CS551 Fourth Increment Report

By
Rishabh Bhojak (SG2)
Bhargava Gellaboina (SG2)

Objectives

The objective of this increment was to give up the finishing touches with new features like mashup of accuweather & google maps and code improvement.

As proposed, we developed the following three applications,

DS-Crowd: Disaster-Scene Crowdsourcing Analyst App
CISA- General Disaster-Scene Crowdsourcing Application
Green Garden- Green Garden Crowdsourcing Application

Development was done on:
Android Development API 19
Google Play Services 5077000_r18
Accweather API

Import Existing Services/API

All Existing Services

Google Maps Android API v2: Coordinator opens the coordinator view in android

application which calls the MapView.getMap() function to load the Google maps for the Coordinator. At the same time, android application also pulls all the images using getAllImageData() function using Asynchronous Http Service it displays the images on google maps and shows it to the Coordinator.

To use the application, we followed the following steps:

1. We created SHA1 for our own signature key

2. We registered with Google APIs

3. We then generated key for our own application

4. Then created the project and use the key for development.

Google Maps Geolocation API: When the observer takes pictures using the application, we are saving Geolocation of the user(where the picture was taken)using geoLocService.getLatitude() and geoLocService.getLongitude() functions in to MongoDB and using the Latitude and Longitude values we are querying for the lo- cation using Geolocation API.

```
public void addGeoLocation() {
geoLocService = new GeoLocationService(ObserverActivi- ty.this);
// check if GPS enabled
if (geoLocService.canGetLocation()) {
double latitude = geoLocService.getLatitude();
double longitude = geoLocService.getLongitude(); latTv.setText("Lat: " + latitude);
lonTv.setText("Lon: " + longitude);
double latlon[] = { latitude, longitude }; scene.setLocation(latlon);
```

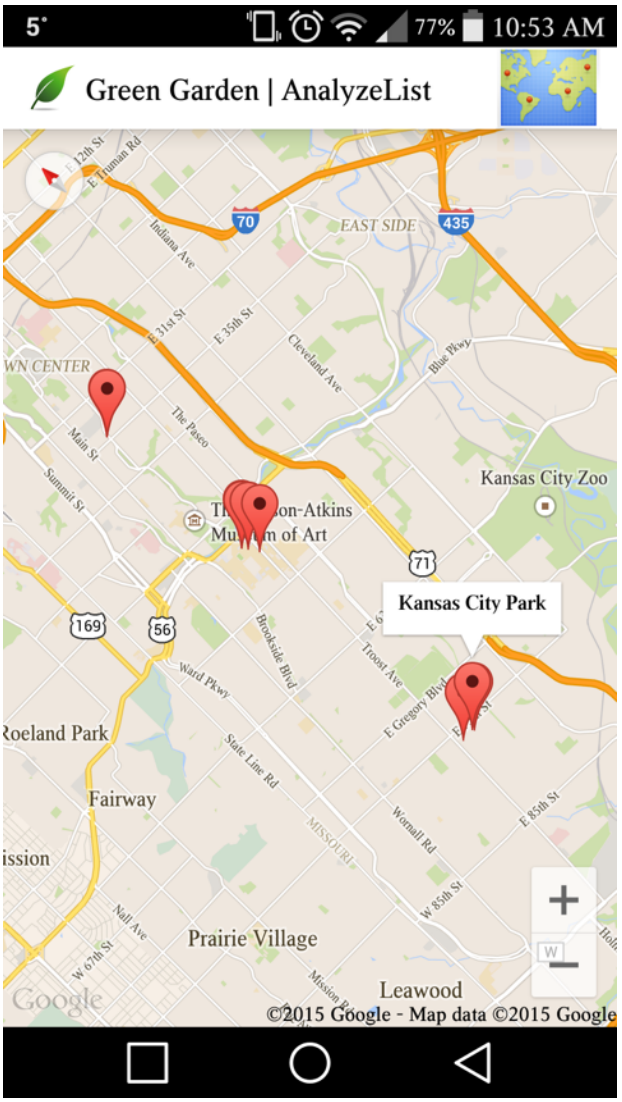
```
} else {
geoLocService.showSettingsAlert();
}
}
```

Example code to load the location of images with their description in google maps:

```
// Code to get the image data from the MongoDB

protected String doInBackground(String... params) {
ByteArrayOutputStream out = null;
HttpClient client = new DefaultHttpClient();
String url = Utils.HOST + Utils.RES_LIST_ITEMS + "?start="+ sceneList.size() +
"&limit=20";
System.out.println(" Url: " + url);
HttpGet get = new HttpGet(url);
HttpResponse response;
try {
response = client.execute(get);
out = new ByteArrayOutputStream();
response.getEntity().writeTo(out);
out.close();
} catch (IOException e) {
e.printStackTrace();
} finally {
get.abort();
}
return out.toString();
}
// Post processing after retrieving the data

@Override
protected void onPostExecute(String result) {
dialogShow = false;
System.out.println("Data from server: " + result);
if(populateSceneData(result)){
//Choose the required data to display on map using below function call
```

```
populateMarkersToMap();
MapsInitializer.initialize(obj.getActivity());

// Updates the location and zoom of the MapView
System.out.println("Before CAMMAP");
CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom(new
LatLng(sceneList.get(0).getLocation()[0], sceneList.get(0).getLocation()[1]), 15);
mMap.animateCamera(cameraUpdate);

//return obj.rootView;
}

// Populate the required data from the retrieved data

private boolean populateSceneData(String data) {
if (data == null) {
//listView.removeFooterView(loadMoreView);
return false;
}
JSONArray jsonList;
try {
jsonList = new JSONArray(data);
if (jsonList.length() == 0) {
loadmore = false;
//listView.removeFooterView(loadMoreView);
} else {
if (jsonList.length() < 10) {
//listView.removeFooterView(loadMoreView);
loadmore = false;
}
for (int i = 0; i < jsonList.length(); i++) {
JSONObject sceneJSON = jsonList.getJSONObject(i);
// Image Mapping
```

```
Image image = new Gson().fromJson(sceneJSON.getString("imageData"),
Image.class);
// Location Mapping
String sceneId = sceneJSON.getString("_id");
String locationString = sceneJSON.getString("lo-
cation");
double location[] =
extractLocation(locationString);
// Description Mapping
String description = sceneJSON.getString("de-
scription");
Scene scene = new Scene(sceneId, image, de-
scription, location);
sceneList.add(scene);
}
}
} catch (JSONException e) {
e.printStackTrace();
}
return true;
}
```



Accuweather API: Mashup applications to show the weather in google maps when the picture was taken. This is then, is used to display weather information with the picture.

Camera Object Android: We are using Android Camera Object to provide the observer functionality to take pictures. Pictures are taken using the camera object and stored into mongoDB.

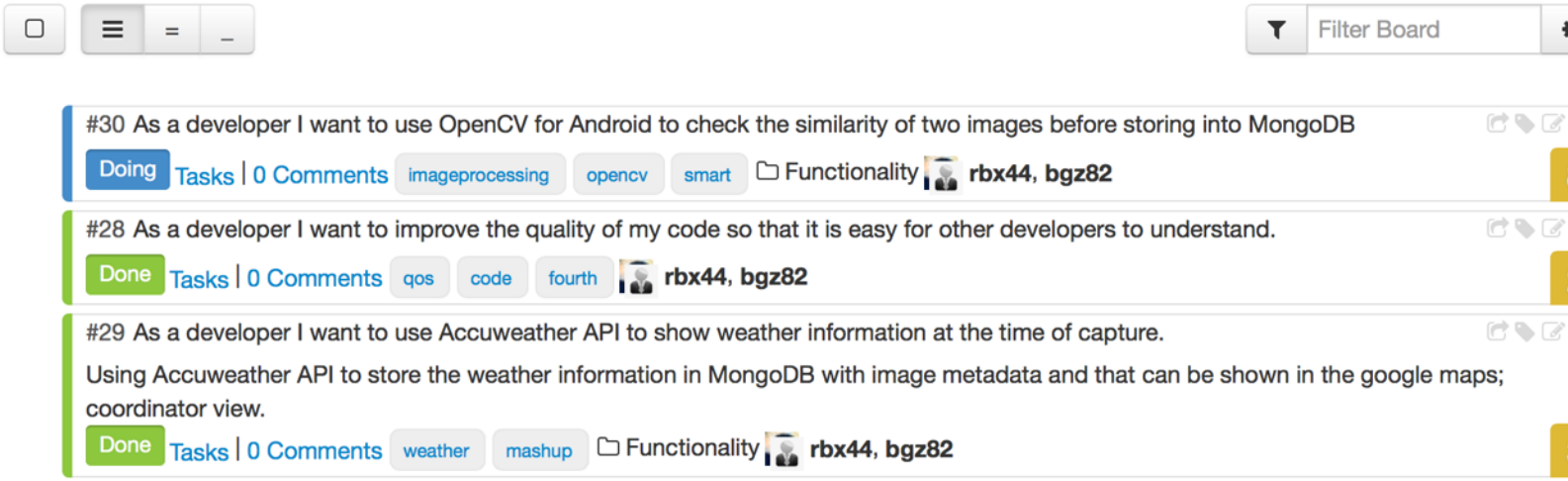
To use camera object we declared the permission in manifest, open(init) to open the instance of the camera, set index value and give this to onClick functions.

Below is the example code for using camera object to take camera pictures in our application:

```
private Camera getCameraInstance() {
Camera c = null;
if (ObserverActivity.this.getPackageManager().hasSystemFeature(
PackageManager.FEATURE_CAMERA_ANY)) {
Log.i("Camera", "There is no Camera on this device");
} else {
Log.i("Camera", "There is no Camera on this ");
}
try {
Camera.CameraInfo cameraInfo = new Camera.CameraInfo();
int cameraCont = Camera.getNumberOfCameras();
for (int camIndex = 0; camIndex < cameraCont; camIndex++) {
Camera.getCameraInfo(camIndex, cameraInfo);
if (cameraInfo.facing == Camera.CameraInfo.CAMERA_FACING_BACK) {
try {
c = Camera.open(camIndex);
} catch (RuntimeException e) {
Log.e("Camera",
"failed to open camera: "
+ e.getLocalizedMessage());
```

```
} catch (Exception e) {
Log.i("camera", "Unable to open camera");
}
return c;}
public void onClickCapture(View v) {
capture_status.setText("scene captured");
capture_status.setVisibility(View.VISIBLE);
capture_status.setAlpha(0.6f);
image_capture.setText("Captured");
mCamera.takePicture(null, null, new MyPictureCallback(
ObserverActivity.this));
image_capture.setEnabled(false);}
```

Stories



Detail Design of Services

Write User Stories using ScrumDo

(Screenshot)

Service Description

Existing Services

Data Layer
MongoDB, for backend storage.
We created the database ‘umkc’, as part of this project we created collection ‘green’.
To make the connection with the application and the server. It makes the Http request to get the ID of the images.
Once the connection has been made and the ID is retrieved it makes another Http request with the MongoDB to get the actual image.
To get the text data associated with image it makes the Http get request to get the attributes(type, degree of damage associated with the image).
Using the Rest service we get the geo location of the images and we are visualizing it in google maps.

