LetsGo

Final Report

Version 1.0

May 3, 2020

–

Suchi Kapur

Nick Garrett

Khaled Khalil

Table of Contents

1. Introduction

1.1 Project Overview

LetsGo is a community-based application that embraces groups with similar qualities and attributes, and allows users to create communities and events for members that align themselves with said qualities, attributes and interests. Events that are created within communities of the application aim to further integrate users with one another, embracing their shared ideals and interests. LetsGo focuses on inclusion of users rather than exclusion; if a user does not fit into a community, there will be others they can join, or they can create their own. Everyone has a place within the LetsGo application, and no user will be left alone. LetsGo can be used by people ages 12 and older of various backgrounds, and is targeted towards people seeking social interaction based on similar interests. Specifically, LetsGo is primarily focused towards social groups that require more coordination, leadership, and structure within organizing events. The purpose of the LetsGo application is to connect users with similar interests, where users will be able to find their niche, or place, within the communities of the application.

This social media application is influenced by the main features of other social media platforms such as Facebook Invite, but will address concerns of similar preexisting applications regarding ease of use, as well as expanding on and creating new features that interest users. LetsGo combines social media features with event planning to create a user-friendly, exclusive application to promote community engagement and individual expression. The major aspects of the LetsGo application include user profiles, communities, and public and/or private events. The social media features of LetsGo include community formation and the ability for users to join, lead, and create groups based on a similar set of interests, or interest tags.

1.2 Definitions, Acronyms, and Abbreviations

Term	**Definition**
Stakeholder	Any individual that influences the development and progress of the project.
User	Any person that uses the application.
Community	Groups of people formed in the application with the same interest(s).
Community Leader	Has authority to remove members from their community, delete public community events and delegate privileges to community members.

Event	A planned and organized occasion created by a user.
Announcement	Only community leaders have access to announcements and are used to help clarify event plans or changes to the community alignment
RSVP	A request to accept or decline an invitation to an event.
Interest/Interest Tag	A tag that defines a user's individual expression and create their identity
Xamarin	An application platform for building iOS and Android compatible apps.
Visual Studio	Development platform.
Firebase	Database used for back-end. Used to store information pertaining to users and user accounts.

1.3 Purpose, Scope, and Overview of this Document

This document describes the system functions and functional requirements of the LetsGo application, the architectural patterns used in planning and implementing the system, and a reflection of the system's implementation. More specifically, this document will outline specific implementation technologies used, main tasks and processes, and a reflection of the implementation's consistency with the system's requirements. Lastly, the document will include a project management overview detailing tasks and work done by the LetsGo team members, as well as any problems that arose and how they were resolved.

2. System Functions

People ages 12 and older who are seeking social interaction based on similar interests are the target audience, and therefore users, of the LetsGo application. Users of the LetsGo application should have previous knowledge and experience of using mobile applications, using and having social media accounts, and navigating through various pages within a mobile application.

Once downloaded, users of the LetsGo application will be directed to a login page. First-time users will be able to create an account. Users that have accounts already will be able to login to the application and view their public feed (homepage) that displays location-based public events.

The application will allow users to navigate to numerous pages such as their own profiles, viewing and managing friends, events, and public feeds. Users will also be able to search for users, events, and communities based on interest tags.

The users will have the ability to join communities that share similar interests and values or create their own, join or create events for their communities, and connect with other users with similar interest tags through a search feature.

This section will discuss user interface, functional and non-functional quality requirements of the LetsGo application.

2.1 Product Functions

Function: Create Account

First-time users will have the ability to create an account for the LetsGo application. When a user creates an account, they will also create their user profiles.

Function: Login

When opening the app, users will be directed to a login page and prompted to login. The login page will contain fields for the user to enter their email address and password to login. The login page should include a “Forgot Password” feature for users to reset their passwords.

Function: Manage Account

Users will have the ability to manage their accounts by updating their profiles. User profiles include their full names, date of birth (cannot be changed), interests, location (city), and a profile picture. Users will also be able to delete their accounts.

Function: Manage Friends

Users will be able to view their friend list in a single page, and are given an option to “unfriend” if they choose to do so. Users will also be able to click any friend on this page to view their profiles. Users will be able to add friends by searching for interest tags and sending a request. Users will also be able to chat in a direct message with friends.

Function: Manage Communities

Users will be able to view communities that they are a member of, view specific community pages, create communities based on interest tags, leave communities, view community announcements, and join communities. Users will be able to participate in (group) chats with communities that they are members of.

Function: Manage Events

Under this section, users will be able to view events that they have accepted an RSVP to. Users will also be able to accept or decline any event invitations from their notifications page, create events, and view/participate in event group chats.

Function: View Feed

Users will be able to view public events in their specified location (city) on their public feed. The feed will appear on a single page, where the user can view and scroll through public events in their city.

Function: Search

Users will be able to conduct searches through a search bar. Searches are queried by interest tags associated with users, communities and events.

Function: Logout

The user will be able to log out of the LetsGo application.

2.2 User Interface Requirements

This section outlines the user interface requirements for the LetsGo application, consisting of the displays (views) for the user's screen, error or success messages, corresponding view files for input or output, and updates in the Firebase database.

2.2.1 Create Account Interface

The "Create Account" function allows first-time users to create an account for the system.

To create an account, users will be prompted to enter their name, date of birth, email address, password, and toggle a switch to indicate their account as public or private.

After a user creates their account, they will be able to log into the LetsGo application for use, as well as update their profile to include additional information such as their location, interests, and profile picture.

Source of Input OR Destination of Output	CreateAccountPage.xaml CreateAccountPage.xaml.cs CreateAccountController.cs
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application

Report Layouts	New user accounts will be added to the Firebase database. After creating an account, users will be redirected to their user profiles (see Section 3.1.4 for managing accounts).
Menu Structures	The "Create Account" option will appear on the login screen.
Error/Other Messages	1. You have successfully created an account! Please update your profile. 2. Account creation was unsuccessful. Please try again.
Function Keys	"Create Account" button.

2.2.2 Login Interface

The "Login" function of the LetsGo application allows users to login to the system.

The login page will display the LetsGo logo, and will contain fields for the user to enter their email address, password, and a login button.

Source of Input OR Destination of Output	LoginPage.xaml LoginPage.xaml.cs LoginController.cs FirebaseDB.cs → Firebase API
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	None.
Menu Structures	Users will be able to login upon opening the application on the login screen.
Error/Other Messages	1. Successful login. 2. Invalid login credentials. Please try again.
Function Keys	"Login" button

2.2.3 Forgot Password? Interface

The "Forgot Password?" function of the LetsGo application sends the user an email with a temporary password for their account, which they can change later upon successful login.

The "Forgot Password?" button will be on the login screen.

