Gaming@Stanford/SUJogger

By Anthony Lai, Felipe Pimentel and James Yang

Our project consists of 2 parts: Gaming@Stanford provides a framework for developers to build applications on, and SUJogger, a complex social jogging application, builds on top of Gaming@Stanford as a proof of concept of the powerful functionality of Gaming@Stanford.

We also hope that through SUJogger, we can encourage students and the community to exercise more, socialize more, be healthier, and enjoy a better life.

Gaming@Stanford

Goals and objectives of the Gaming@Stanford Infrastructure

We have several objectives for Gaming@Stanford:

- Exploration: Explore the possibility of creating a backend that can be used by all sorts of mobile applications, without the need to create their own backends.
- Ease of Use: When developers are using the Gaming@Stanford to build their mobile applications, it should be very easy and simple to use rather than another giant framework they have to learn.
- Authentication: Gaming@Stanford should provide an authentication mechanism for the applications to authenticate and identify their users and their friends.
- Social aspects: Gaming@Stanford should provide ways to make their applications to be social by default.
- Messaging Service: Gaming@Stanford should provide a messaging mechanism for application users to communicate with each other.
- Achievements System: For most games, there are certain achievements or award systems involved. It would be nice to provide a way to keep track of these achievements and provide further capabilities to see leader-boards and scoreboards on particular achievements and statistics.
- Groups/Group Achievements System: We are strong believers that when users engage in groups, they participate more in the application due to self-motivation and peer pressure. Gaming@Stanford should provide ways for users to create groups, and achievements can also be based on group participation.

- Generality: Our infrastructure should be able to handle the retrieval/storage of arbitrary objects needed by the application. If it is not general enough, then no application will use it.

- Limitations: We want to focus on making the platform really good in certain features, and that means it may not be generalized enough to suit all sorts of applications. For Gaming@Stanford, we believe it suits for most applications, even for applications that are not games. However, Gaming@Stanford may not be suited for applications that require intensive database mining operations.