Schema – JSON Format fields explained below with an example:

Field	Type	Description
_id and $oid	JSON Object	The id that uniquely refers the object
sceneId	String	default filename is used here
imageData	Json Object	Image information like type and data is stored in JSON format
type(child of imageData)	String	image type - “JPEG, PNG”
data(child of imageData)	String	Image encoded as base64 string
description	String	Scene description
location	Array of double values	Latitude and longitude values stored in JSON array
results	JSON Array	Stores the region information and from the device from which it is marked
deviceId(child of imageData)	String	Network id used to identify the device uniquely
region(child of imageData)	JSON Array	Selected region information is stored
category	Integer	Used to find type of Object
damageLevel	Integer	Damage level of the selected region
boundary	JSON Object	Selected region in the scene
analyzed_status	String	Status of Image as analyzed or not.
weather_acc	JSON Object	Weather information

Example

{
"_id": {

"$oid": "53d26a76e4b0ad6bce969780" },

"sceneId": "http://esridev.caps.ua.edu/MooreTornado/Images/Day3/Christine/IMG_4311.JPG", "imageData": {

"type": "JPG",

"data": "/9j/4AAQSkZJRgABAgAAAQABAAD/2wBDACgcHiMeGSgjISMtKygwP-GRBPD- c3PHtYXUlkkYCZlo+AjIqgtObDoKrarYqMyP/L2u71////m8H////6/+b9//j/....."

},
"description": "One completely damaged building", "location": [35.31747, -97.563394],
"results": [{

"deviceId": "e8:99:c4:8f:ef:5d",

"region": "[{\"categroy\":3,\"damageLevel\":1,\"boundry\":{\"bottom\": 477.96045,\"right\": 399.5773,\"left\":5.310051,\"top\":145.84058}},{\"categroy\": 1,\"damageLevel\":0,\"boundry\": {\"bottom\":481.42346,\"right\": 390.42566,\"left\":-23.818665,\"top\":154.68658}}]"

}]}

Services Implemented as part of Fourth Increment

Service #1

Using Accuweather API to store the weather information at the time of image capture and storing it into MongoDB, then using this later to show the information on google maps if needed: Mashup Application.

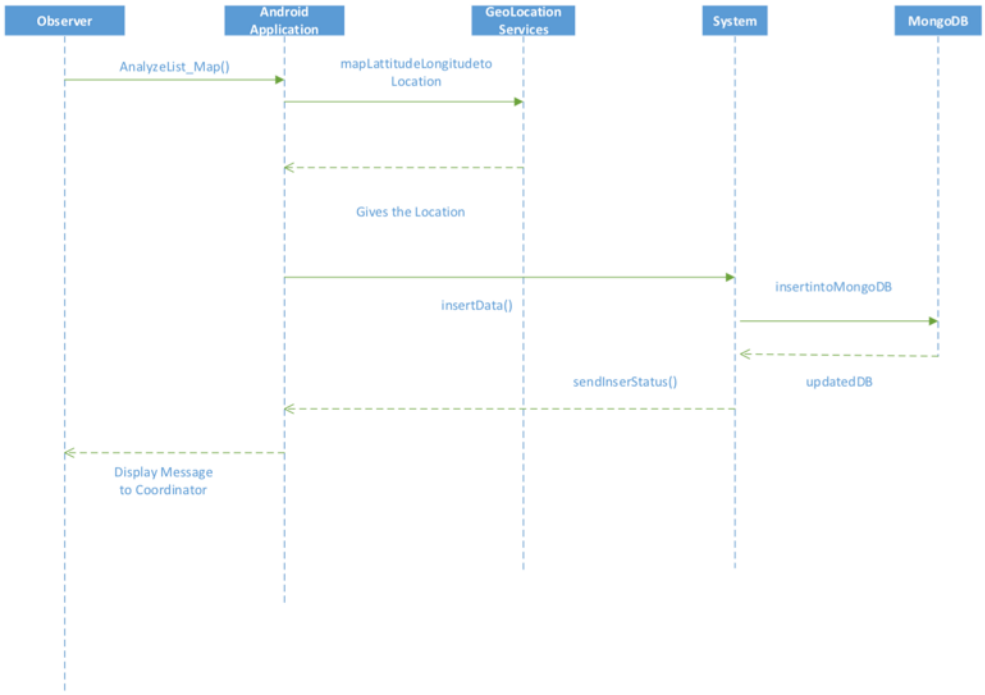
Service #2

Improving the quality of the code, over the entire project code, which includes Web UI, Android application, APIs, Rest Implementation, Test Cases, etc

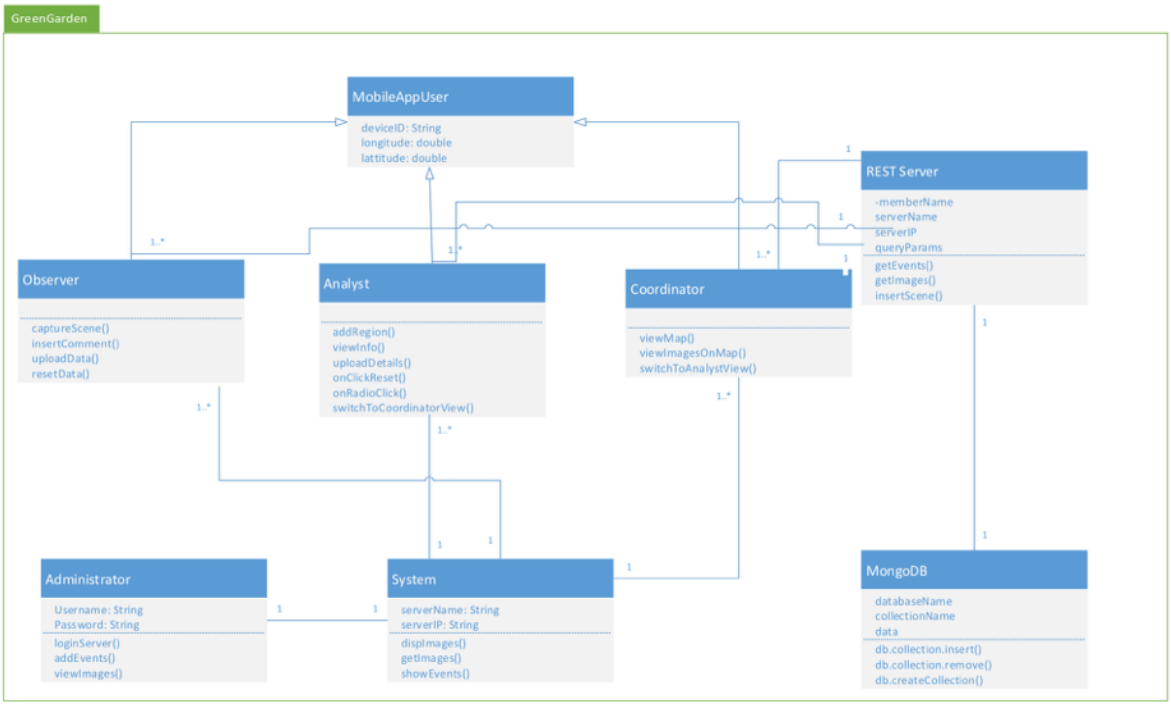
Service #3

Using OpenCV for Android to check the similarity of the images before storing them into MongoDB, if the score is higher a certain number(threshold) only one image will be stored, however, this approach requires lot of work, and we worked so many hours to make it work but still couldn't which also kind of thwarted our deep learning process in Android Application.

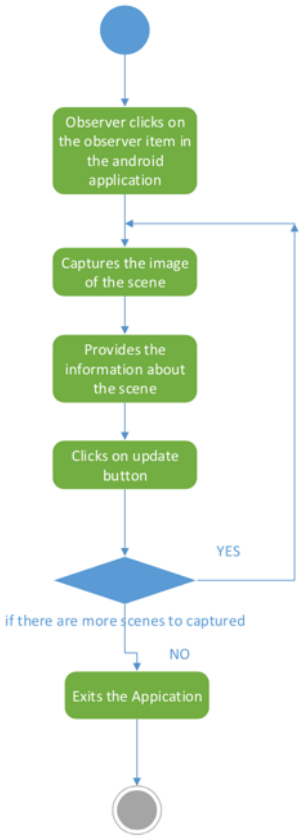
Sequence Diagram

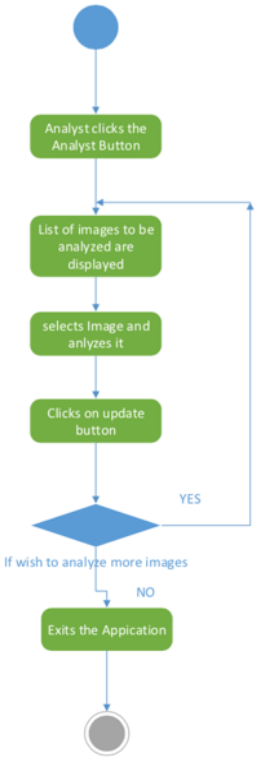


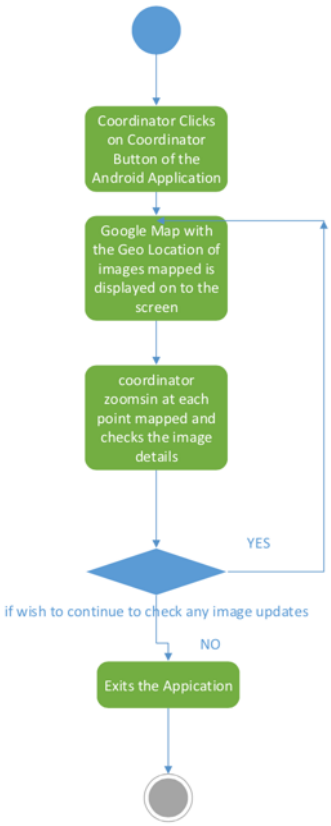
Class Diagram



Activity Diagram




Analyst clicks the Analyst Button
List of images to be analyzed are displayed
selects Image and anlyzes it
Clicks on update button
YES
If wish to analyze more images
NO
Exits the Appication


Coordinator Clicks on Coordinator Button of the Android Application
Google Map with the Geo Location of images mapped is displayed on to the screen
coordinator zoomsin at each point mapped and checks the image details
YES
if wish to continue to check any image updates
NO
Exits the Appication

Design of Mobile Client Interface

Analyst: Responsible for analyzing a provided scene by identifying the type of objects, degree of damage, and marking boundaries.

