My City Fixer

"A creative mobile application that helps Toronto be a cleaner and safer city by allowing citizens to report and track non-emergency issues"

April 7th, 2011

Chris Boyle - 998618642
Thanesh Sunthar - 992420956

FINAL REPORT

Report Word Count: 2460 (Excluding Title page and Table of Contents)

ECE 1778 - Winter 2011
Prof. Jonathan Rose

Table of Contents

1 Introduction and Motivation

Global cities often evoke mental images of clean and futuristic infrastructures. Appearances at the street level can greatly affect a tourist's perception of cleanliness and their safety. A highly ranked, world class city does not evoke images of wrecked sidewalks, litter, and potholes.

Figure 1: Application that simplifies problem reporting

Our motivation was to develop an application that helps Toronto be cleaner and safer city, by allowing citizens to report and track non-emergency issues.

2 Objective

Currently, the City of Toronto allows residents to request clean up and fixing services for metropolis property via phone or online reporting to Toronto 311. A person can report various problems such as potholes, overflowing litter bins, missing street signs, etc. This system is easily accessible but requires time and effort.

Consider the following example: a citizen walking around downtown Toronto encounters a pothole and wishes to report it. Under the current system the individual must either call Toronto 311 or note the problem (and location) to report it later. Phoning is the only immediate option; however, some prefer not to call – perhaps due to the limited cell phone airtime or not wanting to wait to speak to the next operator. Even if the wait time is zero, describing the problem, location, and contact information will take additional time. If the citizen decides to report the problem later, there is a chance of getting distracted with other day-to-day activities and overlooking to report the issue when they return home.

The main objective is to streamline this reporting process with minimal user involvement. Using our unique mobile reporting application, the user takes a picture of the problem and identifies the crisis type from a drop down menu. In addition, our mobile application records the GPS location and resolves the user's current address. The whole reporting process takes less than a minute and more useful information is passed to the City than the phone or other online communication methods.

3 Platform Architecture

MyCityFixer platform has two major components: The android application and the mycityfixer.com web administration tool.

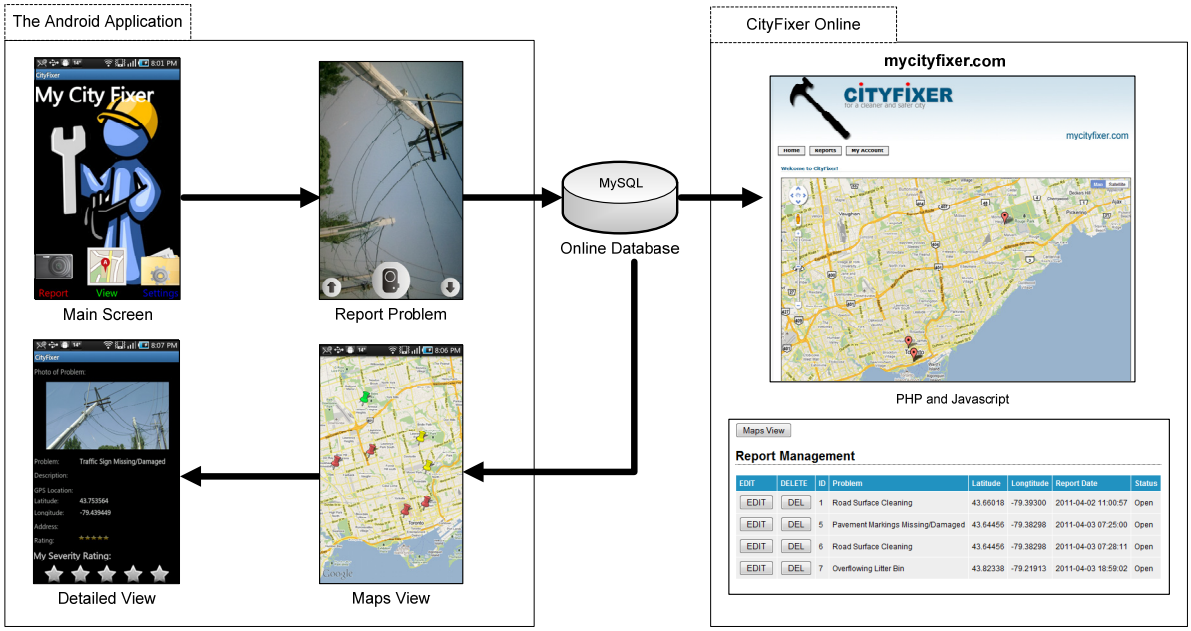

Figure 2: MyCityFixer high level platform architecture

Figure 2, shows the MyCityFixer platform architecture at a very high level. The reporting data and picture flows through several activities within the android application and eventually gets stored in the online database. This data is then available for viewing through two different venues. One is through the mycityfixer.com web interface for the public to view and the other is through the phone app for personal view.