Source of Input OR Destination of Output	ForgotPasswordPage.xaml ForgotPasswordPage.xaml.cs

	ForgotPasswordController.cs
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	User passwords will be updated in the Firebase database.
Menu Structures	The “Forgot Password?” function will appear on the login screen of the application.
Error/Other Messages	1. You have been sent a password recovery email.
Function Keys	“Forgot Password?” button/link

2.2.4 Manage Account Interface

The “Manage Account” function of the LetsGo application allows users to create or edit their profiles, or delete their accounts.

When updating profiles, users can update their full names, interests, location (city), “public” or “private” account status, and profile pictures. Users will not be able to change their date of birth.

Source of Input OR Destination of Output	ProfilePage.xaml ProfilePage.xaml.cs ProfileController.cs UpdateProfilePage.xaml UpdateProfilePage.xaml.cs UpdateProfileController.cs FirebaseDB.cs → Firebase API
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	User profile/account information will be updated in the Firebase database.
Menu Structures	The “Manage Account” function will appear on the user’s profile. The user’s profile will also be an icon on the navigation bar at the bottom of the application’s screen.
Error/Other Messages	1. Profile was updated! 2. Profile has not been updated. 3. Your account has been deleted.

Function Keys	1. “Update Profile” button 2. “Delete Account” button 3. “User Profile” icon

2.2.5 Manage Friends Interface

The “Manage Friends” function of the LetsGo application allows users to add friends, remove (unfriend) friends, chat with friends, and view their friends and friends’ profiles.

Source of Input OR Destination of Output	FriendsPage.xaml FriendPageController.cs ViewFriendProfile.xaml ViewFriendProfile.xaml.cs FriendProfileController.cs ViewPrivateProfile.xaml ViewPrivateProfile.xaml.cs PrivateProfileController.cs ViewPublicProfile.xaml ViewPublicProfile.xaml.cs PublicProfileController.cs ViewChatBetweenFriends.xaml ViewChatBetweenFriendsController.cs FriendChatPage.xaml FriendsChatController.cs Conversation.cs ChatMessage.cs FirebaseDB.cs → Firebase API
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	User friends will be added or removed from the database.
Menu Structures	The “Manage Friends” function will appear under a 3-lined menu icon.
Error/Other Messages	1. Users are friends! 2. Users are not friends. 3. User has been unfriended. 4. You are chatting with your friend.
Function Keys	1. Pressing on a friend’s profile will allow the user to view their friend’s profile (see Figure 3 for mockup). 2. “Unfriend” button 3. “Add Friend” button 4. “Chat with Friend” button

2.2.6 Manage Communities Interface

The “Manage Communities” function of the LetsGo application allows users to join communities, leave communities, view communities they are members of, and chat with their communities.

Users should also be able to leave communities and view other members of each community that they have membership to. Users will also be able to request to join communities that they are interested in.

Source of Input OR Destination of Output **Note:** The communities module is very large and consists of components for both community leaders and community members.	ViewCommunityAsLeader.xaml ViewCommunityAsLeader.xaml.cs ViewCommunityLeaderController.cs CommunityLeaderViewController.cs ViewCommunityAsMember.xaml ViewCommunityAsMember.xaml.cs ViewCommunityMemberController.cs CommunityMemberViewController.cs ViewCommunityChat.xaml ViewCommunityChatController.cs CommunityChatController.cs CommunityAnnouncementsController.cs LeaderAnnouncementsController.cs ViewCommunityEvents.xaml ViewCommunityEventsController.cs ViewCommunityMembersList.xaml LeaderMembersListController.cs CommunityMembersListController.cs ViewPublicCommunity.xaml PublicCommunityController.cs ViewPrivateCommunity.xaml PrivateCommunityController.cs UpdateCommunity.xaml InviteCommunityMembers.xaml InviteCommunityMembersController.cs CommunitiesPage.xaml CommunityPageController.cs CreateCommunity.xaml CreateCommunity.xaml.cs CreateCommunityController..cs CommunitiesChatPage.xaml AppointNewLeader.xaml AppointNewLeaderController.cs CommunityProfile.cs

	Announcement.cs Conversation.cs ChatMessage.cs FirebaseDB.cs → Firebase API
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	Users will be added or removed from communities in the Firebase database.
Menu Structures	The "Manage Communities" function will appear under a 3-lined menu icon.
Error/Other Messages	1. You have joined a new community. 2. You have not joined a new community. 3. You have left a community. 4. You are chatting with your community.
Function Keys	1. If the user selects a community from their "Communities" page, they shall be redirected to the community page. 2. "Join Community" button 3. "Leave Community" button 4. "Chat with Community" button

2.2.7 Manage Events Interface

The "Manage Events" function of the LetsGo application allows users to RSVP to events, create events, view events that they have accepted invitations to, and chat within event pages.

Event owners will be able to invite friends/users to events, and shall be able to edit or update event information, or delete events. Events that are deleted will be removed from the database.

Source of Input OR Destination of Output **Note:** The events module is very large and contains components for both event owners and event members.	EventsPage.xaml ManageEvents.xaml.cs EventsPageController.cs ViewEventAsOwner.xaml ViewEventAsOwner.xaml.cs EventOwnerViewController.cs ViewEventAsMember.xaml ViewEventAsMember.xaml.cs EventMemberViewController.cs ViewEventMembersList.xaml EventMembersListController.cs ViewPublicEvent.xaml PublicEventController.cs

	ViewPrivateEvent.xaml PrivateEventController.cs UpdateEvent.xaml UpdateEventController.cs InviteEventMembers.xaml CreateEvent.xaml CreateEvent.xaml.cs CreateEventController.cs ViewEventChat.xaml EventsChatController.cs ViewEventChatController.cs EventProfile.cs Conversation.cs ChatMessage.cs FirebaseDB.cs → Firebase API
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	Users will be added or removed from events in the Firebase database.
Menu Structures	The “Manage Events” function will appear as a button on the user’s screen.
Error/Other Messages	1. You have created a new event. 2. Event creation unsuccessful. 3. You have deleted an event. 4. You have successfully updated an event. 5. Event update unsuccessful. 6. You have submitted an RSVP to an event. 7. RSVP submission unsuccessful. 8. You are chatting with event-goers of this event.
Function Keys	1. “Create Event” button (see Figure 5 for event creation mockup) 2. “Delete Event” button 3. “Update Event” button 4. “Accept Invitation” button 5. “Decline Invitation” button 6. “Chat with Event-Goers” button

2.2.8 View Feed Interface

The “View Feed” function allows users to view public events in their location on a single, continuous-scroll page. Events are displayed with the name of the event, a description of the event, and an image.

Source of Input OR Destination of Output	FeedPage.xaml FeedPage.xaml.cs FeedController.cs FirebaseDB.cs → Firebase API
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	Events for feed-viewing will be retrieved from the Firebase database for this function. The user’s feed will appear in a single, continuous-scroll page.
Menu Structures	The “View Feed” function will be the homepage of the app. The “View Feed” function will also appear on the navigation bar at the bottom of the application screen.
Error/Other Messages	1. You are viewing events in your location. 2. Error loading feed. 3. There are no events in your location.
Function Keys	“Feed” icon.