Coordinator: Responsible for crowdsourcing management and decision- making based on the available scenes and crowdsourcing results.

Observer: Responsible for capturing and collecting the images of disaster scenes.

• Main Activity: This is the screen which will be displayed when the application is loaded. The options can be (a) Analyst, (b) Coordinator and (c) Observer. On Selecting Analyst will takes to Analyst List page.

• Analyze Item List: The user will be provided with the list of scenes. Users can switch to Co- coordinator page, load more

scenes from the cloud and select a scene to analyze.

• Analyze A Scene: A scene can be analyzed with the detection techniques. We are still thinking what all methods will be and set the damage degree for each categorized object.

• Co-coordinator View: All the scenes are visualized on a Map with markers representing each scene at a particular location. Selecting a marker provides more de- tails of the crowdsourced information for the selected scene. Once more details are listed, if the user wish to analyze the scene identifying different category objects, it can be done by selecting the Analyze button.

• Observer View: Provides the interface to capture the images. User can take picture. These pictures will be uploaded to the cloud automatically. We are thinking to use MongoDB at the backend.

Design of Unit Test Cases

NUnit

1. To retrieve the images from mongoDB
2. To retrieve the image IDs from mongoDB
3. Submit Observer Data to mongoDB

JUnit

For radio button for all events.

Implementation

Implementation of REST services

Rest service is implemented to connect to MongoDB.
Http Request/Response methods are used to pull and push the data.
Below is the example for connecting to MongoDB, and pulling the events data for the Spinner.

```
@Path("/cisa")
public class MyResource {

        @GET
        @Produces("text/plain")
        public String getIt() {
                return "Hi there!";
        }
    @GET
    @Path("events")
    @Produces("text/plain")
    public String getEventsInfo() throws JSONException
     {
            MongoClient mongo;
            JSONObject spinnerArray;
            //JSONArray sArray;
            String result[]= new String[100];
            String concat="";
            int i=0,j=0;
            try {
                    mongo = new MongoClient("lasir.umkc.edu", 27017);
                    DB db = mongo.getDB("umkc");
                    DBCollection table = db.getCollection("events");
                    DBCursor cursor = table.find();
                    while (cursor.hasNext()) {
                            spinnerArray=new JSONObject((cursor.next().toString()));
                            result[i]=spinnerArray.get("name").toString();
                            i++;
                    }
             } catch (UnknownHostException e) {
                    e.printStackTrace();
             }
            for(j=0;j<i;j++)
            {
                concat=concat + result[j] + ";";
            }
            return concat;
     }
```

To make the application efficient, we thought of loading only 6 images at a time. To implement the same, we used REST. Here we are making the connection with the Server using REST to pull 6 images at one time. Below is the implementation.

```
  int start=0;
  String[] ids = new String[6];
  try
  {
   URL dest = new URL("http://lasir.umkc.edu:8080/greengarden/webresources/ggarden/sceneids?start="+start+"&limit=6");
   URLConnection yc = dest.openConnection();
   BufferedReader in = new BufferedReader(new InputStreamReader(yc.getInputStream()));
   String inputLine="";
   String getData="";
   int i=0;
   while ((inputLine = in.readLine()) != null)
     {
       getData+=inputLine;
       i++;
     }
   in.close();
   i=0;
   StringTokenizer st=new StringTokenizer(getData,", ");
   while(st.hasMoreTokens())
    {
      ids[i]="http://lasir.umkc.edu:8080/greengarden/webresources/ggarden/image/"+st.nextToken();
      i++;
    }
   }
  catch(Exception e)
   {
     out.println(e);
    }
start=start+6;
%>
```

We are using AJAX to implement the next & previous button to load the next six images or the previous six images. Connection is made through REST service and we also implemented the design using Jssor_slider.

```
    <script>
        jssor_slider1_starter('slider1_container');
    </script>
</div>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
    <script type="text/javascript">
    function getNextImages()
     {
            var ids;
            var check;
            var starts = document.getElementById("start").value;
            console.log(starts);
            $.ajax({
                    url: "http://lasir.umkc.edu:8080/greengarden/webresources/ggarden/sceneids?start="+ starts + "&limit=6",
                    type: 'get',
                    async: false
            })
            .done( function (data, status) {
                    ids = data.split(', ');
                    var i=0;
                    if(data == "")
                     {
                       alert("End of Images");
                       check="0";
                       return;
                     }
                    while(i<ids.length)
                     {
                       ids[i] = "http://lasir.umkc.edu:8080/greengarden/webresources/ggarden/image/" + ids[i];
                       i++;
                     }
                    console.log(ids.length);
             })
             .fail( function (data, status) {
             });
           if(check == "0")
            {
              return;
            }
            document.getElementById("start").value = parseInt(starts) + 6;
            document.getElementById("image1").src = ids[0];
            document.getElementById("image2").src = ids[1];
```

Rest Code for Handling Multiple Database according to event selecting in Spinner

```
private boolean updateAnalysisToMongo(String sceneId, String deviceId,
                JSONArray jArrayRegion, String event, String analysis) {
        try {
        DBCollection collection=null;
                Mongo mongo = new Mongo("localhost", 27017);
                DB db = mongo.getDB("umkc");
                if(event.equals("Earthquake"))
                {
                  collection = db.getCollection("cisa");
                }
                else if(event.equals("Tornado"))
                {
```

```
collection = db.getCollection("tornado");
}
else if(event.equals("generic"))
{
collection = db.getCollection("generic");
}
else if(event.equals("garden"))
{
collection = db.getCollection("ggarden");
}
else
{

}
BasicDBObject allQuery = new BasicDBObject("_id", new Objec-
tId(
                sceneId));

DBCursor cursor = collection.find(allQuery);
DBObject ob = cursor.next();

/*
 * while (cursor.hasNext()) { System.out.println(cursor.next()); }
 */
/*
 * BasicDBObject query = new BasicDBObject(); query.put("_id",
new
 * ObjectId(sceneId));
 *
 * BasicDBObject push = new BasicDBObject(); BasicDBObject
anaData =
 * new BasicDBObject(); anaData.append("deviceId", deviceId);
 * anaData.append("region", jArrayRegion.toString());
 * push.put("$push", new BasicDBObject("analysis",
 * anaData.toString()));
```

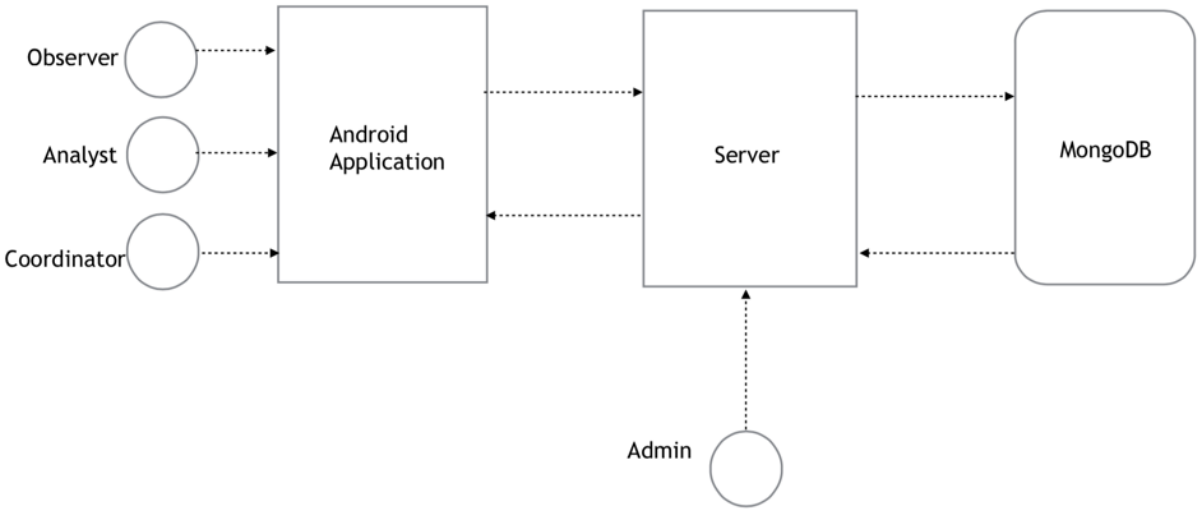
```
                *
                * collection.update(query, push);
                */

                DBObject findQuery = new BasicDBObject("_id", new
ObjectId(sceneId));

                DBObject listItem = new BasicDBObject("results", new BasicD-
BObject(
                                "deviceId", deviceId).append("region",
                                jArrayRegion.toString()));
                DBObject updateQuery = new BasicDBObject("$push", listItem);
                DBObject done = new BasicDBObject("Analyzed", "Yes");
                DBObject updateDone = new BasicDBObject("$apush", done);
                collection.update(findQuery, updateQuery);
                collection.update(findQuery, updateDone);

                // getAnalysisFromMongo(sceneId);
                System.out.println("Done");

                return true;
            } catch (UnknownHostException e) {
                e.printStackTrace();
                return false;
            } catch (MongoException e) {
                e.printStackTrace();
                return false;
            }
        }
```

