PROJECT REPORT

CSC 207: Software Engineering I

LAB 5: SUBSYSTEM DEMO/EVALUATION

Team 5

SESSION 2009/2010
SEMESTER 1
COMPUTER SCIENCE COURSE

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Contents

PROJECT TEAM INFORMATION

Name	Matric No	Role
Cheong Yong Hon	085491L06	Project Manager
Edwin Ang Yew Leng	085465K06	Asst. Project Manager
Ong Ann Aik	085862J06	Backend Programmer
Christopher Khoo Zhan-Peng	085480F06	Database Administrator
Andrew Tan Wei Peng	085797B06	Backend Programmer
Mervyn Chew Yip Chun	085657H06	Web Designer
Lim Qing Yan	085484C06	Backend Programmer

REFINED SOFTWARE REQUIREMENT SPECIFICATION

Nanyang Technological University

SOFTWARE REQUIREMENT SPECIFICATION

My Event Planner

VERSION: [1.3] REVISION DATE: [01/11/2009]

Approver Name	Title	Signature	Date

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Table of Contents

1. Product Description

1.1 Product Vision

The product aims to provide a one-stop web-based event management service for different level of user groups effectively and efficiently.

1.2 Business Requirements

This service should become the preferred channel for event management for a minimum of 300 private event planners, 100 school level planners, and 50 corporate level planners within a year.

1.3 Stakeholders and Users

1.3.1 Stakeholders

- Development team

1.3.2 Users

- Event organizers
- Administrator
- Guest

1.4 Project Scope

The project scope is on event organizing in terms of private level to school level and corporate level. Events include sports events, outings, seminars, meetings; events such as wedding planning are not taken into account yet but will be a fine extension in the future. The project encompasses aspects pertaining to event organizing such as budget planning, event team organizing, contact management, as well as venue and date/time planning.

1.5 Assumptions

- Event organizers are computer literate
- Guest are computer literate
- Guest should have an email account
- Event organizers should have the contacts of the guest

1.6 Constraints

- The project should be developed using Java with JSP and servlets
- The project is to be built on MySQL database
- Version maintaining should be done using TortoiseSVN

2. Functional Requirements

2.1 Event Organizer

2.1.1 MyEventPlanner must allow organizers to add new event(s).

2.1.2 MyEventPlanner must allow organizers to log out after use.

2.1.3 MyEventPlanner must allow organizers to edit and delete his/her existing event(s).

- MyEventPlanner must display weather forecast for the week.

2.1.4 MyEventPlanner must allow organizers to select polling or attending method for a new event.

- If the event is marked to be polled, then MyEventPlanner must display various options (e.g. venue, date and time) for the organizer to choose to poll.

2.1.5 MyEventPlanner must allow organizers to view the statistics of the event

- If event is marked to be pooled, polling results will be showed.
- If event is marked fixed, guest(s) attending and not attending will be showed.

2.1.6 MyEventPlanner must allow organizers to view the statistics of all the current and past events.

2.1.7 MyEventPlanner must allow organizers to add, edit and delete guest contacts.

2.1.8 MyEventPlanner must allow organizers to create groups for his/her guest contacts.

2.1.9 MyEventPlanner must allow organizers to assign guest(s) or group(s) to his/her existing event(s).

2.2 Guests

2.2.1 MyEventPlanner must allow guests to pick a date and time which he/she is available for the event under the polling mode.

2.2.2 MyEventPlanner must allow guests to accept or reject the event under fixed day mode.

2.2.3 When applicable, guests must be allowed to apply for roles they wish to volunteer for the event.

2.2.4 MyEventPlanner must send an acknowledgement upon confirmation of response.

2.2.5 View information of event invited for.

2.2.6 When event details are updated or finalized, MyEventPlanner must notify participating guests.

2.2.7 MyEventPlanner must notify guests attending events 2 days in advance.

2.2.8 MyEventPlanner must allow guests to withdraw from registered events.

2.2.9 MyEventPlanner must prompt guest of reason(s) for withdrawal.

2.2.10 MyEventPlanner must allow guests to view event marked as public.

2.2.11 MyEventPlanner must allow guests to login as organizers if they have an existing account.

2.2.12 MyEventPlanner must allow guests to sign up as new users to the website.

2.3 Administrator

2.3.1 Must be able to view the statistics of all event organized.

2.3.2 Must be able to edit existing event details. i.e when the organizer used inappropriate language, the admin is able to make necessary changes.

2.3.3 Must be able to suspend existing event.

2.3.4 Must be able to manage user.

3. Data Requirements

3.1 Registration and Login page

3.1.1 Username must be alpha-numeric
3.1.2 Username must not be more than 40 characters
3.1.3 Password must be between 5 to 15 characters long
3.1.4 System must check the age of users against the birth year
3.1.5 Name provided by user must not exceed 40 characters
3.1.6 Email address of user must contain '@' and also '.' in it
3.1.7 Postal code must be a positive 6 digit integer
3.1.8 Contact number must be a positive integer
3.1.9 Repeat password must match password entered

3.2 Event page

3.2.1 Upcoming events must not be before the current date
3.2.2 System must check for user's privilege
3.2.3 Budget of event must not be less than 0
3.2.4 Start date of event must be before the end date

3.3 Contact page

3.3.1 Contact email must be a valid email
3.3.2 Date of Birth must be a valid date before current date
3.3.3 Postal code must be a positive 6 digit integer

4. Non-Functional Requirements

4.1 Accessibility

4.1.1 System is able to be viewed/used anywhere and at anytime as long there is an internet connection and a workstation(e.g. PC, laptop, pda, iPhone).

4.1.2 System is able to be viewed on any web browsers correctly.

4.1.3 System is able to display the public calendar and upcoming events once the main page is shown. There is no need to login or sign up if you are not a member.

4.2 Performance

4.2.1 System is able to capture and display new events added on the public or private calendar as soon as the user submits an event.

4.2.2 System is able to log down thousands over events in the database.

4.2.3 System allows the user to store individual guest's email addresses in its database and can be retrieved anytime when adding or modifying events.

4.2.4 An average loading time for a page is 5 to 20 seconds depending on the internet connection speed of the user.

4.3 Reliability

4.3.1 The rate of system breakdown due to hosting server down is as low as 2%.

4.3.2 The mean time between breakdowns should be between 30 minutes to 1 hour's time.

4.3.3 For signing up as premium user, the system uses secured external payment system such as PayPal to make cashless transactions.

4.4 Usability

4.4.1 System allows user to add/modify events within a page.

4.4.2 User is able to view public calendar, click on sponsored pages, or add/modify/view own events at the main page of the system.

4.4.3 User is able to sign up as free member with just one step.

4.4.4 User is able to sign up as premium member with guided steps page by page in less than 10 minutes.

4.4.5 User is able to inform guests of event or request a polling of guest's availability at any one time.

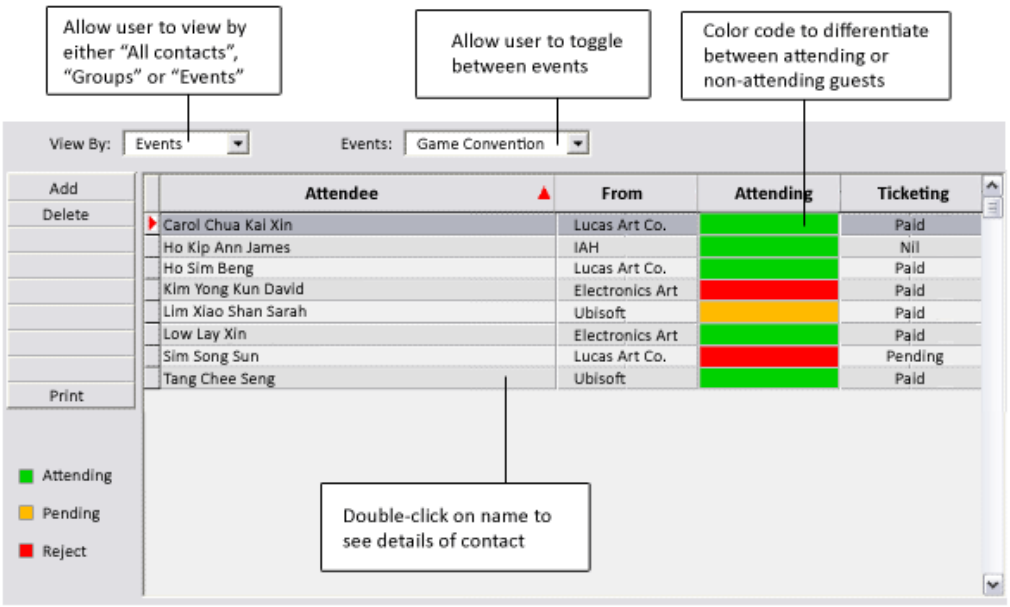
4.4.6 System is able to send out notification emails of events and email reminders for immediate actions to their guests.