Gaming@Stanford Infrastructure/Design Considerations

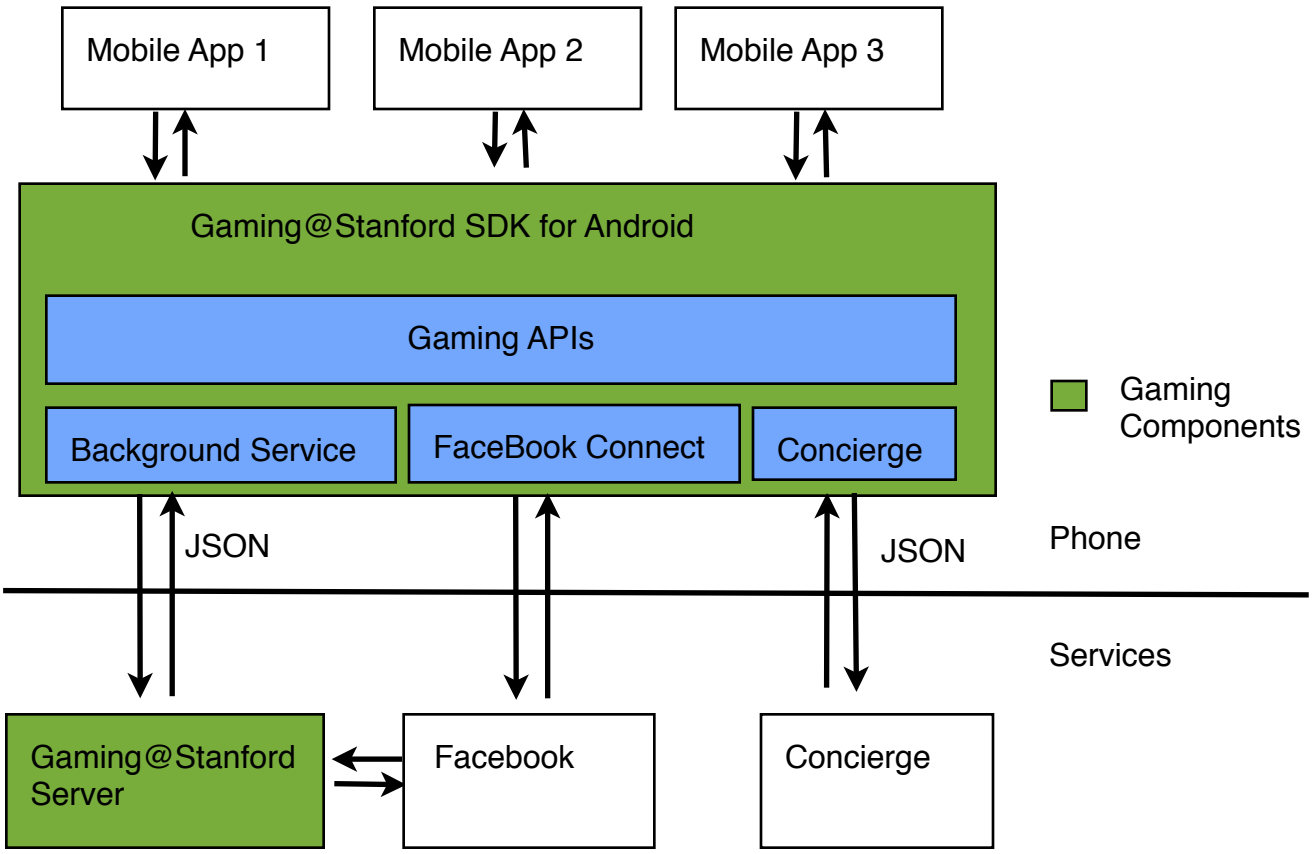


Figure 1. Gaming@Stanford architecture

Overall Infrastructure

Figure 1 depicts the Gaming@Stanford architecture as a whole, with the components shown in green. The Gaming@Stanford SDK resides in the phone, whereas the Gaming@Stanford Server is a standalone server for persisting objects.

The mobile application communicates with the SDK to get the services it wants, and SDK will either handle the requests by itself whenever it can, or pass the request to Gaming@Stanford server, Facebook or Concierge, depending on what the type of request it is. The only thing the application needs to know about is the Gaming@Stanford SDK API. With this architecture, all the services such as the authentication or messaging system can be replaced with other compatible services, without any change to the application. This provides full flexibility to uptake new services in the future if we wish to do so.

The SDK has built-in integration not only to Gaming@Stanford Server, but also with Facebook, and Concierge. Therefore, it can provide authentication and social aspects through Facebook and messaging through Concierge for the mobile applications.

Gaming@Stanford Server

The server is built using Rails and MySQL, and its main function is to be able to receive JSON requests, process them, and return a JSON response back to the SDK. The database schema consists of the following tables:

• Apps
This stores all the mobile applications that use the server.

• Apps_Users
This stores all the users that are using the applications.

• Developers
This stores all the developers of mobile applications. One developer can have more than one application using our services

• Friends
This stores all the friends of a user.

• Groups
This stores the groups created by users. At the current implementation, groups are partitioned by applications and can not be shared across applications.

• Groups_Users
This stores the users who have joined the groups. To provide more flexibility, it is up to the application to decide on how group administration/management is done.

• Objs
This stores generic objects that an application would wish to persist in the server. We use Object_Properties to provide the generality as described below.

• Object_Properties
This stores the object properties of an Obj. Each property can store text, int, float or blob, which covers every types that an app may need to store. It has a property name to identify what property it is and a property type to identify what type (text, int, etc.) it is.

• ScoreBoards
This table is used to generate scoreboards of a game, and through this specialized table, we can retrieve things like the top scorers, the bottom scorers, or the list of scores/achievements of a particular user.

• Users

This stores all the users that is using the Gaming@Stanford Server.

Gaming@Stanford SDK

The Gaming@Stanford SDK consists of 3 main components as depicted in Figure 1.

• Gaming APIs
The APIs provide the interface for the application to interact with Gaming@Stanford. To use Gaming@Stanford, there is only one Java class called GamingServiceConnection that the application needs to use, and everything else will be handled gracefully using its other services.

