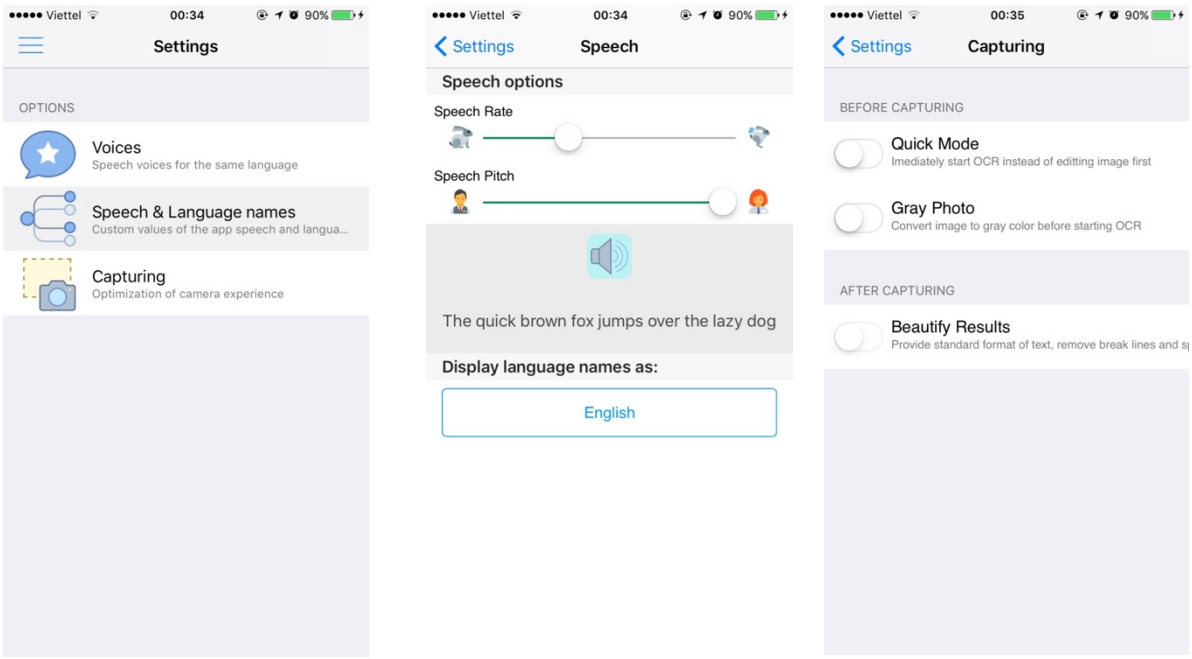


Figure 3.6: Screen shoot – Settings View

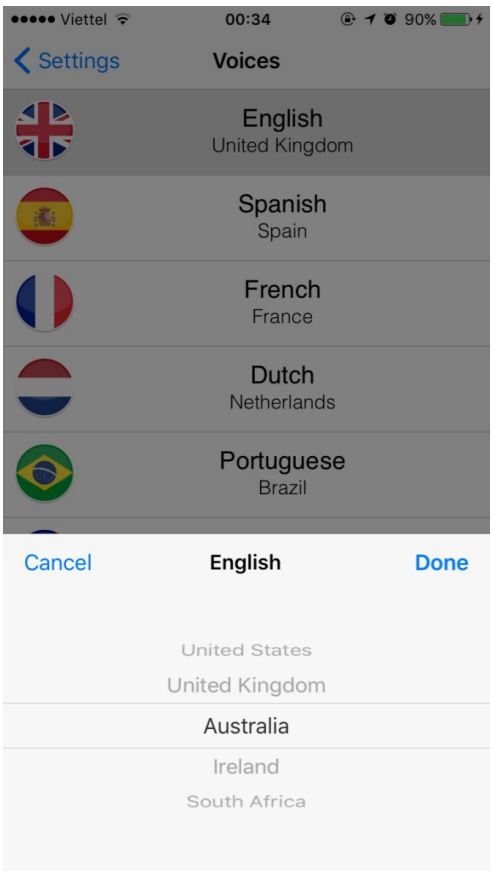


Figure 3.7: Screen shoot – Default Voice Settings

CONCLUSION

♦ Initial result

- *Theory aspect:*
  - iPhone Operating System and iOS development basis.
  - Networking in mobile development with HTTP and REST.
  - Software development: processes and methodologies.
  - The way to build an application from an idea.
- *Practical and applied aspects:*
  - Work with medium products of Google Cloud.
  - Lots of issues, bugs, exceptions and tricks while coding.
  - Apply open source libraries to a real project.
  - Beyond practical level and make a product as production level successfully.
- *Shortcomings:*
  - The app appearance is not united among screens in colors, sizes and designing languages.
  - Combination of open source libraries is not really stable in longtime.

♦ Future development

The application can be scaled in the future following these directions:

- Improve and optimize user experience and application stability.
- Allow user to edit and handle result deeper than now. Because this app just exploits some small parts of Google Cloud Vision and Translation APIs.
- Submit the app to the App Store, make it become real production for real users.

REFERENCES

[1] Apple Developer, "About The iOS Technologies," *Apple Inc.*, Sep. 17, 2014. [Online]. Available: https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html. [Accessed: Apr. 01, 2017]

[2-3] The Swift Team, "About Swift," *Apple Inc*. [Online]. Available: https://swift.org/about. [Accessed: Apr. 01, 2017]

[4] Google Cloud Platform, "Vision API - Image Content Analysis," *Google Inc.*, Jun. 1, 2016. [Online]. Available: https://cloud.google.com/vision. [Accessed: Apr. 01, 2017]

[5] Google Cloud Platform, "Cloud Translation API - Dynamic Translation," *Google Inc.,* Jun. 1, 2016. [Online]. Available: https://cloud.google.com/translate. [Accessed: Apr. 01, 2017]

[6] Bhavani, "AVSpeechSynthesizer: Supported languages for the voice," *Mallow Technologies*, Jan. 21, 2016. [Online]. Available: http://blog.mallow-tech.com/2016/01/avspeechsynthesizer. [Accessed: Apr. 01, 2017]