2.2.9 Search Interface

The “Search” function allows users to search for public users, events, and communities. Searches are conducted by interest tag.

Source of Input OR Destination of Output	SocialPage.xaml SocialPage.xaml.cs SocialController.cs FirebaseDB.cs → Firebase API
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	The “Search” function retrieves information from the Firebase database with matching interest tags, or email.
Menu Structures	The “Search” function will be at the top of the application feed, appearing as a search bar.

Error/Other Messages	1. Results matching your search: 2. No results found.
Function Keys	"Search" bar and button.

2.2.10 Logout Interface

The "Logout" function allows a user to logout of the system. The logout button will appear on the user's profile.

Once logged out, the user will be redirected to the login screen/page.

Source of Input OR Destination of Output	ProfilePage.xaml ProfilePage.xaml.cs ProfileController.cs FirebaseDB.cs → Firebase API
Required Screen Formats/Organization	Android Mobile Application iOS Mobile Application
Report Layouts	The user will be redirected to the login page of the application upon successful logout.
Menu Structures	The "Logout" function will appear on the user's profile.
Error/Other Messages	1. You have logged out.
Function Keys	"Logout" button.

2.3 Functional Requirements

Functional requirements of the LetsGo application are detailed below and organized by feature. Requirements are prioritized as follows:

- Priority 1 – The requirement is a "must have" as outlined by policy/law
- Priority 2 – The requirement is needed for improved processing, and the fulfillment of the requirement will create immediate benefits
- Priority 3 – The requirement is a "nice to have" which may include new functionality

2.3.1 Create Account Requirements

Req. No.	Requirement	Priority	Comments

FR_CR_1	The system shall allow users to enter their first name, last name, date of birth, email address, password, and a public/private account toggle in a form to create their account. (This information will be used to create a new user in the database).	1	
FR_CR_2	The system shall perform validity checks to ensure that email addresses provided by the user are not in use. If the email address is already in use, the user will be prompted to enter a different email or login with an existing account.	1	
FR_CR_3	The system shall validate that the user age is at least 12 years old. If the user does not meet the age requirement, they will not be able to create an account.	1	
FR_CR_4	The system shall authenticate that user-provided passwords meet specified password requirements. (See Section 3.3.1 for password requirement details).	1	
FR_CR_5	The system shall allow the user to login to their new account after account creation.	1	

2.3.2 Login Requirements

Req. No.	Requirement	Priority	Comments
FR_LI_1	The system shall present/call a login form for the user to enter login credentials.	1	
FR_LI_2	The system shall authenticate user login credentials. If user login credentials are invalid, the system shall prompt the user to retry, recover their password (“Forgot Password?” link), or create a new account.	1	See Section 4.3 for changes made during implementation.
FR_LI_3	The system shall provide the user with a “Forgot Password?” option.	2	
FR_LI_4	The system shall redirect the user to the LetsGo homepage after successful login.	1	

2.3.3 Password Reset Requirements

Req. No.	Requirement	Priority	Comments
FR_FP_1	The system shall allow the user to recover/reset their password. The user will receive an email with a link to reset their password.	1	See Section 4.3 for changes made during implementation.
FR_FP_2	To reset their password, the system shall allow the user to enter their new password and confirm their new password. The user's new password must meet password requirements (See Section 3.3.1 for password requirement details).	1	
FR_FP_3	The system shall allow the user to login with their new password after a successful password reset. The new password will be updated in the database.	1	

2.3.4 Manage Account Requirements

Req. No.	Requirement	Priority	Comments
FR_MA_1	The system shall allow the user to edit/update their user profile (full name, interests, location/city, profile picture) through an update profile form. All updated information in a user's profile will be updated in the database.	1	
FR_MA_2	The system shall allow the user to delete their account. The corresponding account information will be removed from the database.	1	

2.3.5 Manage Friends Requirements

Req. No.	Requirement	Priority	Comments
FR_MF_1	The system shall display the user's friends in a single page when the user clicks on their "Friends" tab/icon.	1	
FR_MF_2	The system shall give the users the option to "unfriend" any friends by clicking a button/icon. Users that have	1	

	been unfriended will be reflected and updated in the database.		
FR_MF_3	The system shall allow users to click on any friend on the “Friends” page to view the desired user’s profile.	1	
FR_MF_4	The system shall allow users to add friends by sending a “friend request.” Friend requests can be accepted or denied by the receiving user. Accepted friend requests will reflect both users having a new friend in the database.	1	
FR_MF_5	The system shall allow the user to chat with a friend. The user will click on the desired friend to chat with, and the system will create the chat connection for direct messaging to occur.	2	See Section 4.3 for changes made during implementation.

2.3.6 Manage Communities Requirements

Req. No.	Requirement	Priority	Comments
FR_MC_1	The system shall allow the user to view communities that they are members of through their “Communities” page.	1	
FR_MC_2	The system shall redirect the user to the community entrance that displays the identity of the specified community, if the user clicks on a community on their “Communities” page.	1	See Section 4.3 for any changes made during implementation.
FR_MC_3	The system shall redirect the user to the community page that shows community events and community chats if the user selects the “Enter Community” button.	1	See Section 4.3 for any changes made during implementation.
FR_MC_4	The system shall allow users to create communities by using a form with fields of community name, interest tags, and community owner/leader (which is the user creating the community). The new community will be created in the database, with the owner as a member.	1	
FR_MC_5	The system shall allow users to join communities. Users will be able to immediately join communities that are specified as “public,” or send a request to those	1	

	that are listed as “private.” The community leader must accept requests to join. New communities that a user becomes a member of are updated and reflected in the database.		
FR_MC_6	The system shall allow users to leave communities by selecting a “Leave Community” button/option. When the user leaves a community, it will be updated in the database.	1	
FR_MC_7	The system shall update the community leader to be the member with the longest membership if the community leader leaves the community.	1	See Section 4.3 for any changes made during implementation.
FR_MC_8	The system shall allow community leaders to accept or deny user requests to join their community. Accepted members will be updated in the database.	1	
FR_MC_9	The system shall allow community members to participate in community group chats.	2	

2.3.7 Manage Events Requirements

Req. No.	Requirement	Priority	Comments
FR_ME_1	The system shall allow users to view events that they have RSVP’d to on their “Events” page. If a user clicks on a certain event, the system shall redirect the user to view the event’s full details.	1	
FR_ME_2	The system shall allow users to create events by using and filling a form containing event name, event date, event start and end time, event location, an image, the option of the event being public or private, and the option to invite friends. New events will be created in the database.	1	
FR_ME_3	The system shall allow users to RSVP to events. If a user accepts an invitation, the event will be updated with a new event-goer in the database.	1	
FR_ME_4	The system shall allow event owners/creators to update	1	

	name, date, time, location, and public/private status for events they have created. All updated information will be updated in the database.		
FR_ME_5	The system shall allow users to participate in event chats for desired events. The system will create the connection for the chat.	2	

