TalKivi Mobile Client

Team Members:
Peter Choi
Andrew Furze
Kameron Kincade
Ryan Langewisch
Joshil Singh

Client:
Brian Krzys

June 19, 2013

Introduction

Newmont Mining is one of the world's largest gold mining corporations. With offices and assets worldwide, the company itself is headquartered in Denver, Colorado. The TalKivi Mobile Client is geared to collect and upload different data points to a TalKivi server for reporting. Although Newmont plans to use the application for geoscience data, such as soil samples, the application itself is designed to be more flexible and can be used as a general data collection and aggregation tool. When downloading a form, the view is dynamically generated into a template based on information downloaded from the server. When uploading a form, the data collected from the mobile application is submitted using one of these templates.

The client, Brian Krzys, initially developed the idea for the application for use with Newmont; however, his vision quickly grew to incorporate other forms of data. For instance, a user could use the application to develop their own templates and log their favorite restaurants they have visited across the country. Krzys concurrently developed the TalKivi server in coordination with the development of the application itself. Apart from the application itself, Krzys was also interested in the differences and similarities between two cross-platform implementations - PhoneGap and Titanium Alloy. Therefore, TalKivi was implemented using these two approaches while comparing and contrasting the two methods.

Product Vision:

With regards to data collection, TalKivi Mobile is just the start. One half of the vision handled the user experience itself - downloading forms from a server, rendering them on a mobile device, and allowing users to easily enter and upload data. The interface needed to be clean and uncluttered, despite all the variance in the forms themselves. The other half of the vision dealt with how the application handled data synchronization. The application must take into account the device's data connection and whether or not the data has already been synchronized to the server. TalKivi Mobile is the solution to this problem - intelligently handling network states and data synchronization. Leaving data synchronization issues for the app to handle, the only thing the user has the worry about are the data entries themselves.

Requirements

The exact requirements for the this project were deliberately loose to facilitate the future expansion of the design to any type of data collection. However, the application itself had several specific behavioral and functional requirements. The application needed to connect to a remote server, display a list of available forms, and allow the user to download and fill out these forms. Once the user completed a

form, the TalKivi client needed to store it locally to device and upload it to a server once a data connection is present.

When the user selects a locally available template, that template needs to be dynamically generated based on the information retrieved from the server (parsed from JSON and/or XML objects). The user can then fill out the available fields, just as they would a normal web form. When the user attempts to submit a form, each field is validated according to any constraints provided by the server. Any invalid fields are displayed to the user for correction. Once all fields pass the validation tests, the data is stored to a local database located on the device.

In addition to inputting data, the application also needed to display any locally stored forms back to the user. TalKivi supports this process using both a list view as well as a map view. Selecting a completed form from the list view will allow the user to edit the form. The map view displays the forms as pinpoints corresponding to where the form was filled out (GPS location fields are part of the form for data collection). When viewing data points, the user should have the ability to edit and delete individual points. All the preceding operations, except downloading forms, must remain functional regardless of the devices data connection status in order to facilitate data collection in more remote areas.

It is important to note that a data connection is not needed in order to fill out forms. All completed forms are stored locally on the device regardless of a data connection. When the device achieves a data connection, all forms that have not been synchronized with the server are uploaded. Any local form can be deleted from the device, but once a form is synced with the TalKivi server, there is no way of deleting it from the server itself. Also, due to restrictions with the server, forms that have been uploaded to the server cannot be edited or deleted on the server by the application.

The complete list of functional and nonfunctional requirements are listed below.

<u>I. Functional Requirements:</u>

The specific functional requirements include:

- Auto-connect to a default server
- View the list of forms available for download
    - Form templates are dependent upon the server specifications
- Ability to download forms
- Dynamically generate the view based on the form template
- Enter data into form to be saved locally on the phone
    - Attributes include text fields, combo boxes, GPS data from phone, etc.

- Display all the local entries a user has inputted in a list
  - Allow user to select entry and edit its data
- Display all the local entries a user has inputted on a map
- App needs to be functional without a data connection
  - Save data locally, sync when data connection resumes

II. Nonfunctional Requirements:

The nonfunctional requirements for TalKivi include:

- Code must be open source and hosted on Github
- Application must be implemented using PhoneGap as well as Titanium
  - Compare and contrast the two implementations for future use
- Must work on Android 2.2 and up
- Must work on iOS 5.0 and up
- Application must be submitted to both App Store and Play Store

III. Project Risks:

The TalKivi Mobile Client project risks include:

- App must be approved by the iOS App Store
- Backend server is being developed concurrently by client
- Cross platform development environments can be complicated to setup initially
  - Double standard exists with Windows/Linux and Mac OSX
- Cross platform implementations has inherent design challenges and limitations
  - GUI design, functionality, etc.
  - "Code once, tweak everywhere", not "code once, deploy everywhere"
- Steep learning curve required to perfect Titanium and PhoneGap approaches

System Architecture

TalKivi's system architecture (represented in Figure 1 below) is very similar to many client-server applications currently on the market. The application utilizes both a remote Amazon EC2 server (Amazon Web Services - Elastic Compute Cloud), as well as the phone's local storage. The application stores all completed forms locally to the mobile device's local storage. The Titanium implementation utilizes the application's "properties" to locally persist the data, while PhoneGap's design used the HTML5 local storage to achieve the same effect. If a data connection is present, the locally stored

forms are pushed to the remote Amazon EC2 server. If the user deletes the application, any data stored locally to the phone will be permanently deleted; however, the data stored to the server is permanent and can only be deleted by the server manager.

Figure 1: High-end overview of TalKivi Mobile Client

Technical Design

Two of the more challenging and interesting features of the TalKivi Mobile Client were also the primary features of the application itself - the dynamic generation of fields and the separation of local data from data stored on the server.

The TalKivi Amazon EC2 server allows users to create their own templates, which any user can download. The server contains a list of currently implemented field types that are usable when constructing a personal template (Brian Krzys hopes to continue expanding and adding new fields to the application). In order to incorporate the ability to create personal templates, the TalKivi application needed to dynamically generate fields according to the template downloaded from the server. In order to accomplish this task, each field type was individually parsed and generated as a part of the form.

This process was both challenging and time consuming, as each field required a different look, validation process, and different functionality. For instance, the “Location” field required using the device’s GPS to

acquire the latitude, longitude, and elevation. Not only does TalKivi allow the generation of each of the specified field types, but they can be downloaded on any Android device using 2.2 and up or any iOS device using 5.0 or higher. As previously mentioned, each of these field types were also validated using data from the template. For example, the “Quality” field type needed to be an integer within the range [1-10]. All of this was possible through the individual generation of each field type.

Another interesting feature of the TalKivi Mobile Client is the separation of local storage and server storage. The user has the ability to display a list of their completed forms on their own mobile device. Taking a step further, the user can then upload their data to the TalKivi server. In essence, the application is a collaborative tool in which multiple users can collect data from across the world and upload it to a single server for analysis and storage. This process simplifies the user experience and allows them to separate their own recorded data from the rest of the information uploaded to the server. Beyond the idea of separation, the synchronization of data offers another interesting aspect. The TalKivi Client checks to see if a data connection is present in order to upload the information to the server.

Some of the features of the application’s technical design can be seen in Figure 2 on the next page. Figure 2 shows an initial wireframe of the application depicting the flow that the user sees when first opening and going through the app.

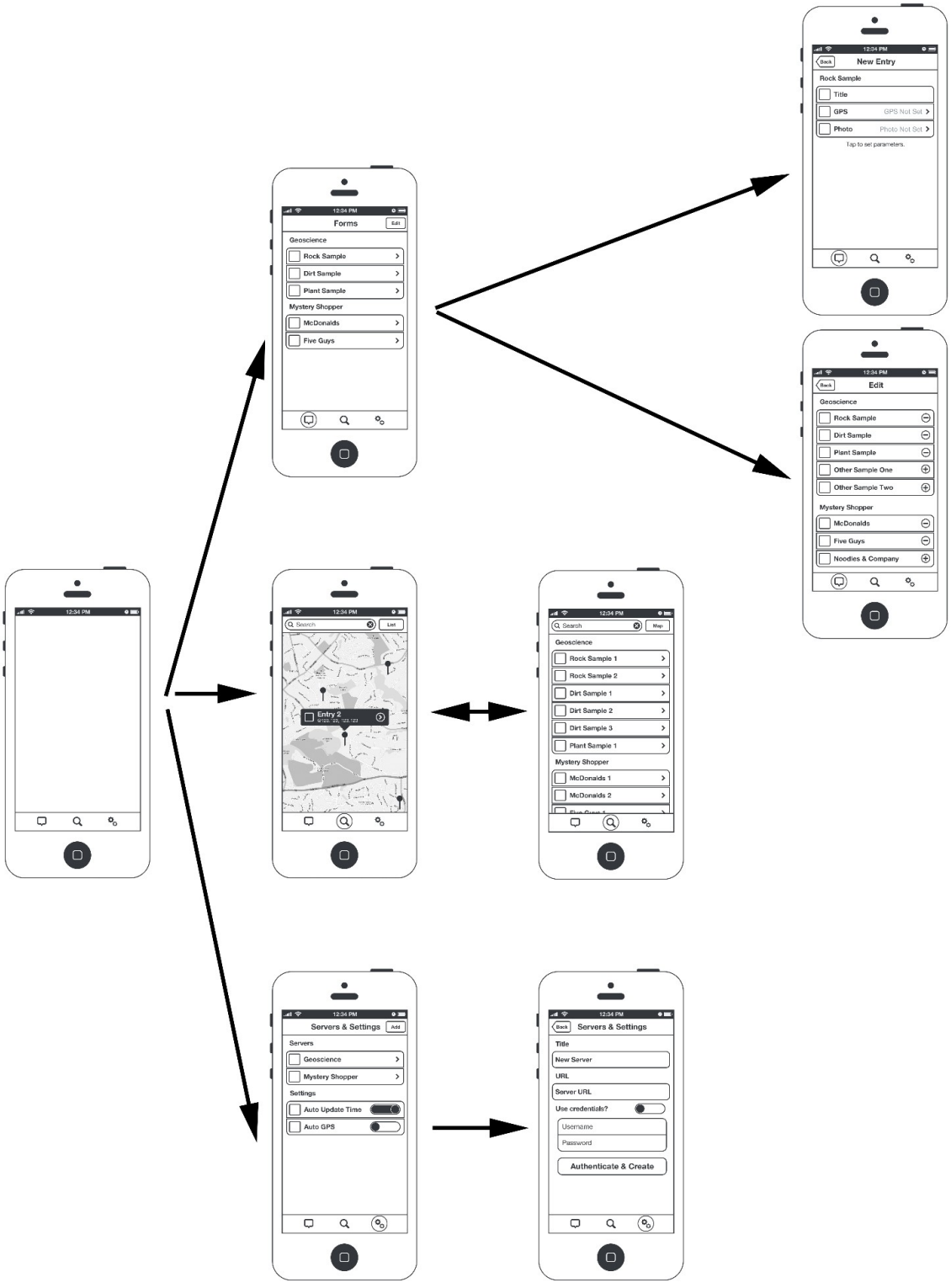


Figure 2 - Initial conception of TalKivi Mobile app.

I. Technical Design Issues:

Throughout the design process of the TalKivi Mobile Client, many design issues were encountered. These included data handling, data entry, and displaying data points on a map.

Since Titanium and PhoneGap utilize different methods of persisting data, both groups were challenged to find the best method of saving local data to the phone. Once the data was saved locally to the phone, the challenge then became managing the synchronization of data with the server. Both groups needed to manage the synced and unsynced data with the remote database (basic CRUD operations), all the while dealing with the possibility that a data connection may not always be present.

Apart from managing the data, another technical design issue was the data entry itself. The application needed to parse JSON or XML objects from the server and convert them into usable forms. These forms needed an intuitive design that match what a first-time user might expect. TalKivi also needed to handle form changes from the server and render them correctly on the mobile devices.

The last technical design issue the TalKivi application faced was displaying the data entries in a map view. The PhoneGap implementation lacked the built-in map object that is included in Titanium, as well as many other common platforms. Therefore, Team PhoneGap resorted to using the Google Maps Web API to display the annotations.

Design and Implementation Decisions

Team Titanium:

Team Titanium chose to handle the JSON format when downloading forms from the server, rather than the XML that was also available. Since Titanium is based in JavaScript, it made a lot of sense to use the JSON format that could be easily parsed in, and converted to send back to the server.

To achieve persistence and local data storage, the Titanium group used Titanium's built in "Properties" feature, rather than implementing their own database system. The Properties in Titanium save between sessions, and also are easily accessible throughout the code structure. The capability of storing objects within the Properties made it an even more fitting choice, and using it saved Team Titanium the time it would have taken to set up a database.

The biggest design decision Team Titanium made was the decision to go with a native visual style, both

on iOS and Android. They felt the style of this application was well represented by native objects, and these native implementations were readily available in the Titanium work environment. They also wanted to take advantage of "making the user feel at home" on both platforms, but especially on Android through the use of the native menu and back buttons. The exploration of Titanium's native support also provided another point of comparison to make between Titanium and PhoneGap.

Team PhoneGap:

Team PhoneGap also chose to use the JSON, rather than XML, for forms from the server. PhoneGap also uses JavaScript, so it was easy to parse JSON objects into a usable format.

Regarding local persistence, Team PhoneGap decided to use HTML5 local storage. Since PhoneGap is built using HTML5, it is an easy process to store all of the form data locally. This storage method allows the application to easily pull necessary information to populate a list, show points on a map, and upload user submitted data to a server.

Unlike Team Titanium, Team PhoneGap decided to use one visual style across both platforms. Both the iOS and the Android version of the application look relatively the same. PhoneGap doesn't possess as much support for native UI design and it was easier for the group to make a sleek interface that was usable on both operating systems.

Lessons Learned

The primary lesson illustrated in the design of the TalKivi Mobile Client was the importance researching a type of framework or library to use. In the development of the PhoneGap application, many decisions were made with little research due to time constraints placed on the project. Deciding which database implementation to use, what mobile styling framework looks best, as well as what form rendering methods to use were all features that changed after working on the project due to the lack of initial research. Team PhoneGap had originally decided to use TaffyDB as a database solution, but after discovering HTML5 local storage, the group implemented a simpler and more elegant solution to the local storage problem. JQuery Mobile was initially agreed upon as the primary method for rendering forms. However, later into development, Team PhoneGap realized the application wasn't using much of the built in functionality of JQuery and decided that a whole UI overhaul was necessary. By doing more research on the frameworks at the beginning of the project, a lot of time could have been saved as well as the pain of refactoring code.

The main design aspect behind the Titanium design was focused on creating native views for both iOS

and Android. Although this created the look and feel each individual to each mobile device, it also entailed a large amount of repeated code to create the displays for each respective mobile platform. The process of creating two unique applications took a significant amount of time to complete and slowed the development of Team Titanium's TalKivi application. Team Titanium also learned that Javascript handles argument passing differently from many programming languages. There were multiple instances in which the group had to rethink the structure of the TalKivi application in order to avoid passing many objects along with functions.

Results

The TalKivi Mobile Client application was designed to dynamically generate forms from a template hosted on a server. After downloading a template, the user is then capable of filling it out and re-uploading it to the server. Although this was the main goal of the application, one of the main project goals was to explore two different cross-platform implementations - Titanium Alloy and PhoneGap.

At the end of this six week development period, the PhoneGap implementation supports the process of downloading forms from a server and dynamically generating them onto the mobile device. The application can then fill out and upload a completed form to the server, when an internet connection is achieved. A list view of completed forms is displayed locally on the device. These completed forms will also show up on a map if there is a connection to the internet and the specific form also includes latitude and longitude points. The application is functional on Android 2.2 and up and iOS 5 and up. Due to time constraints, some of the more advanced field types were not implemented (voice recording, calculated fields, and a strike/dip calculator). However, the application has a solid base for an open source project and can be expanded with relative ease.

The Titanium Alloy implementation accomplished the look and feel of native applications in respect to both iOS and Android. Much like the PhoneGap implementation, many advanced field types were not implemented given the time constraints. However, completed features include: downloading of forms from a server, dynamically generating those forms, the ability to fill out forms, displaying a view of completed forms (using a list or a map), and uploading forms to server when a connection is present. The project can easily be expanded in the future to include other advanced features because it is publicly available through an online repository.

<u>Comparison of Titanium and PhoneGap:</u>

Titanium and PhoneGap, although both cross-platform implementations, are very different. PhoneGap

uses HTML5, Javascript, and CSS3 to develop a roughly identical application for both iOS and Android. Titanium, however, utilizes Javascript, TSS, and XML to develop two different looking applications (one for iOS and one for Android). This allows Titanium apps to be developed using the native layout of iOS and Android; however, this makes Titanium more difficult to pick up and start using. Dependent upon the look/feel of the application a programmer is trying to achieve, he/she might favor one implementation over the other. Below are a list of the major advantages and disadvantages each group has found.

Titanium Pros:

- Easy to get a quick prototype running
- Extensive native support
- Solid online documentation

Titanium Cons:

- Lack of widespread use sometimes leads to unanswered questions
- Native development can be difficult because of inconsistent support between platforms

PhoneGap Pros:

- HTML5, CSS3, and Javascript are easy to work with
- Variety of choices for framework libraries like JQuery mobile
- Good online documentation
- Essentially building a website for a phone application

PhoneGap Cons:

- Lack of styling support for both platforms, must create own icons, tab groups, etc...
- Only able to design simple applications due to limitations of HTML

When choosing one of these platforms to develop on, the decision is going to rely on several factors. First, if it is necessary that the app looks native, Titanium is the clear choice with its support for native objects. If, however, the app is going to be designed with a common design among platforms, either PhoneGap or Titanium are capable. A lot of the decision at that point is going to depend on the developer's current skillset and experience. If he or she is comfortable with web development, they are going to feel right at home with the PhoneGap development environment. Titanium, on the other hand, may be the more familiar choice for developers that have used Ruby on Rails or other MVC (model/view/controller) frameworks. Either way, it should be understood that cross platform development does not simply allow a developer to write code, test it on a device, and expect it to work on all the other devices as well. There will always be device specific issues to address, and it is

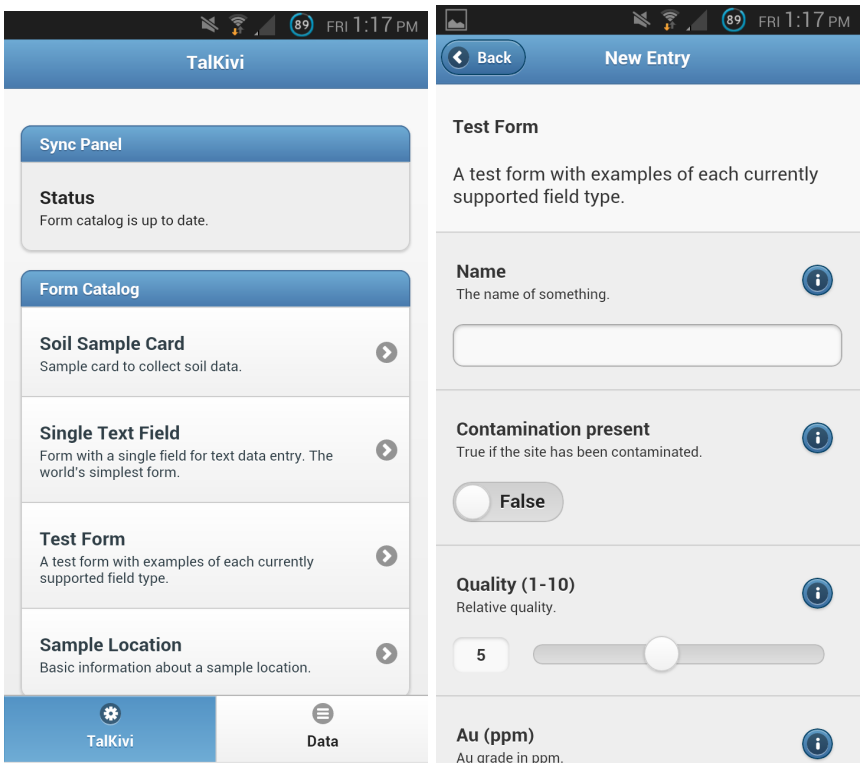
something that just comes with the package of cross-platform implementations.

Future Work

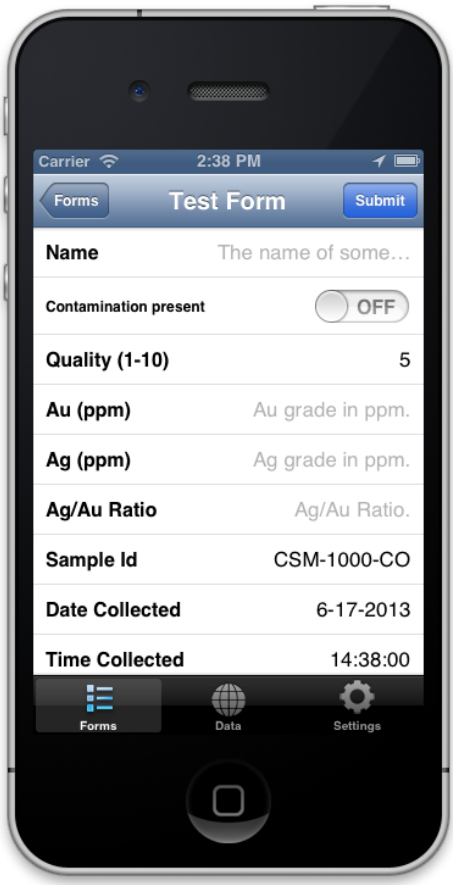
While creating the project, both teams were asked to keep the projects completely open source. Therefore, both versions were designed using the open-close principle, keeping the door open for future additions while not needing to modify much of the existing code base. There is still work to be done in implementing additional field types and adding settings to give the user more control over their individual experience. Both the Titanium and PhoneGap project will be hosted on GitHub to allow anyone to further improve the applications.

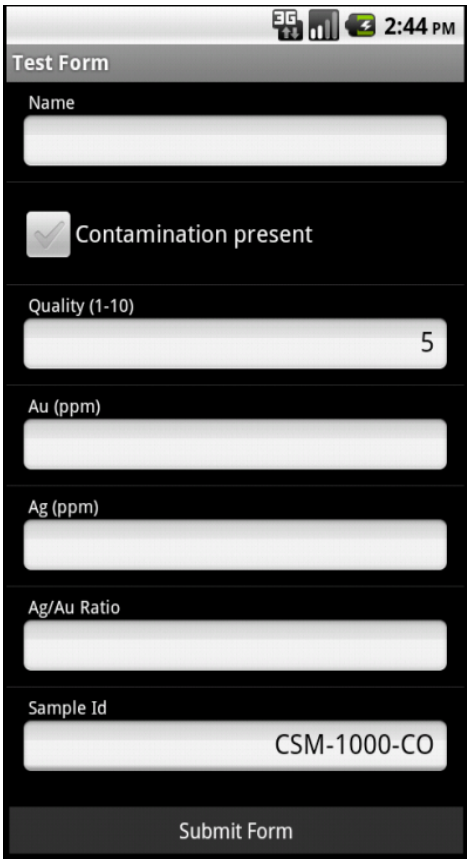
Appendices

Final PhoneGap Application Screenshots:



Final Titanium Application Screenshots:





The source code for both TalKivi implementations can be found using the links below. Both projects are completely open source and welcome the future development of different programmers.

PhoneGap Implementation → **https://github.com/afurze/talkivi-phonegap**
Titanium Implementation → **https://github.com/kkincade/TalKivi**