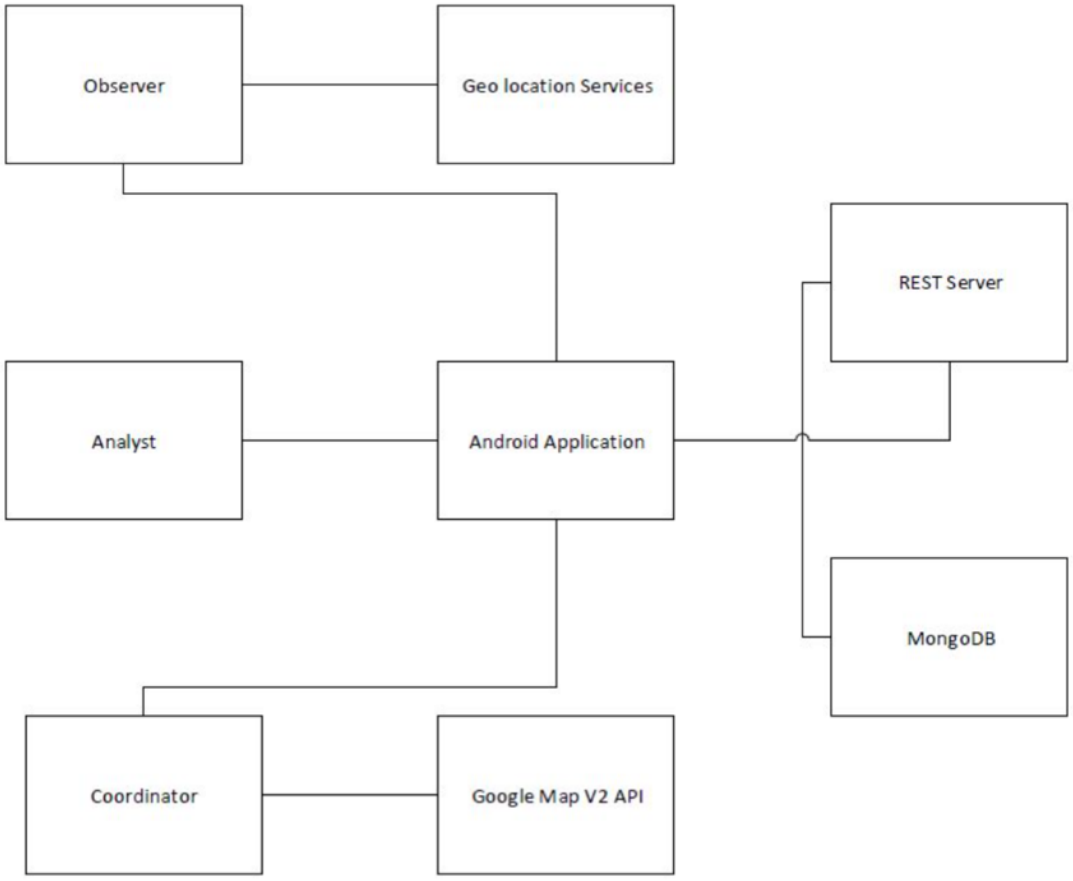
Implementation of Rest services for Excel Database Loading

```
private boolean insertToMongo1(Scene scene, String yes) {
        try {
            Mongo mongo = new Mongo("localhost", 27017);
```

```
                DB db = mongo.getDB("umkc");
                DBCollection collection = db.getCollection("tornado");

                // convert JSON to DBObject directly
                Gson gson = new Gson();
                JsonElement jsonElement = gson.toJsonTree(scene);
                jsonElement.getAsJsonObject().addProperty("Analyzed", "Yes");
                DBObject dbObject = (DBObject)
JSON.parse(gson.toJson(jsonElement));
                collection.insert(dbObject);
                DBCursor cursorDoc = collection.find();
                DBCursor c = collection.find();

                while (cursorDoc.hasNext()) {
                        System.out.println(cursorDoc.next());
                }
                System.out.println("Done");
                return true;
        } catch (UnknownHostException e) {
                e.printStackTrace();
                return false;
        } catch (MongoException e) {
                e.printStackTrace();
                return false;
        }
  }
```

Implementation of User Interface





Implementation of Existing Services

1. The goal of this project is to build an android application which has the following the system features.
Analyst: Responsible for analyzing a provided scene by identifying the type of objects, degree of damage, and marking boundaries. Coordinator: Responsible for crowdsourcing management and decision- making based on the available scenes and crowdsourcing results. Observer: Responsible for capturing and collecting the images of disaster scenes.

2. Add roles and functionalities for the different types of users in this application, including Observer, Analyst and Coordinator.

3. Install Apache Tomcat, update Java, setup Mongo, upload data to Mongo, connect to Application

4. To implement REST service to connect to the server and MongoDB.

5. Use the server memory efficiently by loading only six images at a time. User can move to the next six images using the slider option.

6. Using Web Page Interface , provide an option to administrator to add the events.

7. Login page for administrator and validate the credentials and move to appropriate page.

8. Resizing the images taken by the observer to improve the speed of loading and retrieving in AnalystList

9. Load the thumbnails of the images instead of loading the whole image from MongoDB in AnalystList

10. Cache the first loaded data in AnalystList to make loading faster.

11. Observer image capturing and also Status is sent back to application once the image is loaded in MongoDB.

12. Analyst can analyze multiple regions in an image and there is layout provided where the analyst can see the already analyzed events in an image.

13. Image holder view to select the regions in an image to analyze have a new and improved look.

14. Performance of the application tested using 15 phones at same time.

15. Unit and Integration testing of application.

16. 2013 Moore Tornado Scene Damage Database : Total of 550 images with links and metadata in xls(excel) format. Get all this data and store it in mongoDB.

17. 2014 South Napa Earthquake Field Reconnaissance Database: Total of 3000 categorized images on a website

with location, severity, and other information as metadata, using web crawler command to fetch images with its data and storing the same in mongoDB .

18. Spinner DS-Crowd: Adding spinner to DS-Crowd application, which allow users to make a select of database they want to work on, making the selection only that database images gets loaded.

Services Implemented as part of Fourth Increment

As explained in Service Description, there are 3 services we thought of implementing as a part of our fourth increment, and we call it up for this project.

Service #1

Using Accuweather API to store the weather information at the time of image capture and storing it into MongoDB, then using this later to show the information on google maps if needed: Mashup Application.

Method:

We used the Accuweather API to get weather information using the REST and stored into MongoDB, this was later used with google maps to show the information if needed.

Service #2

Improving the quality of the code, over the entire project code, which includes Web UI, Android application, APIs, Rest Implementation, Test Cases, etc

Method:

Everything for this, was done manually, on developers understanding and which further revealed our problems with data loading in Server and also the processing speed.

Service #3

Using OpenCV for Android to check the similarity of the images before storing them into MongoDB, if the score is higher a certain number(threshold) only one image will be stored, however, this approach requires lot of work, and we worked so many hours to make it work but still couldn't which also kind of thwarted our deep learning process in Android Application.

Method:

Using OpenCV for Android with tools like, Surf extractor, Histogram for RGB & HSV, Contours, Canny Detection, JavaFX, etc. We used OpenCV 2.4.11 for our development.

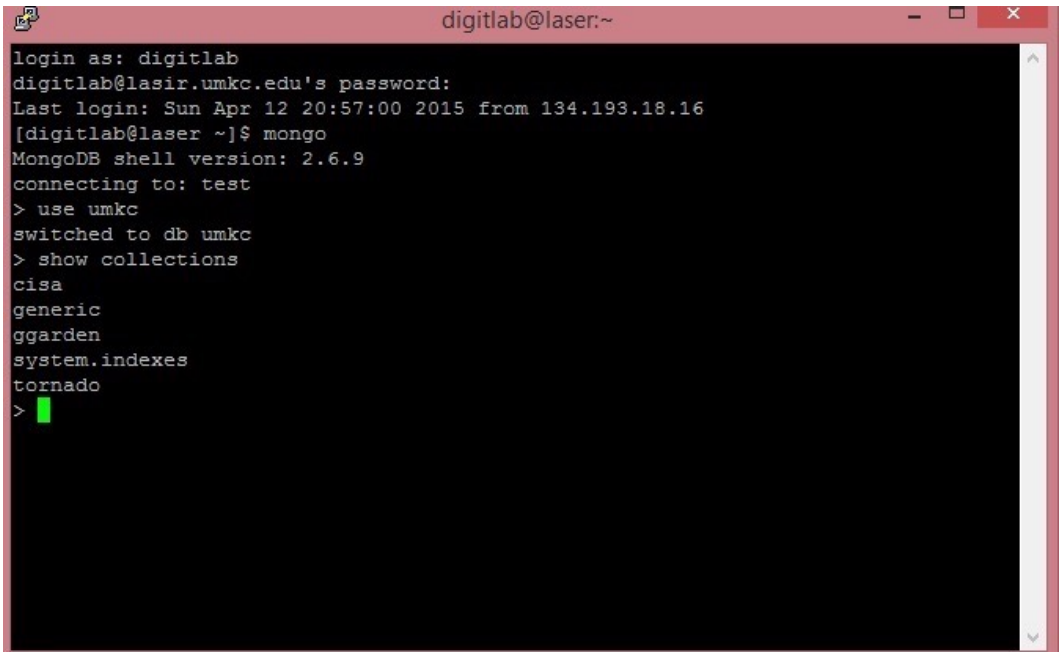
Code

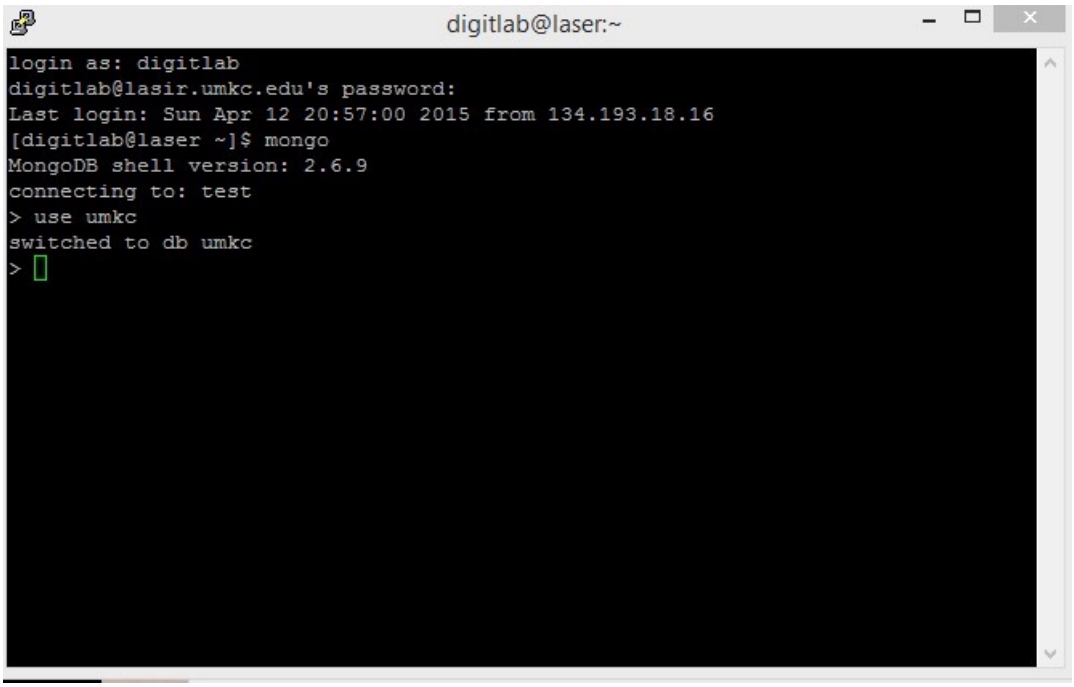
```
FeatureDetector surfDetector = FeatureDetector.create(FeatureDetector.SURF);

DescriptorExtractor surfExtractor = DescriptorExtractor.create(DescriptorExtractor.SURF);

                keyPoint01 = new MatOfKeyPoint();

                surfDetector.detect(grayImage01, keyPoint01);

                keyPoint02 = new MatOfKeyPoint();

                surfDetector.detect(grayImage02, keyPoint02);

                descripters01 = new Mat(image01.rows(), image01.cols(), image01.type());

                surfExtractor.compute(grayImage01, keyPoint01, descripters01);

                descripters02 = new Mat(image02.rows(), image02.cols(), image02.type());

                surfExtractor.compute(grayImage02, keyPoint02, descripters02);

                matchs = new MatOfDMatch();

                matcher = DescriptorMatcher.create(DescriptorMatcher.BRUTEFORCE);

                matcher.match(descripters01, descripters02, matchs);

DMatch[] tmp03 = matchs.toArray();

DMatch[] tmp04 = new DMatch[N];

for (int i=0; i<tmp04.length; i++) {

                        tmp04[i] = tmp03[i];

                }


        if (!this.cameraActive)

                {

                        // start the video capture

                        this.capture.open(0);
```

