Triviarea

CS428 Spring 2011

henke2, nramesh2, rudwick2, rsulliv7
aychan3, smith137, schwab2

I. Introduction....4
A. Project Overview....4
II. Process....4
A. Process Overview....4
III. Requirements and Specifications....5
A. Requirements Overview....5
B. Milestone 1....5
C. Milestone 2....5
D. Milestone 3....6
E. Milestone 4....6
F. Milestone 5....6
IV. Architecture & Design....7
A. Top-Down Description....7
1. Play Game....7
2. View Leaderboard....7
3. Register New User....8
4. Log-In Returning User....9
5. UML Class Diagram....10
B. PHP Server....11
1. Overview....11
2. Play Game....12
3. View Leaderboard....13
4. Register New User....13
5. Log-In Returning User....13
6. Password Functions....13
7. Friends Functions....14
C. iOS....14
1. Overview....14
i. Model....14
ii. View....15
iii. Controller....15
2. Play Game....15
3. View Leaderboard....17
4. Register New User....17
5. Log-In Returning User....18
D. Android....19
1. Overview....19
2. Play Game....21
3. View Leaderboard....22

4. Register New User....23
5. Log-In Returning User....24
V. User Interface Description and Implementation....25
A. iOS....25
B. Android....26
VI. Future Plans....29
A. Future plans....29
B. Personal Reflections....29
VII. Appendix....31
A. Installation Instructions....31
1. PHP....31
2. Android....31
3. iOS....32
B. How to Use....32
1. Main Menu....32
2. Home Page....33
3. Select Category and Leaderboards....34
4. Playing The Game....35
5. Friends List....36
6. Password Resetting....37
C. Testing....38
1. PHP Unit Tests....38
2. iOS Unit Tests....38
3. Android Unit Tests....38
4. Manual Interface Testing....39

I. Introduction

Project Overview

Triviarea is a multi-platform trivia game for mobile handsets. It uses a back-end server to provide users with matches, questions, and scoring while utilizing the phone UI control and information display. The two supported handsets will be the iPhone and Android handsets, while the server will be handled by PHP with a back-end mySQL database. Points will be accumulated based on the "swiftness" of answers, and as time progresses various hints may be given to users still waiting to reply.

Triviarea also has a social aspect to it in which a user can compete against their friends. This is accomplished through the leaderboards feature. A user has the ability to add friends to their friends lists and then view their scores for each category for that day. One can also view a global leaderboard of the top players in the world to see how they match up against the best of the best in a particular category. All leaderboards are reset every day to both entice players to come back and play, and to give the average user a change to get their name on the high score list instead of permanently seeing an unobtainable score at the top of the charts.

II. Process

Design Process Overview

In designing Triviarea, we divided development into 3 distinct and separate components: our backend PHP server and database, an Android application developed in Java, and an iPhone application developed in Objective-C. We immediately divided our group up into 3 sub-groups who would focus solely on one component.

In order to facilitate group cohesion, we all met as a large group every week for a few hours for planning and designing how our components would interact. This typically involved designing our PHP Server's API, specifically how an application would make a request to the server and what type of data would be returned. No coding was done at these meetings.

All of our development was done during meetings of our 3 sub-groups. These meetings took place when convenient for each group. Details on each group can be found at the following pages on our group Wiki.

- Android Team Page
- iOS Team Page
- PHP Server Team Page

In each of our sub-group's meetings, we used a modified form of XP (Extreme Programming) as our software development process. This involved short development cycles and frequent releases, but diverged from test driven development and partner programming. The applications focused more on interfacing with the server, so user interface and controlling views became the central focus as the server implemented more features. This form is hard to test and was easier to streamline with parallel programming between subgroup partners. Because we were developing our Server's API at the same time that we were developing our Android and iPhone applications, this agile development process suited us well.

III. Requirements and Specifications

Requirements Overview

The Triviarea project had multiple goals from the onset. The basic game would require a server capable of talking to the applications and delivering them as sets of questions pulled from our database. The game flow would be a basic trivia game awarding points based on the swiftness of answers. After developing the game flow, we added aspects to make the game more social and enjoyable with friends. To accomplish this, the database needed to hold user accounts, players' scores, friends, and daily leaderboards. The applications had to be able to request and display the returned data as well. A final part of polishing the project would include password changing and resetting, similar user interfaces (between Android and iOS), and the ability to add and remove friends from a user account.

Milestone 1

The first milestone was used primarily as a period for initializing the project. The team was segmented to fill the Android, iOS, and server roles. These teams needed to decide on, and familiarize themselves with, the testing frameworks for their applications. This process took the majority of the milestone. We also defined the necessary databases the project would need based on the decided game play. A PHP page was created to test connection to the database and to provide team members with the ability to add questions to the database.

- PHP - SimpleTest, Android - JUnit, iOS - Google Toolbox testing frameworks
- Database and interacting PHP for adding questions - newGame.php

Milestone 2

The server team created an interface for requesting question sets from the database and delivering them to the applications. Initially this was done by just displaying a JSON string on a php page and parsing the string off of the page from within the application. The next step was to create dummy applications on the iOS and Android. Besides developing a game loop and interface, a large part of this milestone was learning how to communicate back and forth to the server from the applications and testing.

- Dummy applications for iOS and Android
- POST requests - Server communication

Milestone 3

The game was updated to match the planned rules. A game would be 15 questions long. Each question would consist of a second to display the question, 10 seconds to answer, and 4 seconds after the question to display the correct answer. In order to make question retrieval more secure we would need user accounts. This milestone was also used to add a registration functionality to the server and to alter the PHP such that it would send a full 15 questions. In addition to the above requirements the team also began submitting a multitude of questions to the database.

- Registration - register.php (unregister.php for testing purposes)
- Retrieve Questions - retrieveQuestionSet.php

Milestone 4

Applications were expanded to allow user registration and login through the devices. Scores were committed to the server at the end of a successful game, and returned nothing if a game was never finished. The database was expanded to include a total of six categories and security ideas were decided on to protect the questions in transit. The server was also able to return leaderboard scores.

- Login - appLogin.php
- Registration - register.php
- Commit Score - sendScore.php
- Security through randomized question sets and limited to a single category request a day.

Milestone 5

Applications needed to display the requested leaderboards based on the category. The two potential leaderboards for each category were daily standings for all global players or from the user's friends list. Application UI was polished to match. Reset and change password pages were implemented as well as the respective server functionality. Friends lists were implemented both on the server and application.

- Leaderboard - retrieveLeaderboard.php
- Friends - friends.php
- Password - changePassword.php

IV. Architecture & Design

<u>Top-Down Description and UML Diagrams</u>

Play Game

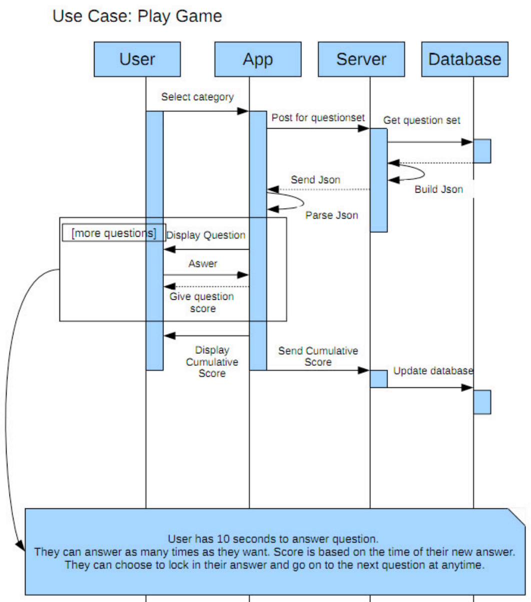


Figure 4.1: Play Game Sequence Diagram

We consider the Use Case of Play Game (Figure 4.1) to be the most critical and central Use Case of the entire Triviarea system. It is the core feature of the applications where each user will most likely spend the majority of their time.

Both Android and iOS versions work identically, so the Use Case applies to both. Through the application, the user first selects the category for the type of questions they wish to receive.

The application then places a request for a set of questions to the server. The server queries the database for 15 questions based on the category provided, builds the questions and answers into a JSON object, and returns the JSON to the application. It is the applications job to then parse the JSON into question objects.

The application will then proceed displaying questions and possible answers to the user. When the user is displayed a question, they have 10 seconds to answer the question correctly. When they give an answer, a score is given to them. This is the score they will receive if they have answered the question correctly at the end of the 10 seconds. After the 10 seconds, they will be informed if they have answered the question correctly, and if not, what the correct answer was. This process will repeat for all 15 questions that the server has provided to us.

At the end of the round, the application will send to the server the score that the player received, along with the player's username and the category they just finished playing. The server will then insert that player's score into the database.

View Leaderboard

The Leaderboard Use Case (Figure 4.2) is the core social aspect of Triviarea. We believe that including a competitive aspect, where you can compete against friends to see who can get the highest score, will draw in more users who will return more frequently.

First, when the user selects to view the leaderboards, the application will provide them some choices for which category they wish to view leaderboards for. The user will select one category they wish to view. The application will then make a request to the server containing who made the request, and for what category. At this point, there will be 2 JSON objects that the server will create. The first will be a global leaderboard of the top 5 players for that particular category. The second will be a friends leaderboard containing all of our friends and their high score is every category. These JSON objects are then returned to the application.

It is then the applications job to parse each leaderboard object and display it in a suitable format. When the user selects to view global leaderboards, we show the 5 top players that the server has informed us about, listed in descending order by score. When the user selects to view their friends leaderboard, the application filters through the friends JSON object and displays a list of their friends and their high score for that category, listed in descending order by score.

