Software Design Document

for

Iris: Email Reader Application

Version 1.04

Scott Arnette

Joseph LaCava

Derek Ouzia

Bryan Smith

University of Virginia’s College at Wise

Department of Mathematics and Computer Science

January 26th 2016

Table of Contents

Table of Contents ii

Revision History iii

1. Introduction 1

1.1 Purpose 1

1.2 Product Scope 1

1.3 Overview 1

1.4 References 1

2. System Overview 2

3. System Architecture 2

3.1 Architectural Design 2

3.2 Decomposition Description 3

3.3 Design Rationale 4

4. Data Design 4

4.1 Data Description 4

4.2 Data Flow 5

5. Component Design 6

6. Human Interface Design 7

6.1 Overview of User Interface 7

6.2 Screen Images 7

6.3 Screen Objects and Actions 9

Appendix A: Glossary 10

Appendix B: To Be Determined List 10

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Scott Arnette | 11/18/15 | Initial creation. | 1.0 |
| Scott Arnette | 11/18/15 | Finished first few sections | 1.01 |
| Bryan Smith | 12/2/15 | Add diagrams and filled in the rest of the sections. | 1.02 |
| Bryan Smith | 12/6/15 | Update Definitions | 1.03 |
| Bryan Smith | 1/26/16 | Update screenshots and other sections based on new features being implemented. | 1.04 |

# Introduction

## Purpose

The purpose of this Software Design Document is to define the architecture and system design of the Email Reader Application, also referred to as Iris in its entirety, showing and describing the subcomponents accurately.We shall refer to the project as Iris, or simply the application henceforth.

## Product Scope

Iris will provide users the ability to be read emails received to the Google Accounts that are logged on a device running the Android operating system (version 3.0 or newer). After being read an email, users will additionally be prompted whether to keep or delete the previously read message. Users will also be provided settings to change how often emails are synchronized to Iris, whether Iris should currently read messages to the user, and whether to keep the device’s screen on (not go into sleep or low-power mode) during application use.

## Overview

This document includes various UML diagrams to both define and describe how Iris shall function and how the components interconnect.

## References

Android Sync Framework - <http://developer.android.com/training/sync-adapters/index.html>

