RUTO SNS prototype Design and Implementation in EJB 3-tier Framework

CONTENTS

Basic Concept 2

Work Breakdown 3

High Level Architecture 8

- The Business Process Flow 8
- EJB 3-tiered architecture 8
- Client Tier 9
- Server Tier 9
- Database Tier 10

Component Level Design 10

- Database Design 10
- Login Design 11
- User Profile Design 13
- Activity 14
- Recommendation 15
- EJB Timer Service 16
- Data Flow of User Scenario 18

Component Model Services 19

Response to Demo Concerns 20

RUTO SNS prototype Design and Implementation in EJB 3-tier Framework

Prepared by:

Rongxin Du (rd2537@columbia.edu) Qiang Deng(qd2114@columbia.edu)

Chao Song (cs2994@columbia.edu) Jen-Po Cheng (jc3781@columbia.edu)

Basic Concept

RUTO will be implemented as a web application in component model framework, which at least includes the front-end client application, the mid-tier server, and the back-end database. In details, the front-end web pages serve to obtain useful information from our users and interact with users. The mid-tier server contains the core business logic of the system and communicates with both front-end client application and the back-end database. User information, the search and query results computed by server, will be stored in our back-end database.

There are two kinds of users in our system: administrators and common users. Administrators are powerful users because they can do privileged functions in the RUTO system. For example, they can view all users' data as well as manage the server and the database. Furthermore, along with creating comments and activities, administrators can delete inappropriate comments and activities. Even they may delete a user account if such user has abused the functionalities of RUTO. Potential administrators need to contact the company representatives in order to get an administrator account. Common users can enjoy functionalities of our system as the follows.

1. Create an Account and Log in

First-time users can click the "sign up!" link on our home page, which directs users to a sign up page. In order to sign in RUTO, a user must create an unique identity verified by our system.

2. Personal Profile

Every user has a private personal profile. The profile displays users' basic information similar to what Facebook does. After completing filling their educational backgrounds, hobbies, locations and some other useful information, users may find like-mined strangers and interesting mingles recommended by RUTO. On the profile, a user can check his/her own message box to see text messages from/to friends.

3. Post and Join real-world social activities

A user can post and join real-world social activities on his/her own activity profile page. Once

wanting to post an activity, a user need to provide a theme of the activity such as starting/finishing time and a location as well. Furthermore, a user can help others to understand the topic of the activity by means of marking related labels of RUTO. It is very intuitive to use a label-marking function established by RUTO. If you intend to join an activity, simply click on the "join" link corresponding to the specific social activity.

4. Recommend social activities and potential friends

As soon as a user has provided the user information fulfilling RUTO's minimal criteria, ROTO then automatically recommends the user activities and RUTO-beings that the user is likely interested in according to the user information with respect to RUTO's back-end database. For example, the similarities in hobbies, activities participated, city of birth, favorite movies, etc., among all users. By suggesting selective social activities, RUTO links users who shares common hobbies, backgrounds, even values and characteristics.

5. Logging out

A user remains in logged-in status while browsing web pages of RUTO. To log out, simply the user may click the "log out" button on the top of the personal profile.

Work Breakdown

	Requirement	Priority	Details
1	RUTO-being Accounts		
1.1	Account Creation		
1.1.1	Sign-up Page	high	1. Home Page provides a link named "Sign up!" for users to sign-up. 2. A user is redirected to a sign-up page if he/she clicks the sign-up link. 3. The data provided by user is stored in database.
1.1.2	Sign-up Form	high	1. In order to register, a user must fill in username, an e-mail address and the password; 2. If a user gives invalid information (where lacking of username/E-mail address, or whose password is unequal to whose re-enter password), RUTO will give a warning and redirect user to refill the registering form

