OPPORTUNITY

SUPPLIER DELIVERABLE 3A

Jake Runzer	Management plan
	UI Walkthrough
	Pseudocode
	Unit Tests
Zev Isert	UML
	Minimal System
	Performance Testing
Claire Champernowne	Executive Summary
	Use Cases
	Proof
Dylan Golden	Use Cases
	Error Handling
	Defense of Integration Plan

Table of Contents

Glossary

REST	Representational State Transfer, an architecture style for designing networked applications, which relies on a stateless, client-server, cacheable communications protocol.
API	Application Program Interface, specifies how software components should interact.
OPP	Event created by the Opportunity application referring to the meeting of all preconditions for a given Opportunity notification.
TRIGGER	The condition or conditions that must be satisfied for the Opp to produce a notification.
SSD	Solid State Drive, a persistent data storage device using integrated circuits and designed without moving parts.
UI	User Interface, the set of design choices for an information device of which a human being may interact with.
IOS	iPhone Operating System, a popular mobile operating system designed by Apple Inc.
OAUTH	Open authentication. A system to provide secure authentication to an application using sign in information from the providing system.

Executive summary

This report describes the mobile application, Opportunity. Specifically, the interface, use interaction, functional specifications, minimum deliverable, testing overview, and plans for implementation and management will be covered.

Opportunity can be used for various functions. Users will utilize the app to help them:

- Remember or plan activities that they want/need to do.
- Fill stretches of free time with appropriate activities
- View and select local events

Users interact with Opportunity in a variety of ways. A user can:

- Create an account using email, Twitter, or Google
- Create custom Opps
- Delete or disable previously created Opps

Though the user will only be interacting with the Opportunity app, the system as a whole is comprised of several parts:

- iOS mobile application
- Backend server
- Database
- SQLite website

To be considered minimally acceptable, the following triggers must be available in Opportunity:

- Time: Represented by the application's ability to specify time in hours or use generalizations such as sunrise and sunset
- Weather Represented by the application's ability to query a weather API, yet restricted to current weather at present location of device
- Availability Represented by the application's ability to query user's calendar to calculate availability.
- Location Represented by the application's ability to determine location, yet restricted to street addresses or geographic coordinates.

Testing for Opportunity is separated into several categories:

- Unit tests
- Integration tests
- UI tests
- Performance tests
- User tests

Functional specifications

Opportunity will be used by people who have difficulty remembering or planning activities that they want or need to do. It allows users who have identified stretches of free time to fill it with appropriate activities. Opportunity will be able to show local events to the user letting them decide what they want to do with their time. Users can do all the things they want to do when the timing is perfect. It also allows users to spontaneously meet up and do activities with their friends. Opportunity is aware of its surroundings location in the world. It uses GPS along with data to determine if an Opp's parameters are met or not. It also monitors the weather; observing the conditions. The application also is aware of the locations of other users and Opps that they choose to share. Opportunity will be aware of local events, store inventories, and ski resort snow levels.

For the minimal implementation, an iPhone will be used as hardware and data will be stored on the phone itself. We will be using the built in GPS, accurate to within 10 meters of location. The application is meant to be dependable so that business and power users can rely on it. For the minimal implementation security is not a high requirement since everything is stored on the phone and none of the information is given out or stored on a server, so creating accounts will not be available. Opps in the minimal implementation will include conditions based on time/date, calendar, specific locations and weather based settings.

For all implementations minimal battery usage will be a top priority, as the application will be running in the background all the time. Depending on the Opps which are enabled, the application will determine what data to collect.

The goal implementation for Opportunity involves a mobile application developed for iOS. The server and database will be deployed onto a DigitalOcean Ubuntu server 14.04 virtual machine which has 1GB memory, 1 CPU core, and 30GB SSD storage. The server can dynamically scale to accommodate a large flux in demand. The response time to the server should be less than 2 seconds. For the goal implementation, security will be a requirement since the application will have users sharing data and their locations (if the user wants to share that data) as well as the information on the servers will have to be secure. For the stretch goals contacts (users meeting up to do things), hours of stores and general locations will be added to the Opps settings. The stretch goal will have the application using stores' inventories, sales, nearby events, and ski resort snow levels being added as conditions for the Opps.

System Model

This section describes some simple usage scenarios for which a normal user may experience. Furthermore, some UML diagrams are provided to demonstrate how the system is laid out and how it functions.

Use Cases

Below are some sample use cases for which FixCode anticipates the average user will encounter. The Opportunity app naturally requires that external parties provide data which may be used to provide the user with timely notifications based on their surroundings. The use cases provided below reference the term actor frequently. In this context an actor is any external party that has influence on the behavior of Opportunity.

Such actors include the following

- Weather API
- Ski Resort API
- Event API
- Calendar API
- Google and Twitter OAUTH APIs
- Location service

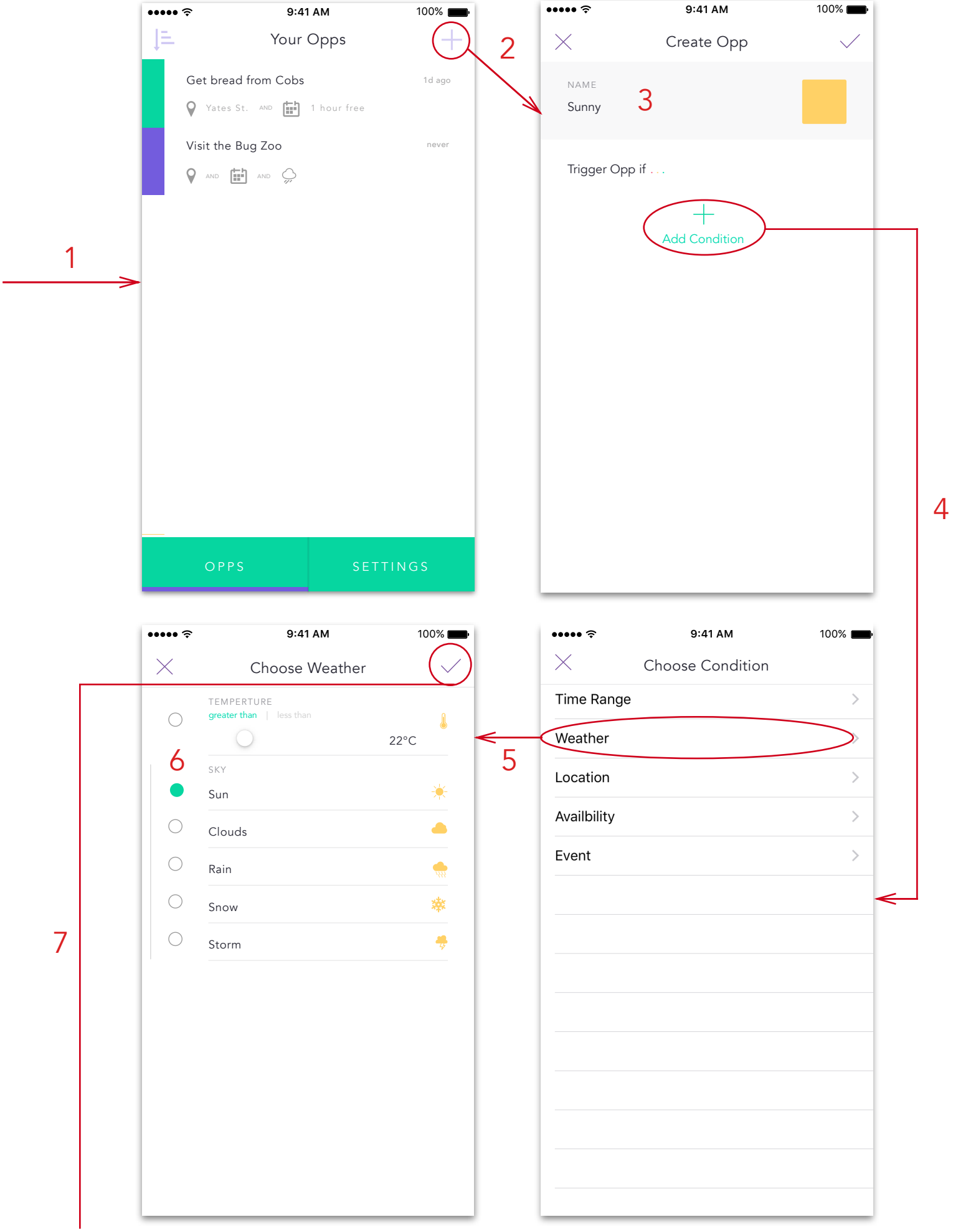
The weather API (open weather map) will affect the app by letting the app know the weather conditions for a given time. The ski resort API (weather2) affects the app by sending information (snow report and weather) about a specific ski resort. The event API (EVENTful) will send information about local events when the application requests for this information. A calendar API (Google Calendars) will send information about when the user has an appointment from which Opportunity will calculate when the user is free. Opportunity will allow users to sign in using Twitter or Google account or to create a new account specifically for Opportunity using the user's email. Opportunity is also affected by the location of the phone, as location data is retrieved using a the built in location services.

