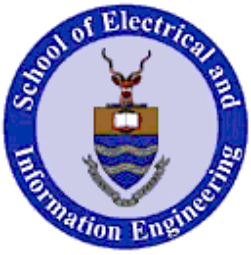

School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg

ELEN 4010 Software Development 3 Project 2010

Dribble: The Design, Implementation and Analysis of a Geosocial Networking System

May 2010

Ashleigh Campbell	0605245J
Chad Epstein	0706965X
Greg Favish	0713568E
Daniel Mankowitz	0616159H
Andrew Paverd	0702663D

http://github.com/ajpaverd/Dribble

DRIBBLE: THE DESIGN, IMPLEMENTATION AND ANALYSIS OF A GEOSOCIAL NETWORKING SYSTEM: GROUP REPORT

A. Campbell, C. Epstein, G. Favish, D. Mankowitz, A. Paverd

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: Dribble, a location-based social networking client-server application, has been developed. A user is able to receive messages called Dribs within three kilometers of the user's current location. The mobile application was developed using Android on the HTC Dream mobile phone. The mobile application was tested on different platforms and utilised logging and debugging techniques. Similarly, the server application was developed using the Java Enterprise Edition (JavaEE) framework and consisted of a multi-tiered architecture that was deployed onto a Glassfish Application server. Dribs have a popularity rating and a unique ID calculated using complex processing algorithms on the server side and data is stored in a database that is accessed using a database interface. The server utilised logging and debugging and both applications were sufficiently tested and integrated successfully. The application has been licensed under the BSD open-source license and version control was implemented.

Key words: Android, Dribble, geosocial networking, Java, software development

1 INTRODUCTION

The focus of this project was to build and release an open-source software application. An application called Dribble has been developed that conforms to the required project specifications and is based on a social networking structure. This application allows users to upload and download messages to and from their mobile phones that are within the range of a user's current location. This provides relevant and informative feedback to users of the application.

A detailed background of this application is described in Section 2. This is followed by the requirements and success criteria discussed in Section 3. Section 4 details the assumptions made for the design as well as the constraints imposed on the design. An overview of the project, including a system model, is presented in Section 5. The individual elements which comprise the overall system are then discussed. This includes the design of the mobile application in Section 6 and the server-side implementation in Section 7. The project underwent a testing phase and the results of this phase are detailed in Section 8. Project management aspects of the system are detailed in Section 9, including the team management, version control as well as open-source licensing. A critical analysis of the project has been conducted in Section 10 which includes a discussion of the trade-off's as well as recommendations for improvements on similar projects.

2 BACKGROUND

The term 'Social Networking' is used to describe a wide variety of online systems and applications. Although the concept dates back to the 1960s, one of the first networking sites, SixDegrees.com was launched in 1997 following the advent of the internet [1]. Over the past 10 years, social networking systems have continued to grow and analysis suggests that Facebook is now the second most popular website on the internet after Google [2]. One of the main aspects of modern social networks is the concept of linkages or friendships between members [3]. These channels provide the structure through which information is shared between users. As these networks continue to grow, privacy of users' personal information has become a prevalent issue [1]. Advances in location-enhanced technology increase the ease with which a user's geographic position can be determined. However, this presents the trade-off between the value of location information and further privacy considerations [4].

Currently, there are a number of social networking systems which integrate location-enhanced technology. This new trend is called geosocial networking. Pioneering applications in this new segment include Brightkite and Google Latitude [4]. One of the most popular geosocial network-

ing services is Foursquare [5]. Foursquare adds location information to the Facebook and Twitter networks and provides users with a location-based game-play element. It has even been suggested that Foursquare has what it takes to become the next Twitter [4].

Dribble can be classified as a geosocial networking system because of its core focus on location-based real-time social information sharing. However, unlike most of the similar existing systems, Dribble is an open-source application.

3 REQUIREMENTS AND SUCCESS CRITERIA

While there was a wide degree of freedom in the choice of software application, development platform and programming languages, the designed software system needed to be developed within certain constraints. Firstly, the application needed to be developed in groups of four or five members, committing to an average of 4.5 hours per week over the course of the allocated seven weeks. Each member was required to make a substantial contribution to the project and explore the use of at least one new development tool, technique or programming language.

The final product needed to incorporate information storage, processing, calculation and visualisation, including the Graphical User Interface (GUI), and to be implemented with a modularity priority, in order to easily add functionality at a later stage. Source code control had to be used to track version and development changes. The final product needed to be released under an open-source licence agreement, and therefore required that appropriately licensed code only (of no more than 60% of the final solution) be used in the development of the project.

Additional requirements defined by the members included the scalability of the final design and the adherence to standardised technologies. The member-defined success criteria for the final designed system was to produce a product that is fully integrated across all systems (highly modular), ready to be released to market. The product therefore needed to be implemented and simulated as a real world application.

4 ASSUMPTIONS AND CONSTRAINTS

Several assumptions were made when designing Dribble. They included considering that the Android cell phone has sufficient access to the internet either over 3G or EDGE as well as sufficient bandwidth and speed to run the application. For certain parts of the Android interface to work, Google Maps had to be available and accessible. The application would still retain most functionality without displaying the maps, but the GPS coordinate retrieval is vital. The Android application also assumes that the server is available and reliable. This means that the server functions correctly, that there is very little downtime, and that the server can handle and process requests. The application was designed with the assumption that there will be a large number of users. The application's true usefulness and functionality is only evident once a critical mass of users is reached and thus enough content can be generated to make Dribble valuable to clients. It is for this reason that scalability was also a high priority during Dribble development. It is also assumed that even when the application is used by a number of users, the server will be sufficiently load balanced such that there is no additional lag when a large number of users use the service simultaneously.

There were constraints that needed to be considered due to the project requirements, the mobile development platform and the application's client-server architecture. The application was developed using the Android development platform and is therefore currently limited to Android devices. Using a mobile device means that there are inherent constraints because of the limited processing power, memory and storage of this type of device. On a physical device, the network bandwidth limits the amount of data that can be sent to the server. The protocol used to send this information to the server cannot be modified and must be used by all mobile devices for the service to work correctly. The application also depends on the server availability at any given time and how quickly it can receive and process inputs and return results. The subjects and messages sent to the server are limited to maximum lengths of 20 and 144 characters respectively to encourage users to be concise. Once completed, the application must be released as open source and this means that all third party libraries used during

development must also be open source.