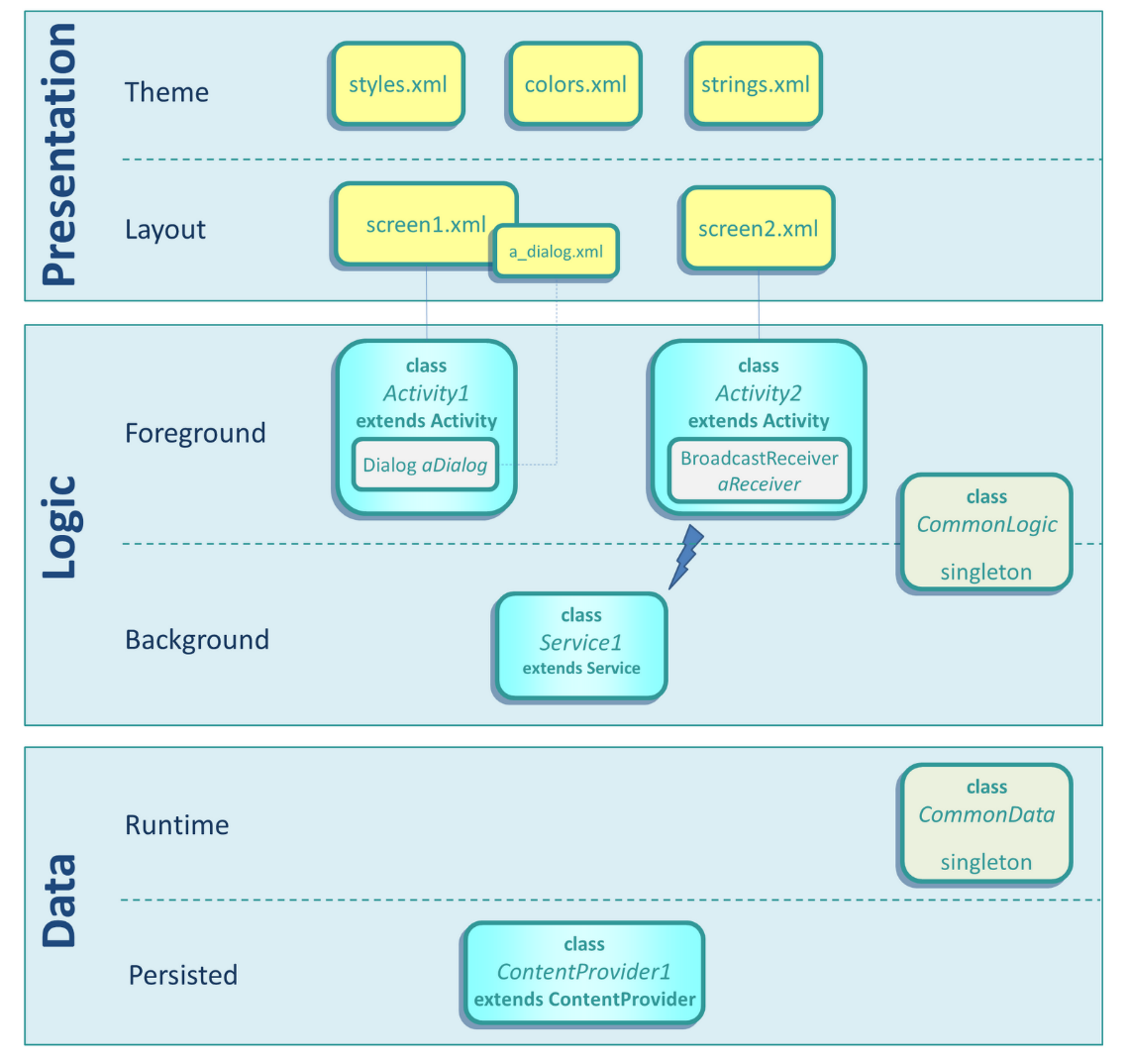
# System Overview

Iris, is a new project that allows users to be read incoming email messages via electronic synthesized voice and to decide whether to keep or delete messages. This is useful in situations where physical interaction with the device is limited or where focus is needed, such as when driving a vehicle.

