Copyright

by

Frank Mintzu Huang

2012

The Report Committee for Frank Mintzu Huang
Certifies that this is the approved version of the following report:

SportMingles Sports Social Network for iOS

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Adnan Aziz

Christine Julien

SportMingles Sports Social Network for iOS

by

Frank Mintzu Huang, B.S.C.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2012

Acknowledgements

I would like to give special thanks to my supervisor, Dr. Adnan Aziz, and reader, Dr. Christine Julien, for providing me guidance, advice, and encouragement as I worked on this project. I would also like to thank my friends for helping me test the application and suggesting improvements. I would also like to thank my family for their continued support.

Abstract

SportMingles Sports Social Network for iOS

Frank Mintzu Huang, MSE

The University of Texas at Austin, 2012

Supervisor: Adnan Aziz

We live in a post-PC era with mobile devices on the rise. We can witness a growing number of applications that utilize the mobility of such devices to create applications that make our lives more efficient and connected. In the United States alone, millions of Americans play sports each year starting from middle school through college. With social media on the rise, there is a need for an application that takes advantage of the technologies incorporated in mobile devices with the ability to connect athletes in an area to one another. There is currently a lack of a social network application in the sports category of the Apple App Store. Therefore, this report describes how an iOS mobile application for a sports social network is designed, developed, and deployed. Specifically, we will explore in depth the iOS framework, user stories and requirements, the design and development of the client and backend application, and the tests and results.

Table of Contents

List of Figures

CHAPTER 1 INTRODUCTION

1.1 Vision for a Sports Social Network

Traditionally, personal computers have been the primary computing device used for our processing needs. Personal computers were either laptops or desktops that lacked mobility and connectivity. With the relatively low cost of smartphones and the rise of Internet infrastructure, we begin to witness a post PC era in which mobile devices such as smartphones and tablets are outselling laptops and desktops. According to TechCrunch 2012 [1], PC growth was at 15% for the 2011 year while mobile devices grew at a staggering 63%. Mobile devices often contain a GPS device that allows applications to become location aware. With the GPS, mobile devices have an advantage over traditional personal computers. Leading the charge for mobile devices is Apple. In 2012, Apple announced at the WWDC that Apple's app store had over 650,000 apps. Apple mobile devices use the iOS operating system. It is derived from Apple's OS X operating systems which traditionally run on Apple laptops and desktops. iOS is currently in its sixth revision with numerous features available to application developers such as push notifications, maps, location awareness, networking, graphics, in-app purchases, social media integrations, and camera. Furthermore, Apple now lets developers store application specific data to iCloud which allows users to access the same data across iOS devices. With over 30 billion app downloads [11] from Apple devices, Apple has placed itself as the development platform to develop for.

1.1.1 User Stories and Application

SportMingles is a sports social network. It provides a way for athletes around the world to connect with each other using their mobile devices. Its main features include:

- ➢ Team managers have the ability to track player game status for games and request substitute players for games that are short on players.
- ➢ An interactive search for games and teams looking for players in the user's area.
- ➢ Ability to create leagues, teams, and games for a specific sport with additional filtering capabilities such as gender, skill, and age.
- ➢ League managers have the ability to add games for teams that belong to the league.

We will now explore some user stories for SportMingles that will help define requirements for the application as well as design decisions. The use cases will also aid in conveying the vision for SportMingles.

User Story 1: James is a team manager of several indoor soccer teams. During his busy work schedule, he also has to send out game reminders to ensure games have enough players else risk forfeiture. Occasionally James may forget to remind players of a game or he may be out for vacation. Currently, James uses a Google Document spreadsheet to manage his teams. The process he uses is manual, tedious, and error prone. James decides to download SportMingles from the Apple App Store. With SportMingles, James creates his games ahead of time and is able to find substitutes with the app to prevent forfeitures.

User Story 2: Bob enjoys playing softball on the weekends with his coworkers. The team has elected him as team captain to manage their games. Bob is not very tech savvy so he resorts to using email. However, this creates long email chains that are difficult to keep track of in people's inbox. Instead, Bob uses SportMingles to create games. These games get pushed to a central data repository that all players on the team are able to see. The central data repository provides a consistent, persistent state for all players who use the application. Additionally, the central data repository allows for future development of applications on different platforms to pull data from a central location. This is beneficial because applications running on different platforms can share data. Furthermore, as Bob creates games for his teams, players on the team are sent a push notification to remind them of the game and to update their status. Bob now has a mobile way of tracking his games and an automated way of notifying his players of upcoming games.

User Story 3: Ryan recently started a new job in a new city. He loves playing volleyball but does not know anyone in the city. Ryan uses SportMingles' search feature to find volleyball games in his area. Ryan has been playing volleyball all his life. To ensure he finds games suitable to his skills, Ryan uses the filter feature to specify his age group, skill, and gender to find the right games. Furthermore, Ryan likes to play Thursday nights, so he creates a pick-up game so other players like himself can join him in a game of volleyball. SportMingles has helped Ryan continue playing his favorite sport.

1.1.2 Goals & Requirements

Using the user stories above, we are able to deduce some goals and requirements for SportMingles. With these goals and requirements, we can begin to explore the design of the system from the mobile application to the web application and finally to the database layer.

1.1.2.1 Goal

SportMingles' goal is to allow athletes around the world to connect with each other to play their favorite games and to allow team managers and players to keep track of their games and teams.

1.1.2.2 User Creation

Today's popular social media websites have provided ways for third party applications to use their system to verify user credentials. The technology driving this feature is called OAuth. SportMingles will provide users with the ease of logging into the system using their Facebook credentials. Alternatively, a user may create a SportMingles account if they choose not to use OAuth.

It is inherent for a sports team to desire to be connected with each other. With SportMingles, a user can use OAuth to retrieve his friends list from Facebook after authorization. Furthermore, this list will be used to determine whether or not the user is a member of SportMingles. If the friend is already a member, then the user will have the option of "following" the player for easy access to inviting the friend to games and teams. If the friend is not a member of SportMingles, then the user can invite the player by sending them an email.

Additional social connectivity is provided by allowing the user to search for a friend's name and then having the option to "follow" the player. A user may also invite players to SportMingles by sending them an email or text message from their contact list. In addition to the benefit of being able to add friends quickly to games, the team manager has the ability to invite friends that he is following quickly to the team.

1.1.2.3 League, Team, Game Creation

A user should be able to create leagues, teams, and games. Leagues, teams, and game creations will require different sets of input. When creating a league, a user must specify a league name, sport, skill, age group, and gender. When creating a team a user must specify a team name, privacy, sport, location, age group, gender, and skill. There are two types of games a user can create, a game for a team or a pick-up game that is not associated with a team. When creating a game for a team, the user must be the team manager of the team. He will then select the date and time for the game and which team the game belongs to. The team manager can then choose whether to use the team's default location as the game location or he can specify a different location. This provides the flexibility for a team manager who may have games at only a single location, such as a public park, to not have to specify the location every time, but also supports those team managers that have games at different locations. A pick-up game is a one-off game that is not associated with a team. It requires a date and time, location, sport, privacy, age group, gender, and skill. When a team manager creates a game for a team, a notification will be sent out to all the players on the team about the new game. The notification informs the player by a sound, badge, or banner. This notification prompts the user to tap on the notifications button within the app for further details about the notification.

1.1.2.4 Game and Team Listings

A user should be able to view all his games and teams in a quick and efficient manner. The teams listing will show teams the user is a member of and teams that are pending acceptance from team managers. The games listing screen will show past, upcoming, and pending games from all the teams the user is a member of as well as pick-up games. This screen shows games sorted relative to time and date. It also shows the team name, their status, and which sport the team is for.

1.1.2.5 Search

As a sports social network, SportMingles allows users of similar sports interest to connect with each other. It has a search functionality that uses the mobile device's GPS coordinates to search for games and teams looking for players. A user is able to specify an age group, gender, and desired sport to find games and teams in his or her area looking for players.

Next we will explain the foundation of the iOS infrastructure. Following this, we will explore some of the more relevant design patterns inherent to iOS. These patterns are important foundational pieces in understanding how to develop iOS applications. Once the groundwork for the iOS platform is discussed, we will analyze the development of the client application and then the backend application. Concluding the report will be the process of testing and deployment to the Apple App Store.

CHAPTER 2 FOUNDATION

2.1 iOS Infrastructure

Now that we have explored the motivation behind SportMingles, the use cases, and the goal and requirements of the application, we can begin understanding the foundation frameworks, design process and development of the mobile application.

2.1.1 INFRASTRUCTURE

SportMingles is developed on top of iOS. iOS is an operating system that helps manage the hardware for Apple mobile devices while providing services for application developers to use to build native applications. Native applications are developed using a language called Objective C. The integrated development environment provided by Apple for developing software is called XCode. The iOS architecture contains four layers: Cocoa Touch, Media, Core Services, and Core OS.

2.1.1.1 Cocoa Touch

The Cocoa Touch layer supports many common applications for iOS such as: document support, multitasking, notifications, gestures, standard view controllers.

The Cocoa Touch provides document support which allows the application developer to manage user documents on the local store as well as data in iCloud. It supports asynchronous reading and writing of files, which prevents blocking of the user from interacting with the interface, detecting conflicts between local and iCloud storage, as well as safe saving of document data.

Multitasking was introduced to iOS for application developers in version 4 and later. Before iOS 4, applications were suspended when entered into a background state to preserve battery life, but applications are now allowed to support three different types of multitasking tasks, including: using a finite amount of time to complete a critical task, informing the operating system that it needs to run special services that require background executing time, or generating local notifications.

There are two types of notifications available for application developers: local notifications and remote notifications. Local notifications allow the application to inform the user with a badge, alert and/or sound locally without the need of having a remote server. Remote notifications are a bit more involved but allow notifications on a user's device to be triggered remotely by a server. SportMingles uses remote notifications to inform users of new games and accepted invites to teams. More technical details will be discussed in the notifications design and implementation section.

The gestures library of Cocoa Touch allows the application view to register for touch gestures such as taps, swipes, pinching, panning, rotating, and long presses. This allows for an interactive experience for the user such as tapping on a game to view more details, pressing on a map to drop a pin for game location, or swiping across a game to delete the game. The Cocoa Touch layer also provides standard view controllers such as taking a picture or video, composing a text message or email, calendar events, and contact information.