5 SYSTEM OVERVIEW

5.1 General System Model

A generalised system model of the overall Dribble system is depicted in Figure 1. The system is primarily divided into two components - the Mobile Application (Android) and the Remote Application Server (Glassfish). The Mobile Application is executed on the HTC Dream mobile phone and provides the GUI with which the users will interact. The mobile application is composed of logical JavaME source code and presentation layer XML scripting. The mobile phone initiates a connection to the server via web services, set up on the remote application server. The mutual packages shared between these two sides, define the two main object classes 'Drib' and 'DribSubject' which are used to convey messages and message subjects, respectively, to and from the server. The objects are serialised in order to be sent sent over an HTTP connection, using one of the following network technologies: 3G, EDGE, GPRS or HSDPA. The sent object is de-serialised on the server side and processed using Enterprise JavaBeans. The database is queried with the relevant request and returns the desired result. This result is returned using Enterprise JavaBeans and serialised to be sent back to the mobile application using RESTful web services. The mobile application uses Java and XML to graphically display the Dribs and DribSubject.

5.2 Use Case Model

The UML Use Case Model of the Dribble application is shown in Figure 2. This model depicts the major functional use cases as seen from the user's perspective. This functionality is provided by the mobile application which is in turn supported by the server application. As shown in the figure, the purpose of the application is for users to upload messages about events relevant to their current locations and then view the messages of other users. Since all messages are anonymous, all messages are available to all users in the area. By providing this functionality, Dribble can be classified as a geosocial networking system.

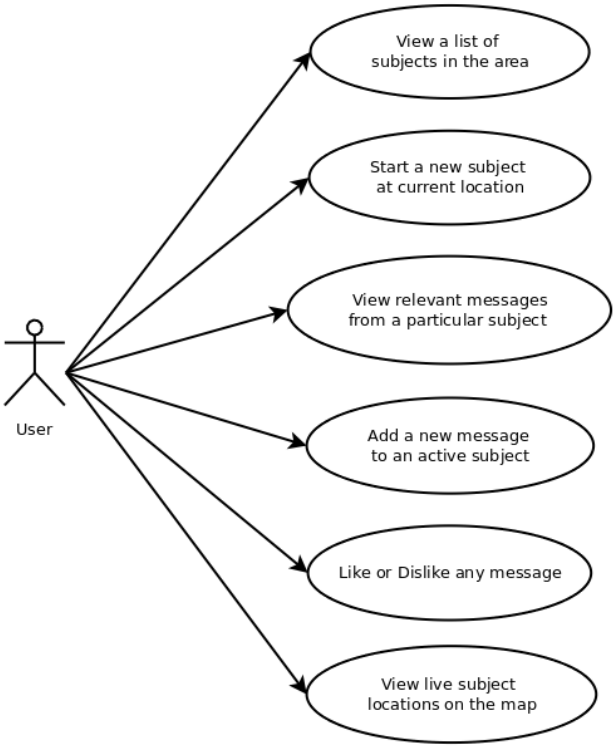

Figure 2 : Use Case Model for Dribble Application

6 MOBILE APPLICATION

The mobile application is the client of the overall Dribble system . A generalised structure is shown in Figure 3.

6.1 Platform Description

The project made use of the Android SDK which provides the tools and APIs necessary for development. The programming language used to develop the mobile application is a form of Java that is specific to the Android platform. This was integrated into the Eclipse IDE using the Android Development Tools (ADT) plug-in which provided many benefits. It generates the correct project structure for the application, and keeps resources, such as XML layout files or images, compiled automatically. It also provides tools integration, custom XML editors, and a debug output pane called LogCat. LogCat was utilised for session logging, and custom-made logs were embedded in the code to highlight function executions, thread and error checking and the ability to monitor warnings and override meth-

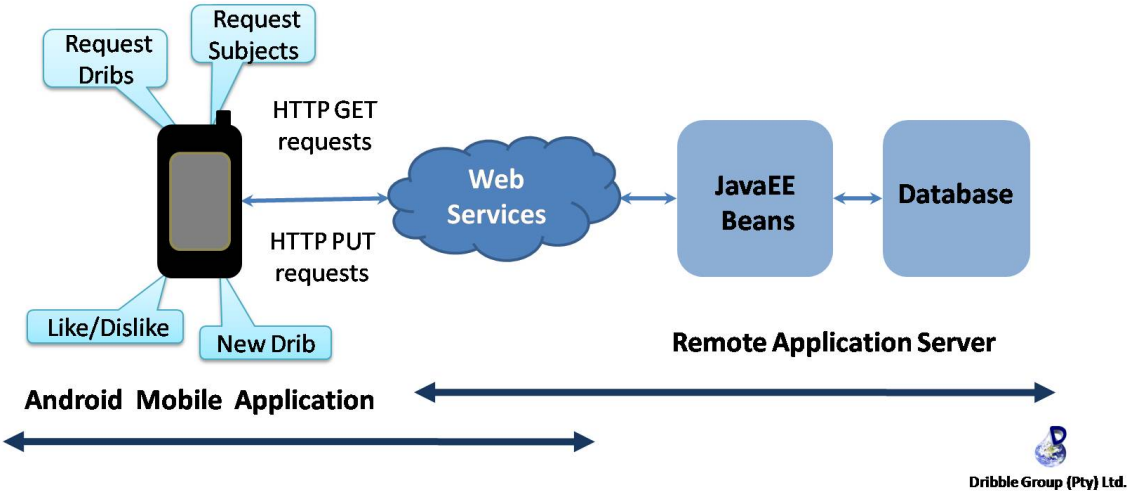

Figure 1 : Generalised Model of the Dribble System

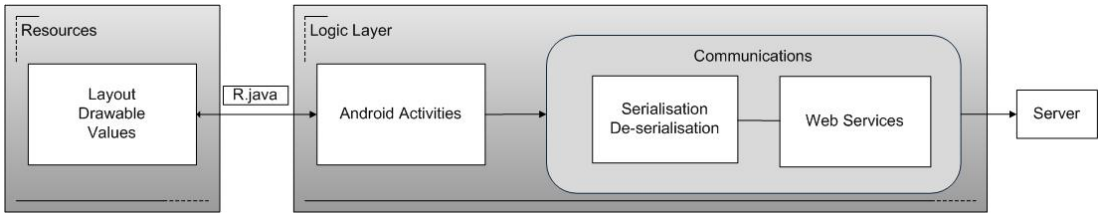

Figure 3 : Generalised Structure of the Mobile Application Implementation

ods. There have been recent releases of the Android operating system and there may be platform incompatibilities when developing for different firmware versions. The phone used for development (HTC Dream) contained Android version 1.5 and the same application may therefore not work on higher firmware versions, the latest being Android 2.1.

6.2 Design

With any software development, design forms a major component, and should be carefully constructed before implementation takes place. A wide range of development tools can be utilised. Firstly, a description was established highlighting the problem statement from the mobile application perspective. Next, a prototype GUI was constructed to illustrate the basic layer and functionality that would need to be employed. Finally a Use-Case model was developed to analyse the scenarios through which the user would navigate.

