Introduction

The North Temperate Lakes Long term Ecological Research (NTL-LTER) program at University of Wisconsin-Madison studies lakes and watersheds as one of a network of 25 sites established by the National Science Foundation to support research on long term change. As part of the research program, there are several instrumented lake buoys conducting automated high frequency measurements of weather (air temperature, relative humidity, atmospheric pressure, wind speed and direction, and precipitation), water temperature and near surface dissolved oxygen.The lake condition data is captured by buoy sensors. The data for Sparking and Trout Lakes are provided from a meteorological/limnological buoy operated by the Center for Limnology, University of Wisconsin-Madison. The data for Mendota Lakes is provided from a meteorological/limnological buoy operated jointly by CFL and AOS, University of Wisconsin-Madison.

Purpose

This document describes the comprehensive architectural overview of the NTL Mobile Client. It is intended to capture and convey the significant architectural decisions which have been made on the system.

Scope

The scope of this application is to provide up to date weather and lake condition information for Trout, Sparkling and Mendota Lakes on Android devices.

Definition, Acronym and Abbreviation

Thermocline Depth: The depth of a steep temperature gradient in the lake marked by a layer above and below which the water is at different temperatures.

Secchi Depth Estimate: The depth at which a Secchi disk (a black & white circular plate) disappears from view as it is lowered into the water. It is used to measure water transparency.

References

- Limnology Department - https://lter.limnology.wisc.edu
- North Temperate Lakes - https://lter.limnology.wisc.edu/about/lakes
- Android API - http://developer.android.com/reference/packages.html
- Java API - http://docs.oracle.com/javase/7/docs/api/
- Retrofit - http://square.github.io/retrofit/
- Java API for RestFul Web service - https://jax-rs-spec.java.net
- Jersey (JAX-RS Implementation) - https://jersey.java.net
- JSON - http://www.w3schools.com/json/

Architecture Summary
The application is built using client – server architecture. A mobile client is developed using Android Native APIs that accesses lake condition information provided by RESTful Web Service.

Mobile Client Architecture
The client side of this app runs on the Android device and uses Android's Content Provider and SQLite database to persist the data on the device. The data retrieved from the web service is parsed and displayed on the mobile user interface.

Key Architectural Decisions:

- **Minimum SDK:** The application is developed using Android 4.1 API (JELLY BEAN, API Level **16**) and above to cover maximum percentage of android users. For more information on API levels and coverage information about Android Users please refer to https://developer.android.com/about/dashboards/index.html

- **Cache implementation:** As application accesses the web service information over network, a cache implementation was provided on mobile client using SQLite Database. This would enable mobile client user to view last available information that was retrieved when mobile device does not have access to network. This cache implementation is done using Content Provider Framework of Android. For elaborate information about Content Providers refer to http://developer.android.com/guide/topics/providers/content-providers.html

- **Cache Timeout:** To prevent excessive usage of network bandwidth, the data is read from the cache. The cache is refreshed by polling latest information from web service, if the cached data is older than **15 min.** This frequency is determined based on data refresh time frame in back-end database. To update cache timeout frequency, change the DEFAULT_TIMEOUT value in WeatherTimeoutCache class.

- **Retrofit Client:** The mobile client invokes the RESTful web service API using Retrofit client that parses and maps JSON formatted data from web service to mobile client objects. For information related to invocation of web service using retrofit refer to http://square.github.io/retrofit/.

***Lake Condition Web Service* Architecture**
The server-side implementation is done using Jersey implementation of JAX-RS (Java API for Restful Web Services). The web service accesses lake condition information from "**buoy_current_conditions**" table on "**dbmaker**" database. The application returns the data in Java Script Object Notification (JSON) format to web service client. For more information on JSON format, please refer to: http://www.w3schools.com/json/

Web Service Interfaces:
The web service provides following interfaces to fetch Lake condition information:

1. Retrieve lake condition.
   http://argyron.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions/

{lakeId}
The above REST URI would retrieve lake conditions of specified lake ids. The lake ids for current lakes supported by mobile client are as follows:
TR = Lake Id of Trout Lake
ME = Lake Id of Mendota Lake
SP = Lake Id of Sparkling Lake,

To retrieve lake conditions of Trout Lake specify it's lake id “TR” in URL path, as follows:
http://argyron.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions/TR

2. Retrieve conditions of all lakes:
   This is a utility interface to fetch information of all lakes at once. Currently this interface is not used by mobile client.
   http://argyron.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions

3. Alternately, the web service can also be accessed at :
   Retreive conditions of all lakes:
   http://thalassa.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions

   Retrieve specific lake condition:
   http://thalassa.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions/{lakeId}

**Important Note for Application Deployers : The database credentials are not checked-in on GIT hub. Please update userid / password in db.properties
On database schema change, change the DATABASE_VERSION in WeatherDatabaseHelper

Data Model
The web service returns following data from buoy conditions table:

sampledate	lakename	lakeid	airtemp	watertemp	windspeed	winddir	phyco_median	secchi_est	secchi_timestamp	thermocline_depth
2015-09-24 10:01:00	Lake Mendota	ME	19.10	NULL	1.9	136	699.20	1.72	2015-09-23 19:00:00	NULL
2015-09-24 10:00:00	Sparkling Lake	SP	14.70	18.58	3.1	171	NULL	NULL	NULL	8.75
2015-09-24 10:02:00	Trout Lake	TR	14.90	18.01	2.6	148	NULL	NULL	NULL	11.68

The description of data is as follows:

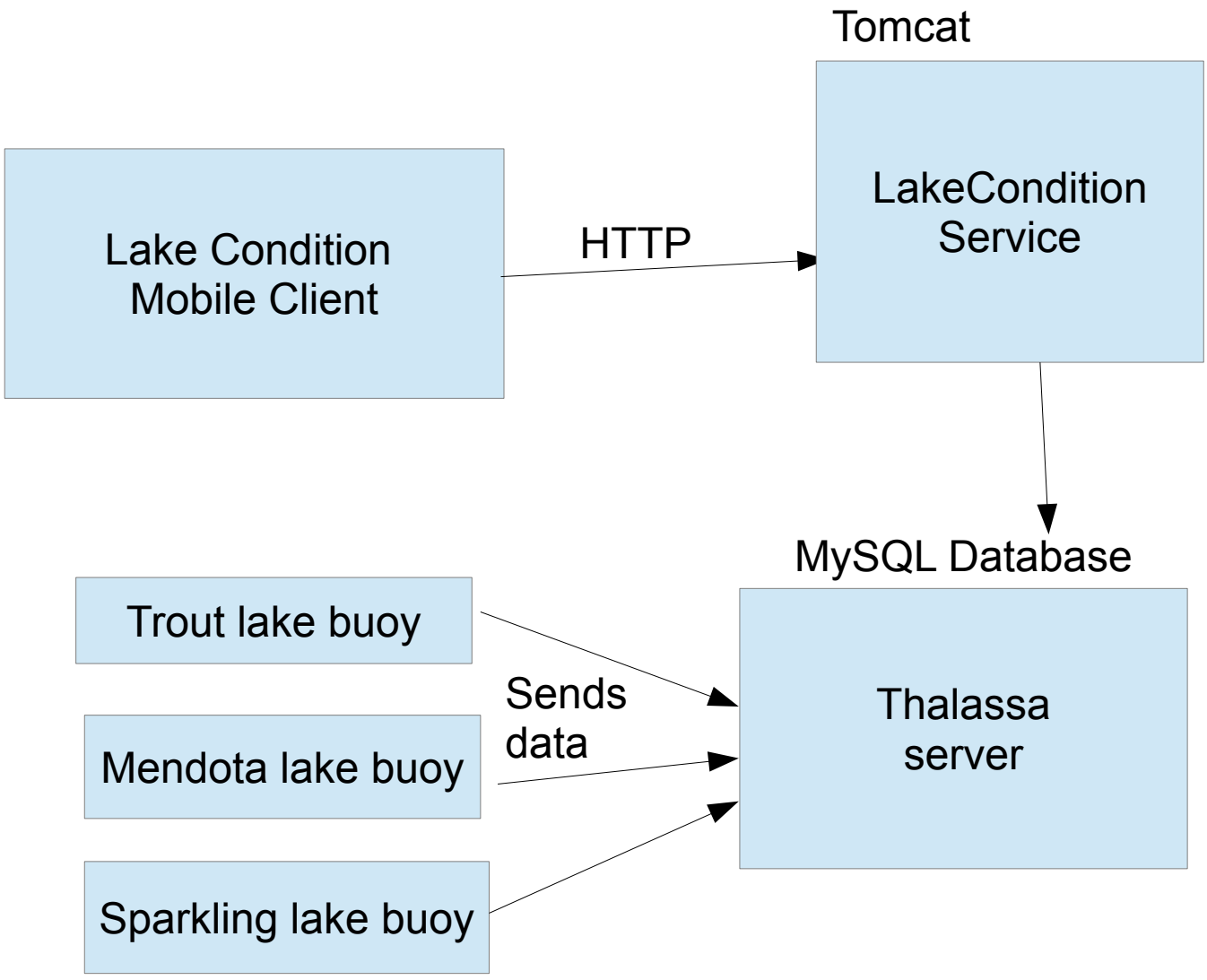
Field	Type	Null	Key	Default	Extra
sampledate	datetime	NO		NULL	
lakename	varchar(20)	NO		NULL	
lakeid	varchar(2)	NO	PRI	NULL	
airtemp	decimal(8,2)	YES		NULL	
watertemp	decimal(8,2)	YES		NULL	
windspeed	decimal(8,1)	YES		NULL	
winddir	int(10)	YES		NULL	
phyco_median	decimal(8,2)	YES		NULL	
secchi_est	decimal(8,2)	YES		NULL	
secchi_timestamp	datetime	YES		NULL	
thermocline_depth	decimal(8,2)	YES		NULL	