5. Interface Requirements

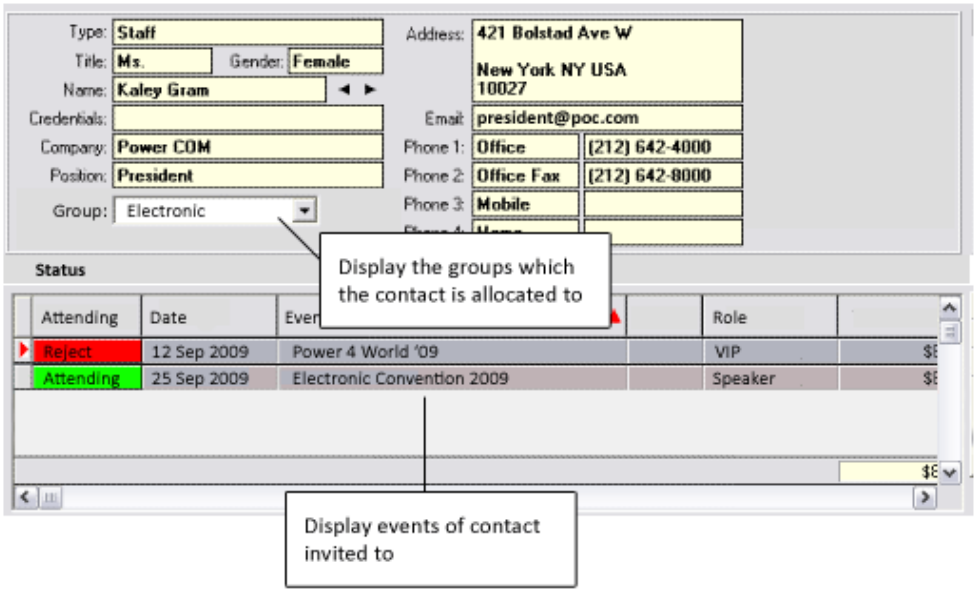
5.1 Initial Conceptual UI

5.1.1 Contacts Interface

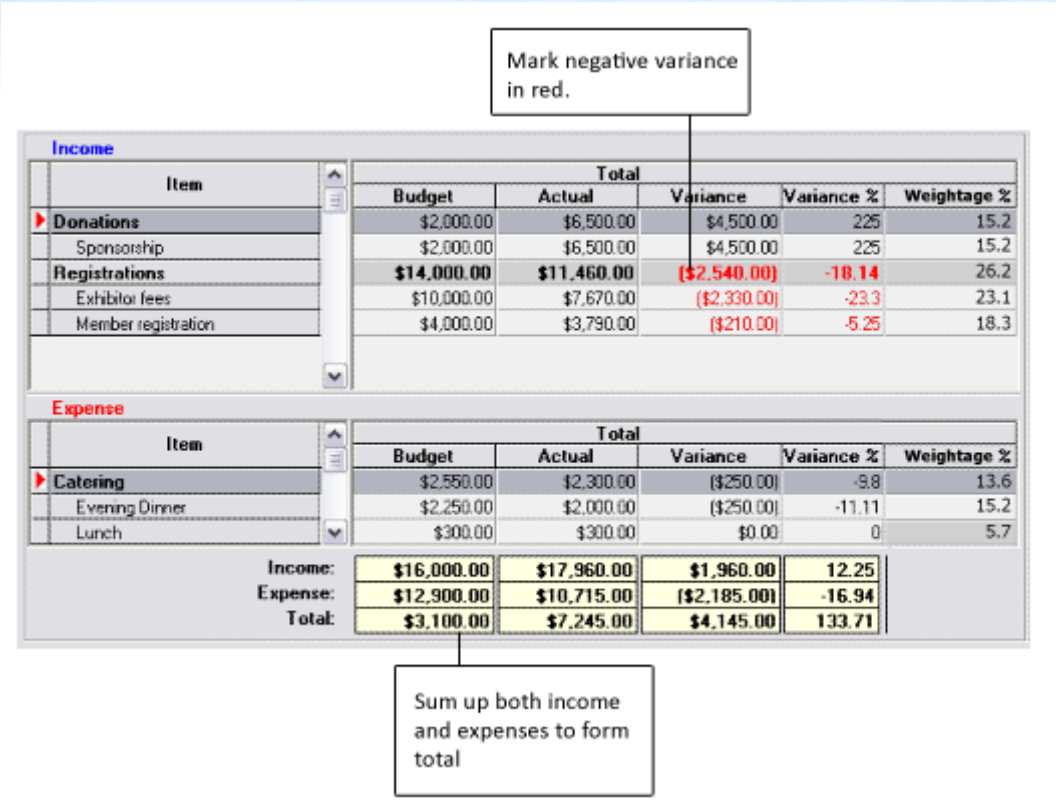
5.1.1.1 Viewing contacts by groupings



5.1.1.2 Viewing individual contact

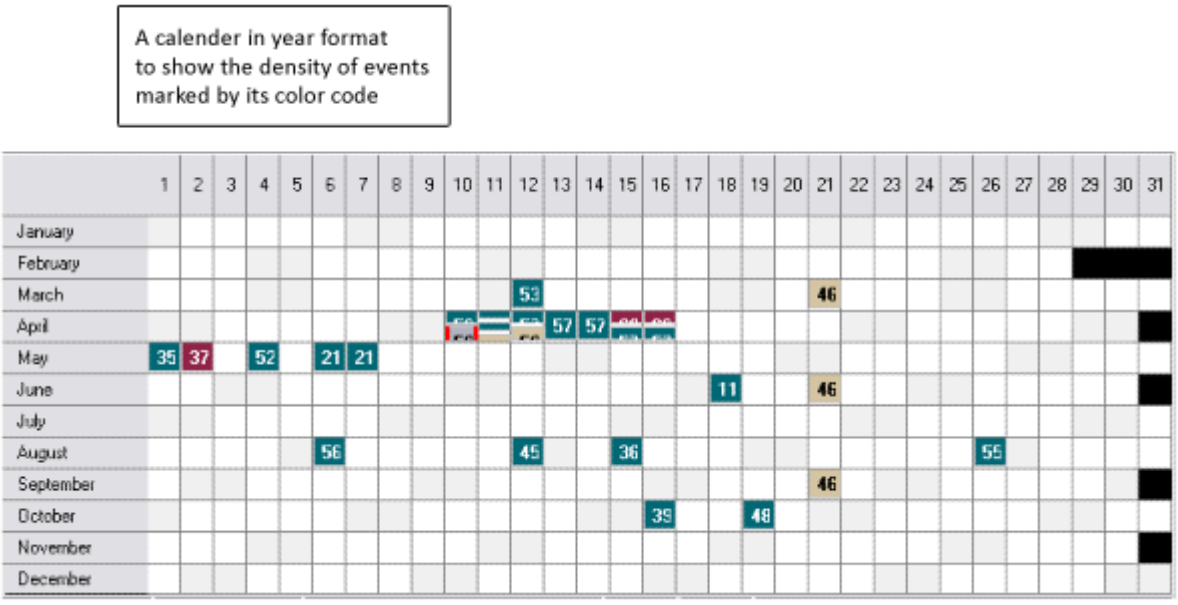


5.1.2 Budget Management Interface

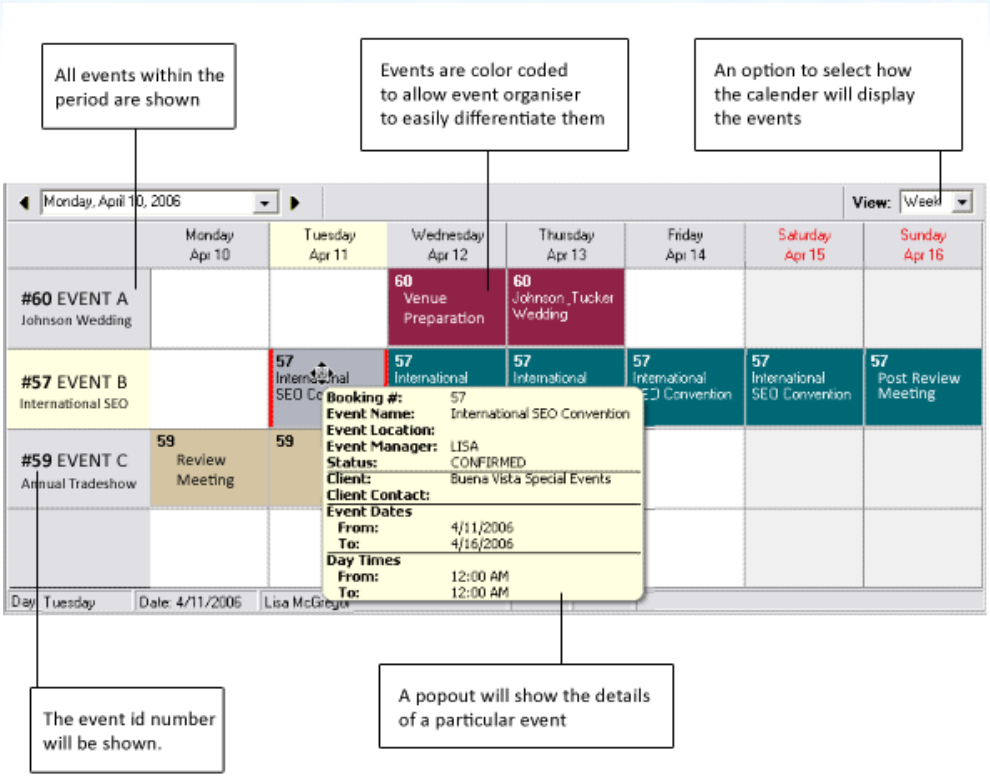


5.1.3 Calendar Interface

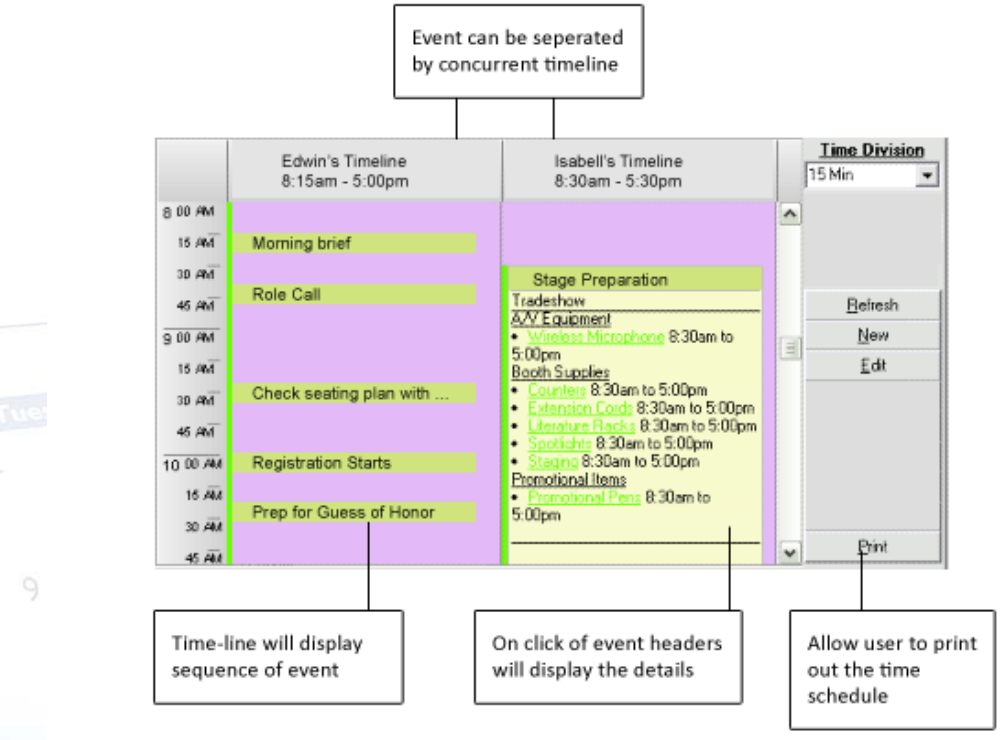
5.1.3.1 Viewing by year.



5.1.3.2 Viewing by week.



5.1.3.3 Viewing an event's timeline.



5.2 Hardware Interfaces

None.

5.3 Software Interfaces

5.3.1 Google Map Server

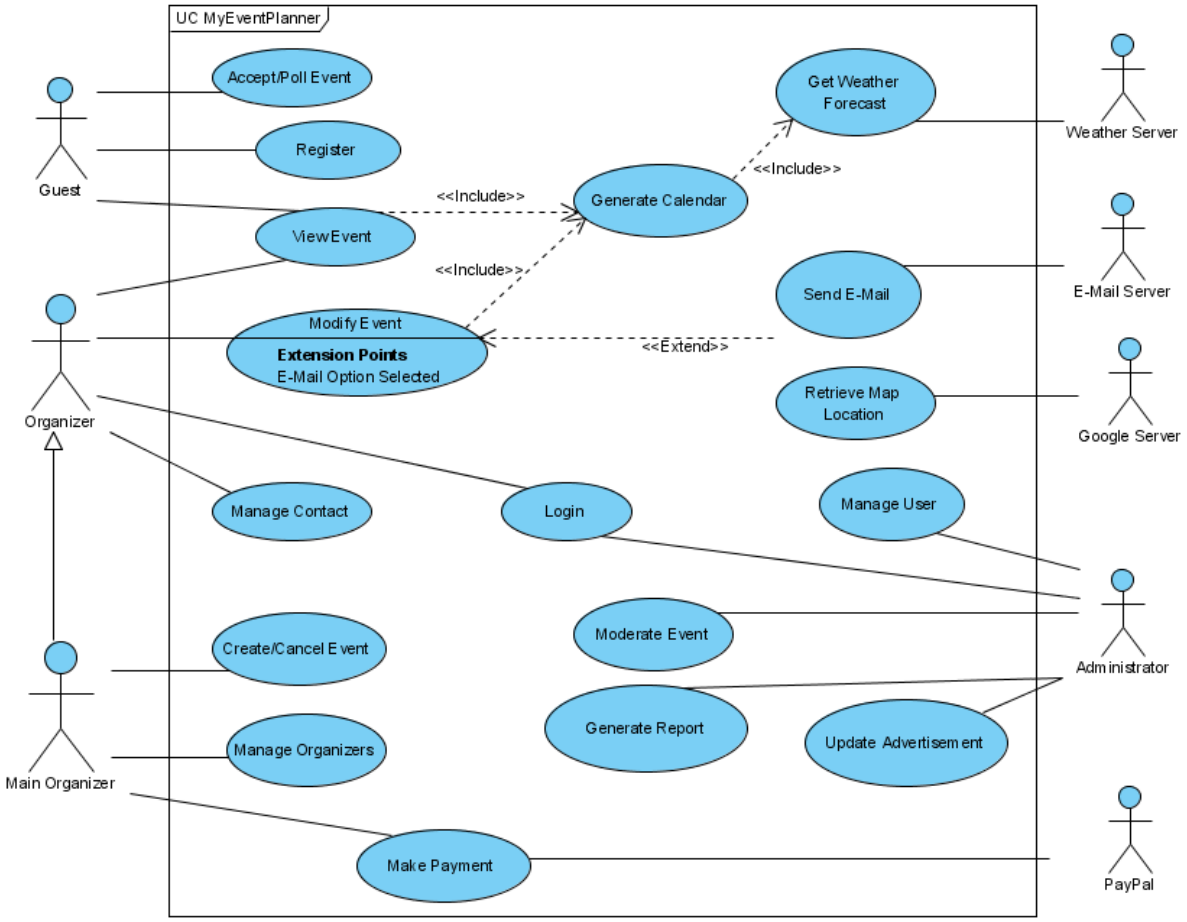
The software will interact with google map server via google map api and api key. The software will send its selected coordinates of the venue to google map server and a map centered at the coordinates will be displayed

5.3.2 NEA RSS

The software will retrieve the RSS feed from NEA. With the use of XMLparser, we will be able to extract the data from the feed.

6. Use Case Model

USE CASE DIAGRAM



6.1 Use Case Description

Use Case ID:	**Use Case 1**
Use Case Name:	**Accept/Poll Events**
Actors:	Guest
Description:	▪ **Guest –** To view the event details and decide whether to accept or reject the event/poll for the event. ▪ **Organizer –** To add the guests who are invited to the event/involved in the poll and view the accepted list of guests/the poll results. ▪ **E-Mail Server –** To send out notification/polling emails to guests for their inputs.
Preconditions:	▪ Organizer has added his/her guests into the guest list. ▪ Organizer has selected a choice to inform guests of event.
Postconditions:	▪ Guest will receive an email with the event descriptions. ▪ Guest will have an individual link from the email to a webpage for further actions.
Trigger:	▪ Main Organizer selected to inform his/her guests during Add Event or during Modify Event.
Normal Flow:	1. Guest click on the link mailed. 2. The system checks link and retrieves event info to display to the user. 2a. If event is on poll, system retrieves poll info and displays poll options. 3. Guest selects to accept invitation. 3a. If event is on poll, guest selects poll options. 4. System records participation information and displays confirmation to user.
Alternative Flow:	1. The link entered is invalid, the use case ends. 2. Guest selects to reject invitation. 2a. System records reject information and displays confirmation to user, the use case ends.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 2**
Use Case Name:	**View Events**
Actors:	Guest, Assistant Organizer
Description:	▪ **Guest –** To view a list of events which are marked as open to public, so that they can sign-up for. ▪ **Assistant Organizer –** To select only to view a list of events which they are currently organizing in order for updates of changes to event.
Preconditions:	Nil
Postconditions:	▪ MyEventPlanner will display a calendar with all the public events that are scheduled. ▪ Events that the organizers are organizing are displayed.
Trigger:	▪ Guest or Assistant Organizer clicks on the view events button.
Normal Flow:	1. Guest or Organizer clicks on view events. 2. MyEventPlanner checks if user is a Guest or Organizer to determine which events to display. 2a. <u>User is a guest:</u> MyEventPlanner sets the events to display as only public events, the use case continues. 2b. <u>User is an organizer:</u> MyEventPlanner checks if the 'Display only my events' checkbox is checked, if so it displays only organizer's events, or it will display all public events as well as organizer's events, the use case continues. 3. MyEventPlanner displays a calendar with the relevant events.
Alternative Flow:	1. No events available: MyEventPlanner displays a page stating that no events are available.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 3**
Use Case Name:	**Manage Contacts**
Actors:	Organizer
Description:	▪ **Organizers** – To insert, delete or update contacts easily.
Preconditions:	User must be logged in to the system.
Postconditions:	Nil
Trigger:	▪ Organizer clicks on the manage contacts button.
Normal Flow:	1. Organizer clicks on the manage contacts button. 2. MyEventPlanner opens up a page listing all existing contacts for the organizer. *The following Organizer action and MyEventPlanner responses may be done repeatedly.* 3. Organizer selects to update a particular contact, or to insert or delete contacts. 4. MyEventPlanner displays contact details for update. 5. Organizer updates the required information and submits. 6. MyEventPlanner acknowledges the update and records update. 7. *The following action(s) ends the use case. 8. Organizer clicks on finish button. 9. MyEventPlanner records changes and brings user back to homepage.
Alternative Flow:	1. Organizer selects contact(s) to delete. 2. MyEventPlanner prompts for confirmation to delete selected contact(s). 3. Organizer confirms deletion. 4. MyEventPlanner deletes selected contacts. 5. Organizer selects to insert contact(s). 6. MyEventPlanner displays page to insert contacts by manual entry or import. 7. Organizer uploads file to import or enters contact(s) to insert and submits. 8. MyEventPlanner acknowledges inserted records.
Exceptions:	**I5.E.1./3.2.E.1** ▪ Information contains illegal values. ▪ MyEventPlanner prompts user of invalid fields and brings user back to the previous page. **3.2.E.2** ▪ File imported is of unsupported type. ▪ MyEventPlanner prompts user of unsupported file type and brings user back to manage contacts page.
Include:	Nil

Use Case ID:	**Use Case 4**
Use Case Name:	**Modify Events**
Actors:	Guest, Assistant Organizer
Description:	▪ **Organizers –** To update details of events when changes arise. ▪ **Guests –** Guests can check the updated information of the event.
Preconditions:	▪ User must be logged in to the system, and have event(s) created under them.
Postconditions:	▪ Updated event information is recorded.
Trigger:	▪ Organizer clicks on the modify events button.
Normal Flow:	1. Organizer clicks on the modify events button. 2. MyEventPlanner opens page listing all events controlled by the organizer. 3. Organizer selects a particular event to update. 4. MyEventPlanner displays details of events for update. 5. Organizer updates the required information and submits. 6. MyEventPlanner acknowledges the update and records update, and prompts organizer if he wishes to send E-mail to guests to notify them of changes, if so the use case extends to Send Email use case.
Alternative Flow:	1. MyEventPlanner fails to fine events associated to organizer, the use case ends. 2. Organizer cancels the modifications, the use case ends.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 5**
Use Case Name:	**User Login**
Actors:	Guest
Description:	▪ **Guest –** To login as valid organizer.
Preconditions:	User is not logged in
Postconditions:	User will be logged in
Trigger:	▪ Guest clicks on Login button.
Normal Flow:	1. Guest enters valid username and password and clicks login. 2. System authenticates user login credentials. 3. System stores user info in session and forwards user to home page
Alternative Flow:	2a. Username/password is empty when clicked on login. 2a1. Page direct to login page. 2a2. Show error message stating fields cannot be empty. 3a. System couldn't authenticate user. 3a1. Page direct to login page. 3a2. Show error message "Username/Password is wrong".
Exceptions:	Nil
Include:	Nil
Business Rules:	▪ Valid user name is a valid email address ▪ Valid password must be alphanumeric of 6-10 characters ▪ Valid login, user name must be present in the database

Use Case ID:	**Use Case 6**
Use Case Name:	**Make Payment**
Actors:	Main organizer, Paypal
Description:	▪ **Main organizer –** To upgrade its privilege to premium member and pay for the upgrade via paypal.
Preconditions:	▪ Premium type is selected and amount needed to pay is passed as session variable.
Postconditions:	▪ Main organizer will be auto login with the privilege that was just applied.
Trigger:	▪ Guest clicks on Upgrade button.
Normal Flow:	1. Select premium type 2. Click on upgrade. 3. System will direct page to paypal. 4. Enter credit card information. 5. Click on confirm payment. 6. System return page to Event page with change of premium privilege.
Alternative Flow:	4a. Credit card information is invalid, use case goes back to step 3.
Exceptions:	Nil
Include:	Nil
Business Rules:	▪ Valid credit card information

Use Case ID:	**Use Case 7**
Use Case Name:	**Moderate Event**
Actors:	Administrator
Description:	▪ **Administrator –** To modify the database. ▪ **Users –** To have inappropriate event.
Preconditions:	▪ User is logged in as administrator.
Postconditions:	▪ The database is modified only if the administrator confirms the changes. ▪ Computer record edits are always saved unless the Administrator cancels the transaction.
Trigger:	▪ Administrator clicks on moderate event button.
Normal Flow:	1. Administrator clicks on moderate event button. 2. MyEventPlanner displays details of all events. 3. Administrator selects event(s) to suspend. 4. MyEventPlanner asks for confirmation. 5. Administrator confirms the changes. 6. MyEventPlanner records the modifications and informs the Administrator of change.
Alternative Flow:	1. Administrator cancels the operation, the use case ends. 2. Administrator does not confirm the changes, the use case ends.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 8**
Use Case Name:	**Manage User**
Actors:	Administrator
Description:	▪ Administrator – To modify the user accounts easily. ▪ Users – To have their account information updated.
Preconditions:	▪ User is logged in as administrator.
Postconditions:	▪ Modifications are recorded if the administrator confirms the changes.
Trigger:	▪ Administrator clicks on manage user button.
Normal Flow:	1. Administrator clicks on manage user button 2. MyEventPlanner displays list of all users and their account information. 3. Administrator selects user(s) to suspend or delete from the system. 4. MyEventPlanner asks for confirmation. 5. Administrator confirms the changes. 6. MyEventPlanner records the modifications and informs the Administrator of change.
Alternative Flow:	1. Administrator cancels the operation, the use case ends. 2. Administrator does not confirm the changes, the use case ends.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 9**
Use Case Name:	**Generate Calendar**
Actors:	None
Description:	▪ Allow user to view calendar
Preconditions:	Nil
Postconditions:	Nil
Trigger:	▪ EventController – Request to display calendar to user.
Normal Flow:	1. MyEventPlanner checks for public or private viewing of events 2. MyEventPlanner gather all events required. 3. MyEventPlanner get the current date. 4. MyEventPlanner generates an array of boxes to represent the calendar according to the specified format, days, weeks or months.
Alternative Flow:	Nil
Exceptions:	2a Public calendar is selected: 2a1. Gather all events which are marked public and have attendee of more than 50.
Include:	Nil