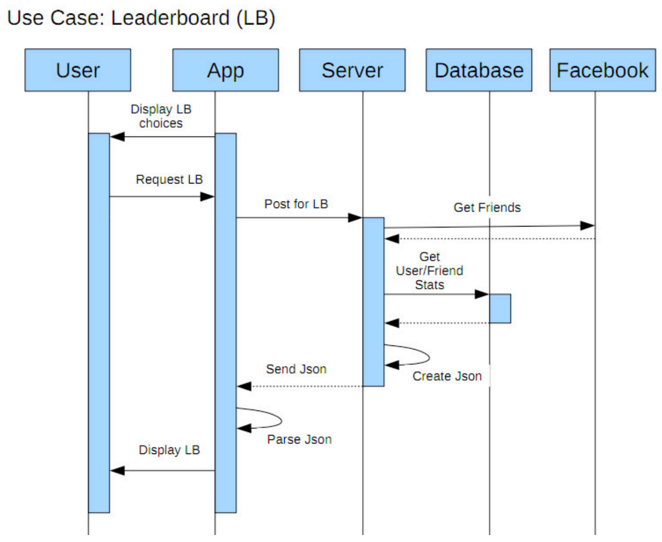


Figure 4.2: View Leaderboard Sequence Diagram

Register New User

In order to keep track of high scores and friends lists, we require every user to register with a unique user ID. This is done through our Register Use Case.

First the user navigates through the application to the page where they can register. This page contains 3 text boxes: one for a username, a password, and an email address. The user will then enter their information and the username they would like to be known by.

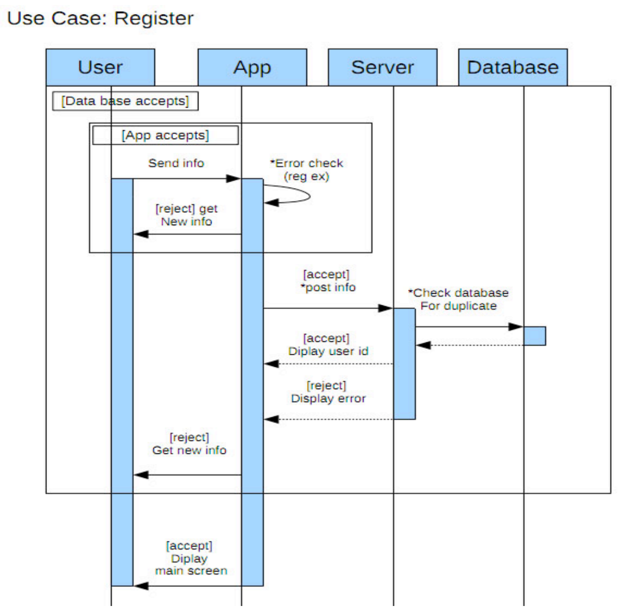


Figure 4.3: Register User Sequence Diagram

It is the application's job to make sure the information the user entered could be accepted by the server. This includes doing basic checks on the username and password fields, and running a regular expression check on the email address to make sure it conforms to a basic email standard format. If any of the fields contain invalid information, the user is informed and given another opportunity to enter their data.

If the application accepts, it makes a request to the server to register this new user by sending the user name, password, and email address. The server queries the database to see if the given user name or email address has already been used. If so, the server responds to the application informing them that their information is already in use, and the application allows the user to try again with new information. If the server accepts the registration request, they respond to the application with the new user's user ID. The user in informed by the application they they have registered successfully.

Log-In Returning User

Before a user can play a game or view leaderboards, we require them to log in. Logging in requires that the user has already registered. This is done on the home screen of the application. Two text boxes are provided for the user: one for a username and another for a password.

After the user inputs their credentials, the application should check that possibly valid information has actually been provided. If the information passes the check, the application sends the credentials to the server to check if this is a registered user. If the server cannot find the user, or if the password is incorrect, the server responds to the application that the credentials are invalid. At this point, the user is given another opportunity to provide valid credentials.

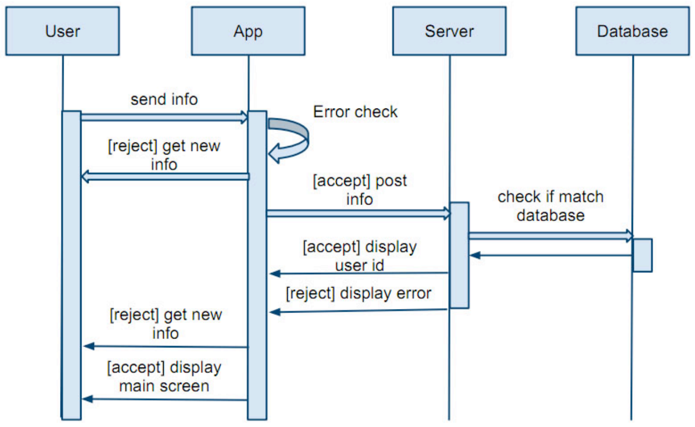


Figure 4.4: Log-In Sequence Diagram

If the user name is found in the database, and the password matches, then the server responds to the application with that user's user ID. The application then informs the user that they have successfully logged in by displaying a new menu. This new menu contains options for playing a game, viewing leaderboards, and other features that would require a user to be logged in.

UML Class Diagram

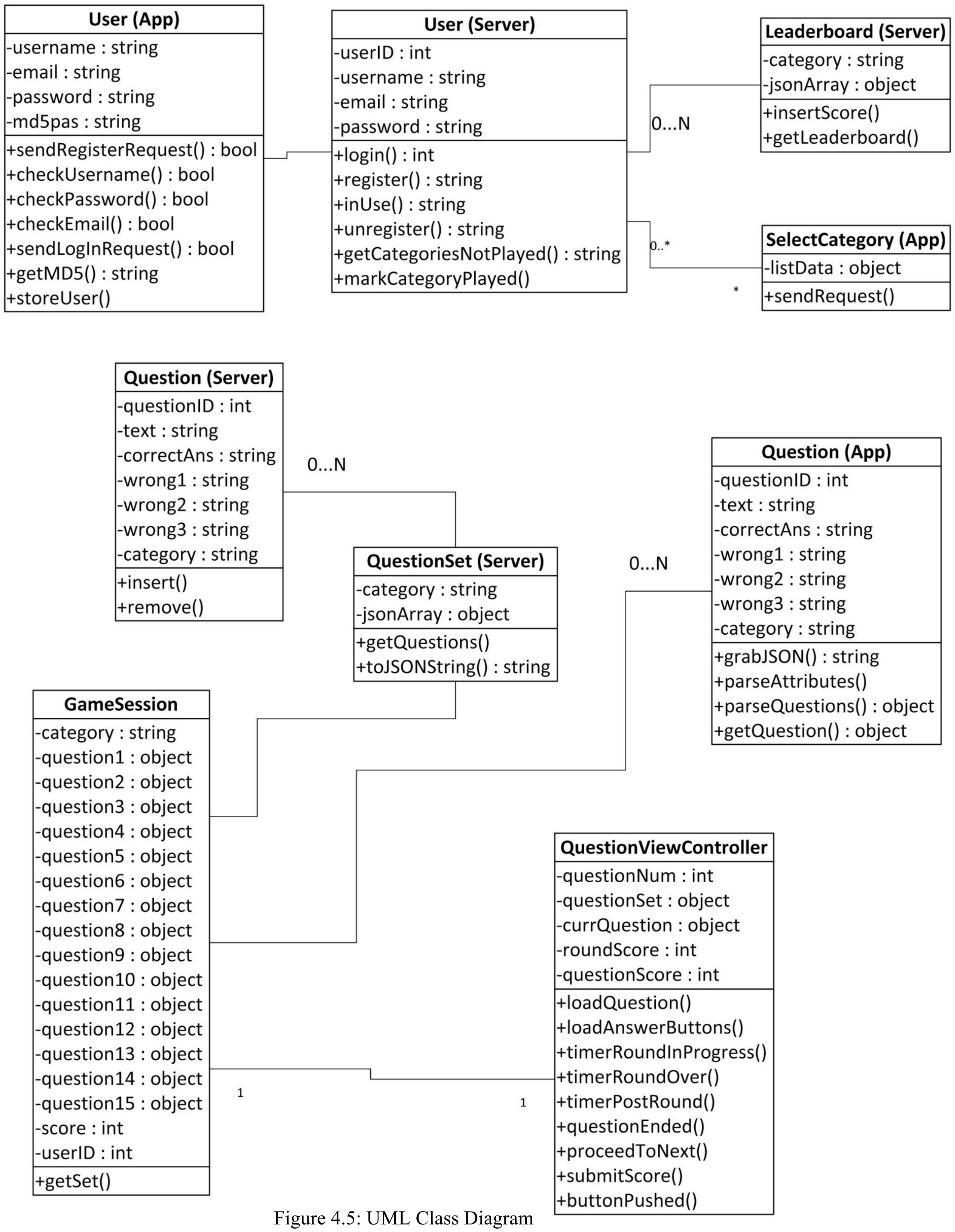


Figure 4.5: UML Class Diagram

PHP Server

Overview

As described above and shown in the UML diagrams, the server has a few main functions: storing information, processing requests for the applications, and creating an interface through which the applications can communicate with the server. The database structure is described as follows:

- users
  - userID
  - username
  - password - stored as a hashed value (md5)
  - email
  - admin - admin flag; 1 if user is admin, 0 if user is not
  - session_id - used for logging into website (not used for application)
- questions
  - questionID
  - question
  - correctAns - correct answer for question
  - wrong1 - a wrong answer to be displayed as a choice
  - wrong2 - a wrong answer to be displayed as a choice
  - wrong3 - a wrong answer to be displayed as a choice
  - category
- scores
  - userID
  - one field for each category - (e.g. userID's score for the category 'Music' would be stored in the 'Music' field)
- setLeaderboards
  - category
  - user1 through user5 - keeps userID of the given place (e.g. first place's userID is stored in user1)
  - score1 through score5 - keeps the score of the given place (e.g. first place's score is stored in score1)
- friends
  - userID
  - friendID - id of userID's friend (friendID does not necessarily have to have userID as a friend)

PHP pages were created as the interface through which the applications communicate with the server. Each page takes a POST request and with the proper information, is able to process each request and give the proper response to the applications. Looking at each of the user stories, we can see how these pages are designed and what functions they call.