[7] Vincent Driessen, "Introducing GitFlow," *Nvie Blog*, Jan. 05, 2010. [Online]. Available: http://nvie.com/posts/a-successful-git-branching-model. [Accessed: May. 04, 2017]

APPENDIX

1. Locale syntax and Swift Locale class

Every operating system has its own method to control the languages represented by applications, iOS too. Technically developers can handle this manually but it is velvety if the app combine well with languages structure of the system. In iOS, there is a structure called *Locale* (for Swift) or *NSLocale* (for ObjectiveC).

Following the document from Apple, we get a quick glance of its syntax which the application applied in the table:

Locale ID Syntax	Examples	Description
Lang	*en*, *vi*, *zh*; *eng*, *vie*, *zho* or *chi*	An unspecified region where the language is used. This is the mandatory parameter of *Locale* class.
Lang_Region	en_US, vi_VN, zh_CN	The language used by and regional preference of the user. This is optional.
Lang-Script_Region	zh-Hans_CN, zh-Hant_TW	The script used by and regional preference of the user. This is optional.

Notes:

Lang: shorthand for *Language Code*, accepts forms of *ISO-639-1* (2-letter code) or *ISO-639-2* (3-letter code). If an *ISO-639-1* code is not available for a particular language, use the *ISO-639-2* code instead. Language code is a combination of lowercase letters (for examples: *en* for English, *vi* for Vietnamese, *zh, zho, chi* is for Chinese, etc.).

Region: shorthand for *Region Code*, but we may understand *region* as *country* although this may be not true in some cases. This code accepts *ISO-3166-1* form only. Region code is a combination of UPPERCASE letters (for examples: *US* and *USA* for United States of America, *VN* and *VNM* for Vietnam, *CN* is for China Mainland and *TW* is for China Taiwan, etc.).

Script: This is a part of *ISO-639-3* form and *BCP-47* form. Script code is Capitalized. It is used to clarify the type of script or character that can describe one or many languages in writing. Some common examples: *Arab* for Arabic script, *Cyrl* for Cyrillic script, *Latn* for Latin script, *Hans* for Simplified Chinese character, *Hant* for Traditional Chinese.