se Case ID:	**Use Case 10**
Use Case Name:	**Send E-mail**
Actors:	Email Server
Description:	▪ **Main/Assistant organizer** – To send email notification of any changes of the event to the guest/attendees
Preconditions:	▪ Contact list has been uploaded into the database server by main/assistant organizer.
Postconditions:	▪ Guest will be aware of any updates regarding the event.
Trigger:	▪ Modification made to existing event.
Normal Flow:	1. Main/assistant organizer modified an event. 2. System verifies the changes and asks for confirmation from main/assistant organizer. 3. Main/assistant organizer confirms the changes 4. Main/assistant organizer sent emails to the list of contacts that was uploaded into the email server. 5. Email server sent the emails and system informs main/assistant organizer.
Alternative Flow:	1. Main/assistant organizer cancels the operation. The use case ends. 1. No event created: 1a. System alerts main/assistant organizer no event was found. 4. No contact list found: 4a. System alerts main/assistant organizer no contact list was found, the use case ends.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 11**
Use Case Name:	**Get Weather Forecast**
Actors:	Weather Server
Description:	▪ **Guest –** To view the weather forecast on the event day. ▪ **Assistant Organizer –** To view the weather forecast on the event day. ▪ **Main Organizer –** To view the weather forecasts over a period of 3 days before deciding which day to plan the event.
Preconditions:	Nil
Postconditions:	▪ MyEventPlanner is updated with 3 days of weather outlook.
Trigger:	▪ Main Organizer clicks on Get Weather Forecast.
Normal Flow:	1. Main Organizer clicks on Get Weather Forecast. 1a. MyEventPlanner will open a popup page. 1b. Retrieve weather forecasts from weather server (e.g. NEA) via RSS feed. 2. MyEventPlanner display 3 days of weather outlook from today. 2a. MyEventPlanner will update the weather forecasts in the database
Alternative Flow:	Nil
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 12**
Use Case Name:	**Retrieve Map Locations**
Actors:	Google Server
Description:	▪ **Guest –** To view the event location. ▪ **Assistant Organizer –** To view the event location. ▪ **Main Organizer –** To select the location list of sponsors' venues or add own custom venue and select from the location list of custom venues.
Preconditions:	Nil
Postconditions:	▪ My Event Planner will store the location selected by the main organizer.
Trigger:	▪ Main Organizer clicks on Retrieve Map Locations.
Normal Flow:	1. Main Organizer clicks on Retrieve Map Locations. 1a. My Event Planner will open a popup page. 1b. My Event Planner will retrieve maps from Google server and center on Singapore. 2. Main Organizer selects the location list of sponsors' venues or adds own custom venue and selects from the location list of custom venues. 2a. Main Organizer adds his/her own custom venue by double clicking the location on the map and enter a name for the location. 2b. The new added location will be stored into his/her records for future references and displayed in the location list of custom venue. 2c. Main Organizer will select a venue either from the location lists of sponsors' or custom venue. 2d. Once the location is selected, the map will center on the selected venue. 3. Main Organizer submits the selected venue.
Alternative Flow:	Nil
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 13**
Use Case Name:	**Generate Report**
Actors:	Administrator
Description:	▪ **Administrator –** To view statistic report regarding the website
Preconditions:	▪ Statistic figures and values are stored in system database. ▪ User logged in as administrator.
Postconditions:	▪ Administrator received statistic report.
Trigger:	▪ Administrator requests system for the statistic report.
Normal Flow:	1. Administrator accessing the system to request statistic report 2. System verifies the identity and proceeds on. 3. Administrator clicks the button, “Generate Report”. 4. System request from database for the figures and values. 5. System show administrator statistic report.
Alternative Flow:	1. Administrator cancels the operation. The use case ends. 2. Invalid access by administrator: 2a. System alerts administrator for correct username and password. 2b. Administrator enters the correct username and password and either resumes basic flow or return to step 2a.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 14**
Use Case Name:	**Update Advertisement**
Actors:	Administrator
Description:	▪ **Administrator –** To update advertisement banners that will be posted on the website.
Preconditions:	▪ Have a ready pool of sponsors to advertise their products.
Postconditions:	▪ Sponsor's advertisement banners will be updated.
Trigger:	▪ Upload advertisement of sponsors' product.
Normal Flow:	1. Administrator accessing the system to upload advertisement. 2. System verifies the identity and proceeds on. 3. System request administrator to upload the advertisement (flash, image, photos). 4. System verifies the content with the administrator. 5. System prompt administrator for confirmation. 6. Administrator acknowledges and updates advertisement. 7. System verifies advertisement had been uploaded to website.
Alternative Flow:	1. Administrator cancels the operation. The use case ends. 2. Invalid access by administrator: 2a. System alerts administrator for correct username and password. 2b. Administrator enters the correct username and password and either resumes basic flow or return to step 2a. 3. Invalid file type uploaded by administrator: 3a. System alerts administrator 3b. Administrator checks for the correct file to upload and either resume basic flow or return to step 4a.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 15**
Use Case Name:	**Manage Organizers**
Actors:	Main Organizer
Description:	▪ **Main Organizer –** To add, edit and delete organizer of an event
Preconditions:	▪ Previously added organizers are stored in system database. ▪ User logged in to the system.
Postconditions:	▪ The organizer list in MyEventPlanner is updated.
Trigger:	▪ Main Organizer clicks on 'Manage Organizers'.
Normal Flow:	1. Main Organizer clicks on 'Manage Organizers' for a selected event 2. Main Organizer select 'Add New Organizer' or select an organizer from its list before click on 'Edit Selected Organizer' or 'Deleted Selected Organizer' 3. Main Organizer fills up the details for both 'Add New Organizer' and 'Edit Selected Organizer' before click on 'Submit' or confirmation of deletion will be displayed. 4. Main Organizer will be returned back to the page where he/she is able to view the updated list of organizers.
Alternative Flow:	2a. Main Organizer select 'Add New Organizer' or select an organizer from its list before click on 'Edit Selected Organizer' or 'Deleted Selected Organizer' 2a1. Main Organizer do not select 'Add New Organizer', 'Edit Selected Organizer' or 'Deleted Selected Organizer' 2a2. He/she views the current list of organizers and their details 3a. Main Organizer fills up the details for both 'Add New Organizer' and 'Edit Selected Organizer' before click on 'Submit' or confirmation of deletion will be displayed. 3a1. Main Organizer chooses 'No' upon confirmation of deletion. 3a2. Main Organizer will be returned to page where he/she first clicks on 'Manage Organizers' where he/she first clicks on 'Manage Organizers'.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 16**
Use Case Name:	**Create/Cancel Event**
Actors:	User
Description:	▪ Users – To create or cancel an event
Preconditions:	▪ User must be logged in to the system
Postconditions:	▪ An event will be created or removed
Trigger:	▪ User clicks on create or cancel event button.
Normal Flow:	1. User clicks on create event button 2. MyEventPlanner prompts for new event details. 3. User keys in the details and clicks 'Create'. 4. MyEventPlanner saves the event under the user's account and displays the new event's sub-page.
Alternative Flow:	1. User clicks on cancel event button 2. MyEventPlanner displays list of all events created by user. 3. User selects the event to delete and clicks on 'Delete'. 4. MyEventPlanner asks for confirmation. 5. User confirms the change. 6. MyEventPlanner records the modifications and informs the user of the change. 7. MyEventPlanner will navigate the user to the show-all-events page.
Exceptions:	Nil
Include:	Nil

Use Case ID:	**Use Case 17**
Use Case Name:	**Register**
Actors:	Guest
Description:	▪ Guest – To register for an account as organizer
Preconditions:	▪ User must not be logged in
Postconditions:	▪ An account is created for the guest
Trigger:	▪ User clicks on sign up button
Normal Flow:	1. User clicks on sign up button 2. MyEventPlanner prompts for new event details. 3. User keys in the details and clicks 'Create'. 4. MyEventPlanner saves the event under the user's account and displays the new event's sub-page.
Alternative Flow:	1. User clicks on cancel event button 2. MyEventPlanner displays list of all events created by user. 3. User selects the event to delete and clicks on 'Delete'. 4. MyEventPlanner asks for confirmation. 5. User confirms the change. 6. MyEventPlanner records the modifications and informs the user of the change. 7. MyEventPlanner will navigate the user to the show-all-events page.
Exceptions:	Nil
Include:	Nil

Glossary

MyEventPlanner	The system of interest
Event	A planned activity that is created by a person with date, venue and time
Organizer	Someone that creates and plans the activity
Guest	Someone that views the system or is invited to a created activity
Poll	The registering of votes pertaining to event's choices
Administrator	Someone who manages the system
Notification	An email that notifies the guest
Database	our comprehensive collection of related data organized for convenient access
Sponsor	A company that our website advertises for
Statistical report	A meaningful compilation of figures and values for admin purposes
Calendar	An entity that shows all activities for that month, week or day
Paypal	A secured medium for cashless transaction

7. References

Provide a list of all documents and other sources of information referenced in the SRS and utilized in developing the SRS. Include for each the document number, title, date and author.

Document No.	Document Title	Date	Author

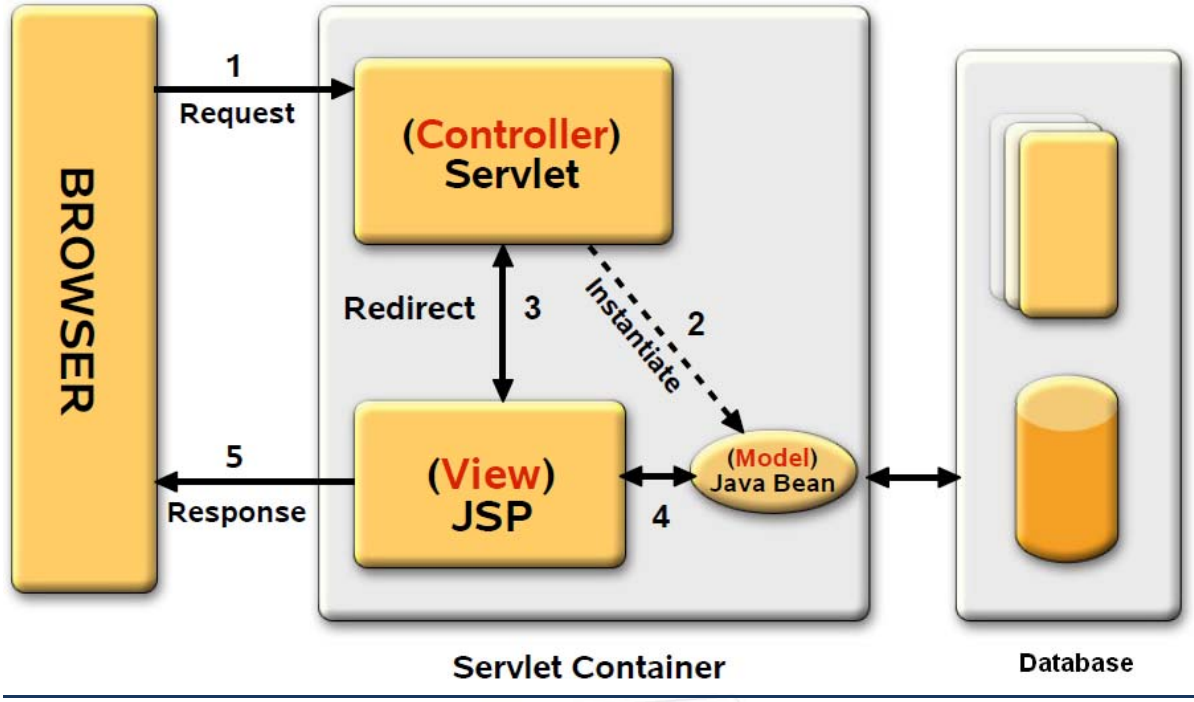
8. Revision History

Identify changes to the SRS.

Version	Date	Name	Description
1.0	25/08/2009	SRS	First draft
1.0a	26/08/2009	SRS	Requirements added
1.1	27/08/2009	SRS(formatted)	Refined requirements and formatting
1.1a	28/08/2009	SRSv1.1a	Added UCDiagram and UCDescriptions
1.1b	29/08/2009	SRSv1.1b	Added UCDescriptions and requirements
1.1c	10/09/2009	SRSv1.1c	Updated Use Case Descriptions
1.2	14/09/2009	SRSv1.2	Added User Interface Specification
1.2a	17/09/2009	SRSv1.2a	Updated Use Case Descriptions
1.2b	17/09/2009	SRSv1.2b	Updated Use Case Descriptions and Data Requirements
1.3	01/11/2009	SRSv1.3	Updated Data Requirements and glossary

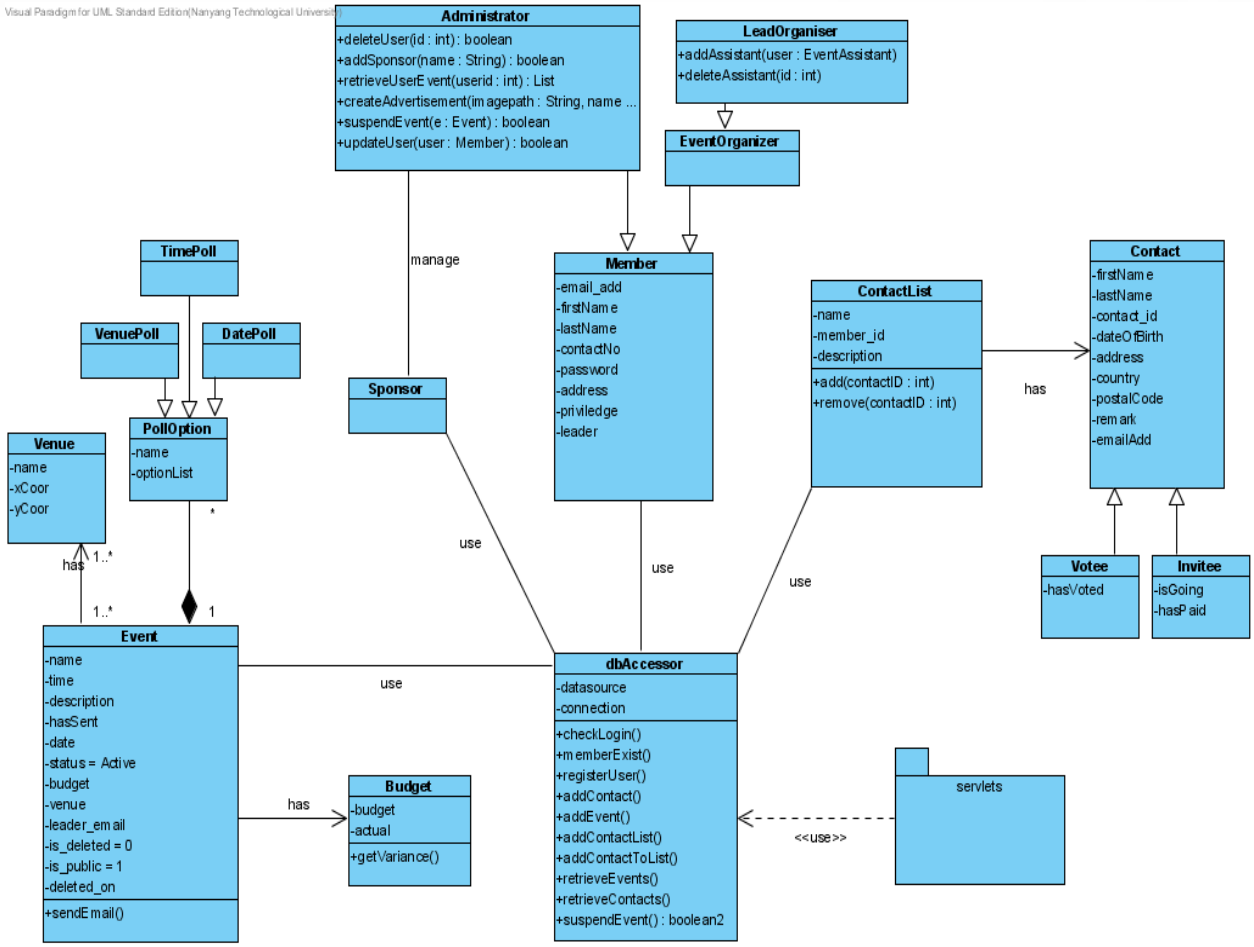
ARCHITECTURAL DESIGN

The project of interest, MyEventPlanner, is developed following the Servlet-centric MVC architecture, where the Model layer would be responsible for modeling the data and behavior of the business entities, the View layer displays information to the clients, and the Controller layer encompasses the business logic and serves as the logical connection between the user's interaction and the business services. By structuring the project with the MVC architecture, the presentation logic is separated from the business logic preventing changes from affecting the whole project. The design implements an integration of a repository architecture concerning how the controller relates to the model. An overall DBAccessor class acts as a centralize model that is in charge of instantiating the other model classes and interaction with the database for all the servlets.



