Name of Document	Design Document Milestone 3: Extended Storage Server
Date	9th of March, 2014
Group	048
Project Manager	Ted Nolan
Tutorial Section	0107
Team Members	Tanveer Fuad 1000056887, Vivek Mogalapalli 999448209, Avishek Bose 999738440.

Table of Contents

<u>List of changes made to Milestone 2</u>

1. Introduction:

- Modified the first paragraph to clarify what the sources were referring to, and to remove the "not receiving information" claim so, as to address a comment made by the CI
- Discussed the current state of the situation to follow a Situation-Problem-Solution structure
- Added a paragraph to discuss the current state-of-the-art
- Changed loose sentence structure here and there, as asked for

2.1 UML Component Diagram

- Added a new paragraph for introduction to the diagram

2.2 UML Sequence Diagram

- Added a new paragraph for introduction to the diagram
- Modified the UML sequence diagram description to make it system centric
- Modified the UML sequence diagram

3.1 Functions

- Modified opening paragraph as per comments so it does not contain irrelevant information
- Added the new Query function to the list of functions

3.2 Objectives

- Modified the opening paragraph to show how two different sets of objectives were generated
- Modified the first user objective to provide a better metric
- Modified the second user objective to give details on the proposed metric as was suggested in Milestone 2

3.3 Constraints

- Modified the introduction to the list of constraints

4.1 Decision on a Data Structure

- Modified to incorporate a lot of the important decision making criteria, from the Appendix

4.2 Communication Protocol

- Reorganized the entire the description
- Rewrote and added new content about the data exchange format

4.3 Connection Status

- Revised the entire description to include objective satisfied by the system
- Also, wrote about the choices available

4.4 Error Handling

- Removed all the error handling procedures given in the Design Specifications to the Appendices, keeping only the new considerations in the main body

6.0 Conclusion

- Rewritten second part of the paragraph so that the point made in the first part is carried through

Executive Summary

Storage servers make the back end of many services and when used effectively, it can greatly enhance the efficiency of not only sharing information but its consumption as well. The client is the University of Toronto where there exists a lack of a definitive way to communicate information regarding University events using a storage server. Currently, information regarding events are communicated via e-mails, the university website or other various social networks. Consequently, there is a no definitive way to communicate information as it is impossible to gauge the successes of these methods.

To start off, the key objectives for this project have been separated into user and coding objectives as each highlight a different part of the design process. The main user objectives include having an intuitive user interface and having high efficiency such that there is no noticeable lag during run time. On the other hand key coding objectives include having a system which is extensible and having an archiving capacity of all records for up to one year. Furthermore, the software system shall be written in the C programming language, work on Red Hat Linux machines, comply with all patent regulations and finally maintain the privacy of its users.

To ensure that the final product meets the needs of all stakeholders, important design decisions were taken. The data structure used is a Hash Table as it has the lowest run time for retrieving and storing information, thus addressing the first user objective. The team also decided on a custom data exchange format to handle client/server interactions as it guarantees that the design would be tailor suited to its function. Moreover, the team chose to use Lex and Yacc parsing tools to provide an interface with efficient error handling and ease of usage. Additional unit tests will also be implemented to verify that the new program works according to specifications, thus ensuring the satisfaction of the client .

The implementation of the storage server will allow users to create, view and register for events all through one unified service which may later be delivered as a mobile application. This system will completely eliminate the fragmentation of information that currently exists by improving on past methods by allowing each user to have their own profile. Upon submission of this document, the team will work on finalizing the code to provide functionality for storage of multiple columns and data types.

1.0 Introduction

The University of Toronto currently has over 750 registered clubs [1]. Each of these clubs organise multiple events [2][3] a month and publish information about these events on their own websites [4][5]. As a result, members of the University, who are registered in many organizations, have to go to various sources to find information regarding club events. This leads to fragmentation of information and burdens many members in spending more time to go through the different sources to find related information for all the events that they are interested in.

The prevailing situation consists of individual systems that provide information regarding events through various websites in the likes of Skule Digest and Faculty webpages, each of which posts information related to specific fields, e.g Skule Digest only posts the major engineering event details [4], and Faculty webpages only post events and news from their own departments [5]. Therefore, a student with various non-related interests will have difficulty in surfing through all these pages to gather information.

The solution to this problem is to create a server that will maintain a **unified events calendar** for all the organizations in the University of Toronto, and will interact with members in the form of a mobile application. This application will benefit the students, as they can then conveniently find all pertinent information from a single source, and as well as organizations, as they no longer have to post and advertise their events in various places. Since this will allow for more focused advertising by organizations, this application is likely increase the students turnout to events and attract more members to the organizations.

The current state-of-the art, upon which Group#048 will benchmark this solution on, is a similar event organisation service provided by Eventbrite [6]. Eventbrite allows its users to access details of upcoming events, and also save the events either on their cellphone or their browser. It also facilitates event hosts to promote their events and set up online ticket sales for their events [7].

The server that Group# 048 is going to design will cater to the problem by storing electronic records using hash tables [8]. The server shall contain two distinct types of tables, of which, the first will be a user profile table where every user, members of the University with a valid UtorID, shall put in information about themselves, and the second will be an events detail page, where the host shall upload details about the event they will organise. The two tables will be linked by an “interest” field, which will allow the server to match users with relevant events.

The design, in addition to achieving the above functionalities, should also meet certain other objectives such as achieving minimum information storage and retrieval time, and providing the user with an interactive user interface. Overall, the new design is envisioned as a mobile application which will create a unified location for accessing event information from all the clubs in the University of Toronto.