For storing/retrieving objects or scoreboards from the Gaming Server, the SDK would send a request for the application in JSON format, and the server would send back a JSON response, which would be converted into a Java object and returned back to the application.

• Background Service
Once the mobile application sends an intent to start using the Gaming@Stanford services, a background service will be started. It listens on Concierge and will pick up and messages dedicated to the application and the specific user and notify the application that a message has arrived.

• FaceBook Connect
The SDK provides the capability to connect to FaceBook Connect. It will help request additional privileges to get the user's email address and information about the user's friends for the application, and such information will be stored in the server for later retrieval purposes.

• Concierge
When the application needs to send messages to other users, it will call the Gaming@Stanford API, and in the background, the SDK would invoke a call to Concierge and put the message into the stream using "Gaming" as the principal.

Gaming@Stanford SDK under the hood for asynchronous processing

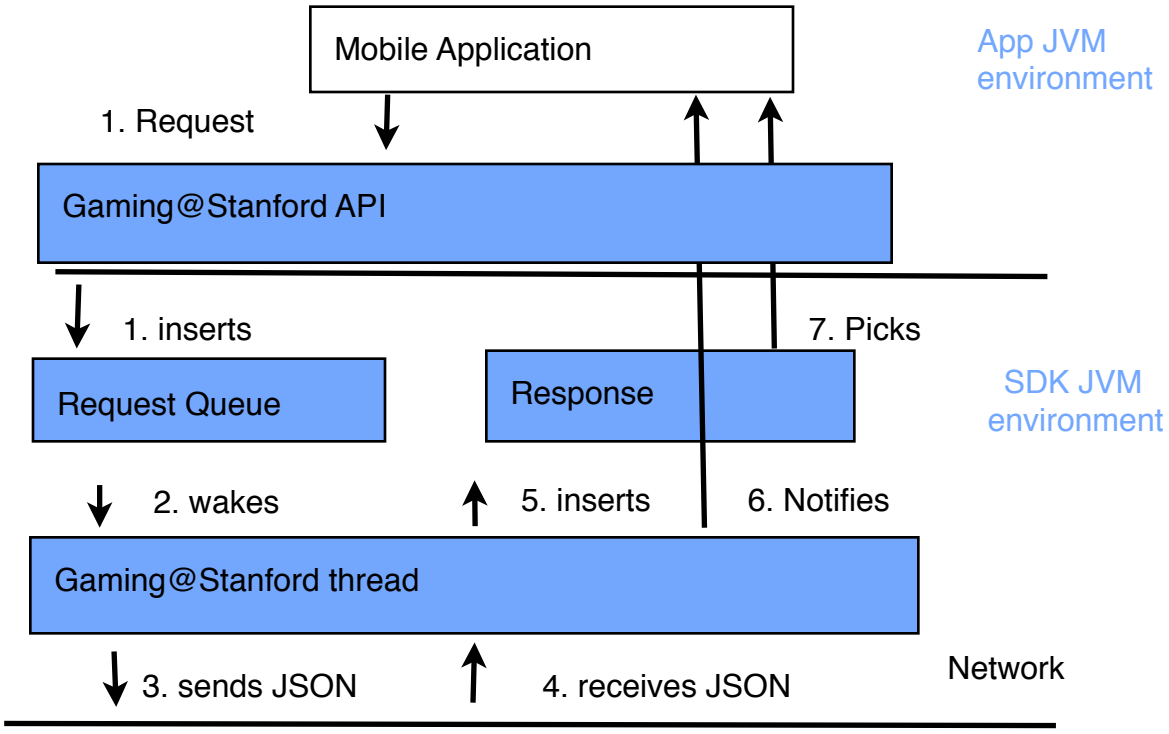


Figure 2 Asynchronous processing

The Gaming@Stanford SDK provides an asynchronous mechanism when the application calls the API, as it is undesirable for the mobile application to block while the SDK is sending requests through the network. The asynchronous processing is depicted in Figure 2 above.