3.1 Android Application Architecture

The Android application has a reporting system and a problem viewing system. Figure 3 below elaborates the various activities that are present and the interaction among those activities. As seen in the diagram the reporting section, the viewing section and the web database are all tightly coupled together for the platform to function smoothly.

MyCityFixer
Application DataFlow

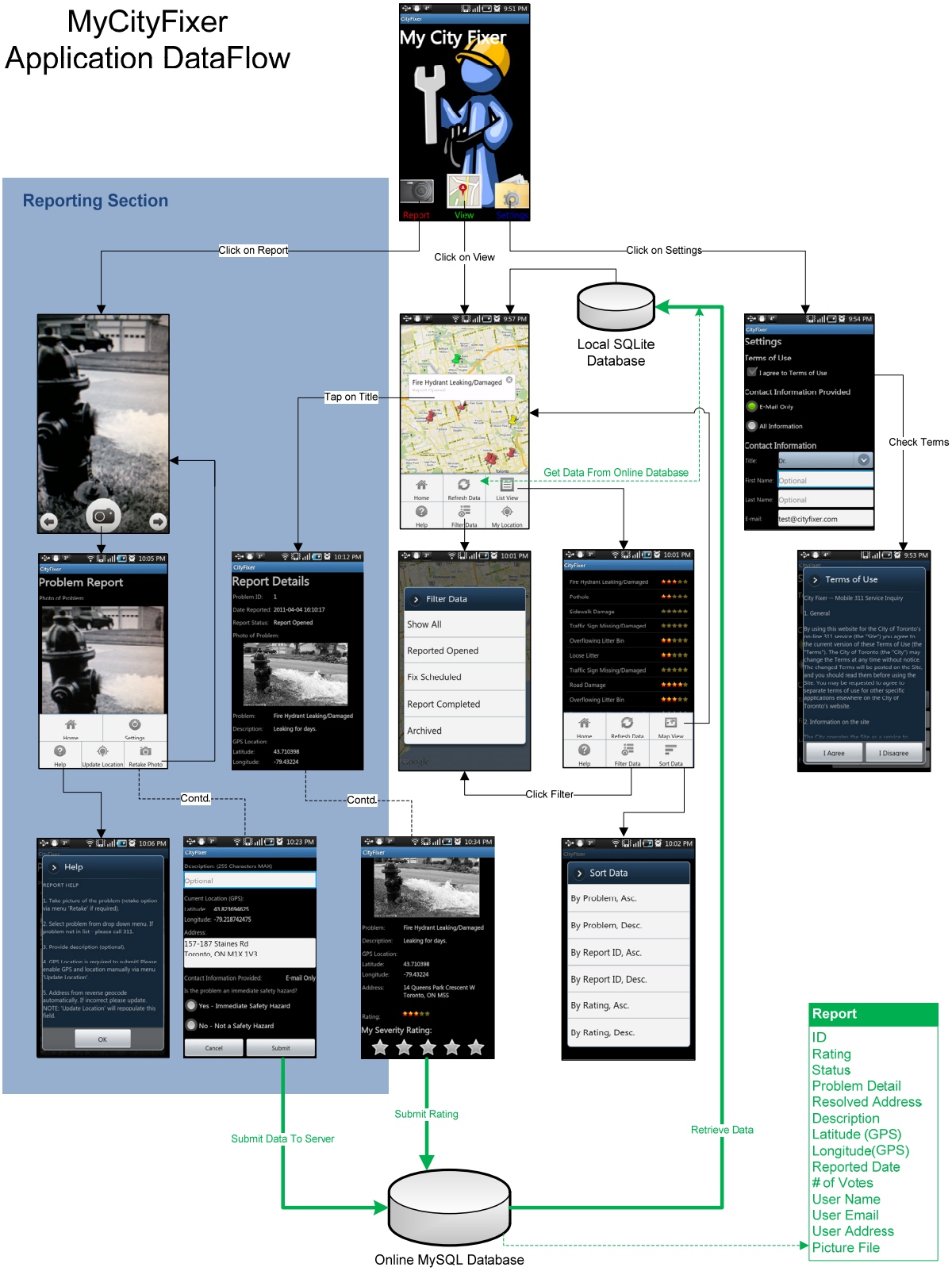

Figure 3: MyCityFixer Android Application Architecture

3.1.1 Reporting System

The reporting system will follow the style currently found on the City of Toronto 311 website (http://www.toronto.ca/311). This allows the user to directly report the following problems:

- Cleaning
  - Overflowing Litter
  - Loose Litter
  - Road Surface Cleaning
  - Blocked drain
- Fixing
  - Pothole
  - Sidewalk and Road Damage
  - Damaged / Missing street and traffic signs
  - Pavement markings missing
  - Fire hydrant damaged
  - Bike ring damaged or missing (added by us!)

These represent the most common problems according to Toronto 311. However, should the user wish to report a problem that is not listed in the application, the user is presented with the option to phone 311 directly. In addition to the problem type, the user must provide the address and optional description of the problem.

To use the reporting system, user has to agree to the terms of use and provide their contact information. To make available additional information for city workers, the user can supply a photo of the problem using the phone camera and attach this to the report. The latitude, longitude, and address location will be automatically populated by GPS and the Google Maps API. Emergency problems can also be reported directly to 311. See Figure 3.

3.1.2 Viewing System

Currently, the City of Toronto 311 system does not allow the users to view the various problems that are outstanding in the city. Our system will implement a feature to view all the troubles reported in the city, increasing the transparency of how the city responds to these infrastructure problems. The user can view the location, type and severity of the problem.

These problems plotted on the Google Maps are represented through various colored icons, corresponding to different problem status. When the user selects a problem, basic information is provided in a popup. Clicking this popup takes the user to a detailed description page with the problem picture. In the detailed view users can choose to rate the problem depending on its severity. This information is then fed back to the server. Consequently, the highly ranked problems can help the City in prioritizing the fixes that are of immediate concern. This in return should increase efficiency in resource allocation and city safety.

The user can also view problems in a list format, with the ability to sort / filter problem by status, ID, type, and severity rating. Clicking on any problem from the list view will also take the user to the detailed information page. See Figure 3.

3.2 Web Server Architecture

The server component was originally developed to centralize the database, such that public can access the problem information posted by individual users. This server component matured in later spirals into a full online platform that allows public to view problem information from anywhere in the world. In addition, problems can also be viewed and edited online by a system administrator. For this project we also purchased a domain called mycityfixer.com and pointed the DNS to a physical server located in California. The server environment hosts RedHat Fedora Core 4 operating system. Server applications such as Apache 2, PHP 5 and MySQL 5 were installed, to handle all server related transactions. Figure 4 shows the basic web data flow of our platform. Web interface used for public consumption is very intuitive. It consists of Google maps view where all the problems are plotted. This works very similarly to the android application. In addition there is also a management console available for administrator to view and edit the problem reports.

Admin user login/pass is admin@mycityfixer.com/testing.

