Software Architecture

Version 2.0

Will Code for Food
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA USA
November 11, 2015

Revision History

Name	Date	Changes	Version
Gavin Scott	11/30/2015	Edited Solution Overview, Common Class Diagram, Ridesharing Class Diagram, Enroll as Driver Sequence Diagram	2.0
(name hidden)	11/30/2015	Edited Get Involved Class Diagram, Join Community Group Sequence Diagram	2.0
(name hidden)	11/30/2015	Edited Event Class Diagram, Deployment Diagram, Resources Activity Diagram, Problem Description Reordered Diagrams, Grammar/Spell-Checking	2.0
(name hidden)	11/30/2015	Edited Introduction, Problem Description, Event to Calendar Sequence Diagram, Event to Ride-Sharing Sequence Diagram, Summer Missions Class Diagram	2.0
(name hidden)	11/30/2015	Edited sections Resources Class Diagram, Access Leader Resources Sequence Diagram	2.0
(name hidden)	11/30/2015	Edited Deployment Diagram, Dialog Map, Rideshare Activity Diagram, Settings Class Diagram	2.0

Credits

Name	Date	Role	Version
(name hidden)	11/10/2015	Resources Activity Diagram, Events Class Diagram	1.0
(name hidden)	11/10/2015	Deployment Diagram, Dialog Map, Rideshare Activity Diagram, Settings Class Diagram	1.0
(name hidden)	11/10/2015	Introduction, Problem Description, Event to Calendar Sequence Diagram, Event to Ride Sharing Sequence Diagram, Summer Missions Class Diagram	1.0
Gavin Scott	11/10/2015	Solution Overview, Common Class Diagram, Ridesharing Class Diagram, Enroll as Driver	1.0

		Sequence Diagram	
(name hidden)	11/10/2015	Resources Class Diagram, Access Leader Resources Sequence Diagram	1.0
(name hidden)	11/10/2015	Get Involved Class Diagram, Join Community Group Sequence Diagram	1.0

1 Introduction

This document describes the architecture of the Cru Mobile app as created by Will Code For Food. It contains a problem description and our proposed architecture for the solution. For more information please see the Vision and Scope document or the System Requirements document.

2 Problem Description

2.1 Android Cru Application

Our product involves three main sections: ride-sharing, feeds and information, and joining community groups. This application aims to reduce the load on Cru leadership by allowing members to easily learn about events, find or offer rides, systematically connect members to community groups, and access relevant information. Our application will be well-modularized to allow for future changes to the Cru structure or server.

2.2 REST API

Our product will also involve an additional API, written in conjunction with the other Cru mobile teams, that interacts with the existing Cru server to access and modify information in the Cru databases.

3 Solution Overview

The system architecture is divided up into three primary sections: model, view, and controller. The components of the view package designate the visible GUI of the program, containing all of the Android-specific code and the code to build visible representations of the objects stored in the rest of the project. The controller package acts as a go-between for the view and the model, which contains all the required data structures for the project.

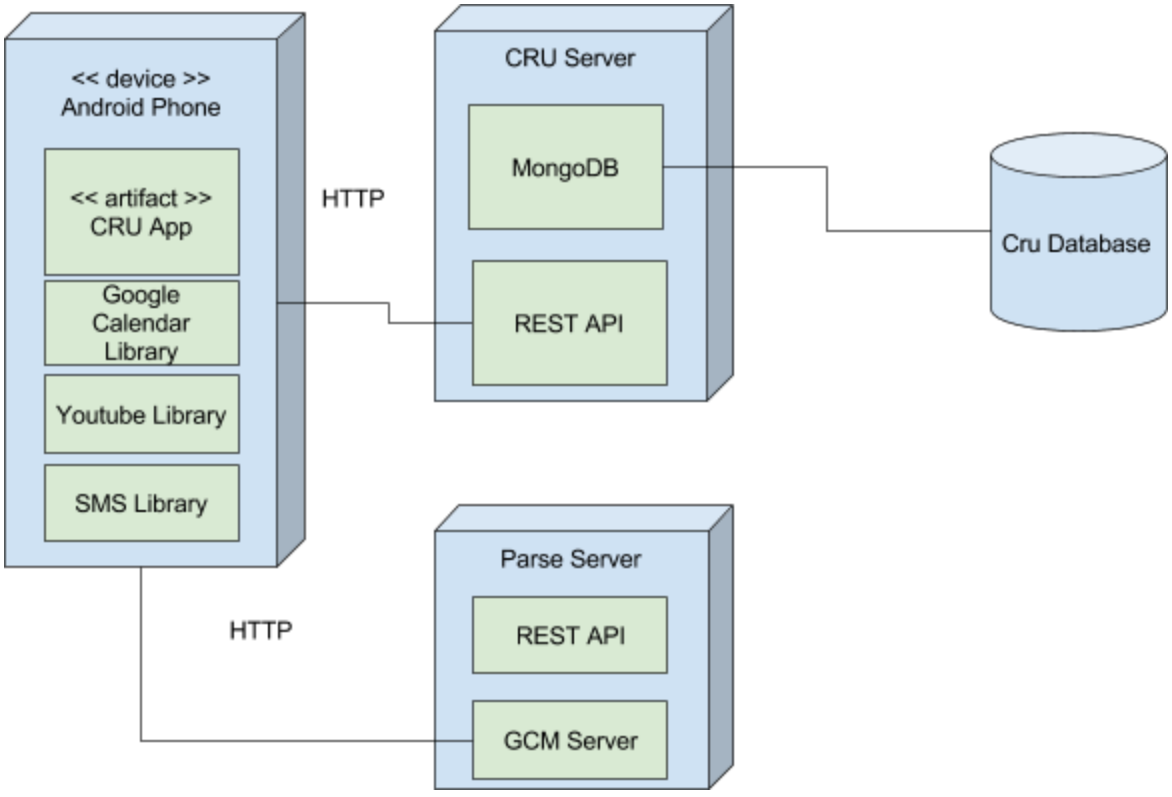
The model package is divided up into multiple subpackages based on the primary sections of the app. For instance, the ridesharing package contains all of the relevant data structures unique to ridesharing, the events package has event related code, etc. To encourage modularity, each of these subpackages will contain no references to each other; this will allow sections of the app to be added, modified, or deleted, without affecting overall functionality. In the cases where some shared data structures are required, such as

the structure for common elements like users, there is a "common" package containing interfaces and abstract classes that the more specific model packages can implement. For example, it contains common interfaces for sending messages to the user using text messaging and email, and sending queries to the CRU database.