When the application sends a request through the Gaming@Stanford API, the request is inserted into a request queue, then the control is immediately passed back to the mobile application to avoid blocking the application and thereby provide immediate UI response to the user. In the background, there is another thread running for each application, and this thread resides in a different JVM environment. When there are no requests being placed in the request queue, the thread is sleeping. Once there are requests received, the thread will wake up, process the request, wait for the response, put the response back into the response queue, and notify the application that there is response in the response queue by sending an intent to the application's BroadcastReceiver. The application's receiver, upon receiving the intent, wakes up, processes the intent, and handles the request, which may or may not involve updating the UI.

SUJogger

Goals and objectives

We have the following objectives in mind when creating SUJogger:

- Recording tracks

Users should be able to record their own tracks, share their own tracks with others, and also download tracks from their friends.

- Statistics

Users should be able to see the statistics of their own tracks and learn about their jogging habits and progress.

- Incentives through achievements

Users should get incentives when they achieve a certain level.

- Jogging mobile application going social

Currently there are no mobile applications that have a significant social component. We wish to do so by allowing users to interact with their friends in our SUJogger application, and introducing a group mechanism for users to be able to enjoy jogging with their friends.

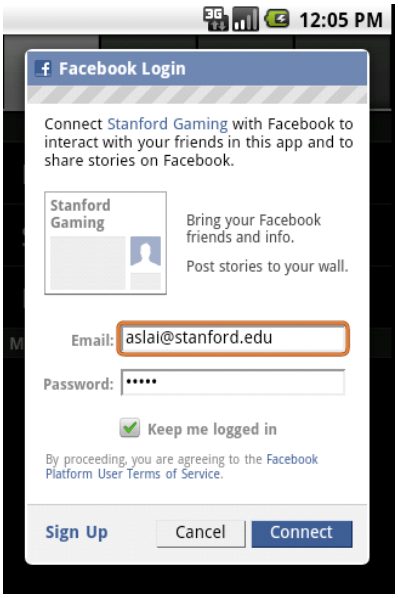
- Proof of concept

One of the main focus of the SUJogger is to prove that using an infrastructure like Gaming@Stanford can ease the implementation effort when building a gaming mobile application.

Result

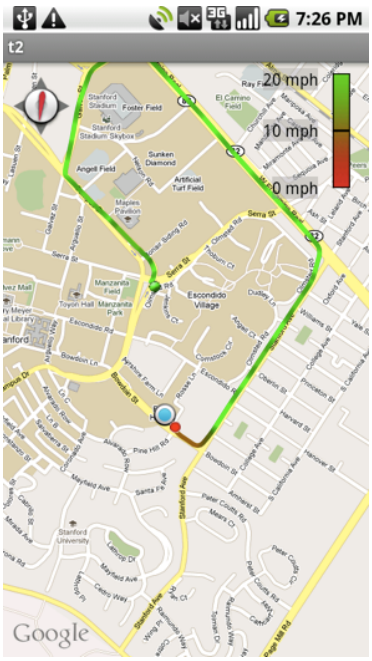
Gaming@Stanford authentication

At the current time of the release, the Gaming@Stanford is using Facebook as the authentication mechanism. SUJogger use Gaming@Stanford to perform user authentication.



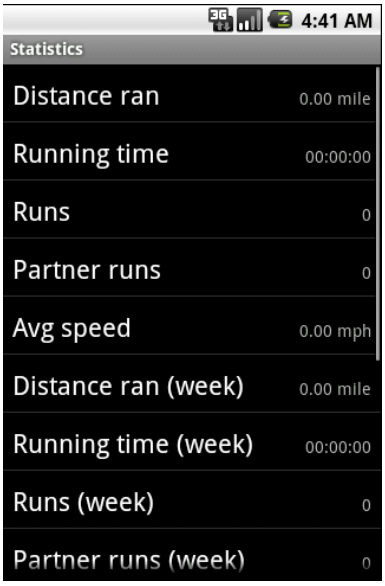
Tracks recording

We started off building on top of an open source project called OpenGPSTracker, and implemented the ability to overlay multiple tracks on the map. We also maintain detailed statistics on the user after each run.