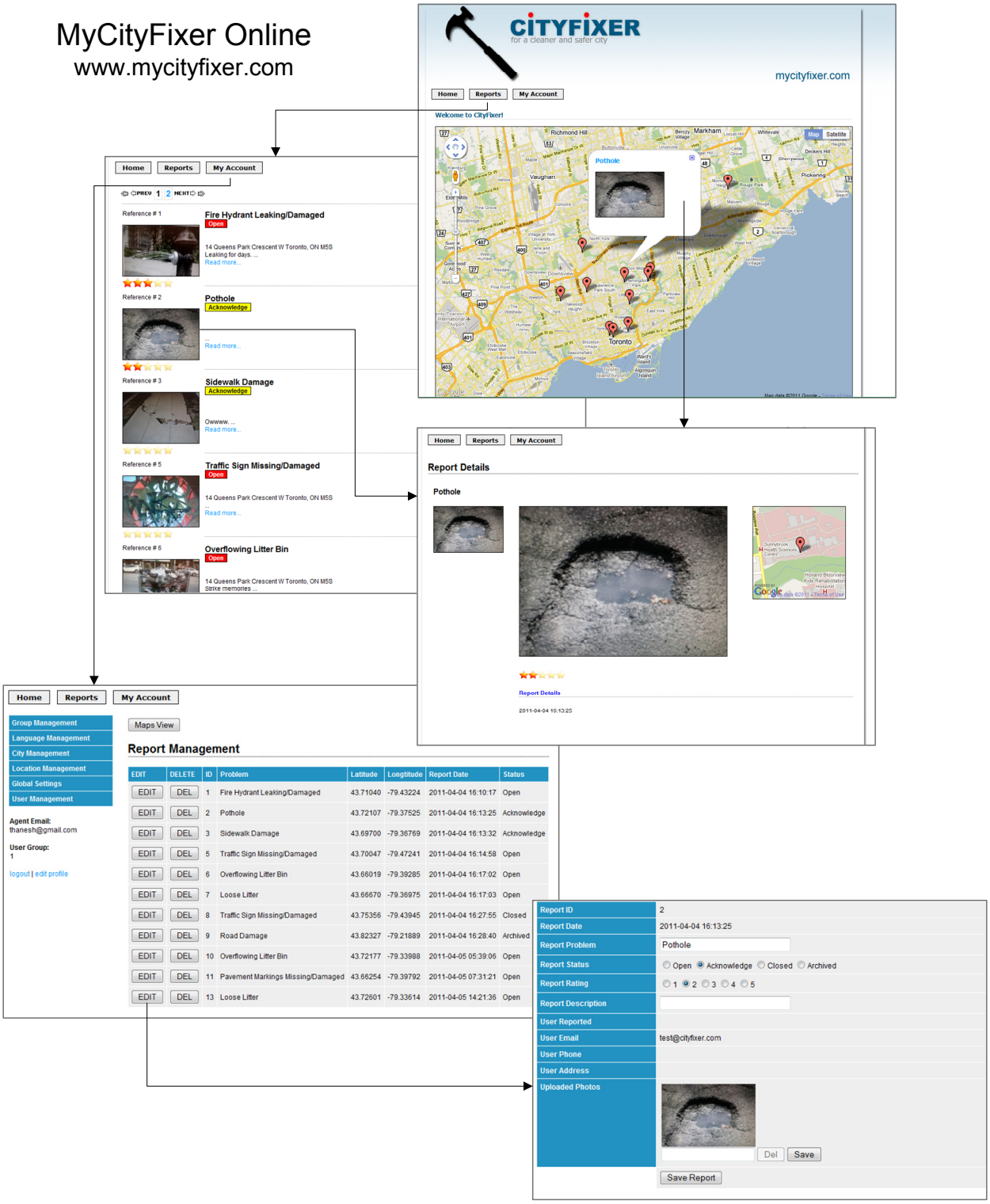

Figure 4: MyCityFixer Web Architecture

4 Application Functionality

This section briefly describes the entity relationship between the various activities and the interaction with the web server. Later in this section detailed description of each android activity is provided.