ANALYTICAL MODEL

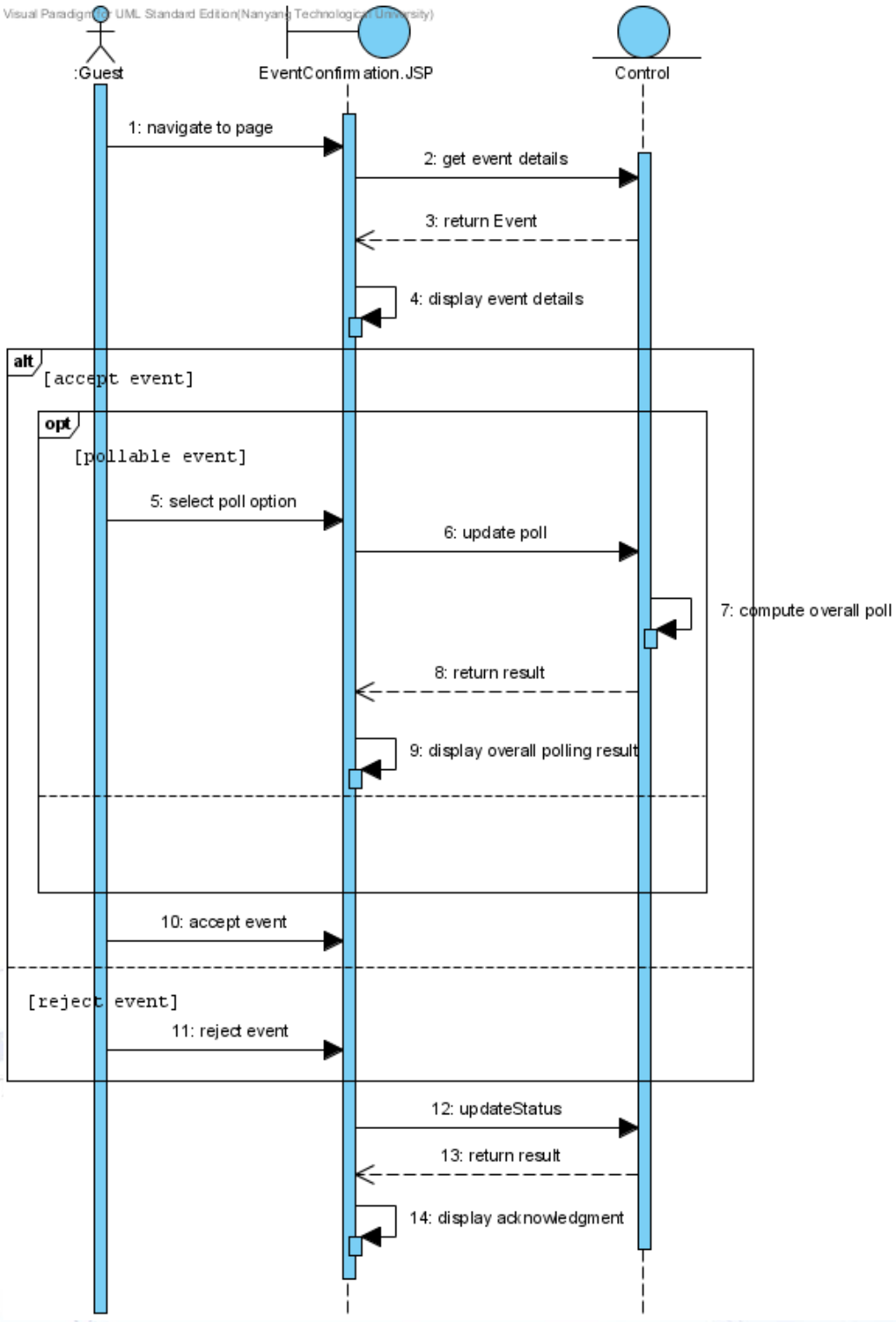
CLASS DIAGRAM



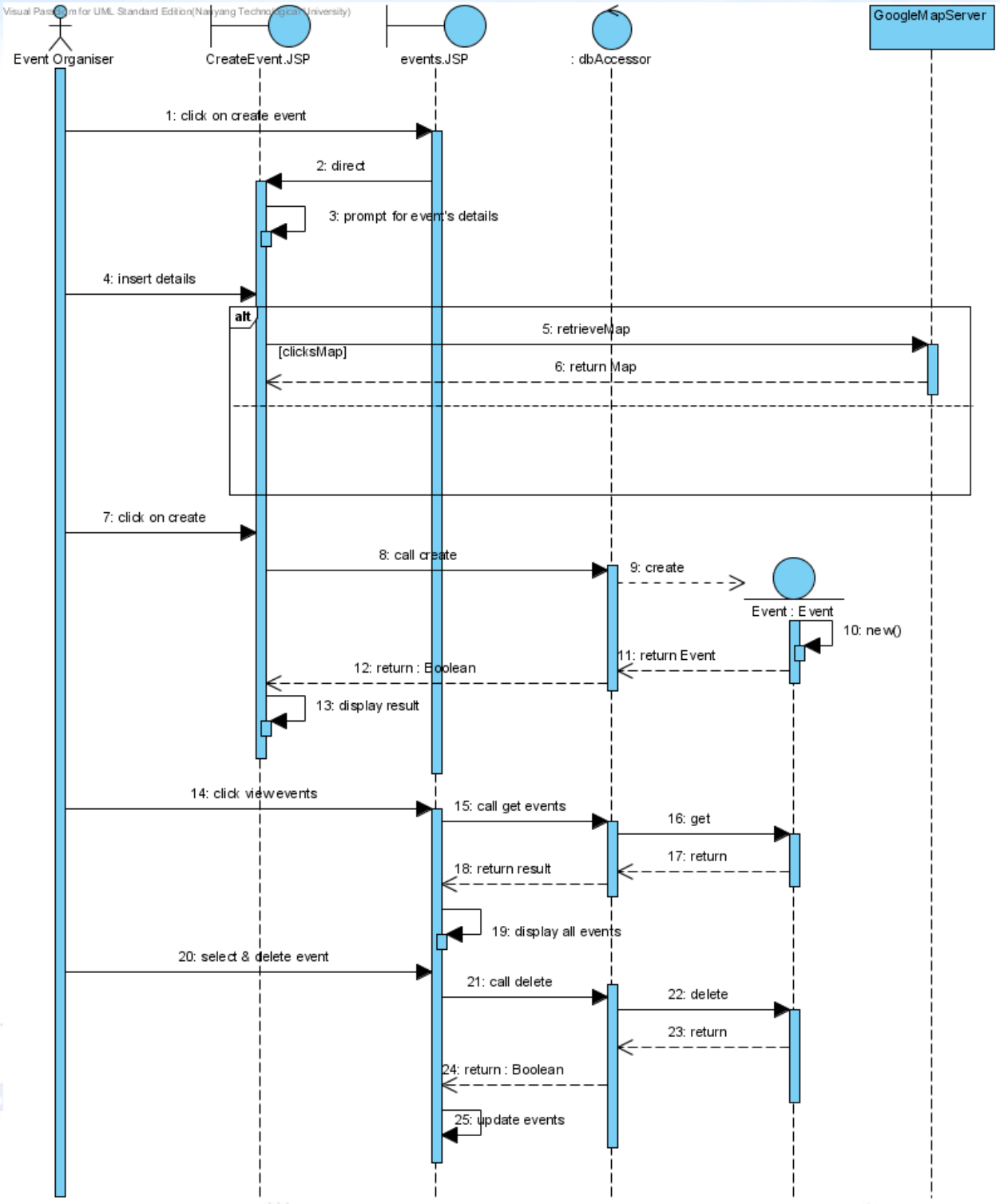
DESIGN MODELS

SEQUENCE DIAGRAMS

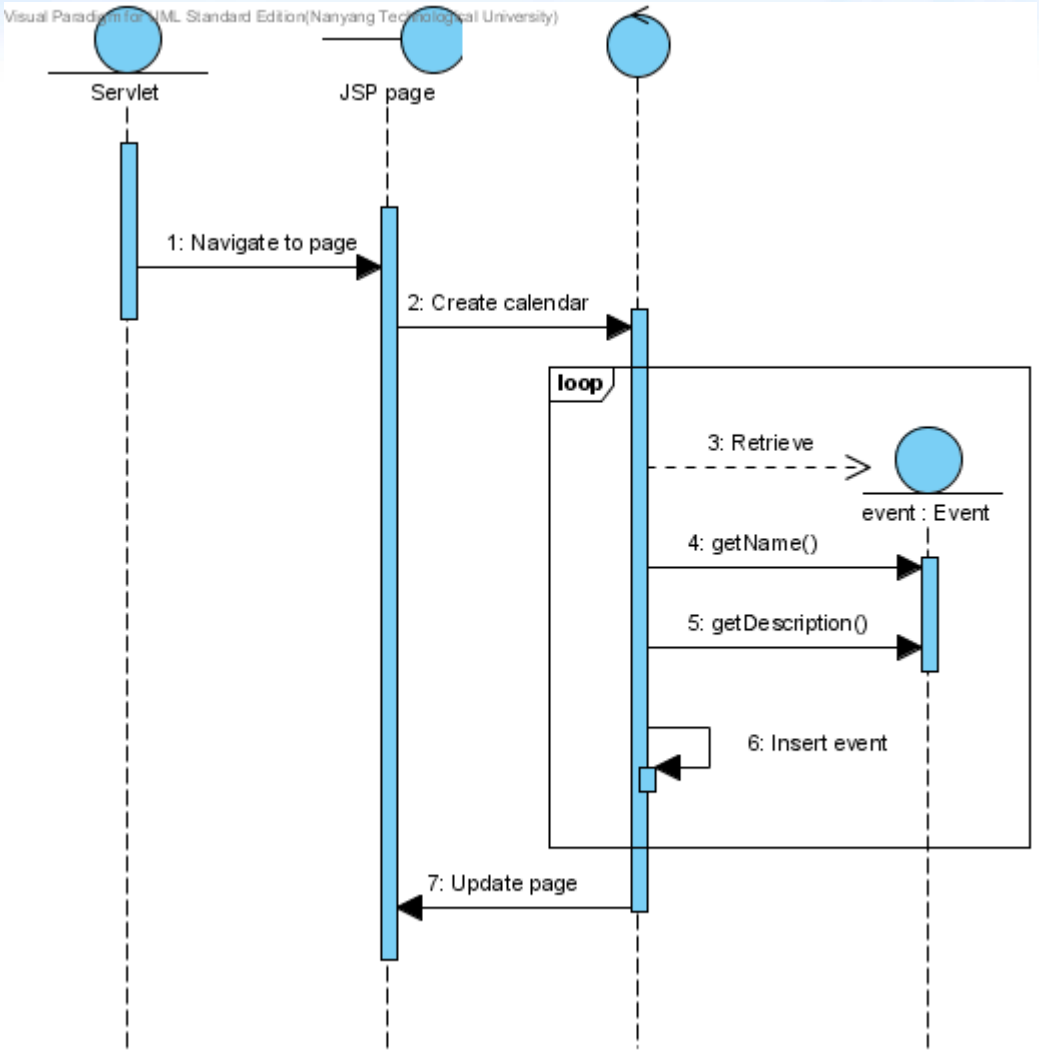
AcceptPoll



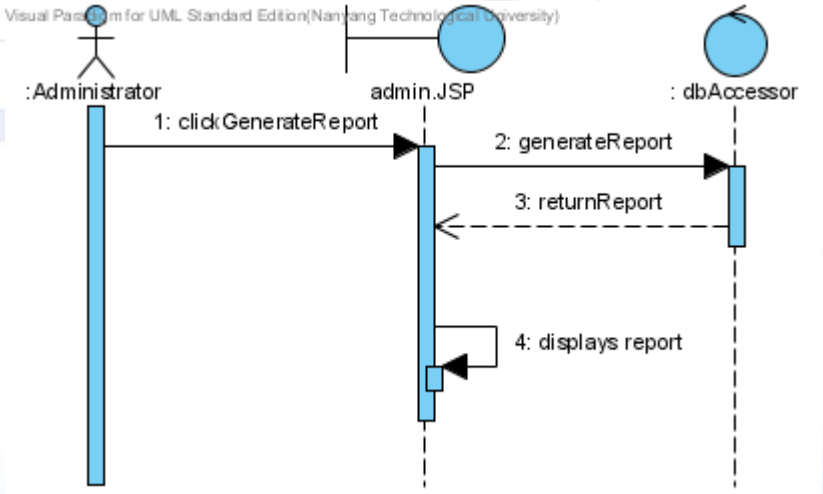
Create_Cancel Event



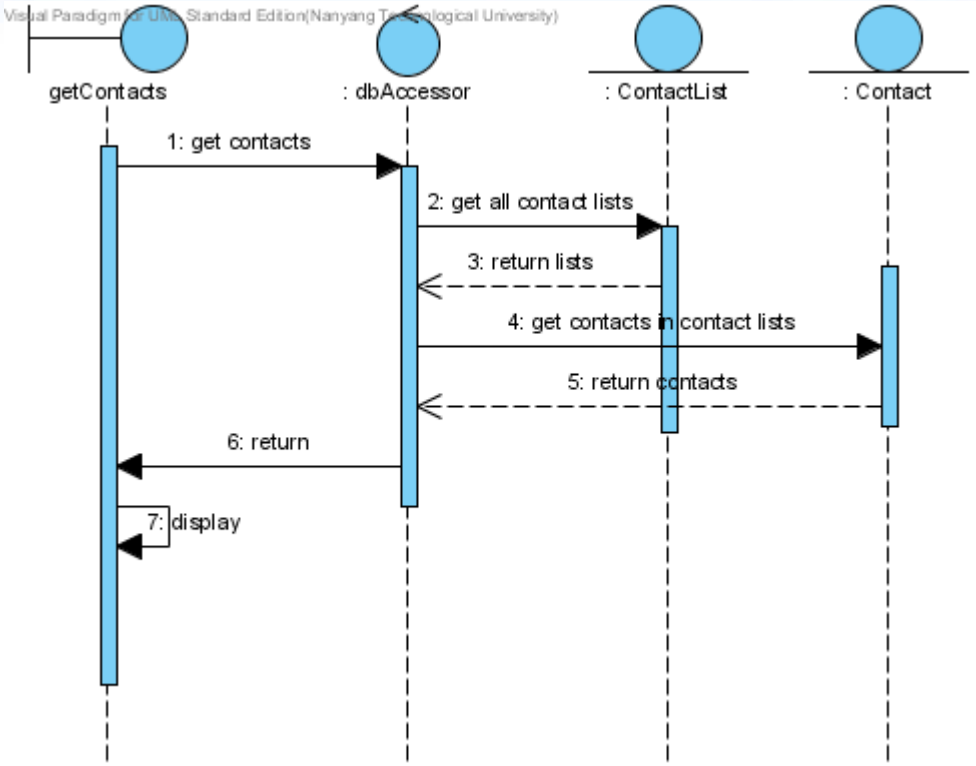
GenerateCalendar



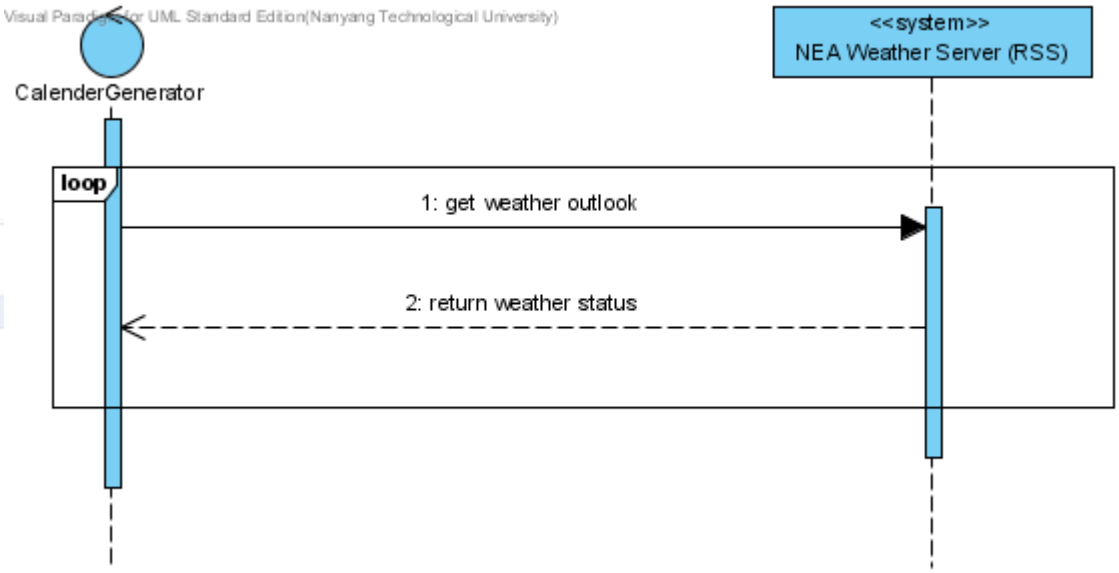
GenerateReport



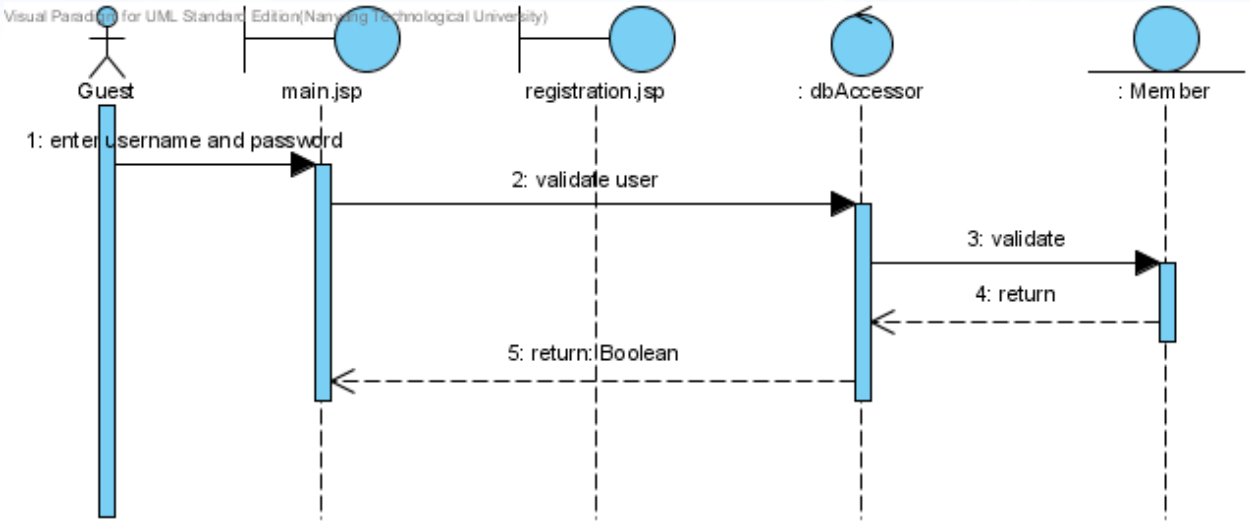
GetContacts



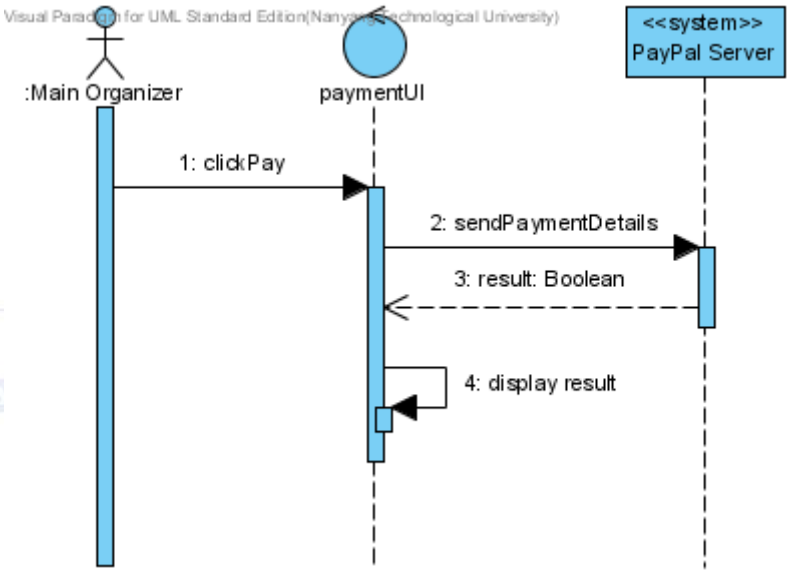
GetWeatherForecast



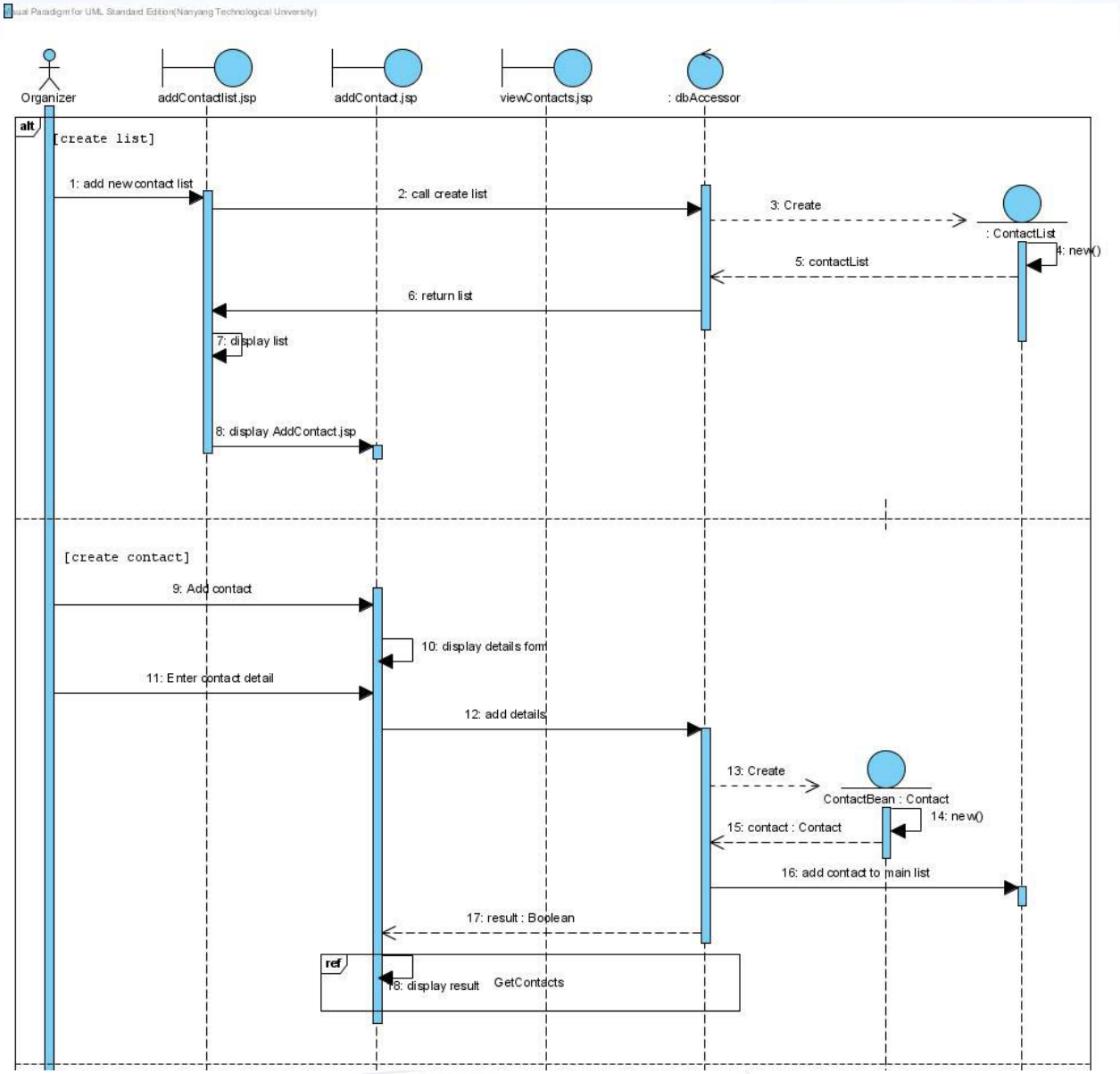
Login

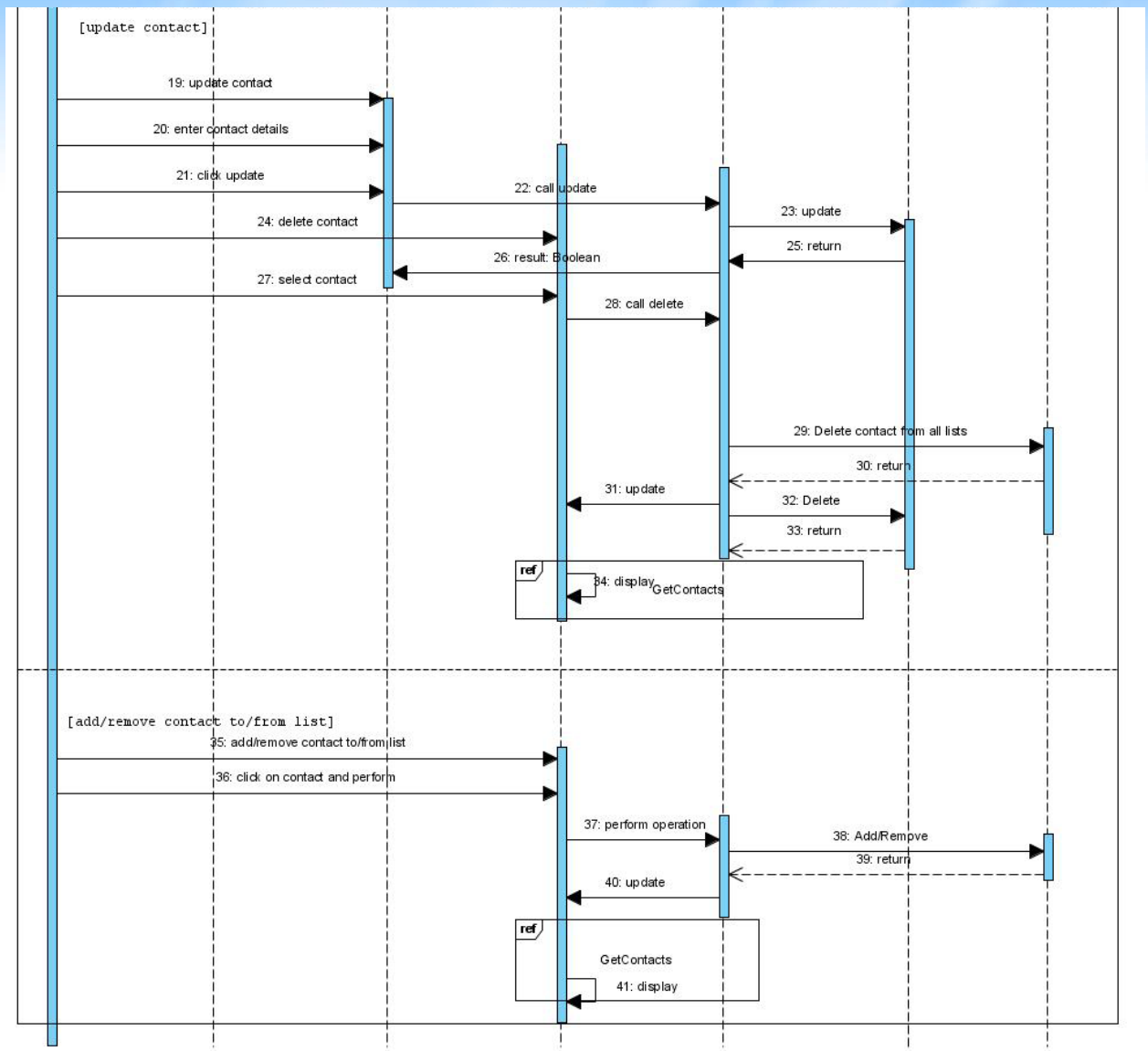