Tools and Technologies

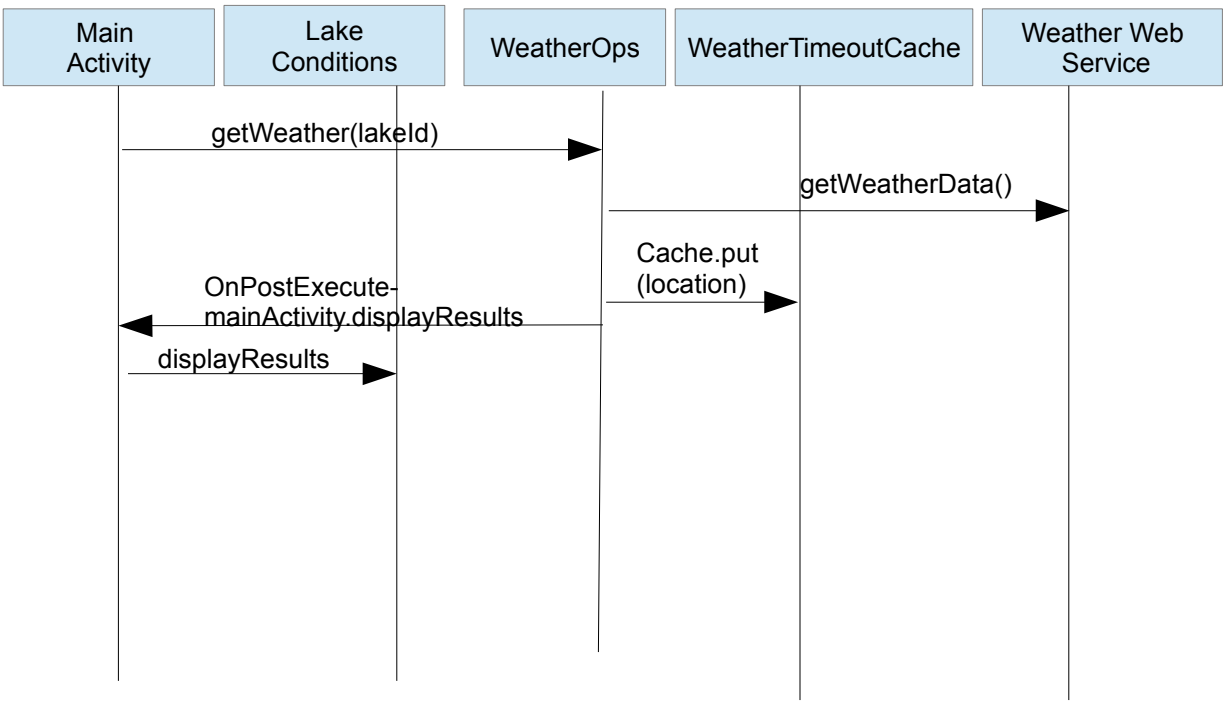
Lake Condition Service:
Tools: Eclipse, Tomcat, My SQL
Technologies: Java 7, JAX-RS (Jersey Implementation)

Lake Condition Mobile Client:
Tools: Android Studio
Technologies: Java 7, Android 4.1 SDK (API Level 16), Retrofit

Deployment Diagram



High Level Sequence Diagram



Environment Properties

Database :
Data Source: dbmaker (MySQL database)
Table Name
MySQL:buoy_current_conditions

REST Service Web Server: Tomcat 7
Primary:
Trout Lake:
http://argyron.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions/TR

Secondary:
http://thalassa.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions/TR

Android Client
Manifest.xml:
This app needs the following permissions that has been added to the manifest.