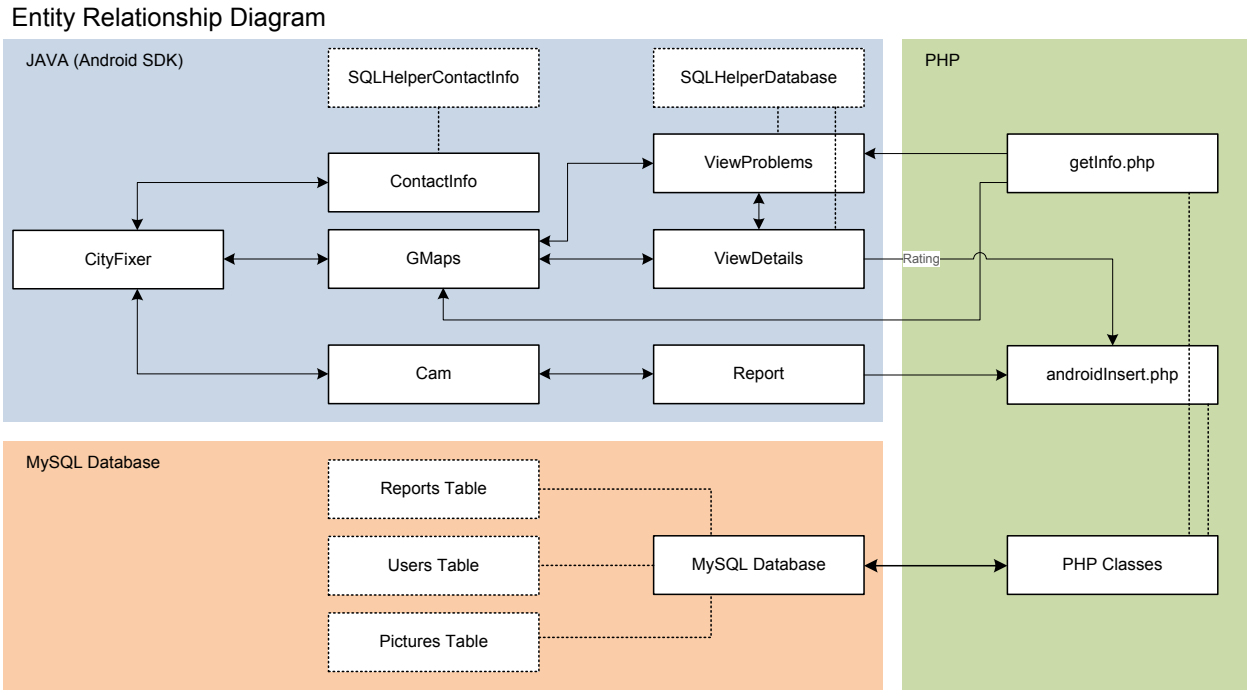

Figure 5: Entity Relationship Diagram of MyCityFixer Platform

Figure 5, shows three major components that construct the MyCityFixer platform and the interaction among them.

The navigation layout provides easy and intuitive access to the reporting and problem viewing systems. Below detailed information on functionality is provided for each screen.

Main Home Screen - MyCityFixer (CityFixer.java)

- Main navigation screen allows user to **Report** and **View** problems, as well as, update their **Settings**.
- The user must first input basic contact information (e-mail only) and agree to the terms of use before submitting a report. If the user clicks "Report" before doing this, a toast will remind them to do so.

Figure 6: Home Screen

Settings Information (ContactInfo.java)

- The user is required to input (at least) e-mail to report and agree to terms of use. This will be completed upon the first loading of the program and stored in database; thereby, preventing the user from having to re-input each time.
- The user can optionally supply additional contact information (e.g. Full Name, Phone Number, Address, etc.) and update the information at any time.
- The user can select what contact information is provided.
- Clicking the "Terms of use" overlays an agreement to which the user responds. It is currently based off the Toronto 311 agreement.

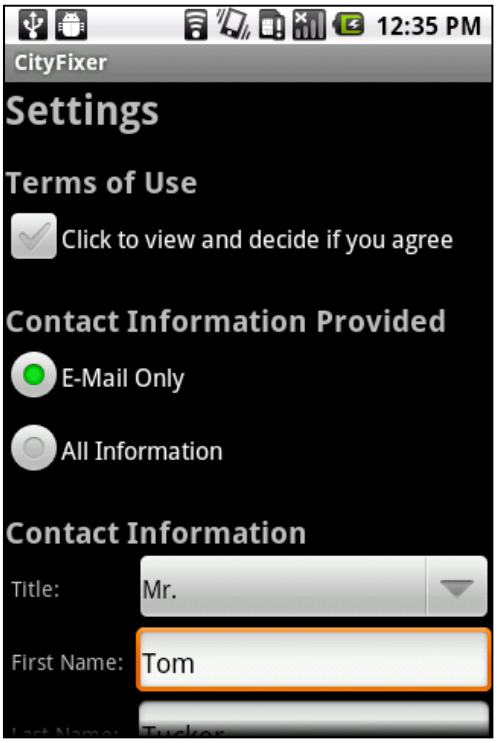

Figure 7: Settings

Problem Photo (Report 1 of 2) (Cam.java)

- The user can take optional photo of the problem to submit with the report.
- The buttons rotate to screen orientation and after a picture is taken a delete and check mark icon appear to facilitate a retake or continue to the **Problem Report page** with the current picture.
- The photo file is rotated to current orientation and compressed to 1 Megapixel to save data.
- The "forward" button will take user to directly **Problem Report** page.