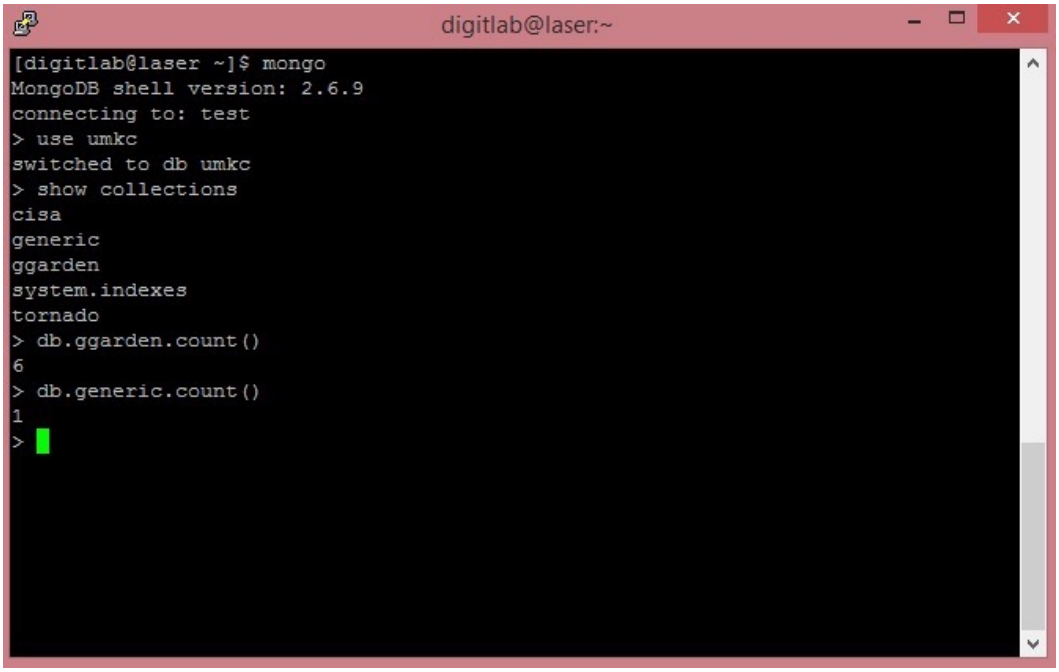
```
// is the video stream available?
if (this.capture.isOpened())
{
	this.cameraActive = true;

	// grab a frame every 33 ms (30 frames/sec)
	TimerTask frameGrabber = new TimerTask() {
		@Override
		public void run()
		{
			// update the image property => update the frame
			// shown in the UI
			Image frame = grabFrame();
			onFXThread(imageProp, frame);
		}
	};
	this.timer = new Timer();
	this.timer.schedule(frameGrabber, 0, 33);

	// update the button content
	this.cameraButton.setText("Stop Camera");
}
else
```

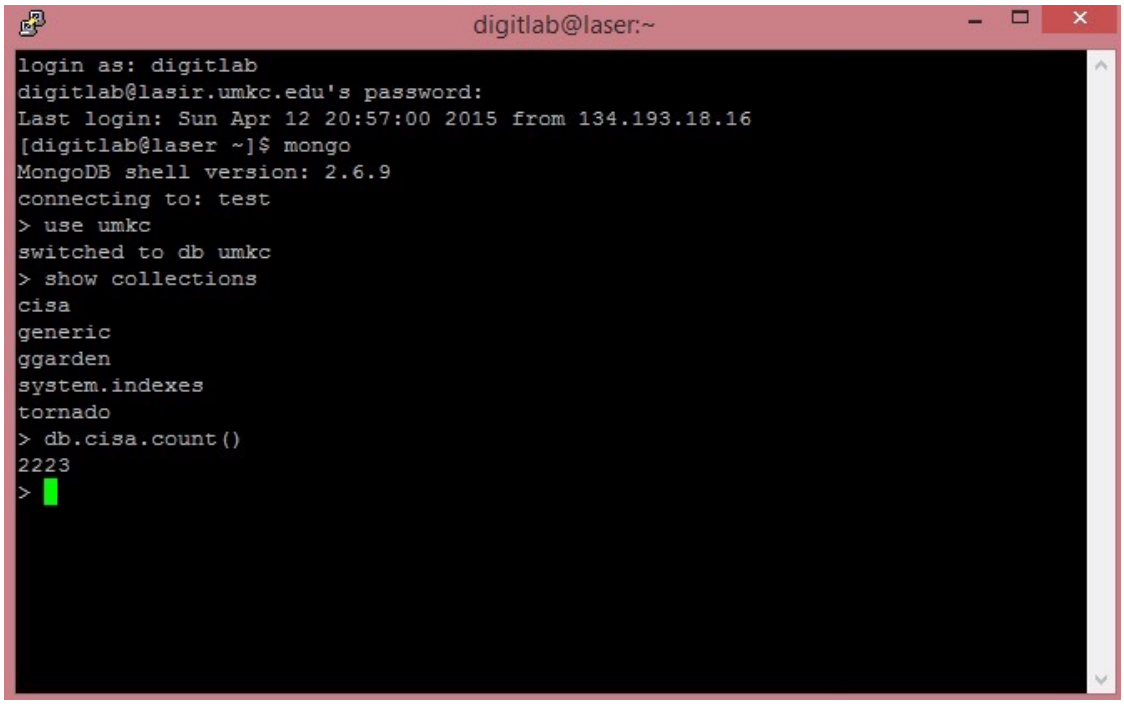
```
{
        // log the error
        System.err.println("Failed to open the camera connection...");
}
```

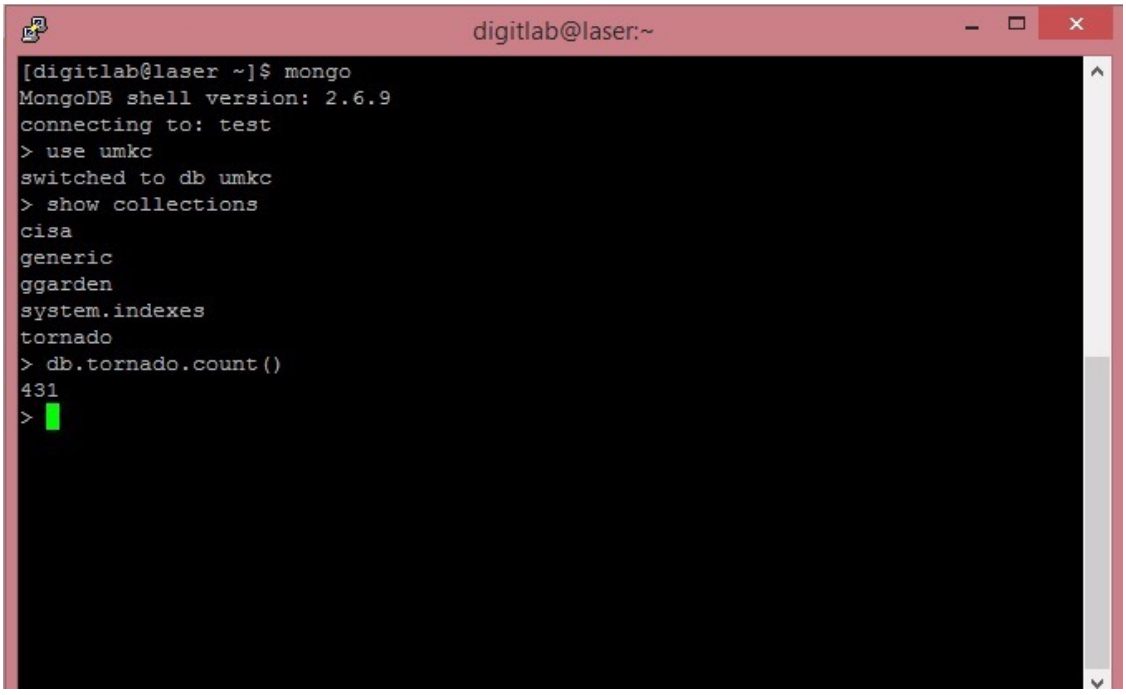
Status of all database(collections in mongo) used for all 3 applications(DS-Crowd, CISA, Green Garden)




digitlab@laser:~
login as: digitlab
digitlab@lasir.umkc.edu's password:
Last login: Sun Apr 12 20:57:00 2015 from 134.193.18.16
[digitlab@laser ~]$ mongo
MongoDB shell version: 2.6.9
connecting to: test
> use umkc
switched to db umkc
>


digitlab@laser:~
[digitlab@laser ~]$ mongo
MongoDB shell version: 2.6.9
connecting to: test
> use umkc
switched to db umkc
> show collections
cisa
generic
ggarden
system.indexes
tornado
> db.ggarden.count()
6
> db.generic.count()
1
>


digitlab@laser:~
login as: digitlab
digitlab@lasir.umkc.edu's password:
Last login: Sun Apr 12 20:57:00 2015 from 134.193.18.16
[digitlab@laser ~]$ mongo
MongoDB shell version: 2.6.9
connecting to: test
> use umkc
switched to db umkc
> show collections
cisa
generic
ggarden
system.indexes
tornado
> db.cisa.count()
2223
>


digitlab@laser:~
[digitlab@laser ~]$ mongo
MongoDB shell version: 2.6.9
connecting to: test
> use umkc
switched to db umkc
> show collections
cisa
generic
ggarden
system.indexes
tornado
> db.tornado.count()
431
>

Test Cases & Performance Evaluation

System Testing: Manually test all functions and check for bugs, in case bugs are found, debugging is done to check for bugs and fixes are proposed and implemented.