MakePayment

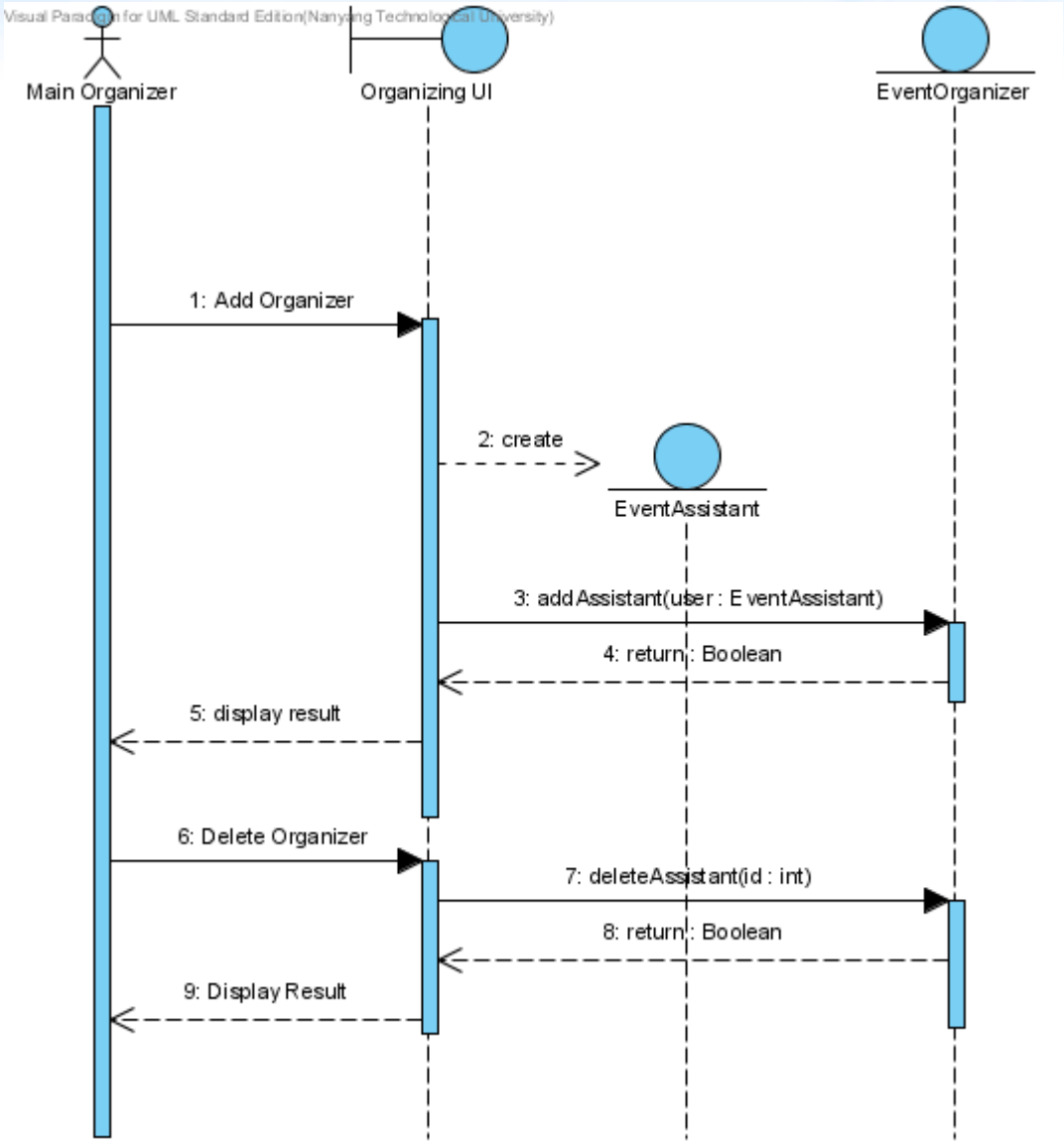


ManageContact

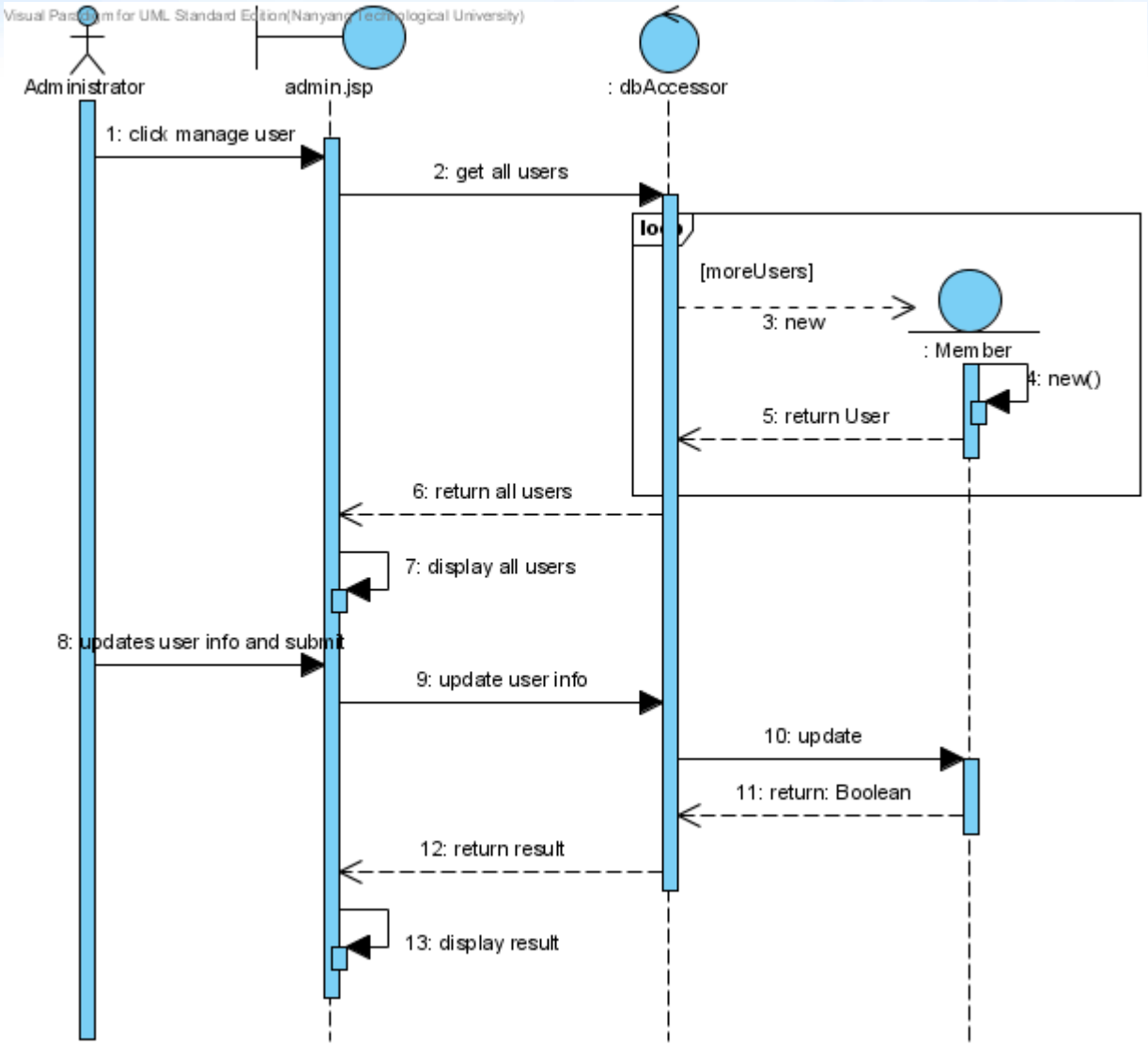



[update contact]
19: update contact
20: enter contact details
21: click update
22: call update
23: update
24: delete contact
25: return
26: result: Boolean
27: select contact
28: call delete
29: Delete contact from all lists
30: return
31: update
32: Delete
33: return
ref
34: display
GetContacts
[add/remove contact to/from list]
35: add/remove contact to/from list
36: click on contact and perform
37: perform operation
38: Add/Rempve
39: return
40: update
ref
GetContacts
41: display