There are three classes in the server-side system: `User`, `QuestionSet`, and `Leaderboard`. Each handles all the functions related to their object. For example, let's look at the `appLogin.php` page. The applications post to this page when they want to login a user.

```
        $username = trim($_POST['username']);
        $password = trim($_POST['password']);
        $u = new User(0, $username, $password, '');
        echo $u->login();
```

As seen, this calls the `login()` function in the `User` class.

```
        function login() {
                        $query = 'select * from users where username =
"'.$this->username.'" AND password = "'.$this->password.'" LIMIT 1';
                        $returned = mysql_query($query);
                if(mysql_num_rows($returned) == 1) {
                    $u = mysql_fetch_assoc($returned);
                    return $u['userID'];
                        }
                else
                                return -1;
            }
```

Most of the pages and functions follow this convention. The pages take some information from the applications, call the proper functions to process them, and then display the response on the page. In our above example, the applications know that a -1 displayed on the page means the login attempt was unsuccessful while a positive number means the login attempt was successful and that number is the user's userID.

Play Game

First the applications post requests to the `retrieveCatNotPlayed.php` page in order to get a list of categories that are available to the user to play (since users can only play each category once per day). The only required information is the userID and the page calls the `getCategoriesNotPlayed()` function in the `User` class. A string of the categories that the user has not played yet is then displayed on the page with the categories separated by commas.

Then, once the user has chosen a category, the application makes another request to `retrieveQuestionSet.php` with the userID and category as the information. The page calls the `markCategoryPlayed()` function in the `User` class, which marks the user as having played the given category and then calls the `toJsonString()` on the `QuestionSet` object, which creates a new set of 15 random questions in the given category and displays the set as a JSON string.

Once the user is finished playing the game, the application posts the userID, category, and the user's score to the page `sendScore.php`, which calls the functions `insertScore()` in the `Leaderboard` class and `recordScore()` in the `User` class. The `insertScore()` function checks if the user has scored high enough to place on the leaderboard for that category. The `recordScore()` function records the user's score in the scores table.

View Leaderboard

The application posts a request containing the category of the desired leaderboard to `retrieveLeaderboard.php`. The page then calls the function `getLeaderboard()` in the `Leaderboard` class, which returns the leaderboard as a JSON string for the applications to parse.

Register New User

The application posts a request containing the username, password (md5 hashed), and the email to the page `register.php`. The page then calls the `register()` function in the `User` class, which checks to make sure the username and email are not in use already and then inserts the user in to the database if the checks pass. Correct username and email format should be checked by the applications (e.g. using regular expressions). The page displays the string "username or email in use" in the case of failure or the userID of the newly registered user in the case of success.

Log In Returning User

The application posts a request containing the username and password (md5 hashed) to the page `appLogin.php`. The page calls the `login()` function in the `User` class, which checks if there is a matching entry in the users table in the database. If a match was found, the page will display the userID of the matching user, else a -1 will be displayed signifying a failed login attempt.

Password Functions

There is a function implemented to allow a user to change his/her password. The application needs to post a request with the userID, current password (md5 hashed with the key password), new password (md5 hashed with the key newPassword) and the string "change" (with the key request) to the page `changePassword.php`. The page then calls the function `changePassword()` in the `User` class. This function first verifies that the current password is the same as the password in the database for the current user. If this checks out, then the password is changed to the new password. The page displays "password changed successfully" or "incorrect original password" depending on the situation.

Another function implemented is the ability to reset a password if the user forgot his/her password. This requires the application to post the username, email, and the string "reset" (with the key request) to the page `changePassword.php`. The page then calls the function `resetPassword()` in the `User` class, which first makes sure there is a matching user with the given username and email, then generates a string of 10 random alphanumeric characters, changes the user's password to that string, and emails the new password to him/her. The page displays the string "password reset" or "could not reset" depending on the situation.

Friends Functions

There are several functions related to the friends functionality in the applications. All the friends functions are handled by the page `friends.php`. The applications simply need to post requests to this page with varying information for the desired functions to be invoked.

The first is the ability to search for a friend based on username or email. The page needs the string "searchUsername" or "searchEmail" (with the key "request") and the username or email being searched. The page will display either the userID if the user is found or a -1 otherwise.

After searching for a friend, the user can add the other user as a friend. The application should post the string "add" (with the key "request"), the userID, and the friend's userID. The page calls the `addFriend()` function in the `User` class, which checks to make sure the users aren't friends already. It then adds them as friends if they are not friends already. The page displays "friend added successfully" or "already friends with user".

Removing a friend is very similar. Simply post the string "remove" (with the key "request"), the userID, and the friend's userID. The page calls the `removeFriend()` function in the `User` class, which checks to make sure the users are friends already. It then removes the friend from the user's list. The page displays "friend removed successfully" or "no friend to remove"

Lastly, the server can get a list of the user's friends. This is done by posting the string "list" (with the key "request") and the userID. The page calls the function `getFriendsList()` from the `User` class, which displays a JSON string containing the userID and scores of each friend for each category. This is also used by the applications to display the friends leaderboards.

iOS

Overview

The iOS application is written using the Model-View-Controller (MVC) architecture. This is the default preferred architecture recommended by Apple when working on iOS applications. An introduction to MVC can be found on Apple's documentation page. The following is a description of which classes belong to each aspect of MVC.

Model

Question.m - Used to store an individual question retreived from the `QuestionSet` object. Contains question text, right answer, wrong answers, member number from set, and category.

QuestionSet.m - Used to store a set of `Question` objects from a JSON received from the server. Parses the JSON string received based on the category requested and creates an array of `Question` objects.

User.m - Class that represents the current player. Used for logging in, registering, submitting scores, receiving question sets, viewing leaderboards, and utilizing friends lists. Contains the username, password, hashed password, userID from database, and email address.

Leaderboard.m - Class that a leaderboard for a particular category. Contains an array `LeaderboardEntry` objects that can be sorted by score. Used to display top scores in a particular category.

FriendsLeaderboard.m - Class for social leaderboard based on entries in a user's friends list. Contains `Leaderboard` objects for each category. Restricts scores to a subset of users.

LeaderboardEntry.m - Class representing a single entry in the `Leaderboard` object. Contains a score, a place (1st...Nth), and the user with the score.

LeaderboardJSON.m - Class for receiving JSON strings for leaderboard entries from the server. Requests and parses the leaderboard data.

View

The view for the iOS application is contained in the xib (nib) files. See interface section for a detailed overview.

Controller

The controller classes are described in the game functions section below. The controller is the interface between the model and the view.
Notable controller classes:

QuestionViewController.m - Controls the main game-flow one question at a time.

LogInViewController.m - Controls the login screen and posts requests for username/passwords.

RegisterViewController.m - Controls the register screen and posts register requests to the server.

LeaderboardViewController.m - Controls the global and friends leaderboard view and handles `Leaderboard` objects.

SelectCategoryViewController.m - Controls the select category view and loads a list of not yet played categories.

Play Game

When a user selects Play from the main menu, `LoginViewController` invokes the `SelectCategoryViewController` and makes the `SelectCategoryView` a subview. `SelectCategoryViewController` sends a POST request for the categories not yet played by the current user. The server returns a JSON string that is then parsed and input as cells in the `ListView` UI element on the screen from the `listData` array.

```
cell.textLabel.text = [listData objectAtIndex:row];
```

The `tableView` function is the function called by the `ListView` element to populate the list with categories. Once a user plays a category that day, they cannot play it again till the following day.

When a user selects a category, `SelectCategoryViewController` calls the `QuestionView`.

```
self.questionViewController = [[QuestionViewController alloc]
initWithNibName:@"QuestionView" bundle:nil category:rowValue];
[self.view addSubview:questionViewController.view];
```

This will invoke the `QuestionViewController` and create a new subview. During loading, all the UI elements are initialized and the main 10 second timer is started. The main execution is dictated by `timerFired`.

```
- (void)timerFired:(NSTimer *)theTimer{
      questionCountdown -= 0.001;
      if (questionCountdown > 1.0 && questionCountdown < 1.001) {
            [self loadAnswerButtons:ourSet andNumber:currentQuestionNum];
      }
      if (questionCountdown > 0 && questionCountdown <= 1.0) {
            [self timerFiredRoundInProgress];
      }
      if (questionCountdown < 0) {
            [self timerFiredRoundOver];
      }
      if (questionCountdown > 1.2 && questionCountdown < 1.201){
            [self timerFiredPostRound];
            [self clearAnswerButtons];
      }
      if (currentQuestionNum > 15) {
            [self endRound];
      }
}
```

The timer dictates what state the game is in.

```
[self loadAnswerButtons:ourSet andNumber:currentQuestionNum];
```

This method is called when the question first starts. It loads the question into the labels and the associated answers into the buttons.

```
[self timerFiredRoundInProgress];
```

This method updates the positions of the score labels as well as the progress toolbar.

```
[self timerFiredRoundOver];
```

This method resets the timer, notifies the user of the correct answer, updates the score.

```
[self timerFiredPostRound];
[self clearAnswerButtons];
```

This method loads the next question and clears the buttons.

```
[self endRound];
```

This method is called after 15 questions have been played. It brings up the `PostGameMenu` and gives the user the option to submit the score to the server.

View Leaderboard

There are two types of leaderboards that the user can choose from. The first is a Global Leaderboard that displays the highest scores in a particular category for that day. All users from all around the world who have submitted scores for that category on that particular day are taken into consideration. The second leaderboard is a Friends Leaderboard that only displays scores posted by the user's friends on that particular day for the specified category.