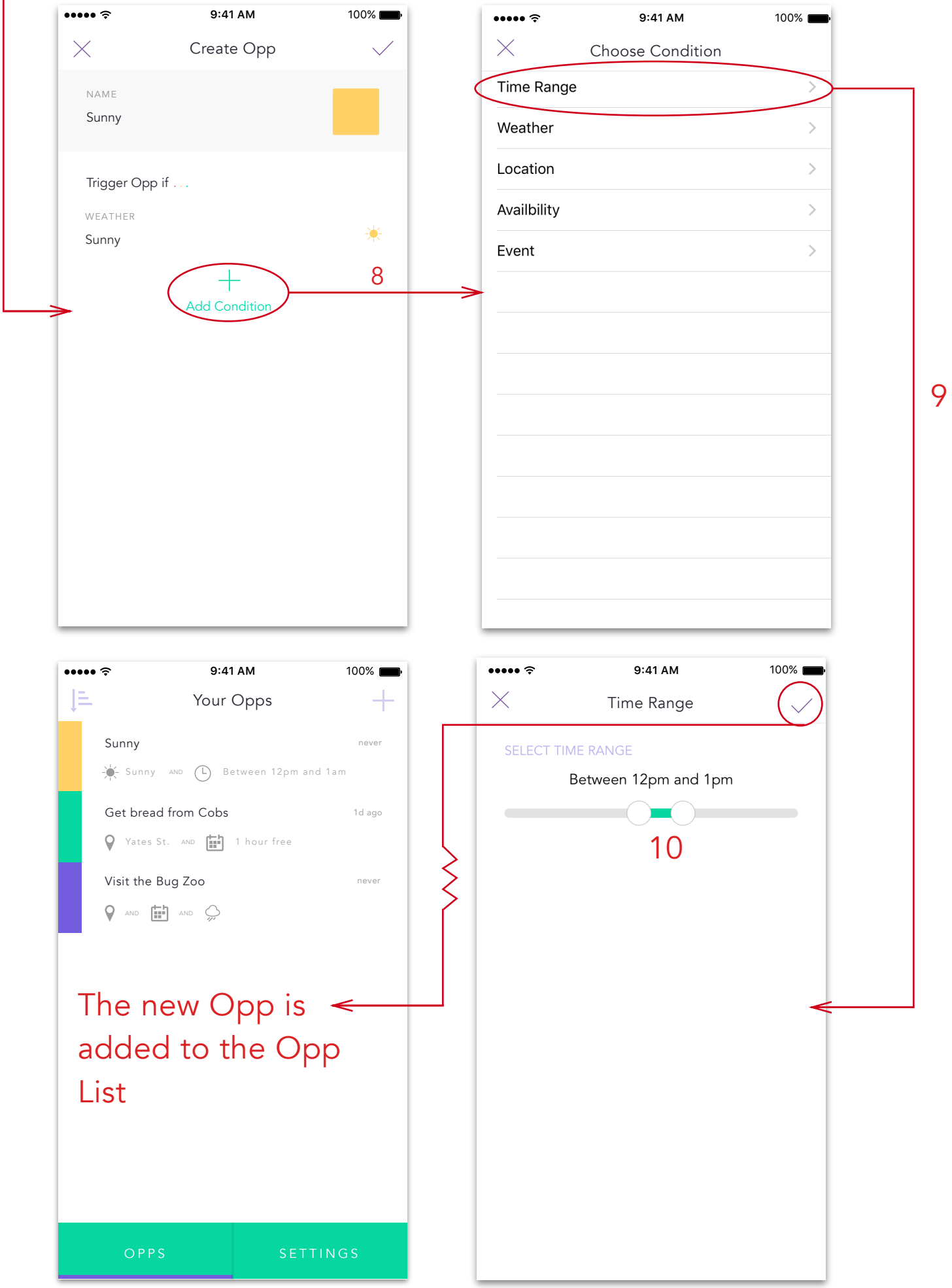
Use Case 1

ID	CREATE AN OPP
EXAMPLE	User wants to create a new Opp to be notified when the weather is sunny at the noon hour
ACTORS	Weather API
PRECONDITIONS	The user has already created an Opportunity account
BASIC STEPS	**1. The use case begins after the user has already opened Opportunity.** **2. User selects (+) allowing them to create a new Opp.** **3. Names their new Opp 'Sunny Day'.** **4. User selects the (+) 'Add Condition' button.** **5. User selects 'Weather' from the available conditions.** **6. User selects 'Sunny' as the weather state.** **7. User selects the (✓) to add the condition.** **8. User selects the (+) 'Add Condition' button again.** **9. User selects 'Time range' from the available conditions.** **10. The user adjusts the time range slider to read "Between 12pm and 1pm"** **11. User selects the (✓) button again to finish creating the Opp.**
POST CONDITIONS	The user has created an Opp, they will be notified if it is sunny and between 12pm and 1pm.

A user interface walkthrough is provided on the following page for this use case.

Walkthrough for use case sunny and between 12pm and 1am




9:41 AM
100%
Create Opp
NAME
Sunny
Trigger Opp if . . .
WEATHER
Sunny
Add Condition
8
Choose Condition
Time Range
Weather
Location
Availbility
Event
9
Your Opps
Sunny
never
Sunny AND Between 12pm and 1am
Get bread from Cobs
1d ago
Yates St. AND 1 hour free
Visit the Bug Zoo
never
AND AND
The new Opp is added to the Opp List
OPPS
SETTINGS
Time Range
SELECT TIME RANGE
Between 12pm and 1pm
10

Use Case 2

ID	ADD A CONDITION TO EXISTING OPP
EXAMPLE	User wants to add further conditions for their local ski resort
ACTORS	Weather API
PRECONDITIONS	An existing use case exists for new snow at a ski resort, user logged in
BASIC STEPS	**The user selects the existing Opp from their list of Opps.** **The user clicks the (+) add condition button,** **Then selects "Weather" from the options.** **From the available choices the user selects "Sunny"** **User selects (✓) to add the condition** **The user taps on the title of the Opp to change the name to "Blue Bird Powder Day"** **User selects (✓) again to update their changes to the Opp**
ALTERNATIVE STEPS	3) User can select any type of condition at this step
POST CONDITIONS	The existing Opp has a new condition in its triggers, it now must have snowed recently and be currently sunny.

Use Case 3

ID	LOCATION TRIGGER
EXAMPLE	A location-based Opp is triggered near a grocery store
ACTORS	Location Services
PRECONDITIONS	The user has created an Opportunity account, and an Opp to be triggered when they are near an appropriate store exists
BASIC STEPS	**1. User is near an appropriate store** **2. Opp's preconditions are met** **3. A Notification is created** **4. The user expands the notification drawer on their device** **5. <user's response> User acknowledges the Opp and dismisses it by selecting 'dismiss'**
ALTERNATIVE STEPS	At <user's response> the user could alternatively choose to 'snooze', allowing the Opp to trigger again the next time the conditions are met
POST CONDITIONS	User has been reminded that they are near a grocery store

Use Case 4

ID	DELETE
EXAMPLE	User wants to delete an Opp they have previously created
ACTORS	None
PRECONDITIONS	The user has a previously created Opp
BASIC STEPS	1. **User selects the Opp they want to delete** 2. **User selects 'delete', destroying the Opp** 3. **The user observed that their Opp is no longer present in the Opp list**
ALTERNATIVE STEPS	2) The user may also disable the Opp by selecting 'disable'
POST CONDITIONS	The user has deleted an Opp

Use Case 5

ID	SIGN UP
DESCRIPTION	User wants to create a new account after downloading Opportunity
ACTORS	Google or Twitter OAUTH API
PRECONDITIONS	The user has downloaded Opportunity
BASIC STEPS	**1. User opens Opportunity and selects 'sign up'** **2. <Sign in method> User chooses to sign up with email and enters their name, email, and desired password** **3. The user is presented with a tutorial on the basic usage of the Opportunity app**
ALTERNATIVE STEPS	At <Sign in method>, user could alternatively choose to sign in using a Google or Twitter account
POST CONDITIONS	The user has created a new Opportunity account

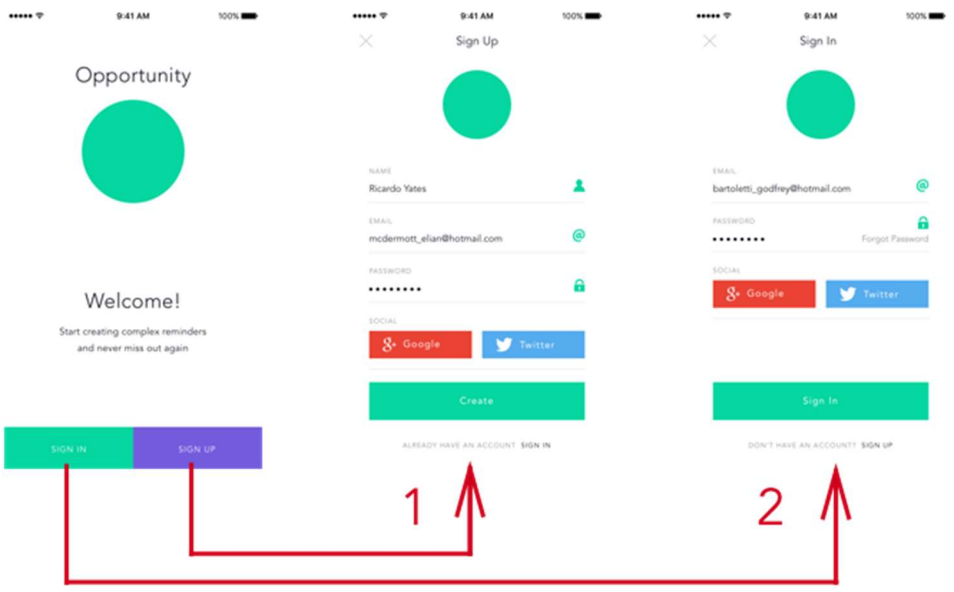


Figure 1 – Example sign-up user interface

UML Diagrams

UML is used as a tool to represent the structure and behavior of portions of the Opportunity system. The following diagrams are provided:

- Opp Dependency Component Diagram
- Opportunity App Class Diagram
- Opp Trigger State Sequence Diagram
- Opportunity Deployment Diagram