To avoid confusion, there will be terms like "Ridesharing API," "Texting API," and "Calendar API" used in some of the diagrams throughout this document. These are not third-party API's, they are simply the portions of our program that control those sections of the app. The different components of the app are kept separate from each other, with small interface methods provided so they can communicate as necessary. We treat them as separate APIs in some of the diagrams because it emphasizes the fact that the different sections are kept as separate as possible to increase the project's overall modularity.

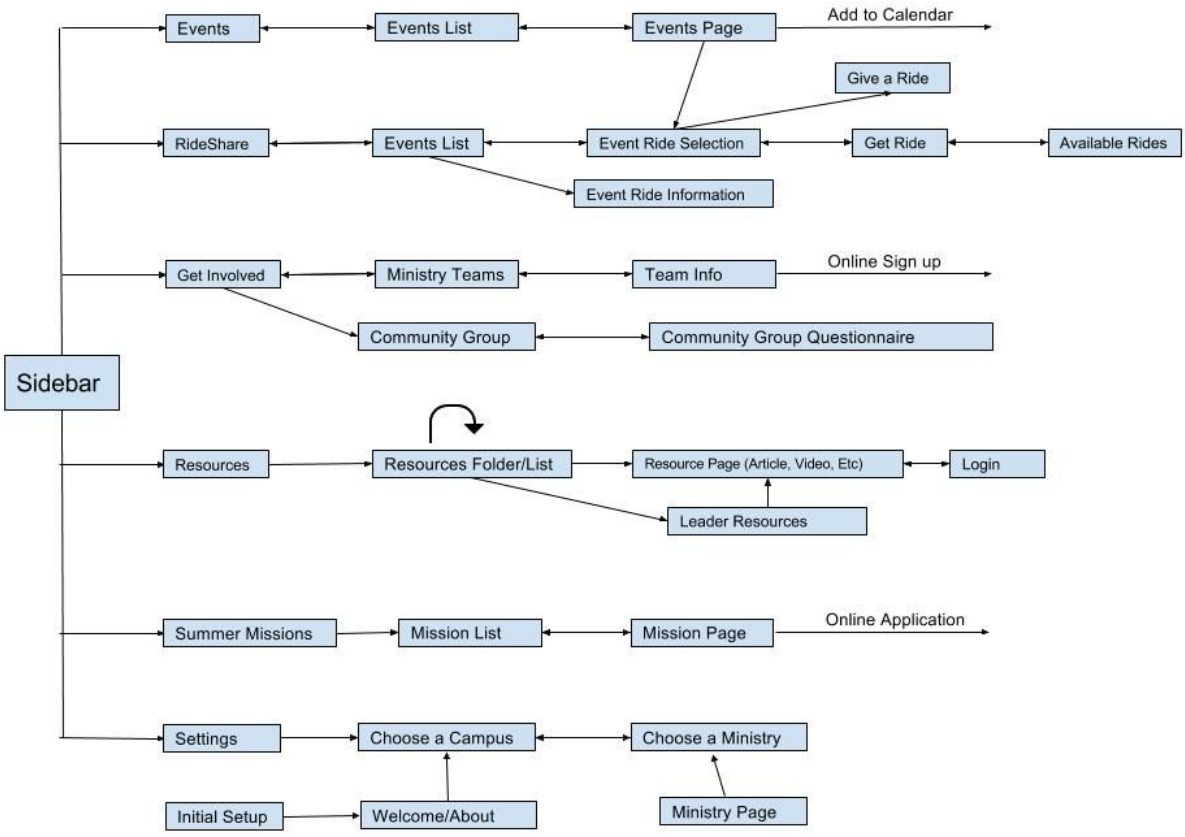
4 Solution Components

4.1 Deployment Diagram



The CRU app will connect to the CRU's mongo database through a REST API on Cru's server. Push notifications will be handled through Parse, which will send the push notifications to the android application. Parse notifications can be sent through many APIs and SDKs, notable through a REST API and Parse.com. Our application also uses connects to Youtube, Google Calendar, and SMS APIs.

4.2 Dialog Map



The dialog map above shows how a user could work through the app, starting at the sidebar and then going through each of the tabs. Each of the boxes represents a page that the user will see and interact with.

4.3 Database Schema

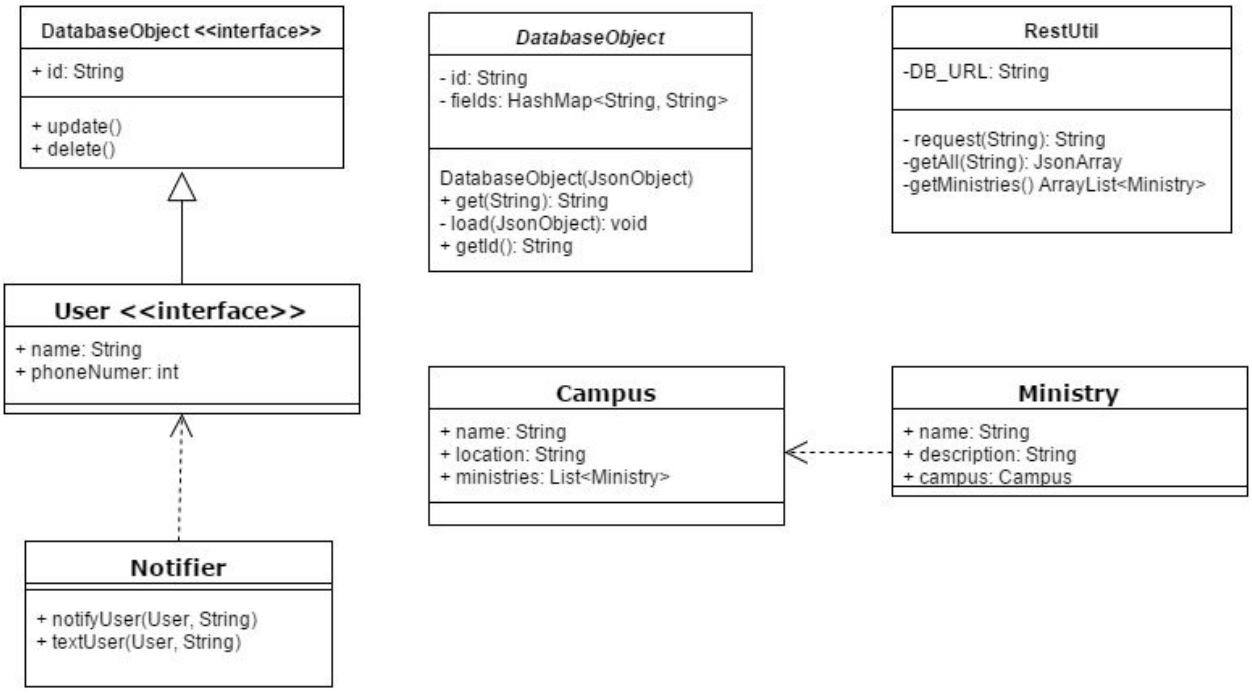
Table	Fields
app_sessions	_id, session, expires
app_updates	_id, key, appliedOn, __v
campus	_id, slug, name, location, __v, url
events	_id, slug, url, description, name, rideSharingEnabled, endDate, startDate, location, __v, image, notificationDate, parentMinistry,