A key concept and trend in software development is the implementation of modular code. This is a technique that is used throughout the design and implementation, and requires that the system be decomposed into separate components [6], which can be added or removed without affecting one another. In this application modularity is achieved between the presentation and logic layers of the software system. The presentation layer is scripted in XML and links to the Java source code (logic layer), through the use of the 'R.java' class which is automatically generated [7]. Modularity was also implemented through the use of packages and web services.

6.3 Main Features

The application begins by displaying a custom splash screen containing the group logo. The user interface was designed with a tab layout as the main navigation tool. This allows for fast and intuitive interaction with the program, especially for touch-screen enabled devices, and creates a distinct separation of application activities. The tabs are (from left to right): Map, Topics, Dribs, and New. Map contains location-related functionality to be discussed later. The Topics tab allows the user to obtain a list of the most highly ranked DribSubjects in the area surrounding him

or her. The Dribs tab contains a list of Dribs for the selected DribSubject and the New tab allows for the creation of new Dribs and DribSubjects. Each time the Topics or Dribs tabs are clicked, the content is refreshed to the most up-to-date information from the server. This ensures that even if the user changes location, they always receive the latest content. This, however, still requires user interaction. A better implementation would be to use a thread that refreshes content automatically in the background at certain intervals which would not require user input. However, a possible disadvantage to implementing this functionality is the added processing requirement for the application which could affect the responsiveness and drain battery power.

6.4 Location-based Implementation

For the Map tab, the Google Maps library and APIs built into the Android SDK were used to perform map activities. This includes: obtaining a GPS location, and the ability to show, pan, and zoom a map. It also allows for overlay items to be displayed on the map, which, for the Dribble application, was used to show topic locations and details within the user's sphere of reference. The Dribble application uses a thread to attempt to obtain the users location from: GPS if available, otherwise from an approximate network location using cellphone towers. While the application is requesting a location, the map is displayed and tiled using data obtained from Google via the Internet. The map contains a default overlay item until topics are fetched from the server. After topics are fetched from the server, an overlay item is generated for each topic and added to the map. The overlay items are cleared and re-generated each time topics are fetched to ensure that the map only shows topics that are applicable to the area surrounding the user. Once the first location fix is obtained, a thread is started that runs throughout the duration of the program that constantly refreshes the user's location. It contains a listener that runs in the background and waits for location changes which are stored by the thread at set intervals which, for testing purposes, was every 10 seconds. This delay could be increased in a production version as it may consume unnecessary resources. Other activities access the thread throughout the program to use the location details when requesting DribSubjects, sending Dribs, and sending likes or dislikes.

6.5 Packages

Android Packages (APK) are used extensively throughout Android application as they add functionality that has be been bundled together within a package, which can be modularly added or removed. Three packages were implemented in this application: 'common.dribble' (contains the classes common to the mobile application and the server, and wrapper classes); 'dribble.dribbleapp' (contains the classes implementing mobile functionality) and finally 'XStream' (discussed below).

The 'common.dribble' package provided a common interface between the server and the client. It contained the main classes 'Drib' and 'DribSubject', which were instantiated every time a user created a message or message subject. These objects contained important processing data such as names, latitude and longitude positions. Before an object could be sent, it needed to be serialized into the chosen communication scripting language, XML (XML is commonly and effectively used with web service applications). Within Dribble, objects are automatically serialized with XStream, and sent via a HTTP Put request (no object is sent when using the HTTP Get request). The HTTP connections are formulated with the RESTful web services on the server-side application (see *Section 7.3*).

Due to the incompatibility of the different serialization packages utilised on the mobile application and the server, two wrapper classes were introduced (one for each common class). The purpose was to ensure the XML root nodes conformed to a standard structure, defined by the server-side. This added a new level of modularity between the serialization package and the communications class.

7 SERVER-SIDE APPLICATION

7.1 Development Framework

The server-side application has been developed using the Java Enterprise Edition (JavaEE) application framework. This framework has a multi-tiered distributed application model [8]. Thus the development of an application under this framework involves a number of tiers with separate

functionality. The JavaEE framework has been utilised as it is a standardised and reliable software architecture [8]. Many of the low level services that generally take time to develop have already been implemented which saves time and allows developers to concentrate solely on the application logic. The JavaEE application is also very flexible as it can be deployed to almost any application server with very few changes [8]. This framework has been implemented in the NetBeans environment and is then deployed to the application server.

7.2 Application Server

An application server was implemented to store and maintain the Dribble server-side application. One of the main reasons the application server was chosen is because of it being centralised allowing client computers to access the application from any platform [9]. Centralising the application holds many advantages for maintenance and management. If a server cannot deal with the current demand, the system can easily be upgraded thus making for a scalable and maintainable system. The application server also has the capability to perform load-balancing and fault tolerance techniques thus allowing for application scalability.

The specific application server utilised is called Glassfish and has been implemented in order to manage and run the Dribble server-side application. The Glassfish application server is open-source and thus meets the open-source requirements of the project [10]. Another distinguishing feature of the server is its high performing message-queuing implementation [10]. This is discussed in more detail in Section 7.3.2.

The Dribble server-side application is administered using the Glassfish Administration console. A large amount of functionality is available to a Glassfish administrator. Applications can be manually deployed and undeployed using the console. Queues discussed in Section 7.3.2 are easily setup for web-tier and business-tier communication. Connection to the Glassfish database can easily be established. These are only a few of the features that makes the Glassfish application server such a powerful tool.

7.3 Web Tier

The server-side application needed an efficient, light-weight method of communicating with the mobile application. These criteria were attained by implementing Representation State Transfer (RESTful) web services. All data and functionality in RESTful web services are considered resources and the resources are accessed using Uniform Resource Identifiers (URIs)[11]. The URIs are generally links that a user will access using HTTP methods such as GET, PUT, POST and DELETE in order to expose a specific RESTful web service. The web services are designed to use a stateless communication protocol which in this case is HTTP [11].

One of the main reasons RESTful web services were chosen is that they scale to meet increasingly high performance demands[12]. This is partly due to the fact that they are stateless and hence provide all the information needed to generate a response in a single request by encapsulating this information in the HTTP headers and body. The server can thus forward, route and load-balance without maintaining state between requests [12].

Another feature of RESTful web services is the standardised way in which data is transferred to and received from client applications as shown in Figure 5. Objects can be sent using RESTful and are de-constructed into a representational form of the object in XML, JSON or both [12]. Similarly objects can be re-constructed when received as JSON or XML. This allows a variety of clients, running different platforms and developing with different languages, to use the service. It also allows development to be purely Object-Oriented and removes the need to apply more complex techniques to re-construct the object such as XML parsing.