When the user selects that they wish to Leaderboards, they are presented with a TableView that contains a list of the possible categories to view leaderboards of. This is taken care by the `SelectLeaderboardCategoryViewController`. After the user selects the category, we initialize our view by overriding the `initWithNibName` function:

```
    self.leaderboardViewController = [[LeaderboardViewController alloc]
                   initWithNibName:@"LeaderboardView" bundle:nil
category:rowValue];
```

`LeaderboardView` is the name of the view we are loading, and the `rowValue` we are passing it as the category is the string containing the category that the user selected.

At this point we make a POST request to the server containing the global leaderboard JSON object by sending the following parameters:

```
      NSString *postBody = [NSString stringWithFormat:@"category=%@",
category];
```

Category contains the category string that the view was initialized with.

The application then parses the returned JSON object, formats them into a block of text in which each line contains the place, username, and score for each of the 5 high scores continued in the JSON object. This block of text is then displayed to the user.

Register New User

Handling new user registration is accomplished through the `RegisterViewController` class and `RegisterView.xib` view, and `User` class. Our .xib file contains 3 text boxes, which are for the user to enter a username, password, and email address. Before we submit this registration request to the server, we must do some sanity checks on the user input. For example, we want to make sure that the email address provided fits common the email format of “xxx@xxxx.xx”. We accomplish this with a regular expression check, performed by `checkEmail` of the `User` class.

```
        -(BOOL) checkEmail{
                //validates email using regular expression
                NSString *emailRegex = @"[A-Z0-9a-z._%+-]+@[A-Za-z0-9.-]+\\.[A-
        Za-z]{2,4}";
                NSPredicate *emailTest = [NSPredicate predicateWithFormat:@"SELF
        MATCHES %@", emailRegex];

                //check regular expression
                if ([emailTest evaluateWithObject:email] == NO){
                        return NO;
                }
                return YES;
        }
```

Assuming that our sanity checks pass for all 3 input fields, we then make a POST request to the server’s register page of the format:

```
        NSString *postBody = [NSString
                                    stringWithFormat:@"username=%@&password=%@&email=%@",
                            username, md5pass, email];
```

The `username` and `email` are directly from the fields the user entered, and the `md5pass` is the MD5 hash of the password the user entered. We do this because we do not want to transmit the users password in the clear, nor do we want to be responsible for storing their actual password.

We then wait for the server to respond to us. A response of “username or password in use” means that we must inform the user that the must try again with new information. This is accomplished by setting the `errorText.text` string, which is an entry in our .xib view file. If the server responds with a number, that number is the user’s userID, and thus they have successfully been registered.

Log-In Returning User

Upon first launching Triviarea, the user is presented with a screen that forces them to log in before they can proceed. Our `RootViewController` class manages this screen. The corresponding view file contains 2 text fields: one for entering a username and one for entering a password. When the user presses the “Log In” button, we make a POST request to the server’s login page of the format:

```
        NSString *postBody = [NSString
stringWithFormat:@"username=%@&password=%@",
        username, md5pass];
```

The `username` is directly from the field the user entered, and the `md5pass` is the MD5 hash of the password the user entered.

The server responds with “-1”, that means the log-in failed. This could either be because the username does not exist in the database, or the password does not match for the username provided. Otherwise, the server will respond with a number that corresponds to the user's userID. (Note: no userID will be -1)

After the server has responded with a valid userID, we want to store this ID in our data.plist file that is located on the device. This is used by other functions, so it is valuable to keep track of this.

Finally, we allow the user to proceed by displaying the `LogInView` view.

Android

Overview

Android Manifest

The manifest presents essential information about the application to the Android system; information the system must have before it can run any of the application's code. Each project has one Manifest that is automatically generated from the start. When adding new Activities, **you must** add the activity to the android manifest before calling them.

Resources (res)

Layout - Each activity should have a corresponding XML file to describe its user interface. Add XML files into the res/layout folder, -land if they are a landscape XML. Set a layout in activity with `setContentView(R.layout.layout_name)`. After calling `setContentView` you can get a handle on View with setIDs from the XML layout folder using `findViewByID(R.id.view_id)`. See Android UI Section for an example; calling before will cause a `null pointer exception`.

Values - Stores constant values (String, Colors, etc) in XML. This can be found in res/values. To access a value in an Activity use `getResources.get[String, Color]R.[string, color].resource_id)`. Storing values in XML allows for easier changes later.

Drawable - All images and icons used by the application are stored in the `res/drawable-hdpi/mdpi/ldpi` with each `dpi` standing for high, medium and low resolutions, respectively.

Classes

Context - Interface to global information about an application environment. Allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting, and receiving intents.

Activity - Subclass of `Context`. Single, focused thing that the user can do. Activities interact with the user, so the Activity class may take care of creating a window to hold the set UI, done with `setContentView(`*`view`*`)`.

View **-** This class represents the basic building block for User Interface components. `View` is the base class for `widgets`, which are used to create interactive UI components like buttons or text fields. `View` are typically defined as XML documents and are stored in the res/layout folder. `Views` can by accessed in an activity. See the Android UI section for more information and an example.

Intent - An abstract description of an operation to be performed. Used with `startActivity(intent)` to load a new activity. Use method `putExtra()` and `getExtras()` to pass variables to and from Activities.

R - Access resources with the automatically generated `R` class. It is advised against modifying this file on your own; instead define new views in XML or new activities in the android manifest. Use in combination with `Context.getResource` to inflate a particular resource.

Packages

android.triviarea.data

This package includes classes with correspond to both the model and controller in a model view controller architecture. These classes store and manipulate the data associated with playing a game.

GameSession.java - Holds all data and methods used to play game. Handles communication between the application and server. It is also used to parse the data returned by the server. The data may include leaderboards, categories, questions, and committing a score.

User.java - Stores all the data and methods associated with the person currently logged into and playing a game. Connects to the server for logging in, registering, and requesting and handling friends.

Questions.java - Data class used by `GameSession` to store an individual question. This includes the question text, three incorrect solutions, and the correct solution.

android.triviarea.main

Classes in this package extend the class Activity for the Triviarea GUI

Some notable classes:

MainActivity.java - This is the first activity launched and provides the user with an interface for logging in. If the user does not wish to log in, then they may either register as a new user or notify the Triviarea database that they have forgotten their password.

StartGame.java - Handles the UI for the game loop and the timers associated with animating the game and displaying the sequential questions. This class interacts heavily with the GameSession class.

FriendScreen.java - Handles the friend screen adding and deleting. Allows user to view friends leaderboard.

Register.java - Handles the new user registration process.

ManualPasswordReset.java/AutoPasswordReset.java - These two activities allow the user to change their password to something new of their choice (Manual) and allow the user to request the database to change it (Auto).

MainScreen.java - Represents the main menu of the game with buttons to leaderboards, friends list, password changing, logging out, and starting a game.

android.triviarea.threads

Classes included extend the `Thread` class. Used to multi-thread a task in parallel to UI threads. Mostly used to handle background requests and commits to the server.

android.triviarea.utility

Static helper functions and utilities can be found in this class. `Internet.java` is found within this package and is used extensively for posting to the server.

Play Game

In the Android OS, the actual game is controlled through an event loop in which different events are scheduled one after another. These events are basically `TimerTasks` that run in a specific order that is maintained by the use of a semaphore. When a timer finishes a task, it releases the semaphore so that the next task can acquire the semaphore and then proceed. This process is depicted by the code fragment shown below:

```
setUpGameTimerTasks();

questionTimer.schedule(startQuestion, 100); // FREE

gameLock.acquire(); // LOCK

displayAnsTimer.schedule(displayAnswers, 1500); // FREE

gameLock.acquire(); // LOCK

countdownTimer = new Timer();
countdownTimer.schedule(countdown, 10); // FREE

gameLock.acquire(); // LOCK

answerTimer.schedule(displaySolutionTallyPoints, 3000); // FREE

gameLock.acquire(); // LOCK
```

The `startQuestion` task retrieves the question for the current round and displays it on the screen. It also initializes and displays other information like the player's score and potential score for that round as well as the time left for that round. It also ensures that the answer choices will be displayed in a random manner so that the correct answer does not always occupy the same position on the screen.

The `displayAnswers` task runs once the `startQuestion` task finishes and it displays the four answer choices to the user, one after another in a delayed manner so that the answers appear to fan out onto the screen.

The `countdown` task which runs next, updates the time bar, time left and potential score on every tick of the timer. This ensures that the user sees the time continuously flowing down in the time bar and also in the other text displays. Before the `countdown` task finishes, it takes the player's selection at the end of the round and awards points if necessary. It also highlights the correct answer in green and the wrong selection in red. `DisplaySolutionTallypoints` is then started; it hides the buttons and resets everything before the next round begins.

If the user decides to quit the game or presses the back button before the end of the game, we ensure that the main event thread is interrupted and that all timers and associated tasks are destroyed. This is done so that if the user then starts a new game, it does not receive interference from any previous games that should not be running. However, if the user does finish the game then his/her final score is sent back to the server.

View Leaderboard

When the user decides to view either the Global or Friends leaderboards, he is first presented with the `categoryScreen`. This screen displays a list of all possible categories and the user selects which category he is interested in. Then, depending on where in the application the user had clicked the View Leaderboard button, the Global or Friends Leaderboard is displayed. The leaderboard displays the current date and category selected on the top as well as a list of users and their corresponding high scores.