2.3.8 View Feed Requirements

Req. No.	Requirement	Priority	Comments
FR_VF_1	The system shall display public events within a user's location (city) on their "Feed" page that the user shall be able to scroll through.	1	
FR_VF_2	The system shall refresh the user's feed if the user swipes down and refreshes the page.	3	

2.3.9 Search Requirements

Req. No.	Requirement	Priority	Comments
FR_SE_1	The system shall query the database when the user conducts a search. Search items that appear include public user profiles, public events, and public communities.	1	See Section 4.3 for any changes made during implementation.
FR_SE_2	The system shall allow users to select a desired search result to view the community page, event page, or public user profile.	1	

2.3.10 Logout Requirements

Req. No.	Requirement	Priority	Comments

FR_LO_1	The system shall allow users to log out of the system. Users will logout by selecting the “Logout” button.	1	
FR_LO_2	The system shall redirect the user to the login page upon successful logout.	1	
FR_LO_3	The system shall end the current application session upon unsuccessful logout, if the system crashes.	1	

2.3.11 Additional Requirements

Req. No.	Requirement	Priority	Comments
FR_AR_1	The system shall log a user out after 10 minutes of inactivity.	3	Not completed due to issues with Firebase Authentication.

2.4 Quality Requirement: Security

Requirements specified under this section protect data of the LetsGo application from unauthorized access and external threats.

2.4.1 Password Requirements

User passwords must meet the following requirements:

- Password must be at least 8 characters in length.
- Password must contain one number.
- Password must contain one uppercase letter.
- Password must contain one special character

Related Functional Requirements: FR_CR_4, FR_FP_2

2.4.2 Password Encryption

User passwords should be encrypted to prevent security issues, hacking of accounts, and potential external threats to the system’s integrity.

2.5 Quality Requirement: Reliability

LetsGo should operate on a consistent basis (everyday) throughout the year. In optimal cases, the application should operate 24 hours a day, 7 days a week. The system will maintain data integrity at all times.

2.6 Quality Requirement: Availability

The LetsGo application will be available at all times, except in the cases of database or network/internet connection failure.

In the event of system, network, or database failure, availability is estimated to be zero until updates occur to ensure a functioning and secure application.

3. Architecture

The LetsGo application was designed to utilize the Model-View-Controller architecture pattern, shown below in figure 1. The Model-View-Controller pattern utilizes three components: a model, a view, and a controller. The model contains information that is used by the application. The view is the screen presentation of the information (i.e. the user's smartphone screen). Lastly, the controller defines the way that the user interface reacts to user input and maintains the view-model consistency. The Model-View-Controller architecture pattern suits the application's context of being a smartphone application with user interaction.

The LetsGo application behaves according to the user's interaction with the application. User events are handled by the controller, which communicates with and updates the model, if necessary, as well as the corresponding view. The model for this application is the Firebase Database, where application-specific information is stored and available for retrieval by the user. The controller behaves as an input handler to the user's gestures with their screen through client-side code, while the view consists of multiple .xaml files. This high level architecture design is shown below.

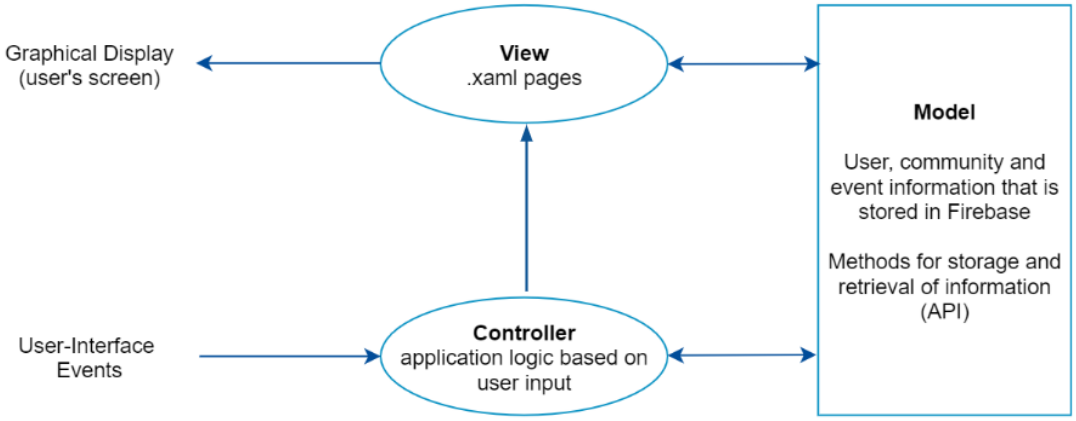

Figure 1. High-Level Architecture of LetsGo MVC Pattern.

3.1 Model Module

Purpose

The purpose of this module is to encapsulate application-specific data. More specifically, the model module acts as a central area for LetsGo application data to be stored and retrieved from.

Rationale

This module allows users to access and retrieve information relative to their interactions with the application that are handled by the controller and accessed by the model. The model will store information related to user accounts, communities, events, user friends, and friend requests. Additionally, all user authentication and profile information will be stored in the model.

High-Level Module Design

The model module is broken down further into two lower-level modules, as shown in Figure 2 below.

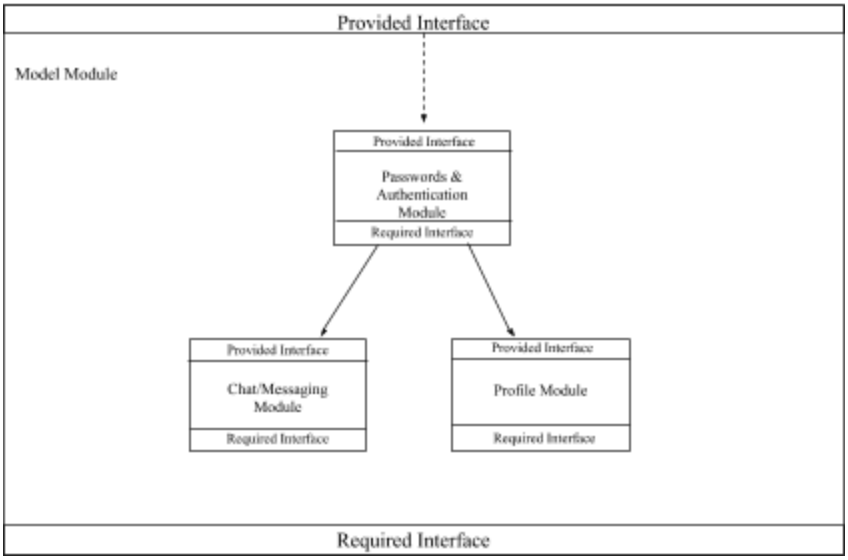

Figure 2. High-Level Model Module Design.

Required Interface
This module has no required interface.