So by combining those codes we can get some more examples to understand despite they are loose sometime:

- *vie-Latn_VNM*: Vietnamese language spoken in Vietnam and written in Latin letters. This is the same as *vi_VN*.
- *cmn-Hans_HK*: Mandarin Chinese language spoken in Hong Kong and written in simplified Chinese character. This is the same for *zh-Hans_HK* and *zh_HK*.

- *zh-Hant_TW*: Mandarin Chinese language spoken in Taiwan and written in traditional Chinese character.
- *en_VN*: English language spoken in Vietnam! (and its script is Latin by default)

2. Adapting locale codes and language codes

After getting the understanding above, we know that there are many forms to represent a language in a correct way. But in this application I just apply common ISO forms only. But it always has exceptions in reality:

- Google Vision API and Translation API prefer using *BCP-47* and *ISO-639-3*.
- Apple's Synthesizing only works with combinations of *ISO-639-1* and *ISO-3166-1*. Apple's Locale prefers using *ISO-639-1, ISO-639-2, ISO-3166-1*. Apple also likes using '-' instead of '_'. We can use "en-US" replaced for "en_US".

And when we combine these 3 features together by default, the application **doesn't** work properly. So I have to find a way to adapt these misleading problems together and make it work well in different particular contexts.

We need a file to store all codes we need to implement, included Language codes and Locale codes, it is a properties file called ***Locales.plist***. By applying the flexibility of *Locale* class, we can extract partial codes and merge them later in our context. In the scope of this project, the problem includes:

- Different voices of common languages:
  - English: language code "*en*", locale IDs: en_US (America), en_GB (Britain), en_AU (Australia), en_IE (Ireland), en_ZA (South Africa).
  - Spanish: language code "*es*", locale IDs: es_ES (Spain), es_MX (Mexico).
  - French: language code "*fr*", locale IDs: fr_FR (France), fr_CA (Canada).
  - Dutch: language code "*nl*", locale IDs: nl_NL (Netherlands), nl_BE (Belgium).
  - Portuguese: language code "*pt*", locale IDs: pt_PT (Portugal), pt_BR (Brazil).
  - Chinese Mandarin: language code "*zh*", locale IDs: zh_CN (China Mainland), zh_TW (China Taiwan).
- Chinese scripts:
  - Chinese Mandarin for China Mainland: language code "*zh*", script code "*Hans*", region code "*CN*". Google Vision uses "zh-Hans_CN", Google Translate uses "zh_CN", Apple's Synthesizer use "zh_CN"
  - Chinese Mandarin for China Taiwan: language code "*zh*", script code "*Hant*", region code "*TW*". Google Vision uses "zh-Hant_TW", Google Translate uses "zh_TW", Apple's Synthesizer use "zh_TW"

- Chinese Cantonese for Hong Kong: language code "*zh*", script code "*Hant*", region code "*HK*". Google Vision uses "zh-Hant_HK", Google Translate uses "zh_TW" (*the same as Taiwan because they both use the same script, the meaning is closed to Mandarin although these 2 languages are different!*), Apple's Synthesizer uses "zh_HK" (*pronunciation is very different from Mandarin*)

Let's take a look in comparisons of Chinese difference:

- The same pronunciation but different script between China and Taiwan, Hongkong:

Script	Expression	Locale ID
Simplified	中国和台湾	zh, zh_CN, zh-Hans_CN, zh-Hans_HK
Traditional	中國和臺灣	zh_TW, zh-Hant_TW, zh_HK, zh-Hant_HK
Pronunciation	zhong^{1}guo^{2} he^{2} tai^{2}wan^{1}	zh-CN, zh-TW
Translation	China and Taiwan	-

- Equivalent script but different pronunciation between China, Taiwan and Hong Kong:

Region	Expression & Pronunciation	Locale ID
China, Taiwan	中国和台湾/中國和臺灣	zh-Hans_CN, zh-Hant_TW, zh_CN, zh_TW
	zhong^{1}guo^{2} he^{2} tai^{2}wan^{1}	
Hong Kong	中國和臺灣	zh-Hant_HK, zh_HK
	zung^{1}gwok^{3} wo^{4} toi^{4}waan^{1}	

We have to focus well on Chinese because there are more than one billion native speakers around the world. Chinese language is complicated so it has a lot of exceptions and misleading thing requires us to solve. And because Chinese is the top common language and this application cannot avoid it.