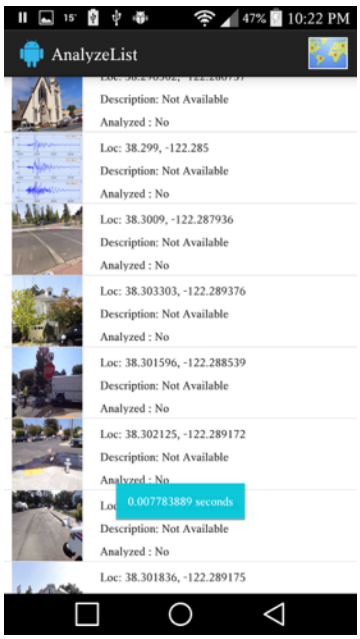
Unit Testing: We performed both NUnit & JUnit testing
NUnit testing to test the services
Test cases generated and tested for,
1. To retrieve the images from mongoDB(REST)
2. To retrieve the image IDs from mongoDB(REST)
3. Submit Observer Data to MongoDB(REST)

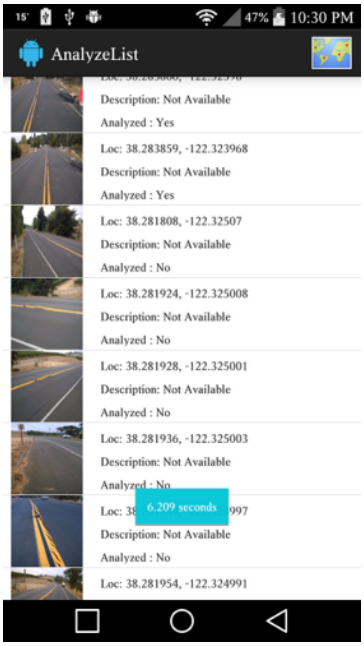
Evaluation:
Manual: Application was installed on 15 phones and it was used by 15 people at same time, no major bugs and performance issue found except the analyst list image loading takes time when all 15 phones are pulling the data at same time. It is mainly because of server limitation(4GB memory). All the major bugs from increment one were found and fixed, performance of application got improve drastically.

Automatic: We used traceview, which is integral part of android development to evaluate the application. It generates a big file, so we are sharing the dropbox link to traceview report.
Report can be seen here.
https://www.dropbox.com/s/4652qdmlx8x28ri/aseproj.html?dl=0

Time Taken

Loading non-cached images in Analyst List = 6.209 seconds
Loading cached images in Analyst List = 0.0077 seconds
Taking picture after being pushed to Mongo = 2.512 seconds

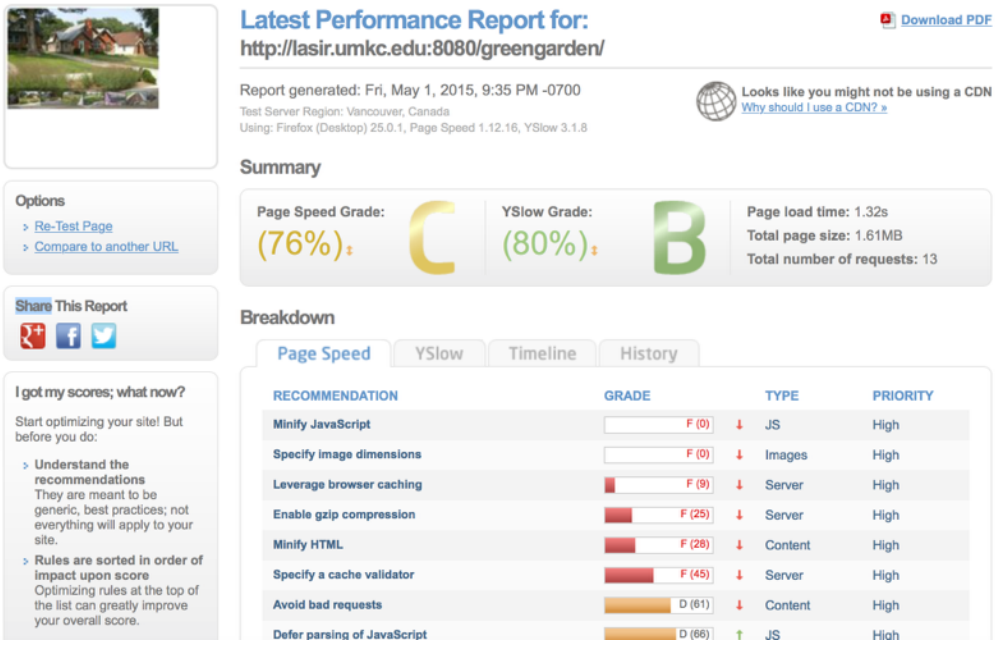




So the evaluation for mobile and web were done differently, after evaluating performance of android application, we used various technologies to test the web app of our project.

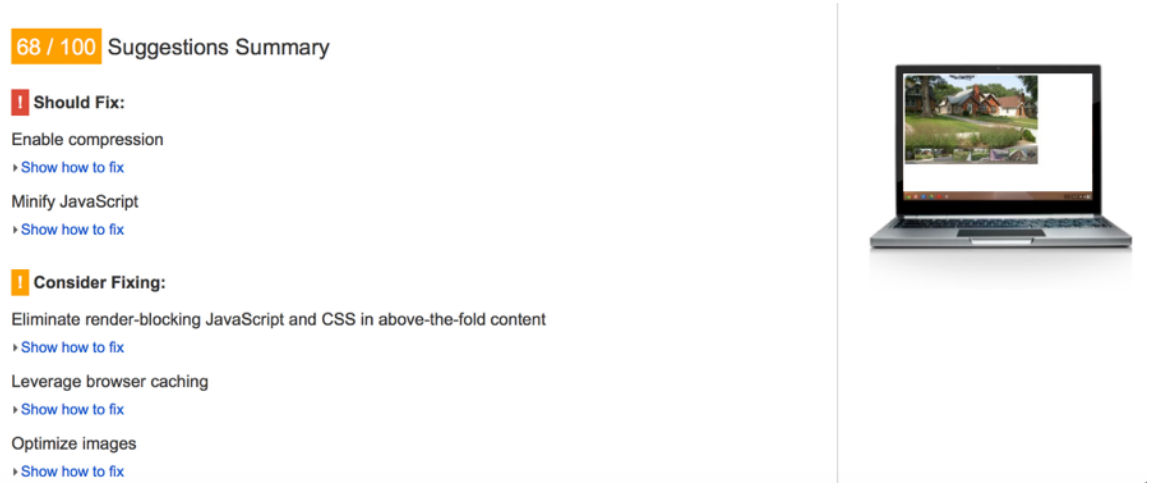
Using YSlow to Evaluate performance for our web page

http://lasir.umkc.edu:8080/greengarden/

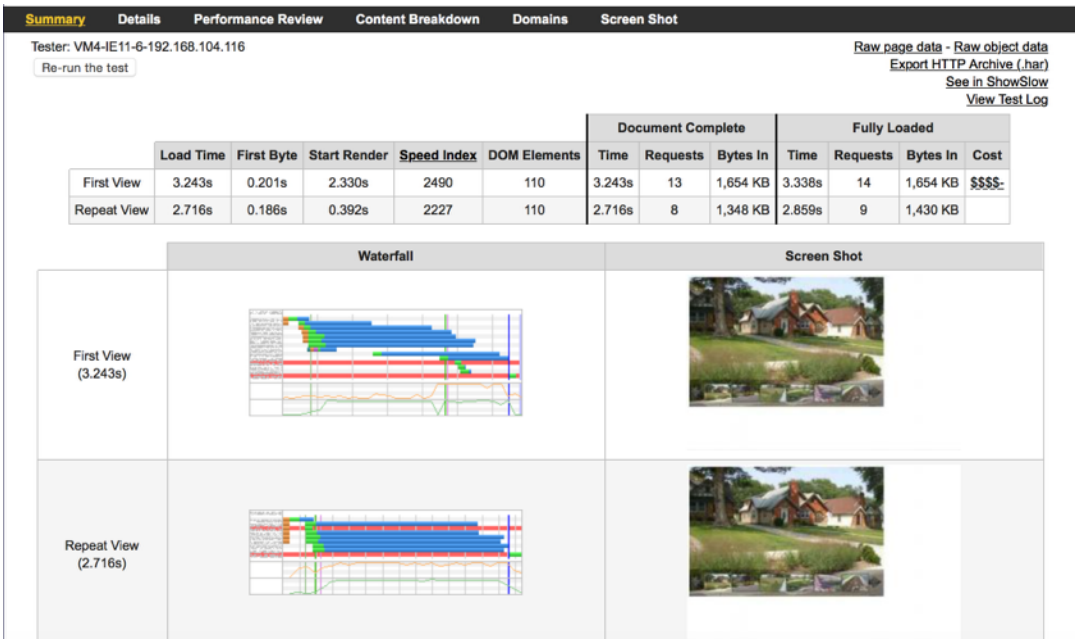


As we can see the results are decent from the test.