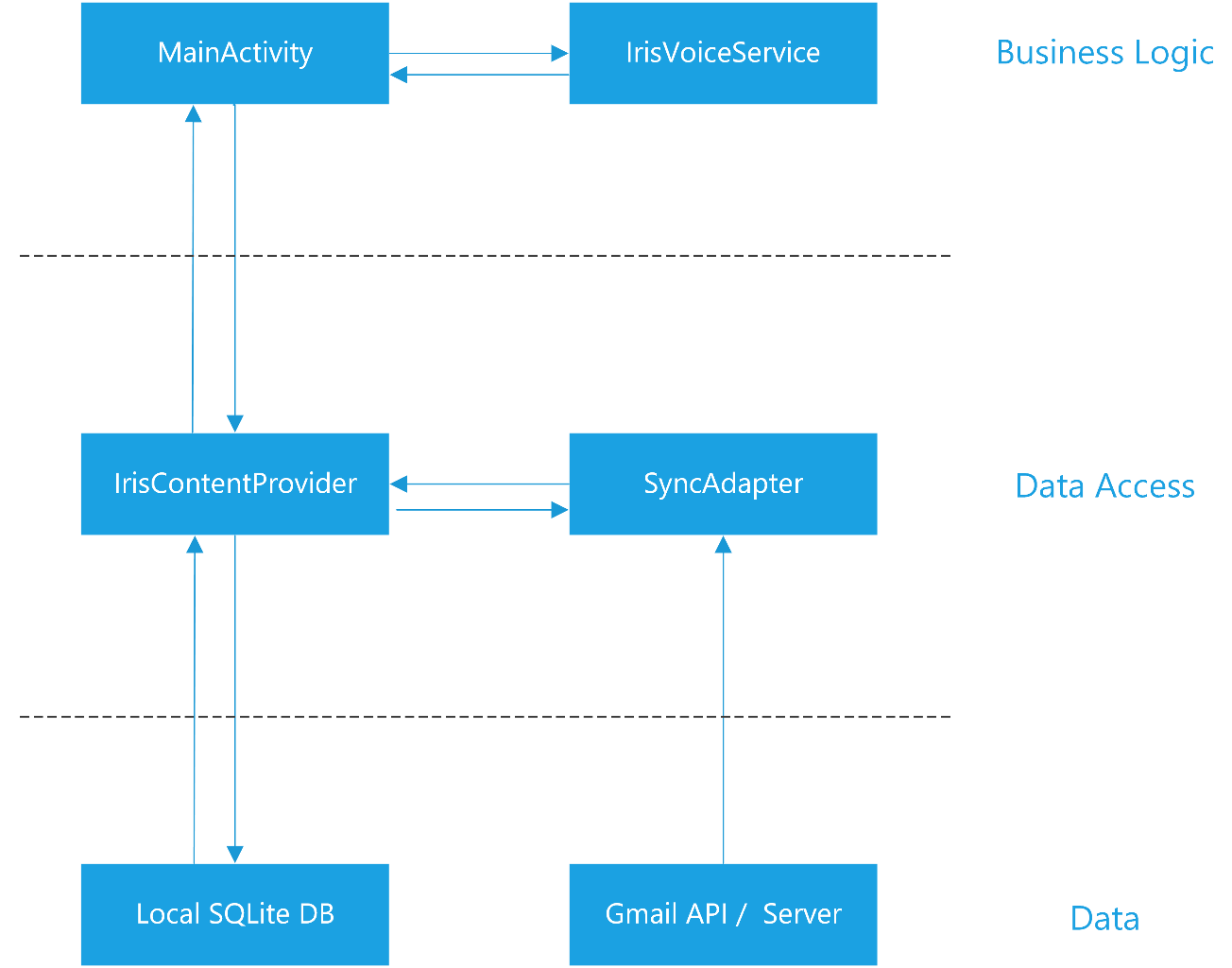
# System Architecture

## Architectural Design

Iris follows a standard Android application architecture. This type of architecture is similar to the MVC (Model View Controller) architecture. The diagram below shows a generic representation of Android applications architecture:



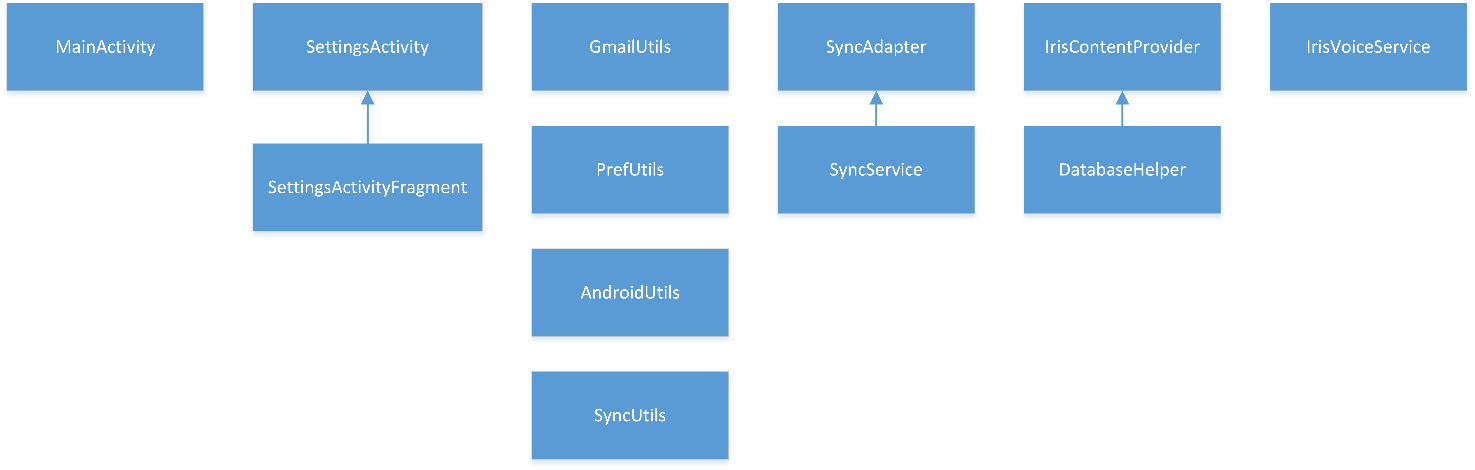
Iris is setup similarly and the diagram below represents its architecture:



The main activity will display the emails. The voice service will read out and display the overlay for new messages when it is turned on by the user. They sync framework manages data access. The content provider is the primary middle man for data access across the application. The sync adapter performs the network interactions with the Gmail server. The sync adapter utilizes the content provider to store the server retrieved messages into the local SQLite database. The main activity also uses the content provider to get the messages from the database for display.