	parentMinistries
galleries	(none)
ministries	_id, slug, description, name, campuses, teams, __v, image
ministryteams	_id, slug, parentMinistry, description, name, __v
objectlabs-system	(none)
objectlabs-system.admin.collections	(none)
postcategories	(none)
posts	(none)
summermissions	id, slug, description, name, endDate, startDate, location, __v, image, url, cost, leaders
system.indexes	v, key, name, ns
users	_id, updatedAt, createdAt, key, isVerified, isAdmin, password, email, notifications, communityGroups, summerMissions, ministryTeams, yearLeading, isSummerMissionLeader, isMinistryTeamLeader, isCommunityGroupLeader, isStaff, isPublic, name, __v, updatedBy, phone, schoolYear, sex
rides	_id, driver, riders, eventId, totalSeats, leaveStart, startLoc, two-way, leaveEvent, leaveLoc

5 Solution Architecture

5.1 Common Package

The Common package contains resources used by all other packages.

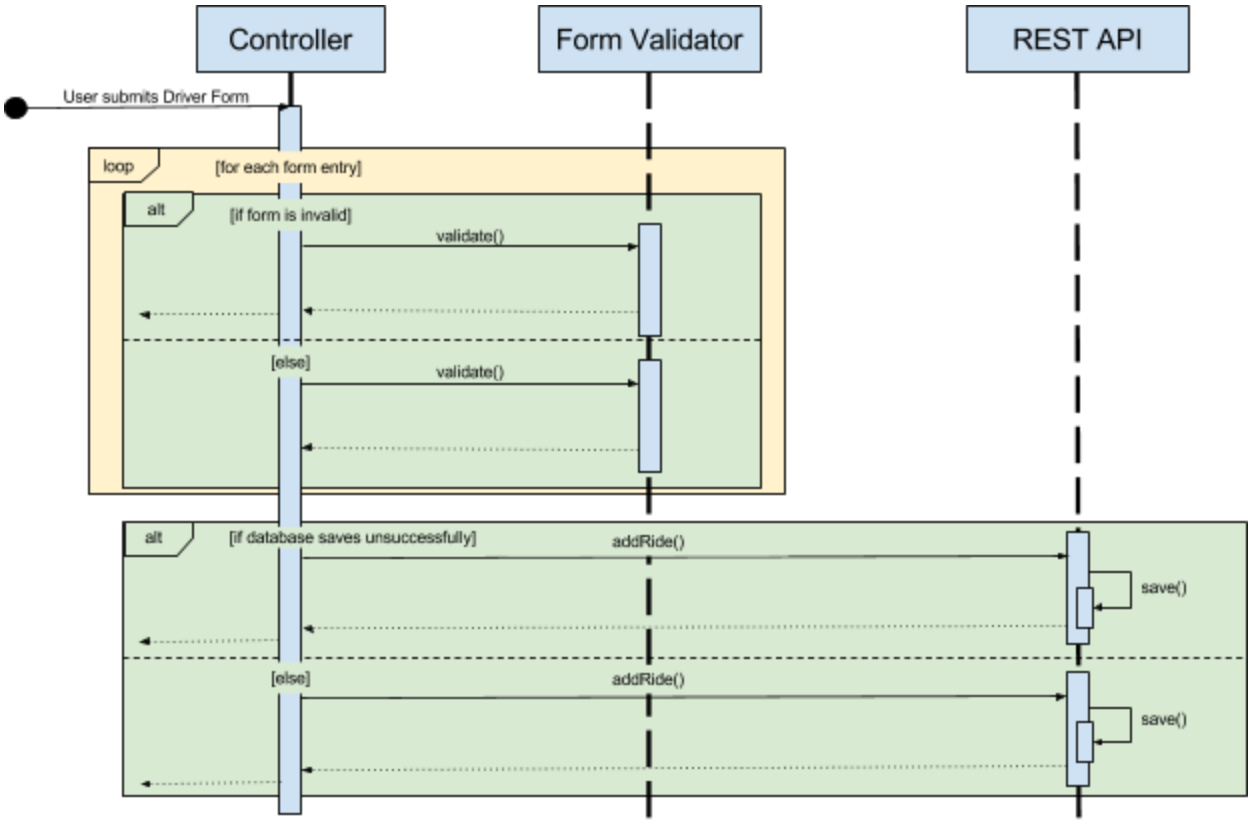


DatabaseObject gets extended by any data structure that will be stored in the database. They contain a map of every field in the object so that, in the future, new fields can be easily retrieved and used throughout the application without changing children classes. Children of this class will set their own fields by calling the get method after loading the object and manipulating the data as needed after load.

Users represent anyone using the app. and Notifier handles the sending of information from the app to a user in the form of text message, or push notifications. Campuses and Ministries are also common components that are kept in the Common package as they are used by almost every other part of the application.