- <uses-permission android:name="android.permission.INTERNET" />
- <uses-permission

android:name="android.permission.ACCESS_NETWORK_STATE" />

COOK BOOK

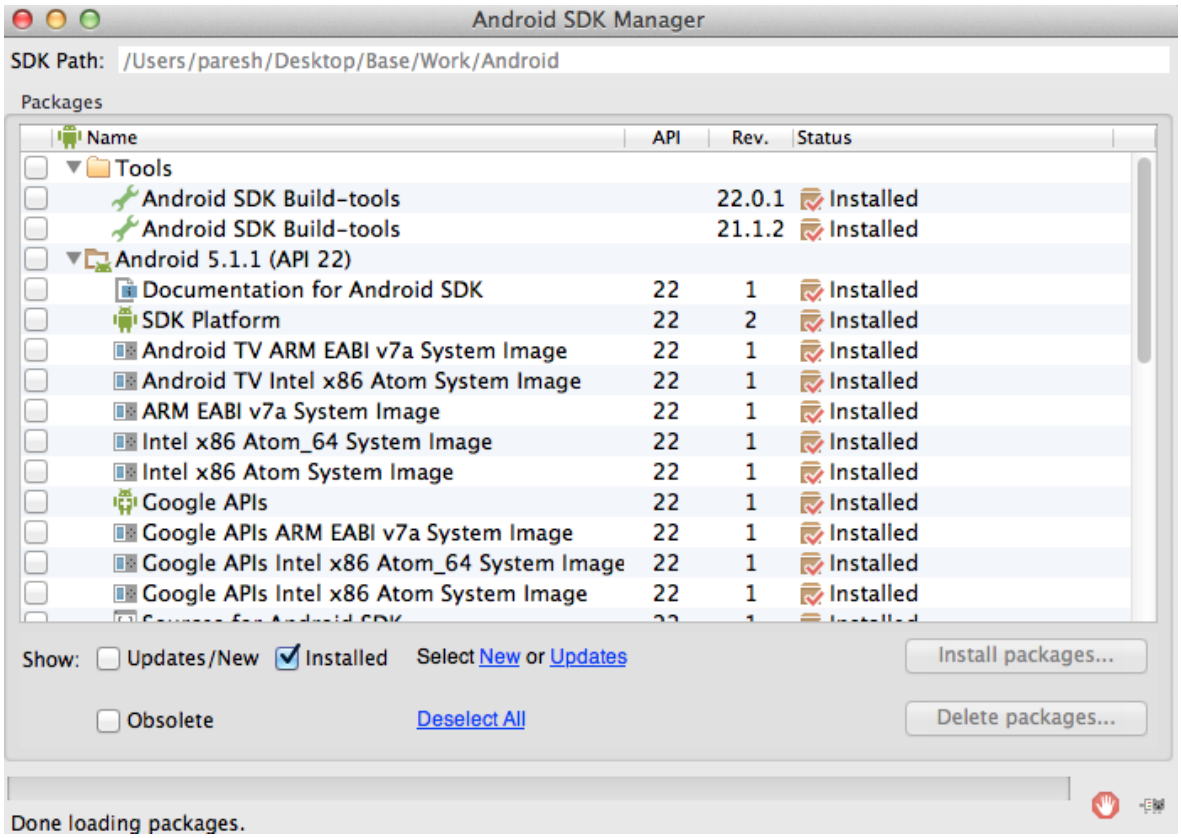
Prerequisites – The following set up needs to be done before importing the appplication.

- Android Studio – Download it from Android Developer website
- LakeConditionService web service – This can be accessed by calling
  http://argyron.limnology.wisc.edu:8080/LakeConditionService/
  OR
  http://thalassa.limnology.wisc.edu:8080/LakeConditionService/

- Icons and images – This isn't required but may need for different look.
- JDK 1.7

Import the project

Import the project LterLakeMobileClient from https://github.com/CFL-UWMadison/LTER-Buoys. The minimum required sdk has been set to API 16 and target sdk to API 22. This sdk version can be installed from within the Android SDK and AVD manager.



Client-side of application

The app handles runtime configuration changes robustly ie. It should handle

screen orientation changes while downloading data. Information on how to handle these type of changes is available here.

App uses Retrofit to communicate with the Web service and does not block the UI thread. Retrofit obtains lake condition data from the web service, and this data is then stored in a timeout cache that uses a database-backed Content Provider for storage on device.