To avoid this problem easier, it is necessary to create a *LanguageHelper* model. *LanguageHelper* is a utility static class made by me to work with *Locales.plist* file, convert strict codes into language names, solve language conflicts, produce language array for the picker, generate flag name to assign language image flag to a language, etc.

3. Solving exception codes

A language in the app which can be synthesized always requires a region flag for it despite it supports synthesizing or not. But it just can show one flag at a moment. This can cause dizzy things if we have many voices / dialect in a language like English; which region flag should we

use? America flag or Britain flag or United Kingdom flag or something else? And the app contains more than 100 languages, just a few of them has one-one relationship between language and region, like Vietnamese is official in Vietnam only, that means we get only "*vi_VN*" code, not "*vi_US*" although this is a right format code.

But there are some languages are not belonged to any region! They have lots of speakers but they are not official in any country, they still have flags. This list includes: *Latin, Esperanto* (artifact language for a `common world`)*, Swahili* (Latin Romanized Arabic version for Africa Community)*, Hmong* (native in Vietnam and China but created in America and it formed by Latin letters).

Some languages of some locations have specific flags despite they are belonged to a country. This happens often in these languages: *Yiddish*, *West Frisian*, *Corsican*, *Scotland*, *Hawaiian* and *Wales*.

Many languages apply a *common writing system* but the pronunciation, grammar, meaning may be different:

- Chinese Mandarin, Cantonese, Japanese use *Han system*;
- Arabic, Pashto, Persian, Punjabi, Hebrew, Sindhi, Urdu... use *Arabic system*;
- Russia, Mongolian, Ukrainian, Serbian, Tajik… use *Cyrillic system*.
- Hindi, Sindhi (Sindhi language uses 2 systems), Nepali use *Devanagari system*.
- *Latin system*: a lot.
- And other systems…

Some languages are formed in more than one script type, they produce the same meaning with different writing styles: *Hebrew* (Hebrew alphabet and Arabic alphabet), *Chinese Mandarin* (simplified and tradition characters), Punjabi (Arabic and Shahmukhi forms), etc.

Determining right system helps us choose correct fonts and text alignments. This is very important because if we produce wrong characters, user experience breaks and the app fails. Luckily iOS contains most of necessary fonts to display characters well, and it can automatically refer Unicode encoding characters to the correct font when needed. The second lucky is Google *often* prefers left-to-right (LTR) writing systems, if a language has LTR system, that system will be chosen first. The only thing is a font for *Amharic* language, then we just need focus on text alignment for some Arabic and Hebrew system languages.

4. Text alignment and single word structure

Most of languages apply Left-to-Right alignment, except those languages refer to *Arabic* and *Hebrew* prefer Right-to-Left (LTR) alignment: Arabic (*ar*), Farsi / Persian (*fa*), Urdu (*ur*), Yiddish (*yi*), Pashto (*ps*), Hebrew (*he, iw*), Sindhi (*sd*). Let's take Punjabi language for an example compared to English:

Language - Writing system	Alignment	Expression
English – Latin	LTR →	What is your name?
Punjabi – Shahmukhi	LTR →	ਤੁਹਾਡਾ ਨਾਮ ਕੀ ਹੈ?
Punjabi – Perso-Arabic	← RTL	تہاڈا نام کی ہے؟
		Tuhāḍā nāma kī hai?

As we can see above, Punjabi is spoken both in Pakistan (the writing system is Perso Arabic) and India (the writing system is Shahmukhi). Google prefers using LTR system in this case.

Word structure of a word is very important when synthesizing. A word consists of some letters. It is easy to separate each letter in Latin system but it may be complicated in some languages, the following table shows an example about this:

Language	Description	Note
English	book → b + o + o + k / buk → b + u + k /	The letters keep their original form
Arabic	كِتاب ← ب + ا + ت + كِ (book) / b + a + t + ki ← kitab /	Each letter changes itself when it isolates with the other, with a reflected direction.