Figure 5 : RESTful's standardised communication protocol

7.3.1 RESTful Web Services Implementation

As can be seen in Figure 5, there are three RESTful web services that were implemented, each providing resources that can be accessed by a link to the relevant web service. The 'putDribResource'

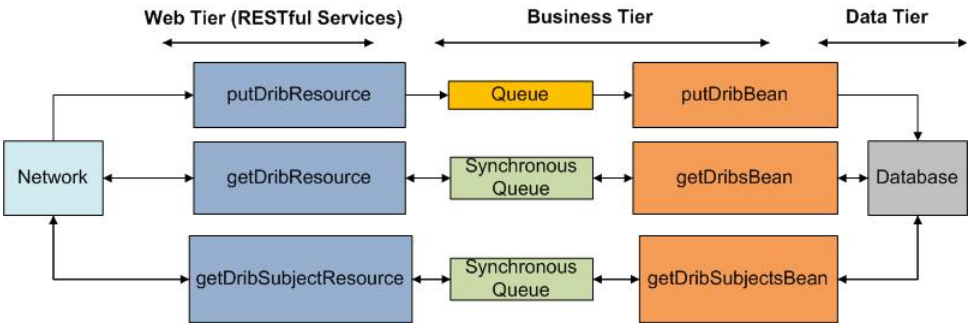

Figure 4 : The server-side flow diagram

receives a Drib object discussed in section 5.1 and adds it to the database. This web service is accessed using an HTTP PUT request. This request needs to be directed to the correct URI specified by the web service. This means that when the client posts a Drib to the application using the HTTP request, the 'putDribResource' web service is exposed and places the Drib on a queue for processing at the business layer.

Subjects can be requested from the 'GetDribSubjectsResource' web service and Dribs for a specific subject can be requested using the 'GetDribsResource' web service. Both these services are accessed using an HTTP GET request. In order for these requests to be completed successfully, query string parameters are passed in the GET request and the Drib data can then be retrieved.

The server-side Dribble web services are also implemented to convert objects sent to the client to an XML representation of the object. Thus the MIME media type that is produced for clients is XML. Similarly when receiving objects, the web service resources are implemented to expect XML to reconstruct the objects.

7.3.2 Queuing Once a web service receives data, it may need to send the data to other objects such as enterprise beans on the business tier for processing. The communication between the web services and enterprise beans as seen in Figure 6 is achieved using a point-to-point queuing implementation [13]. Queues are part of the Java Messaging Service (JMS) which is an API that is part of the JavaEE platform.

The queuing process can be seen in Figure 6. The web service object or client will send a message to a queue. The queue will then retain the message until the enterprise bean, client two, discussed in 7.4 consumes the message in order to process it[13]. The object that sends the message into the queue is known as the message producer and object receiving the message is the message consumer[13].

An important feature in this type of implementation is that the communication is asynchronous. A client does not need to request a message in order to receive it[13]. The communication is also reliable as the JMS API ensures that a message is delivered only once.

In order for the web service to send a message to the queue, the web service needs to be bound to the queue. This is achieved using the Java Naming and Directory Interface (JNDI). This API allows the web service to lookup the name of the queue that has been implemented on the Glassfish server and binds the web service to that queue. In this way the web service 'putDribResource' is bound to a queue and sends Dribs to the queue to be processed by the enterprise bean.

It is important to note that queues are unidirectional. This means that a message can only be placed in a queue and consumed by a client waiting on the other side of the queue. This poses a problem if a user wants to request data from the application. In this case the web service needs to place the request on the queue and receive the requested data from the same queue. This is required when requesting a list of Dribs and a list of DribSubjects. This is solved using Synchronous Queues which allow for bi-directional communication. This is discussed in more depth in *Section 7.4*.

7.4 Business Processing Tier

The business or processing tier of the server application encompassed the processing and business

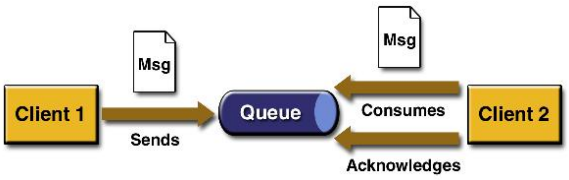

Figure 6 : A JMS queue flow diagram [13]

logic components. The combination of this tier and the web and data tiers formed the complete JavaEE application. This multi-tier architecture allowed the separation and decoupling of the different functional areas. As shown in Figure 4, the business tier components form the link between the web tier and the data tier. This section provides a high level overview of some of the important aspects of these components and explains how they interact with other components and tiers of the application.

7.4.1 Message Driven Enterprise JavaBeans The business tier consists of three main components as shown in Figure 4. These components are all Message Driven Enterprise JavaBeans (EJBs). EJB technology is the standard server-side component architecture of JavaEE. This technology enables rapid development of distributed, reliable and scalable applications [14]. A Message Driven Bean is a specific type of EJB which is designed to process messages from a JMS queue. As explained in the previous section, the web tier components send instructions and requests to the business tier using JMS queues. The application server manages the JMS queues and creates instances of the message driven beans as required to process the incoming messages.

The PutDribBean processes the incoming Dribs from users. When a Drib is received from the web tier, this bean classifies it as either a new message or an update of an existing message and processes it accordingly. One of the important responsibilities of this component is to ensure that every Drib and DribSubject has a unique identifier. The GetDribSubjectsBean and GetDribsBean are respectively responsible for retrieving the subjects and messages from the data tier and relaying them to the web tier. In order to achieve this, these EJBs require the use of synchronous JMS. These components are also responsible for ranking and ordering the lists of Dribs and DribSubjects.

7.4.2 Synchronous JMS JMS technology was used to send messages from the web tier to the business tier because it allows for efficient asynchronous communication. However, when the web tier requests a response from the business tier, two-way communication is required. This occurs in the getDribSubjectsBean and getDribsBean which return a list of subjects and messages to the web tier respectively. In order to achieve this, temporary JMS queues are dynamically created to receive responses as required. This structure is known as synchronous JMS. By using synchronous and asynchronous JMS simultaneously, a request-response model can be achieved whilst still maintaining all the advantages of asynchronous JMS as described in the previous section.

7.4.3 Unique Identifiers One of the critical aspects of this application is the correct identification of messages and subjects. In order to achieve this, every message and subject has to be assigned a unique identifier. It is possible to identify objects based on their contents but this severely limits the flexibility of the system since no two Dribs can then have the same message. In order to minimise network traffic and costs, the length of the identifiers must be limited since they are always transmitted with the messages. To generate these unique identifiers, the Dribble system uses random integer generation and a system-wide list of existing identifiers. A new identifier can only be issued if it is not already in the list and identifiers freed from deleted objects can be removed from the list. This method provides system-wide unique identifiers without creating a large data transfer overhead.