Using Google PageSpeed to test the same page



Using WebPageTest



Testing

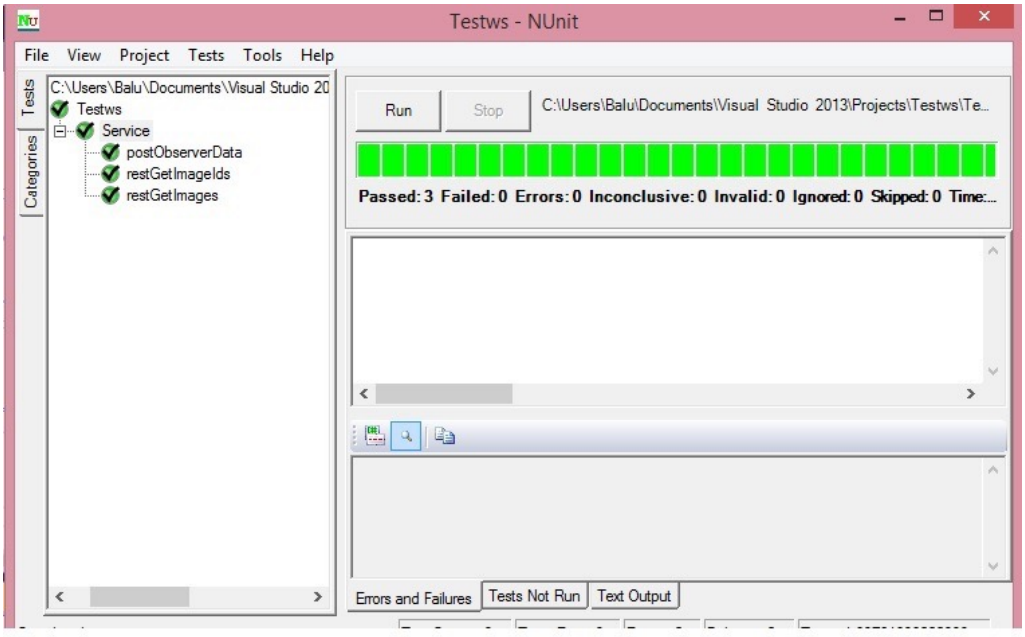
Unit test cases

NUnit

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using NUnit.Framework;
using System.IO;
namespace Testws
{
   [TestFixture]
   public class Service
   {
      [Test]
      public void restGetImages()
      {
         //To retrieve the images

         WebRequest req = WebRequest.Create(@"http://lasir.umkc.edu:8080/green-
gardenservice/webresources/ggarden/itemslist?start=0&limit=1");
         req.Method = "GET";
         req.ContentType = "application/json";
         req.Method = "GET";
         //req.ContentLength = 0;
         HttpWebResponse resp = req.GetResponse() as HttpWebResponse;
         if (resp.StatusCode == HttpStatusCode.OK)
         {
            using (Stream respStream = resp.GetResponseStream())
            {
               StreamReader reader = new StreamReader(respStream, Encoding.UTF8);
               Assert.AreEqual(HttpStatusCode.OK, resp.StatusCode);
            }
         }
      }
```

```
        [Test]
        public void restGetImageIds()
        {
  // To retrieve the Image IDs
            WebRequest req = WebRequest.Create(@"http://lasir.umkc.edu:8080/green-
garden/webresources/ggarden/sceneids?start=0&limit=1");
            req.Method = "GET";
            req.ContentType = "application/json";
            //req.Method = "POST";
            //req.ContentLength = 0;
            HttpWebResponse resp = req.GetResponse() as HttpWebResponse;
            if (resp.StatusCode == HttpStatusCode.OK)
            {
                using (Stream respStream = resp.GetResponseStream())
                {
                    StreamReader reader = new StreamReader(respStream, Encoding.UTF8);
                    Assert.AreEqual("54f3eed2e4b0ceb891145128", reader.ReadToEnd());
                }
            }
        }

        [Test]
        public void postObserverData()
        {
            //Post the Observer Data to MongoDB

            string data = "{ color: \"red\", value: \"#f00\" }";
            WebRequest req = WebRequest.Create(@"http://lasir.umkc.edu:8080/green-
gardenservice/webresources/ggarden/observerdata");
            req.Method = "POST";
            var data1 = Encoding.ASCII.GetBytes(data);
            req.ContentLength = data1.Length;
            req.ContentType = "application/json";
            using (var stream = req.GetRequestStream())
            {
                stream.Write(data1, 0, data1.Length);
            }

            HttpWebResponse resp = req.GetResponse() as HttpWebResponse;
```

```
            if (resp.StatusCode == HttpStatusCode.OK)
            {
                using (Stream respStream = resp.GetResponseStream())
                {
                    StreamReader reader = new StreamReader(respStream, Encoding.UTF8);
                    Assert.AreEqual(HttpStatusCode.OK, resp.StatusCode);
                }
            }
            else
            {
                Assert.AreEqual(null, resp.StatusCode);
            }

        }
    }
}
```



JUnit

```
package com.cisa.androidapp.test;
```

```
import java.util.ArrayList;
import java.util.List;

import android.test.ActivityInstrumentationTestCase2;
import android.widget.RadioButton;
import android.widget.TextView;

import com.cisa.androidapp.*;
public class Test1 extends ActivityInstrumentationTestCase2<MainActivity> {

private MainActivity mActivity;
RadioButton rb;
private String resourceString;

public Test1(Class<MainActivity> activityClass) {
super(activityClass);
// TODO Auto-generated constructor stub
}
@Override protected void setUp() throws Exception {
super.setUp();
mActivity = this.getActivity();
List<String> names = new ArrayList<String>();
rb = (RadioButton) mActivity.findViewById(com.cisa.androidapp.R.id.plant);
rb.setChecked(true);