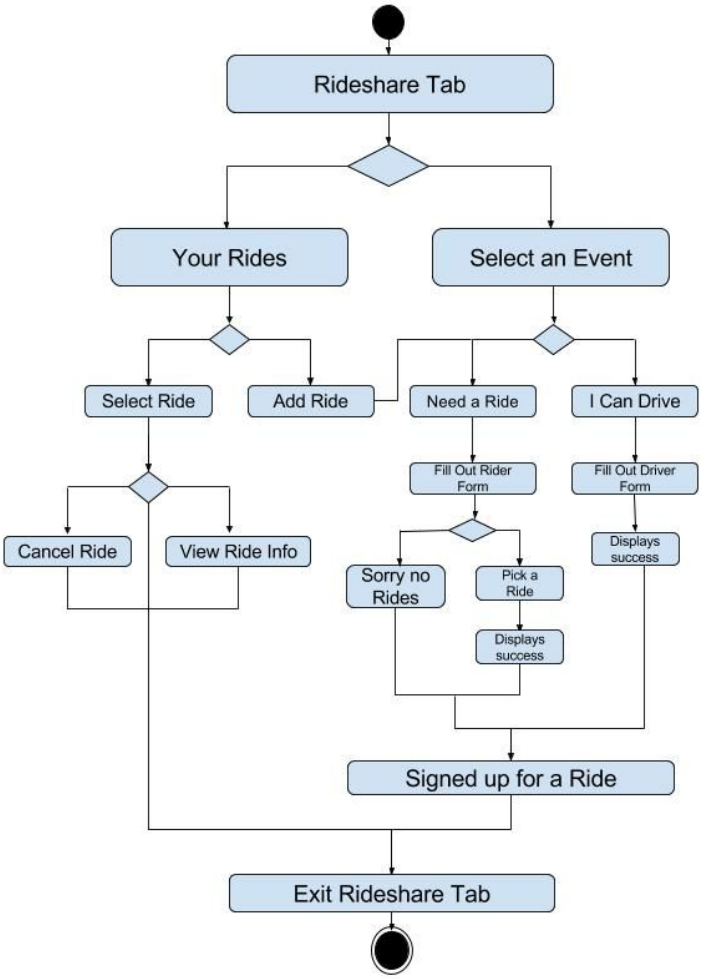
5.2 Ride Sharing

5.2.1 Sequence Diagram



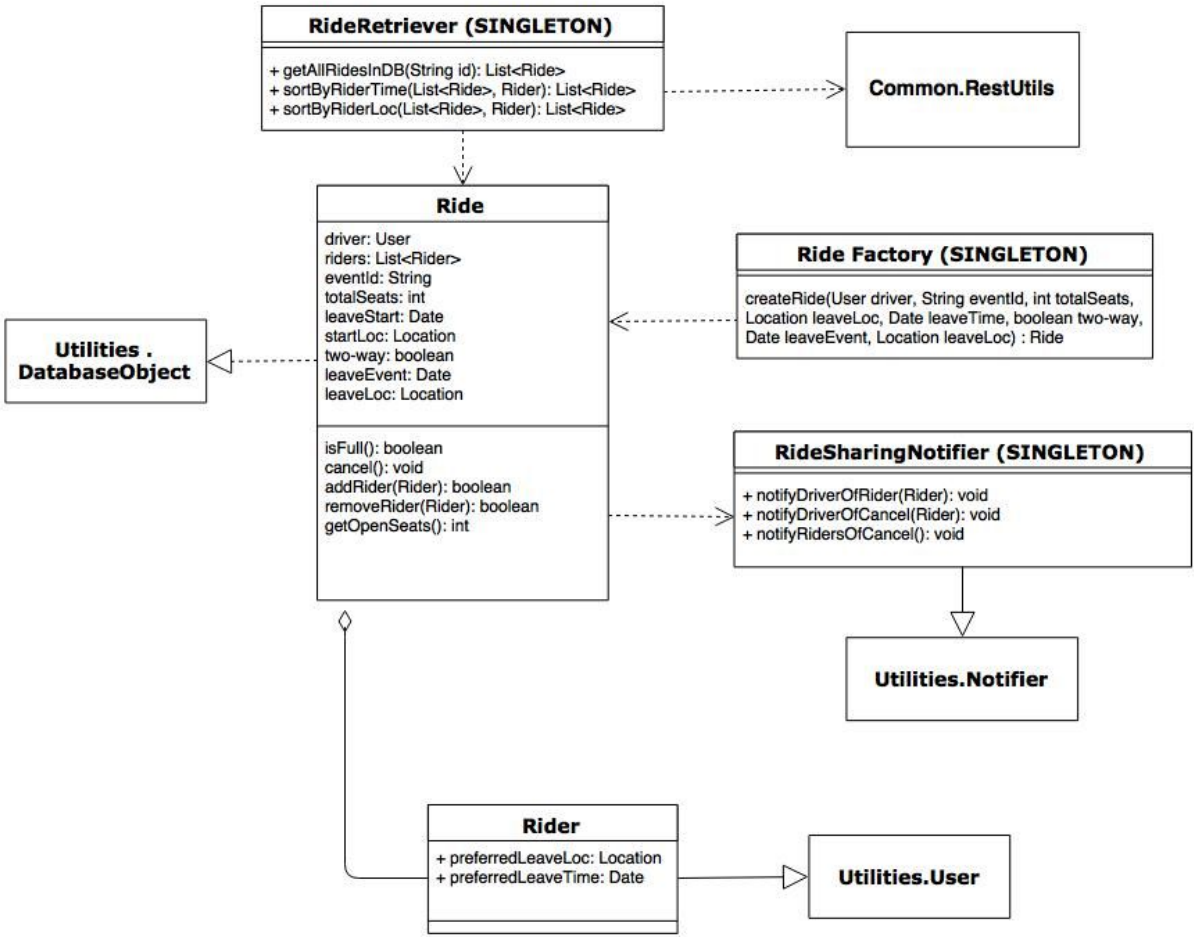
When a user wants to register as a driver they select the "I can drive" option for the event, through either the ridesharing or events tab. They are then presented with a form requesting their information, as it relates to ridesharing. After submitting their data, each form entry is put through a validator to ensure the user's input is valid. If it isn't, the form will return a failure and the UI will display an error. If it is valid, it attempts to save the information to the database; if that is successful, the sequence returns success and the UI displays a success message. A very similar system is used for registering as a rider for a ride to an event.

5.2.2 Activity Diagram



The rideshare tab starts with having the user select an event or their rides. Selecting their rides will show them which rides they are signed up for and the user can cancel or add more. If they select an event, then they need to select if they are willing to drive or need a ride. If the user selects they are a driver then they fill out the form and submit it. If the user selects that would like a ride, they also fill out a form and then select the ride.