If the Global Leaderboard is being displayed, then the `requestLeaderboard()` method in the `GameSession` class is called to retrieve the leaderboard from the server. The application posts a request to the server and the server replies with the leaderboard in the form of a JSON string. A similar process is followed for the Friends Leaderboard feature except that the `requestFriendsScores()` method is called instead. The core part of this method is shown below:

```
for(int i = 0; i < numFriends; i++)
{
	JSONObject friend = json.getJSONObject("friend" + String.valueOf(i));
	myFriends = myFriends + friend.getString("friendName");
	myFriends = myFriends + "\n";
}

for (int i=0; i<9; i++)
{
	String score = "";
	String category = allCategories[i];
	for (int j=0; j<numFriends; j++)
	{
```

```
            JSONObject friend = json.getJSONObject("friend" +
                                                  String.valueOf(j));
                  score = score + String.valueOf(friend.getInt(category));
                  score = score + "\n";
      }
      friendsScores.put(category, score);
}
```

This method retrieves the user's list of friends and their scores from the server and parses it to form the Friends Leaderboard. First, the list of friends is created and then their scores in the category selected are obtained.

Register New User

When a new user wishes to play the game, he must first register with the server. He can do this by pressing the Register button on the main screen. The Register activity is then launched and it allows the user to enter his preferred username, password and email address. The application then ensures that all the information entered by the user is valid. More specifically it checks that the values entered match the regular expressions shown below:

Username regular expression: `\\w+`
Password regular expression: `^[\\S]{5,}$`
Email regular expression: `^[\\w.%+-]+@[\\w.-]+\\.[a-zA-Z]{2,4}$`

If any of these fields are invalid then the application prompts the user to correct the problem. It also ensures that the previous user's information is still maintained in case the new user decides that he no longer wishes to register and presses the back button.

Once validity of entered information is guaranteed, the application posts a registration request to the server. This is done through the code snippet shown below:

```
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("username", username));
params.add(new BasicNameValuePair("password",
Internet.convertToMD5(password)));
params.add(new BasicNameValuePair("email", email));
newUserID = Internet.Post(REGISTER_URL, params);
```

What is interesting to note here is that the password entered by the user is hashed before it is sent over the network. The hashing algorithm used is MD5 hashing and this process ensures that the user's password is never determined even if a third party intercepts communication between the application and server. The code to hash the password is shown below:

```
MessageDigest digest = MessageDigest.getInstance("MD5");
byte[] bytes = digest.digest(string.getBytes());

// Create Hex String
```

```
      StringBuffer hexString = new StringBuffer();
      for (int i=0; i<bytes.length; i++)
            hexString.append(Integer.toHexString(0xFF & bytes[i]));
      MD5_string = hexString.toString();
```

Log-In Returning User

The process to login a returning user is similar to the process of registering a new one. The application reads the username and password entered by the user and posts a login request to the server. Once again the password is hashed to its MD5 value for security reasons. The server checks if the username and password match any pair in its database and returns with a user id if login was successful or an error message if not.

There is one major aspect that differentiates the login process from the registration process; and that is context saving. When a user exits the application, his username and password are stored in a text file on the phone. When he starts up the application the next time around, his username and password are retrieved from this file and are automatically entered for him. This greatly increases the usability and friendliness of our application since the user does not have to waste time entering his name and password every time he wishes to use the application. The following code excerpts depict the context saving and restoring functionality.

```
private void saveInfo(Context save)
{
      FileOutputStream fos = save.openFileOutput(INFO_FILE_NAME,
      ContextWrapper.MODE_PRIVATE);
      fos.write((username + ";" + password + ";").getBytes());
}

private void loadInfo(Context load)
{
      String username;
      String password;
      FileInputStream fis = load.openFileInput(User.INFO_FILE_NAME);
      byte[] buffer = new byte[256];
      fis.read(buffer);
      fis.close();

      //grabbing stored username and password
      String [] result = (new String(buffer)).split(";");
      username = result[0];
      password = result[1];
      //set username and password in the text fields on the screen
      this.setUsername(username);
      this.setPassword(password);
}
```

V. User Interface Description and Implementation

There were two primary goals of designing the Triviarea user interface. The first was to design an interface as simple as possible. That is, we did not want to clutter the screen with unnecessary data. We wanted to provide a layout that was familiar to users and made use of natural "quiz" mappings, and we wanted the flow to be quick and intuitive. The second goal was to unify the Android UI and iOS UI as much as possible.

We wanted to give the users a few basic elements that would make the game familiar and easy to understand. Our top-down design provides the most relevant information at the top of the screen starting with the current question. The users are given a brief period to begin reading the question before potential answers are displayed. The answers come in the form of four large buttons arranged in a 2x2 grid similar to game shows on television and making use of limited screen space. Below the answers is other relevant information such as the timer bar, score countdown, seconds remaining, and the potential score once an answer is selected. The timer bar provides the user with a quick glance at the remaining time, while more detailed information (if desired) can be viewed below that either during or between questions.

Android

The Android UI was designed to primarily use XML layouts. It makes use of the standard interface libraries provided to developers. These two components allowed us to quickly generate layouts, but more importantly kept the code cleaner and gave us the ability to quickly alter UI. The use of standard UI functions prohibited us from wasting large of amount of time and resources developing a custom UI and gave more focus to providing game functionality.

There are roughly a dozen XML layouts, one for each screen of the application. The majority of the layouts are comprised of hierarchical `LinearLayouts`. These layouts bind all components to a restricted area either vertically or horizontally. An excerpt from mainscreen.xml is provided below.

```
<!-- Signed-in User Section -->
<LinearLayout android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight=".2"
      android:gravity="center">

      <TextView
            android:layout_width="wrap_content"
            android:text="Logged in as: "
            android:textSize="18px"
            android:layout_height="wrap_content"
            android:layout_gravity="center"/>

      <TextView
            android:layout_width="wrap_content"
```

```
                        android:text=""
                        android:textSize="18px"
                        android:layout_height="wrap_content"
                        android:layout_gravity="center"
                        android:id="@+id/displayuser"/>
            </LinearLayout>
```

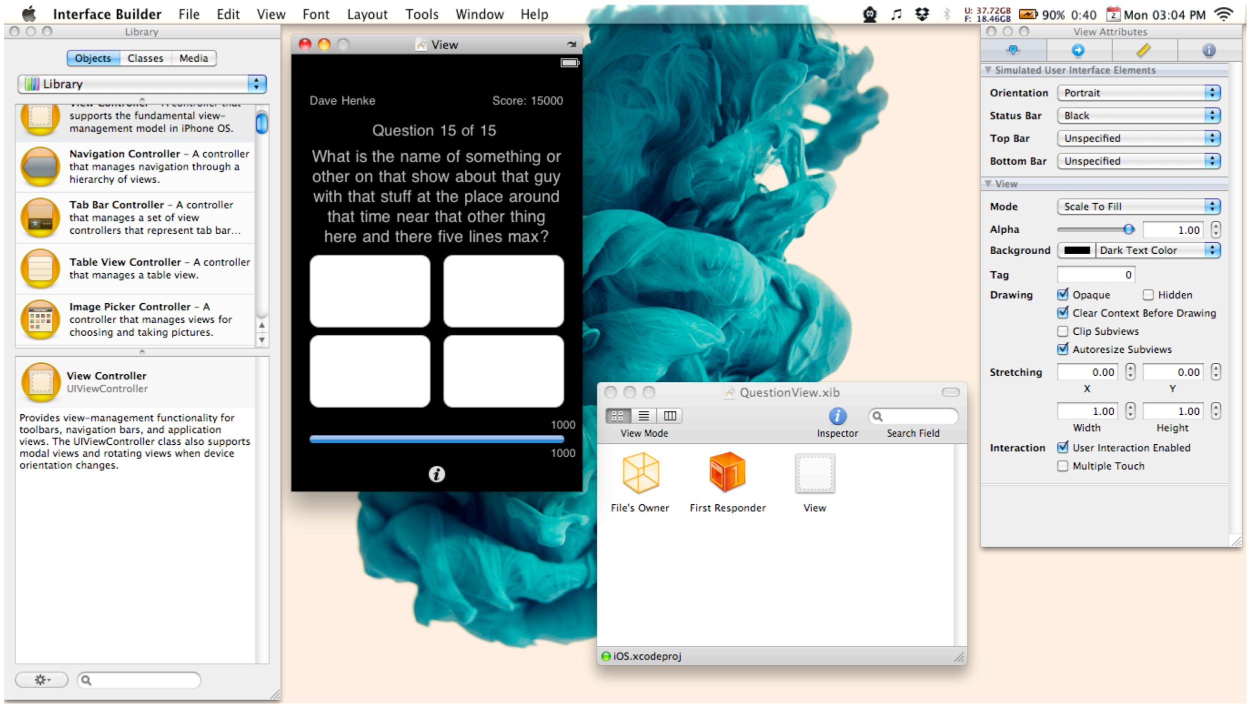
The above XML displays who the current logged-in user is. The `LinearLayout` displays included components in a horizontal fashion and is itself buried in other `LinearLayouts` not shown. The two `TextViews` display the text onscreen. The first displays a static string "Logged in as: " while the second one dynamically changes based on the logged in user. This is done through the id tag at the bottom of the component. The java code interfaces with these id tags as shown below. Note: `user` is a private global variable used to hold a String of the username.

```
user = User.getInstance();
TextView usernameView = (TextView) findViewById(R.id.displayuser);
usernameView.setText(user.getUsername());
```

This code portion grabs the currently logged-in user's name from the User object and sets it to the layout via the Resource id.

iOS

The iOS development framework includes a graphical Interface Builder program. The .xib (pronounced "nib") files store the UI information and can be edited interactively. The classes that interface with the xib files have the name `ViewController`. This is the primary connection between a graphical xib and the project code. Figure 5.1 shows the Interface Builder layout for `QuestionView.xib`. The left panel contains UI elements that can be dragged to the main view and then manipulated using a view controller, the right panel inspector is used to change the attributes of the currently highlighted element, and the browser contains the object classes and elements associated with this xib. For a more detailed layout description, refer to Apple's Interface Builder documentation in the iOS Dev Center.



supports the fundamental view-management model in iPhone OS.