Figure 8: User supplied photo

Problem Report (Report 2 of 2) (Report.java)

Scrollable Layout with:

1. Current problem of photo (if applicable). The user can retake the photo via menu button.
2. Drop down menu of various problems in the 311 database. If the user selects "Other", a prompt will appear to call 311 directly via auto-dial.
3. Optional description of problem.
4. Current location as given by the GPS reported in longitude and latitude. User can update via menu.
5. Reverse Geocoding from Google Maps API automatically populates the address. Editable in case of slight discrepancies. User can update via menu.
6. Shows user preference for contact info, which can be changed without data loss via menu.
7. Immediate safety hazard will prompt user to call 311 via auto-dial feature and prevent upload of data.
8. *Cancel* will prompt a "confirm" screen to prevent accidental closure. *Submit* will close report, only if the data is successfully uploaded.

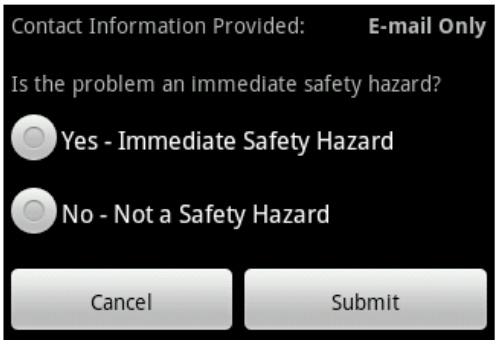

Figure 11: Immediate safety hazard will prompt user to call 311 via auto-dial (3 of 3)

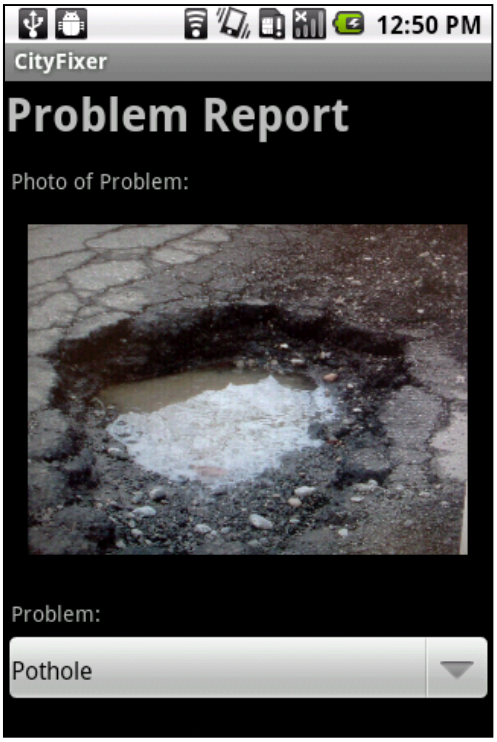

Figure 9: Expanded view of user supplied photo (1 of 3)

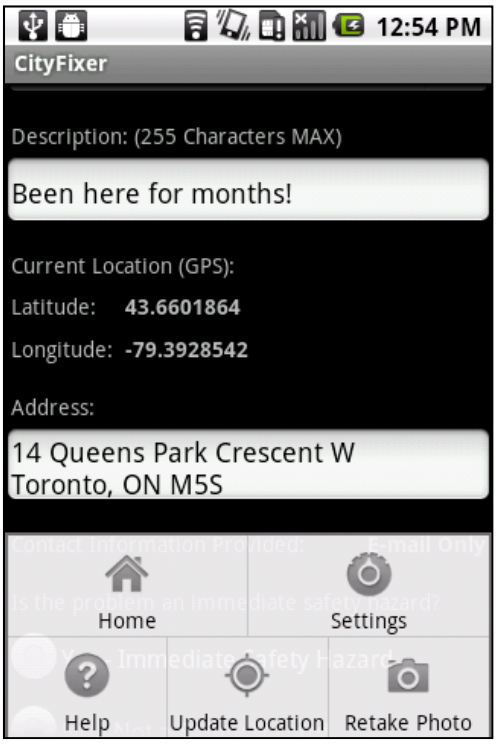

Figure 10: Address auto-populated via reverse Geocoding - user can manually update (2 of 3)

Map View (View Problem 1 of 3) (GMaps.java)

- Google Maps View of all problem data. Full functionality to move around Toronto and click the various problems (pushpins).
- Pin colors correspond to the problem status: Red - Report Opened, Yellow – Fix Scheduled, Green – Complete, White – Archive.
- Problem pins can be filtered by type.
- Clicking on the pin will provide basic and detailed **Problem Information**.
- My location navigates to current user location.