## Decomposition Description

The decomposition of the design led to the below high level representation of Iris’s modules. This took into account the requirements of Android and the Sync Framework.



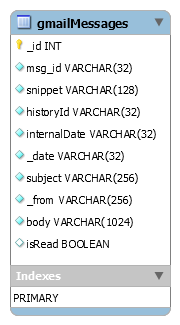
## Design Rationale

The architecture that was described in section 3.1 was chosen for multiple reasons. One of which is because the underlying architecture is shared with all Android applications and Iris was designed to stay aligned with Google’s design intentions. Secondly, using the sync framework made sense, because it is how Google syncs their data, and Iris is designed to be an ‘extension’ of their Gmail service. Integrating the sync in the same way as Google allowed for consistency and provided interaction and functionality that the user is accustomed to from other applications. Lastly, using the sync framework, the OS can handle and manage the syncing in order to best optimize battery life and performance.

# Data Design

## Data Description

Email message data is provided by the Gmail API in the form of JSON. It is parsed into GmailMessage objects by the Gmail API Library for Android. Iris also uses the data provided by the Gmail API to create MimeMessage objects that are part of the Javax.mail library. These objects already implement email header scrapping in order to get email subjects, from addresses, and body data. Iris stores only the message data that is needed by the application in a local SQLite database. Local SQLite ERD:



## Data Flow

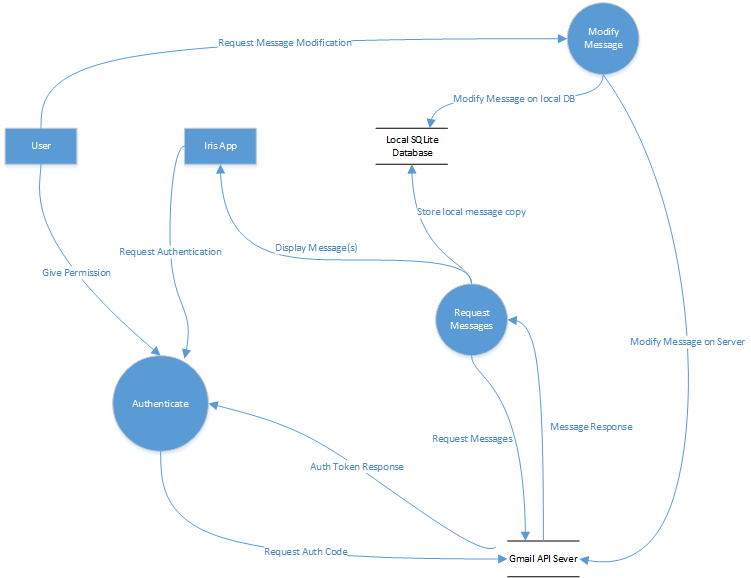
Below is a data flow diagram for Iris (Figure 4‑A):