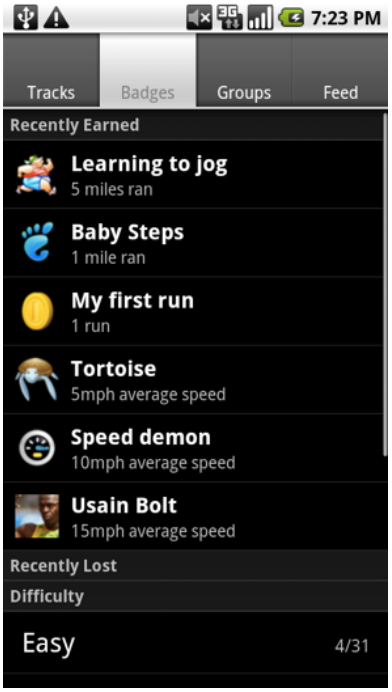
Statistics

Users can see their statistics in the Statistics screen to keep track of their training progress.

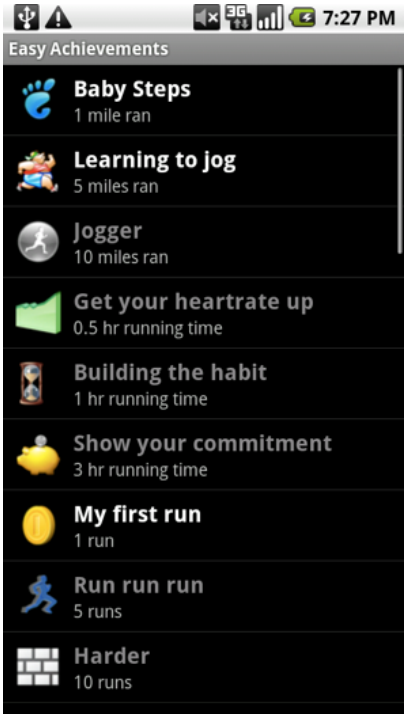


Incentives through Achievements

We have decided to use a badge achievement system as our main gaming award system. There are different metrics in the achievement system. Achievements are earned based on the distance, time, speed, and frequency of runs.

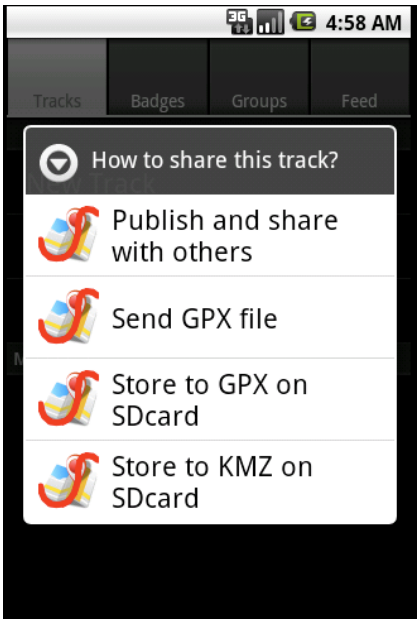


We have many achievements across many categories and difficulties. Novice and experienced runners all have something to gain when using this app.