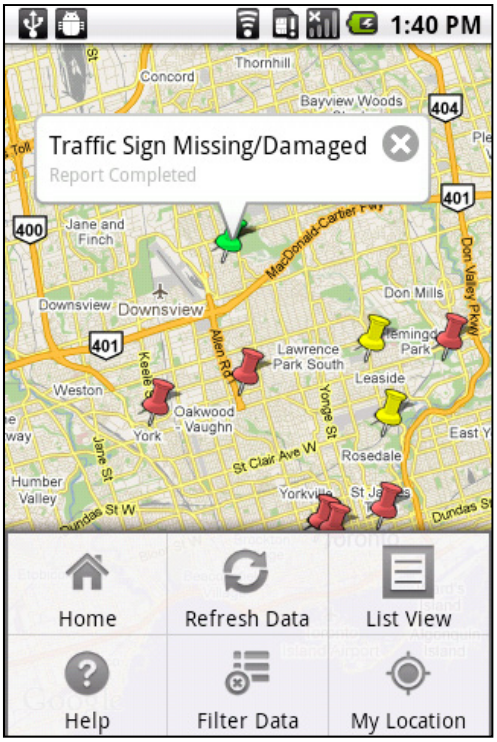

Figure 12: Google Maps View

List View (View Problem 2 of 3) (ViewProblems.java)

- Scrollable list view of all problem data.
- Can be sorted by Problem ID, type, or rating in both ascending and descending order.
- Can be filtered by problem status.

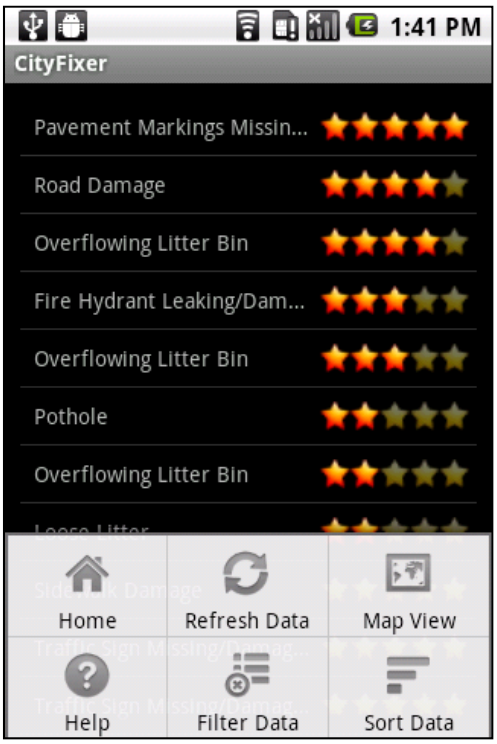

Figure 13: List View

Problem Information (View Problem 3 of 3) (ViewDetail.java)

Shows the following information:

1. Problem ID.
2. Date and time reported.
3. Problem photo (if applicable).
4. Problem type.
5. Description.
6. Location GPS and Address.
7. Current rating
8. My Severity Rating – uses averaging to calculate new rating.

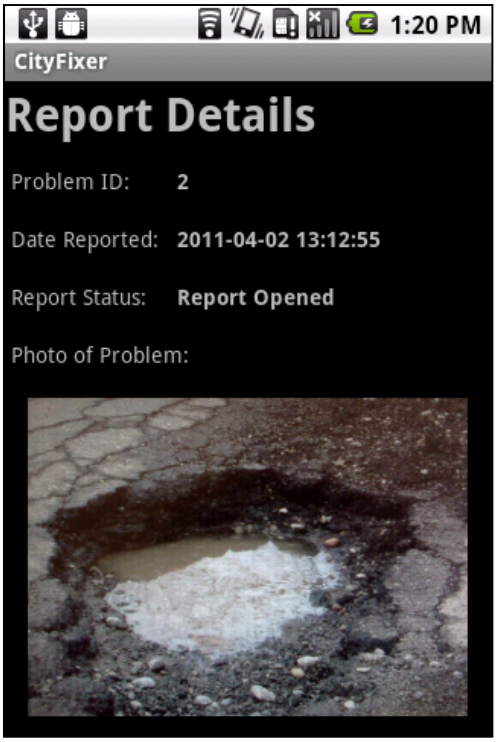

Figure 14: Detailed Problem Description (1 of 2)