5.2.3 Class Diagram

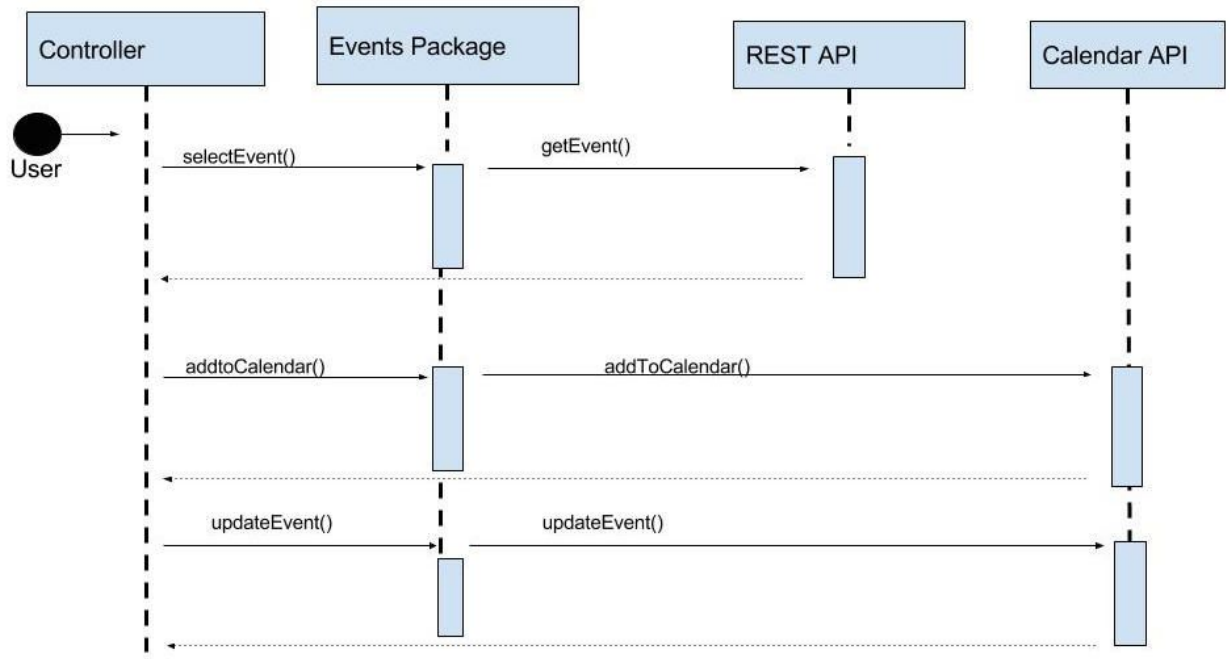


The ridesharing package contains all of the classes and data structures needed for the ridesharing portion of the app. The main class is Ride, which contains all of the information associated with a single ride, such as a driver, riders, when the ride leaves and where it leaves from, etc. It extends the the DatabaseObject class because it will be stored in the CRU database. A Notifier object handles sending messages to riders if the ride is canceled and sending messages to the driver when a rider signs up for their car. RideFactory handles the creation of new rides, and performs input validation beforehand. The Rider class is an extension of the User class and includes a rider's preferences, and the RideRetriever class handles retrieving ride information from the database and sorting it based on a rider's preferred departure time or locations.

5.3 Events

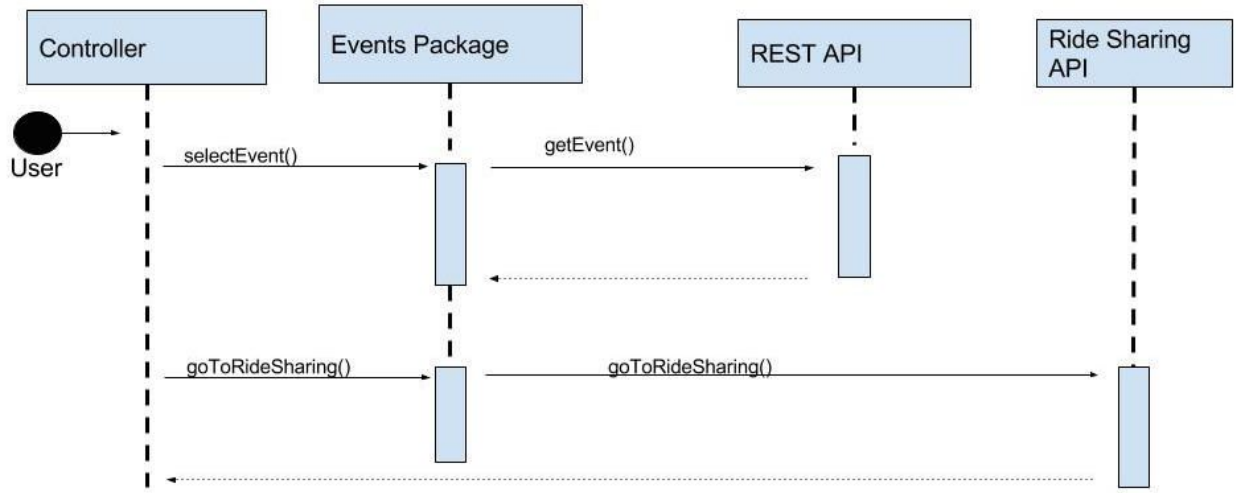
5.3.1 Sequence Diagram

5.3.1.1 Event to Calendar



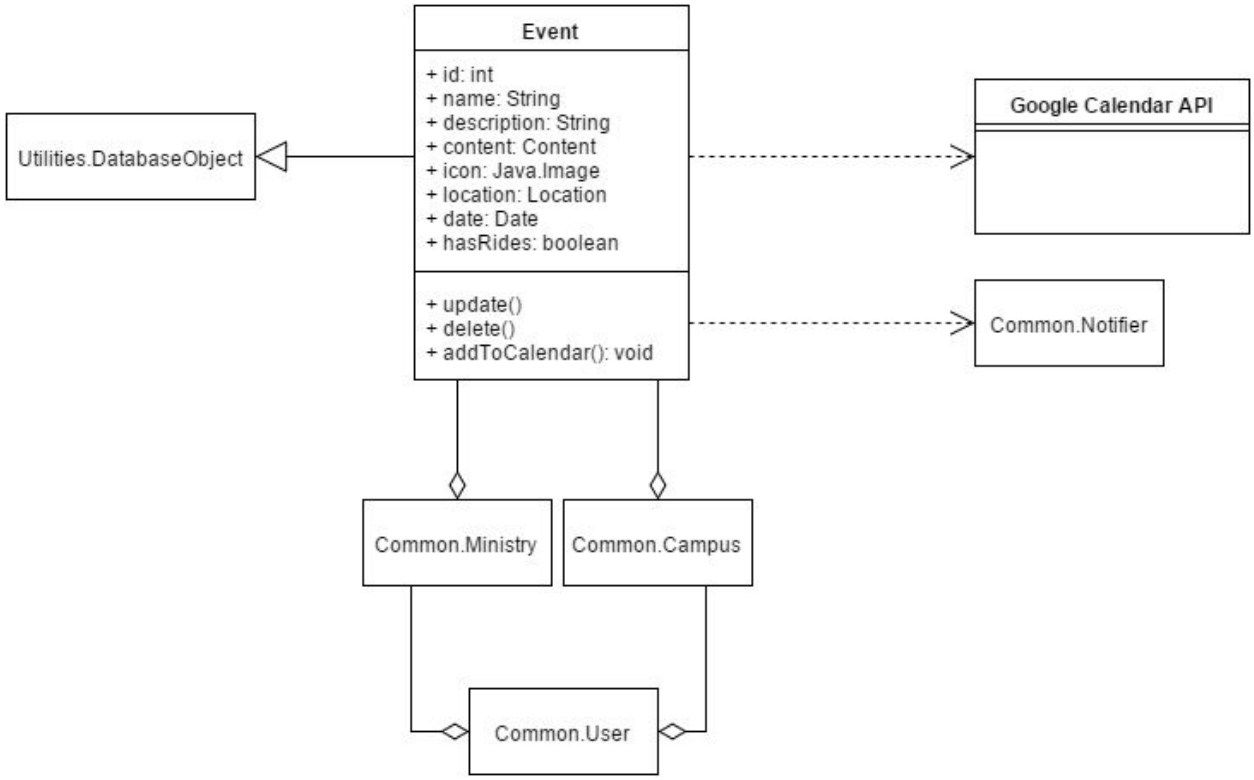
When a user selects an event, the Events Package (model) must retrieve the event details from the database so it can update the UI. When a user chooses to add the event to their calendar, it must communicate with the local Calendar API to add the event. If an event is updated, the Calendar Event is updated using the local Calendar API.