Figure 4‑A

# Component Design

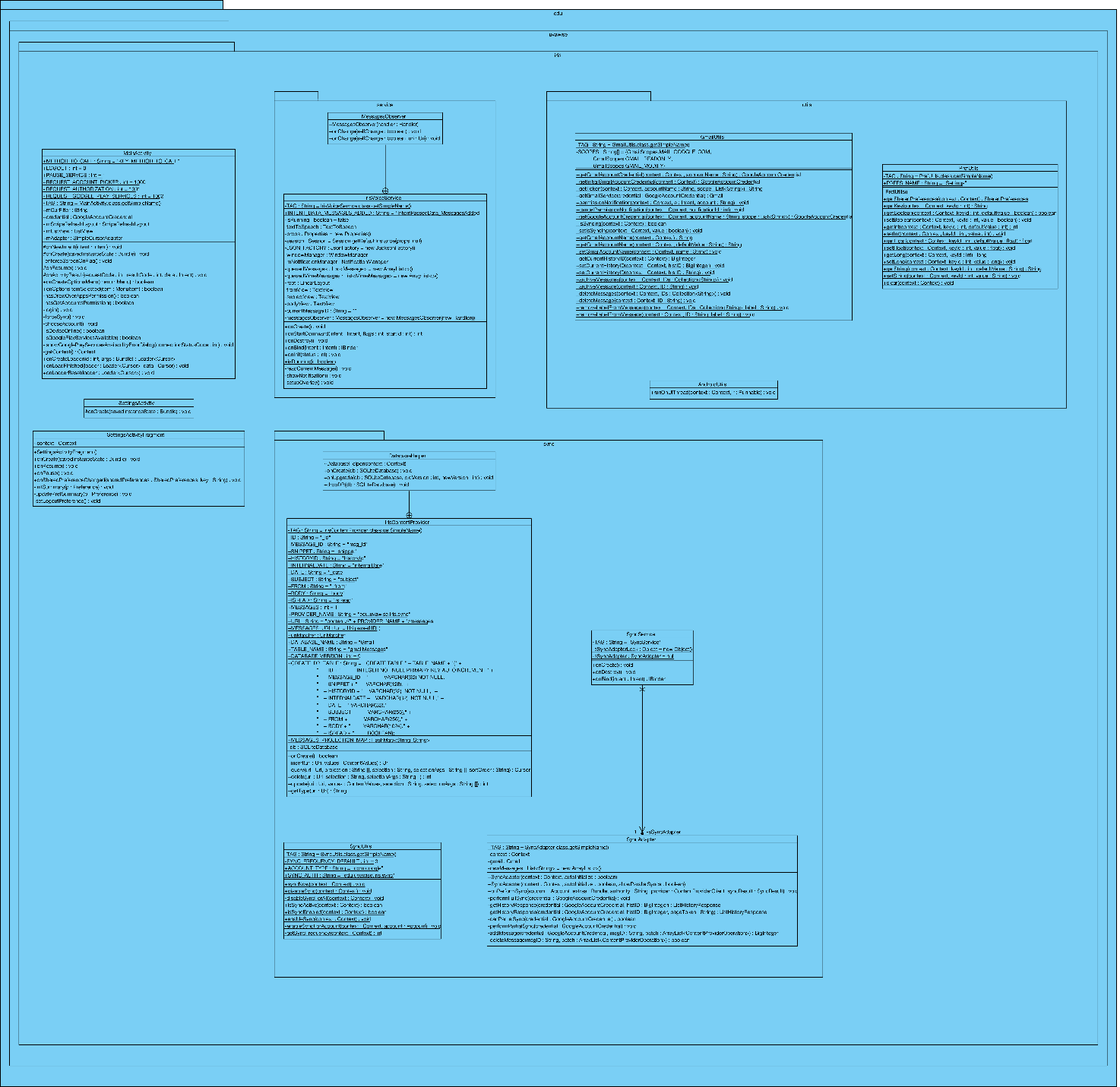
Below is a class diagram Iris (Figure 5‑A): 

Figure 5‑A – A larger resolution version is available.