Opp Dependency Component Diagram

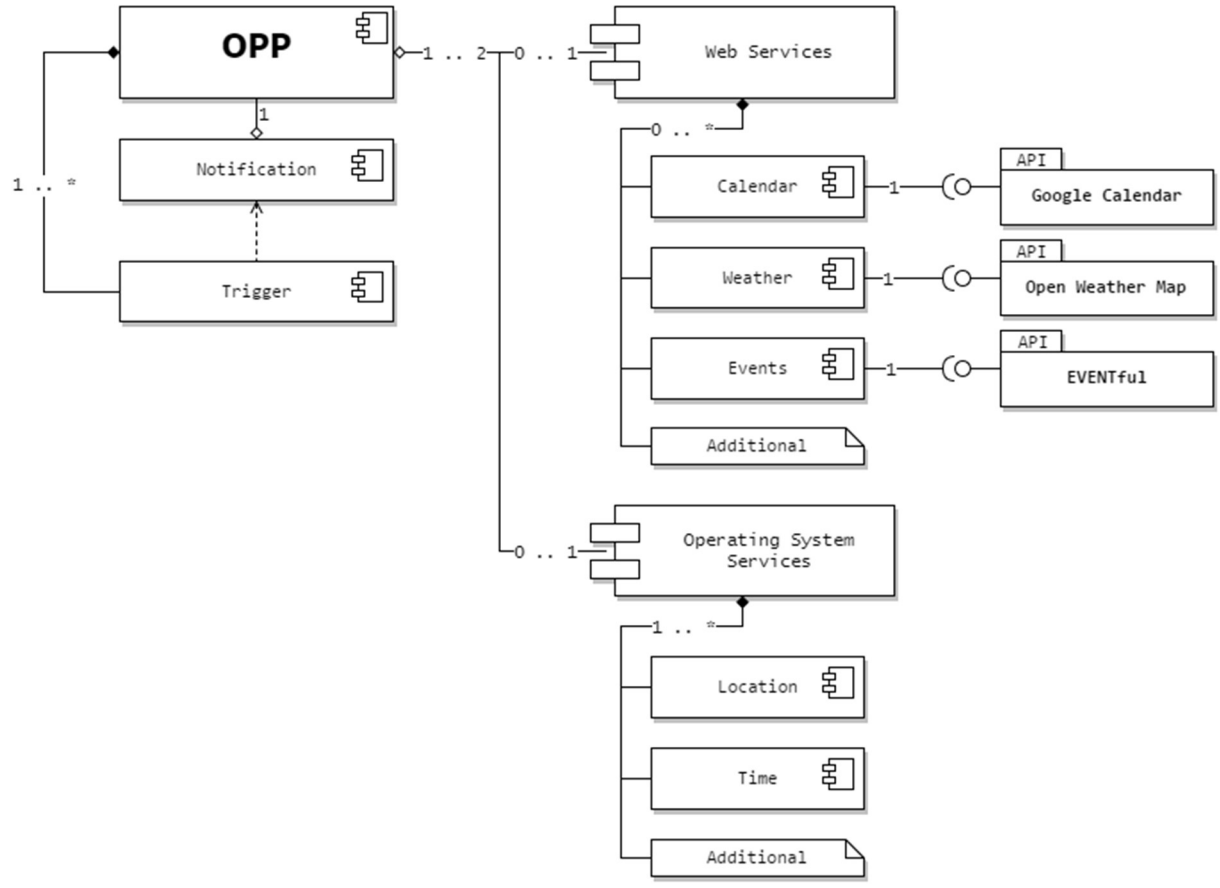


Figure 2 – Component relations for an Opp

Figure 2 depicts the relations of high level system components that an Opp utilizes and depends upon. To summarize the diagram, an Opp is an entity that may produce notifications. The state of an Opp is determined by the triggers it contains. State checking is elaborated in the Opp Trigger State Sequence Diagram section. To check the states of a trigger, the Opp depends on either or both of the services encompassed in the Operating System or Web Service components. These components provide methods to check the state of a trigger. Services

wrapped by the Web Service component further depend on functionality provided by an internet API.

Opportunity App Class Diagram

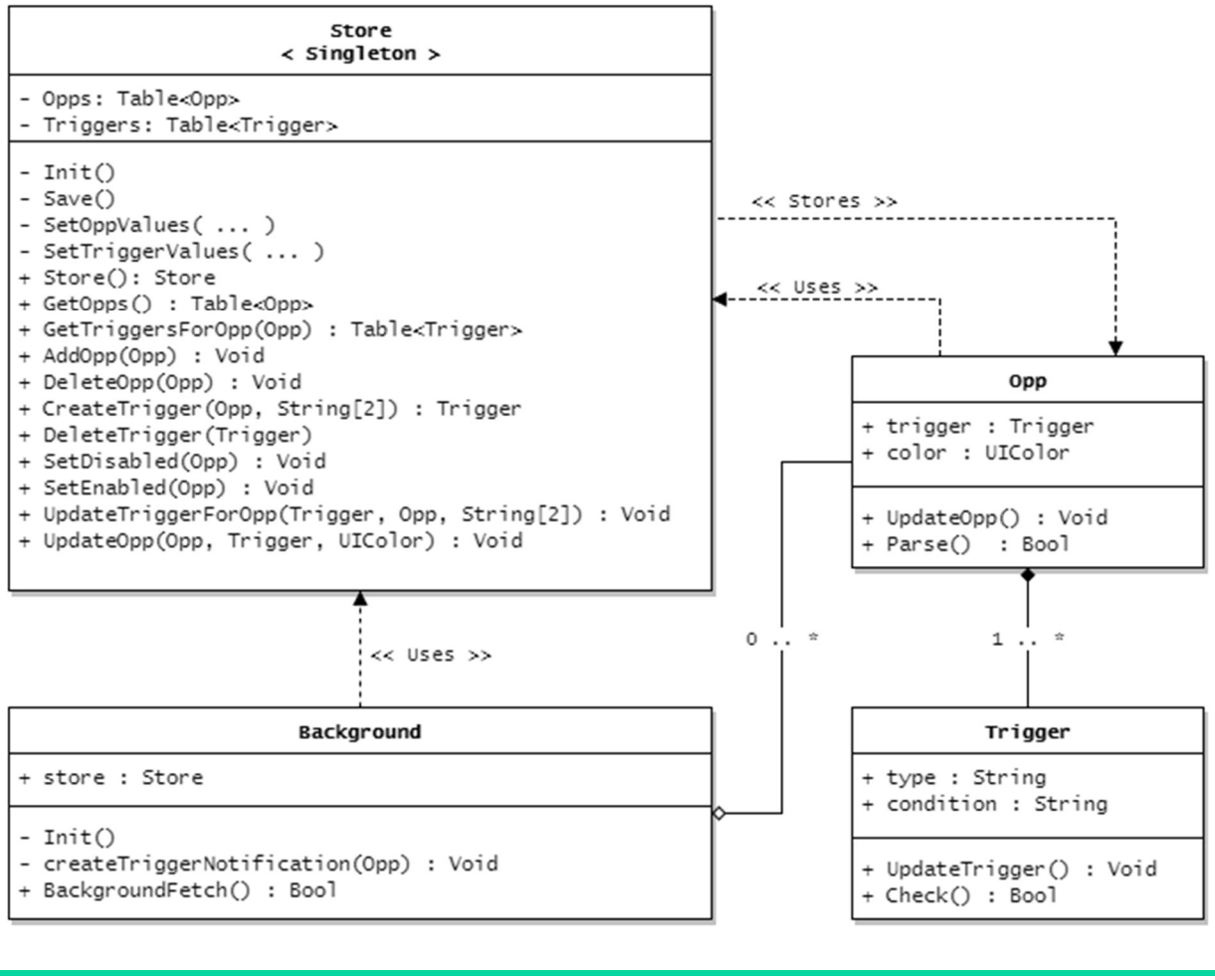


Figure 3 – Opportunity App functional class diagram

The diagram in figure 3 shows the classes and their fields and methods. The store class is persistent, as this is the class that manages local storage for Opps and triggers, it behaves in accordance to the singleton pattern; any other class is able to obtain its instance. The trigger class is part of the composition of the Opp class, in that an Opp is useless without at least one trigger. The background class is responsible for checking the state of an Opp when the Opportunity application is not in the foreground. Note that this diagram does not depict implementation classes used in support of the iOS platform, such as view controller and visual component classes.

Opp Trigger State Sequence Diagram

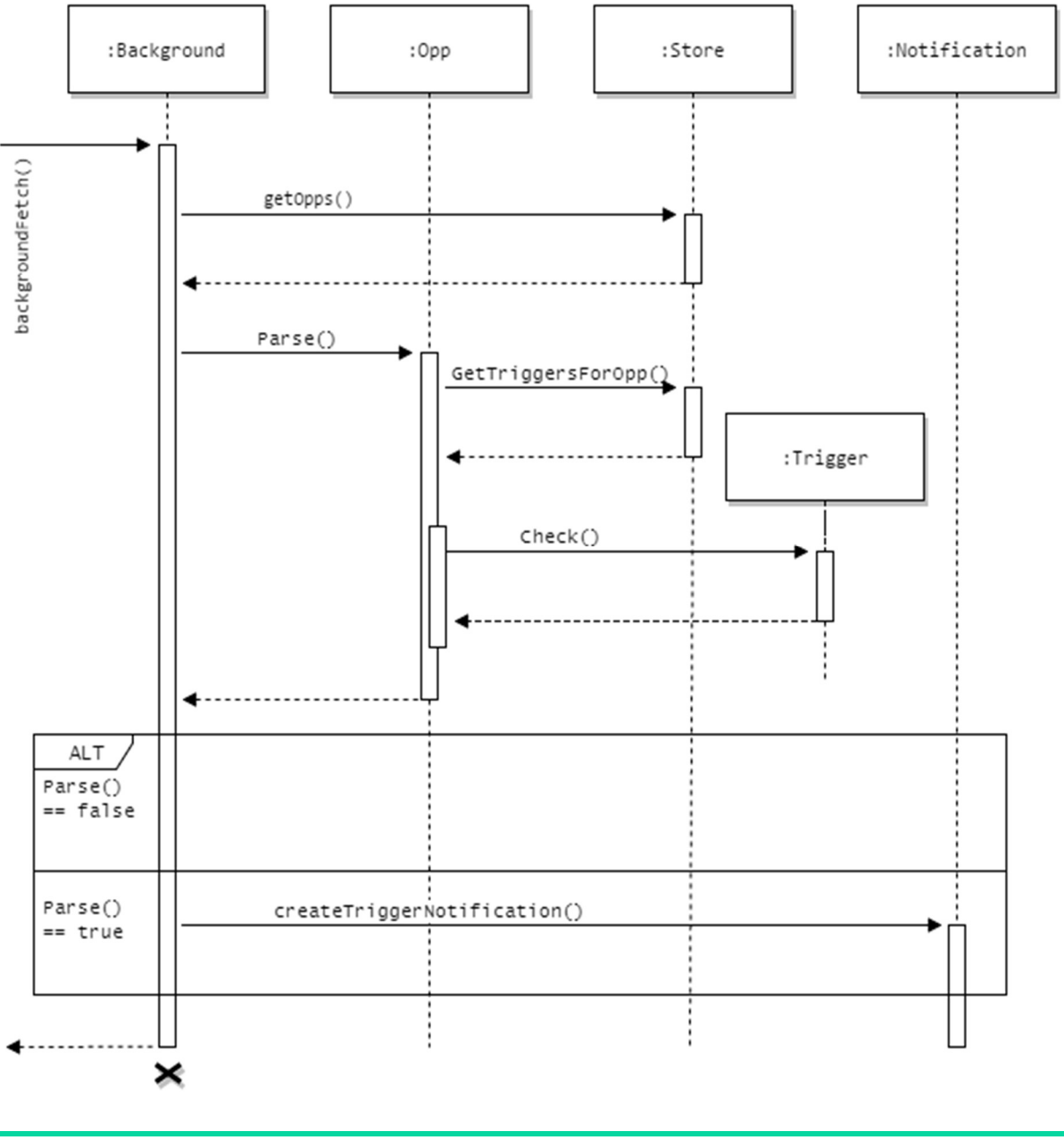


Figure 4 – Trigger checking sequence diagram

The diagram in figure 4 shows the sequence which occurs when the system invokes the background class. This occurs when the system allocates CPU time to Opportunity app when it is not in the foreground, at time the `backgroundFetch()` function is invoked and acts as the entry point at which Opp triggers can be checked. This may occur via push notification from either of the Service Components (Web and Operating System) or on a set interval timer. The background class calls `opp.Parse()` for each of active Opps returned by `store.getOpps()`, if this

function returns true, all of the Opp's triggers have been met, and a notification may be produced.