SportMingles uses several standard view controllers provided by the Cocoa Touch layer such as when changing the profile picture on the user's dashboard, SportMingles uses the standard camera interface. Also when inviting friends via email or text message from the contact list, SportMingles uses the standard view controllers to maintain consistency with other native iOS applications.

2.1.1.2 Media Layer

The media layer of iOS provides several multimedia controls for graphics, audio, and video. Aside from simple graphical animations, the media layer also provides more powerful graphical services like Core Graphics, OpenGL, and Core Animation. Furthermore, it provides automatic upscaling of images for higher resolution screens such as devices using retina technology. AirPlay is another feature supported by the media layer that enables devices to mirror their view onto a TV using Apple TV.

2.1.1.3 Core Services

Core Services provides application developers with features such as:

- iCloud
- Automatic reference counting
- Block objects
- In-app purchases.

With iCloud storage, users can save documents to an online storage that is synced to any number of devices using iCloud. There are two types of storage capabilities: document storage or key-value data storage. Document storage enables application developers to store entire documents to iCloud. A key-value data store only stores parts of a document for sharing between instances of your application.

Automatic reference counting provides memory management at compile time. It evaluates the application variables at compile time and then inserts the necessary retain and release code in the appropriate locations. This frees the developer from having to manage memory and reduces the chance of memory leakage.

Block objects are essentially anonymous functions that can be used in place of callback functions or delegate functions. It is a C programming feature that was introduced to iOS in version 4.0. In SportMingles, we use block level programming to handle callbacks for graphical animations.

In iOS version 3.0, in-app purchase was introduced as another way for application developers to generate revenue for their applications. There are several types of in-app purchases including: auto renewing annual, non-renewing annual, one-time purchases with no expiration, and one-time purchases. SportMingles uses the one-time purchase with no expiration in-app store purchases. For the one-time purchase with no expiration, the user can join an unlimited number of games from the search functionality and can create an unlimited number of leagues.

2.1.1.4 Core OS

The lowest layer of iOS architecture is the Core OS. This layer provides many communication, security, and third party access. This layer provides services for application developers to communicate using the built in Bluetooth technology. Furthermore, it allows for communication with third party accessories that connect to the 30-pin connector dock. In addition to communication functionality, the Core OS layer provides security features to ensure the application data is secure. Shared keychain functionality was introduced in iOS 3.0 in which shared usernames and passwords can be used among applications from the same developer. SportMingles uses the keychain security feature of Core OS to safely store the user's username and password.

2.1.2 iOS Design Patterns

Now that we have explored at a high level some of the features each layer of the iOS architecture provides, we can begin to dive into some design patterns used in Objective C. These patterns are used throughout many of the frameworks provided by iOS. Understanding how these patterns work is important to understanding how to properly use the frameworks. Design patterns help application developers create reusable code that can be easily extended in the future. Design patterns are templates that aid in solving common, recurring problems.

2.1.2.1 Model-View-Controller

The model-view-controller design pattern is a popular pattern in software development. It separates the model, view, and controller into distinct roles that can quickly adapt to changes. "A model object encapsulates the data of an app and defines the logic and computation that manipulate and process that data [3]." In SportMingles our model objects include the game, team, league, and player. The SportMingles models contain only pure data and logic with no connections to presentation or controller logic. Therefore, they can easily be reused for other iOS devices such as an iPad application. The view object is in charge of rendering the presentation layer of the user interface and may handle user interactions. In SportMingles, each screen that the user interacts with is a `UIView` object that takes the model provided by the controller to populate information onto the screen. The controller object is between the model and the view object. It supplies the view with the model data and updates the model when users interact with the view. It also can perform logic for setting up the application and managing the lifecycle. In SportMingles each screen has a view controller that is a medium for coordinating the model data and views for the screen. As users interact with the view, the data gets updated in the model and sent off to the database for persistence.

2.1.2.2 Delegation

Delegation occurs by inserting a desired behavior into a class object from another class object, usually a framework [3]. "A framework is a real or conceptual structure intended to serve as a support or guide for the building of something that expands the structure into something useful [9]." When an event is about to occur, the framework class object sends a message to the delegate object informing them of the action. For example, a view controller can be a delegate for a table view. As the user scrolls, the Cocoa Touch framework can invoke a method on the delegate informing it that the scrolling has ended. This becomes helpful because the view controller will not need to subclass any framework class. In order for delegation to work, the class must hold a property that is a reference to the delegate object and the class must conform to a protocol, which will be discussed in the next section. SportMingles uses several delegation design patterns for its table views, action sheets, Facebook interactions, text fields, map views, etc.

2.1.2.3 Protocol

A protocol is analogous to interfaces in Java from other object-oriented languages. It defines a list of methods that any class can implement. Methods can either be required or optional. Protocols and delegates are coupled together. When a class becomes a delegate for an object, it must explicitly state the protocol in the Objective C header. This means that the class will need to implement any required methods stated in the protocol and can optionally implement any optional methods from the protocol for additional method behavior. The class will also need to declare itself as the delegate so that the remote class can call the appropriate methods when needed. An example of delegation and protocol design pattern used in SportMingles application is when a user searches for games or teams near his or her area, we explicitly state we are using the `CLLocationManagerDelegate` in the header file and we implement the method,

```
- (void)locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation
fromLocation:(CLLocation *)oldLocation
```

, in our implementation and declare the class as a delegate for `CLLocationManager`. When the user's GPS location receives coordinates, the iOS CoreLocation framework calls this method which provides us with coordinates for the application to use.

2.1.2.4 Notifications

Notifications in iOS are similar to the observer design pattern. Objects can register for notifications that will be broadcasted out to all observers by the system. Notifications can be about anything that is going on in the system that has happened or is about to happen. "Notifications broadcast by the notification center is a way to increase cooperation and cohesion among the objects of an app. [3]" Notifications are similar to delegation except they never return any values. SportMingles uses notifications in its in-app purchase store. Whenever a purchase is sent to Apple server's the application store awaits a response from the Apple server. When a response is received, a notification gets broadcasted and the view gets updated with the response from the Apple server.

2.1.2.5 Target-Action

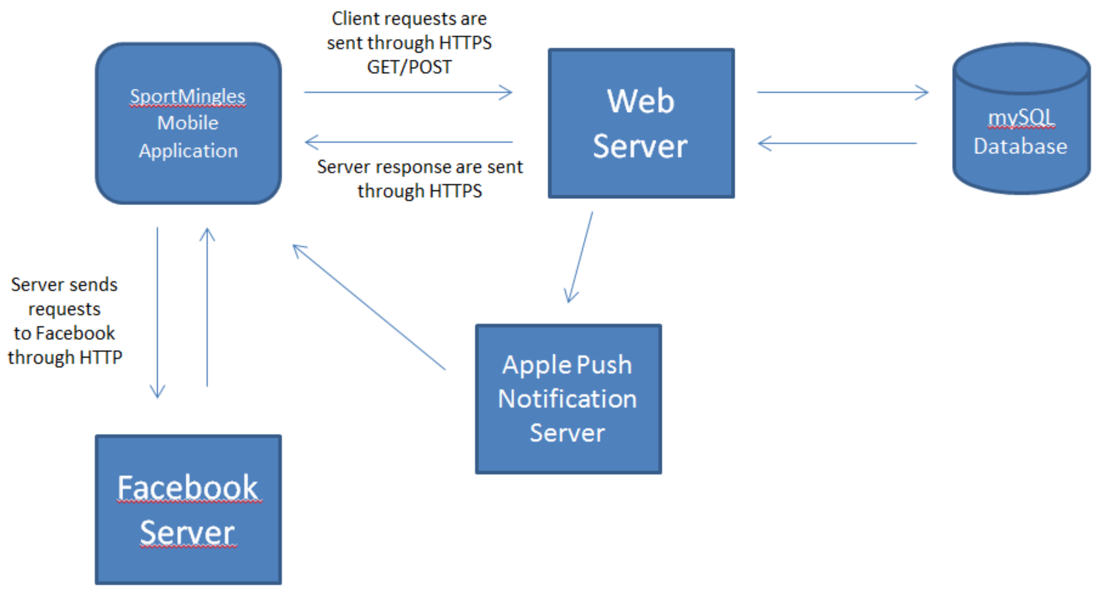
The target-action design pattern is often used for gesture recognition. A target-action consists of a message to send and the target object to receive the message. When the target event occurs, the message is sent to the method of the target object. An example use case of target-action design pattern is the use of `UIButtons` in SportMingles. An action could be creating a game and the target could be a game view controller. When the user presses the create button, the method would get called and the information supplied by the user would be validated and submitted to the backend for processing.

CHAPTER 3 DEVELOPMENT

3.1 Design & Implementation of Client Application

We have now reviewed the iOS architecture and its related design patterns which provided a foundation for understanding the design and implementation of SportMingles. In the first section we will explore the mobile application side including login and user creation, league, team, and game creation, notifications, in-app store purchases, and search. After this we will explore how the web application side works by exploring the technologies that handle the server requests. Then we will take a look at what drives the persistency of the user data by diving into the database layer of SportMingles.

3.1.1 Architecture



Client requests are sent through HTTPS GET/POST
SportMingles Mobile Application
Web Server
mySQL Database
Server response are sent through HTTPS
Server sends requests to Facebook through HTTP
Apple Push Notification Server
Facebook Server

Figure 1. SportMingles Architecture of Entire Ecosystem

In Figure 1, we have depicted the architecture of SportMingles. It consists of five components: SportMingles Mobile Application, Web Server, MySQL Database, Facebook Server, and Apple Push Notification Server. The client interacts with the

SportMingles mobile application, which is built using Objective C on the iOS platform. The web server uses PHP to handle any requests from the mobile application. Requests are sent and responded to using SSL for security. Data is stored in a MySQL database that resides on the same server as the web server. Notifications are sent to the Apple Push Notification Server through HTTPS. The Apple Push Notification Server handles the sending of notifications to the user's device. Login can be achieved using either the Web Server or Facebook Server. Requests are sent to the Facebook server using HTTPS. The following sections describe in detail the components and their interaction.