7.4.4 Popularity Score - DRank Another important function of the business tier is the ranking and ordering of the messages and subjects. This is important because only the most relevant information must be sent to the client so all the processing must be done on server-side. In order to rank the objects, a mathematical popularity score function was devised. This function assigns a numeric value to each object representing its popularity. The value is based on linear factors

such as the number of users who liked a particular message or viewed a certain topic. The popularity decreases as the object becomes older or the user moves further away. This is roughly specified by stating that the popularity score halves for every five minutes that elapse or every one kilometer of positional difference. This popularity score is recalculated whenever Dribs or DribSubjects are retrieved from the dataset. As an added feature, this score is presented to the user under the brand name DRank. The exact formulae for calculating this popularity score are shown below. In these equations, v is the number of views of the subject, m is the number of messages linked to the subject, l is the number of 'likes' for a particular message and x and t are the normalised distance and time quantities.

$$DRank_{subj} = (1000(v) + 1000(m))(\frac{1}{2})^{\Delta x + \Delta t} \quad (1)$$

$$DRank_{msg} = (1000 + 1000(l))(\frac{1}{2})^{\Delta x + \Delta t} \quad (2)$$

7.4.5 Use of Dataset Interface All components in the business tier are designed to use the Dataset interface. By using the interface instead of a concrete implementation, it is possible to change the underlying data storage system without affecting the business tier components. However, due to the decoupled nature of these two layers, the business tier has to assume that the data tier will account for any specific requirements of the chosen data storage system. The data storage system used in Dribble is explained in the next section.

7.5 Database

The data tier of the Dribble application consists of a data storage system. This system is responsible for the storage, management and retrieval of Dribs and DribSubjects. The system implements the Dataset interface to provide a standardised functionality expected by the business tier. A concrete implementation of this interface was created using a Derby relational database. The interface and implementation are explained below.

7.5.1 Interface In order to allow a flexible data storage strategy, the Dataset interface was created. This interface allows for the use of multiple types of storage system whilst remaining transparent to the higher tiers. In this case a relational database was used but a cloud storage solution could have been implemented to conform to the interface. The interface was designed to provide the following basic functionality:

- Storing Dribs
- Retrieving Dribs based on location
- Storing and retrieving DribSubjects
- Updating Dribs and DribSubjects

The aim of this interface was to provide sufficient functionality to the business tier components without placing unnecessary burden on the higher layer. This then assumes that any implementation of Dataset will handle any specific setup and maintenance required by the particular data storage system. The concrete implementation used is a class called SQL communicator.

7.5.2 SQLCommunicator This class implements the Dataset interface. As implied by the name the method of interacting with the Database is SQL. This class establishes the connection to the database using a Java Database Connector (JDBC) driver on initialisation. In order to improve the system efficiency, this connection is obtained from a connection pool maintained by the server. Apart from the methods required by the interface, this class also provides auxiliary methods which are used to facilitate the primary methods.

7.5.3 Database The database used in the implementation is a Derby Relational Database. Derby is an embedded database management system that is included with the Glassfish application server [15]. This type of database was chosen because of its easy integration with the application server as well as JavaEE application. The structure of the database in relation to Dribs and DribSubjects is shown in the Figure 7.

As seen in Figure 7, the database is divided into a number different tables. The main DRIBBLE_SYSTEM_SUBJECTS table stores a list of active subjects. The messages for each subject are stored in that subject's particular table. As new subjects are created by users more tables are dynamically created by the code. This structure was chosen so as to reduce the amount of data

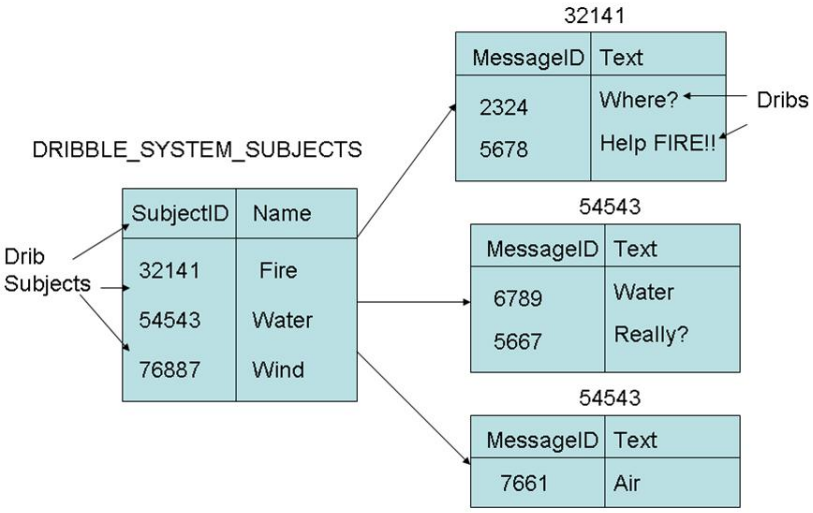

Figure 7 : High-Level Overview Structure of the Database

which has to be searched when messages from a particular subject are required. The maximum number of tables in this particular database is more than sufficient for this application.

8 SYSTEM TESTING

Due to the variety of components which comprise this system, a number of different approaches were used to test and ensure system functionality. These included the extensive use and analysis of log files, the use of a web-services testing framework, multi-platform testing and integration testing. System load testing was investigated but not implemented due to time constraints.

In both the mobile and server applications, log files were used to monitor the status of the application and to detect and identify any errors which occurred. In order to meet the requirement of using standardised components and subsystems, the logging was implemented using the standard capabilities of the Android emulator and the Glassfish application server. By following this recommended convention, the logging output could be processed and displayed in the standard format for each platform.

The NetBeans web-services testing framework was used on the server-side to verify the correct operation of the web-services components and subsequently, the rest of the server system. The framework interacts with the web-services in the same way as a real client would do. However, since the framework can be run locally on the server, it can eliminate any errors caused by the network link. In this way, it is possible to accurately test the server-side components in an idealised environment. To extend this test, the framework was also run over a local area network and the internet.

Multi-platform testing was used on both the mobile and server sides of the system. On the mobile side, the application was tested using both the Android emulator and an HTC Dream device. This was done to confirm that the application functions on both the emulator and a real target device. On the server side, the application server was installed and used on both Windows and Linux platforms. This multi-platform test ensured that the application and application server would be compatible with different platforms in a real server implementation.

The integration testing was the final high level testing of all the functionality in the system. This was performed from both the emulated and physical mobile devices. This type of testing included all the factors which would be present in the production system including the possible performance limitations and reliability issues associated

with using a mobile data network.