Opportunity System Deployment Diagram

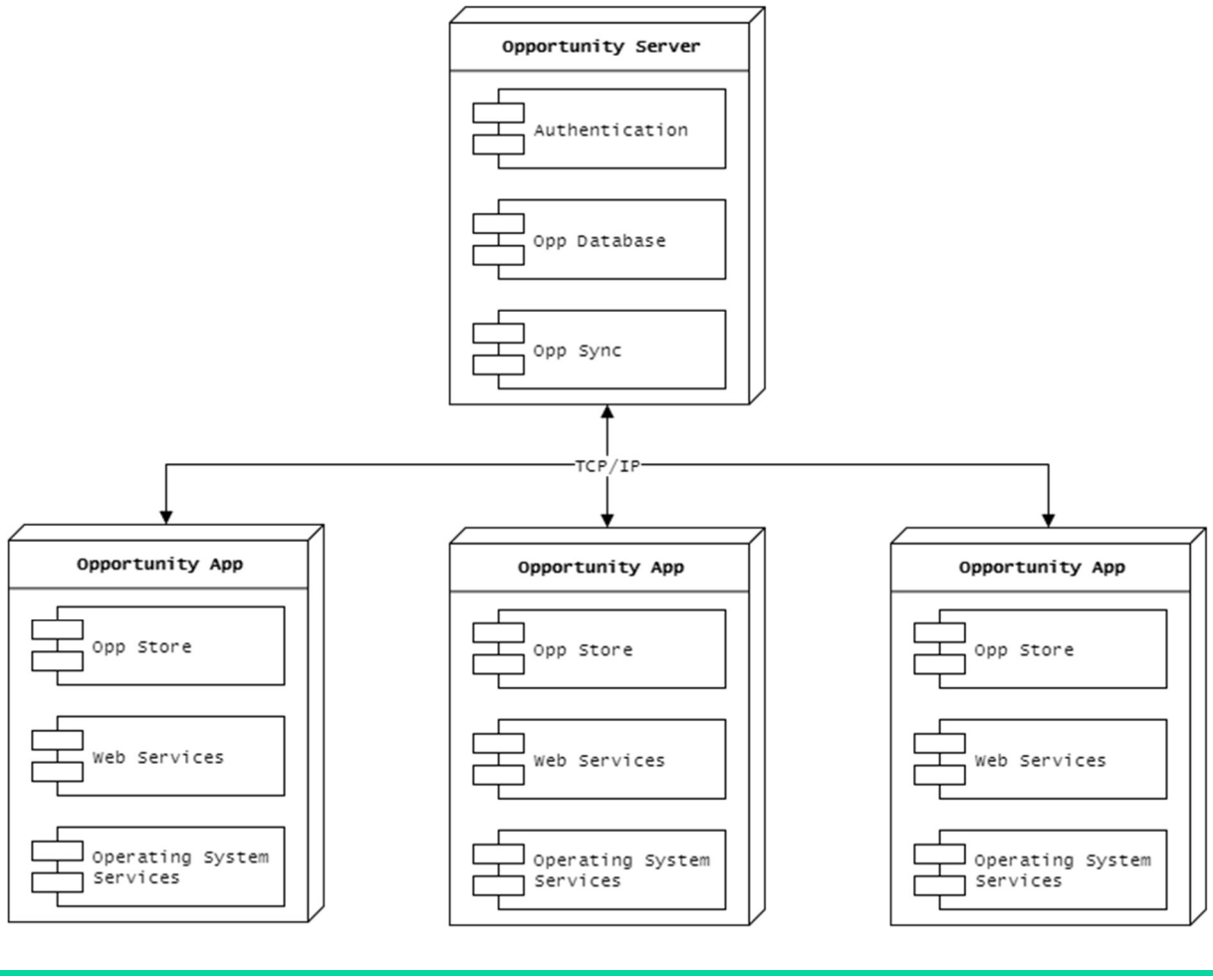


Figure 5 – System deployment diagram

As discussed in the Management Plan section below, the ideal deployment of the Opportunity app involves a central server to allow users to create accounts with Opportunity such that they can back up their Opps from devices they own and synchronize Opps across multiple devices. The minimal system involves only deploying the "Opportunity App" component seen in the deployment diagram in figure 5.

Pseudo Code

Creating Opp

To create an Opp, a type, message, colour, and value needs to be passed in.

```
PROCEDURE createOpp(type, message, value, colour)
    SET opp to CALL dataContext.opps.createEntity(type, message, value, colour)
    CALL dataContext.save()
END PROCEDURE
```

Fetching Opps

Opps can be fetched using a sort descriptor which describes how the returned array is ordered. If the descriptor is not specified, the procedure defaults to "Name"

```
PROCEDURE fetchOpps(sortDescriptor = "Name")
    RETURN CALL dataContext.opps.sort with sortDescriptor
END PROCEDURE
```

Background Fetch

The following sections of pseudocode are used to parse Opps and trigger notifications in the background.

This procedure runs every time the OS wakes the app in the background. This happens periodically. Since checking Opps requires making network requests, this procedure is asynchronous and returns a promise object. If all conditions belonging to an Opp pass, a local notification will be created for that Opp. The background fetch will alert the OS it has finished its operations only after the promise has been resolve.

```
PROCEDURE backgroundFetch -> RETURNS Promise
    SET promise to Promise
    SET opps to CALL store.oppsNotDisabled
    SET oppPromises to []
    SET triggered to FALSE

    FOR opp in opps
        APPEND CALL parseOpp with opp to oppPromises
    ENDFOR

    WHEN each Promise in oppPromises is RESOLVED with oppResults
        FOR result in oppResults
            IF result
                CALL triggerNotification with opp
                CALL setOppRead with opp, FALSE
                SET triggered to TRUE
            ENDIF
        ENDFOR
        RESOLVE promise with triggered
    ENDWHEN
    RETURN promise
END PROCEDURE
```

This procedure parses an Opp and will resolve to true if a notification should be triggered.

```
    PROCEDURE parseOpp(opp) RETURNS Promise
        SET promise to Promise
        SET conditions to store.getConditionsForOpp with opp
        SET conditionsPromises to []

        IF conditions.count is 0
            APPEND CALL checkCondition with condition to conditionsPromises
        ENDIF

        WHEN each Promise in conditionsPromises is RESOLVED with condResults
            FOR result in condResults
                IF NOT result
                    RESOLVE promise with FALSE
                ENDIF
            ENDFOR
            RESOLVE promise with TRUE
        ENDWHEN
        RETURN promise
    END PROCEDURE
```

Triggering Conditions

The opportunity app gains its functionality from various types of triggers, Opps are executed when its trigger becomes active. Below are some of the basic triggers, and their trigger conditions.

Location

The location logic is mostly handled by the operating system. When a user creates an Opp with a location Condition, a geofence is made around the latitude, longitude, and radius the user specified. When the geofence is entered, the Opp is parsed and checked if it's other Conditions pass.

```
PROCEDURE locationTriggered(opp)
    CALL parseOpp with locationPassed = TRUE
END PROCEDURE
```

Time Range

When Opportunity checks a time range Condition, it simply checks if the current time is between the user specified bounds.

```
PROCEDURE checkTimeRange(condition)
    SET lowValue, upValue to CALL ConditionParser.parseTimeRange with condition.value

    SET hour24 to Now.get24Hour
    IF lowValue is 0 AND upValue is 0
        RETURN TRUE
    ENDIF

    IF hour24 >= lowValue AND hour24 <= upValue
        RETURN TRUE
    ENDIF

    RETURN FALSE
END PROCEDURE
```

Weather

Weather is fetched using a Get request to the OpenWeatherMap API. The results are returned in JSON format which is then parsed and compared with the user's weather Condition. If the weather has already been fetched from another Opp in the same background fetch, the events are retrieved from the cache.

```
PROCEDURE checkWeather(condition)
    SET weatherResult to CALL getWeather with currentLocation
    SET condSign, condTemp, condSky to ConditionParser.parseWeather with cond.value

    SET shouldPass to TRUE
    IF condTemp not NULL
        IF NOT weatherResult.main.temp (condSign) condTemp
            SET shouldPass to FALSE
        ENDIF
    ENDIF

    IF condSky not NULL
        IF NOT condSky == weatherResult.weather.main
            SET shouldPass to FALSE
        ENDIF
    ENDIF

    RETURN shouldPass
END PROCEDURE
```

Availability

The user's calendar events are obtained from making a Get request to Google's API with a user authentication key. If the events have already been requested from another Opp in the same background fetch, the events are retrieved from the cache.

```
PROCEDURE checkAvailbility(condition)
    SET googleEvents to getEventsForToday
    SET shouldPass to TRUE
    SET userAvailbility to ConditionParser.parseAvailbility with condition

    FOR event in googleEvents
        IF conflictingEvent with event, userAvailbility
            SET shouldPass to FALSE
        ENDIF
    ENDFOR
    RETURN shouldPass
END PROCEDURE
```

Management plan

There are four main components required for the goal implementation of the Opportunity system. These components are:

- mobile application
- backend server
- database
- website

The user will operate Opportunity through the mobile application. The app is where the user will create and login to an account, create Opps, and be notified of triggered Opps. The application will contact the server through a REST API to authenticate users and create and delete Opps. The server will store user and Opp information in a database. Details on these components are detailed below.

Mobile Application

The application will be designed using Sketch, an interface design tool, and prototyped using Invision, a prototyping platform. Using these tools before actually coding the app will allow us to quickly iterate between designs to efficiently decide on the design with the best user experience.

The mobile application will be developed for the IOS platform. The currently designed app interface can be seen in the UI walkthrough. An Android application may also be made if time permits. The IOS app will be developed using the Swift programming language on the XCode development IDE. The interface will be created using XCode storyboards and coded using view controllers for each view. User info and Opps will be stored locally on the device using a core data database. The app will use background app refresh to wake the app up at specific times, to check the weather, determine the location, and check availability on the user's calendar. Each time background app refresh wakes the app the device's battery will be drained. Therefore, it is important to only wake the app when absolutely necessary. The app will communicate with the

server via a REST API. Using GET and POST requests to specific endpoints, the app will upload user and Opp data to the server.