Synthesizing texts needs to break words into syllables and this process is not easy. I applied Apple's synthesizer in this project, that means everything was done before, but understanding about it is needed for highlighting speaking texts.

5. Flag image names from locale codes

The app includes hundreds of flag images with different sizes. Each language requires at least one flag to describe itself. For examples:

- Britain English: named as "*United Kingdom*" with Locale code "en_UK".
- America English: named as "*United State of America*" with Locale code "en_US".

The name can be produced from *Locale* class as English locale, and *LanguageHelper* carries this responsibility.

The locales codes, which stored in ***Locales.plist*** file that contains mixed of language codes, language code with region codes. Those codes would contain only language code:

- Multi-voices languages: *en, es, fr, nl, pt, zh*.
- Non-region languages: *eo, hmn, la, yi, sw*.
- Specific locations in a country that have their own flags: *co, fy, gd*.

They will be process later to make sure that each of them has a flag in language pickers. For examples:

- English: a global spoken language, stored only “en” in *Locales.plist*, default region code in the app is “US” so by default its locale ID is “*en_US*”. When users select Britain English as a preferring choice, the locale ID becomes “*en_UK*”.
- Swahili: a common language for Africa community, that means this is a non-region language. But the community made a flag for it. Locale code stored only “sw” in *Locales.plist*, and the locale ID is still “sw” (the same as locale code)

6. List of dependencies

This app is impossible to be done without the help of many dependencies during the development. The famous framework of Cocoa helps us applying a lot dependency called “pod”. The following list is those pods which I applied in the app to save time and make the code clean as much as possible:

- **Alamofire** 4.3.0 - *MIT license*
  Alamofire Software Foundation - *http://alamofire.org*
  Alamofire is an HTTP networking library written in Swift.
- **NVActivityIndicatorView** 3.6.1 - *MIT license*
  Vinh Nguyen - *https://cocoapods.org/pods/NVActivityIndicatorView*
  NVActivityIndicatorView is a collection of awesome loading animations.
- **Material** 2.0 - *BSD license*
  CosmicMind, Inc. - *http://www.cosmicmind.com*
  Material is an animation and graphics framework for Material Design in Swift.
- **SwiftyJSON** 3.1.4 - *MIT license*
  Lingoer & Tangplin - *https://cocoapods.org/pods/SwiftyJSON*
  SwiftyJSON makes it easy to deal with JSON data in Swift.
- **ActionSheetPicker-3.0** 2.2.0 - *BSD license*
  Petr Korolev & Tim Cinel - *https://cocoapods.org/pods/ActionSheetPicker-3.0*
  Easily present an ActionSheet with a PickerView, allowing user to select from a number of immutable options.
- **MWPhotoBrowser** 2.1.2 - *MIT license*
  Michael Waterfall - *https://cocoapods.org/pods/MWPhotoBrowser*
  A simple iOS photo and video browser with optional grid view, captions and selections.
- **TOCropViewController** 2.0.12 - *MIT license*
  Tim Oliver - *https://cocoapods.org/pods/TOCropViewController*
  TOCropViewController is an open-source UIViewController subclass built to allow users to perform basic manipulation on UIImage objects; specifically cropping and some basic rotations.

- **GSMessages** 1.3.4 - *MIT license*
  GeSen - *https://cocoapods.org/pods/GSMessages*
  A simple style messages/notifications, in Swift.
- **RealmSwift** 2.7.0 - *Apache 2 license*
  Realm - *https://realm.io/docs/swift/latest/*
  Realm, a mobile database, enables its users to develop applications that are fast and craft amazing user experiences.
- **IQKeyboardManagerSwift** 4.0.9 - *MIT license*
  Iftekhar Qurashi - *https://cocoapods.org/pods/IQKeyboardManagerSwift*
  Codeless drop-in universal library allows to prevent issues of keyboard sliding up and cover UITextField/UITextView. Neither need to write any code nor any setup required and much more.

In the finale I want to say thanks to the open source community, each dependency is a reliable lesson for me to make app better. For newer versions of these pods, please go to the links above for more information.

[this page intentionally left blank]