Provided Interface
The provided interface of this module are the passwords and authentication module, the profiles module and the chat/messaging module.

3.1.1 Profiles Module

Purpose
The purpose of this module is to provide access to user profile information, and the ability to create, join, or leave communities, and community leaders' abilities to add or remove community members. Users can also create and rsvp to events, while event owners' are able to invite users, update or delete events. Users can add or remove friends. Additionally, this module provides access to the user's feed and search capabilities within the application. Essentially, the profiles module provides the details and functionalities available to users.

Rationale
This module allows users to access and retrieve any relevant application information from their accounts after successful login to the application. This module is created to consolidate user data.

Required Interfaces
The required interfaces for this module are provided in Assignment 3.

Provided Interfaces
The provided interfaces for this module are provided in Assignment 3.

3.1.2 Passwords and Authentication Module

Purpose
The purpose of this module is to provide authentication services for login of the LetsGo application. This module validates user login credentials against user login information stored within the Firebase database (the model).

Rationale
This module is created to provide a login service to users and promotes the security of user login information with login authentication.

Required Interface

The required interfaces for this module are provided in Assignment 3.

Provided Interface
The provided interfaces for this module are provided in Assignment 3.

3.1.3 Chat/Messaging Module

Purpose
The purpose of the chat module is to provide access to chats within events or communities, or between users. This module acts as a central storage for all messages within the LetsGo application. Users will be able to send messages to users, communities that they are members of, and events that they are attending. Users will also be able to retrieve all messages with a user, community, or event that they are attending.

Rationale
This module is created to centralize and encapsulate chat services provided by the LetsGo application.

Required Interface
The required interfaces for this module are provided in Assignment 3.

Provided Interface
The provided interfaces for this module are provided in Assignment 3.

3.2 View Module

Purpose
The purpose of this module is to provide page layouts for information to be displayed on.

Rationale
This module is created to encapsulate display choices and information to the graphical display. The view module allows users to interact with the system through the application, and allows users to view their account, profile, and application-related information.

High-Level Module Design
The view module is broken down into lower-level modules, shown in Figure 3. All user interactions with the LetsGo application are done through the User Interface module (Section 3.2.1), where events are then handled by the controller module.

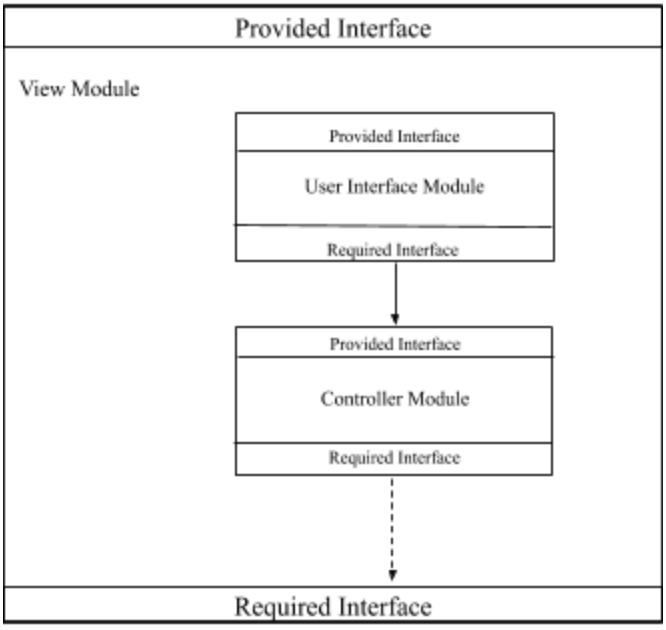

Figure 3. View Module High-Level Architecture.

Required Interface

The view module requires services from the model module and any of its necessary interfaces.

Provided Interface

The provided interface for this module is provided in Assignment 3.

3.2.1 User Interface (UI) Module

Purpose

The purpose of the UI module is to describe the user interface on the user's device.

Rationale

This module allows users to make changes to the model through the controller module, and further through the communications module within the controller.

Required Interfaces

The required interface of this module is provided in Assignment 3.

Provided Interfaces

The provided interface for this module is provided in Assignment 3.

3.3 Controller Module

Purpose

The purpose of this module allows for communication between user events and data which are then updated within the view or model accordingly. Given user interaction, this module communicates with the model and loads updated and correct views of any information to the view module.

Rationale
This module is created to encapsulate interaction semantics of user-interface events, and strengthen interactions between user events and view or model changes.

High-Level Module Design
This module intercepts user interactions and completes its role accordingly. The user events trigger the controller to interact with the model and update, or change, any necessary data.This module is broken down into two lower-level modules, as shown in Figure 4. All user interactions are sent to the controller are sent to the communications module. The high-level controller component and the communications module handle the same requests: user interaction, and are therefore the same.

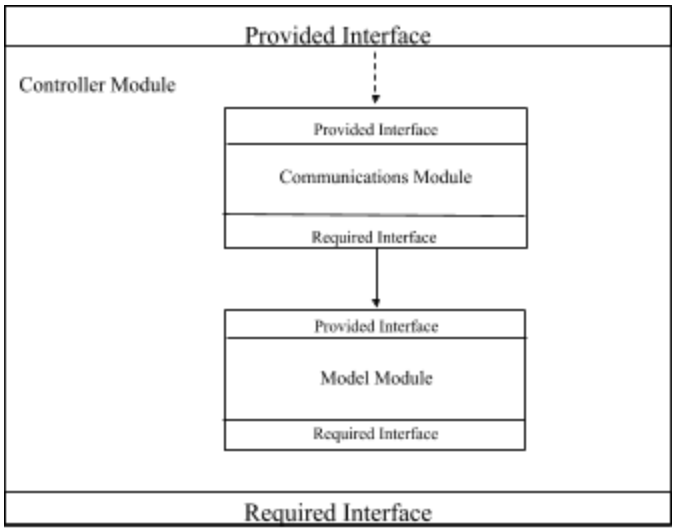

Figure 4. Controller Module High-Level Architecture.

Required Interface
The required interface for the controller module are the services of the model module and any necessary interfaces.

Provided Interface
The provided interface for this module is provided in Assignment 3.

3.3.1 Communications Module

Purpose

The purpose of this module is to provide communication services between the user interactions and the model or view. This module represents the link between the controller, model and view modules based on user interaction. The communications module details how the client retrieves and stores data for the LetsGo application in the Firebase database (the model).

Rationale
This module is created to encapsulate data from user interactions to update the model, and to receive data from the model to be sent to the view.

Required Interfaces
The required interfaces for this module are provided in Assignment 3.

Provided Interface
The provided interface for this module is provided in Assignment 3.

4. Implementation

This section provides a description of the work that has been done over the development process of the LetsGo application.