User Permissions

Since the mobile application heavily relies on system notifications and the user's location, both of which require user permissions, special care has been taken to ensure the user knows why we are requesting access to these features. Figure 6 shows what is presented to the user when they first launch the app. If the user enables both features this screen will never be shown again. If the user closes the popup or at some time revokes Opportunity's access to notifications or location, this popup will appear on next app launch, explaining to the user that these permissions a required for Opportunity to correctly run. Showing the user this popup instead of immediately requesting permissions provides a better user experience as the user does not feel as overwhelmed or pressured to give a potentially unknown app permission to track their location. This popup is friendly and kindly explains what this app needs, allowing the user to enable permissions when they are ready.

The application will require the following permissions to be accepted by the user.

Location services

In order to provide the user with location based Opportunity notifications, the application should be able to access the device location in the background.

System notifications

Notifications are the fundamental service used to communicate the availability of an Opportunity to the user when the Opportunity app is not in the foreground.

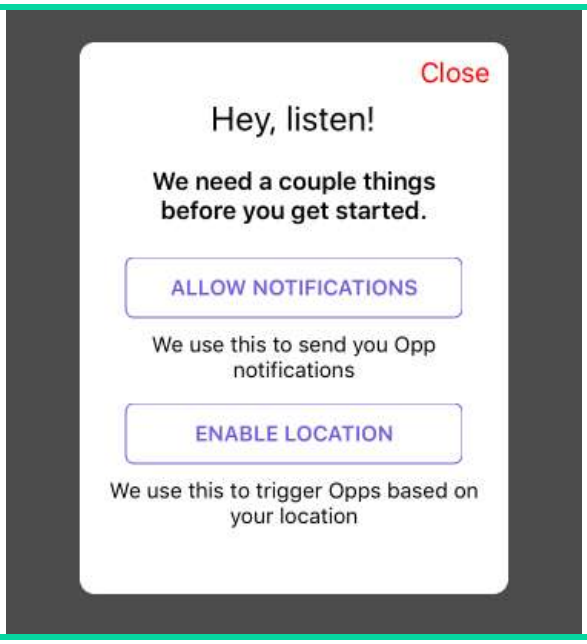


Figure 6 – User Permission Warnings

App Icon

The app icon was created based off the Opportunity logo and is seen in figure 7. It follows the style of many other popular IOS apps available, such as YouTube, Reddit, Photos, and Safari. The icon can be seen on the IOS home screen in figure 8.

Figure 7 – Application icon

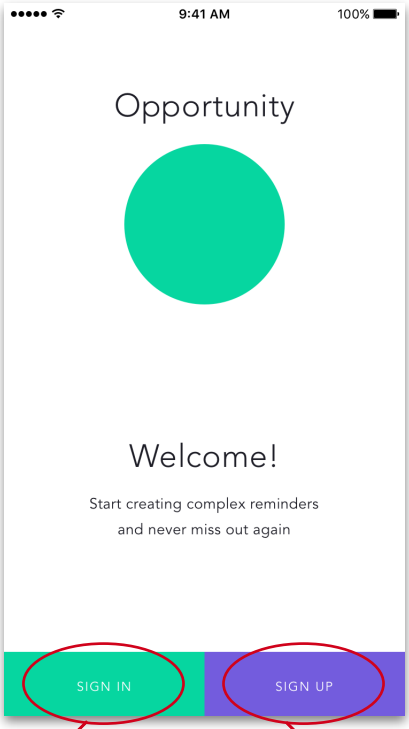
Figure 8 – Application icon on home screen

Application Walkthrough

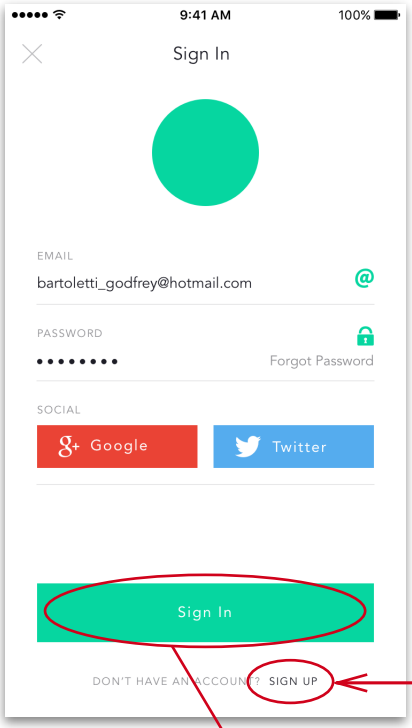
The following pages contain a walkthrough showing what is currently implemented in the mobile application.

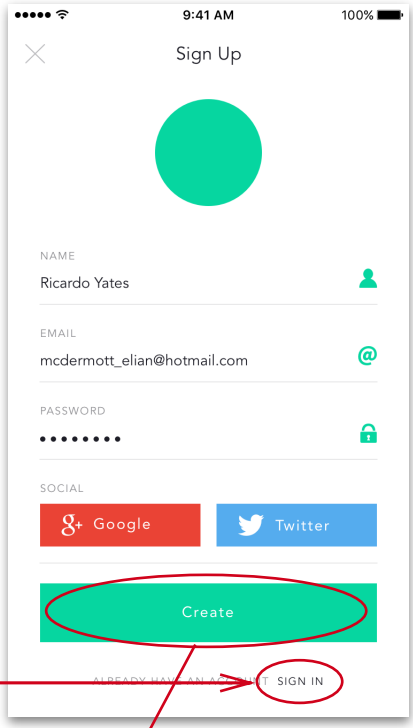
Opportunity app walkthrough

The first page to load when the app is installed or the user is not logged in



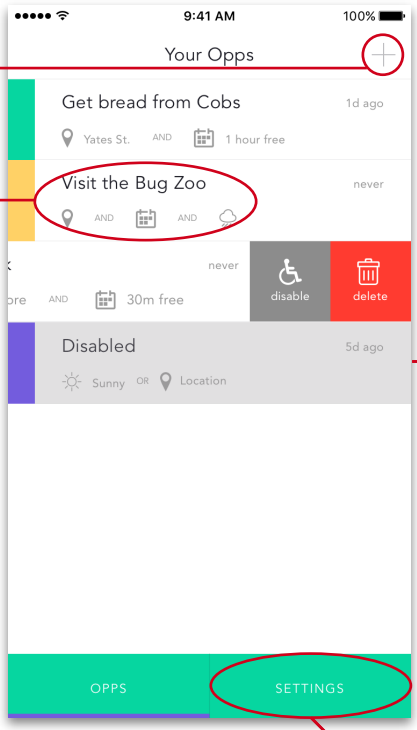
The user can sign in with email and password or Google or Twitter





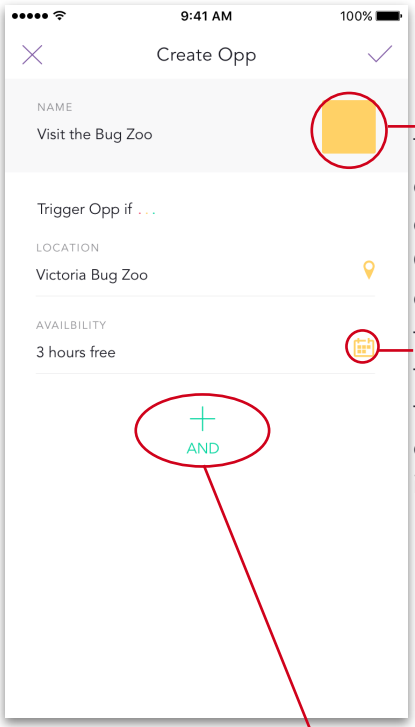
The user can sign up with email and password or with other social platforms

The first page to load when the user is logged in. A list of all the users Opps are shown here

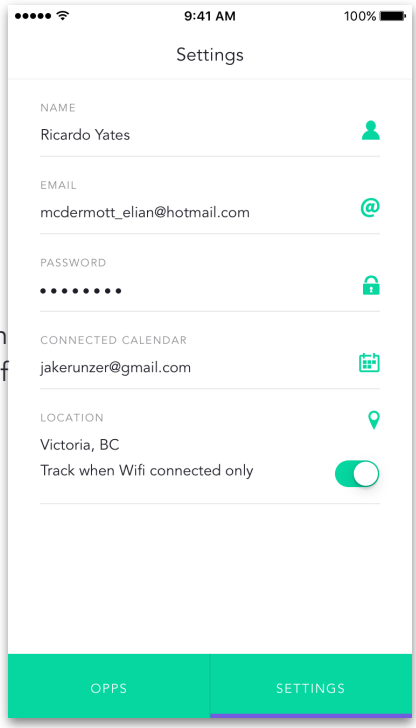


A disabled Opp will not be triggered

The edit and create Opp page. Here the user can add conditions to the reminder which will determine when the Opp is triggered

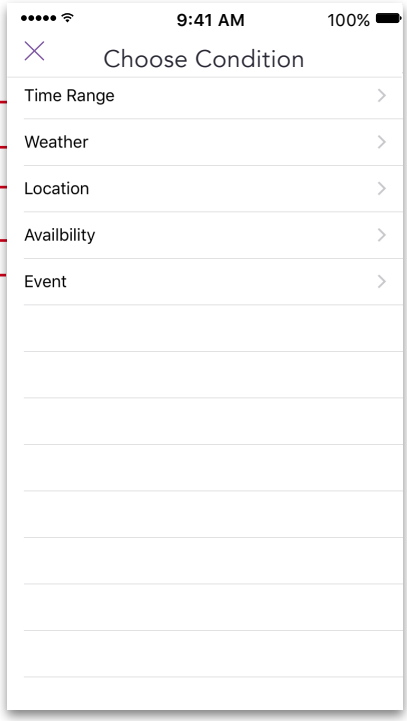


The user can change the colour of the Opp. This colour will follow through the creation of the Opp changing the icon colours

