**RU GOING**

Application for Aggregating and Suggesting Campus Events

**FINAL PROJECT REPORT**

PROJECT MEMBERS

FAN ZHANG

NAINISHA WAVHAL

RAUL S LASLUISA

SHARAT CHANDRA

SNEHA GALA

Contents

[Chapter I Project Summary 3](#_Toc279700233)

[1. Introduction 3](#_Toc279700234)

[1.1 Purpose of the system 3](#_Toc279700235)

[1.2 Scope of the System 3](#_Toc279700236)

[1.3 Objectives and Success Criteria 4](#_Toc279700237)

[1.4 Definitions, Acronyms and Abbreviations 4](#_Toc279700238)

[1.5 References 4](#_Toc279700239)

[2. Functional Requirements 5](#_Toc279700240)

[2.1 Functional Requirements 5](#_Toc279700241)

[3. Nonfunctional Requirements 13](#_Toc279700242)

[4. System and Object Design 14](#_Toc279700243)

[4.1 System Design Overview 14](#_Toc279700244)

[4.2 Class Design 17](#_Toc279700245)

[5. Testing Approach 19](#_Toc279700246)

[6. Assignment of Responsibilities 22](#_Toc279700247)

[7. Team Communications 24](#_Toc279700248)

[7.1 Project Meetings 24](#_Toc279700249)

[7.2 Email Communications 25](#_Toc279700250)

[Chapter II Project Status 26](#_Toc279700251)

[1. Current Status Review 26](#_Toc279700252)

[2. Appendix 27](#_Toc279700253)

[2.1 Testing Output 27](#_Toc279700254)

[2.2 SVN Link and Description 28](#_Toc279700255)

[Chapter III Project Discussion 29](#_Toc279700256)

# Chapter I Project Summary

## 1. Introduction

### 1.1 Purpose of the system

College life can sometimes be hectic with several curriculum and non-curriculum events that are organized on campus for the students throughout the year. It is difficult for the students to be aware of all the events on campus that they might find interesting. So to plan a schedule based on one’s interest becomes important in order to get full exposure to one’s interests.

In this project we will try to aggregate the entire Rutgers event catalog and find what events one should go to based on one’s personal preferences and trends with your friends. This site will inform the users of great events that they might not have even heard of until now. This will be useful for the university students to know of the vast range of events that take place on campus and also match the events with their interests and social circle. Besides the recommended events the user can also search for specific upcoming events. The user can rate an event and upload photos for an event. He/She can find the exact location of the event on Google Maps. The students can share an event using the *Like* button on his/her Facebook page. The user can also comment on an event (via Twitter and Facebook).

### 1.2 Scope of the System

* All users should be able to access the RU Going as a web application.
* This application makes the user aware of various departmental and sporting events.
* This application uses Google Maps for locating the events for the user.
* This application uses *Like* button of Facebook for the user to share an event.
* This application uses the comment widgets from Facebook and Twitter to connect RU Going to user’s cyber world social networks.

### 1.3 Objectives and Success Criteria

* Provides a GUI for the user to sign up for this application in which the user provides personal data.
* Provides a GUI enabling the user to find campus events based on specified interests and preferences.
* Provides the student with data about the time, location, subject and social rating of the event.

### 1.4 Definitions, Acronyms and Abbreviations

* **User:** End user, use this application to be notified about campus events. Most users of this category are typically university students.
* **Remote Server:** The server/Database, which provides a suggestion based on the events that match user’s data.
* **GUI:** Graphical User Interface, is a type of user interface, that allows users to interact with programs in more ways than typing such as computers, hand-held devices such as MP3 players, portable media players or gaming devices, household appliances and office equipment with images rather than text commands.
* **Campus Event:** Curriculum events within the departments and co-curricular like sports and cultural events that are organized throughout the year
* **Administrator:** Manages the student accounts and data corruptions in events database

### 1.5 References

* <http://www.squarespace.com/>
* <http://www.wikipedia.org/>
* Object-Oriented Software Engineering

- Bernd Bruegge and Allen H. Dutoit

* Enterprise Architecture As Strategy: Creating a Foundation for Business Execution

## 2. Functional Requirements

### 2.1 Functional Requirements

RU Going supports two types of users:

1. The students should be able to sign up and register to become a member of the application. The application provides a member form that lists the details that the user should be able to provide to shortlist the events according to his interest. The user should be able to update (add, modify, delete) his profile anytime. The user should be able to tag the event as “attending” if he decides to attend the event.
2. The administrator should be able to manage student and event records. He should be able to, install, support, and maintain servers.