4.1 Implementation Technologies

To create the LetsGo application, we needed to use an integrated development environment, a development platform, a cloud services/database platform, and a version control system. We used Visual Studio to write all code and run debugging tests for the application. For our development platform, we chose Xamarin Forms, as it provided a way to develop both iOS and Android applications using one shared codebase. We chose Google Firebase, for our NoSQL realtime database and web connection, as Visual Studio provides packages for Firebase integration in Xamarin Forms applications. Lastly, for version control, we used Git, and Github for source code hosting and project management.

For our database design, we followed Firebase documentation's suggestions of storing data in shallow trees with little nesting, as Firebase stores data in a JSON tree. We chose to design our database for the LetsGo application this way for efficient querying, resulting in faster queries and therefore, a faster application. This design can be seen below in Figure 5. Each user's account and profile information is stored under the "userprofiles" branch, while communities are stored under "Communities", events under "Events" and their corresponding chats in "CommunityChats" and "EventChats." The "Announcements" branch stores all announcements made by communities, and is referenced by the community's ID. Additionally, direct messages between friends made in the friend chat option are stored in the "FriendsMessages" branch of the database tree.

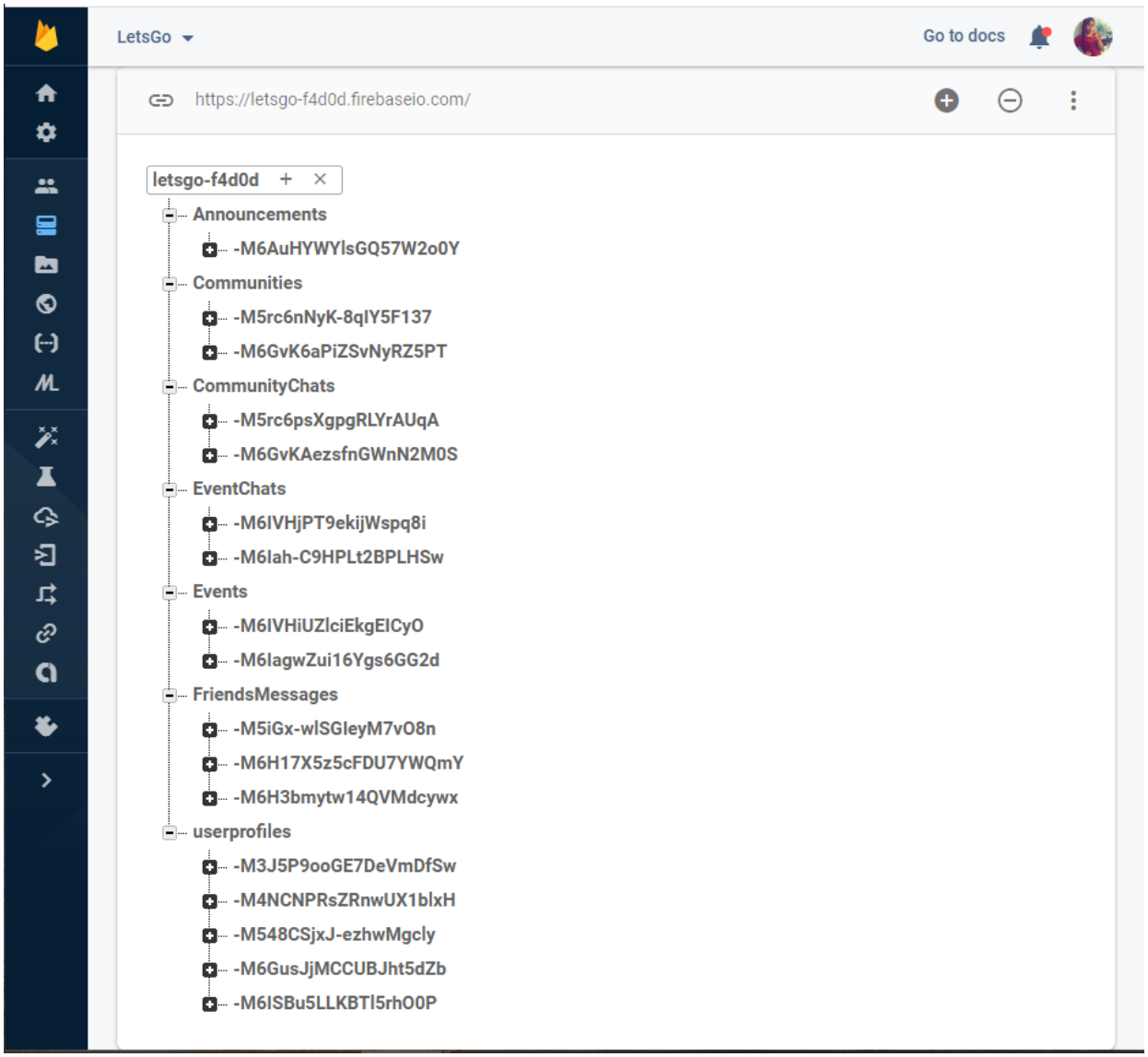

Figure 5. LetsGo Database Hierarchy/Design in Firebase.

For storage of user profile pictures, community images, and event images, we chose to use Firebase Storage. Firebase Storage works similar to Firebase Database, and provides a means to store larger data, such as images. For each user profile picture, community image or event image, we stored the image chosen under the user's email or community/event ID in Firebase Storage, and the URI for the image under the corresponding branch ("userprofile", "Communities", or "Events"). Default images were not stored to save memory and space in our database, and were instead stored in and made as part of our LetsGo project in Xamarin Forms. Our Firebase Storage can be seen below in Figures 6-8.

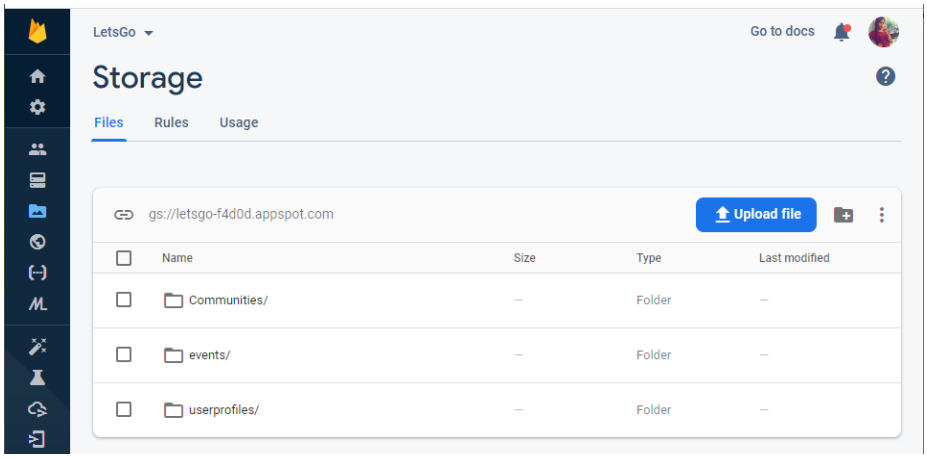

Figure 6. Firebase Storage Overview.