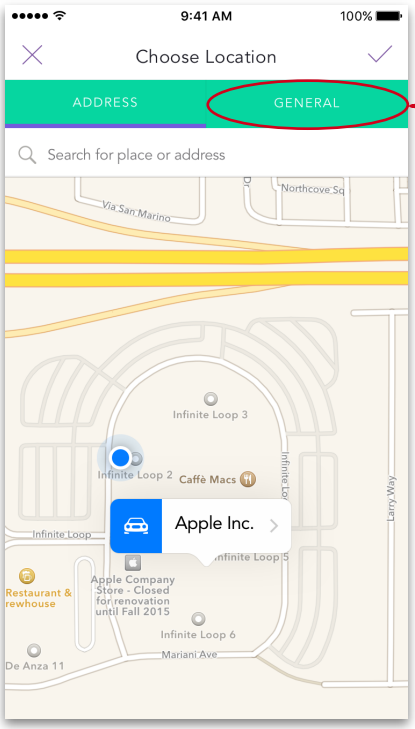


The settings page where the user can

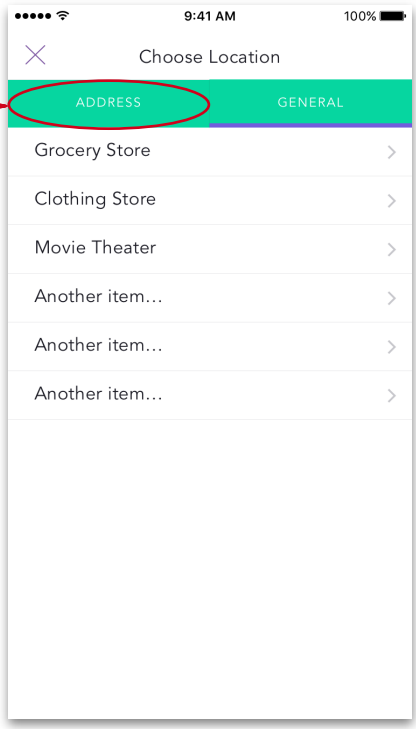
- edit account information
- connect a calendar
- change location
- set to track while connected to Wifi only



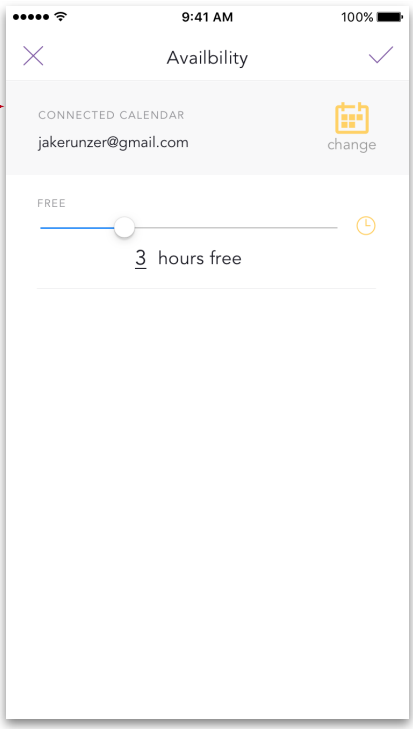
Here the user can choose what condition to add to the Opp



The user can enter an address. The Opp will be triggered when the user is near this location

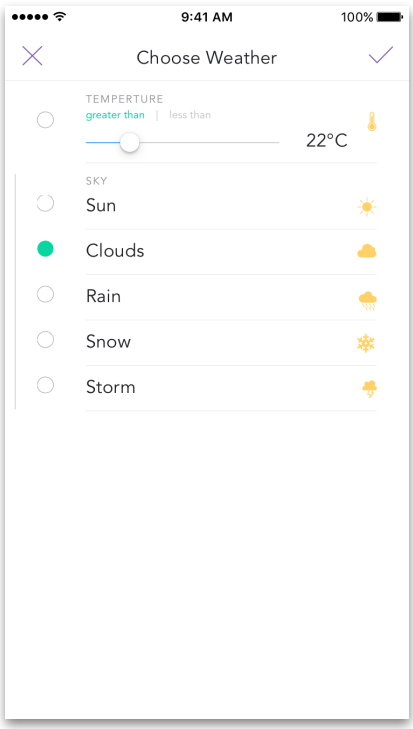


Here the user can choose a general location to trigger the app



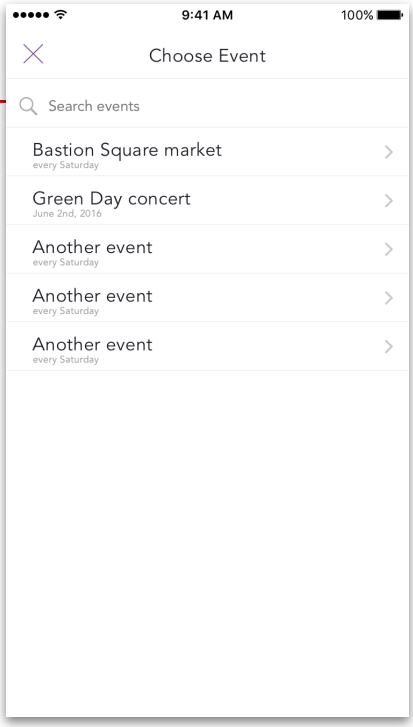
Here the user selects how much free time on their calendar will trigger an Opp

The connected calendar can be setup or changed here aswell

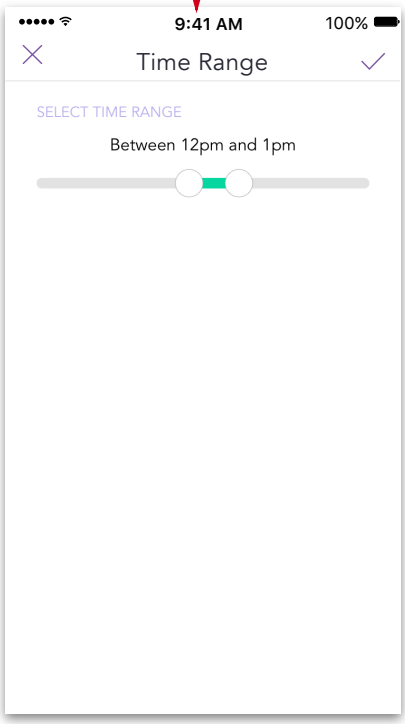


The user can set to trigger an alarm based on temperture (greater or less than) or by the status of the sky (sunny, cloudy, etc.)

The sky section is a radio group, where only 1 option can be selected



The user can search for and select events that will trigger the Opp



The user can use the double ended slider to select a time range

Opps

Opps are created and triggered using the mobile application. This section will detail how each condition in the Opp will be technically implemented. Many of the conditions require waking the app in the background periodically to check data. These checks will be squished into one check for all Opps to save device battery.

Location

There are two components to location in Opportunity. The first is a specific address the user wishes to be notified at. When the user has an Opp with a location condition, the app will use the device functions to be woken when the device approaches a specific location. Device battery will drain when the location is checked with GPS, so the location checks for all Opps that use location will be combined into one. The second component of location is to trigger an Opp when the user is near a general location, such as grocery store, clothing store, or gas station. This is not a minimum requirement as a more complex use of Google or an external API is needed to find a list and location of stores and businesses.

Weather

We are using an external API to query for weather data for the user's location. The API we are using is the OpenWeatherMap API (http://openweathermap.org/api). The free plan allows for 60 request per second and 50 000 requests per day. This is within our expected demand. When the user creates an Opp that includes a location condition, the app will periodically check the weather at the user's location to see if they meet the Opps conditions.

Availability

In the app the user will be app to connect a Google calendar. Authentication with Google will be done using secure OAuth2 using pre-existing libraries. The app will refresh the calendar once every hour to pull in new events. Within the app, the calendar events will be analyzed against the user's Opps to determine if the conditions are met.

Events

We will be using the Eventful API (https://api.eventful.com/) to get event information. This API will provide the app with information about concerts, festivals, sports, etc. The user will be able to search for upcoming events and the application will alert the user near the time of the event.

Server

The server will be implemented if time permits. It is not an essential part of the system as the mobile application can function on its own. The server will be implemented using the Python programming language using either the Django or Flask framework. The server should be lightweight as it solely serves a REST API.

Authentication Server

authentication is one of our stretch goals. To implement this a backend server is required to make API calls to. The user will either create an account with social platform Google or Twitter, or with email and password. The server will generate the client with a token when valid credentials are provided. This token will be used in the header of all requests that change user Opps or account information. Passwords on the server will be encrypted with bcrypt, an irreversible encryption algorithm.

Opp Sync

The data for authenticated users will be stored on the server in a database. This data will be synced to the user's device to make sure they have the most up to date data. Storing user data on the server will allow multiple devices to be used for the same account. Also, if the user's device is lost or broken, they will easily be able to restore all data to a new device.

Database

The database will be SQLite. SQLite is a lightweight relational database that will allow us to quickly setup and configure it. We are not expecting to be storing large amounts of data so a more advanced database (PostgreSQL, MySQL, etc...) is not needed. The database will store user and Opp information which will be served by the server. The database will be hosted on the same machine as the backend server.

Website

The website is a small component of the system. It will describe what the app is and have a link to the app store where you can download the app. There will be screenshots of the interface and short descriptions of all the functions. This will be the main place to market the Opportunity system.

Testing

The following section details the types of tests we will run to verify that the system meets the specification. The various tests have been created to ensure all aspects of Opportunity are functioning as intended, and the interface is user friendly and manageable.

Timeline:

Unit Tests	
Completed by	Jake Runzer
Date Completed	Automated, run daily
Individual Tests	Fetching weather data
	Getting user's location
	Searching location with text strings
	Fetching google calendar events
	Parsing conditions

Integration Tests	
Completed by	Dylan Golden
Date Completed	Automated, run daily
Individual Tests	Creating an Opp with each a single Condition (repeated for each Condition)
	Creating an Opp with all possible combinations of Conditions

UI Tests	
Completed by	Claire Champernowne
Date Completed	14/04/2016
Individual Tests	Time range double ended slider
	Opps/History tab
	Selecting pins on the map view
	Add condition button on edit/create Opp page
	Editing Opp name
	Deleting Opps from `tableview` using swipe method
	Disabling Opps from `tableview` using swipe method
	Deleting Conditions from edit/create Opp page
	Sky button radio group
	Temperature slider and less than/greater than buttons
	Availability logarithmic slider
	Opps sorting button

Performance Tests	
Completed by	Zev Isert
Date Completed	14/04/2016
Individual Tests	System resources
	Battery life
	Requests per second
	Response time

Unit Tests

Unit tests will be automated using the XCode IDE. All tests are performed each time the app is built or changed. This lets us know immediately if a new change to the code is breaking an existing feature. With this system, once a feature is implemented, it is guaranteed to work in future versions of the app. Our unit tests will test the following features of the Opportunity application:

- Fetching weather data
- Getting the user's location
- Searching location with text strings
- Fetching Google calendar events
- Parsing conditions

Fetching Weather Data

We are using the OpenWeatherMap API (http://openweathermap.org/current) to fetch weather data for a specific latitude and longitude. An example request is

```
http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139&appid=b1b15e88fa797225412
50c122a
```

The data from the fetch we are interested in is `weather.main` and `main.temp`. Weather is an array which provides information about the status of the sky (clear, clouds, rain). Main is an object which provides basic information about the request, such as latitude and longitude, and temperature in Kelvin. The unit test testing this feature will make subsequent requests to OpenWeatherMap and parse each result looking for these two properties and fail if they are not found.