Navigation Controller – A controller that manages navigation through a hierarchy of views.

Tab Bar Controller – A controller that manages a set of view controllers that represent tab bar...

Table View Controller – A controller that manages a table view.

Image Picker Controller – A controller that manages views for choosing and taking pictures.

View Controller
UIViewController

Provides view-management functionality for toolbars, navigation bars, and application views. The UIViewController class also supports modal views and rotating views when device orientation changes.

Dave Henke Score: 15000

Question 15 of 15

What is the name of something or other on that show about that guy with that stuff at the place around that time near that other thing here and there five lines max?

1000

1000

▼ Simulated User Interface Elements

Orientation: Portrait
Status Bar: Black
Top Bar: Unspecified
Bottom Bar: Unspecified

▼ View

Mode: Scale To Fill
Alpha: 1.00
Background: Dark Text Color
Tag: 0
Drawing: ☑ Opaque ☐ Hidden
☑ Clear Context Before Drawing
☐ Clip Subviews
☑ Autoresize Subviews
Stretching: 0.00 X 0.00 Y
1.00 Width 1.00 Height
Interaction: ☑ User Interaction Enabled
☐ Multiple Touch

Figure 5.1: Interface Builder

In order to associate a UI element with an object, a referencing outlet must be created. For example, in `QuestionViewController.h`, in order to control the progress view element seen on the screen, first the header file must declare an `IBOutlet` property for the `UIProgressView` class.

```
@property (nonatomic, retain) IBOutlet UIProgressView *questionTimer;
```

This assumes there is a pointer `UIProgressView *questionTimer` in the header interface section. Now, in the implementation, the property must be synthesized.

```
@synthesize questionTimer;
```

After saving the implementation and interface files, the `UIProgressView` element in Interface Builder can be associated with `questionTimer`. The synthesized member object must be linked as the referencing outlet to the `UIProgressView` element. To accomplish this, select the `UIProgressView` element in Interface Builder and navigate to the second tab in the Inspector panel (on the right). Now, simply drag from the radio button to the File's Owner object in the browser and select `questionTimer`. Figure 5.2 shows what this process should look like and figure 5.3 shows the resulting options.

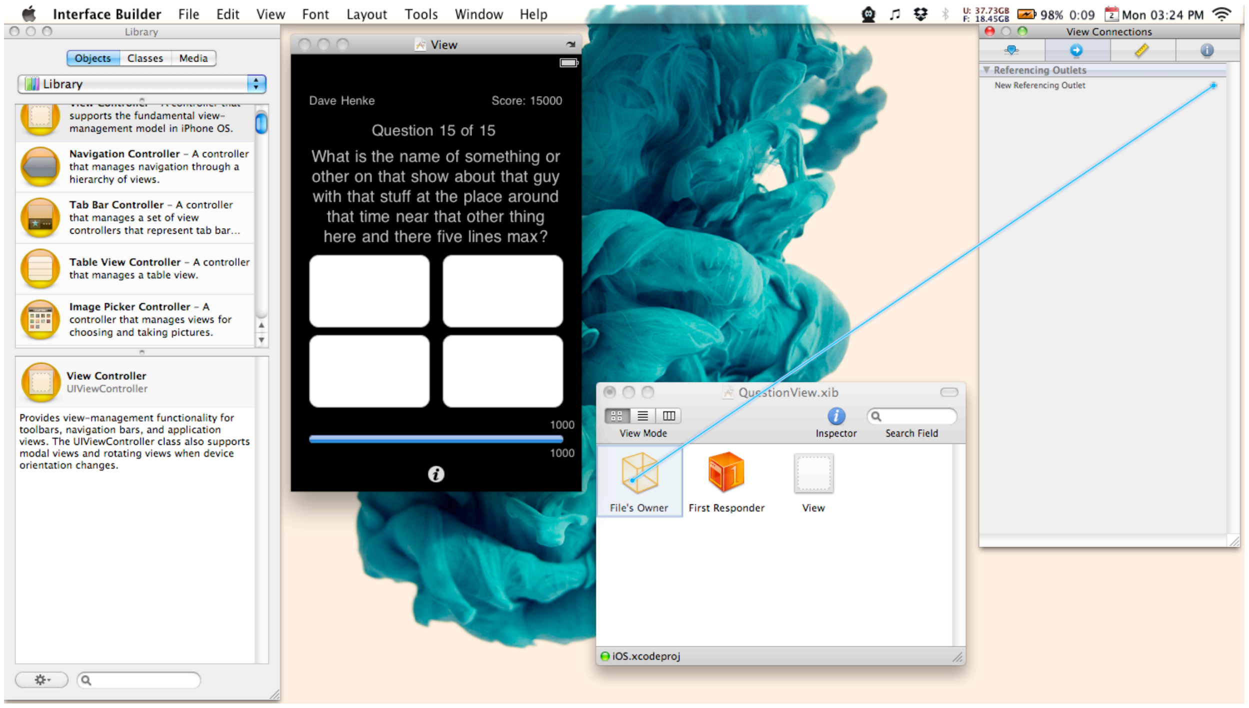

Interface Builder File Edit View Font Layout Tools Window Help
Library
Objects Classes Media
Library
Navigation Controller – A controller that manages navigation through a hierarchy of views.
Tab Bar Controller – A controller that manages a set of view controllers that represent tab bar...
Table View Controller – A controller that manages a table view.
Image Picker Controller – A controller that manages views for choosing and taking pictures.
View Controller
UIViewController
View
Dave Henke
Score: 15000
Question 15 of 15
What is the name of something or other on that show about that guy with that stuff at the place around that time near that other thing here and there five lines max?
1000
1000
QuestionView.xib
View Mode
Inspector
Search Field
File's Owner
First Responder
View
iOS.xcodeproj
View Connections
Referencing Outlets
New Referencing Outlet

Figure 5.2: Linking in interface builder

`questionTimer` can now be used to reference and manipulate that element on the screen. For example, the bar gets updated whenever the timer changes value to reflect question time remaining.

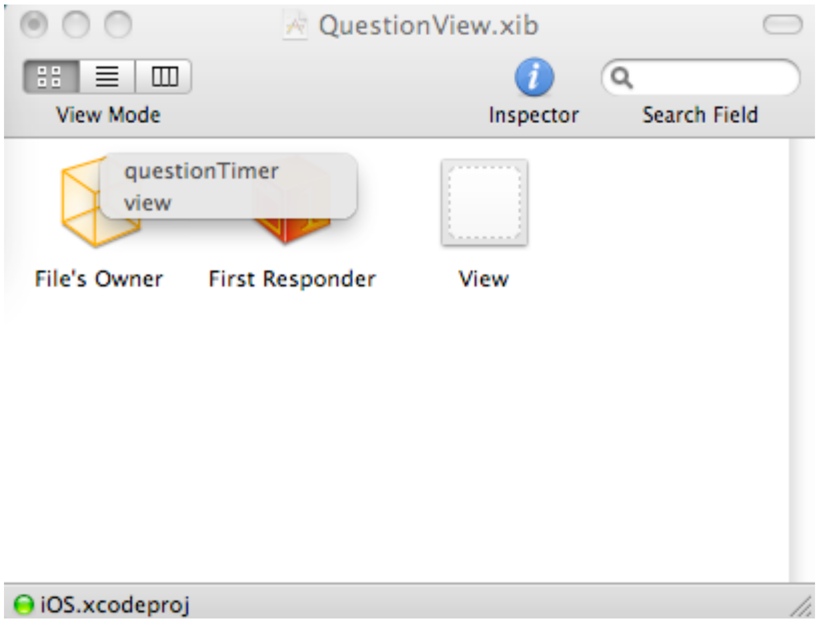


Figure 5.3: Choosing referencing outlet

In order to unify the interface with the Android OS, `UIButton` elements had to be sub-classed in order to draw a custom button style that reflects that of Android. The class `CustomButton` overrides the `drawRect` method to display a custom image on the `UIButton`, and the button element's class is changed to this subclass in Interface Builder. A similar approach was used to create a orange progress bar using `PDColoredProgressView`, an open-source class found online.

VI. Future Plans

Future Plans

We plan to submit our applications to both the Apple App Store and Android Market. However, we have a few things to polish up before they are ready. Most importantly, we will need to find a new server, since we are currently using the server provided by the university. Also we had wanted to implement Facebook Connect (so users can play against their friends on Facebook as well as friends in the app), but we ran out of time during the semester. This is a feature we would like to include before submitting the applications. Ultimately, we will keep in contact with each other with statuses of the various parts of this project. All the code from all three teams will be packaged and hosted on a cloud storage sites, accessible to all members.

Personal Reflections

Dave Henke - henke2

I had a great experience working on Triviarea. This project was a great opportunity to hone my iOS development and team-work skills. I got to enjoy working with great people on an idea that ended up fun and exciting. Even with something as small as a trivia game, I learned to appreciate the intricacies of writing software from the ground up, and the game itself also seems viable for further maintenance and development. I only had experience with a particular open-source iOS game framework before this project. After using the standard iOS interface and framework, my Objective-C and interface builder skills can now handle any idea I might cook up. This was one of the most rewarding experiences of my undergraduate career, and I can now apply software engineering techniques where needed, and begin releasing personal iOS apps.

Neeraj Ramesh - nramesh2

Working on the Triviarea project was a great experience for me. I had the opportunity to continue working with some of my peers whom I had worked with during CS427 and also got the chance to pair program with a couple of new developers on the Android team. Overall it was a very fulfilling experience. I was able to hone my Android developments skills by picking up a few tips and tricks along the way. I was also exposed to a number of new concepts like JSON strings, TimerTasks and MD5 hashing and I definitely feel that my overall programming knowledge has increased tremendously.