5.3.1.2 Event to Rides



When a user selects an event, the Events Package (model) must retrieve the event details from the database so it can update the UI. When a user chooses to view ride sharing for an event, it must communicate with the ride sharing API to open the ride sharing capabilities for the event.

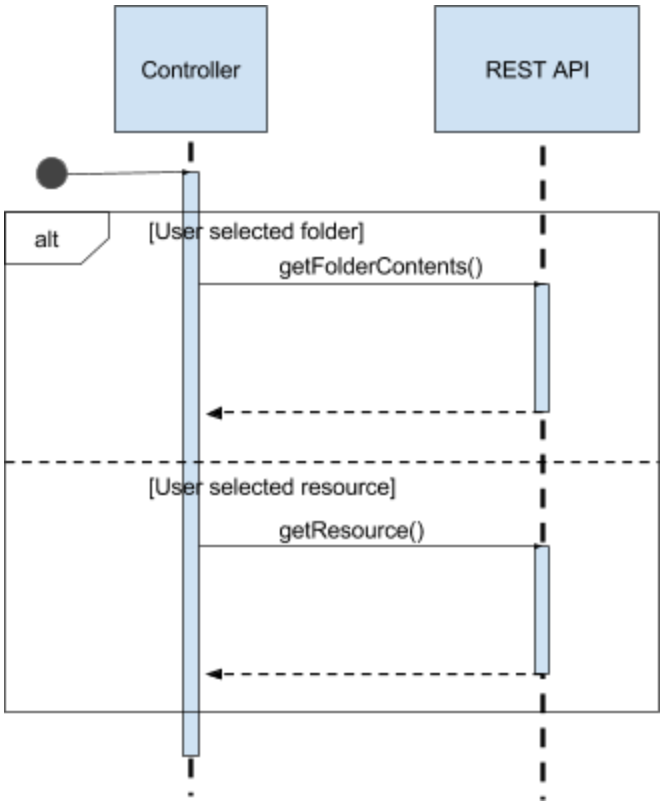
5.3.2 Class Diagram



The Events package provides a model of an Event for the EventController to use. An Event is a DatabaseObject, meaning it is retrieved and updated from the Cru database. Both ministries and campuses can contain events, and a User contains both ministries and campuses. An Event would use the common Notifier class to send messages and notifications, and would also interact with the Google Calendar API when the user wishes to add an event to calendar.