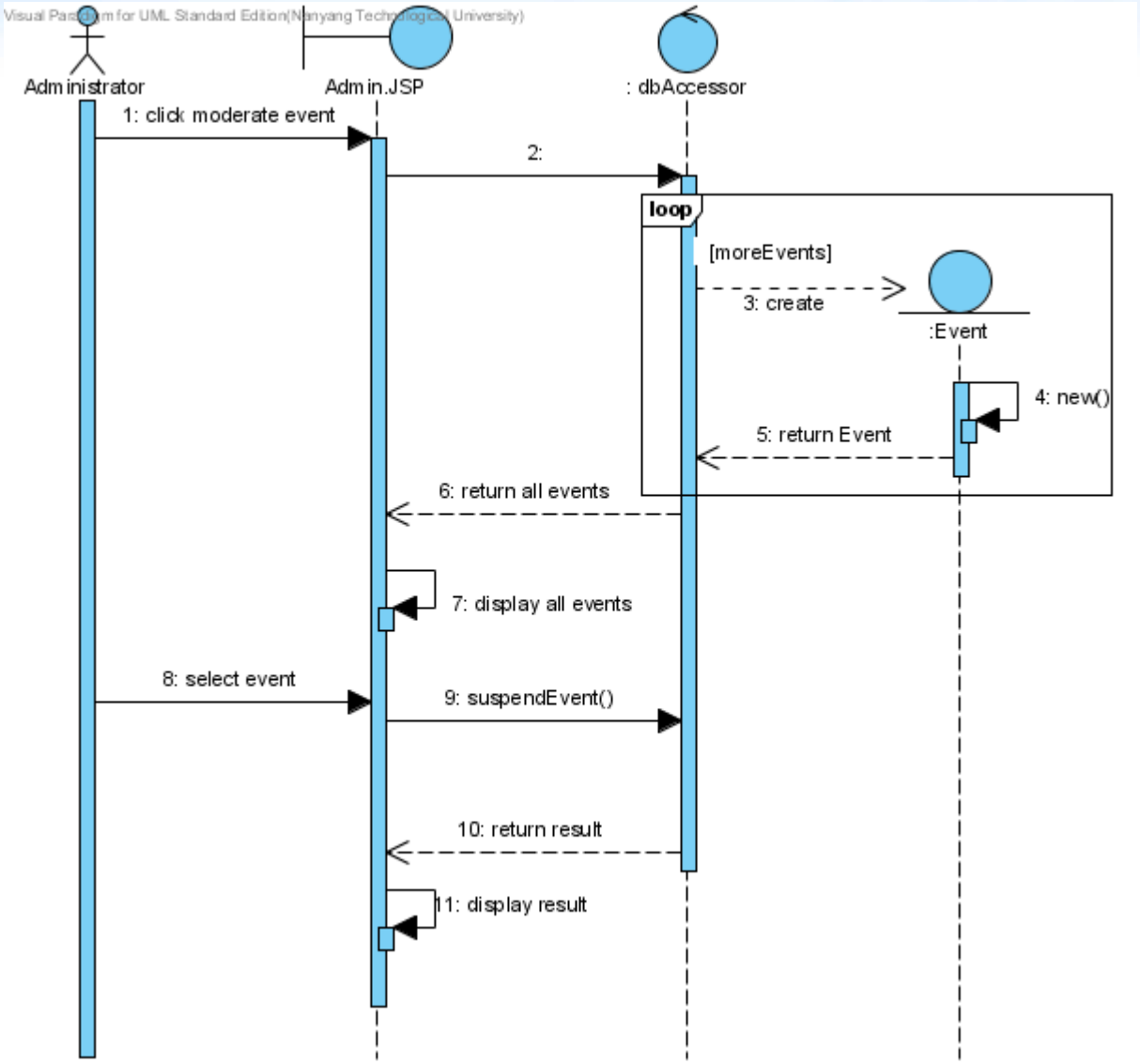
ManageOrganizers



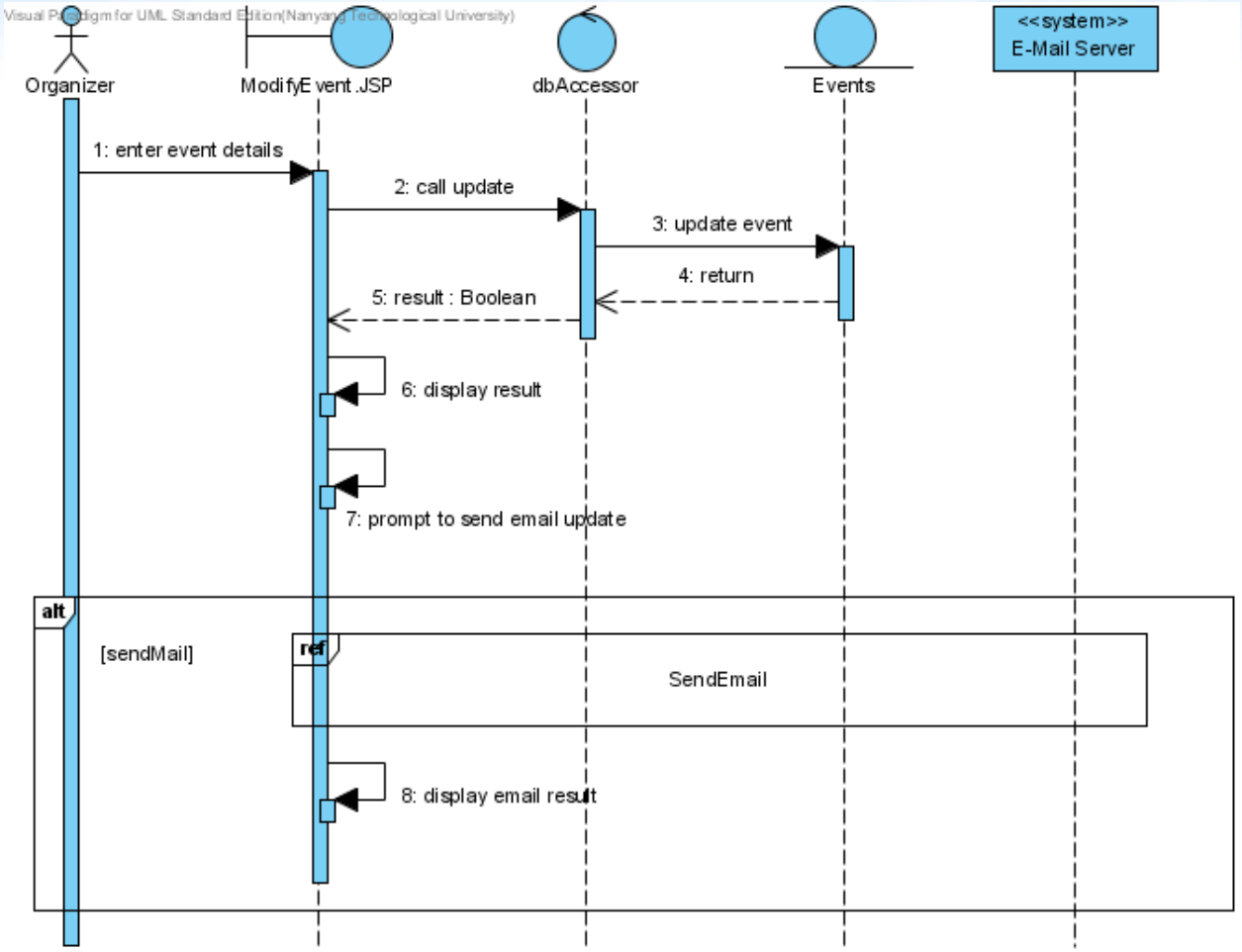
ManageUser


Administrator
admin.jsp
: dbAccessor
1: click manage user
2: get all users
[moreUsers]
3: new
: Member
4: new()
5: return User
6: return all users
7: display all users
8: updates user info and submit
9: update user info
10: update
11: return: Boolean
12: return result
13: display result

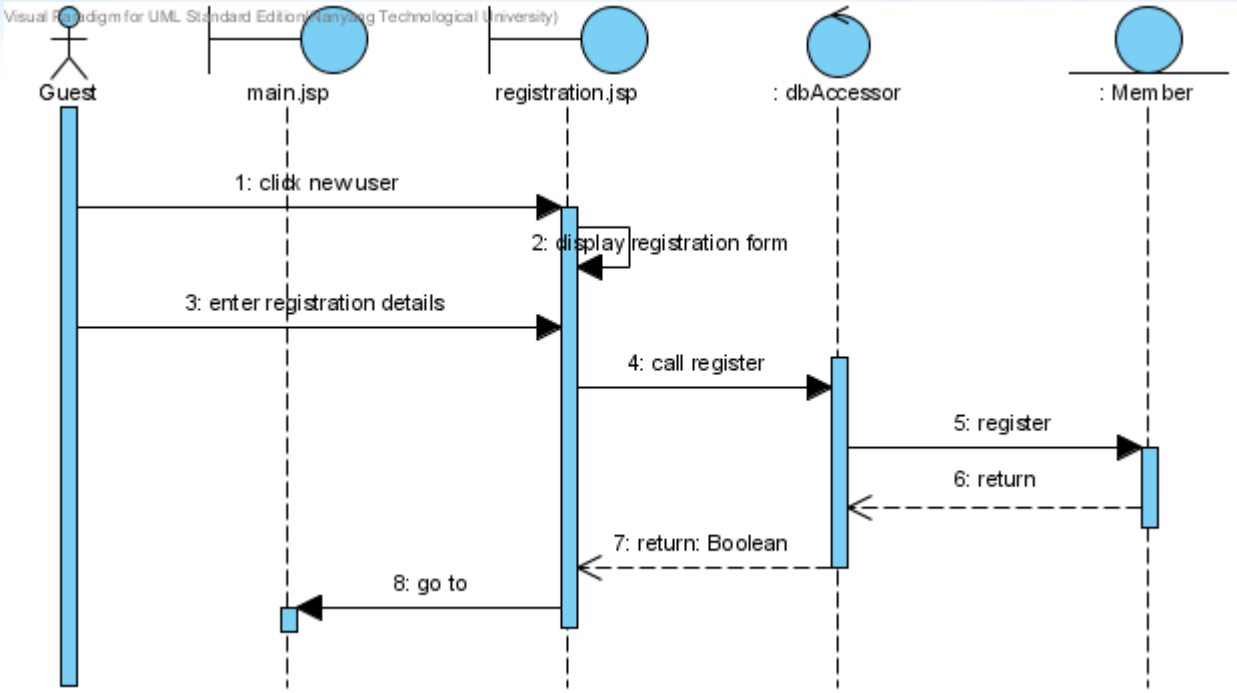
ModerateEvent



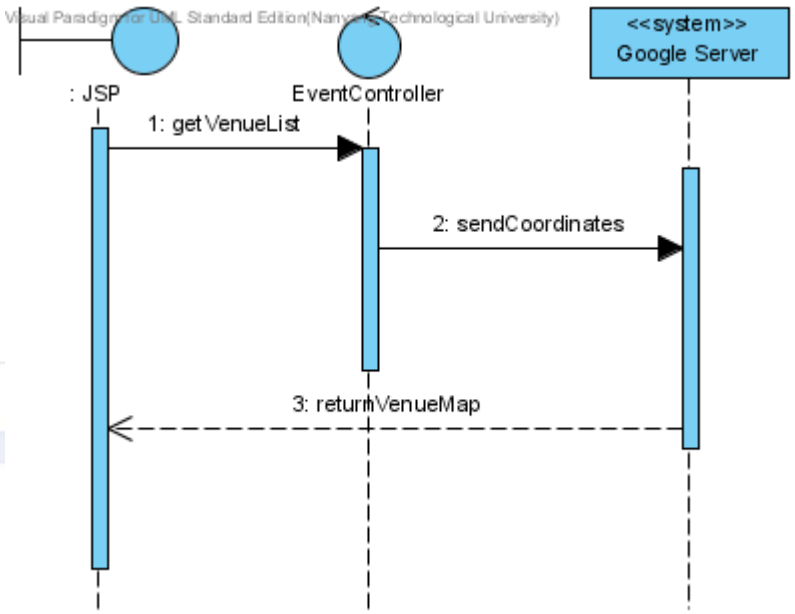
ModifyEvent



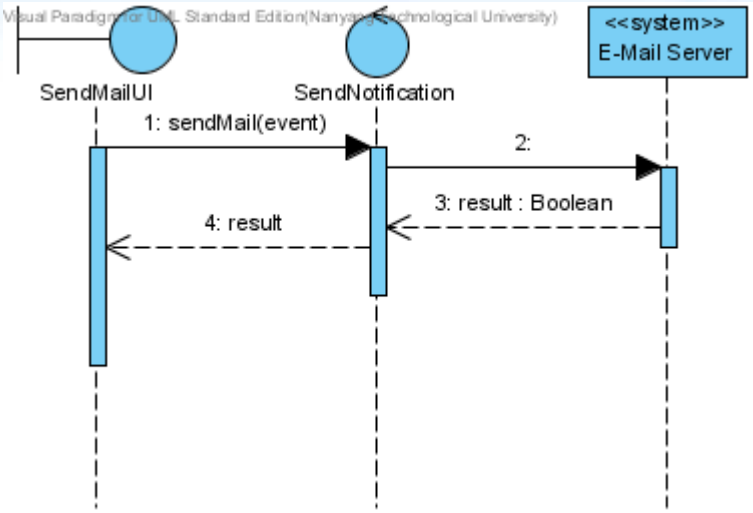
Register



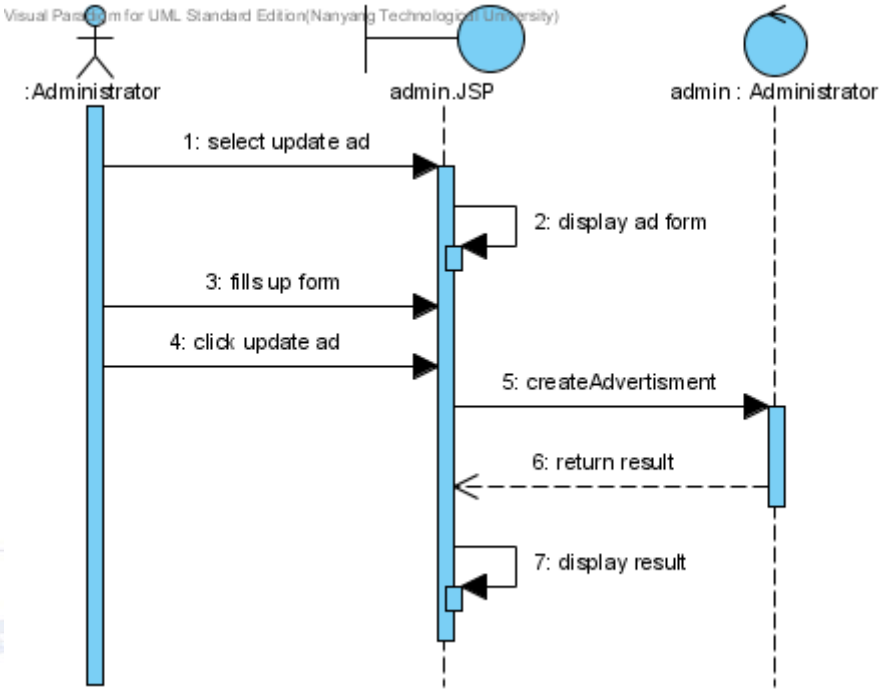
RetrieveMapLocation



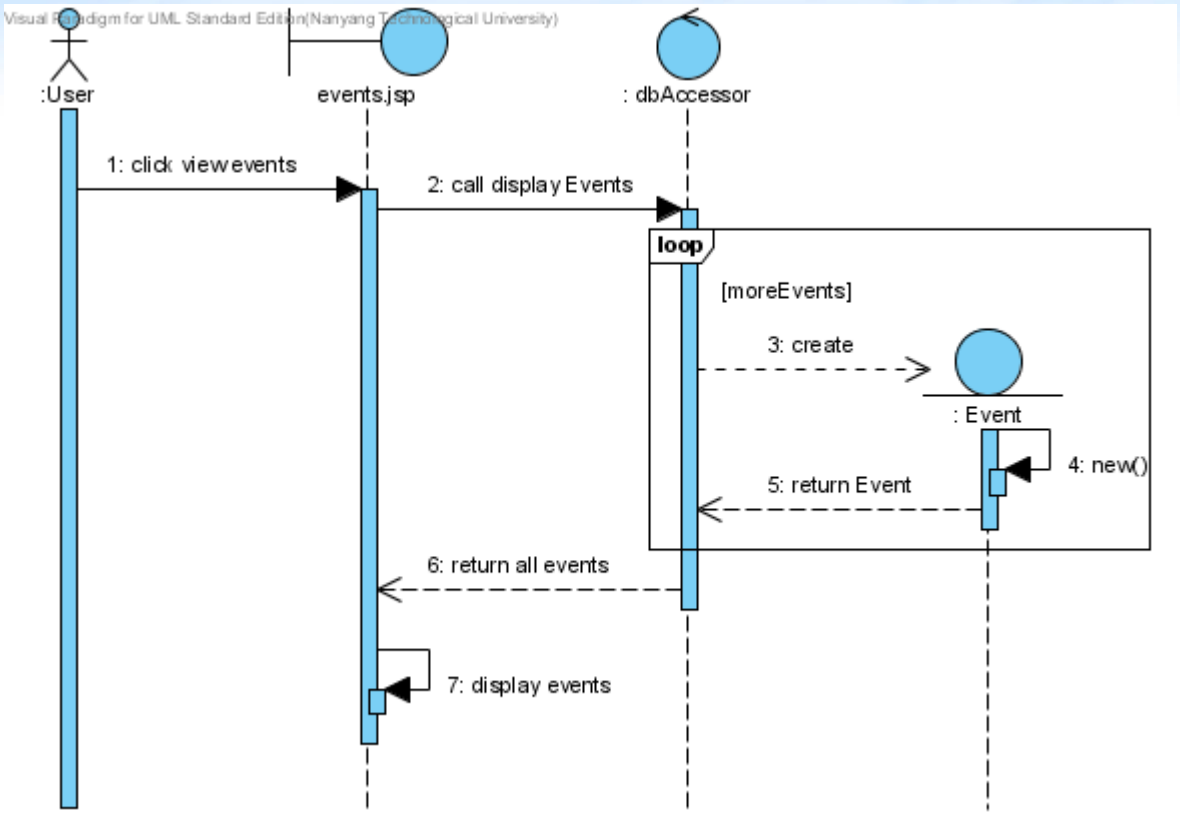
SendEmail



UpdateAdvertisement



ViewEvent


:User
events.jsp
: dbAccessor
1: click viewevents
2: call displayEvents
loop
[moreEvents]
3: create
: Event
4: new()
5: return Event
6: return all events
7: display events

Black Box Testing

LOGIC BASED TESTING

Sub-System tested: Login

Rules	1	2	3	4
Valid username	T	T		
Valid password	T		T	
Login fail		X	X	X
Login success	X			

TEST CASES GENERATED

Test ID	Description	Expected Results	Actual Results
checkLogin1	Precondition: User not logged in Input -Username : Invalid user -Password : Invalid password Subprogram -Login	User fails to login and gets forward back to login page	System inform user of wrong login credentials and forward back to home page
checkLogin2	Precondition: User not logged in Input -Username : valid user -Password : Invalid password Subprogram -Login	System informs user of wrong password and forward back to login page	System inform user of wrong login credentials and forward back to home page
checkLogin3	Precondition: User not logged in Input -Username : Invalid user -Password : Valid password Subprogram -Login	System informs user of invalid user and forward back to login page	System inform user of wrong login credentials and forward back to home page
checkLogin4	Precondition: User not logged in Input -Username : valid user -Password : valid password Subprogram -Login	Login success, system forwards user to home page	Login success, system stores user detail in session and forwards user to home page