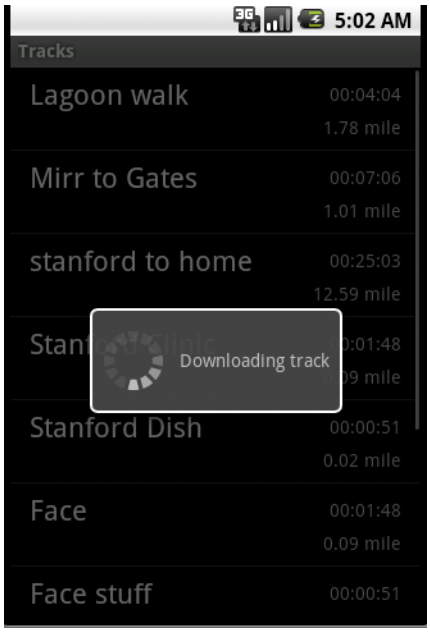


Sharing and downloading tracks

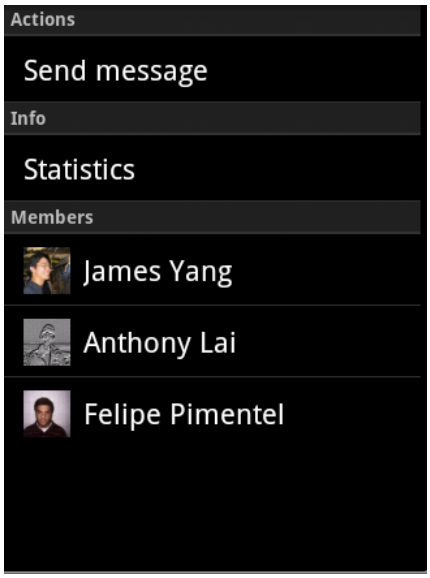
Users can publish their own tracks to the Gaming@Stanford server, which can be downloaded by others at any time.

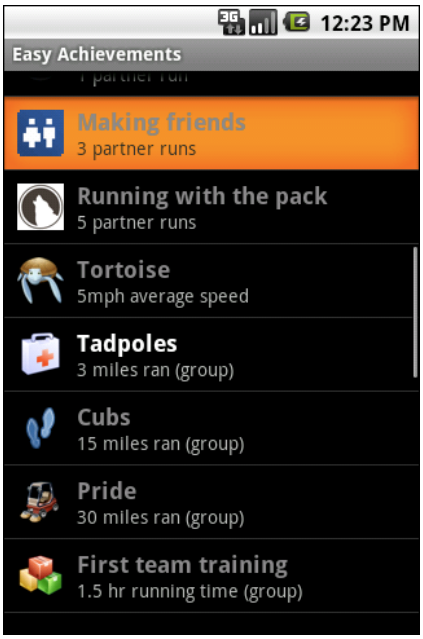


Any published tracks are associated to the user and can be downloaded and stored in the phone to be displayed on the map as any other regular tracks.



Users can create groups within SUJogger. When a group gets created, we would automatically generate the statistics associated with the group. Users can also send messages to members within a group. The group functionality is implemented easily through the Gaming@Stanford SDK.

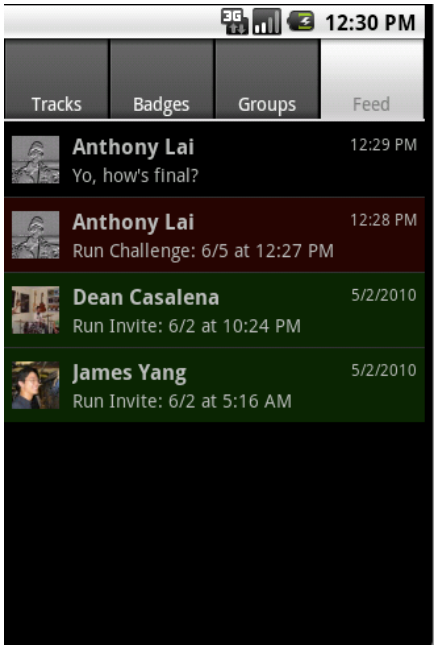




There are group achievements so that users can collaborate within the group. Through this mechanism, we hope that each user can encourage one and other to jog more.

Group achievements are automatically stored on the Gaming@Stanford server.

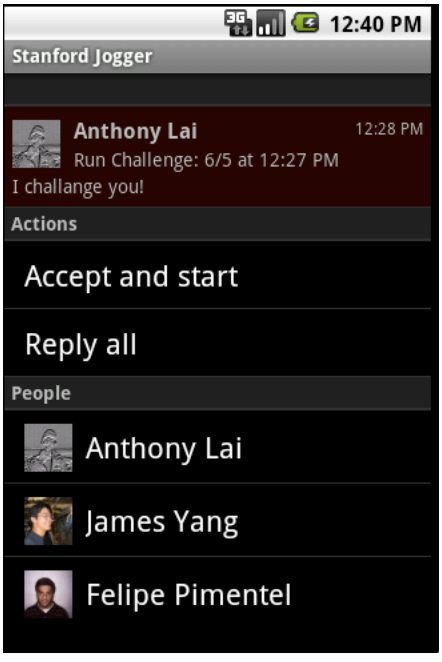
Messaging



As a social mobile application, it is important to provide in-app messaging. SUJogger uses Gaming@Stanford SDK, which in turn pushes the messages into Concierge. The SDK's background service polls the stream and notifies the application if a message directed to the user is received.