3.1.2 Mobile Application

3.1.2.1 Login/User Creation

A user can log into SportMingles by using a SportMingles account or a Facebook account. To log into SportMingles with a Facebook account, we use OAuth to authenticate the user and to request additional data stored on Facebook. OAuth is an open authentication protocol used by many corporations such as Facebook, Twitter, and Google. The user is presented with a button called "Login with Facebook." By tapping on this button from the SportMingles application, the user is then redirected to Facebook where he or she can login and authorize SportMingles to have access to data. A client ID and token is sent to the Facebook server for validation. The ID and token were received during the registration process of SportMingles. Once the user has successfully logged in, the user is sent back to the SportMingles mobile application along with an access token and expiration date. The access token can then be used to access the user's Facebook graph for the user's information such as profile picture, friend list, and name. The Facebook server is referred to as the Resource Server and Authorization Server. The Resource Owner is the user using the Client Application. Alternatively, the user may login using the web server. The user email and password are sent to the web server using HTTPS. The web server then validates the user email and password and responds to the request using JSON, formatting of either success or failure.

The user can create an account by logging in with Facebook or registering for an account with SportMingles. If the user logs into SportMingles with Facebook, the account is automatically created for them by using information from Facebook's OpenGraph which responds with a JSON object. The user's name, profile picture, and Facebook ID are then stored onto the Web Server. If the user chooses to register for an account with SportMingles, then the first name, last name, email, and password are sent to the Web Server for processing and a JSON response of success or failure will be received.

3.1.2.2 League, Team, Game Creation

One of the main functionalities of SportMingles is the ability to create leagues, teams, and games. We will start by discussing how a user can create a league. Any user can create a league after they have purchased a single membership of ninety-nine cents through the in-app purchase. Once a user has successfully purchased the item, the user specifies a league name, privacy, sport, skill level, gender, age, and location. This information is then submitted to the web server using HTTPS and stored in the MySQL database. If the data is successfully stored in the MySQL database, then a success JSON response is sent, otherwise an error JSON response is sent. JSON stands for JavaScript Object Notation. JSON is used instead of XML because of its lightweight file structure when compared to XML. When a user has created a league, he can then either search for teams in his area that meet the league's criteria or teams the user has created. If the user chooses to search for teams in his area, the `CLLocationManager` provides the user's

GPS coordinates. These coordinates are then sent to the web server. The web server then searches through all the teams that match the league criteria and within a one-hundred mile radius of the GPS coordinate. A one-hundred mile radius is used to ensure a sufficient amount of results will be shown. Each item is displayed by distance from the user's current location. In the future, the application can support a feature that allows the user to specify the number of miles for the search radius. The response is sent back to the client application through a JSON response. If the user chooses to add a team he created to the league, then the server responds with a list of teams encoded in a JSON response. The league manager can create games for teams in the league. This allows for league managers to setup game seasons, tournaments, or one-off games for teams.

Teams are created similarly to the leagues. The user specifies the team name, privacy level, sport, location, age group, gender, and skill. The privacy level for a team can be public, protected, or private. A public team means anyone can join without the team manager's approval and will be shown in search results. A protected team requires the team manager's approval and will be shown in the search results. A private team means players can only join the team if the manager invites them. Additionally, the team is not listed in the search results. The information is then sent to the web server via HTTPS. If the team is successfully created, then the status of the JSON response will be "success" otherwise it will return "error".

When a game gets created, the user can view game details that show them the game date and day, time, the game attributes such as sport type, gender, skill, and age, the user's status, and a description. The user is also presented with the roster of players for the game along with their status. From here, if the user created the game, tapping on the settings button brings a `UIActionSheet` that lets the user delete the game, set their game status, edit game details, request subs, and invite players. When a user requests subs, he can specify the number of substitutes needed. This request is posted to the web server using HTTPS. Using the GPS location supplied by `CLLocationManager` of the user's current location and the coordinates of the game location, users are able to search for games in need of players. Alternatively, the user can invite players from the friends list that they are following, search for a player name, or a contact list. If a user searches by name, the name is sent to the web server for processing and then returned through a JSON response to the client application.

A team manager has the options of adding a game to the team, deleting the team, and inviting players. When a team manger creates a game for the team, notifications are sent to all accepted players on the team using Apple's push notification system. We will take a look at how Apple's push notification works in more detail in the next section.

3.1.2.3 Notifications

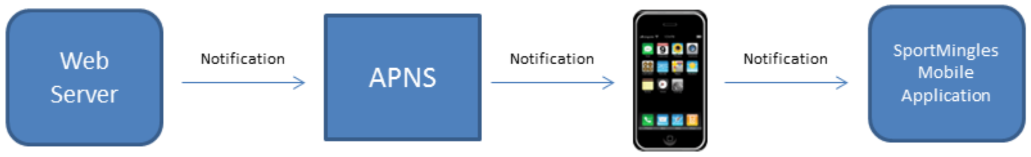


Figure 2. SportMingles APNS

Apple Push Notification Service is a feature Apple provides to developers to notify Apple devices. Devices use a persistent, secure, IP connection to Apple's push notification server. The application developer uses persistent, SSL connection to Apple's push notification server with a certificate for identification [4]. The developer provides a JSON payload object to the open connection's stream. The payload can include an alert message, a badge number, and a sound to play. Per Apple's push notifications specifications, the max size of the payload is 256 bytes. This max size is set as the packet size in the backend. Along with this payload is also the device's token, which is used as an identifier for Apple to know which device to send the notification to. The device token is decrypted by Apple using the token key to extract the device id of the device. The device token is retrieved when a user first launches the application and is prompted to either allow or cancel push notification server for the application. If the user allows the application to use push notification, then the method,

```
-(void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken
```

, is called in the AppDelegate of the application. Here the developer can retrieve several device attributes including: alert type, sound type, badge type, device token, app name, app version, device model, and device version. This information is then sent to the web server over HTTPS for storage in the MySQL database.

When a user turns off notification for SportMingles or deletes the application, the web server needs to know to stop sending notifications to the user. Apple provides a feedback service that keeps track of user devices that have failed to receive notifications per application. This information can be retrieved by the web server by connecting to feedback.push.apple.com on port 2196. Analogous to sending a notification, connecting to the feedback service involves an SSL connection and certificate for identification. After a connection has been established, Apple sends back to the web server a stream of device tokens. This feedback service is handled by Apple's feedback service because there is currently no way of informing the developer of the app that the user has uninstalled the application. Without a way of knowing if the user has uninstalled the application, the developer will still continually deliver push notifications to the client until the application has processed the feedback service response.

A permanent, wireless, mobile connectivity cannot be guaranteed for application developers. Thus, when applications send push notifications and the client loses connectivity, then the notification can be lost. Therefore, Apple provides a default quality of service for applications. It uses a store-and-forward approach in which if a notification fails to send to a device because it is turned off, Apple will store the notification for a limited time before retrying to send the notification. A limit of one notification record is stored in the database tables to prevent user applications from receiving multiple notifications per push session.

With device tokens being exposed to developers, privacy can be a concern for users as notifications can be abused by malicious third party vendors. Therefore, security is of utmost importance to Apple's push notification service. It uses two trust services to ensure notifications are sent to the correct device and that notifications are sent to Apple's server with integrity. A connection trust and a token trust are used to ensure this security. A connection trust ensures that only authorized providers are allowed to send notifications to Apple servers. A token trust is based on the device token associated with each device that is shared with the provider and Apple. To establish a trust between the device and Apple's servers, TLS peer-to-peer authentication is used. TLS stands for transport layer security, which provides cryptographic connections for internet services. The device starts by initiating a connection to APNS which sends back a server certificate. This server certificate is validated on the device, and a device certificate is sent to APNS for validation. If validation is passed, a TLS connection is established. After the device has established a secure connection, it can register with APNS to receive push notifications. "APNs generates a device token using information contained in the unique device certificate [4]". The device token is further encrypted with a token key and gets sent to the device which subsequently gets sent to the application and then the web server.

SportMingles uses the nohup POSIX command which allows for processes to be executed in the background. The PHP file sleeps and wakes every three minutes to process any pending push notifications. It first creates a persistent connection to Apple's push notification servers. When the thread wakes up, it checks if the connection is still valid, if not it will reconnect. Next it queries the database and writes the notification JSON message onto the connection stream. Once the results have completed processing, the temporary table is emptied and the thread goes to sleep.

3.1.2.4 In-App Store Purchases

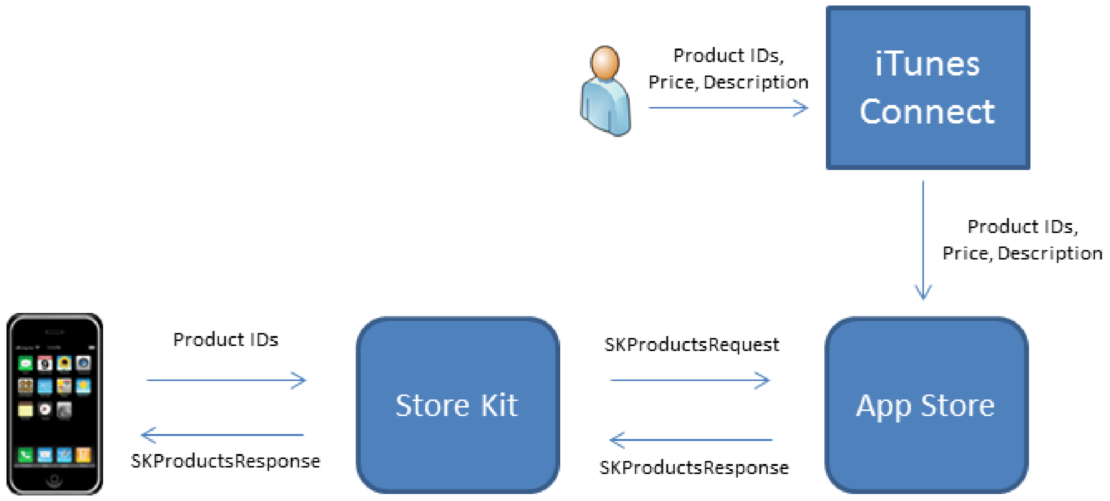


Figure 3. In-App Purchase Workflow

Part of the Core Services architecture layer is the Store Kit. This kit enables application developers to communicate with the App Store. The App Store is another means of generating revenue for the application. Developers can sell items through an in-app store to users. There are several types of in-app store types including:

consumable, non-consumable, auto-renewable subscriptions, free subscription, and non-renewing subscription.