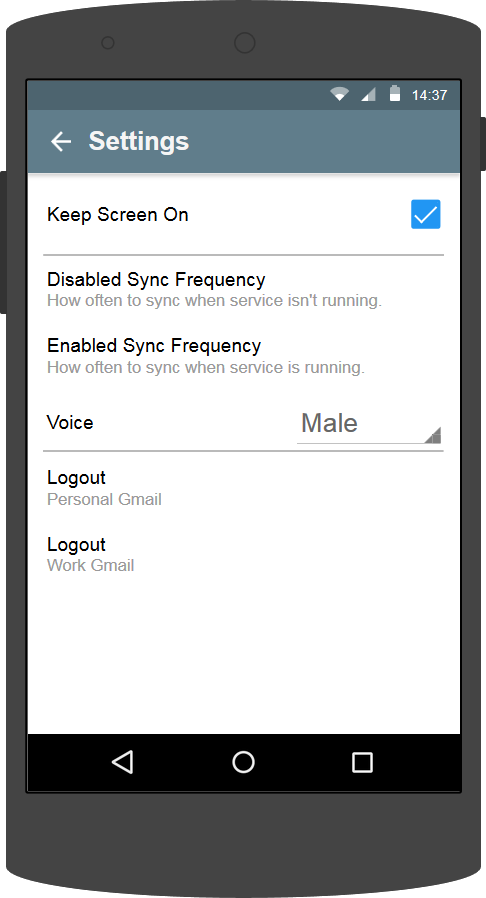
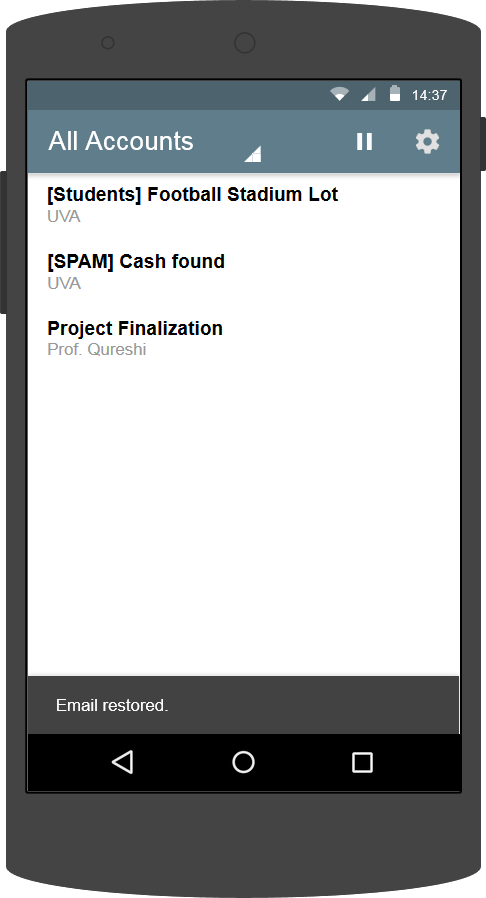
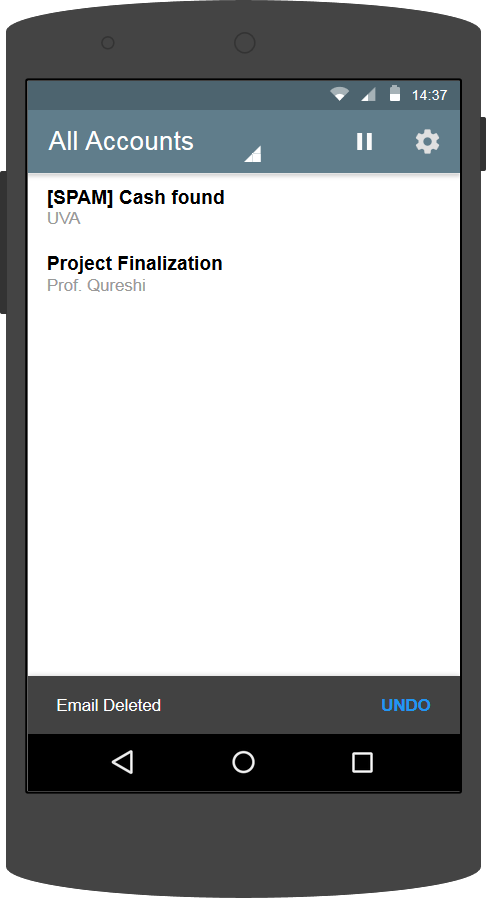
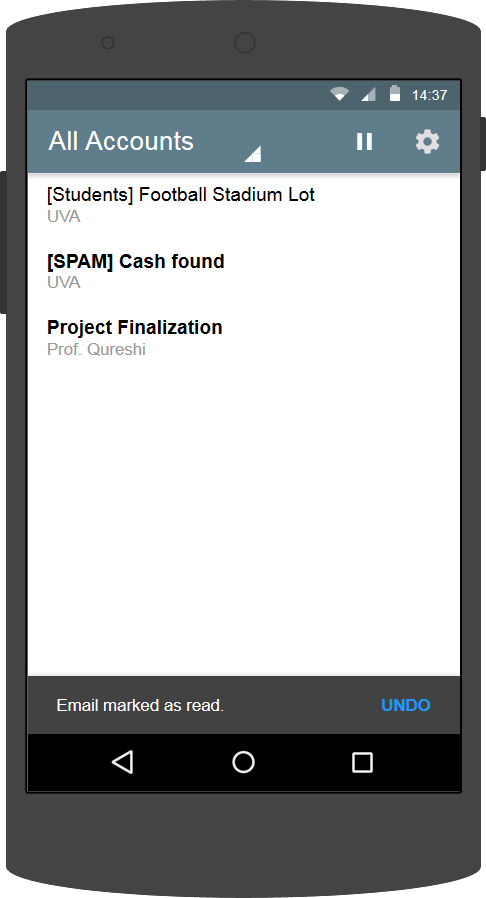