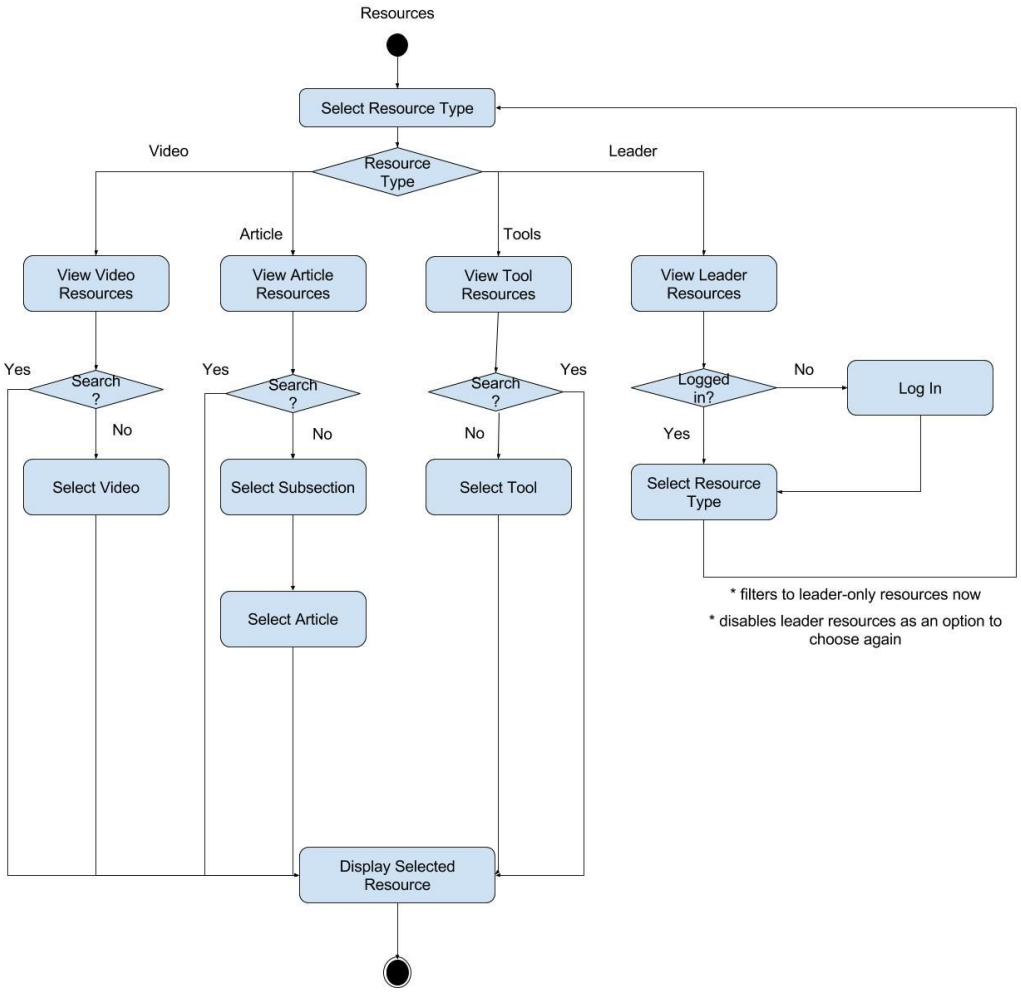
5.4 Resources

5.4.1 Sequence Diagram



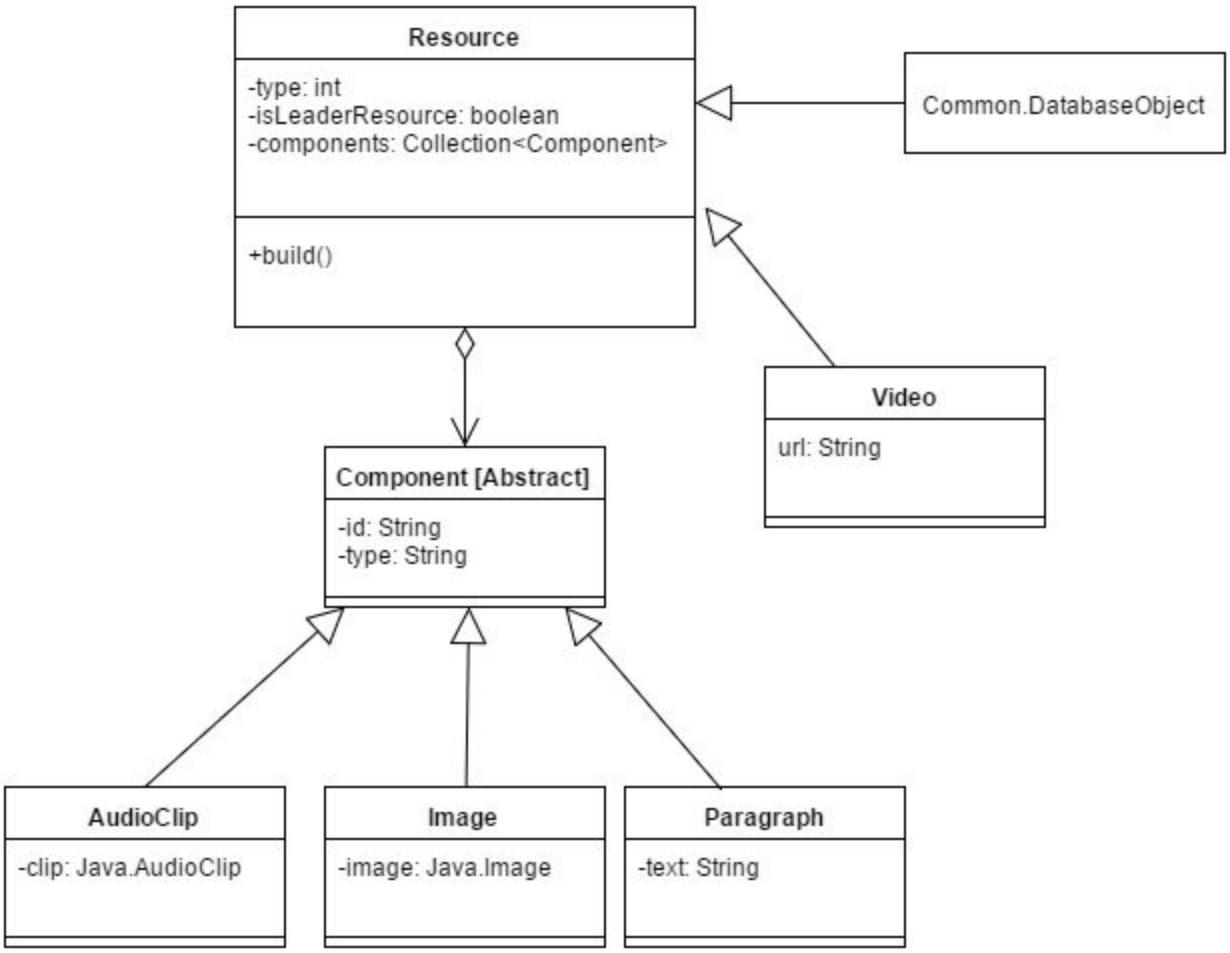
When a user selects a folder or a resource in the Leader Resources section, the Rest API sends the corresponding data back to be used by the app.

5.4.2 Activity Diagram



Starting from the Resources tab, the user selects which resource type they wish to access, which creates a guard that branches based on the resource type. The available resource types are videos, articles, and tools. The user can also view community leader only resources, which will prompt them to log-in if they haven't already, then filter to display only resources for community leaders. Some resources types may have additional filters, and all resource types can be search/filtered to display a single or limited choice of resources.. Eventually a single resource file is selected, at which point the system will display or play that resource, ending the flow.