***Register***

|  |  |
| --- | --- |
| *Use case name* | **registerAccount** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | 1. **User** requests a registration form from the homepage. 2. **RUGoing** returns the online registration form. 3. **User** fills out form (user id, password, name, major, email, interest/events preference etc). Once the form is completed, the **User** submits the form. 4. **RUGoing** receives the form and stores data. 5. **RUGoing** returns acknowledgement to **User**. |
| *Entry condition:* | * User clicks the RUGoing register button on homepage |
| *Exit condition:* | * **User** receives acknowledgement OR **User** receives an explanation indicating why the transaction could not be processed. |
| *Quality requirements* | * System should display the acknowledgement within 5 seconds |

*Figure 2-1 The registerAccount use case*

***Search***

|  |  |
| --- | --- |
| *Use case name* | **searchEvents** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | 1. **User** types keywords in **RUGoing** search box and submits the search query. 2. **RUGoing** returns a list of events. |
| *Entry condition* | * **User** must be logged in and click search button. |
| *Exit condition:* | * **User** receives the list of events OR **User** receives an explanation indicating why the transaction could not be processed. |
| *Quality requirements* | * System should display the acknowledgement within 5 seconds |

*Figure 2-2 The searchEvents use case*

***Recommended Events***

|  |  |
| --- | --- |
| *Use case name* | **getRecommendedEvents** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | 1. **User** request recommendations by choosing the recommended events button. 2. **RUGoing** returns a webpage that displays recommended events (which are automatically selected by application system, based on user’s interest and events preference information). |
| *Entry condition* | * **User** must be logged in and click request recommendation. |
| *Exit condition:* | * **User** receives recommended events OR **User** receives an explanation indicating why the transaction could not be processed. |
| *Quality requirements* | * System should display the acknowledgement within 5 seconds |

*Figure 2-3 The getRecommendedEvents use case*

***Upcoming Events***

|  |  |
| --- | --- |
| *Use case name* | **getUpcomingEvents** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | * + - 1. **User** requests upcoming events by choosing the upcoming events button.  1. **RUGoing** returns a webpage that displays upcoming (which are automatically selected by the application system, based on the scheduled time of the events). |
| *Entry condition* | * **User** must be logged in and click request upcoming. |
| *Exit condition:* | * **User** receives upcoming events OR **User** receives an explanation indicating why the transaction could not be processed. |
| *Quality requirements* | * System should display the acknowledgement within 5 seconds |

*Figure 2-4 The getUpcomingEvents use case*

***Comment (Facebook)***

|  |  |
| --- | --- |
| *Use case name* | **CommentonEvent** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | * + - 1. User request the event attended by searching the event using search tab  1. RUGoing returns the event 2. User adds the comments. 3. Comment is processed by Facebook |
| *Entry condition* | * **User must be logged in and search the event attended**. |
| *Exit condition:* | * **Event will appear with comments**. |
| *Quality requirements* | * System should display the acknowledgement within 5 seconds |

*Figure 2-5 The* **CommentonEvent** *use case*

***Comment(Twitter)***

|  |  |
| --- | --- |
| *Use case name* | **TwittonEvent** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | * + - 1. User request the event attended by searching the event using search tab.       2. RUGoing returns the event       3. User is presented with a Twitter feed to see related tweets |
| *Entry condition* | * **User must be logged in and search the event attended**. |
| *Exit condition:* | * **Event will appear with twitt comments**. |
| *Quality requirements* | * System should display the acknowledgement within 5 seconds |