Getting the User's Location

This unit test will simply fetch the device's current location. This test ensures we are able to read and parse a location. The user's location will be simulated using XCode debugging tools and then the code will fetch the user's location. This test will be successful only when the simulated location and fetched location match.

Searching Location with text Strings

When a user is creating a location condition, they are able to search with autocomplete for a specific address. Address lookup is done using Apple map kit local search requests. This process is done asynchronously off the main thread. The method used to lookup the search query will be tested using known address and coordinate pairs. The tests will only pass if an address lookup successfully returns the correct coordinate.

Fetching Google Calendar Events

The user's calendar is used to trigger Opps based on the user's availability. Fetching data from Google calendar requires OAuth authentication. Authentication is done per user.

Testing of the Google calendar API will be done using a predefined account with predefined calendar events set. The tests will include querying the Google API for calendar events on a specific day which will be compared with known events for that day.

Parsing Conditions

Conditions are stored in a `CoreData` database on the user's IOS device. There is a one-to-many relationship between Opps and Conditions. Condition values are stored in a formatted string property in the Condition. There is a parser for each Condition type whose purpose is to parse the formatted string into the expected Condition object. We will have unit tests which create Conditions of each type from objects and then parse these Conditions back into objects. The tests will only be successful if the parsed Condition object matches the respective input object.

Integration Tests

Integration tests combine several modules and tests their functionality as a whole. These tests will ensure the modules are compatible with each other and that data is able to flow from module to module successfully. Integration tests will test the following app functionalities

- Creating an Opp with each a single Condition (repeated for each Condition)
- Creating an Opp with all possible combinations of Conditions
- Attempting to trigger a disabled Opp
- Attempting to trigger non-disabled Opps

Creating Opps

To test the creation of Opps, the full process will be tested hundreds of times with various parameters at each step. The process to create an Opp will be tested as follows

1. Select random Opp name (use random selection of Unicode characters, including emojis)
2. Select random colour from five available colours
3. Select random number between 1 and the number of Conditions available
4. Create this number of Conditions attached to the Opp. The parameters for each Condition will be randomly selected to fully test everything the user could potentially enter.
5. Save the new Opp to the database to persist the changes
6. Fetch all the Opps and check if the new Opp is there

Triggering Opps

To test the triggering of Opps, Opps will be created that can go off immediately. Some of these Opps will be disabled and some will not be. The apps background fetch will

continuously programmatically be called. This integration test will only be successfully if all Opps that should trigger a notification pass each time background fetch is called and all disabled Opps are ignored each background fetch.

Deleting Opps

This integration test is very simple and simply tests if the user interface and the database are in sync. For a successful test, a number of Opps will be created with various conditions, the Opp count is checked and verified. Then, a single Opp or a number of Opps are delete and the Opp count is checked and verified after each deletion. The Core Data database should agree with the number of Opps we have created - deleted. Also, the number of Conditions in the database will be checked and verified as well. If an Opp with three conditions is deleted, all of those Conditions should be removed from the database as well.

UI Tests

UI Tests will be implemented to ensure all the controls the user interacts will perform their correct functionality. The main controls to test are

- Time range double ended slider
- Opps/History tab
- Selecting pins on the map view
- Add condition button on edit/create Opp page
- Editing Opp name
- Deleting Opps from `tableview` using swipe method
- Disabling Opps from `tableview` using swipe method
- Deleting Conditions from edit/create Opp page
- Sky button radio group
- Temperature slider and less than/greater than buttons
- Availability logarithmic slider
- Opps sorting button

The tests for the main application screens are detailed below.

Opps List

On the Opps list the user should be able to view a list of all the Opps they have created, disabled and enabled. There are two main buttons on this page represented by icons, which are “Sort Opps” and “Create Opp”. The Opps list UI test will only pass when the sort button correctly changes the ordering of the Opps and when tapping the “+” icon successfully navigates the user to the Create Opp page. Also, if the user taps on any of the Opps in the list, they should be navigated to the Edit Opp page with all of the name, colour, and Conditions prefilled.

Create/Edit Opps

From the create and edit Opp page, the user should see and be able to change the Opp name and colour. There is also a “Add Condition” button below the list of Conditions for the Opp. This UI test will only pass successfully if the Opp details are persisted to the database after the user sets them and taps the ‘Check’ button. If the user edits the Opp then taps the ‘Close’ button, the creation of the Opp should be cancelled or the Opp is returned to its previous state.

Create Condition

There is a specialized screen for each Condition the user can create. These include time range, weather, availability, and location. The functionality of each screen will be tested and only be successful if the user can intuitively create a Condition of that type. The controls for each Condition will be thoroughly tested. These include

- Time Range
  - Double-ended slider
- Weather
  - Temperature slider
  - Greater/Less than buttons
  - Sky radio group
- Availability
  - Connect Google calendar
  - Logarithmic free time slider
- Location
  - Text based address and location search and autocomplete
  - Displaying locations on map view
  - Map view panning

Performance Testing

Performance testing is the testing practice that allows developers to determine how a system performs in terms of criteria such as responsiveness and stability. For the opportunity mobile application, the performance tests below are intended to help identify bottlenecks in the system and as such are supplied more for the baseline development of a testing platform, than day to day testing. The testing environment for the following performance tests is detailed under the application and server side headings below.

Application Side

The complete implementation of the Opportunity system involves multiple components, one of which will be the mobile application installed on the user's mobile phone. Some base performance criteria that the application should adhere to are a minimal usage of system resources when running in both foreground and background, as part of this the application should also not be a major battery drain on the user's phone while running in the background. A typical test environment for testing application side performance will involve using XCode's debugging tools and a connected physical device to monitor dynamic performance measures.

System Resources

The amount of configurable information and size of user options is very low in the mobile application, therefore it is reasonable to expect low heap memory usage, low thread workload, low to medium CPU time, and medium network consumption. To test this in the case of the mobile application, some load tests may be performed. The model for testing system resource criteria will be based on the number of Opps the user has configured. Because there is no upper bound on the number of Opps a user may configure, the developers have chosen to release Opportunity on an Apple Test-Flight beta program, using the data from the beta program the average number of Opps configured can be found for a given time - since as more features are added, the number of configured Opps is expected to increase. In general, the number of configured Opps will fall in a Gaussian distribution, so 4 tests should be performed for each metric.

Zero Opp Test

This test has the goal of determination of the base memory, CPU, and network usage. Expected results are that the application uses no little to no memory, no background CPU, and no background network usage. Maintained CPU usage and memory in the foreground should be minimal as well, since without any configured Opps, there is no requirement for any activity aside from displaying the user interface.

Half average configured Opps test

This test used to determine the rate of resource usage between having no Opps and the average number of Opps configured. One should expect that the application uses more only slightly memory and CPU time, and only a small amount more, if any, network usage. All three metrics should be less than in the average configured Opps test case.

Average configured Opps test

This performance test is used to gain a feeling for the typical user experienced performance while using Opportunity. This test, like the others above, should also pass the requirements for performance testing, since this will be the test that shows typical performance.

Triple average configured Opps stress test

This test shall be the test that reaches the performance requirements. Under the Gaussian curve, triple mean covers 99.9% of the population. Therefore, if the triple average configured Opps test passes, requirements, we can be nearly certain that the system will perform as intended for nearly all users. Performance requirements in this case should show that Opportunity requires at most 5% of the available application memory, and has a background CPU time ratio of no more than 10%.

Battery Life

Battery impact was identified during design as a critical target to minimize for. Since battery usage is a very subjective measure, and measurement tools add overhead to actually measuring this, only one vague performance requirement has been imposed. FixCode wishes that the Opportunity app drain no more than 2% of the device battery in a period of 1h. To provide a feedback metric, for this criteria, two tests can be performed with our Test Flight users. iOS supports untethered logging of energy usage. With the compliant Test Flight Users enabling energy logging on their devices then supplying the data to us, we can observe energy usage statistics for:

Screen on, foreground

This test data reveals the amount of energy usage when the screen is on and Opportunity is in shown, i.e. Opportunity is running in the foreground. This test will include the amount of energy the mobile device is using to show the application user interface.

Screen off, background

This test data shows the energy that Opportunity requires to run the background trigger checking for determination of when a notification should be shown to the user. This test does not include any user interface processing energy consumption.

Network usage

A third performance requirement is that the Opportunity app does not incur more than 10MiB of data per day for 99.9% of users. Since most users will have limited data allowances, we would like limit the amount of data that Opportunity requires. We can use Test Flight users in a similar fashion to the Battery Usage tests, since iOS supports untethered network usage logging.

Server Side

To support synchronization of Opps between a single user's multiple devices, login authentication, and to decrease device side battery drain by implementing a push-trigger system, FixCode forecasts the development and deployment of - or set of - Opportunity central servers. These servers would offload some of the background trigger checking computation from the client's mobile devices. Such a system would require a high concurrency such that supporting many users does not require deploying many servers, and would require a high availability. As such, some critical performance measures can be provided even before beginning to implement such servers. At peak load FixCode expects to deploy a solution with a minimum performance in terms of number of requests served per second, and a maximum request received to response sent time for 99% of the time.

Availability

To deliver Opportunity's promise of convenient timely notifications any system which is responsible for issuing notification of an Opp which has reached a trigger state must be active as much as possible. An industry standard availability rate for online services is 24 hours a day, 7 days a week with 99% confidence. Opportunity servers should adhere to the same availability for the duration of their operation, changes non-withholding. To provide this level of availability, information stored should be replicated in more than one place, and the architecture should allow for servers to come online and go offline without disruption of service. This way, at times of high load more resources can be allocated, and should a server crash for whatever reason, others can quickly be allocated to cover the repair period.

Requests per Second

Due to the offloading of calculations from many mobile users to fewer servers, Opportunity servers need to provide a high level of parallelism. A request in this context represents an inbound request for Opp synchronization, authentication, or an outbound notification of a user's Opp reaching a triggered state. Since this is yet to be tested, an initial estimate of 100 represents a required upper bound number of requests per second handled by a single server. This is subject to change as the market reach for Opportunity is unknown. Once the test flight program begins, this number can be re-estimated, as the estimated market reach will become more accurate.