SportMingles application uses non-consumable in-app store items. For non-consumable items, Apple requires applications to provide a restore button that allows users to restore previous purchases in case the application data get wiped out. When a user searches for games or teams to play in their area, the distance from their current location, date, time, and attributes are presented to the user, but not the specific location. To view where the game will be, users need to purchase a ninety-nine cents non-consumable in-app store item to unlock the location feature. A non-consumable item type is an item that only needs to be purchased once and will not expire. When a developer creates an in-app store item, he must specify a product ID that will be supplied to the Store Kit. This product ID will then be used to retrieve the item from Apple servers through the Store Kit to get the item attributes such as: name, description, and price. Store Kit does not provide a view for the application to utilize; SportMingles uses a `UITableView` to display the store items to the user. The `SKProductsRequestDelegate` is implemented to connect to Apple servers. When sending the list of product IDs to the Store Kit, the response is an array of `SKproductsResponse,` which contain the localized name, description, and price that the developer entered in the iTunes Connect interface. When the user is ready to purchase an in-app store item, the product ID is placed inside a `SKPayment` object that is then placed in a `SKPaymentQueue`. A persistent transaction is created to connect to Apple's servers for processing the payment transactions. SportMingles implements the method `paymentQueue:updatedTransactions` to listen for events from `StoreKit`. If the event receives a type of `SKPaymentTransactionStatePurchased` then the user has successfully purchased the item and we store the product ID in a globally accessible `NSMutableSet`. This set keeps track of items purchased by the user and will be used to determine when to show the unlocked feature.

3.1.2.5 Search

To connect players of all skill levels with each other, SportMingles features a "Find" function that allows players to find games and teams in their area to join. When the user first taps on the "Find" button in the main bottom navigation, we use the `CLLocationManager` to find the user's GPS coordinates. The user can then choose the type of sport they would like to find as well as filter the search results by skill, age, and gender. This information is packaged in a HTTPS GET request and sent to the web server. The web server then responds with a JSON object of games and teams that it found that match the criteria. The JSON object is then parsed and displayed to the user. When the user taps on a game or team, the application first checks to see if the user has purchased the "Join" in-app store item. If the user has not purchased the item, the user is presented with a button to visit the in-app store. Otherwise, if the game is public, the user can view all the information about the game or team and directly click on the "Join Game/Team" button. If the game is protected/private, then the user is presented with a button to "Request to Join" which will need the team manager or game creator's approval before the map location is revealed. This type of process is implemented to give game and team creators a sense of controlling the privacy and the number of players for the game or team. The algorithm for determining games and teams nearby will be explored in detail in Chapter 4, Section 3.

3.1.2.6 Social Connections

Aside from finding games and teams to join using the "Find" functionality, players can also follow each other. The benefit of following other players is the ease of inviting them to games and teams. When a player creates a game or team, they can directly invite those that they are following to the team without the need of inviting through email or text messages.

With Facebook OAuth connections, SportMingles can pull the user's Facebook friends list and compare against the MySQL database for friends that are already members of SportMingles. This makes it easy for users to quickly follow friends that they are already friends with on Facebook. If the friend is not a SportMingles member, the user can tap on the "invite" button, which will use the `MFMailComposeViewControllerDelegate` to bring up Apple's native mail composing view to send an email to the friend with a predefined body. The user needs to fill out the friend's email and press send. This makes it easy for SportMingles users to invite their Facebook friends to join SportMingles.

3.2 Design & Implementation of Web Application

The SportMingles web server runs on a Virtual Private Server with an Intel(R) Xeon(R) CPU E5620 @ 2.40GHz. It uses Linux 2.6.18 as the operating system with 1 Gig of RAM. SportMingles uses Object Oriented PHP 5 as the web application language for processing requests and responses from the mobile application. We will now take a detailed look at the architecture for the web application and the core logics used to process the client requests.

3.2.1 Architecture

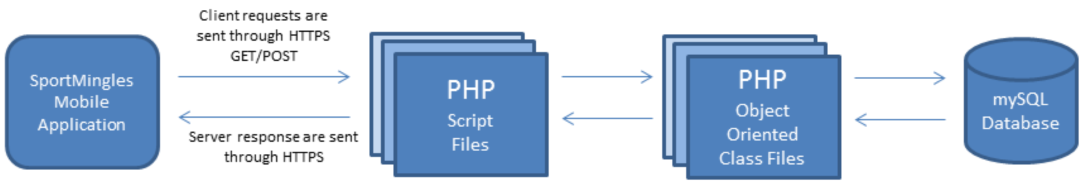

SportMingles Mobile Application
Client requests are sent through HTTPS GET/POST
Server response are sent through HTTPS
PHP Script Files
PHP Object Oriented Class Files
mySQL Database

Figure 4. Web Application Architecture

The SportMingles mobile application interacts with the web server using HTTPS GET/POST requests. The mobile application only interacts with a small subset of PHP script files that is responsible for parsing out the GET/POST variables, creating the appropriate PHP objects, calling the correct function, and returning the response to the mobile application. The PHP script files are unaware of the MySQL database layer. The PHP object oriented class files contain public and private functions that process the desired logic from the user interacting with the mobile application. The PHP object oriented class files interact with the MySQL database to SELECT, UPDATE, DELETE, and INSERT data. These files do not interact with the mobile application. In the following sections we will take a look at the class hierarchy and the core logic for processing user actions.

3.2.2 Class Hierarchy

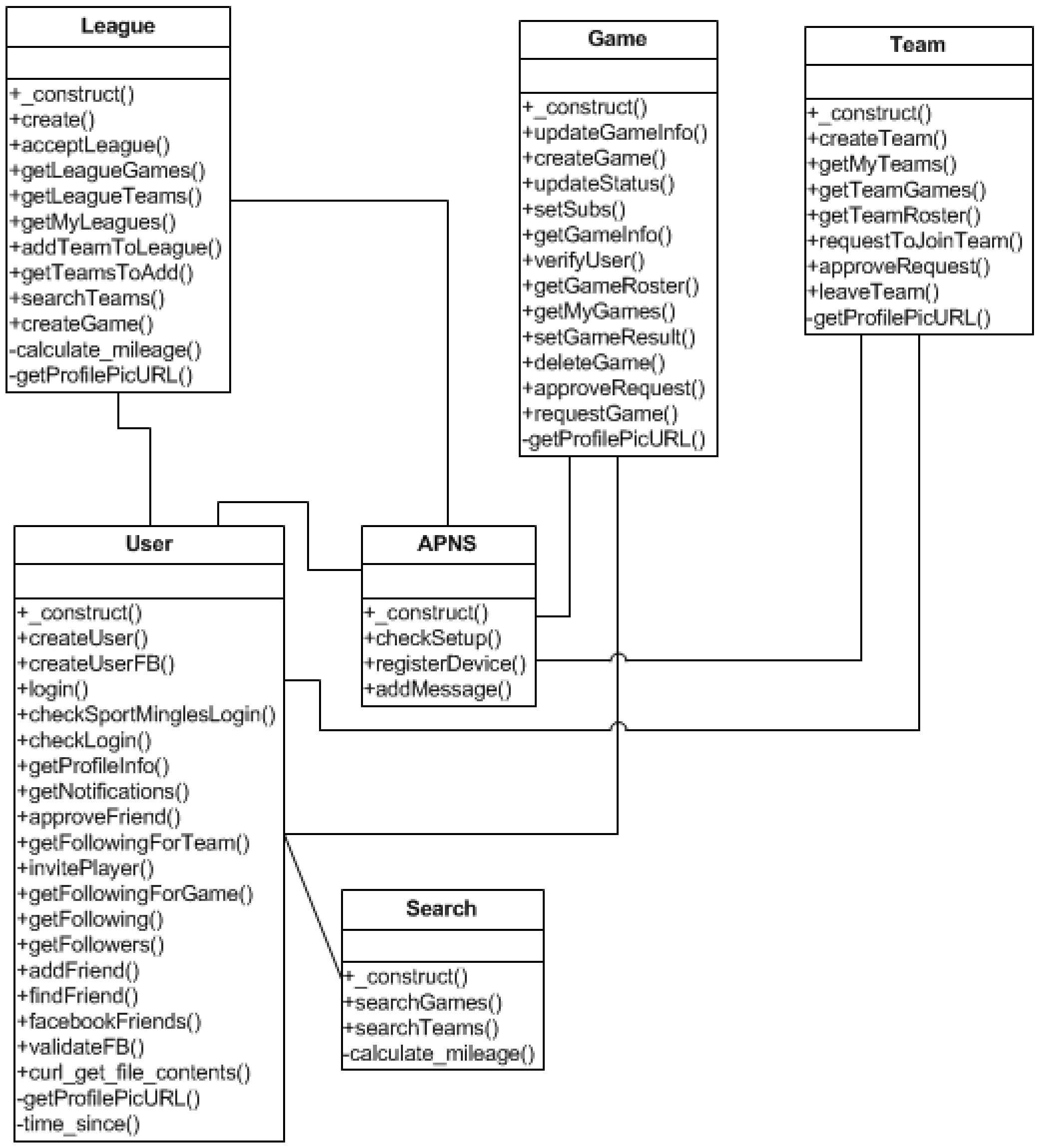


Figure 5. PHP Class Hierarchy

3.2.3 Implementation

The SportMingles web application consists of six PHP class files: league, team, game, user, APNS, and search. These six files, depicted in Figure 5, encompass all the logic needed to handle user interactions from the mobile application. A function with a + indicates that it is a public function. A function with a – indicates that it is a private function. They handle user creation and verification, database access, and searching. The following sections will describe in more detail the algorithms and functions used to complete the system implementation.