*Figure 2-6 The* **TwittonEvent** *use case*

***Share (use Facebook like button)***

|  |  |
| --- | --- |
| *Use case name* | **LikeEvent** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | 1. User request the event attended by searching the event using search tab. 2. RUGoing returns the event 3. User likes the event. |
| *Entry condition* | * **User must be logged in and search the event attended**. |
| *Exit condition:* | * **Event is liked and shared on facebook wall**. |
| *Quality requirements* | * System should display the acknowledgement within 5 seconds |

*Figure 2-7 The LikeEvent use case*

***Rate event***

|  |  |
| --- | --- |
| *Use case name* | **rateEvent** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | 1. **User** click on attended event. 2. **RUGoing** displays event requested by user. 3. **User** will rate the event by clicking on the respective star rating. |
| *Entry condition* | * **User** is logged in and request for attended event. |
| *Exit condition:* | * Event will be rated based on user experience. |
| *Quality requirements* | * System should display the requested event within 5 seconds. |

*Figure 2-8 The rateEvent use case*

***Upload Photos of Event***

|  |  |
| --- | --- |
| *Use case name* | **UploadPhotos** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | 1. **User** selects an event from displayed list of events. 2. **User** requests to upload photos. 3. **RUGoing** receives input and uploads the photos and stores it on the server. 4. **User** can view the photos. |
| *Entry Condition* | * **User** must be logged in and click UploadPhotos |
| *Exit Condition* | * **User** can view the images he uploaded |
| *Quality Requirement* | * Systemshould display the request within 5 seconds |

*Figure 2-9 The UploadPhotos use case*

***Modify Account***

|  |  |
| --- | --- |
| *Use case name* | **modifyAccount** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | 1. **User** requests to modify account. 2. **RU Going** returns form. 3. **User** modifies old information (e.g. reset password, edit personal interest) and submits. **User** may also request to have their account deleted. 4. **RU Going** receives and stores data or flags for deletion. 5. **RU Going** returns acknowledgement to **User**. |
| *Entry condition* | * **User** must be logged in RUGoing and click modify account button. **User** must already have an active account on the system. |
| *Exit condition:* | * **User** receives acknowledgement OR **User** receives an explanation indicating why the transaction could not be processed. |
| *Quality requirements* | * System should send an acknowledgement within 5 seconds |

*Figure 2-10 The modifyAccount use case*

***Like Event(via Facebook)***

|  |  |
| --- | --- |
| *Use case name* | **LikeEvent** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | * + - 1. **User** request attended event.       2. **RUGoing** displays the event requested by user.       3. **User** will click on the “like” button next to the event. |
| *Entry condition* | * **User** is logged in and request for attended event either by searching OR viewing past recommended events. |
| *Exit condition:* | * **User** receives acknowledgement OR **User** receives an explanation indicating why the transaction could not be processed. |
| *Quality requirements* | * System should send an acknowledgement within 5 seconds |

*Figure 2-11 The LikeEvent use case*

***ViewMap(via Google Map)***

|  |  |
| --- | --- |
| *Use case name* | **ViewMap** |
| *Participating actor* | **Initiated by User** |
| *Flow of events* | 1. **User** request for the location of the event. 2. **RUGoing** returns the Google Map of location for the requested event. |
| *Entry condition* | * **User** must be logged in and click ViewMap. |
| *Exit condition:* | * **User** receives the Map displaying event location OR **User** receives an explanation indicating why the transaction could not be processed. |
| *Quality requirements* | * System should display the map within 5 seconds |

*Figure 2-12 The ViewMap use case*

## 3. Nonfunctional Requirements

**Usability:** The user should be able to use this application without any specialized training. The user should only be familiar with filling out basic html forms such as input boxes and dropdowns. The user shall not feel like they are filling out a questionnaire. In order to do this we will limit our registration form to 10 fields. The user in the case they don’t understand what information is needed in a particular field shall be provided with a hover over help button which explain what should be the input.

**Reliability:**  The application once running may be restarted at anytime and keep the state in which it was before reboot. Any errors or warnings that are encountered by any part of the application shall be logged in a centralized file. Errors based on user input shall be returned back to the user. All login credentials shall be encrypted with standard SSL.

**Performance:** The system should be able to handle 50 simultaneous request and up to 300 request in a queue. The system should be able to do any event search within 5 seconds of user submission.

**Supportability:** Any updates to the system will have to be installed manually by the administrator of the application. Any updates will be given in tar balls and windows executables to simplify installation.

**Implementation:** Entire application will need 3 servers or virtual machines to run. All users having an internet connection should be able to access RU Going through a web interface.

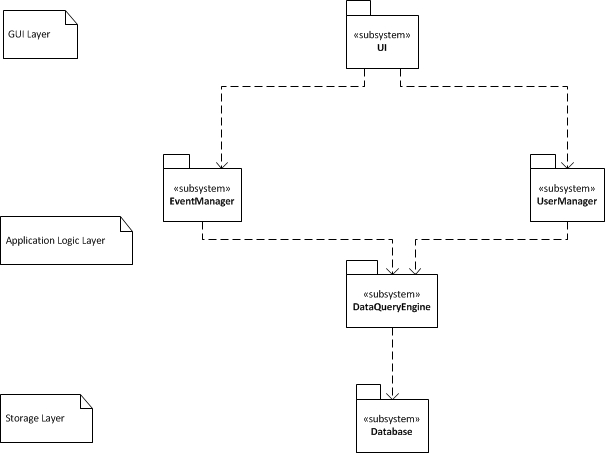
**Interface:** This application will be able to interact with the outside world using web services using SOAP. These services will allow data to be exported in XML format.

**Packaging:** The application will be packaged in 1 tar ball and 1 windows executable. Installation instruction will be provided in a separate README file.

**Legal:** The application will need to adhere to federal laws in the USA in terms of the length to which data must be stored. In addition, RU Going would adhere to the Twitter and Facebook users privacy policies.

4. System and Object Design

### 4.1 System Design Overview



*Figure 4-1 Sub-System Design Diagram*

The RUGoing system is efficiently decomposed into the hierarchical 3-teir architecture, which consists of the interface layer, application logic layer, data query layer and storage layer. The interface layer has only one subsystem, **UI**, which encapsulates all classes and objects related to presenting/rendering application web pages of forms and data to the users, collecting data input from users and sending requests for data. The application logic layer includes the following subsystems: **RecommendationManagement**, **EventManagement**, **UserManagement**, **InputValidationManagement**, **CalendarManagement, DataQueryEngine**. This layer deals with application procedures/functionalities related to users, events, calendars, recommendation, it also creates and issues database queries. Finally, the storage layer also has only one subsystem as **DataBase**. The tasks of **DataBase** subsystem include storing persistent data records, responding to SQL queries from the above **DataQueryEngine** subsystem.

RUGoing hierarchical decomposition satisfies the requirements of subsystem coupling and cohesion. Though obvious coupling links exist between layers, subsystems within one specific layer (e.g. application logic layer) are mostly independent and loosely coupled. Two *exceptions* exist: The **DataQueryEngine** subsystem would be used by **EventManagement**, and **UserManagement**. The introduction of **DataQueryEngine** greatly reduces the number of direct dependency links between other application logic subsystems and **DataBase**, which make the overall coupling of the subsystem decomposition decreased. For example, when the storage layer is modified (e.g. use different database vendors or different storage mechanism), change only needs to be made on **DataQueryEngine** subsystem. To further minimize the coupling, instead of having a centralized maintenance subsystem that could cause a 1-to-all coupling links, RUGoing distributes system maintenance duty among all individual subsystems. Each subsystem in RUGoing has highly cohesive design, and encapsulates all related objects/processes/classes in a single subsystem. The most important reason for the **DataQueryEngine** is to hide any changes to the database from the rest of the system. Since our data is the most part of the system over time there will come a time when the underlining database will have to change. So as a consequence the **DataQueryEngine** will allow the database to have a dramatic change and update other subsystems gradually. The other main reason is to allow new modules to be made from the data that is being created by RUGoing. This is more a play on enterprise level architecture to allow the actual people that will be administrating RUGoing easy access to the data. (Reference <http://www.amazon.com/Enterprise-Architecture-Strategy-Foundation-Execution/dp/1591398398>)

The first alternative which was brought up was the traditional connection between each module to the database. This would be the exact same architecture but without the **DataQueryEngine**. What normally happens when a system like this is created is that when changes happen to the database the entire system must be updated. What this means is that if the database is changed the UI must be change every time and rerun test to ensure the system still properly works.

The second alternative is to put the system entirely on one box and just use one framework to host the application. An example of this would be an ASP .NET application. These applications are completely windows based and easily integrate with other windows products. The problem with this would be it ties you down to one technology base and you can no longer go with best of breed. If RUGoing were to become a company like say a Facebook this would be highly detrimental to scaling the product.

**User Interface (UI)**

This subsystem belongs to interface layer of RUGoing system architecture. Classes, objects and procedures encapsulated in this subsystem will handle the presentation, layout arrangement, rendering of data records, forms, and graphics on the RUGoing web pages. This subsystem has direct dependence links to most subsystems in the application logic layer, because all the data presented by UI is from the application logic layer, and user input data should be sent from UI to various application procedures defined in the logic layer.

**EventManagement**

This subsystem, part of the application logic layer, encapsulates the objects and procedures related to the following: Event Insertion--there is an daemon thread procedure (runs in the background) named Data Miner (which we consider as “internal actor” in this subsystem) to automatically crawl new events information from Rutgers Events web page and update the events library of RUGoing; Event Expiration check--this subsystem checks the event date and marks past events as “expired”; In addition, this subsystem is responsible for events deletion, events commenting, facebook sharing, photos upload and rating. **EventManagement** depends on **DataQueryEngine** to get/put event data from/into storage. Also, this subsystem provides event-related information to upper-level UI subsystem.

**UserManagement**

This subsystem, part of the application logic layer, encapsulates the objects and procedures related to user registration, user login/authentication, user account modification (e.g. reset password, add personal interest). This subsystem depends on **DataQueryEngine** to get/put user related data records from/into storage. Also, this subsystem provides user-related information to upper-level UI subsystem.

**DataQueryEngine**

This subsystem, part of the application logic layer, encapsulates the objects and procedures related to issuing SQL queries to underlying data base system. This subsystem provides data query service. It directly interacts with the **DataBase** subsystem in storage layer.

**DataBase**

This subsystem, part of the storage layer, includes a relational database system for management (using tables), and storage (using disks) of persistent data records generated in RUGoing. This subsystem would respond to SQL queries from **DataQueryEngine** subsystem.

### 4.2 Class Design

 *Figure 4-2 RUGoing Class Diagram 1*



*Figure 4-3 RUGoing Class Diagram 2*

## 5. Testing Approach

We choose PyUnit the standard unit testing framework module for Python, to implement our test cases for RUGoing.

**Test RegisterUser Use Case**

**Purpose:** registerUser() function allows user to register in the application. User has to provide data that includes username, firstname, lastname, interests fields required to maintain his/her profile information. To make sure user provides correct information (valid input data) we run 3 test. **Boundary Testing Test1:** User fills out all required input field (check for valid inputs) The function tested is registerUser() which takes in input fields such as username, password, firstname, lastname, interest as parameters and it is observed that the function accepts the input and returns TRUE.

**Equivalence Testing Test2:** Check for not accepting same input data by different users Test is run providing same username, firstname, lastname as given for the first test case and this time return value is FALSE as it invalidates the input considering it incorrect because it already exist and should not be accepted again. There can be several classes of invalid or incorrect inputs, but this one of the important one to consider for testing.

**Boundary Testing Test3:** User does not fill in all required input field User does not provide complete information by leaving some input fields blank, function returns FALSE as it is not a valid or complete data.

**Modifyaccount Use Case:**

**Purpose:** We tested the Modifyaccount function provided to the user using which he/she can modify his/her account details. The password, interests, first name and last name can be changed according to the user will once he logins in using the correct username and password and then chooses the option to modify his account. We have tested for three conditions using PyUnit:

1. Test case for correct username and password

2. Test case for correct username and wrong password.

3. Test case wrong username.

**Equivalence Test 1:** Test case for correct username and password. Allows user to modify her password. The user can then login with the new password. When the input data is correct username and password then the test returns TRUE

**Equivalence Test 2:** Test case for correct username and wrong password does not allow the user to modify. If the username and password are not matching the user cannot modify his/her account details. When the input data is correct username and wrong password then the test returns FALSE

**Equivalence Test 3**: Test case wrong username does not allow the user to modify. When the input data is wrong username the test returns FALSE

**Test RatingEvent Use Case**

**Purpose:** A person can by-pass the UI and insert rating values apart from the 5-stars visible on the events page by communicating with the underlying webservice .To make sure that the rating of an event by a user does not violate the norms , we have included 3 test cases that guard the system from such meddling. Requirement Report

The conditions that we have tested for are :

1. The rating should not be negative. –(Equivalence Testing)

2. The rating should not be greater than 5 – (Equivalence Testing)

3. The eventID should be of a valid event – (Equivalence Testing)

**Equivalence Test 1:** Rating should not negative

a) If the input data is negative, the test returns FALSE (Boolean)

b) If the input data is a positive integer , the test returns TRUE (Boolean)

**Equivalence Test 2:** Rating should not be greater than 5

a) If the input data is negative integer , it returns TRUE

b) If the input data is positive integer between 0 and 5 it returns TRUE

c) If the input data is greater than 5 , it returns FALSE

**Equivalence Testing 3:** A valid eventID that is present in the database

a) If eventID for which the rating was done is not found, then it returns FALSE

**Test InsertEvent use case:**

**Purpose:** To make sure the events that inserted into RUGoing by the data miners do have valid content and values, we have used 2 test cases to protect our system from being polluted by invalid events data. The conditions that we have tested for:

1. The eventtime of inserted event should be equal to or larger than the BoundaryDate = date of today.

2. The eventname of inserted event should be a string with number of characters as NameStrLen, and 6<=NameStrLen<=60.

**Boundary Test 1** Inserted event should have valid eventtime The tested action is “InsertEvent”, and we use 3 input data:

a) Input: event that have eventtime < BoundaryDate.

Output: it returns FALSE.

b) Input: event that have eventtime = BoundaryDate.

Output: it returns TRUE.

c) Input: event that have eventtime > BoundaryDate.

Output: it returns FALSE.

**Equivalence Test 1** Inserted event should have valid eventname The tested action is “InsertEvent”, and we divide input data into 3 partitions:

a) Input: event that have NameStrLen < 6.

Output: it returns FALSE.

b) Input: event that have 6<= NameStrLen <=60.

Output: it returns TRUE.

c) Input: event that have NameStrLen > 60.

Output: it returns FALSE.

**Test Get Recommended Events Use case**

**Purpose:** To test proper functionality of getRecommndedEvents use case. In this use case there are two types of users, ones that exist in the system and those that don’t. Because of this we have decided to use equivalence testing to test the functionality of this function. The conditions that we have tested for are :

1. The user is in the system. –(Equivalence Testing)

2. The user is not in the system – (Equivalence Testing)

3. Invalid username due to size of username being provided – (Boundary Testing)

**Equivalence Test 1** User should exist in the system

* If the input data is a valid user the system will return a list of events.
* The application will return at least one recommended event when the above condition is met.

**Equivalence Test 2** User does not exist in the system

If the input data is an invalid user the system will return an empty list of events.

The application will return no events when this above condition occurs.

**Boundary Testing 3** Username cannot be greater than 20 characters long

If the input data is invalid (too many characters) the system will return an empty list of events.

The application will return no events when this above condition occurs

## 6. Assignment of Responsibilities

|  |  |
| --- | --- |
| **RAD** | **Member** |
| Introduction | All members |
| Current System, Functional Requirement | All members |
| Non- Functional Requirement | Fan |
| System Models | All members |

|  |  |
| --- | --- |
| **OOA** | **Member** |
| Class Diagram | All members |
| Sequence Diagram | All members |
| State Chart Diagram | Raul |

|  |  |
| --- | --- |
| **SDD** | **Member** |
| Introduction, Software Architecture(except A  ccess Control) | All members |
| Access Control and Security | All members |
| Global Software Control | All members |
| Boundary Condition | All members |
| Subsystem Services | All members |

|  |  |
| --- | --- |
| **Implementation Part** | **Member** |
| User Interface | Raul |
| ViewMap(usecase) | Sneha |
| UploadPhotos(usecase) | Nainisha |
| RateEvent(usecase) | Sharat |
| InsertEvent | Fan |
| DataBase | Raul |
| Integrated Testing | All members |

## 7. Team Communications

### 7.1 Project Meetings

A weekly meeting was scheduled on Wednesdays, 5:00PM-6:20PM. Apart from that we had regular touch with each other on email and IM when we required each other’s assistance. One of us used to write on the white board during discussions while another of us used to jot down notes. A sample of the minute looks somewhat like this

**Project Meeting No.5** Sept. 19 CoRE Building 6th floor , Rutgers University

**Meeting Minutes**

1. Opening Remarks
   1. Ms.Sneha Gala details the agenda for the meeting and the issues that are to be addressed
2. List of Participants
   1. Fan Zhang
   2. Nainisha Wavhal
   3. Sharat Chandra
   4. Sneha Gala
   5. Solomon Lasluisa
3. Review current status

Team members give updates on the work they did over the week and bring out any problems if faced

1. Project Progress
   1. Noted : Professor Gruteser request Project Project Requirements Analysis Report
   2. Actions : Discussion over the Use case diagrams and UML diagrams on a whiteboard.
   3. Actions : All of us jot down the UML diagrams that we want to draw in a notebook.
   4. Agreed : Distribute work among ourselves with each person taking responsibility to draw UML diagrams after the meeting.
   5. Agreed : To follow the textbook rules as closely as possible and keep that as guideline
   6. Agreed : To share the UML diagrams via email once we are done with drawing UML diagrams

|  |  |  |
| --- | --- | --- |
|  | Our Plan | Professor Gruteser’s Plan |
| Requirements Analysis | Sep 28th,2010 | Sep 30, 2010 |
| Object Oriented Analysis | Oct 13th 2010 | Oct 15th, 2010 |
| Architecture and Design report | Oct 23rd, 2010 | Oct 25th, 2010 |
| Prototype and testing Report/demo | Nov 2nd , 2010 | Nov 4th 2010 |
| Design iteration and Test Drive development | Nov 21st, 2010 | Nov 23rd, 2010 |

### 7.2 Email Communications

To keep everything well informed we use email for communication. Usually it will be task assignment, question clarification.Chapter II Project Status

## 1. Current Status Review

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Status** | **In Demo?** |
| registerAccount | implemented and passes all tests | **Y** |
| searchEvents | implemented and passes all tests | **Y** |
| rateEvent | implemented and passes all tests | **Y** |
| Share | implemented and passes all tests | **Y** |
| Comment | implemented passed initial tests | **Y** |
| Get Live Feed | implemented passed initial tests | **Y** |
| getUpcomingEvents | implemented and passes all tests | **Y** |
| UploadPhoto | implemented passed initial tests | **Y** |
| ViewMap | implemented passed initial tests | **Y** |

*Figure 1-1 Current Use Case Status*

**NOTE:** implemented passed initial tests means that the function passed manual testing and automating tests have not been created.

Objects demonstrated in last class:

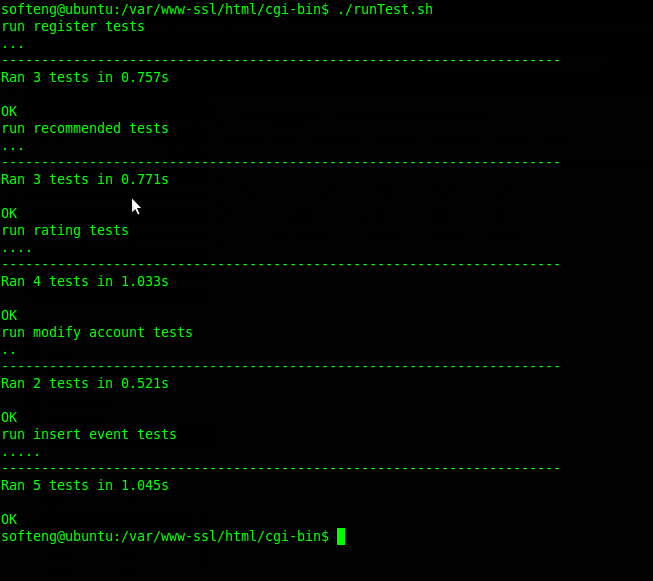
(1) registerAccount

(2) searchEvents

(3) getUpcomingEvents

## 2. Appendix

### 2.1 Testing Output

****

*Figure 2-2 Test Output*

### 

### 2.2 SVN Link and Description

Home Directory: <http://code.google.com/p/gradsefall2010/source/browse/#svn/trunk>

Web Service: <http://code.google.com/p/gradsefall2010/source/browse/#svn/trunk/UIservice>

HTML/GUI: <http://code.google.com/p/gradsefall2010/source/browse/#svn/trunk/html_final>

Web Crawler: <http://code.google.com/p/gradsefall2010/source/browse/#svn/trunk/data-miner>

Database Scripts: <http://code.google.com/p/gradsefall2010/source/browse/#svn/trunk/dbSQL/rugoing>

# Chapter III Project Discussion

***‐ Which software engineering techniques did you find helpful in your project and which techniques did not seem to improve your productivity?***

The iteration-based software development process gives us the opportunity to discuss and rethink the initial design choices (including functional requirements, classes design and system architecture design). After the iteration I, we ceased the implementation of events calendar, and added new functionalities such as comment (via Facebook), share (via Facebook), view event location in the map (Google Map), upload event photos. This revision helps us select and implement the most interesting, ambitious but realistic features, within the given time frame.

Following the philosophy of software module / framework / design pattern reuse, we avoid “rebuilding the wheels” and make the development of RUGoing more efficient and robust. For example, from the very beginning, we chose to use Microsoft .NET framework for developing the RUGoing web service. We can use software tools like Visual Studio 2010 to develop our server-side web service codes under the .NET framework. This makes the coding task like connecting to DB more simple and efficient. Also, we used Scrapy – a high-level screen scraping and web crawling framework, to develop our python codes to crawl ruevents.rutgers.edu websites and extract data from their html pages. In this way, we do not need to build our web pages crawler from the scratch, which also saves us a lot of time and efforts.

Using Google Code as our project source code repository is another correct choice. Because the project codes is under development by different team members, it would be easy to get into the situation of have multiple copies of inconsistent source code files. With the SVN repository hosted by Google Code, we do not need to worry too much because each team member share the same view of the source code, and all the history versions of the code are cached.

***‐ Did your testcases actually identify any bugs? Did you notice any benefits of writing them?***

Yes, Test cases are great help in noticing the bugs. While testing the RegisterAccount Test case we observed we had forgot to consider the duplication of the username and made the changes in the code.

While doing testing for the ModifyAccount use case we purposely gave “0” as username, the test returned true and we discovered this bug.

***‐ Why does your current design meet your project requirements better than your iteration I design? If you could start over again, how would you change it?***

This design has data management at heart. By having this as the central design future scaling and manipulation of data will be a lot easier than having modules tightly coupled with the database itself. This application sacrifices some optimization to allow data flow to happen easily while creating and destroying outside connections.

If we had the change to do it over we would take modules out of the web service. This would have allowed us to further prove that our design is correct and the best for this type of application.

***‐How would you improve team communications and how would you change the assignment of responsibilities?***

We think that the assignment of responsibilities of our team members worked out almost perfectly. All of us had great team-spirit and we communicated regularly thorough e-mails and met once in a week. But we think that the tasks assigned would have taken less time to finish if we could have arranged meetings often. Unfortunately we could not meet every time due to our different schedules.