}
public void testPreconditions() {
assertNotNull(rb); }
public void testText() {
assertEquals(1,rb.isChecked());
}}
```

Deployment

ScrumDo

https://www.scrumdo.com/projects/project/dscrowd/iteration/121775

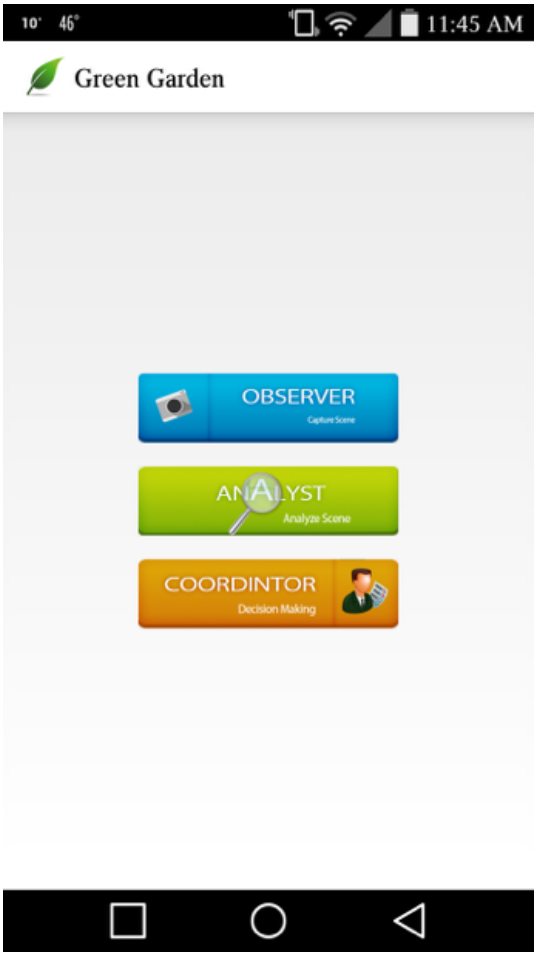
http://lasir.umkc.edu:8080/

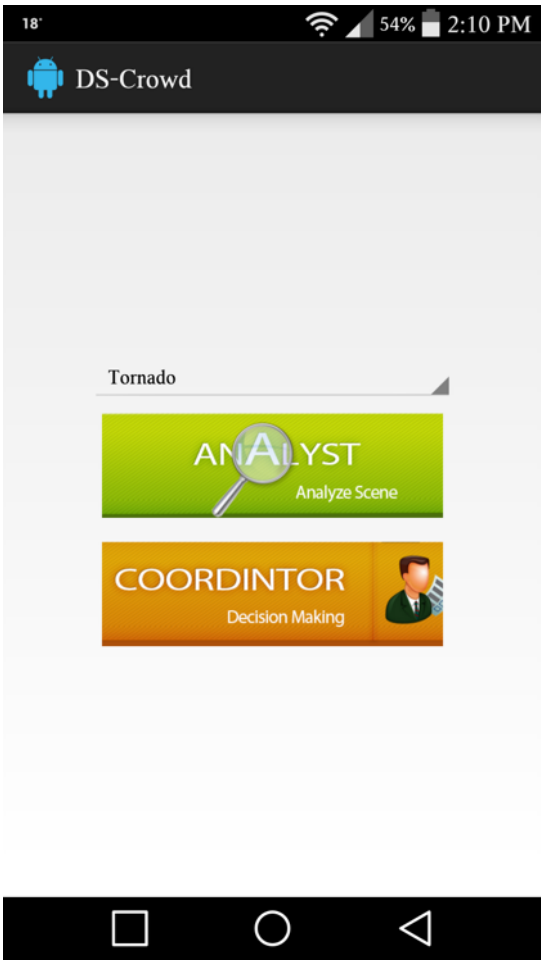
Mobile App Website

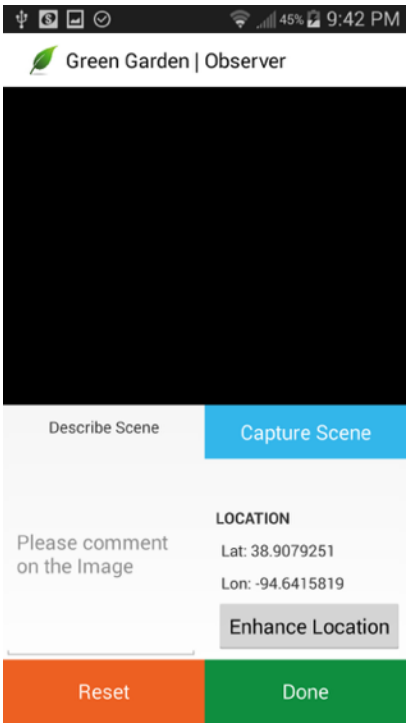
http://lasir.umkc.edu:8080/greengarden/ (Soon to be updated)
http://lasir.umkc.edu:8080/serviceengine/ (Soon to be updated)

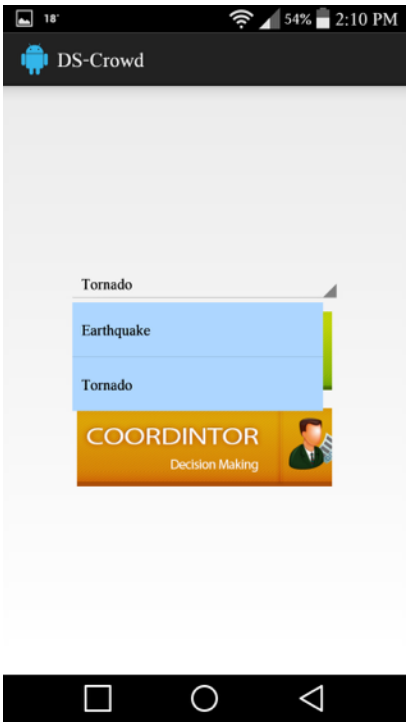
Report:

Screenshots



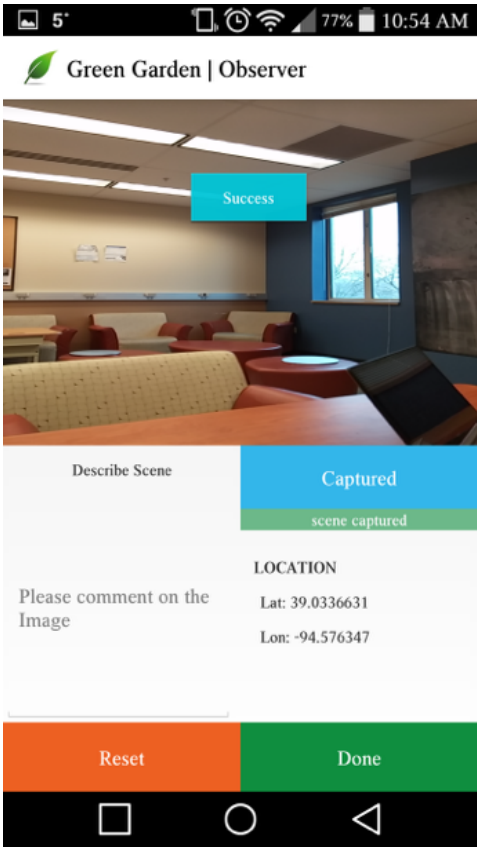


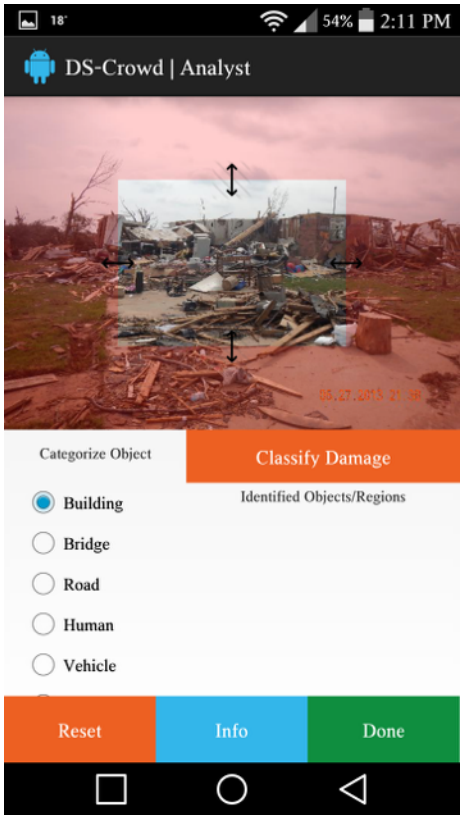

45% 9:42 PM
Green Garden | Observer
Describe Scene
Capture Scene
Please comment on the Image
LOCATION
Lat: 38.9079251
Lon: -94.6415819
Enhance Location
Reset
Done

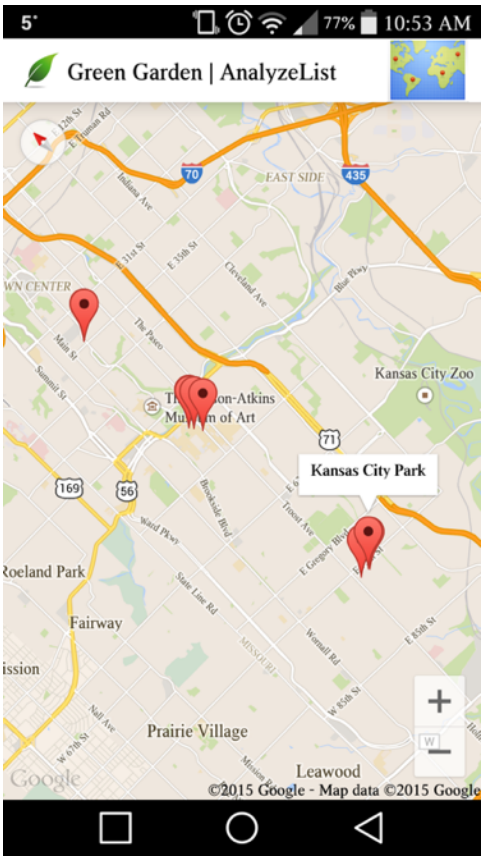

54% 2:10 PM
DS-Crowd
Tornado
Earthquake
Tornado
COORDINTOR
Decision Making


54% 2:10 PM
AnalyzeList
Loc: -97.464066, 35.332771
Description: Residence , some foundation remains...
Analyzed : Yes
Loc: -97.464056, 35.332957
Description: Residence , roof and garage damaged
Analyzed : Yes
Loc: -97.464567, 35.333256
Description: Residence , negligible roof damage
Analyzed : Yes
Loc: -97.488576, 35.329382
Description: Residence , negligible roof damage
Analyzed : Yes
Loc: -97.488179, 35.32977
Description: Residence , roof damaged
Analyzed : Yes
Loc: -97.487978, 35.329748
Description: Residence , negligible roof damage
Analyzed : Yes
Loc: -97.487962, 35.329365
Description: Residence , roof and side damage
Analyzed : Yes

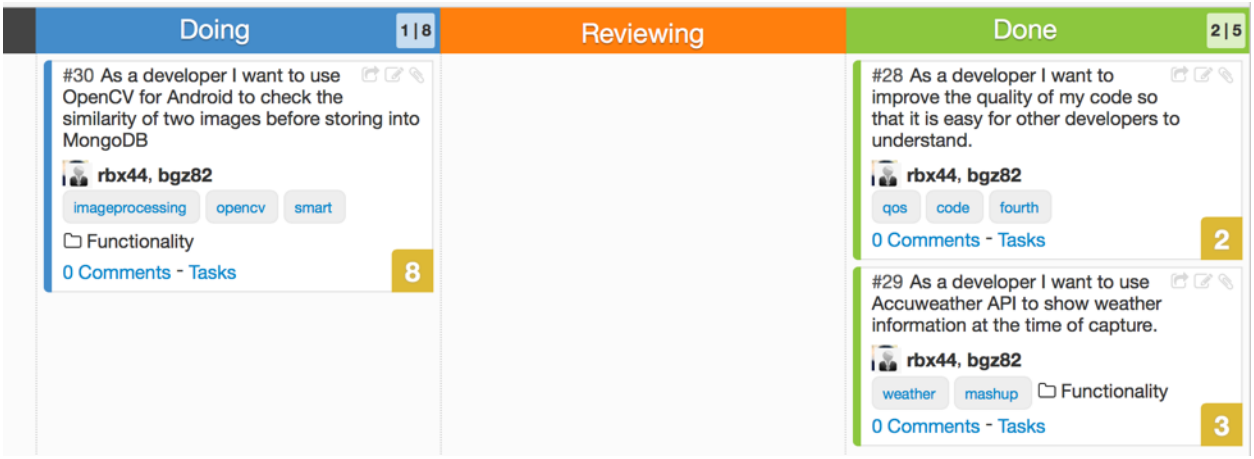

53% 2:11 PM
Coordinator | Map
Morocco
Algeria
Libya
Egypt
Iraq
Saudi Arabia
Niger
Chad
Sudan
Yemen
Burkina Faso
Nigeria
South Sudan
Ethiopia
Gulf of Guinea
Somalia
Kenya
Gabon
DR Congo
Tanzania
Angola
Zambia
Mozambique
Madagascar
Botswana
South Africa
Google
©2015 Google - Map data ©2015 Google

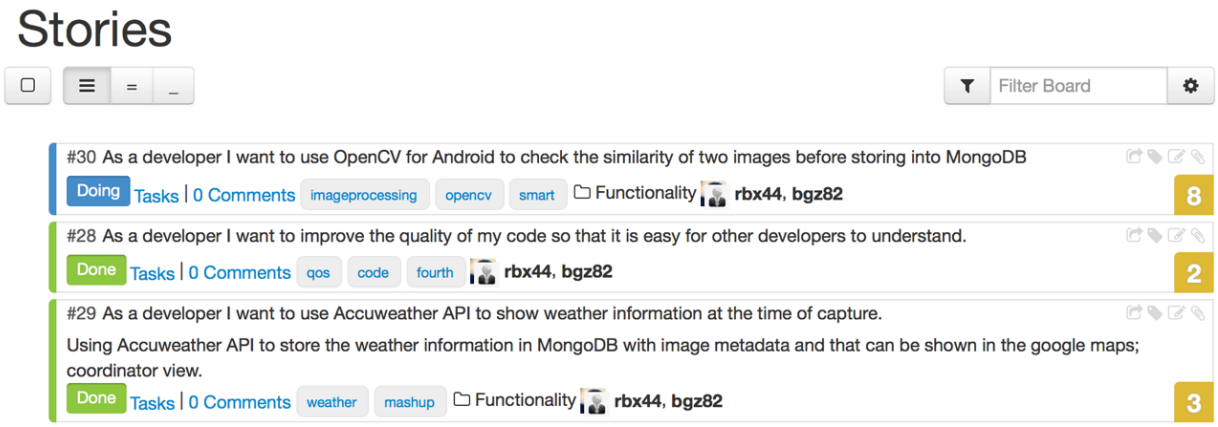

5°
77% 10:54 AM
Green Garden | Observer
Success
Describe Scene
Captured
scene captured
Please comment on the Image
LOCATION
Lat: 39.0336631
Lon: -94.576347
Reset
Done


18°
54% 2:11 PM
DS-Crowd | Analyst
Categorize Object
Classify Damage
Identified Objects/Regions
Building
Bridge
Road
Human
Vehicle
Reset
Info
Done


5°
77% 10:53 AM
Green Garden | AnalyzeList
EAST SIDE
Kansas City Zoo
Kansas City Park
Fairway
Prairie Village
Leawood
©2015 Google - Map data ©2015 Google


18°
53% 2:11 PM
GS-Crowd | Details
[-97.464066, 35.332771]
Description: Residence , some foundation remains, severe debris
Exit
Analyze





Implementation status report

Story #28:As a developer I want to use Accuweather API to show weather information at the time of capture. Using Accuweather API to store the weather information in MongoDB with image metadata and that can be shown in the google maps; coordinator view.

Status: Done

Responsibility: Bhargava & Rishabh

Time Taken: 3 Hours

Contributions: Bhargava(50%) & Rishabh(50%)

Story #29: As a developer I want to improve the quality of my code so that it is easy for other developers to understand.

Status: Done

Responsibility: Rishabh & Bhargava

Time Taken: 6 Hours

Contributions: Bhargava(50%) & Rishabh(50%)

Story #30: As a developer I want to use OpenCV for Android to check the similarity of two images before storing into MongoDB

Status: Couldn't be done in the context

Responsibility: Rishabh & Bhargava

Time Taken: 15 Hours

Contributions: Rishabh(50%) & Bhargava(50%)

Total Time Spent: 24 Hours (Done), 18 Hours (Doing)

Issues/Concerns

Doing more to what could be possibly a public application, to find users and taking their input to make it better.

No background in Image processing. Have to read a lot, went through so many papers and codes to understand and to be able to use in application

Another difficulty is team size(2 people) and the amount of work that is required to carry the endeavor is a lot & timing issues.

Dealing with very flexible requirements Low server memory and high CPU usage.