3.2.3.1 User Creation and Verification

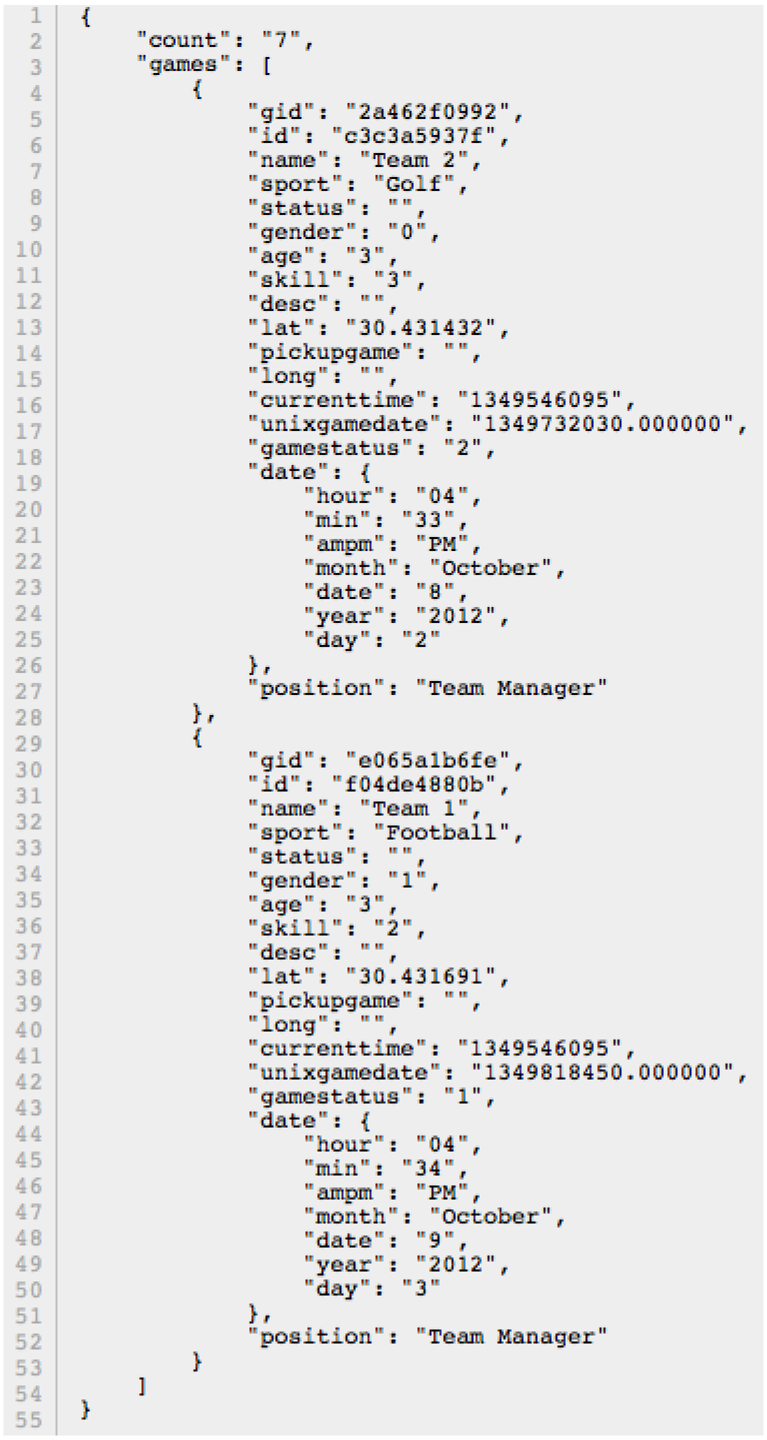
When a user signs up for a SportMingles account or logs in with Facebook, the first logical step is to create a user account in our database for this user. A unique id is generated for the user using PHP's `mt_rand()` function. This output is then passed into the `uniqid` function, which "gets a prefixed unique identifier based on the current time in microseconds [5]." These two functions are combined to ensure a unique id is generated for the user. If the user signs up for a SportMingles account, the password is passed into the md5 function, which is a cryptographic hash function. To increase security measures, we further append a salt value to the password before passing the value to MD5.

To verify user login or actions, each public function first checks the user's credentials before executing any logic. If the user logged in with a SportMingles account, then the password and salt are passed to the md5 function and verified if it matches the user's email. If it does, then the function can proceed, otherwise the function returns an error response. If the user logs in with Facebook, then the web server makes a request to Facebook's opengraph server with the access token as a parameter. If the response is an error, then the user will need to reauthorize the account. If the response is a success, then we verify the user's email address that is stored in MySQL with the response from Facebook to determine if the two accounts match.

3.2.3.2 Database Access

Each function performs retrieving, updating, deleting, and adding similarly among all the classes. The only real variance between these actions is the tables and parameters that are passed into the query. Database queries are constructed using PDO. PDO stands for PHP Data Object. It is a layer abstraction for database access. Application developers use the same functions independently of the type of database used. For SportMingles, the database we use is MySQL. PDO features prepared statements to be used for database access. Prepared statements have the added benefit of running efficiently. A prepared statement is similar to a compiled template that can have variables inserted into them dynamically. Rather than having to analyze, compile, and optimize the query every time, if an application executes the same query multiple times, the prepared statement will skip the analyze, compile, and optimize steps the second time. "This means that prepared statements use fewer resources and thus run faster [5]." Additionally, prepared statement variables are bound to parameters in the statement. The benefit of using this approach is that it circumvents any possibility of a SQL injection attack.

3.2.3.3 JSON



```
{
    "count": "7",
    "games": [
        {
            "gid": "2a462f0992",
            "id": "c3c3a5937f",
            "name": "Team 2",
            "sport": "Golf",
            "status": "",
            "gender": "0",
            "age": "3",
            "skill": "3",
            "desc": "",
            "lat": "30.431432",
            "pickupgame": "",
            "long": "",
            "currenttime": "1349546095",
            "unixgamedate": "1349732030.000000",
            "gamestatus": "2",
            "date": {
                "hour": "04",
                "min": "33",
                "ampm": "PM",
                "month": "October",
                "date": "8",
                "year": "2012",
                "day": "2"
            },
            "position": "Team Manager"
        },
        {
            "gid": "e065a1b6fe",
            "id": "f04de4880b",
            "name": "Team 1",
            "sport": "Football",
            "status": "",
            "gender": "1",
            "age": "3",
            "skill": "2",
            "desc": "",
            "lat": "30.431691",
            "pickupgame": "",
            "long": "",
            "currenttime": "1349546095",
            "unixgamedate": "1349818450.000000",
            "gamestatus": "1",
            "date": {
                "hour": "04",
                "min": "34",
                "ampm": "PM",
                "month": "October",
                "date": "9",
                "year": "2012",
                "day": "3"
            },
            "position": "Team Manager"
        }
    ]
}
```

Figure 6. Sample JSON Object

Communication between the SportMingles mobile application and the Web Server are transferred over HTTPS using JSON. Using the open source JSONKit [6] for iOS, the mobile client app is able to consume the JSON object returned by the server, parse the response, and convert the results into Objective C classes. An example of a JSON response from the web server can be seen in Figure 6. This JSON object returns all the needed information to populate the game listings screen.

3.2.3.4 Searching

When a user searches for teams or games in his or her area, the GPS coordinates of the user's location are passed into equations for determining the relative distance between two coordinates. These equations will produce the minimum and maximum longitude and latitude within one-hundred miles of the user's location. Using the minimum and maximum longitude and latitude coordinates, we can compare them against the longitude and latitude coordinates of games and team locations where the privacy level is either public, protected, or if the game is looking for substitute users. When the list is retrieved, we can calculate the distance between two coordinates by using the Haversine formula to calculate the great-circle distance between two points [10]. The games and teams are then compiled into a JSON list along with game date, time, team name, distance from the user, and game or team attributes.

3.2.3.5 MySQL Database

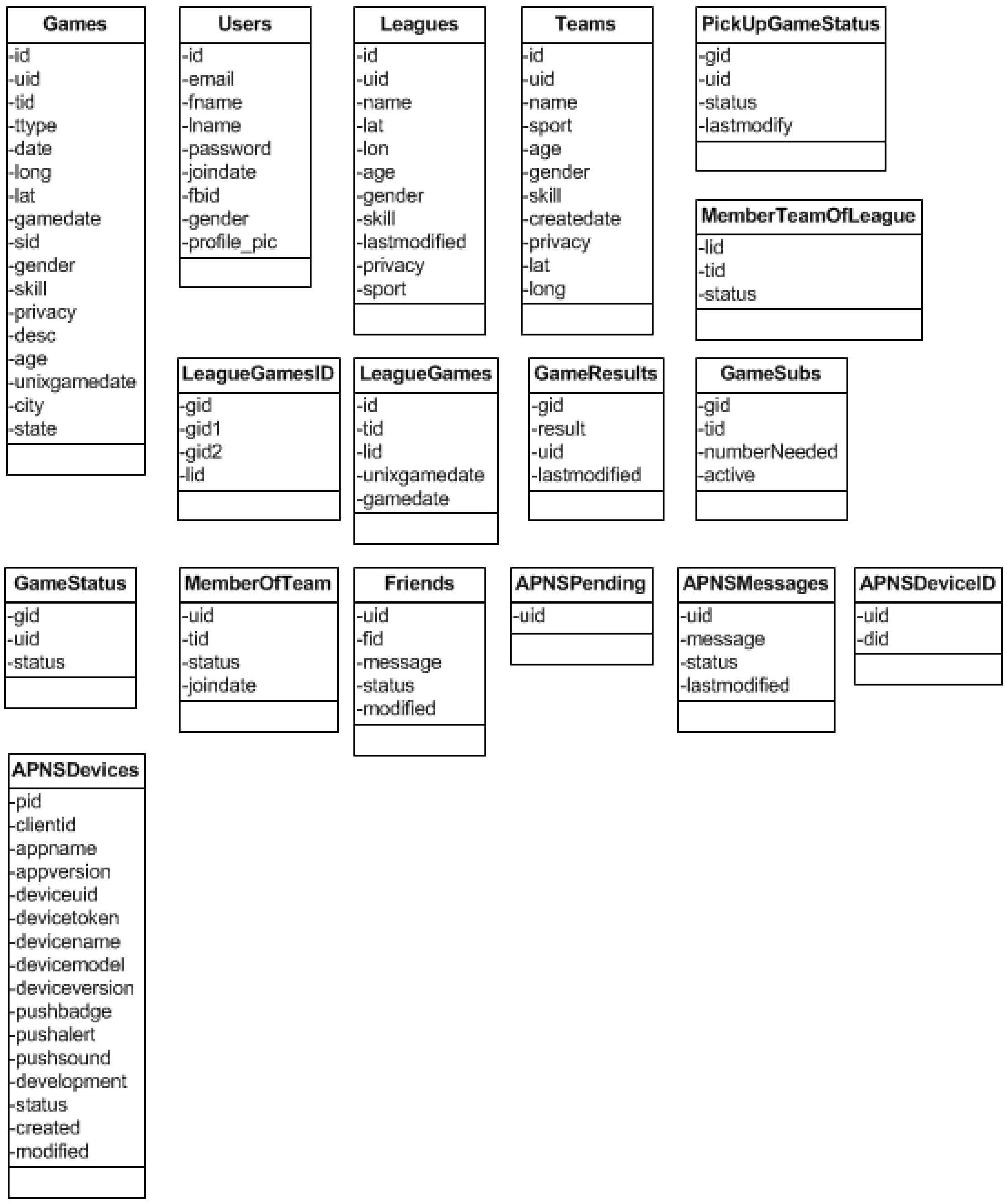


Figure 7. MySQL Database Tables

The database layer resides on the same server as the web server. Tables are created with indices for query optimizations. Database creation and management are handled through PhpMyAdmin. PhpMyAdmin provides a simple graphical user interface for the user to create tables, indices, and primary and foreign keys. Additionally, schemas can be imported and exported from different database platforms.