In an application such as this, it would also be important to test the capabilities of the server system by simulating a high volume of traffic. This type of testing is sometimes referred to as load testing. Using tools such as JMeter and SoapUI (for RESTful web-services), it would be possible to generate an artificial load for the server and then determine its capabilities. Although the required tools were investigated and acquired, this testing did not take place due to time constraints. However, it may not have been practical to use this type of testing because the server application is currently running on a standard desktop machine instead of a specialised server machine as would be the case in the production version.

9 PROJECT MANAGEMENT

Project Management is a vital and often underestimated component of software development, especially within development teams. Resources such as time and skills, need to be effectively distributed and assigned to tasks. Members need to be allocated tasks or sub-tasks (Team Management) and need to keep track of both the time spent on that particular aspect, as well as changes made to the source code (Version Control). Finally, the overall project needs to be managed and licensed adequately and effectively to maintain the professionalism of the software.

9.1 Team Management

The work of the project was divided initially amongst the five group members into two main sections: Android Application (client) and Remote Application Server. The following members were involved with the respective components:

Android Application (client):

A. Campbell
C. Epstein

Application Server:
High level web services and processing:

A. Paverd
D. Mankowitz

Database and Database Interface:

G. Favish

A large portion of the project was coded using the technique of pair programming [16]. This technique involves two programmers sharing one workstation, both contributing equally to the development of the same source code. The main programming pairs that worked together were:

A. Campbell and C. Epstein
A. Paverd and D. Mankowitz

G. Favish also paired with D. Mankowitz to integrate the database into the rest of the web server. It is believed that pair programming was beneficial to this project as it made navigating new technologies quicker and easier and reduced the time taken to find and fix bugs.

Tables 1 and 2 show the time management implemented by the group.

Table 1 : Showing the Estimated and Actual Time Spent on the Project

Group Member	Estimated time(hours)	Actual time (hours)
A. Campbell	31.5	46
C. Epstein	31.5	49
G. Favish	30	32
D. Mankowitz	40	50
A.Paverd	38	48
Total	171	225

9.2 Version Control

In order to manage the source code and configuration of the project, version control systems were used. This facilitated individual development but also ensured that changes were integrated back into the central project.

On the mobile application side, the two group members worked closely together following a pair-programming methodology. This meant that there was no need for extensive use of a version control system. However, in order to centralise and backup development work, the DropBox collaboration system was used [17]. The Dribble mobile application is not the standard type of project for which version control systems are designed, therefore the advantage of DropBox is that its simplicity avoided any major conflicts, and maintained an adequate backup of the system.

Table 2 : Showing the Estimated and Actual Times Allocated for each Sub-task within the Project

Sub-task	Estimated time(hours)	Actual time (hours)
Server		
Setting up Web-server	10	12
Web Services	16	18
Queues	6	10
Enterprise Beans	16	20
Processing Algorithms	8	8
Dataset Interface	4	6
Database Implementation	16	20
Integration	16	18
Testing	16	18
Total	108	130
Client		
Location-based Functionality	18.9	25
User Interface	18.9	40
Activity Implementation	15.75	20
Communications	6.3	6
Additional Functionality	3.15	4
Total	63	95

For the server-side development, the Git version control system was used [18]. Since Git is a distributed version control system, it is slightly more complex to use than traditional centralised equivalents such as Subversion. However, the significant advantages of Git are its flexibility and it ability to merge development branches without requiring a high degree of user involvement. These characteristics make Git a popular choice for many current projects. To allow collaborative development on this project, a central Git remote origin was created. This repository is hosted on the GitHub service at: http://github.com/ajpaverd/Dribble. This is an open-source repository which anyone can access to download the current source code. However, the ability to modify this project is restricted to the current group members. The use of this repository fulfills the requirement that the project source code should be available on a publicly accessible website.

9.3 Licensing

When code is released without a licence, the code is copyrighted by default [19]. This means that the author needs to be contacted directly to grant permission to use his/her code. It is possible to view the code but users have no legal right to use it. This defeats the purpose of releasing open-source code.

A number of open-source licences have been considered for this application. The General Public License (GPL) is considered to be a 'viral' license whereby derivative works of the code need to be released under the same licence condition [20]. This licence prevents the use of the code in non-open source programs and is thus restrictive on the commercial use of the code. This is not ideal as this application may be considered for commercial use. Another way of releasing the code is using the GNU Lesser or LGPL. This places copyleft restrictions on the code itself but does not apply these restrictions to software that merely links with the code [21]. This means that LGPL can be linked to non-(L)GPL programs regardless of whether the program is proprietary or free. However, there is still a restriction on the commercial distribution of the program itself and hence is non-ideal for this application.

Dribble may be marketed in the future and thus BSD is the most appropriate licence for this application. This licence allows developers to create proprietary forks [22]. A proprietary fork is a fork in the code where the new code is kept proprietary by the developer. This enables developers of the application the opportunity to develop code in a new fork and market the code at the same time. This is an open-source licence but provides opportunity to developers of the code.

10 CRITICAL ANALYSIS

10.1 Trade-offs

Some aspects of the project needed to be reconsidered to make it easier to implement and faster and easier to use. Initially, the ranking of Dribs was to be performed by a neural network, however this was deemed too complex to implement and calculations for ranking were eventually implemented on the server instead. Also, a

tag cloud was initially going to be used to represent DribSubjects on the mobile device and this was changed to use ordered lists which essentially perform the same task. To provide faster transmission, whole DribSubjects are not sent to the server but rather only the unique ID representing the DribSubject which enables less overhead over the network. Other trade-offs occurred when dealing with positioning. The radius around DribSubjects and Dribs is represented in meters, however latitude and longitude obtained from the mobile device are returned as micro-degrees, thus a conversion is necessary. An estimation of this conversion factor for South Africa was found to be approximately 8000 degrees per kilometer by trial and error. Also for simplicity, the area around Dribs and DribSubjects was also represented as a square and not circular. These factors combined gave an approximate area around the user.

Although the database fulfilled the requirements and operated effectively, it would greatly simplify the code if the database were an Object-Orientated Database. Instead of using SQL strings (which are prone to SQL Injection), the database would be accessed using methods to store and retrieve entire objects from the database[23]. The advantage of this approach is increased flexibility and extensibility (if more parameters are added to the object class, it should not affect the database structure)[24]. However object-orientated databases are less standard and cater for a smaller niche than normal relational databases. Further investigation into the intricacies of this type of database is required to determine its suitability for the Dribble application.