EQUIVALENCE PARTITIONING

Sub-System tested: Registration

Input	Input Conditions
First name	Alphanumeric
Last name	Alphanumeric
E-mail	Valid email with @ and .
Password	Alphanumeric of 5 to 15 characters
Home address	Alphanumeric
Country	Within Set specified
Postal code	Positive Integer of length 6
Contact number	Positive Integer of length 8
Account type	Within Set specified

Sub-System tested: Events

Input	Input Conditions
Event name	Alphanumeric
Event date	Date value after system date
Event time	Within Set specified
Event description	Alphanumeric
Event budget	Positive integer
Event venue	Alphanumeric
Event public	Yes or no

Sub-System tested: Contacts

Input	Input Conditions
First name	Alphanumeric
Last name	Alphanumeric
E-mail	Valid email with @ and .
Date of birth	Valid date before system date
Address	Alphanumeric
Postal code	Positive integer of length 6
Country	Alphanumeric
Remarks	Alphanumeric

TEST CASES GENERATED

Test ID	Description	Expected Results	Actual Results
chkReg1	Precondition: User not logged in Input -Username : bill@gates.com -Password : t3jt -First name : Bill -Last name : Gates -Address : Silicon Valley -Country : Singapore -Postal Code : 333333 -Contact Num : 87654321 -Account Type : Personal Subprogram -Registration	Registration fail	Registration fail, system informs user of password length requirement and prompts for another password
chkReg2	Precondition: User not logged in Input -Username : s@s.com -Password : t3jtwe -First name : Bill -Last name : Gates -Address : Silicon Valley -Country : Singapore -Postal Code : 333333 -Contact Num : 87654321 -Account Type : Personal Subprogram -Registration	Registration fail	Registration fail, system informs user of existing account for username and prompts for another username
chkReg3	Precondition: User not logged in Input -Username : bill@gates.com -Password : t3jtyw -First name : -Last name : -Address : -Country : Singapore -Postal Code : 333333 -Contact Num : 87654321 -Account Type : Personal Subprogram -Registration	Registration fail	Registration fail, system informs user of missing required fields and highlights missing required information
chkReg4	Precondition: User not logged in Input -Username : bill@gates.com -Password : t3jtlo -First name : Bill -Last name : Gates -Address : Silicon Valley -Country : Singapore -Postal Code : code -Contact Num : 87654321 -Account Type : Personal Subprogram -Registration	Registration fail	Registration fail, system informs user of invalid postal code and prompts for a new postal code

chkReg5	Precondition: User not logged in Input -Username : bill@gates.com -Password : t3jtwe -First name : Bill -Last name : Gates -Address : Silicon Valley -Country : Singapore -Postal Code : 333333 -Contact Num : 8472637one -Account Type : Personal Subprogram -Registration	Registration fail	Registration fail, system informs user of invalid contact number and prompts for a new contact number
chkReg6	Precondition: User not logged in Input -Username : bill@gates.com -Password : t3jtmf -First name : Bill -Last name : Gates -Address : Silicon Valley -Country : Singapore -Postal Code : 333333 -Contact Num : 84726371 -Account Type : Personal Subprogram -Registration	Registration successful System forwards user to home page	Registration successful, system stores user account info and forward user to home page to login

Test ID	Description	Expected Results	Actual Results
chkAddEvent1	Precondition: User logged in Input -name : zoukout 2009 -date : 2009-11-16 -desc : A night of fun -budget: 8000 -public: yes Subprogram -Create Event	Create event fail	Create event fail, system informs user of missing required fields highlights missing required information
chkAddEvent2	Precondition: User logged in Input -name : zoukout 2009 -venue : Sentosa -date : 2008-11-16 -time : 2200 -desc : A night of fun -budget: 8000 -public: yes Subprogram -Create Event	Create event fail	Create event fail, system informs user of invalid date, time and prompts for a new date, time later than the current.
chkAddEvent3	Precondition: User logged in Input -name : zoukout 2009 -venue : Sentosa -date : 2009-11-16 -time : 2000 -desc : A night of fun -budget: around 8000 -public: yes Subprogram -Create Event	Create event fail	Create event fail, system informs user of invalid budget and prompts for a new budget
chkAddEvent4	Precondition: User logged in Input -name : zoukout 2009 -venue : Sentosa -date : 2009-11-16 -time : 2000 -desc : A night of fun -budget: -8000 -public: yes Subprogram -Create Event	Create event fail	Create event fail, system informs user of invalid budget and prompts for a new budget
chkAddEvent5	Precondition: User logged in Input -name : zoukout 2009 -venue : Sentosa -date : 2009-11-16 -time : 2000 -desc : A night of fun -budget: 8000 -public: yes Subprogram -Create Event	Creation of event successful System forwards user back to add event page	Creation of event successful System stores the event information and forwards user back to add event page

chkAddEvent6	Precondition: User logged in Input -name : zoukout 2009 -venue : Sentosa -date : 2009-11-16 -time : 2000 -desc : -budget: 8000 -public: yes Subprogram -Create Event	Creation of event successful System forwards user back to add event page	Addition of event successful System stores the event information and forwards user back to add event page

Test ID	Description	Expected Results	Actual Results
chkAddContact1	Precondition: User logged in Input -firstname : peter -lastname : -postal code: 567890 -remark: tester -email: Subprogram -Add Contact	Add contact fail	Add contact fail, system informs user of missing required fields highlights missing required information
chkAddContact2	Precondition: User logged in Input -firstname : peter -lastname : lim -date of birth : 2010-11-16 -postal code: 356864 -remark: -email: petrel@test.com Subprogram -Add Contact	Add contact fail	Add contact fail, system informs user of invalid date of birth and prompts for a new date of birth.
chkAddContact3	Precondition: User logged in Input -firstname : peter -lastname : tan -date of birth : 1980-11-16 -address : Bedok -country : Singapore -postal code: 800 -remark: nil -email: petert@test.com Subprogram -Add Contact	Add contact fail	Add contact fail, system informs user of invalid postal code and prompts for a new postal code.
chkAddContact4	Precondition: User logged in Input -firstname : Peter -lastname : Soh -date of birth : 1988-11-16 -address : Bedok -country : Singapore -postal code: 800800 -email: petert@test.com Subprogram -Add Contact	Contact Successfully added	Contact successfully added, system forwards user to contact listing.

WHITEBOX TESTING USING JUNIT

DBACCESSORTEST.JAVA

```
package dataaccessor;

import beans.*;
import org.junit.Test;
import junit.framework.*;
import java.sql.*;

public class DBAccessorTest extends TestCase {

    private static Connection conn;

    @Override
    protected void setUp() throws Exception {
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://merv.kicks-ass.org:3307/er7project";
        conn = DriverManager.getConnection(url, "root", "password");
    }

    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    @Test
    public void testEmail1() throws Exception {
        System.out.println("TESTING testEmail1");
        String contact_id = "s@s.com";
        int event_id = 4;

        DBAccessor instance = new DBAccessor(conn);

        String result = instance.sendEmailToContact(contact_id, event_id);
        System.out.println("result " + result);

        String expResult="";
        String query = "SELECT hexvalue FROM event_contact WHERE event_id = ? AND contact_id =
?";
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setInt(1, event_id);
        statement.setString(2, contact_id);
        ResultSet rs = statement.executeQuery();
        if (rs.next()) {
            expResult = rs.getString("hexvalue");
        }
        rs.close();
        assertEquals(expResult,result);
        System.out.println("END OF TESTING testEmail1");
    }

    @Test
    public void testEmail2() throws Exception {
        System.out.println("TESTING testEmail2");
        String contact_id = "s@s.com";
        int event_id = -1;
```

```
        DBAccessor instance = new DBAccessor(conn);

        String result = instance.sendEmailToContact(contact_id, event_id);
        System.out.println("result " + result);

        String expResult=null;
        String query = "SELECT hexvalue FROM event_contact WHERE event_id = ? AND contact_id =
?";
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setInt(1, event_id);
        statement.setString(2, contact_id);
        ResultSet rs = statement.executeQuery();
        if (rs.next()) {
            expResult = rs.getString("hexvalue");
        }
        rs.close();
        assertEquals(expResult,result);
        System.out.println("END OF TESTING testEmail2");
    }

    @Test
    public void testEmail3() throws Exception {
        System.out.println("TESTING testEmail3");
        String contact_id = "";
        int event_id = 6;

        DBAccessor instance = new DBAccessor(conn);

        String result = instance.sendEmailToContact(contact_id, event_id);
        System.out.println("result " + result);

        String expResult=null;
        String query = "SELECT hexvalue FROM event_contact WHERE event_id = ? AND contact_id =
?";
        PreparedStatement statement = conn.prepareStatement(query);
        statement.setInt(1, event_id);
        statement.setString(2, contact_id);
        ResultSet rs = statement.executeQuery();
        if (rs.next()) {
            expResult = rs.getString("hexvalue");
        }
        rs.close();
        assertEquals(expResult,result);
        System.out.println("END OF TESTING testEmail3");
    }

    @Test
    public void testCheckLogin() throws Exception {
        System.out.println("TESTING checkLogin");
        String username = "s@s.com";
        String password = "23456";
        String name = "Edwin";
        DBAccessor instance = new DBAccessor(conn);
        MemberBean expResult = new MemberBean();
        expResult.setEmail_add(username);
        expResult.setPassword(password);
        expResult.setFirstName(name);
        MemberBean result = instance.checkLogin(username, password);
        System.out.println(result.getEmail_Add());
        System.out.println(result.getFirstName());
```

```
    assertEquals(expResult, result);
    System.out.println("END OF TESTING checkLogin");
}

@Test
public void testCheckLoginFalse() throws Exception {
    System.out.println("TESTING checkLoginFalse");
    String username = "false@false.com";
    String password = "false";
    String name = "noname";
    DBAccessor instance = new DBAccessor(conn);
    MemberBean expResult = new MemberBean();
    expResult=null;
    MemberBean result = instance.checkLogin(username, password);

    assertEquals(expResult, result);
    System.out.println("END OF TESTING checkLoginFalse");
}

@Test
public void testMemberExist() throws Exception {
    System.out.println("TESTING memberExist");
    String username = "s@s.com";
    String name = "Edwin";
    DBAccessor instance = new DBAccessor(conn);
    MemberBean expResult = new MemberBean();
    expResult.setEmail_add(username);
    expResult.setFirstName(name);
    MemberBean result = instance.memberExist(username);
    assertEquals(expResult, result);
    System.out.println("END OF TESTING memberExist");
}

@Test
public void testMemberExistFalse() throws Exception {
    System.out.println("TESTING memberExistFalse");
    String username = "false@false.com";
    String name = "noname";
    DBAccessor instance = new DBAccessor(conn);
    MemberBean expResult = new MemberBean();
    expResult=null;
    MemberBean result = instance.memberExist(username);

    assertEquals(expResult, result);
    System.out.println("END OF TESTING memberExistFalse");
}

@Test
public void testRetrieveEventDetails() throws Exception {
    System.out.println("TESTING retrieveEventDetails");
    int eventid = 68;
    DBAccessor instance = new DBAccessor(conn);
    String expResult = "CSC 207 Web Application Review";
    EventBean result = instance.retrieveEventDetails(eventid);
    String res=null;
    if(result!=null)
        res = result.getName();
    assertEquals(expResult, res);
    System.out.println("END OF TESTING retrieveEventDetails");
```

```
    }

    @Test
    public void testRetrieveEventDetailsFalse() throws Exception {
        System.out.println("TESTING retrieveEventDetailsFalse");
        int eventid = -1;
        DBAccessor instance = new DBAccessor(conn);
        String expResult = null;
        EventBean result = instance.retrieveEventDetails(eventid);
        String res=null;
        if(result!=null)
            res = result.getName();

        assertEquals(expResult, res);
        System.out.println("END OF TESTING retrieveEventDetailsFalse");
    }
}
```

DISCUSSION

DERIVATION OF MODELS

ANALYTICAL MODEL

The initial technique of deriving our analytical model is by identifying the nouns in the Software Requirements Specifications, which each will then be evaluated if they hold substantial functionalities to hold as a class, or exist as an attribute in a class.

After we have a list of nouns identified as our classes and attributes, we include considerations on generalising/specialising classes by inheritance, which allows classes to be specialise in specific responsibilities increasing cohesiveness of the classes. They are then evaluated for what functionalities they provide and methods are added. Next, we introduced controller classes which hold the methods for the system to interact with the entities.

Subsequently, we derived the relationship and dependencies between classes, and introduced interfaces and relationship classes where possible to reduce the coupling of the classes in the system. Functions are then decomposed to have classes in charge of specific tasks, controller classes that held methods to interact with entities are delegated to servlet classes with its dependencies on models provided by the DBAccessor that will instantiate the Java Beans(Model) and encompass all the SQL related tasks.

DESIGN MODEL

As for the design models, we first start off by mapping each use case with a design model, and identify the actor which invokes and interacts with it. Next, steps in the basic flow of the use case are translated into messages in the design model, and alternate flows will be modeled as an alternate combined fragment.

We have modeled our design models with reference to the MVC architecture, where the actor sends messages to the Servlets as the Controller and receives information displayed via the JSP files acting as the View layer, while in the backend, the controller executes business logic and in turn instantiates model classes where necessary.

PERFORMING USER-INTERFACE DESIGN

Good UI design facilitates in managing the events and contacts at hand without drawing unnecessary attention to the users. MyEventPlanner focuses on intuitive user interface for easier and less expensive to use. Thus, it minimizes training and support costs. Therefore, it increases the user satisfaction.

In designing our UI design, we take into consideration users of age group from age 12 – 60, of these users, some may be non-computer literate but have a common knowledge of using web applications such as web-based e-mail. Intuitively, UI instructions on the website must not be length. Some advance users may skip over these instructions as they assume they know how to use things and does not have the time to read complicated instructions. Rather, short instructions are introduced to various sections of the website. Menu tabs that are selected or mouse-over appears different from the others so as the keep users in context, and navigation menu bar provides a quick way for users to access important functionalities which minimize articulatory distance. The calendar implementation in the system also minimizes semantic distance when users are browsing for events as opposed to having plain texts.

We also assume these users are very busy working people and do not have the time to read the help manual before using this application.

WEB TECHNOLOGY USED

An increasing numbers of websites are developing new types of user interface design, taking advantage of users' increasing levels of Internet-sophistication and faster connections. These new interfaces often allow users to view and manipulate large quantities of data.

Our design utilizes the jQuery library and Prototype JavaScript Framework to provide the document traversing, event handling, animating, and Ajax interactions. Thus, it gives the user the software application experience with minimal loading or refreshing of pages.

WEBSITE LAYOUT

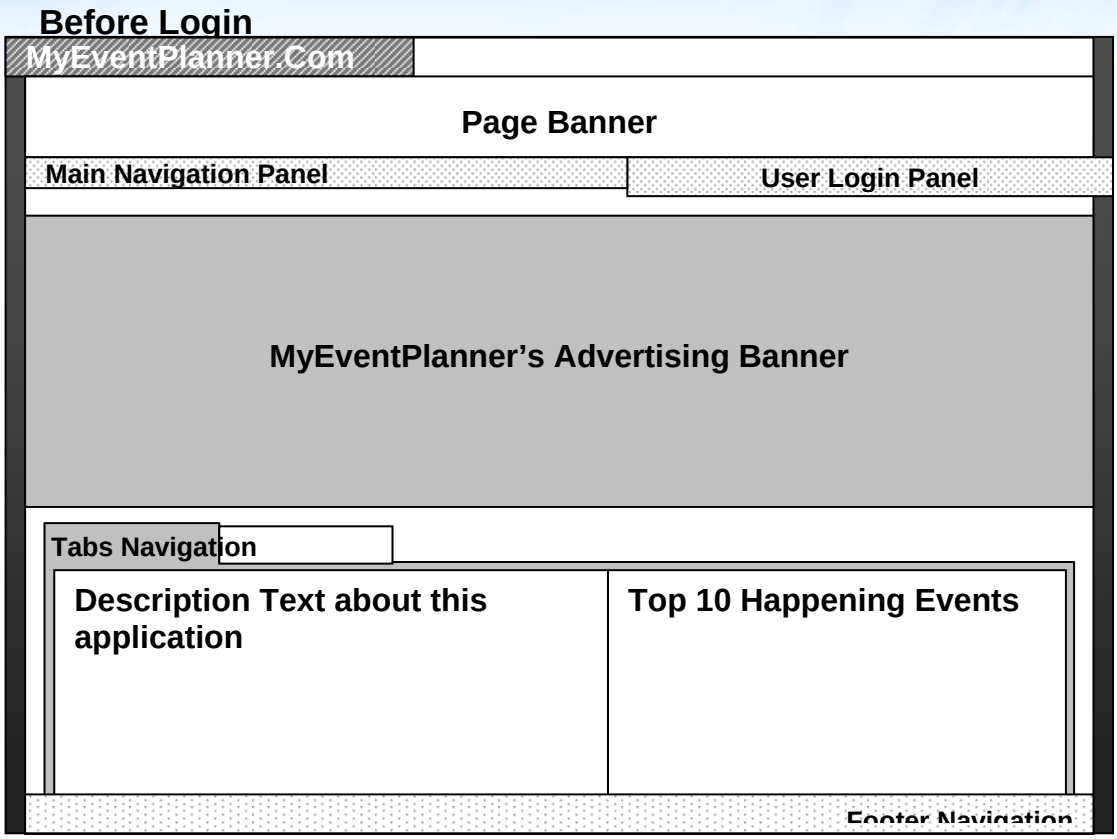


Figure 4.2.1. Homepage Layout before Login

Users entering our website will see the above homepage layout of the website. As visitors will be visiting our site, the homepage will serve the main publicity for the promotion of MyEventPlanner services. There will be a horizontal advertising banner to showcase our products and services and display our promotions if available. Besides it, user can view our public events from the Top 10 Happening Events and Public Calendar view at the tab navigation. If the user has an account, he/she can login to the account at any point of time. The page banner greets the visitors.