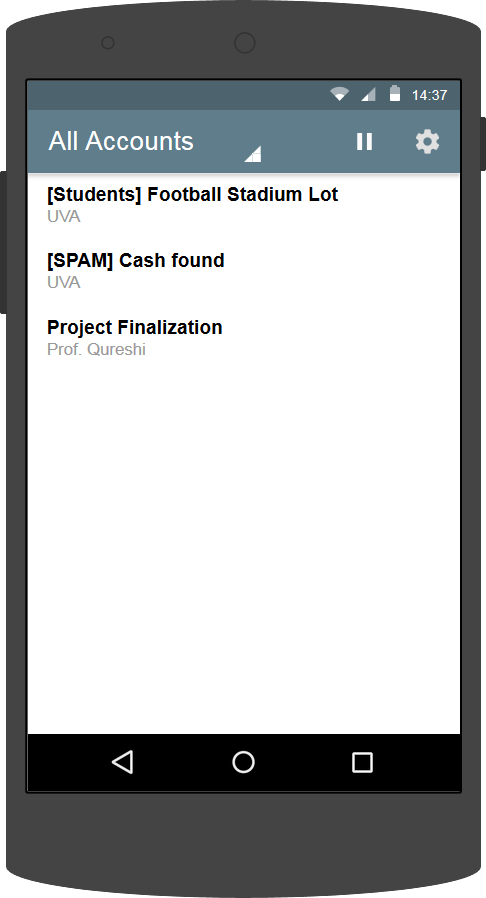
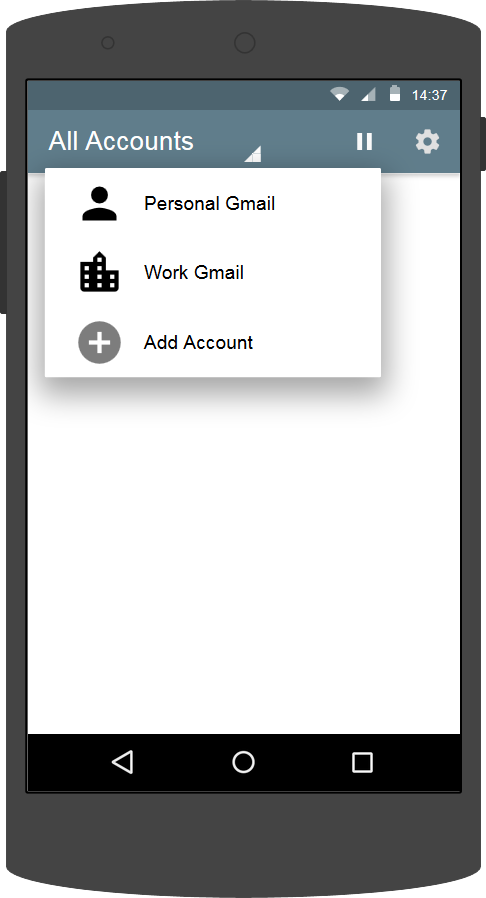
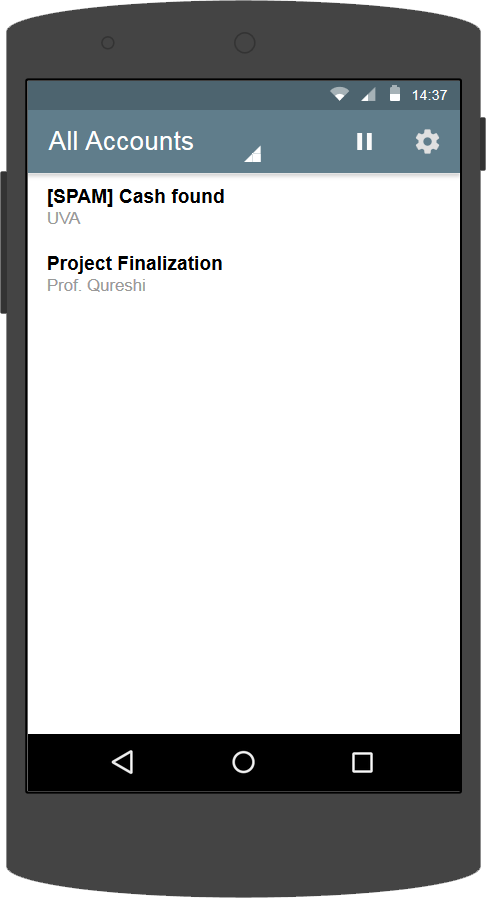
# Human Interface Design