Tom Rudwick - rudwick2

I had very informative experience this semester on our project. This was my second time working with PHP, but this time, I actually had a partner who knew what he was doing. I basically learned the whole language from scratch. It was nice to have a project that had real deadlines, and unlike Photran, this seemed like a realistic project. I thought my group members were all very hard working. I had never created a working server back end before, and I thought this was a great learning experience, and well most certainly help in the future, as my job after college will be directly related to database access. In general, this was a useful project that was quite enjoyable to work on, and I am glad everything worked out as well as it did.

Ryan Sullivan - rsulliv7

I had a great time working on this semester's project. Triviarea was my brain child, so it was extremely fulfilling seeing an idea that you came up with be turned into a working application that actually has future potential. The development of the project was fairly straight forward with no major curve balls being thrown at us at the last minute. I feel like the modularized development process that we used worked great for this project, will still giving us the feeling that we are working as a larger team. I learned valuable Objective-C skills in which i had no prior experience with, and I also feel like I picked up a lot of project management skills. I would like to comment that, oddly enough, one of the most challenging parts of the project have been coming up with new questions. There are no free question databases, and coming up with incorrect but plausable wrong answers is tougher than it sounds!

Anthony Chan - aychan3

I found this semester's project to be a much better experience than last semester's. I actually wanted to work on this project and enjoyed working with my group members. This project gave me some good experience in working in a small software engineering group using XP. I really liked the schedule of weekly meetings and frequent releases. Because of them, I was able to discipline myself and set weekly goals to meet. In addition, I also honed my skills in PHP, which I know will benefit me in my future career. All in all, this was a fun project on which to work and I now have a product I can show others.

Samuel Smith - smith137

This semester has been a fantastic learning experience. The fall semester was a good way to meet new people and work on a development team. However, this semester was great in that I was able to work with a project that interested me. I chose Triviarea because I wanted to become a stronger developer with Android and a casual trivia game sounded like a good starting point. The project itself and working in a group environment not only motivated me, but also allowed for the freedom to branch our XP knowledge and make the learning experience more memorable.

The ability to strengthen my programming skills on an intriguing platform has piqued my interest in developing something on my own.

Chad Schwab - schwab2

At the start of this semester, I couldn't decide between working in this group as an inexperienced android developer or in another as an experienced XNA programmer. Fortunately, I decided at the last moment that gaining experience in android development was too intriguing and idea to pass up. The team is comprised of some awesome people and the other android programmers taught me a lot. Throughout the project my knowledge of the android SDK has grown immensely. I believe this knowledge is the greatest I will take out of CS 428. I plan on releasing a series of android apps over the summer, which will be great to show off when looking for jobs in the fall.

VII. Appendix

Installation Instructions

PHP

1. Check out the code from **SVN**.
2. Create folders inside the `~/public_html/` folder called `classes`, `tests`, and `scheduledJobs`.
3. Place all the class files into the `classes` folder and test files into the `tests` folder. The Simple Test library should also go in the `tests` folder.
4. The file called `doMe.php` should go in the `scheduledJobs` folder. Create a cronjob in cPanel to run this php file everyday at midnight.
5. The rest of the files should sit in the `~/public_html/` folder.

Android

1. The initial steps are provided by Google, and include the following.
   a. Installing Eclipse
   b. Installing the Android SDK
   c. Installing the ADT Eclipse Plugin
   d. Adding an AVD (Android Virtual Device, emulator)
2. Check out the code from **SVN**
   a. Specifically Android.Triviarea and Android.Triviarea.Tests
3. Import Android.Triviarea and Android.Triviarea.Tests into Eclipse.
4. Use Eclipse to run the application on a virtual device. You must initially set up a device using the SVD Manager in Eclipse.

5. (Optional) To run on an Android device you will need to allow your handset to install unverified code. This a check-mark in the system settings. You will also need to find a free, third-party application on the Android market that can install .apk files. After running the code in Eclipse an .apk should be generated in the /bin folder of the project workspace. Move this .apk to the device and use the third-party application to install.

iOS

iOS Version requires iOS SDK Version 3.2 or higher and the Xcode 3 IDE.

1. Check out code from **SVN**.
2. From the iOS folder, open iOS.xcodeproj.
3. The groups pane on the left contains all relevant classes and resources.
4. We must now import our JSON library. It can be found here. The contained folder should be copied into the repository we checked our from SVN.
5. (Optional) To run Triviarea on an iOS device, see the Provisioning and Certificate resource on the iOS Dev Center. Requires an iOS Developer subscription.

How to Use

Main Menu

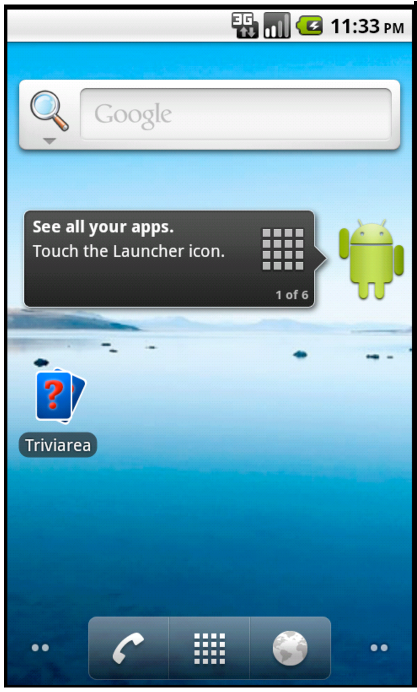
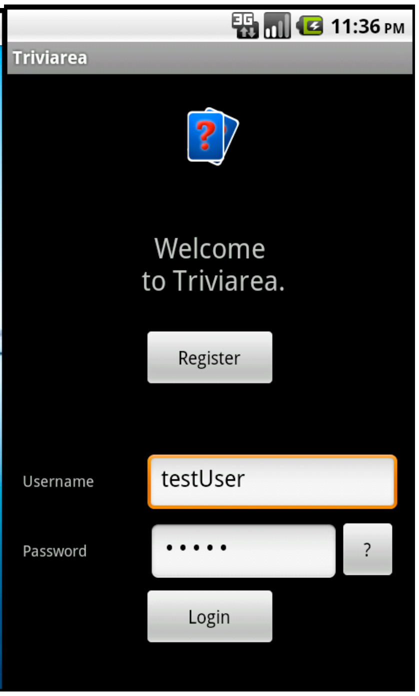


Figure 7.1



Welcome to Triviarea.

Username testUser

Password •••••

Figure 7.2

The home page will look something like figure 7.1. Select the Triviarea icon to start the app. This will then direct you to the Login page (Figure 7.2). If you do not have an account you will select register. This will send you to the register page, which is described below. If you have already logged in on this particular phone, it will remember your information, and it will fill it in. Otherwise you will have to type it in manually. If you have forgotten your password, you can click the question-mark, which will refer you to page to reset your password, which is also described below. Once you input your information, you click the Login button, which will send you to the Home Page.

Home Page

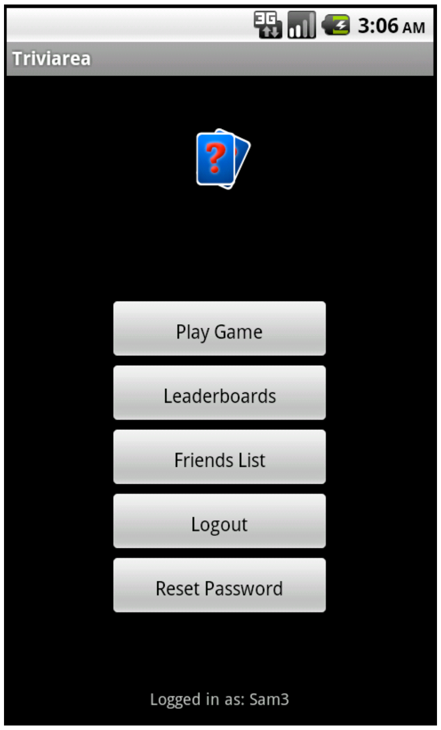
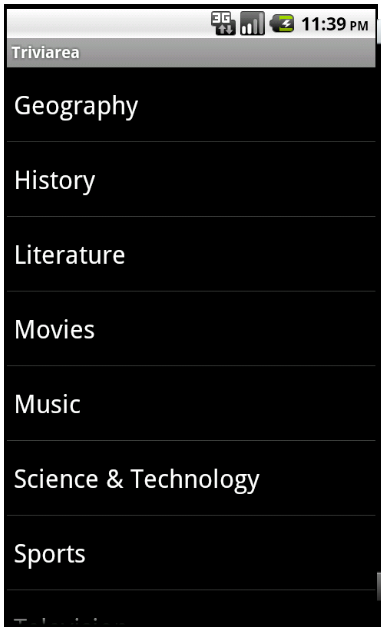
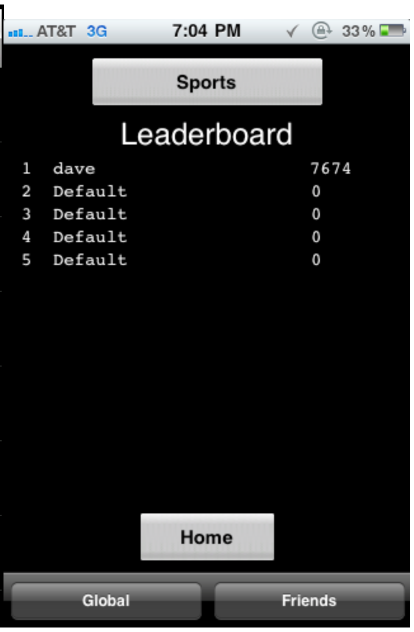


Figure 7.3

Once you login, you will see the Main Menu, which will look like Figure 7.3. There will be options to play the game, view Leaderboards, select your Friends List, Logout, and Reset your password. For a description of each of these, refer to the corresponding section. If you select Play Game, it will first direct you to Select Category. Once you select one, you will then start the game.