10.2 Recommendations

The project has potential for future work and there are many improvements that can be made. Currently, the application does not support having punctuation marks in Dribs as a result of a bug. This is an issue that would need to be resolved for future work as it limits the type of messages a user can send. Drib content can be refreshed manually by the user but if not performed for some time it could mean that the user may be viewing out-of-date information. It would therefore be better to use a thread that refreshes content automatically in the background at certain intervals. The chosen platform limits the number of users that can benefit from the application. The current application runs on one platform and one version only (Android v1.5). To grow the user base, it would be beneficial to port it to other platforms such as Nokia's Symbian or the Blackberry architecture, and to higher firmware versions. To handle a larger growth in users of the application, the Glassfish server would have to be deployed on a dedicated server machine or a cluster of machines that can deal with a large number of requests. The cluster would be necessary for efficient load balancing and to enable high availability of services as a result of its fault tolerance capabilities. The project was designed to be sustainable, has real-world applications and could be marketed and developed further to gain popularity amongst users with a main goal of eventually generating a profit. The application source is currently hosted on a central version control server as an open source project, however if the application were to make a profit, the project would need to be forked to another development branch which would have to be closed source to protect intellectual property.

11 CONCLUSION

In designing the Dribble application, a number of new technologies were investigated. This included Android development on an HTC Dream and server-side development using the JavaEE framework and a Glassfish application server. A standardised protocol was defined for sending and receiving messages and was implemented using XStream on the mobile side and RESTful web services on the server side. The mobile application consisted of a GUI complemented with a Google Maps overlay feature. The mobile application also uses threads to constantly update the users location. It included a debugger called LogCat and the application was tested on an emulator as well as on the HTC Dream. Similarly, the server, comprised of a multi-tiered JavaEE architecture, utilised queuing as well as synchronous queuing to communicate between the business and web tiers. The business tier, consisting of message-driven beans processed the objects and interacted with the database interface. The processing used complex algorithms to calculate popularity scores and assign unique IDs to messages and subjects. Both applications utilised version control and the project was completed on time as a result of efficient team management and pair

programming. Sufficient testing was conducted on both the client and server applications to show that the system operated as required. The applications were tested on multiple platforms and logging was implemented in both scenarios. The application has been released as an open-source project under the BSD license.

REFERENCES

[1] A. A. Gross R. "Information Revelation and Privacy in Online Social Networks." In ACM, editor, *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. New York, 2005.

[2] T. Economist. "A Special Report on Social Networking." *The Economist*, January 2010. URL http://www.economist.com/specialreports/displaystory.cfm?story_id=15351002. Last Accessed: 29 April 2010.

[3] T. A. Kumar R., Novak J. "Structure and evolution of online social networks." In ACM, editor, *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, 2006.

[4] Mashable. "Foursquare: Why It May Be the Next Twitter." URL http://mashable.com/2009/07/25/foursquare-app/. Last Accessed: 29 April 2010.

[5] Foursquare. "foursquare." URL http://foursquare.com. Last Accessed: 29 April 2010.

[6] H. van Vliet. *Software Engineering: Principles ans=d Practise*, chap. 12, pp. 333–334. Wiley, 2008.

[7] F. Ableson. "Develop Android applications with Eclipse.", Feb 2008. URL http://www.ibm.com/developerworks/edu/os-dw-os-eclipse-android.html. Last Accessed: 30 April 2010.

[8] "JEE framework." URL http://www.exforsys.com/tutorials/j2ee/j2ee-overview.html. Last Accessed 30 April 2010.

[9] "Application Server." URL http://www.bestpricecomputers.co.uk/glossary/application-server.htm. Last Accessed 29 April 2010.

[10] "Glassfish feature." URL http://weblogs.java.net/blog/arungupta/archive/2008/02/whats_the_big_d.html. Last Accessed 30 April 2010.

[11] S. Microsystems. "RESTful Web Services Developers Guide.", 2009.

[12] "RESTful Web Services - The basics." URL https://www.ibm.com/developerworks/webservices/library/ws-restful/. Last Accessed 29 April 2010.

[13] "JMS Overview." URL http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/overview.html#1027335. Last Accessed 29 April 2010.

[14] S. D. Network. "Enterprise JavaBeans Technology." URL http://java.sun.com/products/ejb/. Last Accessed: 30 April 2010.

[15] A. Derby. "Apache Derby." URL http://db.apache.org/derby/. Last Accessed: 30 April 2010.

[16] D. Wells. "Pair Programming.", 1999. URL http://www.extremeprogramming.org/rules/pair.html. Last Accessed: 29 April 2010.

[17] D. . Home. "DropBox." URL http://www.dropbox.com/. Last Accessed: 29 April 2010.

[18] Git. "Git Fast Version Control System." URL www.git-scm.com. Last Accessed: 29 April 2010.

[19] "Choosing the correct software license." URL http://www.codinghorror.com/blog/2007/04/pick-a-license-any-license.html. Last Accessed 30 April 2010.

[20] "HOWTO: Pick an open source license." URL http://blogs.zdnet.com/Burnette/?p=130. Last accessed 30 April 2010.

[21] "Preamble to the Gnu Lesser General Public License." URL http://opensource.franz.com/preamble.html. Last Accessed: 30 April 2010.

[22] "Choosing an open source license." URL http://www.airs.com/ian/essays/licensing/licensing.html. Last accessed 30 April 2010.

[23] B. Obasanjo. "An Exploration Of Object-oriennted Database Management System.", 2001. URL http://www.25hoursaday.com/WhyArentYouUsingAnOODBMS.html. Last Accessed: 30 April 2010.

[24] M. D. Zdonik, S. B. *Object-oriennted Database Systems*. 1990.