## Overview of User Interface

Iris will follow a similar design patter found in common email client applications for Android. Iris also attempts to follow the Android Design Guidelines as closely as possible using the Material Design. Upon launching the application for the first time, users will be greeted with a series of dialogs that will help them get familiar with using Iris and understating how it can be controlled. It will also ask for them to specify which Google account they would like to use with Iris initially. Once selected, the application will sync the users Gmail messages and display them in a list on the screen. Accounts can be selected or added from a dropdown menu at the top of the application’s main screen. Each list item will show the message subject, the date it was received, and the address that sent the message. Pressing and holding on a message will initiate the selection mode. This will allow you to select one or more messages and then delete a message or archive it. After deleting or archiving a message will appear for a few seconds toward the bottom of the screen that states what has occurred and give an option to undo that action. Performing an action or pressing on an empty area will exit the selection mode. Pressing the three-dot or cog button will open the settings screen. This will allow the user to customize parts of the application, such as toggling an option that will keep the screen on while the app is active. Pressing the play button at the top of the app will start Iris’s background service. The background service is what will read the messages as the come in and display an overlay with the email data that is being read. In the overlay there are two options. Keep or delete. If keep is pressed, the message is marked as read. If delete is pressed, the message is moved into the trash bin in Gmail.

## Screen Images

Below (Figure 6‑A) are screens from our interactive prototype:



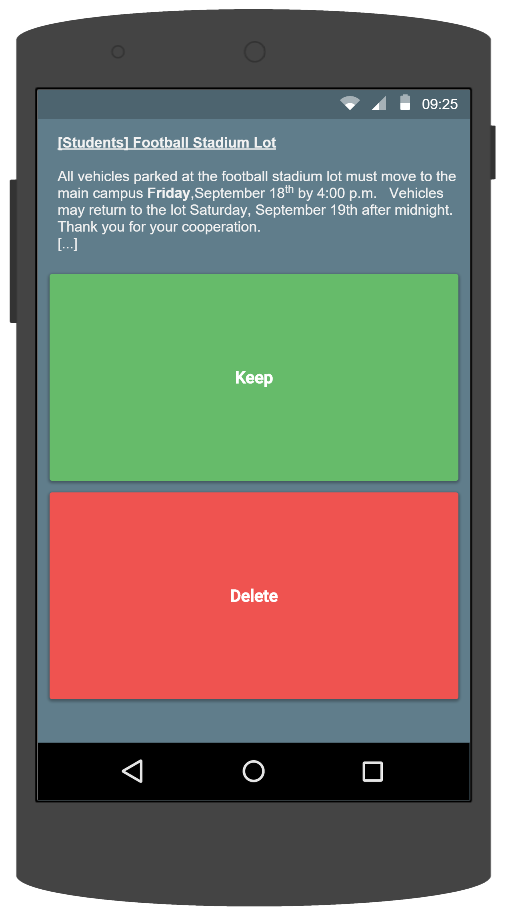


Figure 6‑A

## Screen Objects and Actions

Not applicable.

Appendix A: Glossary

* ADB – Android Debug Bridge. Versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device
* APK - Android application package is the package file format used by the Android operating system for distribution and installation of mobile apps and middleware.
* API – Application Programming Interface. A set of routines, protocols, and tools that govern a software specification.
* GUI – Graphical User Interface. An interface that allows users to interact with electronic devices through icons and visual indicators.
* Iris – The name of the Email Reader Application. The project that this SDD is for.
* OS – Operating System. The software that the product runs on.
* REST API - Representational state transfer application programming interface. A light weight web based API. The client does not need to know the structure of the API but the server provides the information the client needs to interface with the service.
* SDD – Software Design Document
* SRS – Software Requirements Specification.
* STP – Software Test Plan

Appendix B: To Be Determined List

None