Select Category and Leaderboards





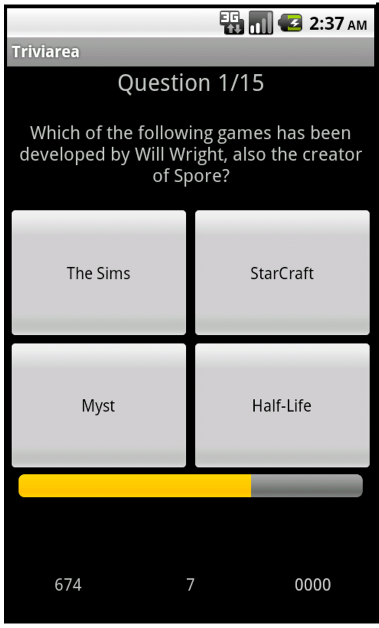
```
1  dave              7674
2  Default           0
3  Default           0
4  Default           0
5  Default           0
```

Figure 7.4 Figure 7.5

When you select Play Game, you will see Figure 7.4, which will be the category of questions you are about to play, or the scores you are about to see. Since you can only play each category once per day, so if you have already played a specific category already, it will be greyed out until the next day.

When you select Leaderboard, you will then select the category to view. All of the categories will be an option for Leaderboards. Once you select a category, you will then see Figure 7.5. It will display the scores of the top five games in that particular category. If you select the friends list option at the bottom, it will display the scores of all of your friends in that particular category. If they have not played it, it will display as zero, similar to the default zero on the overall category Leaderboard as shown in the figure.

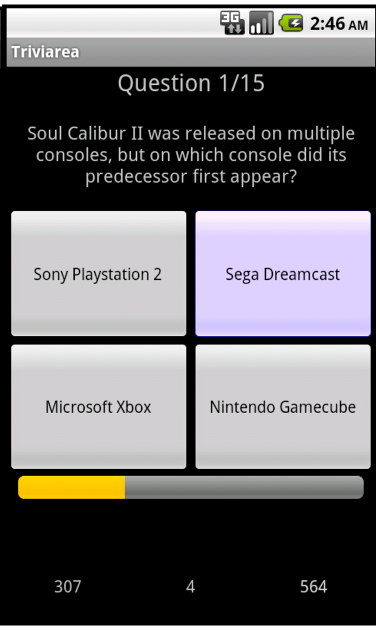
Playing The Game


Triviarea

Question 1/15

Which of the following games has been developed by Will Wright, also the creator of Spore?

Figure 7.6


Triviarea

Question 1/15

Soul Calibur II was released on multiple consoles, but on which console did its predecessor first appear?

Figure 7.7

Once you have selected a category, the game will start. There will be 15 questions in that particular category. A question will look like Figure 7.6, but the answers buttons will be blank to start. Once a question has displayed for two seconds, the answers will appear. You will then have 10 seconds to guess an answer. There will also be a timer, which is displayed as the yellow bar. This will move from right to left, which is signaling how much time you have left to answer, as well as how many points you receive once you select an answer. For example, if you select an answer when the timer says 600, you will get 600 points if you picked the correct answer, and obviously 0 if you answer incorrectly. You can change answers, but you will receive less points, as the timer does not stop. You can also lock in your answer by holding down this selection. That way you do not need to wait the whole time if you are confident in your selection. When you select an answer, it will turn the purple color, shown in the middle.

Once time runs out, if you have selected the correct answer, you your selection will turn green, and the score that was locked in will be added on. If you selected the incorrect answer, your answer will turn red, and the correct answer will display as green.

Once you have answered all 15 questions, your final score will display. Assuming you select this, your score will be submitted to the database. You will then return back to the main menu. Note that the next time you try to play a game on the same day, the previously played category will not be available.

Friends List

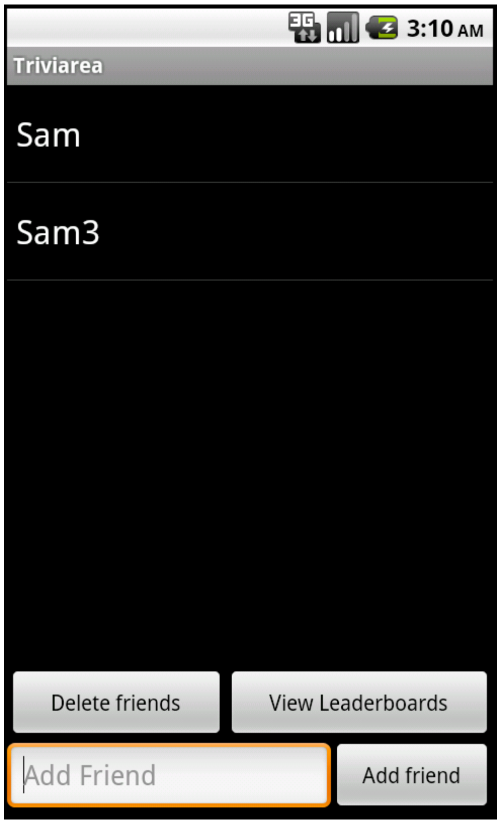


Figure 7.8

The Friends list has a number of features. You can add any user to your friends list. They do not have to add you, but you can add them. We figured that since it is harmless, there is no problem adding other people without confirmation. To add someone, simply type in their username, and then click the Add friend button.

If you choose to delete a friend, you simply click on the user's username, and then select the delete button, as shown in Figure 7.8. You can select multiple people to delete at the same time.

Password Resetting

If you are not logged in yet, and you click the question mark next to the password box, you can reset your password. It will refer you to Figure 7.9, where you will input your username and email, and then click submit. If will send you an email with a randomly generated password. If you incorrectly input this information, you will receive an error message. If you are already logged in, you will go to a very similar page. You will enter your current password, along with a new one (twice to ensure you entered it correctly). This will change the saved password in the database. Similarly, you will get an error if information is inputted incorrectly.

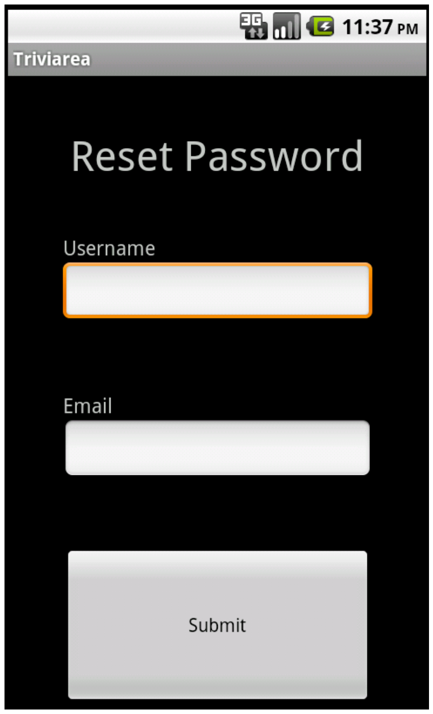


Figure 7.9

Testing

- PHP Unit Tests
  - We use Simple Test (http://www.simpletest.org/) for our testing framework. The library should be included in the `~/public_html/tests/` if the code is checked out from SVN. Else, the library should be downloaded from the website and extracted in that location.
  - To run the tests, simply go to http://triviarea.projects.cs.illinois.edu/tests/all_tests.php in a web browser.
  - To write new tests, follow the instructions located here http://www.simpletest.org/en/start-testing.html.
- iOS Unit Tests
  - Installation
    The iOS tests are written using the Google Toolbox for Mac iPhone Unit Testing toolkit. The kit must be downloaded from http://code.google.com/p/google-toolbox-for-mac/wiki/iPhoneUnitTesting. In order to run the Triviarea test suite, the tester must change the script file-path from step 16 to point to the location of his Google Toolbox download.
  - Running Tests
    Unit tests can be found in the Tests grouping folder in the iOS xcode project. The project also includes a unit testing target. To run the tests, select Set Active Target->TriviaUnitTests from the Project menu. The tests execute during the build phase, so it is not necessary to execute the program. Build the TriviaUnitTests target. The results from the unit testing script will be displayed in the Xcode Build Results window.
  - Creating Tests
    Detailed instructions for creating tests can be found at http://code.google.com/p/google-toolbox-for-mac/wiki/iPhoneUnitTesting. Create a new unit testing class in xcode as needed and add it to the project. Then change the subclass to the google testing kit class, and use the ST assertion macros to implement tests. All test methods must begin with the word `test`. Additionally, `setUp` and `tearDown` methods are available.
    Note: It may be necessary to remove the entry `MainWindow` from the target plist file until to fix a potential build bug in the toolbox. More information can be found in the comments section of the google toolbox wiki.
- Android Unit Tests
  - Installation
    The Android Testing Framework is included in the SDK and based on the JUnit testing framework. See Android installation instructions for details about installing the SDK.

- Running Tests
  The tests are stored in the default package of the src folder for the project Android.Triviarea.Tests. To run a test, open it in the eclipse editor window, click Run->RunAs->Android Junit Test.
- Creating Tests
  Right click the src folder in Android.Triviarea.Tests. Go to new->JunitTestCase. Change the superclass from junit.framework.TestCase to android.test.AndroidTestCase. This allows you to get a handle on Triviarea's Context with the inherited method getContext() which is necessary for some classes in the Triviarea project.
- Writing Tests
  Tests are written using JUnit version three syntax. For more information on Junit, see the testing fundamentals section of the Android Development guide, http://developer.android.com/guide/topics/testing/testing_android.html.

- Manual Interface Testing

Manual testing documentation can be found on our group wiki at Manual Testing Documentation.