Page Banner

The page banner greets the visitors a warm welcome.

Navigation Panels

There are two types of main navigation, visitor navigation menu (without login) and registered user navigation menu. From the above illustration, user is given the visitor navigation menu. The navigation will produce an animation when mouse cursor is hovered over, providing a visual feedback to the user. When the navigation is clicked, the page will be loaded with the tab highlighted in green. It tells the user the page he/she is currently visiting now.

Some of our webpage will display the contents in a tab container fashion. A typical example is the homepage where user can display the Top 10 public events or public calendar.

After Login

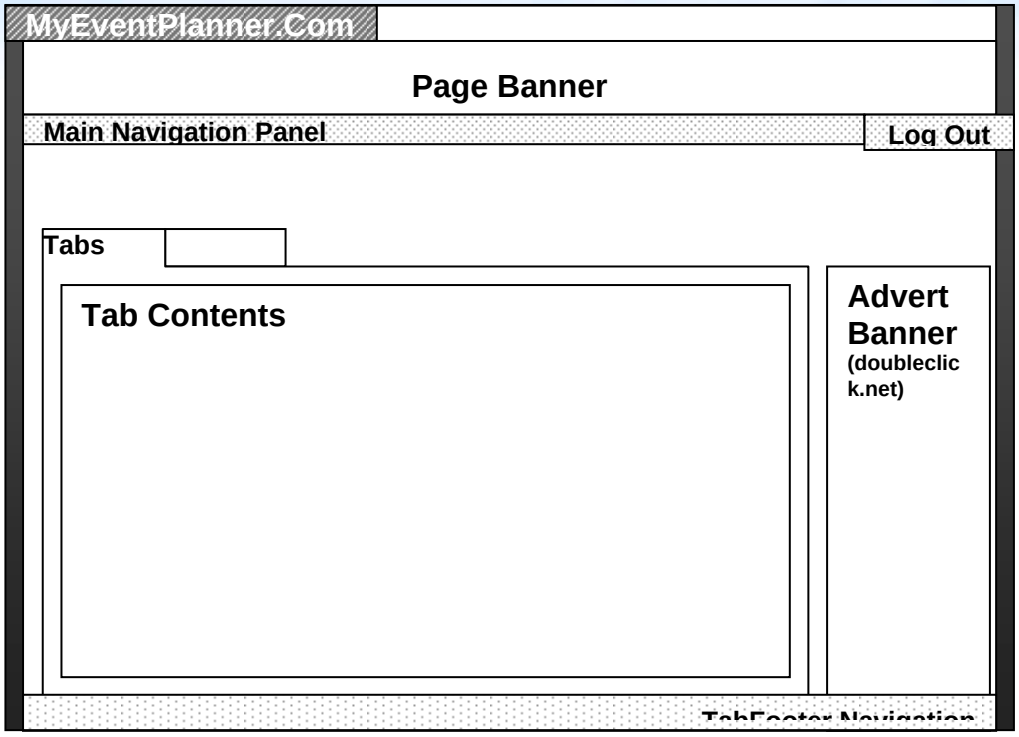


Figure 4.2.2. Homepage Layout after Login

After user login into his/her account, they will see the above homepage layout of the website in figure 4.2.2. Similarly, homepage shows the public events from the Top 10 Happening Events and Public Calendar view at the tab navigation. The user can log out the account at any point of time.

From the comparison of Figure 4.2.1 and 4.2.2, we had maintained the website design consistency regardless the user has login or not. Consistency makes the website easier to use, because users will not have to learn new tricks as they move about.

This website is best viewed with a screen resolution of 1024 by 576 (netbooks) and 1024 by 768 onwards.

Page Banner

Registered user will be greeted and it will display their first name and account type.

Navigation Panels

The navigation changes to the menu of choices in their respective account. Likewise, the navigation will produce an animation when mouse cursor is hovered over, providing a visual feedback to the user. When the navigation is clicked, the page will be loaded with the tab highlighted in green. It tells the user the page he/she is currently visiting now.

Some of our webpage will display the contents in a tab container fashion. A typical example is the homepage where user can display the Top 10 public events or public calendar.

NAVIGATION HIERARCHY

This website has two types of navigation hierarchy as mentioned previously. Figure 4.3.1 shows the navigation flow from the homepage when the visitor/user has not login their user account (if exist).

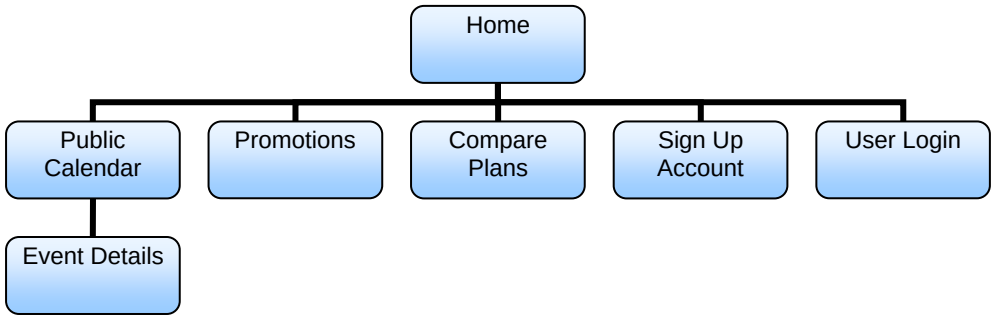


Figure 4.3.1 Navigation Flow before User Login

After the user have login into their user account, the navigation changes according to their privilege assigned to their account. Figure 4.3.2 shows the account with leader privileges.

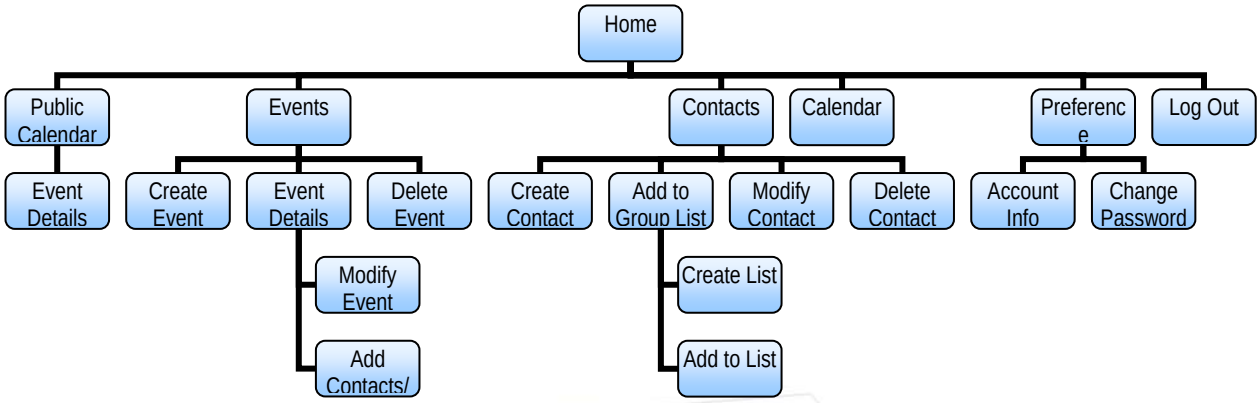


Figure 4.3.2 Navigation Flow After User Login

COLOUR SCHEME

Figure 4.4.1 Homepage screenshot

Colours serve as a means of communication. The choices of colours can create an impact and affects the mood of our visitors and users. Also, the creative use of colours can make the website feels professional.

MyEventPlanner needs a pleasant feel, and easy reading. We have chosen the following colours to create the impact and mood for the website:

Colour	Usage	Objective
White	Main website content	White works best for easy reading
Green	Headings, Left title bar, User Login panel	It has a soothing quality to our eyes as its associated with nature and symbolizes growth, hope, success, and money
Sky Blue	Website banner, Navigation Highlight, hyperlinks	Sky Blue has a calming effect. It's associated with trust, dependability, faith, and healing.
Black	Website background, Top 10 events, exposed dialog boxes	Black is the absence of all colour and makes surrounding colours standout.

HELP MESSAGES

With the use of latest web technologies, we have introduced the exposed dialog to replace the old fashion pop up alert or question boxes. It blurs out the website to draw the attention to the exposed black dialog box. As this is a unique feature from most websites, user will be astonished to experience it.

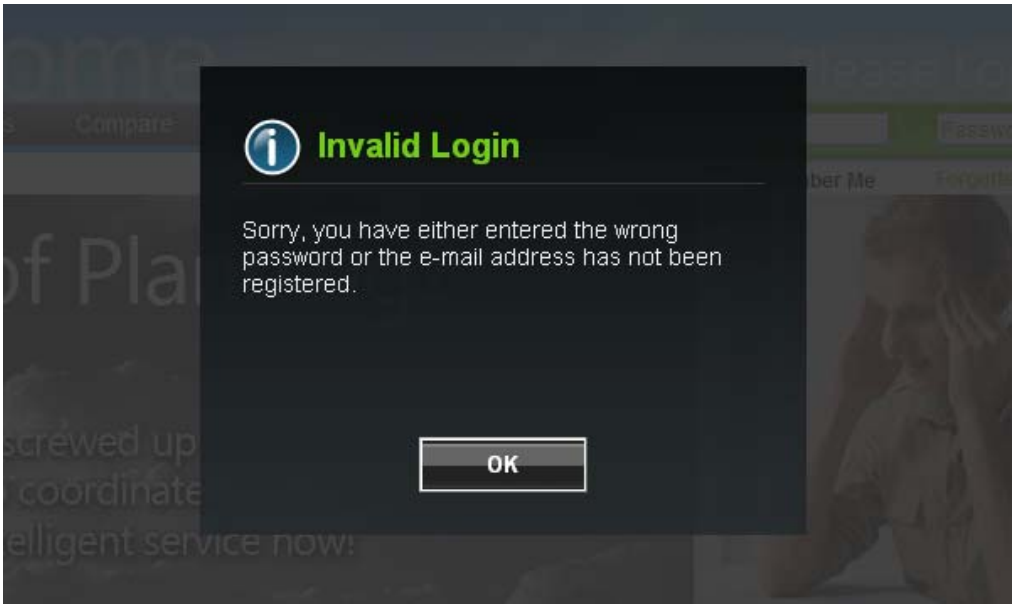


Figure 4.5.1 Exposed Dialog Box

Every submission form in the website will be validated using the latest AJAX technology. It checks for invalid user input such as non-numeric, minimum and maximum length, e-mail. As such, the web application will be protected from invalid entries from the user-end.

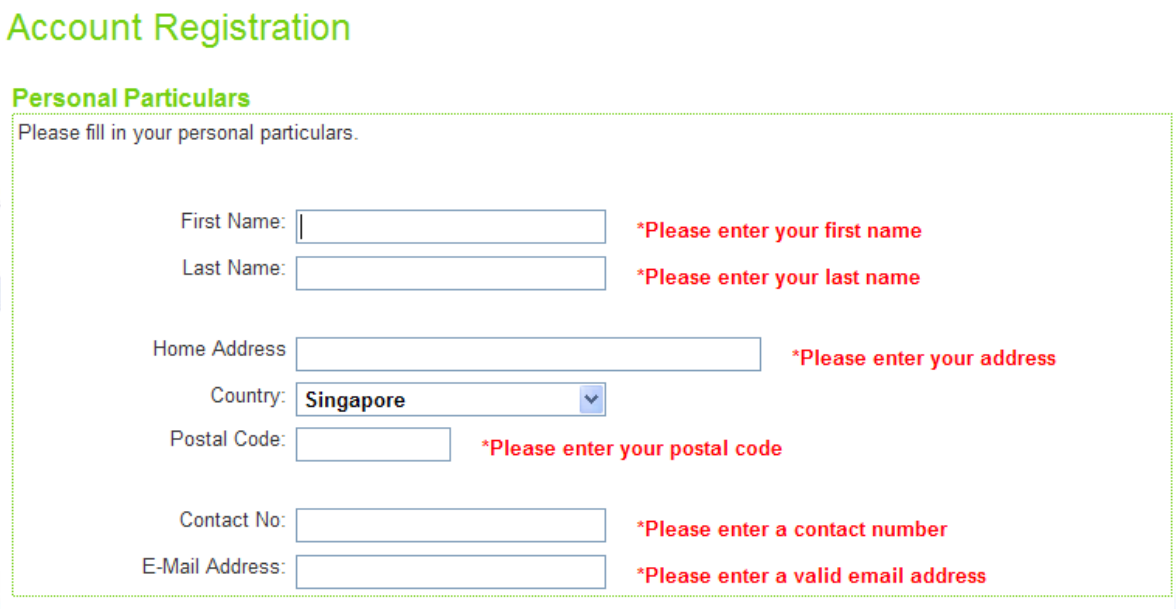


Figure 4.5.2 AJAX Form Validation

INPUT COMMANDS

Mouse Movement

Sometimes user may be using suboptimal pointing devices, like touchpads, trackballs therefore the UI will need to minimize the movement of the mouse cursor to navigate the website. The above layout of the website optimizes these movements for the user.

Keyboard Commands

There are a few common keyboard commands that user can utilize when they encounter an exposed message or choice dialog and forms.

Keyboard Command	Action
ESC	Close Exposed Dialog Box
Tab	Move selected item (textbox) to the next
Enter	Submits the form
Backspace or Alt+Left	Navigates to the previous page/tab
Alt+Right	Navigates to the next recently accessed page/tab

Cross Browser Platform

Today, there is a great number of web browsers one can use to surf the internet. These web browsers are developed by various companies and have different standards. With different standards, a website may appear very differently compared with another brand of web browser. Therefore, we can conclude that this website has a compatibility issues with other types of web browsers. This occurs as various web browsers have different methods of interpreting the HTML, CSS and JavaScript. To cope with cross browser compatibility, our website is always tested on popular web browsers such as Mozilla Firefox 3.0, 3.5, Microsoft Internet Explorer 7.0, 8.0 and Google Chrome.

Figure 4.7.1 Comparison between Internet Explorer 8.0 and Google Chrome 4.0 Beta

DIFFICULTIES ENCOUNTERED AND SOLUTIONS APPLIED

Problem: Invalid update of files in SVN

Part of our implementation is to put all our SQL queries into a centralized java file. So whenever we need to fetch any records from the database, we just need to access DBAccessor.java and retrieved the necessary java bean by calling the functions in that java file. However, all the team members are also accessing the file. In that case, conflict might arise when both team members trying to edit their own calling function in DBAccessor.java, which result in file being overwritten by others.

Solution:

Whenever we make any changes on the shared file in SVN and committed the changes for that file, we will inform the other team members to an update on the file so that changes will be reflected and eliminate the above problem mentioned. Should there be two people trying to make changes to the file at the same time, they will negotiate and wait for the other to finish his work and continue to work on his after updating the file in SVN.

Problem: Synchronization of database records

Our group hosts our database at a centralized location and all member access the database remotely from each of our workstation. It's because by hosting our database at one centralized location, changes make to will be reflected to all since everyone will be accessing the same database. However, it also incur problem whenever one make changes to one of the database table, like adding, deleting columns or records. After the changes made to the database, other members might not realize the changes made until we encounter errors with our SQL statements when we tried to run the project using NetBeans.

Solution:

With proper communication between each team members, such a problem can be resolved. For example, whenever one makes any changes to the database, he will inform the other members of the changes and so as to eliminate the time we might spend on debugging the error encountered.

Problem: Creeping Requirement

In all software development the issue of creeping requirements is a problem, implementations in terms of codes, pages and database schemas often require changes to deal with the creeping requirements. This is inevitable because usually users do not really realize what they want or need until they actually see it.

Solution:

Add in the necessary attributes into the database and inform everyone in the team about the changes. This is to ensure that our team members who are using the particular in the table in the database know about the changes made.

Problem: Cross Browser Incompatibility

Different browsers might display a particular page differently. A particular page might be displayed in this particular way in Internet Explorer but the page might be displayed differently in Firefox or any other web browser. The differences are size of pop up etc. Basically the problem is with the differences in appearances of the webpage being displayed in different web browsers.

Solution:

Use an incremental approach. Open various web browsers to see if there is any compatibility problem when we are developing a particular web page. If there is any compatibility problem, we will solve it by changing the code.

Problem: Conflict of JQuery with other libraries

JQuery and Prototype share the similar syntax of calling their functions therefore it causes a clash between both of the libraries.

Solution:

By default, jQuery uses "$" as a shortcut for "jQuery", which is similar to that of Prototype. However, the default can be override by calling jQuery.noConflict() at any point after jQuery and Prototype have both loaded.

```
<html>
<head>
  <script src="prototype.js"></script>
  <script src="jquery.js"></script>
  <script>
    jQuery.noConflict();

    // Use jQuery via jQuery(...)
    jQuery(document).ready(function(){
      jQuery("div").hide();
```

Problem: Similar copies of work

Duplication of work, due to testing, causes confusion for other team members.

Solution:

By constantly updating fellow team members of intent and changes, the confusion between team members is greatly reduced to the minimum and saves the team much time as a result.

Problem: Limited or no prior knowledge

Some team members have limited or no prior knowledge of using java for web applications.

Solution:

Team members practiced self learning via eBooks and internet to get a basic idea of java web application or to refresh their memory. After most of the team members have a better understanding, team members with better knowledge would help to introduce the rest to more advanced programming. This has helped all the team members to have the sufficient knowledge in developing the web application.