The application has three activities. Below is the description of the activities and the layouts associated with them:

MainActivity – Layout associated with it is thumbnail.xml.

Main activity is the main entry point of application. Main activity is the generic activity which means the view is going to work together with the WeatherOps which is the presenter layer in order to be able to manage WeatherOps object and make it easy to handle configuration changes.

Also it displays a toast for the error if server is not reachable or phone is not on network.

LakeConditions - Layout associated with it is activity_lake_conditions.xml.

Extracts all the data from the weather data object and initializes the views and displays all the views to the UI interface. If results came back as nothing then it displays data not found.

About - Layout associated with it is about.xml.

About class provides information about the application and disclaimer to the user.

Other classes:

WeatherOps – It is the business logic of the Presenter layer. This class implements the client-side operations that obtain lake conditions data from the LakeConditionService web service. It implements ConfigurableOperations so an instance of this class can be managed by the GenericActivity framework. It also implements GenericAsyncTaskOps so its doInBackground() method will run in a background thread.

It initializes a Retrofit proxy named mWeatherWebServiceProxy that sends requests to the LakeConditionsService web service and converts the JSON response to an instance of WeatherData POJO class.
It keeps track of whether a call is already in progress via mCallInProgress and ignores subsequent calls until the first call is done.

WeatherProvider:

WeatherProvider extends ContentProvider that is used to store information about weather data returned from the Lake condition web service. Details on Content Provider can be accessed here.

Weather Contract:

Weather contract defines the metadata for the Content Provider

WeatherDatabaseHelper:

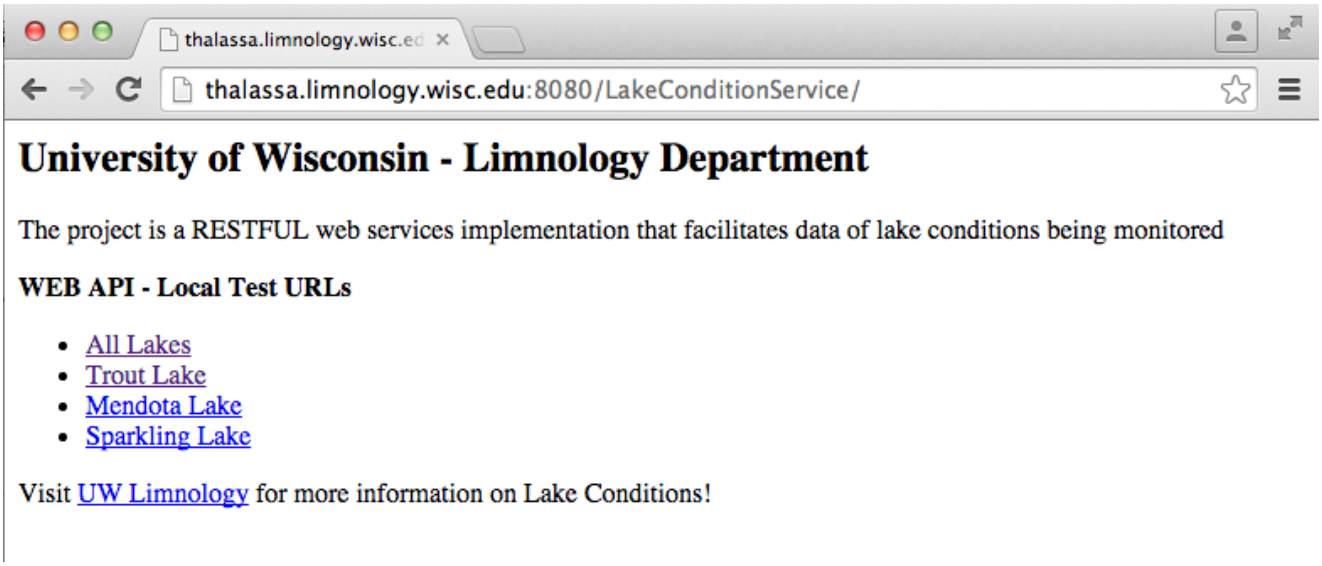
The database helper used by the Content Provider to create and manage its underling database. If the database schema is changed, change the DATABASE_VERSION in the class

WeatherTimeoutCache:
It keeps track of data persistently.

Fetch data from LakeConditionService

Application gets the current lake data from the web service in the JSON response.
For eg:
http://argyron.limnology.wisc.edu:8080/LakeConditionService/webapi/lakeConditions/TR

The response from the web service looks like this:



For specific lakes: