**Collaborative Participatory Environmental Observation System**

**Chris Loeschorn**

**Montclair State University**

**Department of Computer Science**

**Spring 2013**

**Mentor: Dr. Fails**

Contents

[**1.** **Introduction** 2](#_Toc355218424)

[**2.** **Project Description** 2](#_Toc355218425)

[**3.** **Impact** 3](#_Toc355218426)

[**4.** **Relevant Research** 4](#_Toc355218427)

[**5.** **Implementation** 5](#_Toc355218428)

[**5.1.** **Project Approach and Overview** 5](#_Toc355218429)

[**5.2.** **Database Implementation** 6](#_Toc355218430)

[**5.3.** **Database Tables:** 7](#_Toc355218431)

[**5.4.** **Stored Procedures, Functions, and Queries** 8](#_Toc355218432)

[**5.5.** **PHP Web Service** 13](#_Toc355218433)

[**5.6.** **Android Implementation** 14](#_Toc355218434)

[**5.6.1.** **Login** 14](#_Toc355218436)

[**5.6.2.** **Registration** 14](#_Toc355218437)

[**5.6.3.** **Home Screen** 15](#_Toc355218438)

[**5.6.4.** **Add Tag** 16](#_Toc355218439)

[**5.6.5.** **My Tags** 18](#_Toc355218440)

[**5.6.6.** **Tag View** 18](#_Toc355218441)

[**5.6.7.** **Profile View** 20](#_Toc355218442)

[**5.6.8.** **Edit Profile** 21](#_Toc355218443)

1. **Introduction**

The field of human-computer interfacing is one that is ever-growing, as technology, along with the ways that we use it, is always evolving. Computers are becoming more mobile and integrated into our daily lives. Since we are adapting so quickly and becoming reliant on the use of mobile computers, the technology needs to adapt to this as well.   
 In their beginnings computers were primarily used by scientists in their labs but as time progressed, the technology trickled down into the business world, then into people’s homes with the PC, and now they are ubiquitous. Everywhere we go there are computers…they are essential to our economy, our infrastructure, our vehicles, and with the widespread use of mobile devices, they play a large role in our everyday activities. A new dynamic that is being introduced to the field of developing computers and software is the fact that children are now beginning to commonly have access to technology. Devices and their software are now being developed to cater specifically to children, and this is an interesting area that this project will explore.

1. **Project Description**

This project encompasses multiple disciplines of Computer Science, mainly database design and implementation, as well as interface design. The idea was to create an application that is aimed at children that will allow them to take pictures and create geo-tags with description information to share with other children to create a social learning experience. For example, person could take a picture of a pine tree with a geo-tag of where they found it along with a short description, then other people that log in to the application will be able to log into the application and view other people’s tags, as well as add tags of their own. Eventually there is a web component to the overreaching research project, but that is beyond the scope of this Masters project.  
 While the overall scope of the research project is broad, the focus of my Masters project was to implement the framework of the application. I have worked on the database and mobile interface with a focus on the social aspects of sharing information and the goal of creating an implementation that will be easy to expand upon in the future.

Some major features of the application that I have implemented are the database in its entirety, a php web service to allow interaction with the database, and many of the main functionalities of the Android application.

1. **Impact**

Even though the widespread accessibility of computers to everyone has only really taken off over the past decade or so, the idea of focusing research specifically on children and computing is not new, and the information available is fairly widespread. One area that has been explored is the development of programming languages designed to be used by children. For example, in the 1960s, a programming language called “Logo” was developed with the purpose of education and research involving children. It used a robotic turtle to help teach children the basics of programming languages (What is Logo?). This idea was re-visited in 2005 by Microsoft’s Kodu project which is another programming language that was created with the intent of allowing children to explore the concepts of computer programming in a way that is engaging and also fun for them.

The fact that there are resources dedicated to teaching children about computer programming shows that there is an overall interest in the way that children interact with computers. The 2011 paper, “The Nature of Child Computer Interaction” explores this idea, and even defines a sub-field of HCI called CCI, or Child Computer Interaction (Read, Bekker 2011). This paper discusses the field of CCI as a whole, indicating how it generally differs from HCI and the general uses that children have for computers. In this paper, Read and Bekker state that “More than anything else - children’s computing is discretionary – the computer dissolves into the current activity.” This quote defines a major challenge in the CCI field, which is taking the focus off of the technology that is being used, and keeping the children engaged in the provided activity. I believe that this quote helps to define a goal of this project: to try to design an application that children will view as engaging and fun rather than as a chore or homework assignment. It should allow them to explore their creativity while developing social skills and becoming acquainted with technology at the same time.

1. **Relevant Research**

One major research development in participatory learning for children was the e-Bag project [1]. This infrastructure, officially called “A Ubiquitous Web Infrastructure for Nomadic Learning,” provided a way for students to seamlessly share and access their data from multiple places without having to physically carry books or a computer around. The students were equipped with small Bluetooth devices (mobile phones for the trial period), and different areas of the school (classrooms, labs, smart boards, etc.) contain sensors that can receive the Bluetooth signal from the devices. When a student is within range of one of these terminals, they are automatically authenticated, giving them access to the contents of their “e-bag”. Teachers also have access to a Shared Project folder that is shared among the students, so they can distribute assignments to the group, and group members can share data with each other.

The e-Bag project was implemented using the HyCon framework (“framework and architecture for context-aware mobile hypermedia” [2]). This is a four-layer architecture that accounts for all aspects of the system, from the Storage Layer (Database), all the way up to the Sensor layer (the mobile devices the students carry). The system also includes an “offline mode” in which users can use normal password authentication to access their data in an eBagExplorer application.

While the main focuses of the e-Bag application are security and ease of access (the proximity-based authentication, for example), whereas my project is more focused on the actual data sharing and participation between students, there are some similarities in the concepts. The idea that teachers can eventually assign work to a group and letting them all share their findings and collaborate seamlessly with one another is a central idea behind my project. I believe that being able to collaborate over the web and making this kind of collaboration accessible even to children will be extremely relevant in the future of education.

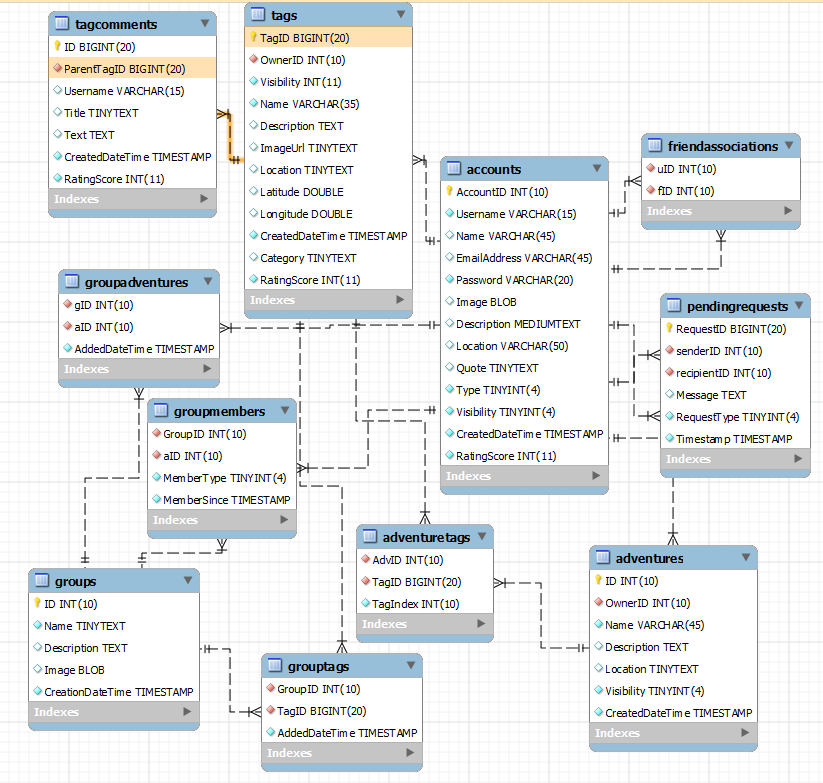
1. **Implementation**
   1. **Project Approach and Overview**

The focus for my implementation was to lay the foundations for this project, with the intent of making the framework flexible enough so that it can be expanded upon in the future. I began my planning with this in mind, and tried to keep this mindset with everything that I created for this project. My planning phase consisted of brainstorming in regards to the feature list and database design, overall specifications, and interface sketches.

First, I implemented the database that stores all of the data for this application, such as the user accounts, friend associations, tag data, etc. This is a MySQL database that is hosted on a Mac server in the HCI lab. This database was built with future extensions to this application in mind so that the schema will allow for more functionality and also be easily extendible for new features.  
 After completing the database, I set up the framework for the web service that allows users to send and retrieve data from the database. This is a PHP web service that runs on the same server as the database. This script takes http post requests and is the interface between the end users and the database. I initially set up the connection information and basic outline for the web service, and as the Android interface evolved, new methods were added to support the new functionality.   
 The largest and most time consuming part of this process was the Android Application development. This application was designed with children in mind, so the goal was to have a look-and-feel that would keep children engaged, but also with the goal of being smooth and robust enough that users of any age could enjoy it. In the initial brainstorming, I had many ideas as to how to extend the application and many different features that would be great to have, but as I continued working on the project it became evident that it would not be possible to implement all of the ideas given the target time frame. I decided to focus on the core functionalities, with a bias toward the social aspects of the application to build a proper framework that can be easily built upon. The future work section of this paper will discuss some of the other ideas and possible features that I thought about during the design process (some of which are already implemented at the database level).

* 1. **Database Implementation**

The backend for the application is a MySql database. This database stores all the user and tag information, as well as relationships such as friend associations and tag comments. The overall schema diagram is shown in Figure 1 below:



**Figure 1. Database Schema**

* 1. **Database Tables:**
     1. **Accounts** The Accounts table is the main table that stores the user account information. The main fields are AccountID(INT) which is the primary key, and Username, which is a unique name that users choose when they sign up for an account. Other fields are descriptive of the account, including User Profile information (about me, ImageURL, Location, etc), as well as some fields that are reserved for future use, such as visibility, type, and rating score. Future ideas for the accounts include setting privacy levels via the visibility field, allowing different account levels (e.g. student/teacher) with the type field, and allowing users to accrue ratings, with the rating score field.
     2. **FriendAssociations**  
         The FriendAssociations table is a simple table that keeps track of which users are “friends” with each other. When users add each other as friends, two entries are added: one with the userId and their friendId, then the reverse is added so the association will also appear in the other user’s friend list.
     3. **Tags** The Tags tablestores individual tag objects. The Primary Key (TagID) is a long value to account for the large amount of tags that can potentially be created if many users are using the application. The rest of the fields in this table store data that is relevant to each tag, such as the descriptive fields, the tag’s owner id (foreign key referencing accounts table), and the gps location data for the tag (if available).
     4. **TagComments** The TagComments table stores the comments that users leave on the tags. The comments are linked back to the tags via the foreign key ParentTagId, which references the TagID field in the Tags table. This table also stores the text of the comment, its timestamp, the Username of the user who added it, an optional Title field for future use, and the comment’s Rating Score.
     5. **PendingRequests** The PendingRequests table was created to allow for a messaging/request system in the future. This table provides the backend to allow users to eventually send messages, group requests, or any other type of requests that may be needed within the application to other users.
     6. **Group Tables** The various group tables have been created toallow for future group functionality within the application. The main table is the Groups table which will store the group’s general information. There are three other tables, GroupMembers, GroupAdventures, and GroupTags which can be used to store references to the tags that have been added to the group, as well as the group’s members.
     7. **Adventure Tables** Similar to the group tables, the Adventure tables (Adventures and AdventureTags) have been implemented for future Adventure functionality. The Adventures will allow users to group multiple tags into an “Adventure” and share this as a whole with the community.
  2. **Stored Procedures, Functions, and Queries**

This section details the MySQL stored procedures, functions, and queries that were used throughout the application. These include the queries that are currently being used in the application, as well as some stored procedures that were created to support future functionalities (such as the groups and adventures).

* + 1. **CheckUsername**

This query is run when a user attempts to register. It checks the database to see if the chosen Username already exists. If it returns any rows, then the user is prompted to choose a different name.

|  |
| --- |
| SELECT Username from Accounts WHERE Username = '$uName'; |

* + 1. **AddUser** The AddUser query adds a new user to the database. This is performed when a user is registering a new account (after the CheckUsername verifies that the Username does not exist).

|  |
| --- |
| Insert into Accounts(Username, Password, Type, Visibility, CreatedDateTime) VALUES('$username', '$pass', 1, 1, now()); |

* + 1. **GetUser**(by Username)  
        This query looks up a user by their Username (which is unique) and returns the user account object.

|  |
| --- |
| SELECT \* FROM Accounts WHERE Username = '$username'; |

* + 1. **GetUser**(by ID)  
        This performs the same action as above, but allows retrieving a user account object by looking up its ID.

|  |
| --- |
| SELECT \* FROM Accounts WHERE AccountID = '$id' |

* + 1. **GetTags** The GetTags query retrieves all tags for a given user ID, along with that user’s Username (for display purposes). The results are ordered by the timestamp on the tags, displaying the most recent ones first.

|  |
| --- |
| Select t.\*, a.Username From Tags t, Accounts a Where t.OwnerID = '$id' and t.OwnerID = a.AccountID Order By t.CreatedDateTime Desc; |

* + 1. **GetNameFromID** At some points in the application, a user’s ID is known but their username is not. This query retrieves the username for a given ID when the whole user account object is not necessary.

|  |
| --- |
| Select a.Username From Accounts a Where a.AccountID = '$id'; |

* + 1. **AddTag** The AddTag function adds a new tag to the database from the given inputs, and returns the ID of the newly inserted tag.

|  |
| --- |
| BEGIN  Insert into Tags(ownerID, Visibility, Name, Description, ImageUrl, Location, Latitude, Longitude, CreatedDateTime, Category)  Values(ownerID, Vis, tagName, Descr, Image, Location, Lat, Lon, now(), cat);  Return LAST\_INSERT\_ID();  END |

* + 1. **RemoveTag** Query to remove a tag from the database.

|  |
| --- |
| Delete from Tags Where TagId = '$tId' |

* + 1. **AddFriend** AddFriend creates a friendassociation between two users, or returns an error code based on different problems that may prevent the adding of the friend.

|  |
| --- |
| **BEGIN**  **-- get user's id from their name**  **Declare searchUserID Int;**  **Declare result Int;**  **Set searchUserID = (Select AccountID from Accounts a where a.Username = searchUserName);**  **-- add friend only if the two users are not already friends**  **IF (searchUserID IS NOT NULL) Then**  **IF (Select Count(\*) from FriendAssociations f where f.uID = curUserID and f.fID = searchUserID) = 0 THEN**  **Insert into FriendAssociations(uID, fID)**  **Values (curUserID, searchUserID);**  **Insert into FriendAssociations(uID, fID)**  **Values (searchUserID, curUserID);**  **Set result = 1; -- return 1 for success**  **Else**  **Set result = 0; -- return 0 if already friends**  **End IF;**  **Else**  **Set result = -1; -- return -1 if user does not exist**  **End IF;**  **Return result;**  **END** |

* + 1. **RemoveFriend**  
        Removes a FriendAssociation from the database, for both users involved.

|  |
| --- |
| Delete from friendassociations where (uID,fID) IN (($uId,$fId),($fId,$uId)) |

* + 1. **GetFriends**

GetFriends returns the friends list for a user with the given id. This returns full user account records.

|  |
| --- |
| Select \* From Accounts Where AccountID In (Select fID From FriendAssociations Where uId = '$userId') |

* + 1. **EditProfile**  
        Updates the User’s UserAccount entry with new values for their profile fields that were changed within the application.

|  |
| --- |
| Update Accounts set Image = '$imgUrl',Description = '$desc', Location = '$location', Quote = '$quote' Where AccountID = $uId; |

* + 1. **AddTagComment**  
        Allows a user to add a comment to a tag.

|  |
| --- |
| Insert into tagcomments(ParentTagID, Username, Text, CreatedDateTime) Values ($tagId, '$uName','$comment', now()) |

* + 1. **GetTagComments** Returns all comments for a given tag

|  |
| --- |
| Select \* From tagcomments Where ParentTagID = $tagId |

* + 1. **DeleteTagComment**

Allows a user to delete a tag comment

|  |
| --- |
| Delete From tagcomments Where ID = $commentId |

* + 1. **AddAdventure**

Creates a new Adventure.

|  |
| --- |
| Insert Into Adventures(OwnerID, Name, Description, Location,  Visibility, CreatedDateTime) Values(oID, Name, des, loc, vis, now()); |

* + 1. **AddTagToAdventure**

Adds an existing tag to an existing adventure. It first checks to make sure this tag does not already exist in this adventure, and also calls a function to get the next index in the adventure (to keep the tags ordered).

|  |
| --- |
| Declare i INT;  IF(TagExists\_InAdv(advID, tagID)) = 0 THEN  SET i = (Adv\_GetNextIndex(advID));  Insert into AdventureTags(AdvID, TagID, TagIndex) VALUES(advID,tagID, i);  END IF; |

* + 1. **TagExists\_InAdv**

Function to check if a tag has already been added to an adventure. Returns 0 if it does not exist, 1 if it does.

|  |
| --- |
| Declare chk INT;  -- if the tag does not exist in the adventure, 0 will be returned  Select Count(\*) into chk from AdventureTags A Where A.AdvID = advID AND A.TagID = tagID;  return chk; |

* + 1. **Adv\_GetNextIndex**

Returns the next available index for a tag to be blaced in an adventure to keep the tags in order.

|  |
| --- |
| **Declare nextIndex Int;**  **Declare count Int;**  **Declare done Boolean;**  **Set count = (Select count(\*) from AdventureTags where AdvID = advID)+1;**  **Set done = FALSE;**  **-- check the index to ensure it is unique, if not increment until a unique index is found**  **While !done DO**  **IF (Select count(\*) from AdventureTags where AdvID = advID and TagIndex = count) > 0**  **THEN**  **set count = count + 1;**  **Else set done = TRUE;**  **End IF;**  **End While;**  **RETURN count;** |

* + 1. **AddGroup**

Creates a new group and adds the group owner to the GroupMembers table

|  |
| --- |
| Insert INTO Groups(Name, Description, CreationDateTime)  Values(n, descr, now());  Insert INTO GroupMembers(GroupID, aID, MemberType, MemberSince)  Values(LAST\_INSERT\_ID(), creatorID, 0, now()); |

* + 1. **AddTagToGroup**

Adds an existing tag to an existing group

|  |
| --- |
| IF(**TagExists\_InGroup**(gID, tID)) = 0 THEN  Insert Into GroupTags(GroupID, TagID, AddedDateTime)  Values(gID, tID, now());  End IF; |

* + 1. **TagExists\_InGroup**

Function returns 0 if a tag does not already exist in a group.

|  |
| --- |
| Declare chk INT;  -- if the tag does not exist in the group, 0 will be returned  Select Count(\*) into chk from GroupTags g Where g.GroupID = gID AND g.TagID = tID;  return chk; |

* 1. **PHP Web Service**

The PHP web service is the connector that allows the android application to send data to and retrieve data from the database. It calls the queries and procedures mentioned in section 5.4, and formats the results as JSON objects (or JSONArrays, depending on the resultset) to return to the Android application. The PHP layer runs on the same server as the database and is made up of four different scripts that interact with each other.   
 The first is DBConfig.php which simply contains the connection information for the database. This is kept in a separate file so it is very easy to modify should the database be moved or if settings are changed. I originally started the development with the webserver hosted on my own machine, and the migration of everything to the server in the HCI lab was very straightforward thanks to this setup.   
 The second piece is the DBConnect.php script. This script performs the actual connection to the database (using the connection info from DBConfig.php) when needed so that the queries can be run.   
 The third script in the PHP layer is the DBProcedures.php script. This script contains the methods that actually query the database and retrieve the results. This uses the PHP “msql\_” API. This API was pretty straightforward to use and it is able to support the types of queries that this application needs. The only drawback is that it cannot return results selected within a stored procedure, whereas the other PHP MYSQL extensions *Mysqli* and *PDO* can. This forced me to use direct queries for the simpler operations and some function callsfor the more complex ones where I would have liked to use stored procedures. After some more research it turns out that this is not really detrimental to the application because MYSQL’s stored procedure cache is on a per-connection basis, so in in application with many different user connections, the stored procedures would need to be re-loaded often anyway and would not see the full benefit of the caching [3]. The query cache on the other hand is shared across user sessions, so its benefits can be utilized even in an environment which may potentially have many simultaneous connections [4].  
 The final script, simply named index.php is the part that connects the Android application to the database functions. This script accepts http POST requests, and based on the ‘operation’ that is posted, redirects to the appropriate function or query. This passes the parameters given via POST to the functions in DBProcedures, which then sends the queries/function calls to the database. When the DBProcedures script returns the data from the database (or an error), the index script encodes the response as a JSON object or JSON array, and returns it to the Android application.



**Figure 2: Index.php and DBProcedures.php interaction**

* 1. **Android Implementation**

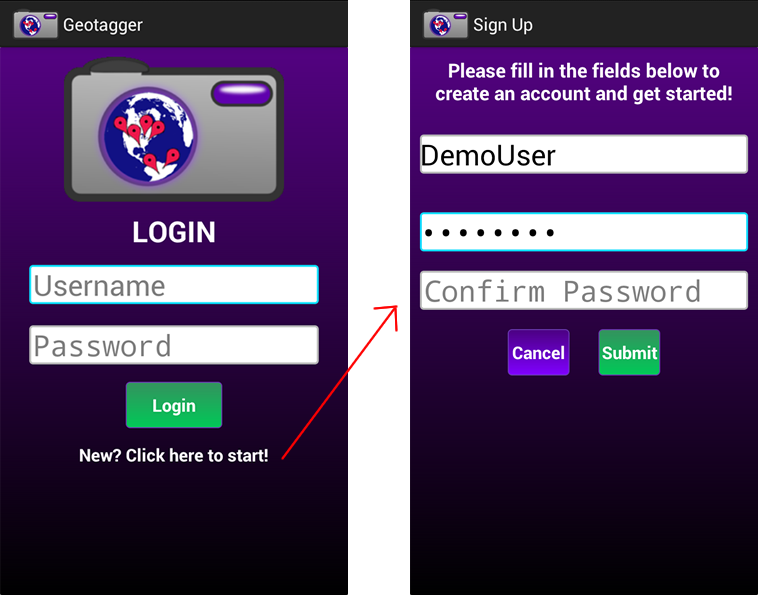
The following section of this document will provide a walkthrough of the android interface for the application as a User would see it, along with some technical details where applicable. Full source code is available at the end of the document.

* + 1. **Login**

Upon opening the application for the first time, users are presented with the login screen. From this screen they can either log in if they already have an account, or follow the link to the registration page to create a new account. Once a user has logged in, if the app is closed for any reason, even force-closed by a task killer or some other application, the device stores shared preferences that are loaded upon the application’s startup which will perform an auto-login if available. This prevents the user from having to log in again every time he or she closes the application.

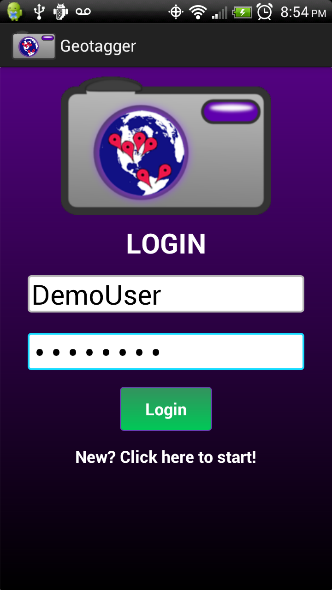
* + 1. **Registration**

Signing up for an account is very simple. All that is needed is for the user to choose a Username and Password, then confirm their password to ensure they typed it correctly. Many applications require an email address to sign up, but I felt that this would slow the children down and many of them may not even have an email address. Although I decided against using it for registration on the initial implementation, there is an email address field in the Accounts table of the database should we decide that this will be useful in the future. Figure 3 below shows the initial login screen and a sample registration.

****

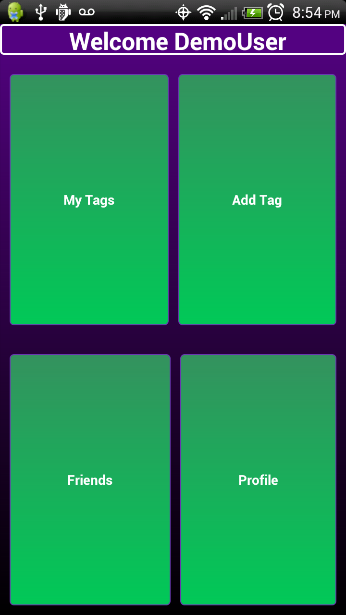
**Figure 3: Login and Registration screens**

The technical requirements for registration are quite simple—the user must choose a username that does not already exist in the database, and their username and password must follow loose guidelines (No spaces/special characters in username, at least 6 letters in password, etc). Once the user successfully completes their registration, they are then returned to the login screen so they can login with their new account (Fig. 4). If a problem occurs during login, such as an incorrect password being entered, the user is prompted with a toast message to try again.

****

**Figure 4: Logging in with the new account**

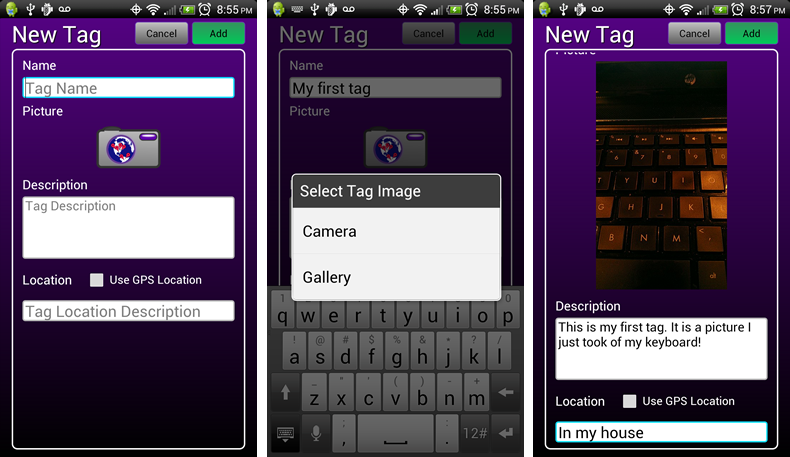
* + 1. **Home Screen** After logging in successfully, the application moves to the Home screen (fig. 5). The home screen is a simple tiled layout consisting of large buttons and text for easy navigation. The four buttons link to the four main features (currently) of the application – “My Tags”, “Add Tag”, “My Friends”, and “My Profile”.



**Figure 5: Tiled Home Screen**

* + 1. **Add Tag**

When a new user creates an account and starts using the application, no initial setup is required—they can simply start adding tags from the beginning. Tags are essentially the heart of the application as they are the objects that allow users to make observations about things that they find in the world. A tag consists of a few fields – Name, Description, and Location (a string description of the tag’s location), and optionally can have an Image to show the object being tagged, and gps location data to share the tags on a map in the future.

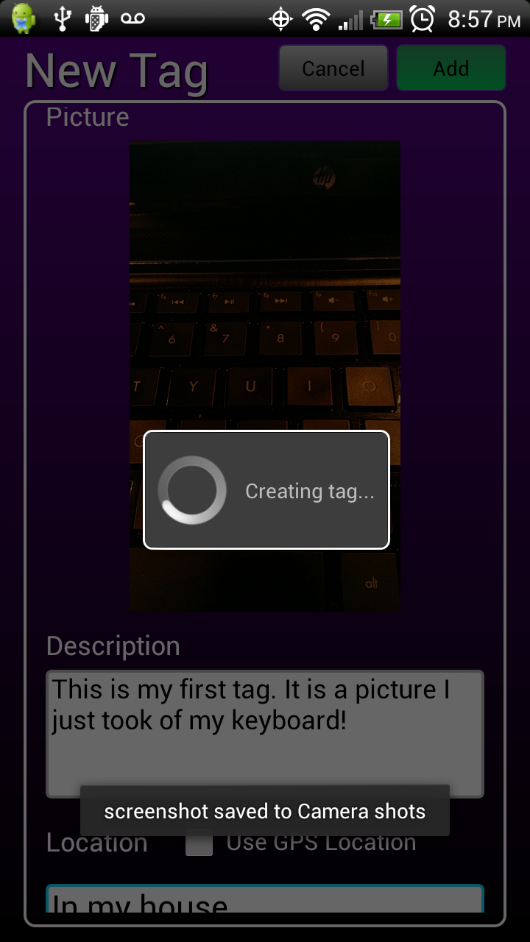


**Figure 6: Adding a new Tag**

The fields are set up as EditText boxes that are large and easy to select. The field that has focus is highlighted in blue (this is consistent throughout the application) to make it easier to determine which one is selected.

Adding an image to the tag (by pressing the camera icon) presents two choices: adding an existing picture from the device’s gallery, or taking a new picture with the camera. Selecting the camera option from the dialog opens the device’s camera, and once the image is taken and accepted, the camera is closed and the image is displayed where the camera icon was previously. If at any point the user decides that they no longer want the image for the tag, they can remove it by long-pressing the image and selecting “Clear”. Images captured within the application while adding tags are also saved to the device’s storage for later viewing (the images are saved to the device even if the tag is not saved).

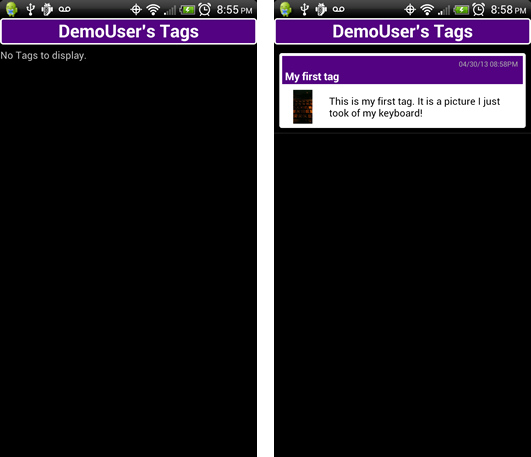
Once all fields are entered and the “Add” button is pressed, a progress dialog is displayed which prevents any further changes while the tag is processed (Fig. 7). First, if the tag has an image, it is uploaded to the file server and an image URL is returned to be stored in the database. Storing the images on the file server rather than in the database directly reduces the load and IO times on the database.

  
**Figure 7: Tag Creation**

After the tag is added, the user is directed back to the home screen so they can choose the next activity they want to perform.

* + 1. **My Tags**

From the home screen, the user can choose to view “My Tags”, which contains a list of all of the tags they have created. The tags in this list are ordered by date/time, with the newest tags displaying on the top of the list. Each tag is displayed with the tag name, the date and time it was created, a thumbnail of the image (if applicable) and part of the description. Initially there are no tags in the list, but once a tag is added (e.g. Fig. 7), the list begins to populate (Fig. 8).



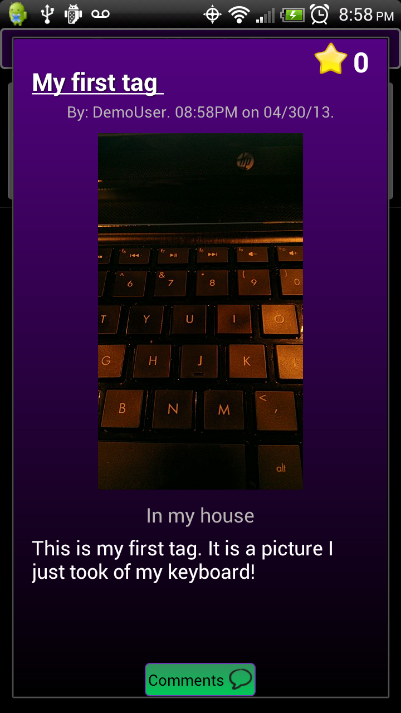
**Figure 8.Tag list before and after adding a tag**

If the user is viewing their own tag list, they have the option to delete tags directly from here by long pressing one of them in the list and selecting “Delete Tag” from the context menu. Deleting the tag from here will permanently remove it from the database. If a tag is deleted from the list, the list automatically refreshes to reflect the deleted tag.

Clicking on a tag item in the list opens the full tag view.

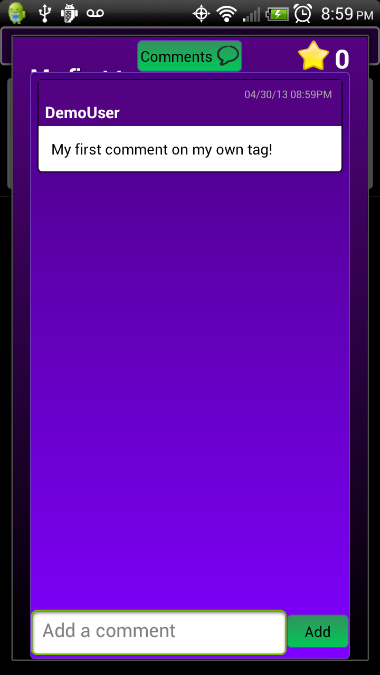
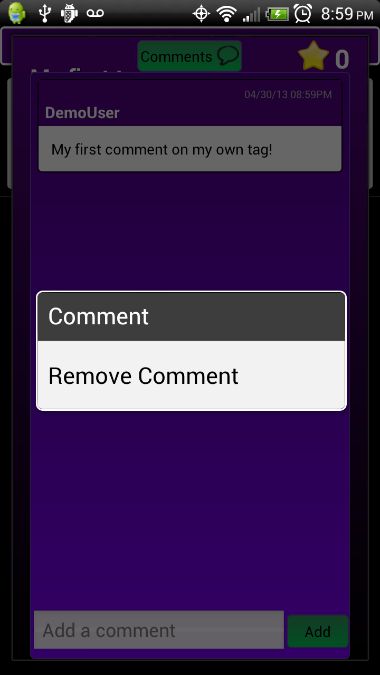
* + 1. **Tag View**

Clicking on a tag item from the tags list opens the tag object in the full Tag View (Fig. 9). The view opens as a full-screen dialog over the tag list. This view allows users to view whole tags—all of the fields are visible along with the image. From the tag view, users can also give Ratings to the tag and add Comments to it as well.



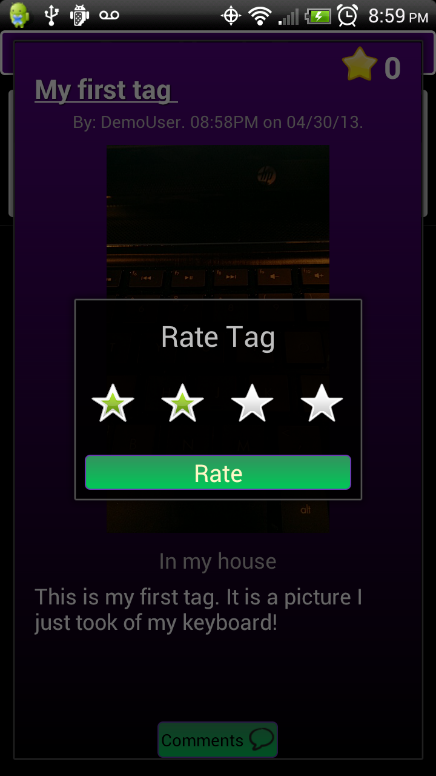
**Figure 9: Tag View**

The green “Comments" button on the bottom of the tag view is the handle for a sliding drawer that holds the comments. Tapping this button or sliding it up the screen opens the comments menu. From this menu, users can add comments to the tag and view previous comments. Each comment is displayed with the name of the user who left the comment, the date/time it was left, and the comment text (Fig. 10). Comments can be deleted via the long press context menu (long pressing on the desired comment to delete. A user can only delete their own comments (if left on other peoples tags), or they can delete any comments (regardless of the user who left them) if they were made on one of the user’s own tags (Fig 11).

**Fig 10. Adding a new comment Fig 11. Removing a comment**

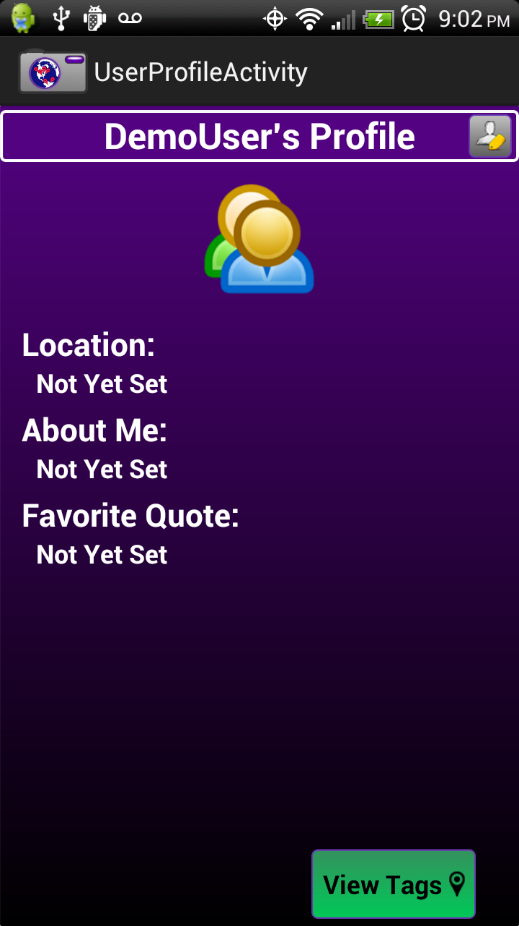
In addition to leaving comments, users can add ratings to tags. This is done by pressing the Star icon in the top right corner of the tag view. The number next to the star indicates the tag’s rating score. Tapping the star icon opens the rating dialog (Figure 11). The ratings are currently done on a four-star scale. 1 star is a poor rating—it removes 1 point from the tag’s total rating. Two stars is neutral—it has no effect. Three stars is a good rating and it adds 1 to the score, and four stars is the best, adding two to the score. This rating implementation may change in the future, based on user feedback that was received.



**Fig 11. Tag rating dialog**

* + 1. **Profile View**

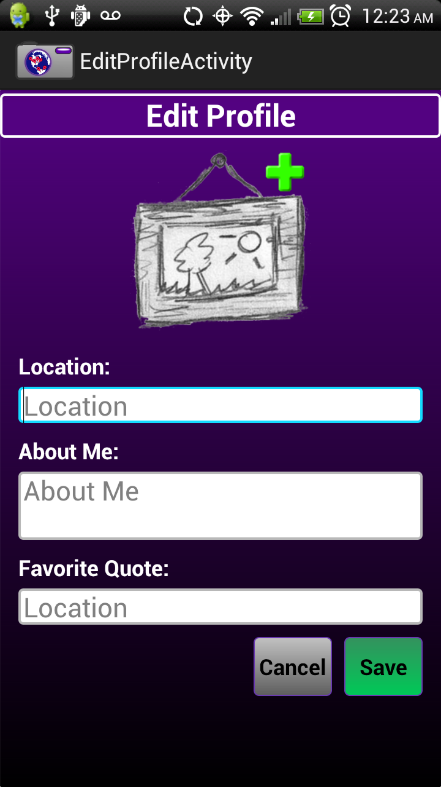
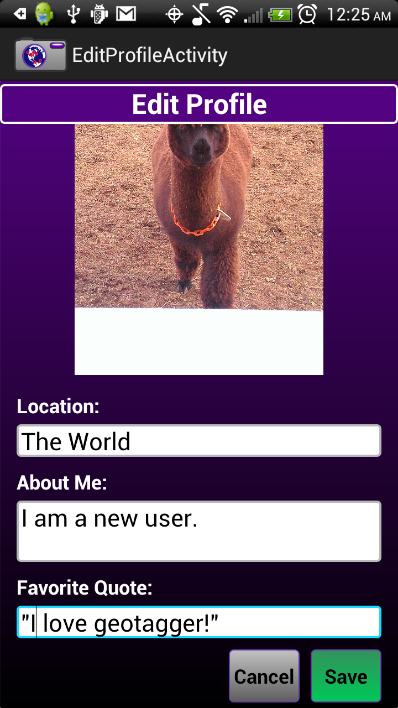
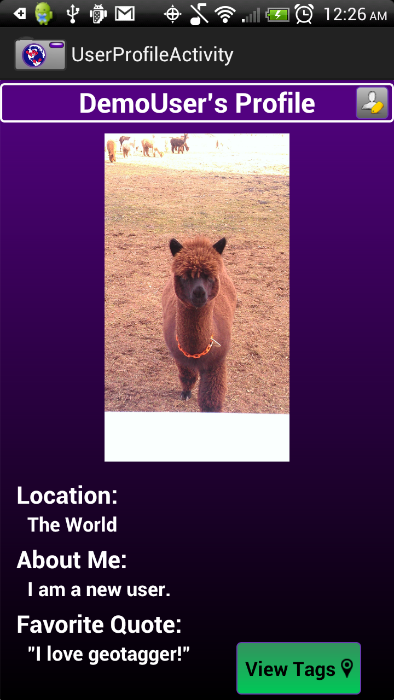
The third option available from the home screen is the profile view. The user’s profile displays a few basic fields to describe the user. There is a profile picture, location, about me, and favorite quote. When a user first creates an account, these fields are not yet filled out (Fig. 12). If the user is viewing his or her own profile, a button appears in the top right corner of the profile which allows them to edit their information. The bottom right hand corner of the profile displays a button which links to the user’s tag list.



**Figure 12: Initial Profile View**

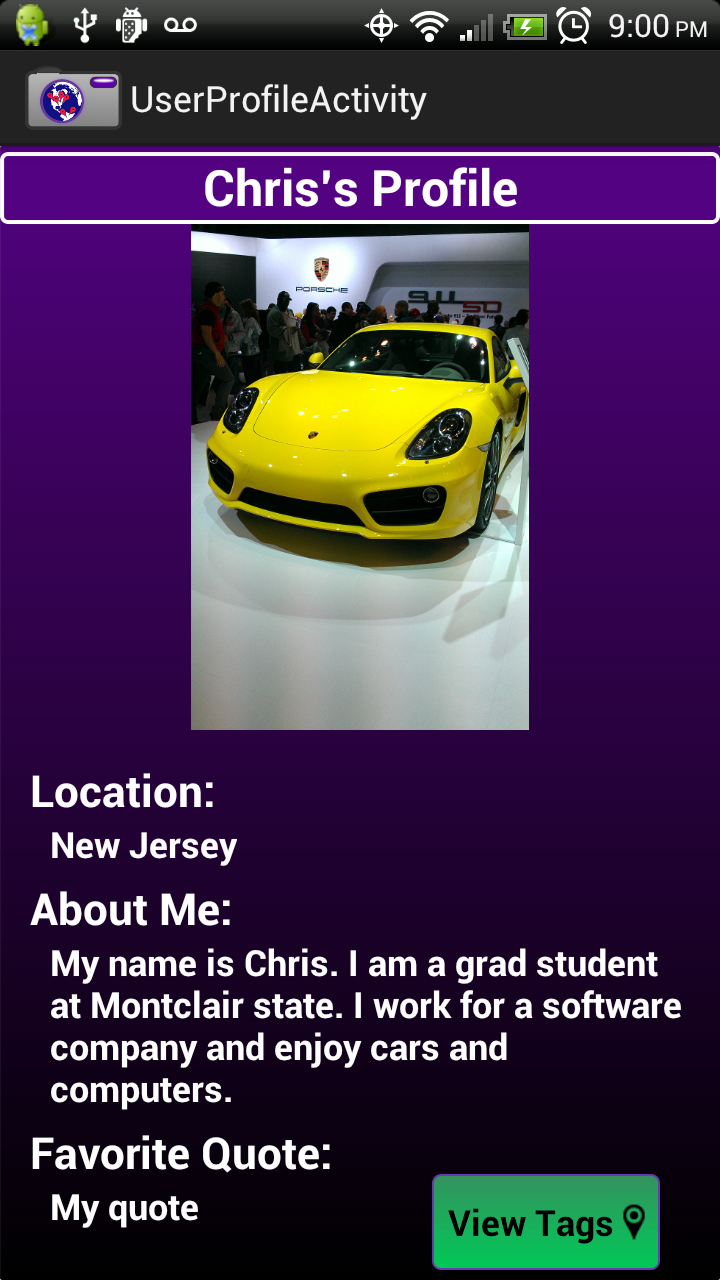
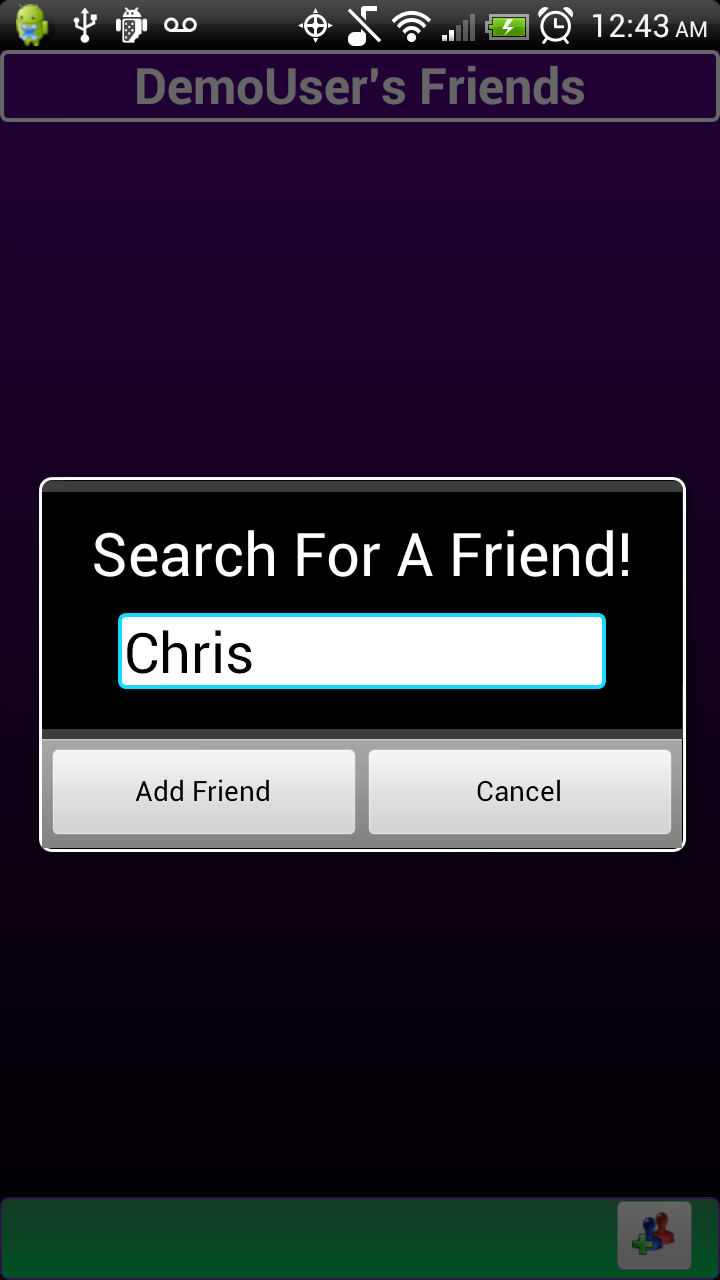
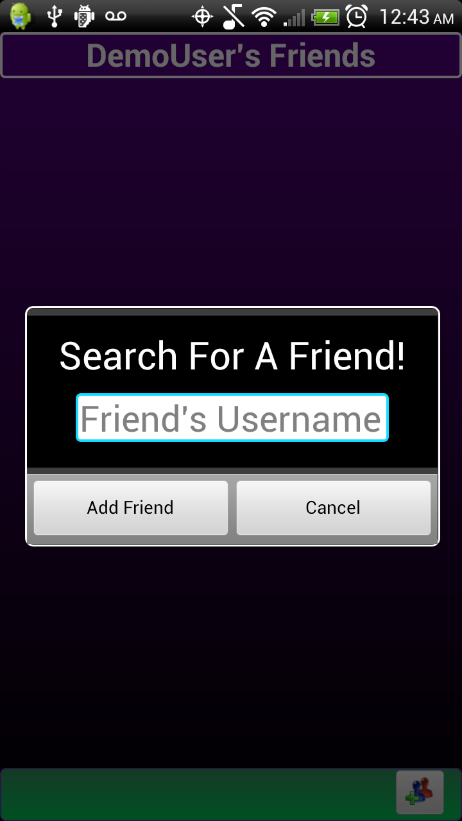
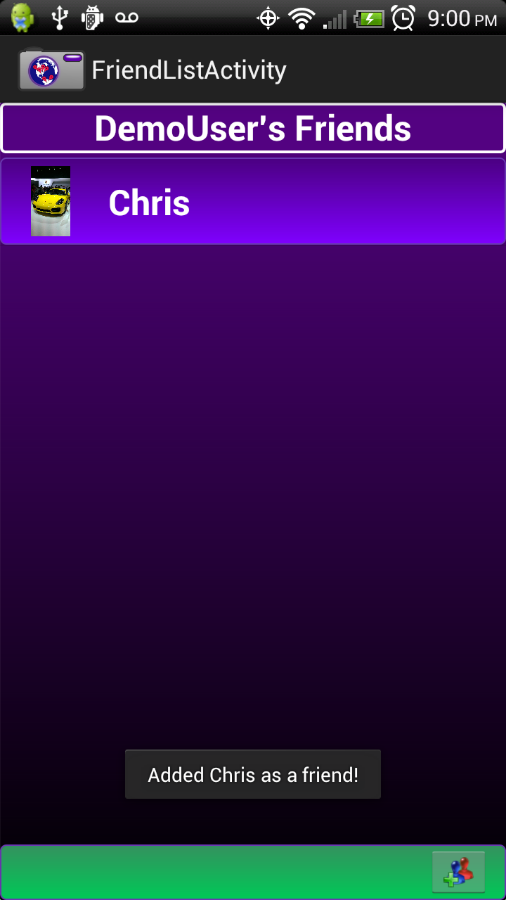
* + 1. **Edit Profile**

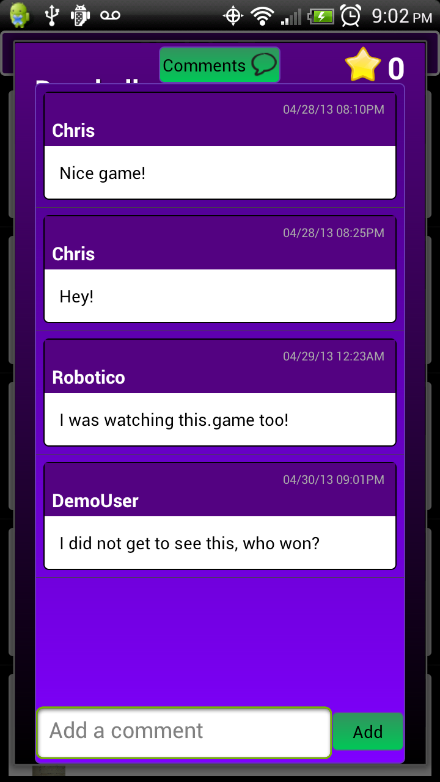
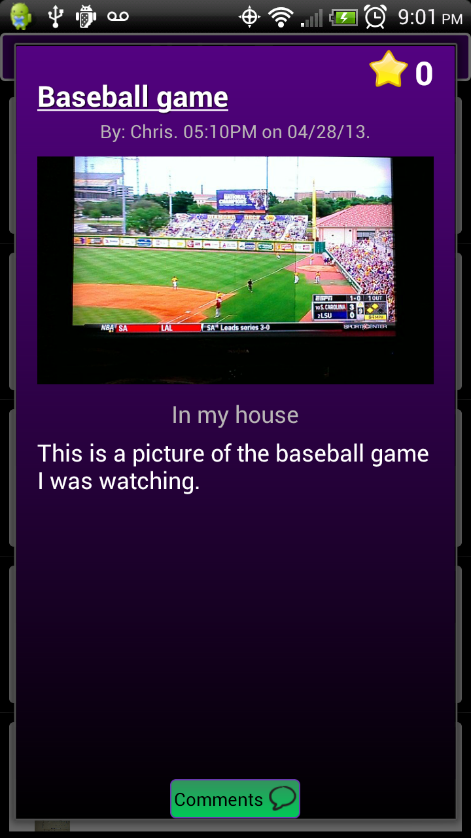
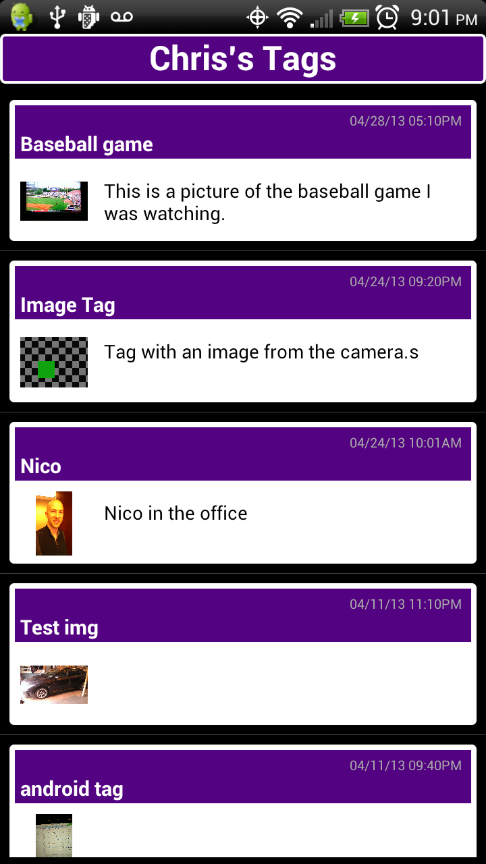
If the user is viewing their own profile, they have the option to edit their information by pressing the edit button in the top right corner. This will open the edit profile screen and all fields will become fillable (Fig 13). The user also has the option to add an image. Adding a profile image works the same way as adding a tag image—pressing the icon will open a dialog giving the choice of opening the camera or gallery. The chosen (or captured) image is uploaded to the file server when the changes are saved. After saving, the user’s profile is opened again, and the changes are immediately visible.

**Fig 13. Edit Profile and changes saved**

* + 1. ****





**Future Work and Add-Ons**

This project implementation is highly extensible and has many opportunities for advancement. There are some features that I thought about during the design processes that were ultimately excluded due to time constraints. One main concept that I believe can be very useful in the application is the idea of Groups and Group Tags. These could be useful in a classroom or research environment. For example, a teacher could create a group and assign a task to the users within that group. Those users could then go out and capture their tags, and share them back with the group. Another concept, called the “Adventure”, would allow a user to string together multiple tags in a logical order and allow users to share multiple tags (that create an adventure) with other users, rather than just sharing single disjointed tags. The group, group tag, and adventure functionality are already implemented in the database (as discussed in sections 5.2-5.4), and just need a UI and web service methods to be implemented.

A Map View (which was discussed in a previous section) is another future enhancement that could provide a new depth to the sharing that is possible, so long as the caveats that go along with it are worked out. The framework for this is in place because currently, tags can optionally be stored with GPS data, which would allow them to be shown in some kind of map view in the future.

This framework also provides the opportunity for migration to other platforms. An iOS application could be created to mimic the Android side for Apple users, and also a Web Interface could be useful for more detailed viewing and management of objects.

**Security**

One area of social applications that is always a matter of concern is security and privacy. This concern is especially prevalent when dealing with children. These aspects of security are not something that was within the scope of my project. My project implementation makes the assumption that the application is being used in a controlled research environment, so the extent of the security implementation is password authentication for accounts.

**References**

[1] C. Brodersen, B. Christensen, K. Grønbæk, C. Dindler, B. Sundararajah. *eBag - A Ubiquitous Web Infrastructure for Nomadic Learning. '05 Proceedings of the 14th international conference on World Wide Web. 298-306, 2005.*

[2]*N. O. Bouvin, B. G. Christensen, K. Grønbæk, and F. A. Hansen. HyCon: A framework for context-aware*

*mobile hypermedia. The New Review of Hypermediaand Multimedia, 9:59–88, 2003.*

[3] The MYSQL Stored Program Caches <http://dev.mysql.com/doc/internals/en/sp-cache.html>

[4] The MYSQL Query Cache <http://dev.mysql.com/doc/refman/5.1/en/query-cache.html>

*Kodu.* Microsoft Research Labs

<http://research.microsoft.com/en-us/projects/kodu/>

Read J., Bekker M. *The Nature of Child Computer Interaction*

|  |
| --- |
| BCS-HCI '11 Proceedings of the 25th BCS Conference on Human-Computer Interaction |
| Pages 163-170 |

*What is logo?* Logo Foundation, 2011. Retrieved from http://el.media.mit.edu/logo-foundation/logo/index.html