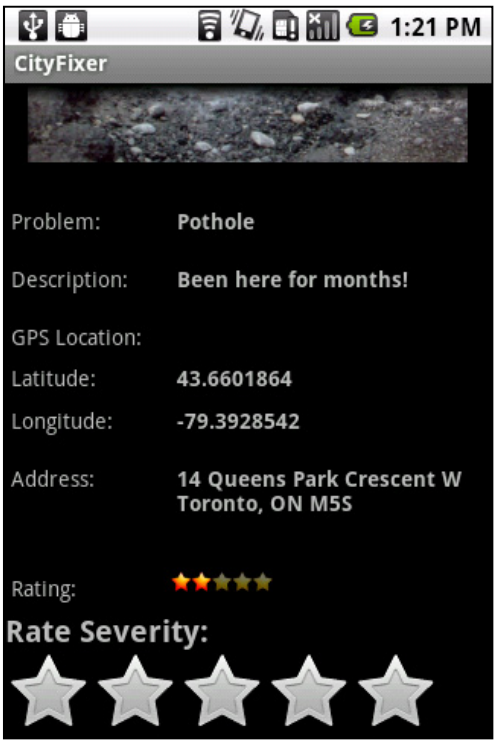

Figure 15: Detailed Problem Description (2 of 2)

5 Statement on Functionality: What works? What doesn't?

The final application works with all the must have features that were originally planned, plus features that we added along the way. Reporting of problem and viewing of problem through an intuitive interface was the required core functionality. We added numerous features: sorting, filtering, help pages, overlaying current location, photo retake, etc. to augment the experience.

The only feature, which we initially proposed and did not implement, was the checking system to prevent multiple reports of same problem. We decided it was computationally intense as it required a persistent data connection with the database. This would also significantly increase the bandwidth used by the application. One solution was to implement this in a non-persistent way. Since this was a nice to have feature, we decided to leave this for later revisions.

6 Lessons Learned

The process of creating a complete mobile application was both extremely challenging and rewarding. This application, and corresponding server implementation, was a ***huge*** time investment. The key lessons learned are summarized below:

1. **Code Management**: We utilized Apache Subversion (SVN) for code management. This system maintains the current and historical versions of the source code on a central server. This allowed both of us to work on the most recent code *simultaneously* without constantly having to e-mail the code back and forth.

2. **Time Management:** It is extremely difficult to estimate the time required to implement a feature or debug. For example, implementing the camera took less time than removing an XML layout bug that stretched the columns (took 4 hours). We found that if a bug was difficult to resolve, it was better for fresh eyes to debug the problem.

3. **Spiral Design Method**: The spiral or incremental design methodology worked well. Both of us have experience in this method; as a result, we set hard deadlines for core functionality and mapped out the most valuable additional features. We also set contingency plans for the unknown difficulty level of certain implementations (e.g. proposed web server vs. local SQL database contingency plan).

7 Next Steps and Conclusions

We believe our application is nearly production ready. However, the features mentioned below and additional testing would be necessary before going online for public use.

1. **Production Level Testing:** An extended beta test with multiple phones for interoperability. Tests with large sets of data, simultaneous user interaction with online database for concurrency testing.

2. **Filtering of Data:** Filtering of the data by administrator, before posting online would be a useful feature to have.

3. **Application Features:** Ability to search by problem, area, status, rating etc. would be very useful. Showing features relevant to user location would also help reduce the bandwidth used by the application.

We definitely believe it can be commercialized and used in various cities as a problem management system. At present we are engaging with the City of Toronto to get their interest.

Appendix: Contribution of Group Members

We in cooperation believe that the division of work was completely appropriate and both members had significant and equal contributions to the project.

Chris Boyle: Worked exclusively on the mobile application side of the project. Main contributions: creating the reporting system (both camera and main report page features – GPS, reverse geocode, etc.), the settings system (contact information, SQL database, and terms of use), and the detailed view page. Added the menu features for all pages (help pages, retake photo, reload data, home button, etc.). Modifications to map and list views: various pin overlays, my home location, reloading, and the filtering/sorting functionality.

Thanesh Sunthar: Setup the SVN/Trac and web server environment (in Linux). Worked on the Google Maps view, List View, Rating System, interaction with the server to report the problem, (including storing of online information to local database). mycityfixer.com development - Google maps (V3) view, web administration, web list view. There were also few overlaps with Chris's work during the debugging/testing phase.