Response Time

To provide a satisfactory user experience when mobile application usage requires communication with an Opportunity server, a requirement for maximum time between request received and response sent is also imposed. Since user interactions with the mobile partner being less frequent than interactions of the server sending notification of an active trigger, a 5 second maximum response time is sufficient. This higher than industry normal will help to reduce costs associated with running servers for this purpose.

User Testing

TestFlight

Beta testing of the app will be done using the Apple TestFlight service (https://developer.apple.com/testflight/). Beta builds of the app can be deployed to users with automatic updates. When a new update is sent out, the user is notified with what aspects of the app they should be testing. The user can easily provide feedback to the developers through the TestFlight app. The app will be beta tested by a number of users until the app is ready for a full release to the iOS App Store.

Observation

User observation testing will be done to evaluate the user experience of the Opportunity mobile application. While a member of FixCode is observing, the user will be asked to complete one of the following tasks

- Create an Opp with a specific set of Conditions
- Disable an Opp
- Delete an Opp
- Acknowledge a triggered Opp
- Edit various all Opp Conditions

We will record the time to complete each task. After making small changes to the user interface and recording the times, we will be able to optimize the user experience of the app by making tasks easy and intuitive to complete.

Objectives and Success Criteria of Tests

Unit Testing

Objectives
1. Get weather data from API
2. Obtain smartphone location (latitude and longitude)
3. Find location using text
4. Get availability of user
5. Separate conditions

Specific Test	Success Criteria
1. Fetching weather data	1. Weather data for location is obtained
2. Getting the user's location	2. Location of smartphone is found
3. Searching location with text strings	3. Able to find searched location and correct address for location given
4. Fetching google calendar events	4. Availability obtained matches free space in google calendar
5. parsing conditions	5. Conditions in Opp are separated

Integration Tests

Objectives
1. Ensure that all conditions work independently
2. Ensure conditions work with other conditions
3. Ensure that a disabled Opp does not send a notification
4. Ensure that an Opp sends a notification when conditions met

Specific Test	Success Criteria
1. Creating an Opp with single condition (repeated for all conditions)	1. Opp with a single condition send a notification when conditions met
2. Creating Opp with all possible combinations of conditions	2. Opp with multiple conditions sends a notification when all conditions are met
3. Attempting to trigger a disabled Opp	3. Opp that is disabled does not send a notification even if all conditions are met
4. Attempting to trigger non-disabled Opps	4. Opp sends a notification when conditions met

UI Tests

Objectives
For all the UI tests, the objective is to ensure that all the UI buttons and actions are working and have the correct functionality. As well as, ensuring that the user is able to identify the controls and what they do.

Specific Test	Success Criteria
1. Time range double ended slider	1. Time range slider, slides and displays correct time range
2. Opps/History Tab	2. When button is activated the user is taken to the correct page (either history or Opps)
3. Selecting pins on the map view	3. User is able to select an individual pin on map
4. Add condition button on edit/create Opp	4. User is able to add another condition to a new created Opp
5. Editing Opp name	5. User is able to change Opp name after it has been created
6. Deleting Opps from table using swipe method	6. Opp is deleted when swipe is used on it
7. Disabling Opp from table using swipe method	7. Opp is disabled from table after swiping
8. Deleting conditions from edit/create Opp page	8. Conditions from Opp are deleted
9. Sky button radio group	9. Only a single status of the sky can be selected at a time.
10. Temperature slider and less than/ greater than buttons	10. Slider slides and displays correct temperature and buttons work when pressed
11. Availability logarithmic slider	11. Slider allows entering time availability from 10 minutes to days intuitively on non-linear scale
12. Opps sorting button	12. Opp are sorted when sorting button pressed

Performance Tests

Objectives
1. For performance testing the objective is to make sure that the application is running in optimal conditions and that the app runs smoothly. Also make sure that the app performs better than other apps like Opportunity.

Specific Test	Success Criteria
1. System resources	1. Ensure that in 99.9% of cases app does not use more than 5% available memory, 10% of CPU time, and no more than 10 MiB network data per day.
2. Battery life	2. Ensure app does not drain battery more than 2% per hour.
3. Requests per second	3. Server can handle up to 100 requests per second.
4. Response time	4. Ensure response time is less than 5 seconds under peak load.

User Tests

Objectives
To find any bugs we may have missed through our other tests, and to get feedback on the app in general.

Specific Test	Success Criteria
We will be using TestFlight to allow users to test the app	Users are happy with Opportunity and do not report any bugs or deficiencies in the app

Monitoring, Reporting and Testing Procedures

Test monitoring, reporting and testing procedures will be done mostly through GitHub integration. In particular, with GitHub issues. Monitoring will be done by the person conducting the specific test. The input, output, and what needs to happen will be recorded by the tester. Results will be stored in a text file, that is shared and public on a site like GitHub. Everyone with access to the file will be able to edit, view the file, and track changes. Reporting will happen as soon as possible after a test has taken place. For the tests that run automatically this information will be shared immediately on GitHub. Reporting should be done in a way that is easy to read, and in a way that information about test results is readily accessible. All reports will be made accessible to developers, clients, and shareholders. The testing procedures will be outlined for each individual test, and the tester will follow the steps. Tests should be completed on the schedule outlined in this document. Additional information and comments about testing should be included in the testing report.

Defense of Integration Plan

The integration plan we came up with was to create the app in stages which are:

1. Interface
2. Setup
3. Database
4. Background
5. Conditions
    A. Time
    B. Weather
    C. Location
    D. Google Calendar

We chose to implement Opportunity in this order because it allows each new module to be tested and fully integrated into the system. The interface was needed first to understand how the application would look and to get the general feel for how the next parts should be implemented. Next we needed to setup the app for all the functions, this included planning for how they would interact with each other as well. Next databases were added, so the app had all the data for all the functions, following this background functionality was added so that the conditions could be implemented and everything would work. Lastly, the actual conditions were to be implemented, this is done last to make sure everything else is working making troubleshooting and debugging for this last stage. The integration plan for the conditions were done in the way that was seen as the easiest and simple to implement. Thus time was implemented first, then weather, followed by location and finally integration with google calendars. The integration plan we set out was done in this order, so that the app could be

made in the easiest and simplest way, while still being able to test full functionality of the next module being added.

Minimal System by EOT

Should time restraints affect the progress of the management plan outlined above, a minimal implementation is forecasted herein for delivery by the end of the development term. FixCode assures that a minimally viable system composed of a self-supported application - that is, without dependence on any internal system servers - is produced by the development deadline. Such a system will consist of the ability to create Opps with triggers dependent on weather, user availability, and user location. As mentioned, the minimal implementation does not require the use of any servers internal to Opportunity to provide the described functionality. Furthermore, a minimally complete system will have a smaller set of accessible states within the application user interface.

Short list for triggered Opps

Below are trigger types considered as required in a minimally complete system.

Time

All Opps should have the ability to be triggered at a certain time. The user may specify the time in hours, or by general times, such as sunrise or sunset.

Weather

Any Opp should have the ability to query a weather API and react to the resulting data as one of the specified triggers. To refrain from over-extending the minimally viable system, weather based Opp triggers are restricted to the current weather at the present location of the device. The user should be able to specify triggers for weather events including active precipitation, and current temperature above or below a set value.

Availability

All Opps should have the ability to query the user's calendar. Minimal state triggers are if the triggered time doesn't conflict with an existing calendar event, or when a calendar event begins or ends.

Location

Any Opp should also be able to use device location as a trigger. In the minimal implementation only address or pin based locations will be provided, and will trigger the Opp when the device location is within a specified radius of the trigger location.

States accessible in minimal user interface

Due to the possibility that not all of the projected features make it into the delivered product, the outline below lists the portions of the app that the user will be able to navigate to in any implementation.

Opp list

Considered to be the main screen of the application, this screen shows the list of active Opps that are awaiting their trigger conditions to be satisfied.

Opp configuration

Opp creation and editing can be done using the same interface. This interface will likely consist of a set of a screens for viewing and configuring a selected Opp.

Error handling

Error states may arise during normal usage. FixCode has anticipated the following and will provide a convenient handler for each scenario.

A. User tries to create an Opp with 2 of the same conditions
B. User turns off location services
C. Notifications turned off
D. Trying to connect to Google Calendar but is not using Google Calendars

A

If a user wants to try and set an Opp that has 2 of the same conditions, the user will be unable to do this. Opportunity will grey out a condition that is already set for the Opp and thus only allow one type of condition for each Opp to be set. The tutorial will solve this by telling the user to set another Opp if they want to set an Opp with the same condition type.

B

If the user turns off location services on their phone, Opps that have location as a condition will not work. To handle this error a pop-up message will appear telling the user that location services is turned off for the application and to turn it back on if they want to receive notifications for Opps that have location as a condition.

C

If the user turns off notifications for Opportunity, the main functions will not work. To handle this error a pop-up message will appear, telling the user to turn notifications back on, so that Opportunity can function normally (see figure 9).

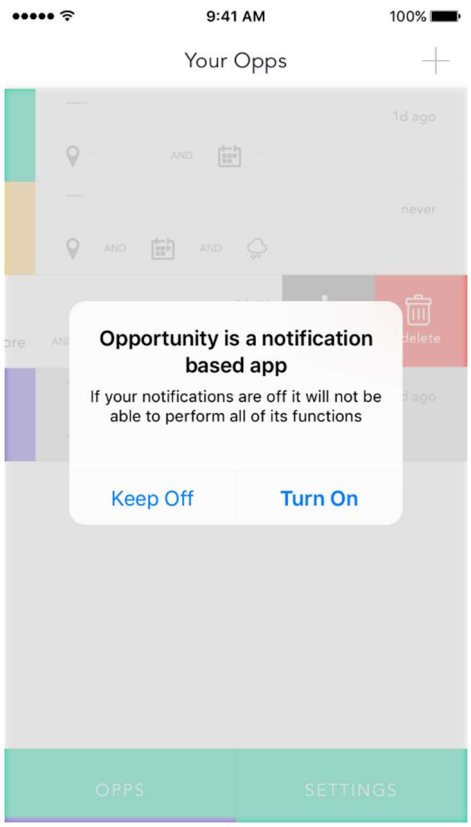


Figure 9 – Notifications off warning

D

If a user tries to connect a calendar app that is not google calendars, will cause an error for Opportunity. To handle this error Opportunity will not allow you to connect the non-Google calendar app and will have a pop-up come up that tells the use that Opportunity only works with google calendars. Opportunity does not need google calendars to be connected to work, however any conditions that use the user's schedule will not be available.