CHAPTER 4 TESTING RESULTS

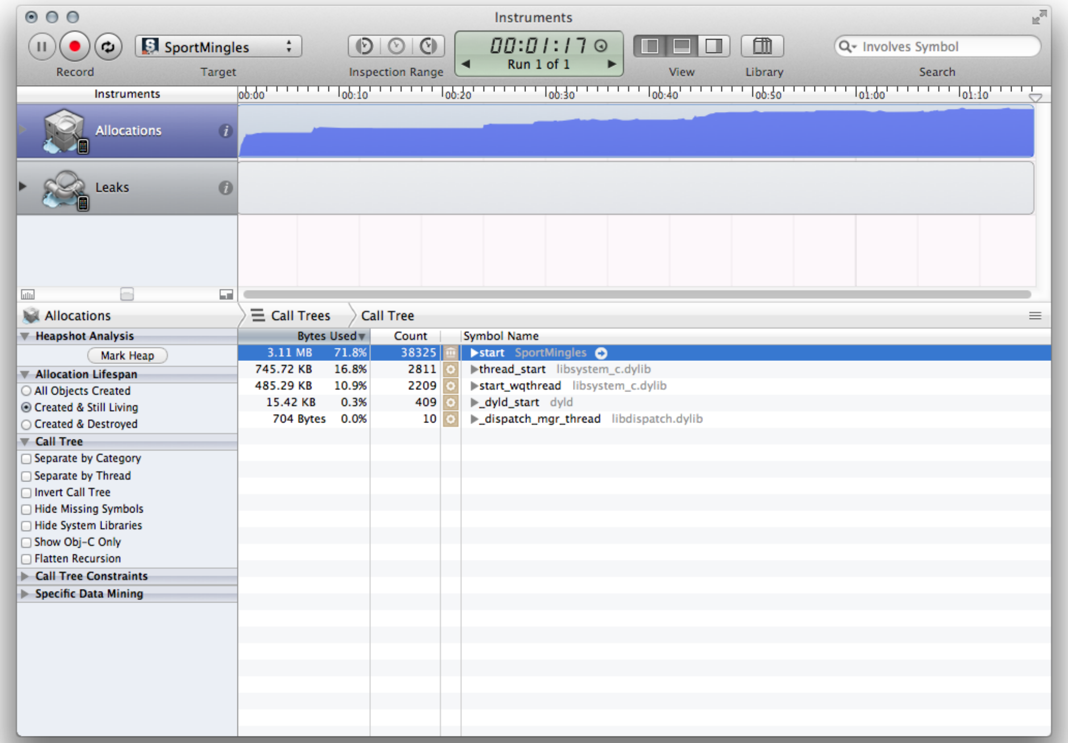
4.1 Testing

4.1.1 INSTRUMENTS

Instruments is a tool built into XCode that allows developers to profile their application while running on a mobile device. Developers analyze their applications for memory allocations, leaks, activity monitor, time profiler, system trace, automation, and energy diagnostics. To run Instruments, the user runs the application on their iOS device in Profile mode and selects which type of tests they would like to perform. Once a test is selected, real time streaming of data can be seen in Instruments. The developer then has the ability to save the data for further analysis. Instruments is an invaluable tool for developers to profile and analyze their application to ensure the best experience for its users.

Memory management in iOS has always been a pain point in developing iOS applications. When an object is allocated, the developer must remember to retain and release the object in order to prevent null pointer exceptions. In iOS 4 and beyond, Apple introduced the concept of Automatic Reference Counting [7]. ARC works at compile time by conceptually inserting the memory management retains and releases for the allocated objects. In the following sections we analyze memory leaks, energy usage, CPU activity, and network activity. We had Instruments record data while running through major functionalities of SportMingles.

4.1.1.1 Memory Leaks



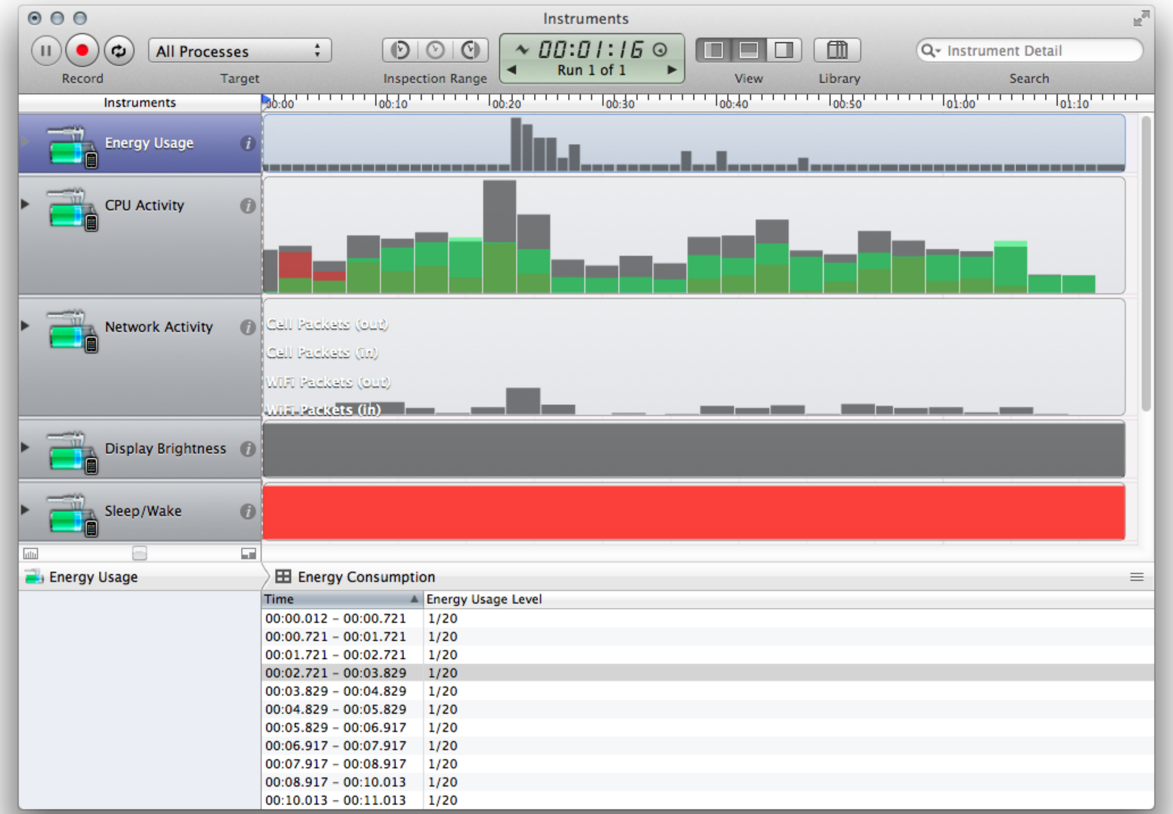
○ All Objects Created
● Created & Still Living
○ Created & Destroyed

☐ Separate by Category
☐ Separate by Thread
☐ Invert Call Tree
☐ Hide Missing Symbols
☐ Hide System Libraries
☐ Show Obj-C Only
☐ Flatten Recursion

Bytes Used		Count	Symbol Name
3.11 MB	71.8%	38325	▶start SportMingles
745.72 KB	16.8%	2811	▶thread_start libsystem_c.dylib
485.29 KB	10.9%	2209	▶start_wqthread libsystem_c.dylib
15.42 KB	0.3%	409	▶_dyld_start dyld
704 Bytes	0.0%	10	▶_dispatch_mgr_thread libdispatch.dylib

Figure 8. Memory Leaks with Instruments

Using instruments, we can run memory leak checks to determine if we are using ARC correctly. After running through the major functionalities of SportMingles for over a minute, we can see in Figure 8 there are no red bars in the Leaks section of the Instruments screen. Without running ARC and manually handling memory allocation, there were quite a number of leaks in the code which became stressful in debugging. Thus, with ARC, we can ensure the application does not randomly crash because of memory leaks while the user is interacting with the application.