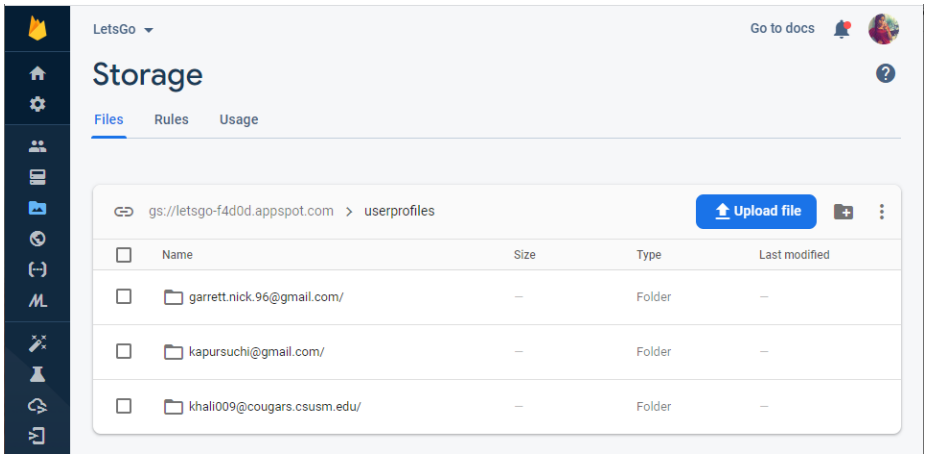

Figure 7. User Profile Storage Folders in Firebase Storage.

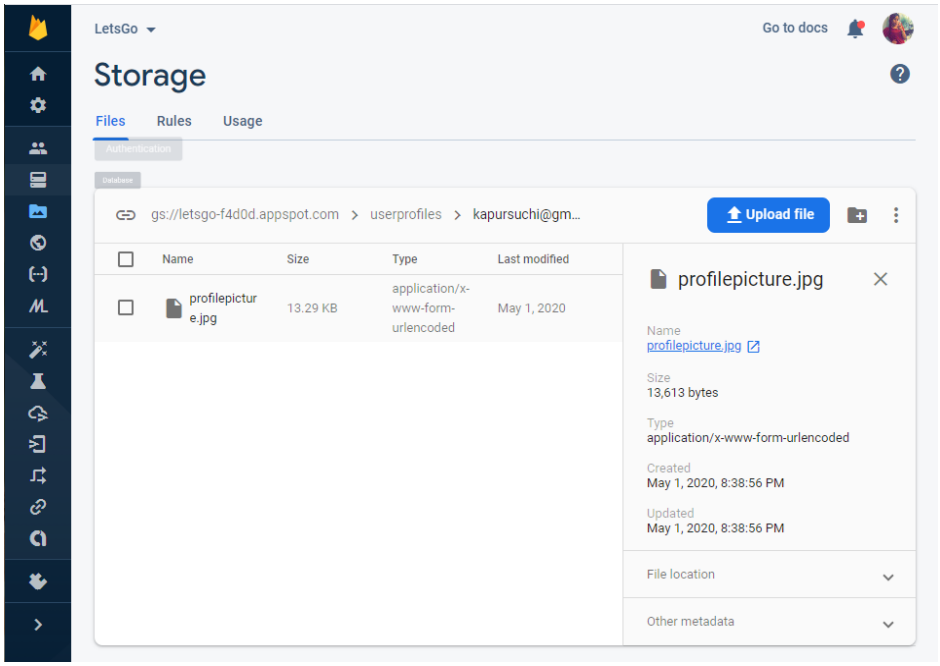

Figure 8. Profile Photo Stored Under User in Firebase Storage.

4.2 Implementation Tasks

Prior to implementation for the LetsGo application, the LetsGo team had to gather requirements from our project owner as part of requirements engineering and decide on an architectural design for the application. We decided to use the Model-View-Controller architecture pattern for our application as LetsGo is a mobile application that relies on user interaction to behave and respond accordingly.

With our development environment and the Model-View-Controller architecture pattern, the development process can be divided as follows:

For each navigation page in the application, files were made for each architecture component:

> **Model:** a file to connect the controller and view pages to the database + Firebase API
>
> **View:** a file dedicated solely to the graphical display of the page
>
> **Controller:** a file that handles user inputs, further navigation, update the view and when to call the model

Additionally with the model, we implemented a Firebase API. It contains many methods that make asynchronous calls to the database through an internet connection, and handles information storage and retrieval, as well as information initialization and deletion.

An example of implementing the task of creating a user account would be to write the pages, where the controller page would call the model page's database method when the new account credentials are entered and the submit button was pressed. The model page's database method calls to the Firebase database page, which would then initialize the user. The user is able to enter their new credentials through form entries defined in the view page. Then, when the account was successfully created, the user is redirected to the login navigation page from the controller, and a new set of model, view, and controller files are defined for the login page.

4.3 Consistency

Over the course of developing the LetsGo application, the developers of LetsGo have intentionally made changes to certain requirements outlined previously in this document. The main reason for many of these inconsistencies are due to a change in idea for the application to improve the application overall. These inconsistencies pertain to the following functional requirements specified in Section 2.3 of this document and Section 3.2 of our Requirements Specification Document (Assignment 2):

Login Modifications:

FR_LI_2: Originally, this requirement stated that the system shall prompt the user to retry, recover their password, or create a new account. In our implementation, the forgot password and create account buttons/links are already on the login screen, so we changed the requirement to simply prompt the user with an “Invalid credentials” message.

Password Reset Modifications:

FR_FP_1: Instead of resetting the user’s password with the forgot password link, we decided to use this service to send the user an email with a temporary password that was updated for their account in the database. In the application’s implementation, the user is required to enter the email associated with their account to receive a temporary password.

Manage Friends Modifications:

FP_MF_5: Instead of clicking a friend from the manage friend’s page to chat with, we decided to include chat as its own page. For chatting with a friend, the user can search for the friend they wish to chat with and start a conversation. Once a conversation is created, the two friends will be able to participate in direct messaging.

Manage Communities Modifications:

FP_MC_2 & FP_MC_3: We decided not to include multiple pages and barriers to view a community’s details, as this is not user friendly. During implementation, we decided that when a community is clicked on, the user will see it based on whether or not they are a member or community leader. If the user is not a member (and therefore, not the community leader as well), they will be able to see the community in its public or private mode. This change essentially combined FP_MC_2 and FP_MC_3 into a single requirement.

FP_MC_7: The original requirement stated that if the leader of a community leaves the community, the system updates the new leader to be the member with the longest membership. As we decided not to track when a member was added to a community, we thought it was a better approach to allow the current community leader to appoint the next leader prior to leaving. This way, the community leader will choose a member that they believe exhibits the traits and responsibility to be the leader.