Appendix A - Screenshots

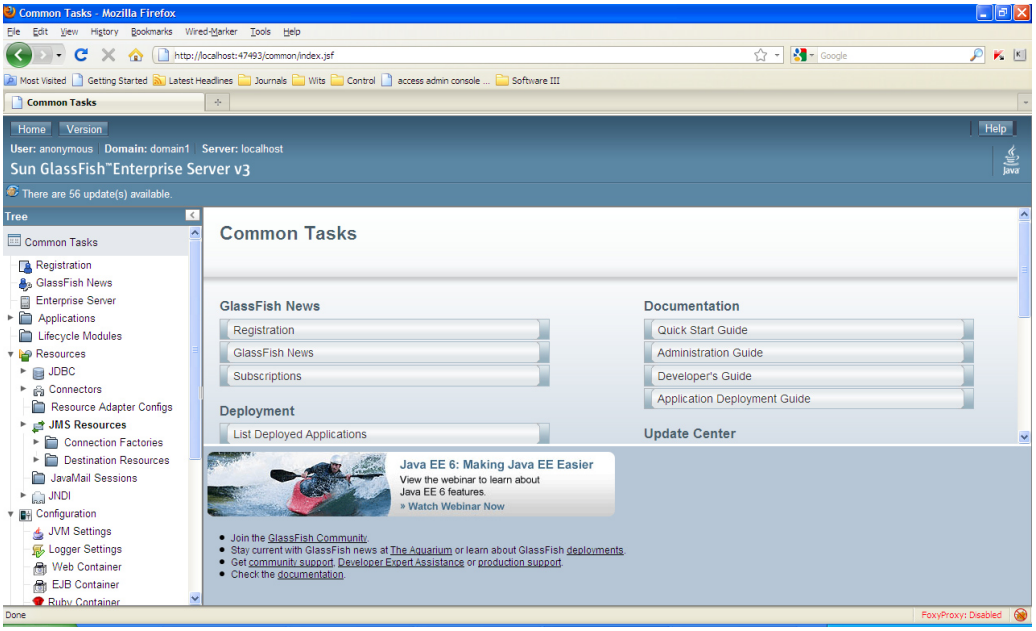

Figure 8 : Glassfish Admin console

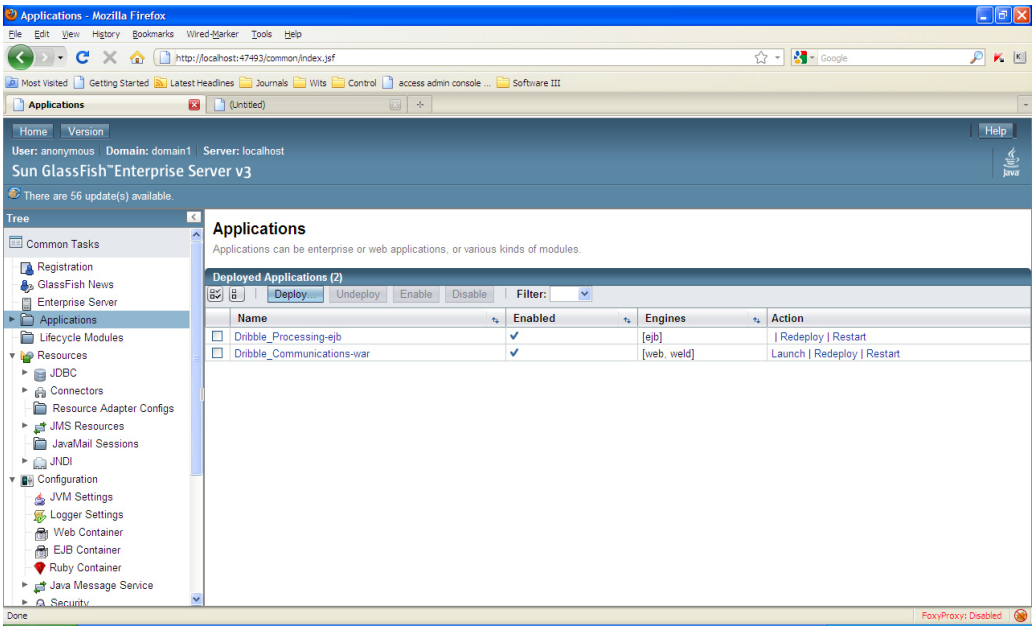

Figure 9 : Glassfish Applications

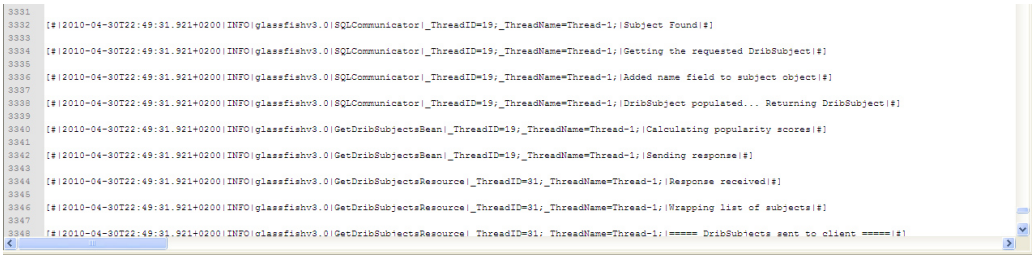

Figure 10 : Glassfish Logging

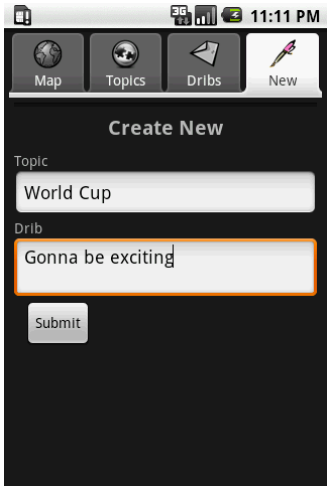

Figure 11 : Android Create

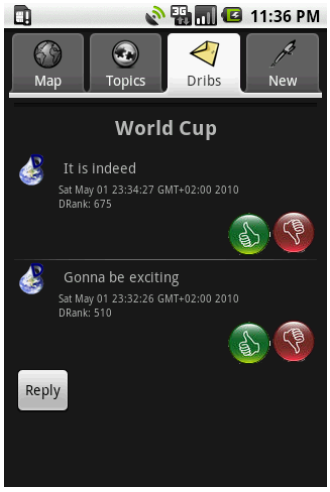

Figure 12 : Android Dribs

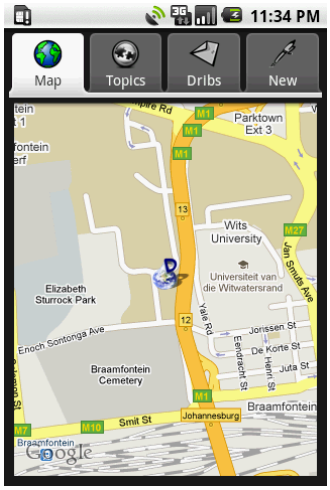

Figure 13 : Android Map

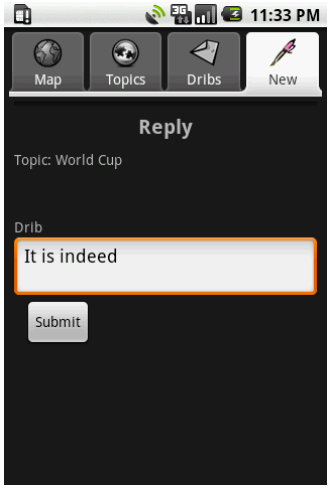

Figure 14 : Android Reply

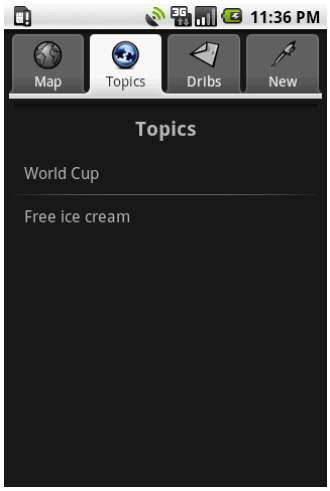

Figure 15 : Android Subjects