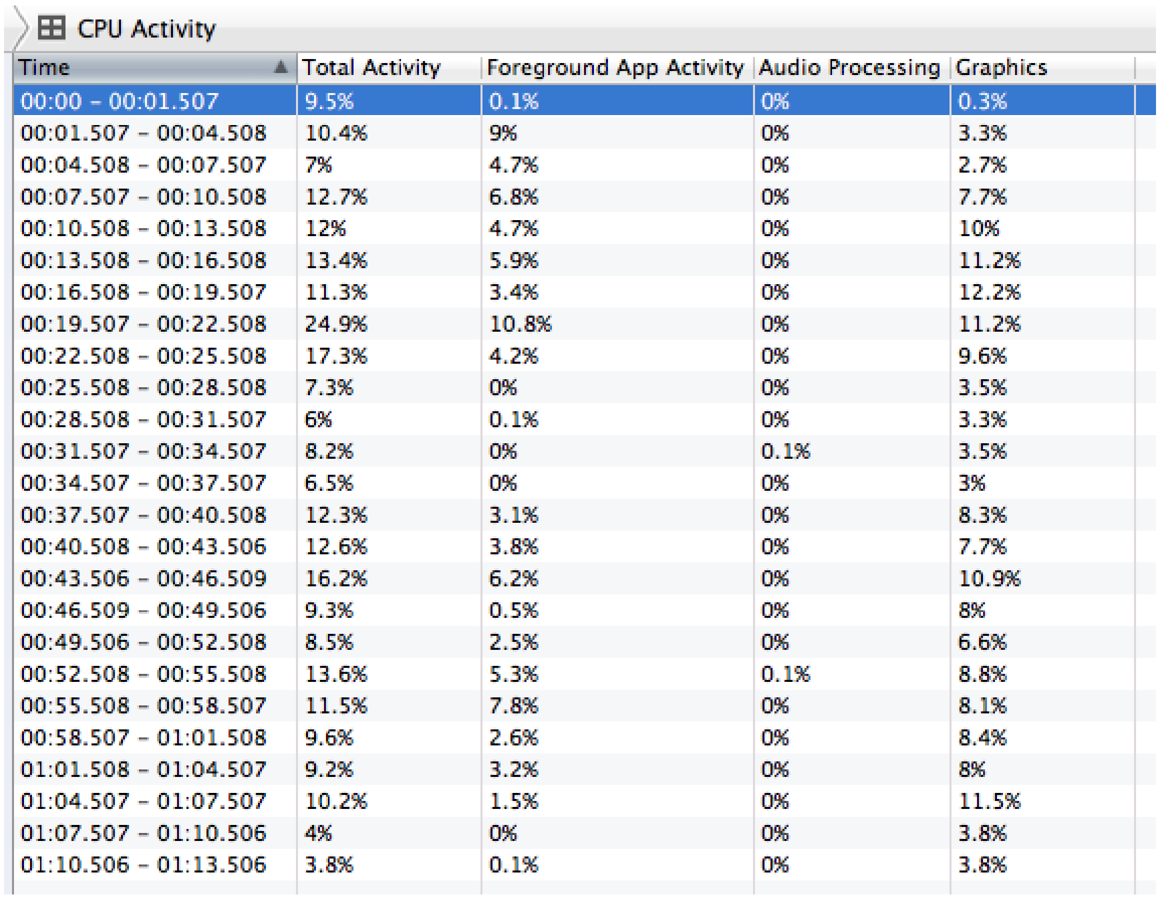


Time	Energy Usage Level
00:00.012 - 00:00.721	1/20
00:00.721 - 00:01.721	1/20
00:01.721 - 00:02.721	1/20
00:02.721 - 00:03.829	1/20
00:03.829 - 00:04.829	1/20
00:04.829 - 00:05.829	1/20
00:05.829 - 00:06.917	1/20
00:06.917 - 00:07.917	1/20
00:07.917 - 00:08.917	1/20
00:08.917 - 00:10.013	1/20
00:10.013 - 00:11.013	1/20

Figure 9. Energy Diagnostics with Instruments

4.1.1.2 Energy Diagnostics

Using the energy diagnostics while running the SportMingles application, we stepped through the major functionality of the application while recording energy usage, CPU activity, and network activity. As we can see in Figure 9, the energy usage while using SportMingles was relatively low throughout except for spikes when using GPS. However, we can see the energy level decreased once the GPS functionality was completed. The reason for this is because we developed the application to turn off the GPS usage once we have located the user's location.



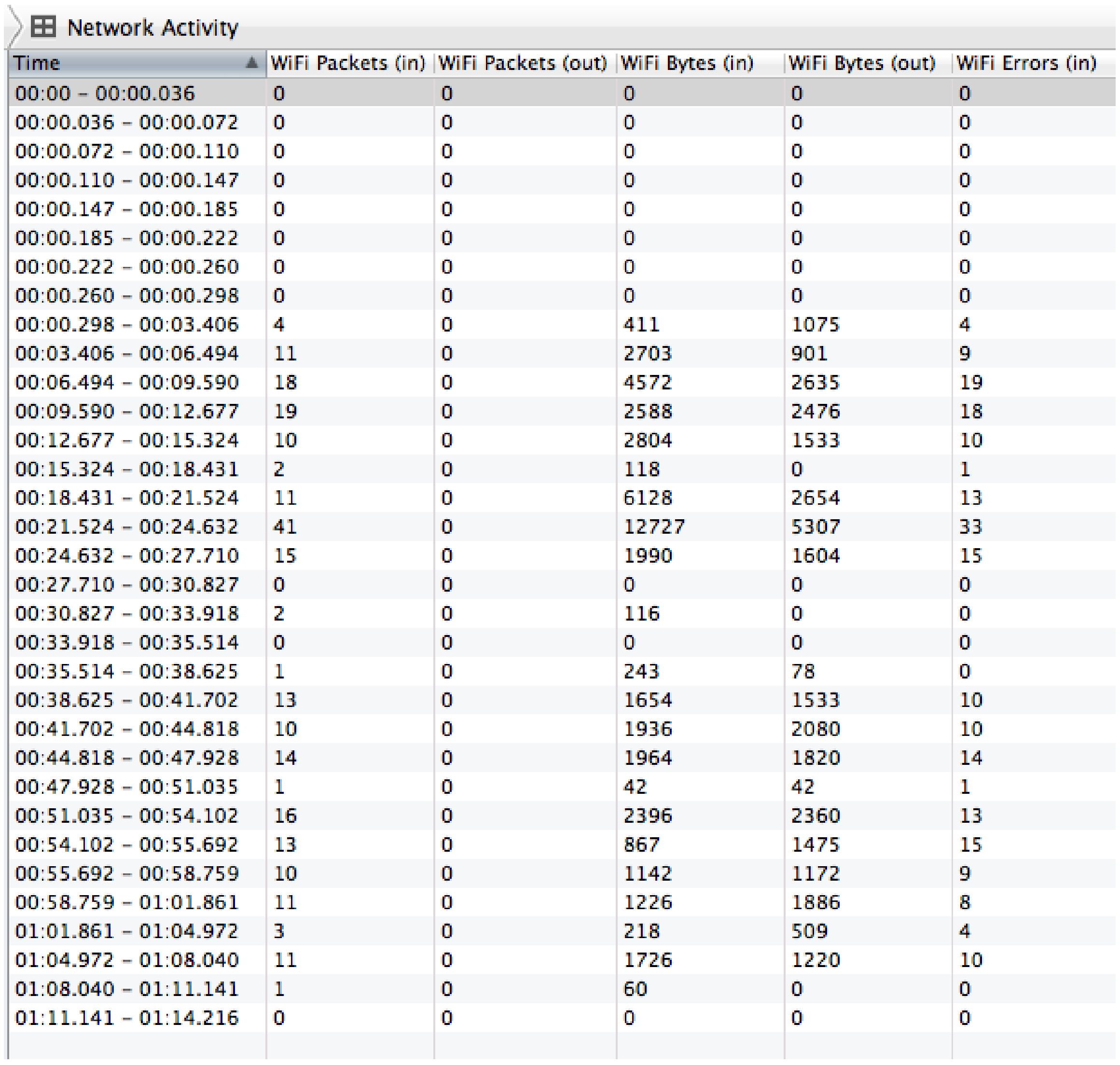
CPU Activity

Time	Total Activity	Foreground App Activity	Audio Processing	Graphics
00:00 - 00:01.507	9.5%	0.1%	0%	0.3%
00:01.507 - 00:04.508	10.4%	9%	0%	3.3%
00:04.508 - 00:07.507	7%	4.7%	0%	2.7%
00:07.507 - 00:10.508	12.7%	6.8%	0%	7.7%
00:10.508 - 00:13.508	12%	4.7%	0%	10%
00:13.508 - 00:16.508	13.4%	5.9%	0%	11.2%
00:16.508 - 00:19.507	11.3%	3.4%	0%	12.2%
00:19.507 - 00:22.508	24.9%	10.8%	0%	11.2%
00:22.508 - 00:25.508	17.3%	4.2%	0%	9.6%
00:25.508 - 00:28.508	7.3%	0%	0%	3.5%
00:28.508 - 00:31.507	6%	0.1%	0%	3.3%
00:31.507 - 00:34.507	8.2%	0%	0.1%	3.5%
00:34.507 - 00:37.507	6.5%	0%	0%	3%
00:37.507 - 00:40.508	12.3%	3.1%	0%	8.3%
00:40.508 - 00:43.506	12.6%	3.8%	0%	7.7%
00:43.506 - 00:46.509	16.2%	6.2%	0%	10.9%
00:46.509 - 00:49.506	9.3%	0.5%	0%	8%
00:49.506 - 00:52.508	8.5%	2.5%	0%	6.6%
00:52.508 - 00:55.508	13.6%	5.3%	0.1%	8.8%
00:55.508 - 00:58.507	11.5%	7.8%	0%	8.1%
00:58.507 - 01:01.508	9.6%	2.6%	0%	8.4%
01:01.508 - 01:04.507	9.2%	3.2%	0%	8%
01:04.507 - 01:07.507	10.2%	1.5%	0%	11.5%
01:07.507 - 01:10.506	4%	0%	0%	3.8%
01:10.506 - 01:13.506	3.8%	0.1%	0%	3.8%

Figure 9. CPU Activity with Instruments

4.1.1.3 CPU Activity

In Figure 9, we measured the CPU activity while running SportMingles. The peak CPU usage for SportMingles was 10.8% for the application while the peak total activity, which was the total CPU activity used for both the application and other iOS processes, was 24.9% which occurred when the GPS functionality was used. Overall the CPU usage for SportMingles stayed below 11% which means that the relative CPU activity for just running the application itself is quite low.