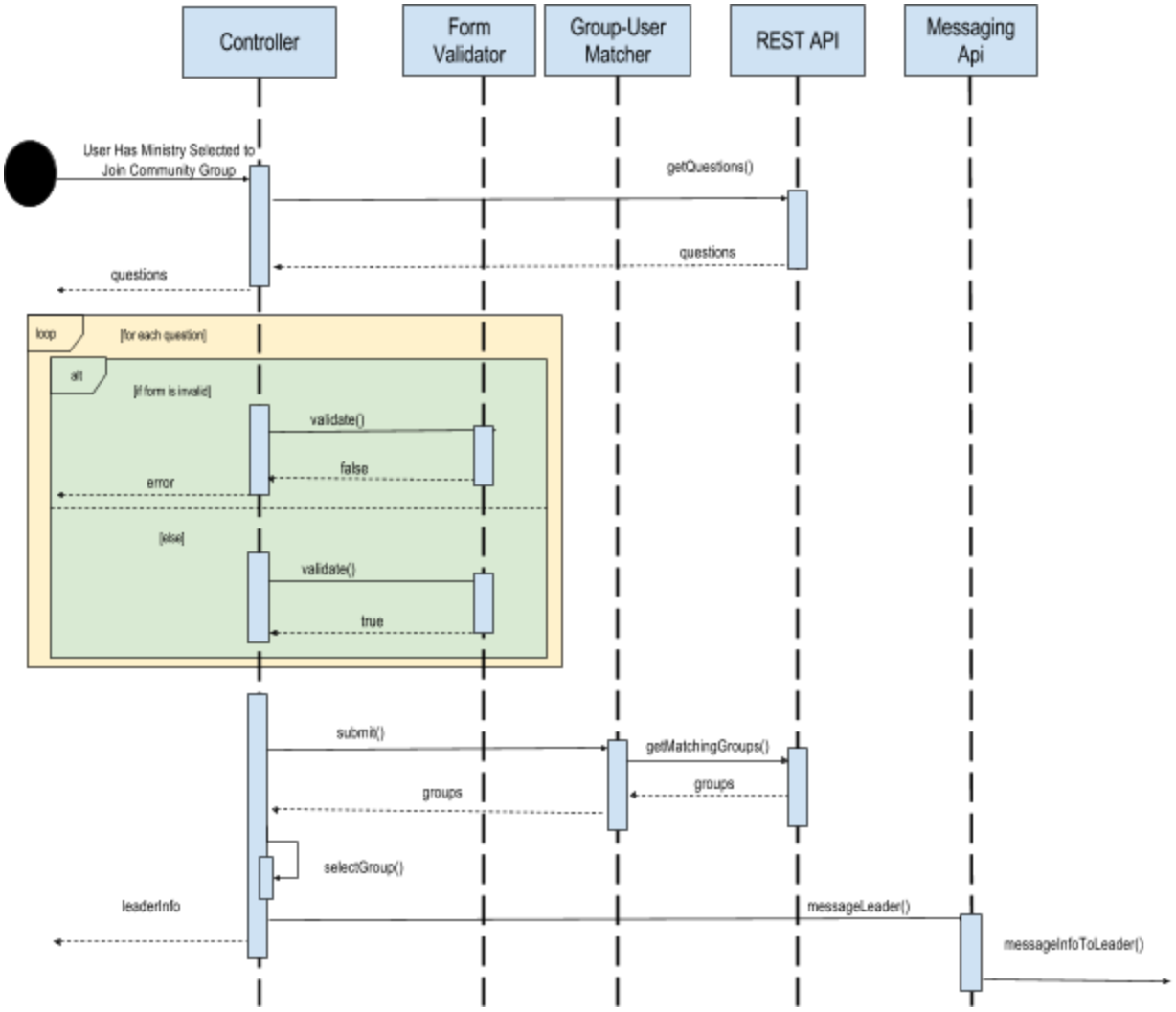
5.4.3 Class Diagram



The Resources package handles the accessing of resources by both regular users and community group leaders. It provides a model of the three types of resource—Video, Audio, and Article—which are used by the controller. Audio and Article resources get rolled into the the Resource class, which extends Card.Content. A resource consists of multiple components such as Images and Audio. A video is just an outside link to a YouTube video, so all it needs to contain is the URL.

5.5 Join a Community Group

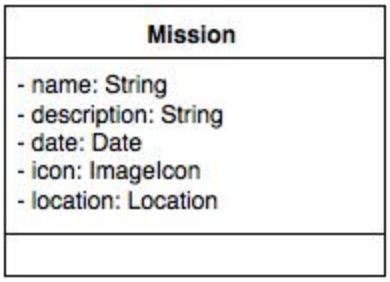
5.5.1 Sequence Diagram



When a user selects a ministry in order to join a community group, the application figures out which custom questions to display based on the ministry. Question answers will be validated when a user fills them out. Upon submitting, the application finds matching community groups. When the user selects a group, the applications sends the user's info to the leader, and the leader's information to the user.

5.6 Summer Missions

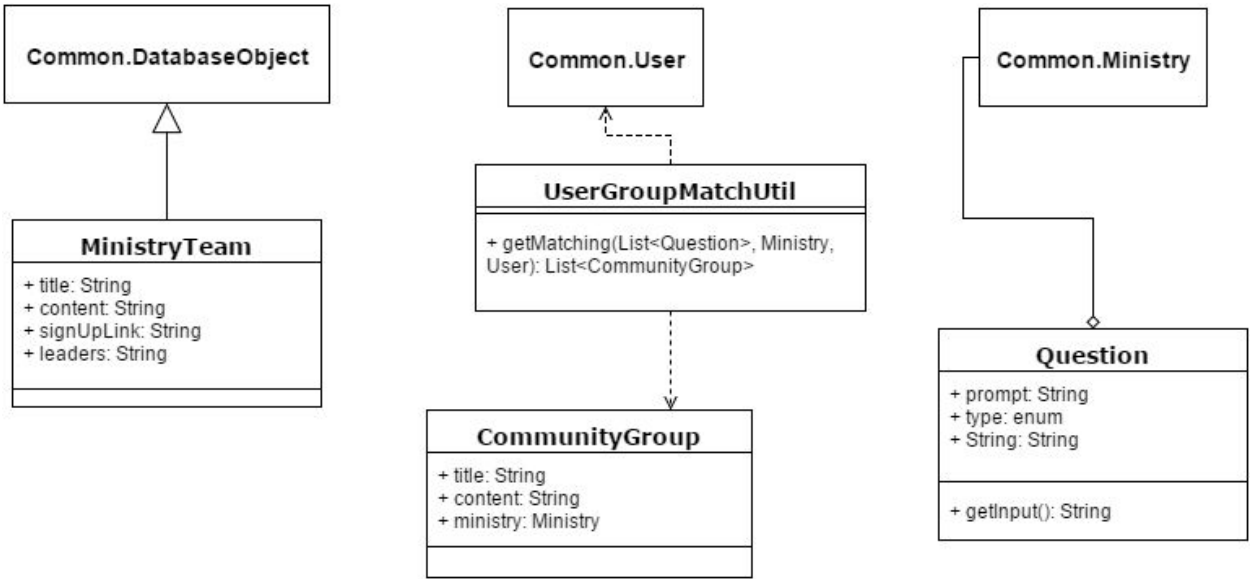
5.6.1 Class Diagram



This class diagram represents the class entities associated with the Missions tab. A Mission is a basic class with details on the mission that will be displayed in the application.

5.7 Getting Involved

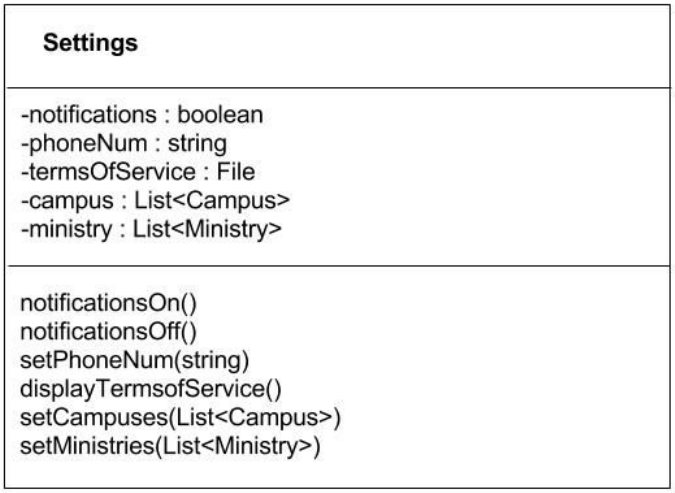
5.7.1 Class Diagram



Get Involved uses ministry teams and community groups from the database. There are default and ministry specific questions for joining a community group. A matcher class matches a user to a community group and the Common.Notifier will be used to give a group leader the information about the user.

5.8 Settings

5.8.1 Class Diagram



Settings is a simple class that will hold the preferences of the user. It will hold the campuses, ministries and their phone number for the apps use. Settings also has the ability turn on and off notifications.

6 Issues

There are currently no outstanding issues affecting the development process.

A Glossary

Not applicable at this time

B Issues List

Issue	Description	Completed?
I-1	Add database schema section	Yes