Search Modifications:

FP_SE_1: For the search requirement, we made a small modification to instead display all matching search results (public and private) based on the interest tag that the user used to search. We chose to conduct our search this way as it allows private communities, events, and users to be found by others, but still provides these groups the privacy they wish by displaying a private view to outside members or users that are not friends with a private user.

Chat Modifications:

We made a change to Section 2.1 of the Requirements Specification (Communication Interfaces), where we originally stated that we would utilize XMPP for direct or group/community messaging. As we had an issue with Firebase Authentication and using Firebase services (other than the realtime database and Firebase Storage), we decided to manually implement the chat function for the LetsGo application. The manual implementation does not require any protocols or frameworks, as it solely relies on our realtime Firebase database.

4.4 System Availability

LetsGo application source code (Github repository): https://github.com/ntgarrett/LetsGo

LetsGo application video demo (YouTube link): https://youtu.be/5lxtrTpT9GI

5. Project Management

This section outlines each team member's contributions to the LetsGo project and application throughout the semester. Specific contributions are detailed in the tables below, while implementation contributions can also be seen in the GitHub snapshot of commits (Figure 9).

5.1 Team Contributions

	Suchi	Khaled	Nick
Requirements Specification	X	X	X
Architecture Design	X	X	X
Implementation	X	X	X
User Interface/Design	X		X
Requirements Verification and Validation	X		
Final Report	X		X
Made Presentations: 1, 2, 3	1, 2, 3	1	1, 2, 3

Table 1. Application Planning and Overall Contributions.

Developer	Account Setup	User Profiles	Feed	Search	Friends	Commun-ities	Events	Chat	Video Demos
Suchi	X	X	X	X	X	X	X	X	
Khaled	X						X		
Nick	X	X				X		X	X

Table 2. Implementation-Specific Contributions.

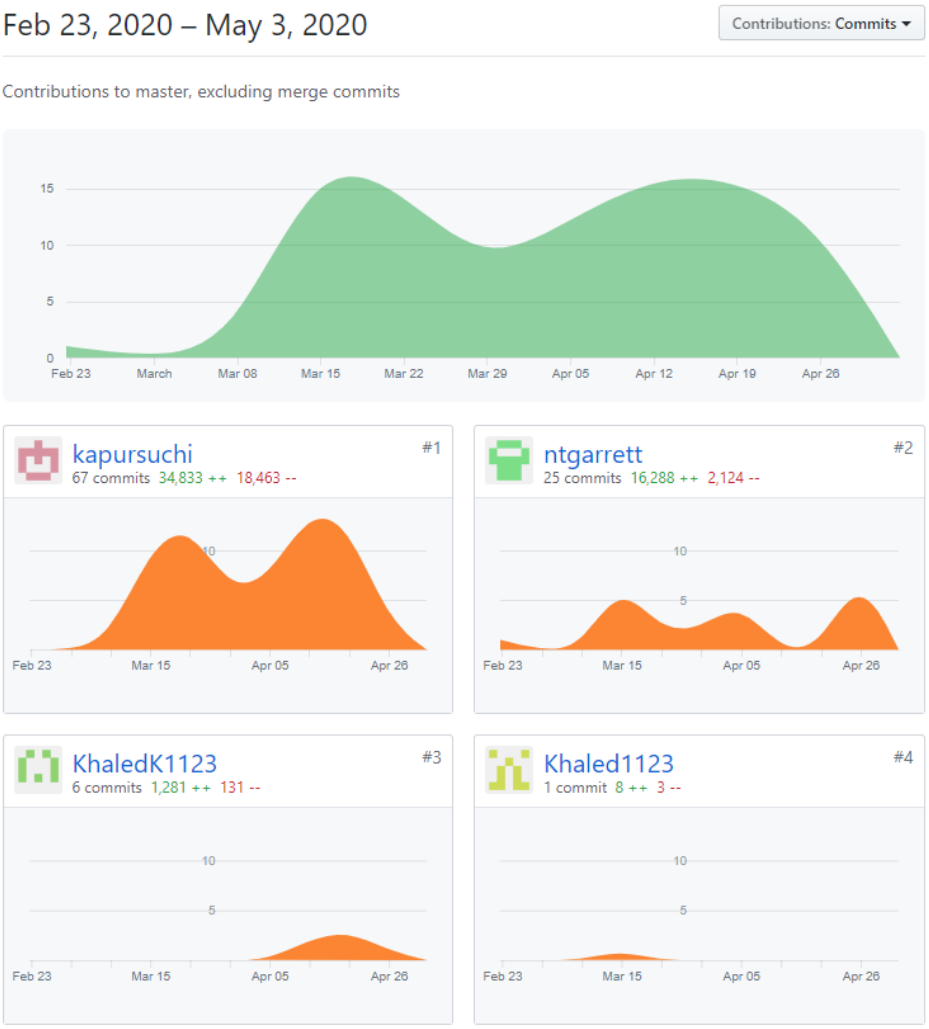

Figure 9. GitHub Contributions for the Implementation/Development of the LetsGo Application.

5.2 Development Process Problems

During the development of LetsGo, we faced a few roadblocks along the way. The first major issue was implementing Firebase Authentication for the user profiles, which would provide additional authentication layers beyond an email address and password per user. We found

incompatibilities and inconsistencies with different versions of Visual Studio framework packages that would enable the authentication to work across both iOS and Android platforms using the Xamarin Forms shared codebase process. Our solution to this issue was to design authentication and user login manually, which would allow users on either iOS or Android devices to login and use all features of the application.

One other issue we found while developing LetsGo was that the shared codebase for iOS and Android wasn't exact. Although developing cross-platform applications is not guaranteed to only develop in one codebase with Xamarin Forms, we found that certain image files, screen resolution views, and other minor displays differed when testing on both iOS and Android devices. Our issues with this resided mostly with the iOS implementation; while certain displays, tools, or aspects were available in Xamarin Forms for Android, they were not for iOS.
Our solution to this was to enter the platform-specific folders for iOS and Android, and manually adjust code and image sizes to achieve the desired results. After completing all functions and tasks necessary as part of implementation, we then had to ensure that our iOS implementation was essentially up to speed with Android. This was also a challenge for our team because of Apple's guidelines for developing where developers may only test iOS applications for free on Xamarin Forms with an Apple computer.

6. Conclusion

LetsGo is a cross-platform mobile device application for iOS and Android devices, with the goal of embracing community interaction based on similar qualities and attributes. Users of LetsGo can create their own social media profiles, and add other users as friends. Users can also plan events with their friends, communities, or create public ones to meet new people. Users can create their own communities, join others, and chat with each other. To develop this application, we used the Xamarin Forms cross-platform development format, which includes the C# language for the program code, as well as XAML (a declarative markup language) for the view. In Microsoft Visual Studio, a shared codebase was used for main development, and iOS and Android folders were created for platform specific code. We designed the application with the Model View Controller architecture pattern, and used Google Firebase for our backend and database services.

7. References

Xamarin Forms documentation: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/

Google Firebase Database documentation: https://firebase.google.com/docs/database