Network Activity

Time	WiFi Packets (in)	WiFi Packets (out)	WiFi Bytes (in)	WiFi Bytes (out)	WiFi Errors (in)
00:00 - 00:00.036	0	0	0	0	0
00:00.036 - 00:00.072	0	0	0	0	0
00:00.072 - 00:00.110	0	0	0	0	0
00:00.110 - 00:00.147	0	0	0	0	0
00:00.147 - 00:00.185	0	0	0	0	0
00:00.185 - 00:00.222	0	0	0	0	0
00:00.222 - 00:00.260	0	0	0	0	0
00:00.260 - 00:00.298	0	0	0	0	0
00:00.298 - 00:03.406	4	0	411	1075	4
00:03.406 - 00:06.494	11	0	2703	901	9
00:06.494 - 00:09.590	18	0	4572	2635	19
00:09.590 - 00:12.677	19	0	2588	2476	18
00:12.677 - 00:15.324	10	0	2804	1533	10
00:15.324 - 00:18.431	2	0	118	0	1
00:18.431 - 00:21.524	11	0	6128	2654	13
00:21.524 - 00:24.632	41	0	12727	5307	33
00:24.632 - 00:27.710	15	0	1990	1604	15
00:27.710 - 00:30.827	0	0	0	0	0
00:30.827 - 00:33.918	2	0	116	0	0
00:33.918 - 00:35.514	0	0	0	0	0
00:35.514 - 00:38.625	1	0	243	78	0
00:38.625 - 00:41.702	13	0	1654	1533	10
00:41.702 - 00:44.818	10	0	1936	2080	10
00:44.818 - 00:47.928	14	0	1964	1820	14
00:47.928 - 00:51.035	1	0	42	42	1
00:51.035 - 00:54.102	16	0	2396	2360	13
00:54.102 - 00:55.692	13	0	867	1475	15
00:55.692 - 00:58.759	10	0	1142	1172	9
00:58.759 - 01:01.861	11	0	1226	1886	8
01:01.861 - 01:04.972	3	0	218	509	4
01:04.972 - 01:08.040	11	0	1726	1220	10
01:08.040 - 01:11.141	1	0	60	0	0
01:11.141 - 01:14.216	0	0	0	0	0

Figure 10. Network Activity with Instruments

4.1.1.4 Network Activity

The network activity can be seen in Figure 10 demonstrating how many data packets get sent out and in. The network level stayed relatively low throughout the test. The max outbound traffic was 5,307 bytes while the max inbound traffic was 12,727 bytes when running through major functionalities of SportMingles in a one minute and thirty second time frame using the iPhone device by hand.

CHAPTER 5 TOOLS AND TAKEAWAYS

5.1 GIT

For version control of both the client and backend application, we used GIT. GIT is a distributed source code management tool [11]. There were a total of 540 commits for the client application and 19 commits for the backend application. The client application was 43,130 lines of code while the backend application was 7,582 lines of code.

5.2 TAKEAWAYS

There are several ways to create iOS graphical user interfaces such as using Interface Builder, programmatically, or Story Boards. We chose to programmatically create the user interfaces because it gave us easier controls to skin the user interface with custom graphics. The downside to creating user interfaces programmatically is the tedious need to specify exact x and y coordinates and width and height. A large part of designing the iOS application was dedicated to this to create a user interface to our liking.

When we first set out to create the application, we began with manual reference counting. We thought it would be a good way to learn how memory management works and it would also allow larger support of users with older versions of iOS. However, after a third of the way through development, we began spending major time debugging memory leaks and handling reference counting. Upon further research, we realized that the adoption rate of newer iOS versions was quite high and thus, decided to convert the application to use Automatic Reference counting. This allowed us to focus on developing features for the application rather than worrying about the memory.

Be prepared to create two versions of every graphic created. The reason for this is because of Apple's Retina and Non-Retina device resolutions. Two versions of each graphic were created to support users on the different devices. Thus, a portion of development time was dedicated to ensuring two versions of each graphic existed in the project. This also means that developers need to optimize their graphics since there will be two versions of each file in the project.

Submitting and getting an app into the Apple App Store requires patience. The entire review process is in a black box. After reading several blogs about people's experiences with app submissions, application review times can vary from a couple of days to several months depending on the number of applications in the queue for review and the type of application being submitted. Apple provides a handy guideline document that outlines some of the key points Apple is looking for during their review process [12].

CHAPTER 6 APP STORE RESULTS

6.1 App Store Submission Process

Once an app is ready for submission to the App Store, several steps are to be completed before it is put into a queue for Apple to review. These steps include: creating an iTunes Connect account, signing digital contracts, creating app icons for retina and non-retina display, application screenshots, and Meta information about the application. Once the application is ready for upload, the application developer can use XCode to build and archive the application. XCode then goes through a validation phase and then an uploading phase. Once uploaded, the application is in the state “Waiting for Review,” which means the application is now in the queue to be reviewed. This process can take several days to months depending on the volume of apps Apple needs to review.

6.2 SportMingles Results

Once SportMingles was “In Review” phase, this process can also take couple of hours to days. For SportMingles it took 8 days for the application to be in “Waiting for Review” status and 1 day for “In Review”. Once the app is approved it goes through a “Processing for App Store” phase and then “Ready for Sale” phase. Once the application reaches the “Ready for Sale” phase, the application can be downloaded and searched within the App Store within a couple of hours.

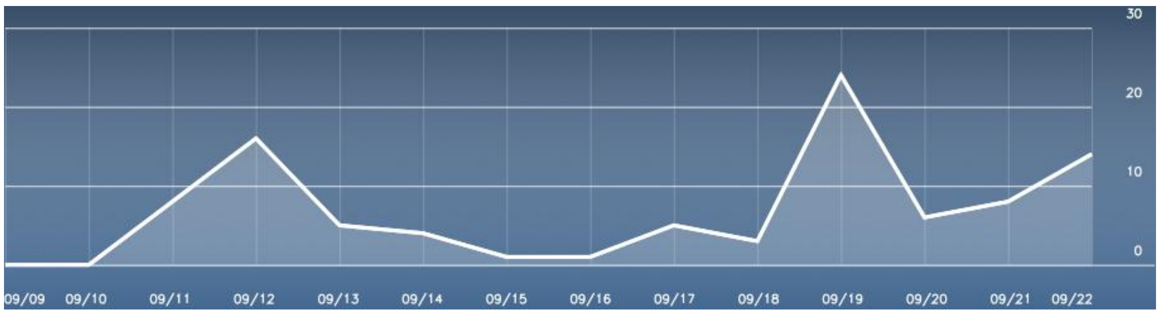


Figure 11. Week 1 Results

On September 11, 2012, SportMingles was approved and in the App Store. During the first week of release, the app received 35 downloads.

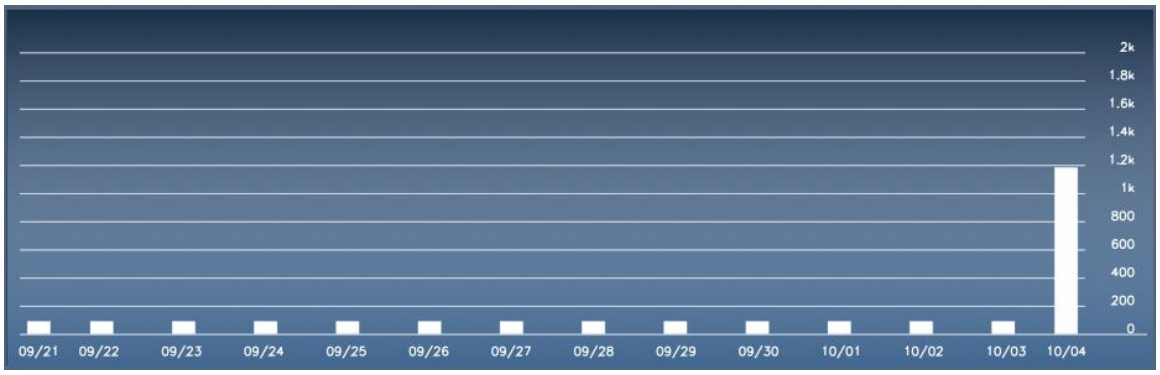


Figure 12. Cumulative Results

From September 24 to October 4, SportMingles received 1,142 downloads which pushed the application rank as high as #24 in the Free Applications category of Sports.

CHAPTER 7 FURTHER WORK

SportMingles can be extended and further improved upon in several different ways. As we saw from downloads and usage reports of the application, more features can be implemented to make playing and finding sports easier. Features such as incorporating a chat feature for games and teams, profiles for players, and suggesting games and teams based on user preferences. As the application scales, further improvements can be added to the infrastructure of the backend application such as placing the backend database into a cloud solution for instant scalability.

Further integration with social networks can also be added such as integration with Twitter for single sign on and posting tweets of game victories or game creation. Additionally, the teams screen can be improved upon by adding team pictures and videos. Platform extension can also be addressed as Android and Windows Mobile begin to gain market share of mobile devices.

CHAPTER 8 CONCLUSION

As mobile devices increase in popularity, the Apple's App Store enables developers to create applications and distribute them through a low cost channel. This reduces time of delivery and increases market reachability. Using the iPhone's computing power, Internet connectivity, and location based awareness, powerful, smart apps can be built. Using these technologies, we explored the concept of a sports social network. We analyzed the current market, developed user stories that drove requirements, discussed the implementation of both the client and backend, performed tests, and analyzed the results of the application in the market.

References

[1] When Will The Post-PC Era Arrive? It Just Did. http://techcrunch.com/2012/02/06/when-will-the-post-pc-era-arrive-it-just-did/, 2012.

[2] Apple's WWDC Keynote by the Numbers. http://allthingsd.com/20120612/apples-wwdc-keynote-by-the-numbers/, 2012.

[3] Streamline Your App with Design Patterns, https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/StreamlineYourAppswithDesignPatterns/StreamlineYourApps/StreamlineYourApps.html, 2012.

[4] Apple Push Notification Service, http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html, 2012.

[5] PHP, http://us2.php.net/

[6] JSONKit, https://github.com/johnezang/JSONKit, 2011

[7] Transitioning to ARC Release Notes, http://developer.apple.com/library/mac/#releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html, 2010.

[8] phpMyAdmin, http://www.phpmyadmin.net/home_page/

[9] Framework, http://whatis.techtarget.com/definition/framework, 2005

[10] Haversine Formula, http://en.wikipedia.org/wiki/Haversine_formula

[11] GIT, http://en.wikipedia.org/wiki/Git_%28software%29

[12] App Store Guidelines, https://developer.apple.com/appstore/guidelines.html

[13] Duckett, Tim. Pro iOS table views for iPhone, iPad, and iPod touch. New York: Apress Distributed to the book trade worldwide by Springer Science+Business Media New York, 2012.

[14] Kochan, Stephen. Programming in Objective-C. Harlow: Addison-Wesley, 2012.

[15] Hillegass, Aaron. Objective-C programming : The Big Nerd Ranch Guide. Atlanta, GA: Big Nerd Ranch, 2011.