There are 3 types of messages: invites, challenges, and generic messages.

Generic messages are just chat messages users can send to each other.



Users can also send invites and challenges to their friends. Accepting and starting an invite/challenge means that the new track is flagged as a partner run, which is one statistic we maintain for the user.

Lessons Learned

Android

None of us had prior Android development experience before this class. After building SUJogger, we believe we now have the ability to build production-quality apps. The Android learning curve was steep at the beginning, especially because the SDK's background service needed to be multi-threaded, but once we got over it, things became much easier and more manageable.

We also had some difficulty integrating the SDK's asynchronous communication into the application. We eventually settled on the design pattern of having one BroadcastReceiver for every Activity that requires network communication.

Gaming@Stanford as a platform

As we indicated earlier, one of the main reasons to build SUJogger is to see if the idea of having a gaming platform would work. It turns out that such the usefulness of such a framework far exceeded our expectation.

One thing we realized is that no framework can be a good framework without stress testing it using a complex application. We built things from the ground up, creating the SDK first before attempting to integrate it into SUJogger. It turned out that when we started on the integration process, there were many loopholes and limitations in the SDK that needed to be fixed to allow the application to really be able to use the features. Work then continued concurrently on the app and the SDK, with improvements in both informing the other.

We know that phones have limited resources in terms of CPU power, battery and memory. We have structured the SDK so that even when there are multiple applications using the SDK, only one SDK service is created to handle requests from all applications. We have tested this feature with two applications, and it worked seamlessly. We believe that it is a huge win when there are more applications using the framework within a phone, as it would cut down resource usage significantly, compared to one service per app.

Almost all mobile applications need to have mechanisms for user authentication, social, server processing and messaging. Gaming@Stanford provides these mechanisms in an asynchronous fashion. This eliminates the duplication of work on each application necessary for asynchronous communication.

We also find that Gaming@Stanford is not only a gaming framework, but it is generic enough to handle most mobile applications. At the same time, it provides an achievement system that would be well suited for gaming applications.

Design and feature tradeoffs

There were many cool features we came up with during the development of SUJogger that we wanted to implement but couldn't because of time limitations. For example, we wanted to provide favoriting, commenting, ghost tracks (users can see in real time how they well they are doing against a moving track downloaded from their friends), and many other features. These features look easy to implement, but each feature also requires UI design and implementation. Phones have small screens, so the UI is even more important than on the desktop. There is a trade off between simple UI and functionality, and we found ourselves constantly deciding between the two.

Conclusion

We believe Gaming@Stanford is a great framework, and from what we've seen from the other CS294S groups, it looks like Gaming@Stanford would have been useful for many of the projects. It saved us lots of time when building SUJogger, and it should help other developers when building their own mobile applications.

We spent a lot of effort making SUJogger easy and fun the use with a polished UI, including badge icons for all 75 achievements. We personally enjoyed building and using it a lot, and we sincerely hope you like as well, and start jogging along with SUJogger today.

Appendix A. Related Work References

Related works for Gaming@Stanford

- XBoxLive http://www.xbox.com/en-us/live/
- FaceBook http://www.facebook.com/
- OpenFeint http://www.openfeint.com/
- Plus-plus http://plusplus.com/
- ScoreLoop http://www.scoreloop.com/
- Apple Gaming Center http://developer.apple.com/technologies/iphone/whats-new.html

Related works for SUJogger

- Run Coach http://www.runningmethod.com/
- Couch to 5K http://www.c25k.com/
- Run Keeper http://runkeeper.com/
- GPSies http://www.gpsies.com/page.do?page=iPhone
- Every Trail http://www.everytrail.com/iphone.php
- Trail Runner http://trailrunnerx.com/