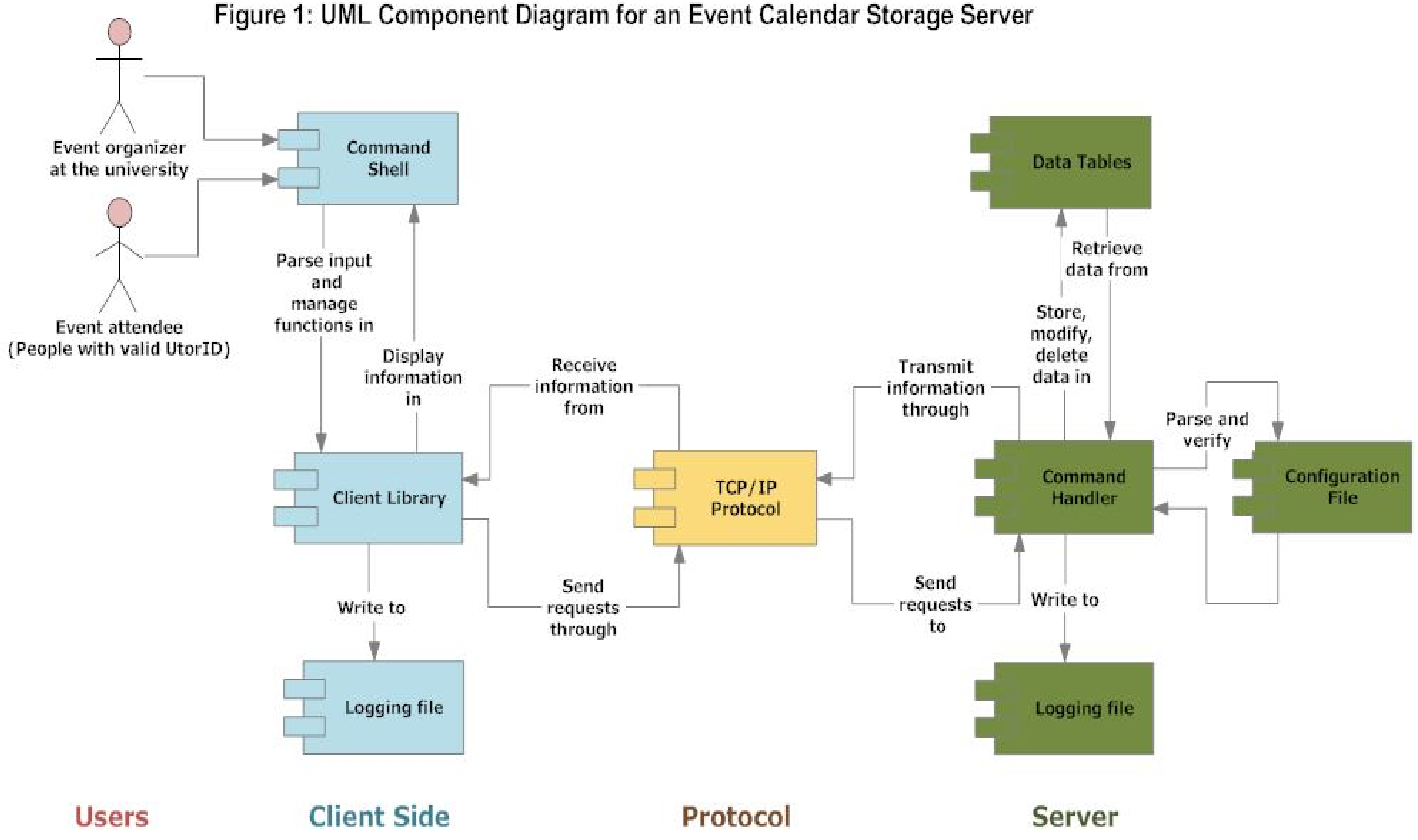
2.0 Software Architecture

2.1 UML Component Diagram

The figure below is a component diagram, which is a diagram that describes the organization of components within a system [9]. In the diagram below, every component is a physical building block of the storage server and is represented as a rectangle with tabs [9]. Arrows are used to show the dependencies between various components of the storage server [9].

As the diagram below shows, the unified event calendar storage server for the University of Toronto consists of 4 main components. They are the users, a client side, a transmission protocol, and a server. The users of the calendar are the event organizers who can create a new event and event attendees who can access details of upcoming events. These users interact with the client side of the system through a command shell which communicates with a client library. The client library creates a socket to communicate with the Transmission Control Protocol (TCP/IP) [10][11]. The server communicates with the protocol through a socket created by the command handler [10] after verifying the configuration file. In brief, the command handler manages the events and user data stored in the data tables. Additionally, the command handler along with the client library uses a logging file to record user activity.

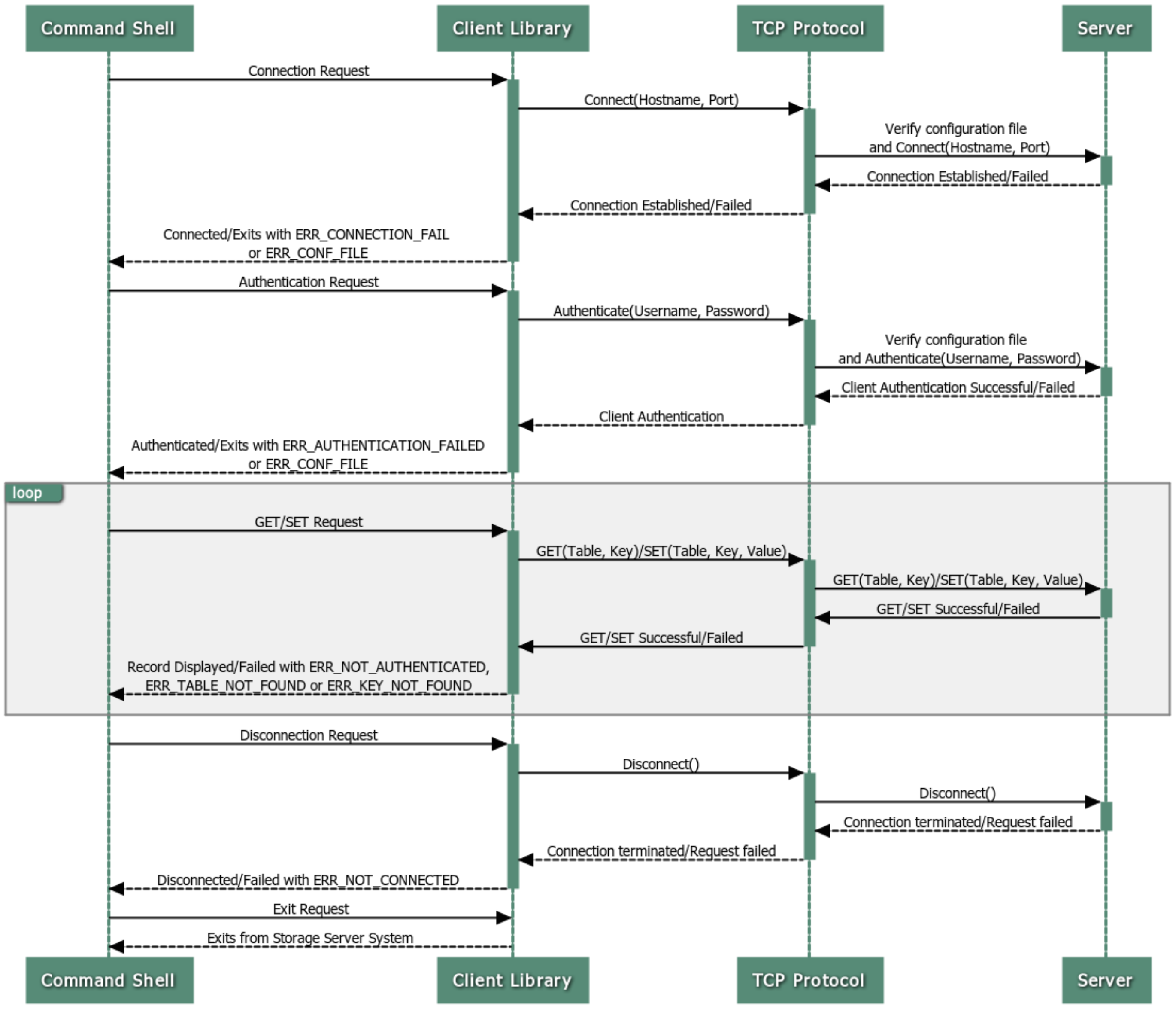
Figure 1: UML Component Diagram for an Event Calendar Storage Server



2.2 UML Sequence Diagram

The figure below is a sequence diagram, which is a diagram that shows the interaction among classes in terms of exchange of messages [12]. In the diagram below, the solid-line arrow represents a message sent by an object and the dotted-line arrow represents the value returned by an object [13].

As the diagram below shows, the interaction between a user and the events calendar storage server is through the command shell. The command shell provides the users with a list of various options available within the system [10]. Initially, the system creates a connection between the client side and server side by processing a connection request from the command shell. Upon a authenticate request, the system authenticates the connection by validating user credentials [10]. As shown by the loop in the diagram, authentication allows the storage server to process multiple GET/SET requests to store, retrieve, update, and delete data without the need to reauthenticate [10]. The connection is terminated by the system when it receives a disconnect request [10]. The system then allows a user to exit the command shell effectively ending the interaction with the system [10].



Command Shell
Client Library
TCP Protocol
Server
Connection Request
Connect(Hostname, Port)
Verify configuration file and Connect(Hostname, Port)
Connection Established/Failed
Connection Established/Failed
Connected/Exits with ERR_CONNECTION_FAIL or ERR_CONF_FILE
Authentication Request
Authenticate(Username, Password)
Verify configuration file and Authenticate(Username, Password)
Client Authentication Successful/Failed
Client Authentication
Authenticated/Exits with ERR_AUTHENTICATION_FAILED or ERR_CONF_FILE
loop
GET/SET Request
GET(Table, Key)/SET(Table, Key, Value)
GET(Table, Key)/SET(Table, Key, Value)
GET/SET Successful/Failed
GET/SET Successful/Failed
Record Displayed/Failed with ERR_NOT_AUTHENTICATED, ERR_TABLE_NOT_FOUND or ERR_KEY_NOT_FOUND
Disconnection Request
Disconnect()
Disconnect()
Connection terminated/Request failed
Connection terminated/Request failed
Disconnected/Failed with ERR_NOT_CONNECTED
Exit Request
Exits from Storage Server System
Command Shell
Client Library
TCP Protocol
Server

3.0 System Requirements

3.1 Functions

The storage server is a standalone program that can accept and process commands on the front end while retrieve or store information in the form of digital memory (data) on the back end. The list of functions below were provided to the team by the detailed specifications section of the course website [10]. Upon the input of appropriate commands, the software system shall:

1. Establish a connection from a client side to the server with a specified hostname and TCP port number
2. Authenticate a session by accepting the correct username and password from the user
3. Accept and store digital information in tables as records in main memory
4. Retrieve information as a digital record from the storage server
5. Update information which are digital records in the storage server
6. Delete information in the form of a digital record from the storage server permanently
7. Query the table for all records which match the predicates specified
8. Encrypt information sent from the client side in the server's database
9. Terminate the connection from the server to the client

3.2 Objectives

The objectives are a set of measurable goals that define what a final design should be. The team decided on two sets of objectives, namely user and coding objectives. User objectives are identified as the goals for the user, whereas coding objectives are the ones from the programmer's perspectives. The separation of objectives was done because each set of objectives relate to different parts of the design process. The following is a list of objectives, ranked according to importance:

User Objectives:

1. **Intuitive user interface**: The design should provide the user with an intuitive and self-explanatory user interface. On a System Usability Scale [14], this interface should have a **score of no less than 68**, which is standard for a good interface according to this scale. However, the fulfilment of this objective is out of the scope of this milestone.
2. **Efficiency**: The lesser the execution time of a server is, the faster it is at inserting and retrieving information. The server should, therefore, try to achieve a **minimum execution time**. After implementation of milestone 3, the system should have an end-to-end processing time of no more than 40 ms for a GET/SET request in a database with 7000 records, as determined by the performance evaluation report for milestone 2 [15].

Coding Objectives:

1. **Duplicates**: The server should identify if the user is trying to enter a record of an event that already exists in the server, and not allow that record to be entered. At the end, the duplication shall be measured using the formula: **Duplication Detection= No. of duplicates accurately detected/ Total number of duplicates in the system**. The higher the ratio, the better the system is at detecting duplicates.
2. **Extensible**: The server should be coded in a such a way so as to **minimize the time taken**, in human hours, to add new features to the system [16].
3. **Archiving capacity**: The server should have the capacity to store records of up to **one academic year**

at the University of Toronto, since clubs will have new members every year and there will be new user profiles [17].

4. **Coding standards**: The coding design of the server should follow **standard coding conventions**, e.g indent-style conventions, naming conventions, etc. to enable a system developer to easily understand the code for systematic debugging [18] and extension purposes. In addition, the system should use a Lex and Yacc parser [27] as opposed to a custom-made one (discussed in Section 4.5).

3.3 Constraints

Constraints are the requirements which the design shall meet. The constraints include those mentioned in the design specifications and additional ones specific to the event calendar. The software system shall:

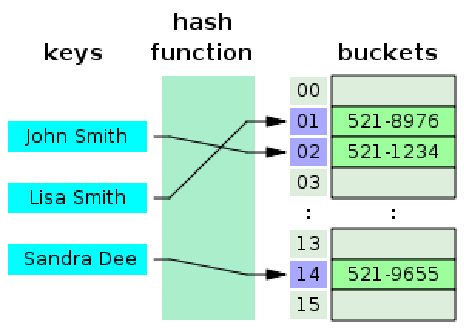
1. Contain code written in C [10]
2. Comply with all patents [19]
3. Function on Red Hat Linux machines [10]
4. Work only if the prerequisites of functionalities are met, e.g. a user shall not be able to authenticate before connecting to the server
5. Not modify the file storage.h [10]
6. Abide by the Freedom of Information and Protection of Privacy Act of the University [20] to maintain the privacy of its users

4.0 Design Decisions

4.1 Decision of a data structure

The data structure that the team has decided to use for the implementation of the server is a **Hash Table.** As seen in the figure below, this particular data structure has two components: an array to store the table of event details and user profiles, and a hash function that assigns indexes for input data [8]. Thus, all the records will be indexed according to the number given to the record by this function(See Figure above). However, this means a record can sometimes be assigned the same "bucket" in the array. To tackle this, a linked list will be implemented at every bucket of the array, where a new addition can be inserted at the end of the linked list [21].

Figure 3: A Hash Table [22]



The criteria which the team used to assess the different structures are execution time for data handling (time complexity), memory usage (space complexity), and ease of coding. The team decided that, since retrieval and insertion of records and having access to them swiftly is an important issue, having a faster execution time is of prime importance. The team ranked each of the objectives against each other to prioritize them accordingly [Appendix A].

A list of alternatives were generated which included a sorted linked list, a sorted array, a binary search tree, and a hash table with a linked list. The alternatives were weighed against each other and the Hash Tables proved to be the optimal data structure due to its faster access time and relative ease of coding. Since, execution time was a major user objective, the decision making process for this criteria is stated below.

Execution time is measured in the Big-O notation which is a measure of time as a function of the number of records into the system. After analysing the complexities of different data structures, it can be seen below that Hash tables have an advantage in both retrieving and inserting execution time.

Table 2: Comparison of Complexity of various data structures [23]

	Array	**Sorted Linked list**	**Hash table**	**Binary Search tree**
Insert	O(n)	O(1)	**O(1)**	O(log(n))
Search	O(n)	O(n)	**O(1)**	O(log(n))

Keys:

- "n" represents the number of data records
- log(n) means log to the base 10 of n which is smaller than n [24]. O(1) is the fastest of all the complexities [24].

More details of the decision making procedure can be found in Appendix A.

4.2 Communication Protocol

4.2.1 Basic Description of a Communication Protocol

A communication protocol is a set of rules for data exchange between different network systems [25]. The Transmission Control Protocol (TCP) used for this system works in collaboration with IP (Internet Protocol) to communicate between systems by sending data packets through sockets [10][11]. These data packets would contain various messages encoded in a particular format which will be used for transmitting messages through TCP for authentication, error handling and data exchange. Therefore, communication between systems requires that they agree on the format of data exchange [25].

4.2.2 Deciding on a Data Exchange Format

After evaluation on the tradeoffs between different data exchange formats like JSON and XML, the team has decided to develop a custom format for information exchange between the client side and server [26]. This data exchange format would be specially designed for the event calendar storage server and hence, it would allow the team to meet the constraints and objectives of developing a fast and secure system. The structure of the custom data format has two main components: Response Type and Message . The Response Type refers to various types of responses like Connect, Authenticate, GET, SET, Query and Error. The Message contains the response of the client side or the server side. A snippet for a Query request is

[26]: “Query”, “Marks > 90”.
Additional examples of the data format and a detailed analysis about its various advantages over alternatives can be found in Appendix C and Appendix D.

4.3 Connection Status

An authenticated connection between the client side and server of the system ensures that the system is secure and complies with privacy regulations. Therefore, a connection would be considered active only after a user authenticates his/her connection. Before processing an authentication request, the username and password of the user is verified and a unique ID is created and assigned to this connection. This unique ID is known as a session ID [10]. Every session ID expires within 2 hours after which the user has to re-authenticate the connection for additional security. The connection is reused for multiple GET/SET requests but would be automatically terminated once the session ID expires or once the user chooses to disconnect. Another possible secure system would be one that asks a user to re-authenticate the connection after every GET/SET request. The team, however, did not prefer this approach as it would not meet our objective of having a fast and efficient system.

The system would handle multiple clients by prioritizing its clients (will be implemented in future milestones). The event organizers would be given a higher priority over event attendees in situation of multiple requests. Therefore, this means that the connection request for an event host would be processed before that of an event attendee. In this case, the event attendee would be notified and placed on a waiting list. A user on the waiting list would be automatically connected once other users have left and would not be required to re-authenticate.

4.4 Error Handling

The error handling procedure for some of the cases has been specified in the instructions documentation. This can be found in Appendix B. In addition, the team decided on a few other error handling procedures, which is documented below. The team has a specific error regarding invocation of a DISCONNECT request without first invoking a CONNECT request, so that the user does not try to disconnect from a NULL connection. In addition, since information from the configuration file contains important information about connection, authentication and the data structure, the team decided to add an error specifically denoting a faulty configuration file. The following table describes the response by the system when incorrect parameters, other than the ones specified in the document, are fed:

Table 1 : Error Handling procedures

Possible incorrect action	Response of the system
The user invokes disconnect without first connecting	1. Error: **ERR_NOT_CONNECTED** 2. A list of possible actions are printed
The user does not specify a configuration file or it is incorrect	1. Error: **ERR_CONF_FILE** 2. The command shell will exit

4.5 Parsing Algorithms

Within computational linguistics, parsing refers to the formal analysis by a computer of a sentence or other string of words into its constituent elements. Parsing can be done in a variety of ways the most prominent of which are using a custom parser for a set of very specialized inputs or using a more robust built in parser generators provided by the UNIX system such as Yacc (Yet another Compiler Compiler) [27]. When considering parsing algorithms the team evaluated the advantages and disadvantages of the methods mentioned with the primary focus being on the long term effects of the system beyond this milestone. The main advantages of using any kind of lexer/parser generator is the amount of flexibility provided if the language (input) evolves. Furthermore, Yacc parsers have reliable error handling with numerous documentation available online for support. Consequently, the team has decided to implement the Lex and Yacc parser provided by the UNIX operating system as opposed to a custom parser as this best meets the objectives of having a more extensible system and following good coding standards. Lex or a lexical analyser will break up the input stream into usable elements or will identify useful strings in a text file while the Yacc parser analyzes the validity of the input [27]. Thus, by combining the UNIX tools Lex and Yacc, a very efficient and robust parser can be created.

4.6 Modification from New Requirements

To implement the new requirements for the storage server, changes have to be made to the previous code. The first major feature of the new storage server is the support for tables with multiple columns containing different types of values. This would require the modification of the configuration file parser that reads column names and the record data structure that contains information stored in the server. The other major feature of the new storage server is a function called query, which would allow users to access all values satisfying a particular criteria. Its implementation would require changes to client and server parsers along with the client library. Therefore, the addition of the new features would cause the storage server to be neither backward or forward compatible.

4.7 Unit Tests

Unit tests are short program modules that check if individual sections, units, of a large program are fully functional [28]. To run the unit tests, the team decided to break the code down to a few testable sections: the server side, the data structure, and the configuration file parser.

First, the team has decided to implement the server on the skeleton code provided and run tests to check the functionality of the server. Simultaneously, the team will create the data structure as a completely separate program, independent of the server and client library, and test the following features of the structure: GET, SET, DELETE, and QUERY. Once, these sections are fully functional, the team will merge the two sections and incorporate the configuration file and its parser. This will ensure that the program, as a whole, is fully functional. A detailed list of the tests run on the system and their methodology can be found in **Appendix E**.

5.0 Use case

5.1 Primary Actor

The system that is being designed allows for the primary actor to interact with the storage server using a series of intercommunicating components. The primary actor for our system are event attendees and event hosts who can generally be thought of anyone within UofT who has a valid UTORid. The primary actor will be able to browse, join and manage a complete list of events that are a provided by the University of Toronto.

5.2 Level

The primary actors in the case are both the event attendees and event hosts from organizations in the University. The attendees will be able to keep track of which events they are specifically subscribed to and interested in, and the clubs will be able to keep track of who and how many are attending the events. Since it serves a different purpose for our primary actors, the design follows a *Summary level*.

5.3 Stakeholders and Interests

The following is the list of stakeholders generated along with their interests that would be affected by the design of the software system.

1. Event Attendees (People with valid UTORID's)
   - Interested in gaining information about school events
   - Interested in reducing the amount of effort/time in looking for information
2. Event Hosts
   - Interested in advertising their events to the UofT public
   - Interested in knowing the amount of people attending their events
3. University of Toronto
   - Provide funding only with proof of fiscal viability [29]
   - Have a system that is durable and will be operational longer than the time it takes to reimburse the capital used to implement
   - Streamline the delivery of information to students and staff

5.4 Precondition

The primary actors of the system must have valid UTORids which will be used to authenticate them into the system, after which the actor will be allowed access to information on the server.

5.5 Minimal Guarantee:

The storage server will provide conditions to ensure the primary actor's information are protected.In the case where the server may crash prior to the completion of storing, modifying or deleting a patient record, automatic logging will temporarily save this information until the primary actor manually chooses to save the updated information. After the storage server is operational again the primary actor will have to re login and authenticate to continue. Furthermore, if there is a period of no activity greater than 2 hours the server will automatically close the connection to protect sensitive information; the primary actor may retrieve this data by restarting the connection.

5.6 Success Guarantee

As a success guarantee, the storage server will provide its users with messages to confirm that user goals have been accomplished. For example, the goal of an event host adding a new event to the server, success will be indicated upon establishing a connection, upon providing correct authentication information, and upon correctly inputting event information into the database. In addition, the event host's successful entry will be shown in a message in the command shell or in a file.Ex

5.7 Main Success Scenario

The main success scenario of the system would be as follows:

1. The primary actor connects to the server by entering connection information.
2. The primary actor provides authentication information. The server verifies the information.
3. The event attendee searches the server for a specific event information. The attendee requests the server to modify his user profile with the event information, i.e add this event to his profile.
4. The system validates the request and modifies the user profile.
5. The system sends a message to the user verifying accomplishment of the task.

5.8 Extension

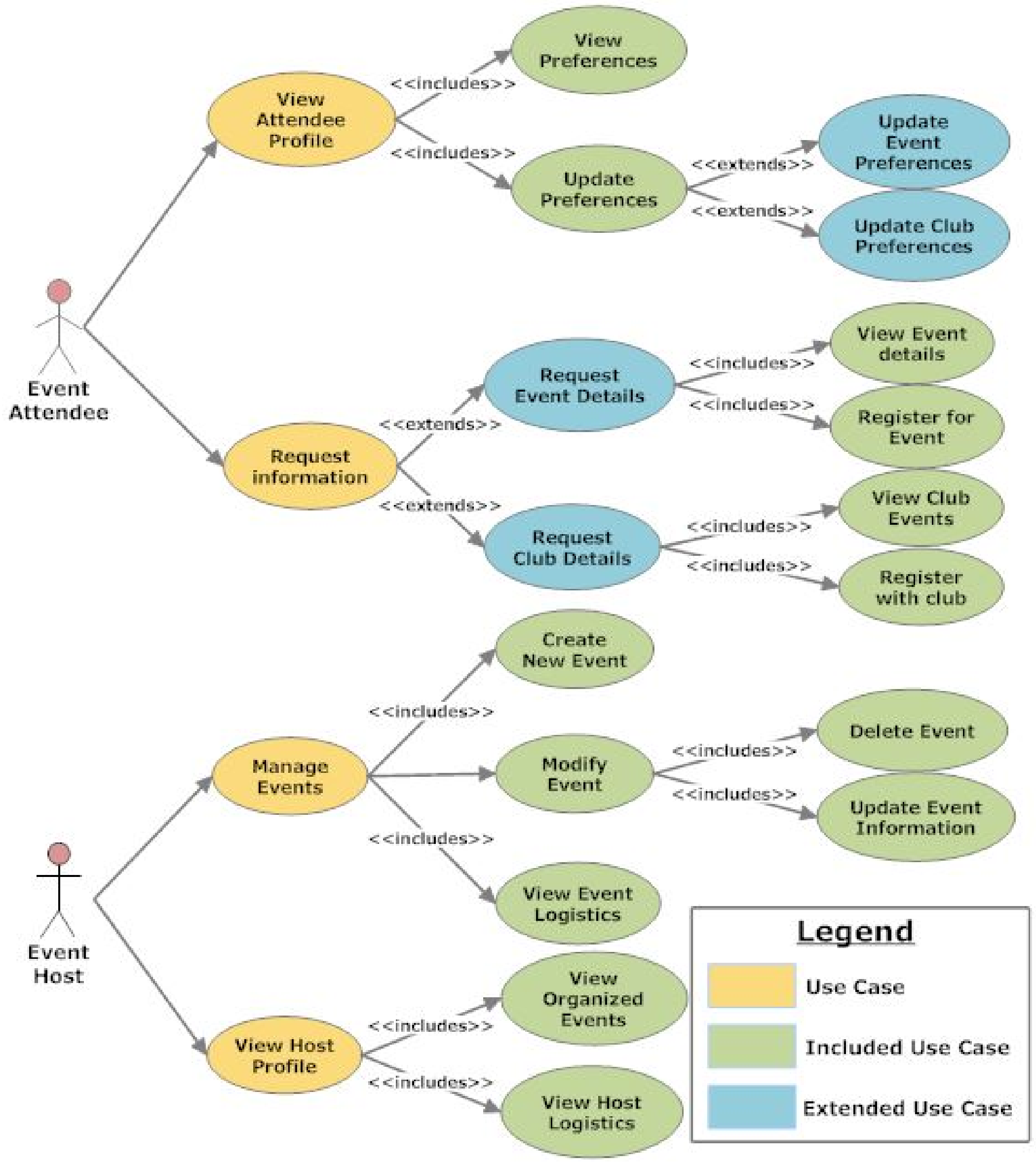
Extensions are a list of possible outcomes if the main success scenario fails. The following list describes the possible extensions to the main success scenario :

1. Actor inputs incorrect port name or port number. The actor, in this case, cannot connect to the server.
2. Actor inputs incorrect authentication information. The server shuts down shell and does not allow user access to information.
3. Actor asks to retrieve information which does not exist. The server lets the user know of the error and shuts down.
4. Actor tries to enter information into a table that does not exist. The server sends error feedback to the host and shuts down

5.9 Use Case Diagram

The figure below is a use case diagram, which is a graphical depiction of a user's interaction with the system [30]. In the diagram, the <<includes>> arrow is a relationship directed towards a use case that is included in the base use case [31]. The <<extends>> arrow is a relationship directed towards a use case that extends the behaviour of the base use case [32]. The use case diagram below shows the possible uses of the unified event calendar system for an event host and attendee. The diagram assumes that the user is logged into the system.

Figure: Use Case Diagram for a unified event calendar



6.0 Conclusion

The University of Toronto has a myriad of different clubs which range in size from a couple of members to ones which encompass a large section of the student populace. However there is a lack of organized information as each club has its own unique method to communicate with its members. Furthermore, the need for a unified media outlet which organizes information in a user friendly manner is evident and thus, team# 048 has proposed a software system which uses hash tables to store digital records. Also, the system now supports a query function allowing users a better search functionality and on the back end it uses the Lexx and Yacc parsers to ensure excellent input recognition and error handling. Thus, the system would greatly enhance the status quo by allowing event organizers to create, update and delete event information rapidly and in one common location. The project currently is right on track for its next stage which is to allow for the design of multiple clients simultaneously.

7.0 List of References

[1] Ulife. (2014). *Recognised Campus Groups* [Online]. Accessed February 4, 2014. Available: https://ulife.utoronto.ca/organizations/list

[2] The Faculty Club, University of Toronto. (2014). *Events Calendar* [Online]. Accessed February 4, 2014. Available: http://www.facultyclub.utoronto.ca/About-The-Club/Events-Calendar.aspx

[3] UTRA. (2014). *Events*[Online]. Accessed February 4, 2014. Available: http://www.utra.ca/events

[4] Skule. (2014). *Skule announcements[*Online]. Accessed February 4, 2014. Available: http://digest.skule.ca/

[5] Faculty of Applied Science and Engineering. (2014). *Events Calendar[*Online]. Accessed February 4, 2014. Available: http://www.engineering.utoronto.ca/Page4.aspx

[6] Eventbrite. (2014). *At the door* [Online]. Accessed March 8, 2014. Available: http://www.eventbrite.com/atthedoor/

[7] Eventbrite. (2014). *How Eventbrite works for organising events* [Online]. Accessed March 8, 2014. Available: http://www.eventbrite.com/features/

[8] Sparknotes. (2013). *What is a Hashing Table?[*Online]. Accessed February 4, 2014. Available: http://www.sparknotes.com/cs/searching/hashtables/section1.html

[9] SmartDraw (2014). *Software Design Tutorials - Component Diagram* [Online]. Accessed March 8, 2014. Available: http://www.smartdraw.com/resources/tutorials/uml-component-diagram/#/resources/tutorials/UML-Component-Diagram

[10] ECE297. (January 28, 2014). *Milestone 2 Detailed Specifications* [Online]. Accessed January 28, 2014. Available: https://sites.google.com/a/msrg.utoronto.ca/ece297/assignment-2/detailed-specifications#TOC-Suggested-Development-Plan

[11] jGuru. (February 15, 2000). *What is TCP and how does it work?* [Online]. Accessed February 8, 2014. Available: http://www.jguru.com/faq/view.jsp?EID=14663

[12] SmartDraw (2014). *Software Design Tutorials - Sequence Diagram* [Online]. Accessed March 8, 2014. Available:

http://www.smartdraw.com/resources/tutorials/uml-component-diagram/#/resources/tutorials/UML-Sequence-Diagram

[13] Trace Modeler (2008). *A Quick Introduction to UML Sequence Diagrams* [Online]. Accessed March 8, 2014. Available: http://www.tracemodeler.com/articles/a_quick_introduction_to_uml_sequence_diagrams/

[14] Measuring Usability. (February 2, 2011). *Measuring Usability with the System Usability Scale(SUS)* [Online]. Accessed March 7, 2014. Available: http://www.measuringusability.com/sus.php

[15] Tanveer Fuad, Avishek Bose, Vivek Mogalapalli. (February 15, 2014). *Milestone 2 Performance Evaluation Report* [Online]. Accessed March 7, 2014. Available: https://www.dropbox.com/s/muds308vml61w3a/EvaluationReport.pdf

[16] Eprints. (May 2, 1997). *An Analysis of OO Software Metrics* [Online]. Accessed February 9, 2014. Available: http://eprints.dcs.warwick.ac.uk/1438/1/cs-rr-324.pdf

[17] Teaching at University of Toronto. (2013). *The Academic Year* [Online]. Accessed February 9, 2014. Available: http://www.teaching.utoronto.ca/teaching/essentialinformation/workinguoft/academicyear.htm

[18] Kernel. (2002). *Linux kernel coding style* [Online]. Accessed February 9, 2014. Available: https://www.kernel.org/doc/Documentation/CodingStyle

[19] eHow. (2014). *What Happens When You Violate a Patent?* [Online]. Accessed February 4, 2014. Available: http://www.ehow.com/facts_6925624_happens-violate-patent_.html

[20] University of Toronto. (June 10, 2006). *FIPPA and its Application to the University of Toronto* [Online]. Accessed February 9, 2014. Available: http://www.fippa.utoronto.ca/about.htm

[21] National Institute of Standards and Technology. (26 June 2013). *Linked list* [Online]. Accessed February 8, 2014. Available: http://xlinux.nist.gov/dads//HTML/linkedList.html

[22] Wikipedia. (2014). *Hash Table* [Online]. Accessed February 7, 2014. Available:http://simple.wikipedia.org/wiki/Hash_table

[23] Big-O Cheat Sheet. (2014). *Know Thy Complexities* [Online]. Accessed February 8, 2014. Available: http://bigocheatsheet.com/

[24] Amit Sharma. (2014). *BIGO_GRAPH* [Online]. Accessed February 4, 2014. Available: http://i287.photobucket.com/albums/ll134/amit_9b/BIGO_GRAPH.jpg

[25] webopedia. (2014). *communications protocol* [Online]. Accessed February 8, 2014. Available: http://www.webopedia.com/TERM/C/communications_protocol.html

[26] w3resource. (2004). *JSON Tutorial* [Online]. Accessed February 9, 2014. Available: http://www.w3resource.com/JSON/introduction.php

[27] "What is Lex? What is Yacc?," . Accessed March 9, 2014. Available: http://luv.asn.au/overheads/lex_yacc/index.html

[28] Agile Alliance. (2013). *Unit Testing[*Online]. Accessed March 8, 2014. Available: http://guide.agilealliance.org/guide/unittest.html

[29] Joanne Fritz. *What Nonprofits Need to Know About the Zone of Insolvency* [Online].Accessed March 6, 2014. Available: http://nonprofit.about.com/od/nonprofitmanagement/a/insolvent.htm

[30] WhatIs.com (March 2013). *Use Case DIagram* [Online]. Accessed March 9, 2014. Available: http://whatis.techtarget.com/definition/use-case-diagram

[31] UML Diagrams (2014). *UML Use Case Include* [Online]. Accessed March 8, 2014. Available: http://www.uml-diagrams.org/use-case-include.html

[32] Andrew.cmo (2014). *UML Use Case Diagrams: Tips and FAQ* [Online]. Accessed March 8, 2014. Available: https://www.andrew.cmu.edu/course/90-754/umlucdfaq.html

[33] Rogue Wave. (2014). *Common Data Structures* [Online]. Accessed February 8, 2014. Available: http://www.roguewave.com/portals/0/products/threadspotter/docs/2012.1/linux/manual_html/ch05s05.html

[34] Udemy. (August 16, 2013). *JSON vs XML: How JSON Is Superior To XML* [Online]. Accessed February 8, 2014. Available: https://www.udemy.com/blog/json-vs-xml/

7.0 Appendices

Appendix A: Data Structure Decisions

The alternatives that the team generated include a full sorted linked list, a binary search tree, an array and a hash table. A linked list is a structure where a new record is linked to the tail of the last record, a binary tree is a structure where records are laid out in nodes, where each node can contain at most two other nodes, one with a larger index and one with a smaller index.

The following is a weighted decision matrix that shows results of the comparison of the data structure criteria:

Table 3: Weighted decision matrix (Ratings given from 1 to 5)

Criteria	Array	Linked List	Hash Table	Binary Search tree
Complexity of search (40%)	2	2	5	3
Complexity of insert (30)%	2	5	5	3
Memory (20%) [33]	3	4	3	4
Coding complexity (10%)	5	4	4	2
Score	2.5	3.5	**4.5**	3.1

Selected Data Structure**: Hash Table**

Appendix B: Error Handling Procedures

The following table describes the response by the server when incorrect parameters(the ones specified in the document) are fed [10]:

Table 1: Error handling procedures [10]

Possible incorrect action	Response of the server
The user does not enter in the correct hostname and port number	1. Error: **ERR_CONNECTION_FAIL.** 2. The command shell will exit.
The user does not enter the correct authentication information	1. Error: **ERR_AUTHENTICATION_FAILED** 2. The command shell will exit.
The user enters an invalid request, e.g. user invokes a GET/SET request with a NULL connection parameter	1. Error: **ERR_INVALID_PARAM** **2.** The command shell will exit**.**
The user invokes GET/SET request without	1. Error: **ERR_NOT_AUTHENTICATED**

authenticating	2. The command shell will exit**.**
The user tries to GET/SET from or to a table that does not exist	1. Error: **ERR_TABLE_NOT_FOUND** 2. The command shell will exit**.**
The key that user tries to GET from a table does not exist	1. Error: **ERR_KEY_NOT_FOUND** 2. The command shell will exit**.**
Any errors other than the ones listed above occur, e.g. not enough memory	1. Message: **ERR_UNKNOWN** 2. The command shell will exit**.**

Appendix C: Comparison of popular data exchange formats

After a thorough research and analysis, a custom data exchange format was developed to **increase the reliability** of the event calendar storage server for the desired functionality. As the data format is tailored made for the storage server, its complexity would be appropriate for the system, making it faster than alternatives to parse data. Thus, this would **increase the efficiency** of the system. Also, since the data contained within the format is controlled, it **increases the security** of the system. The table below compares the custom data exchange format with JSON and XML, which are popular alternative data formats, against criteria selected by the team.

Table 4: Weighted decision matrix [34] (Ratings given from 1 to 5)

Criteria	**Custom format**	**JSON**	**XML**
Sharing traditional data (Strings, integers, etc.) (40%)	5	5	5
Security (25%)	5	4	3
Human readable (15%)	5	5	3
Platform and language independence (10%)	5	5	5
Ease of use (5%)	5	5	3
Non-traditional data (Documents, executables etc.) (5%)	1	3	5
Overall Score	**4.80**	4.65	4.10

Key: 1 - Terrible; 2 - Satisfactory; 3 - Good; 4 - Very Good; 5 - Excellent

Selected Data Exchange Format**:** **Custom format**

Appendix D: Examples of Custom Data Exchange Format

After analysing the functionality required by the storage server, the table below contains a few examples for data transfer between the client side and server side employing the use of the custom format.

Table 4: Examples of custom format encoded messages sent and received by the server side

Functionality	**Message received by the Server Side**	**Message sent by the Server Side**
Authenticate	"Authenticate, Username, Password"	"Success" or "Error, error code"
Get	"Get, Table, Key"	"Success, Value" or "Error, error code"
Set	"Set, Table, Key, Value"	"Success" or "Error, error code"
Query	"Query, Criteria"	"Success, Array of Keys" or "Error, error code"
Error	--	"Error, error code"

<u>Appendix E: Unit test cases</u>

The following table describes the unit test cases that must be conducted on different parts of the code to check if it is fully functional.

No.	Portion of program	Test methodology
1.	Testing the server side	1. Enter correct connection information to see if CONNECT works. This should also be checked with incorrect information to see if it fails. 2. Enter correct authentication information to see if AUTHENTICATE works. This should also be checked with incorrect information to see if it fails.
2.	Testing the configuration file parser	1. Make a configuration file with duplicate entries and check if this returns an error message with the correct errno. 2. Make a configuration file with two entries for "TABLE", with one of them commented with a "#". Ensure that only the correct table is made.
3.	Testing the Data Structure : The	1. Enter a large number of records into a single

	SET function	table to ensure that the data structure can SET records into the system. 2. Enter records across different tables to ensure that the structure is capable of handling multiple tables accurately. 3. Enter a record with the same key twice, to check whether SET updates the old entry.
4.	Testing the Data Structure : The GET function	1. Retrieve an existing record to check the GET function. 2. Try to retrieve a non-existent record to see if GET fails.
5.	Testing the Data Structure: The DELETE function	1. Delete a record and try to GET it back to check if the delete functionality actually works. In addition, after deletion, try to get any record to check if the delete function crashes the data structure.
6.	Testing the Data Structure : The QUERY function	1. Ensure that once a query is made, all the keys to records with matching predicates are retrieved. It should return NULL if no predicates match. 2. Make a query where the number of returned keys should be more than the maximum number allowable. Ensure that the program truncates the other results and returns the maximum allowable number.

9.0 Notes

Student names and IDs

Joey Bose (999738440)
Tanveer Fuad (1000056887)
Vivek Mogalapalli (999448209)

Executive Summary
Prepared by: Joey Bose
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

1.0 Introduction
Prepared by: Tanveer Fuad
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

2.0 Software Architecture

2.1 UML Component Diagram
Prepared by: Vivek Mogalapalli
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

2.2 UML Sequence Diagram
Prepared by: Vivek Mogalapalli
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

3.0 System Requirements

3.1 Functions
Prepared by: Joey Bose
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

3.2 Objectives
Prepared by: Tanveer Fuad
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

3.3 Constraints
Prepared by: Vivek Mogalapalli
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

4.0 Design Decisions

4.1 Data Structure
Prepared by: Tanveer Fuad
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

4.2 Communication Protocol
Prepared by: Vivek Mogalapalli
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

4.3 Connection State
Prepared by: Vivek Mogalapalli
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

4.4 Error Handling
Prepared by: Tanveer Fuad
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

4.5 Parsing Algorithms

Prepared by: Joey Bose

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

4.6 Modification from New Requirements

Prepared by: Vivek Mogalapalli

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

4.7 Unit Tests

Prepared by: Tanveer Fuad

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

5.0 Use Case

5.1 Primary Actor

Prepared by: Joey Bose

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli=

5.2 Level

Prepared by: Tanveer Fuad

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

5.3 Stakeholders and interests

Prepared by: Joey Bose

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

5.4 Precondition

Prepared by: Tanveer Fuad

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

5.5 Minimal Guarantee

Prepared by: Joey Bose

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

5.6 Success Guarantee

Prepared by: Tanveer Fuad

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

5.7 Main Success Scenario

Prepared by: Tanveer Fuad

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

5.8 Extensions

Prepared by: Tanveer Fuad

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

5.9 Use Case diagram

Prepared by: Vivek Mogalapalli

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

6.0 Conclusion

Prepared by: Joey Bose

Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

7.0 List of References

Prepared by: Vivek Mogalapalli, Tanveer Fuad
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

8.0 Appendices

Prepared by: Vivek Mogalapalli, Tanveer Fuad
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli

9.0 Notes

Prepared by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli
Edited by: Joey Bose, Tanveer Fuad, Vivek Mogalapalli