			again.
1.2	Log in	high	1. A user may log in through the home page by using an authorized e-mail address with a corresponding password. 2. RUTO checks whether the user information have already registered before. If so, the user will be redirected to the personal profile page, otherwise, the user will be required to input again.
1.3	Retrieve Password	low **To be finished**	1. After clicking on "Forget password?" link, a user will be required to fill in the email address which is used for registration, and then an e-mail containing a temporary password will be sent to the user's e-mail address by RUTO.
2	RUTO-admin Accounts		
2.1	Log in	high	1. Administrator could log in through our home page by inputting email address as admin@RUTO.com, password as admin. This is the only valid administrator assigned in our application. 2. The system will check whether the current user is administrator by matching email address and password.
2.2	Admin Page		
2.2.1	Manage User Information	high	1. The administrator can view a list of all the registered users. 2. The administrator can remove any user from the database.
2.2.2	Manage Activity Information	high	1. The administrator can view a list of all the posted activity. 2. The administrator can remove any activity from the database.
3	User Profile		
3.1	Display User Profile	high	1. Display the user's basic information, including user name, email address, full name, gender, relationship status and educational background. ~~*2. Display the user's personal information, including*~~

			~~*their hobbies, educational background, etc.*~~
3.2	Edit User Profile	high	1. Edit the basic information, including name, gender, birthday, relationship status, state, city, detailed address and zip-code. 2. Edit hobbies and educational background.
3.3	Message Board	high	1. User can click "MessageBox" button on their personal profile to receive messages from other RUTO-beings. 2. User A can search a specific user B by email and leave messages on user B's profile. 3. Users can reply messages in their message box.
3.4	Post Activity	high	1. There is a button called "Post Activity" under the function section of user profile, clicking on which will direct you to an activity creation page. 2. On activity creation page, a user can create an activity by filling in the topic, description, location, detailed address, start date, end date, start time and end time.
3.5	Account Configuration	low **To be finished**	1. User can change the account's user name, email address, and password. 2. User can change the privacy level, meaning user can change whether his/her profile or message board are visible to other RUTO-beings.3. User can change the automatic recommendation preference, both for RUTO-beings and for activities.
3.6	Display Recommendation Result	high	1. click "Recommend Friends" button in the function section of user profile to display RUTO-beings who are automatically recommended by our system. 2. click "Recommend Activity" button in the function section of user profile to display activities which are automatically recommended by our system.
3.7	Search	medium	1. User can search people by email. 2. User can search activity by location (state, city, zipcode) or label (activity label tagged by creator).
3.8	Display Posted and Joined Activities	medium	1. User can view the activities he/she has posted by clicking on the button "View Activities(Posted)" on the user profile page. 2. User can view the activities he/she has joined by

			clicking on the button "View Activities(Joined)" on the user profile page.
3.9	Log out	high	1. User can log out safely by clicking on the "log out!" link on the top of personal profile.
4	Activity		
4.1	Display Activity Information	high	1. Display information of an activity on the activity profile page, such as its theme, organizer, description, time, location, tag and its current status(not started or undergoing or finished).
4.2	Display Members	high	1. click on "ShowMember" button on the activity profile page to display all the RUTO-beings who have joined in this activity; 2. By clicking on the "Visit this member" button, you can browse that user's profile.
4.3	Join the activity	high	1. User can join this activity by click on "Join" button on the activity profile page; 2. If you've already joined before, our application will give a warning as "Already joined". If you joined successfully, our application will notice you that "Joined Successfully!".
4.4	Edit Activity Information	high **To be finished**	1. The user who posted this activity can edit the information of the activity, for example, update the location of an activity.
5	Friends and Activities Recommendation		
5.1	Friends Recommendation	high	1. By clicking on the "Recommend Friends" button in the function section on the user profile page, our application can automatically recommend some other registered users you might feel interested in, ranked in decreasing order with respect to similarity rate (percentage).
5.2	Activity Recommendation	high	2. By clicking on the "Recommend Activity" button in the function section on the user profile page, our application can automatically recommend some activities you might feel interested in, ranked in decreasing order with respect to similarity rate

			(percentage).
6	Additional Feature		
6.1	JAAS		1. We use EJB security services(JAAS) to check the login of administrator.
6.2	EJB Timer		1. We use EJB Timer to keep track of the status of an activity. Before the start date and time, the activity is labeled as "not started". After time passes starting time, the activity is labeled as "undergoing". And finally, when the finishing time of activity has passed, the activity is labeled as "finished".
6.3	Asynchronously Notification		1. We use AJAX to asynchronously notify users if they have unread messages. While users stay on their personal page, a pop up window will show up if there is a new message coming. Or if users are off-line when they receive messages, they will be notified next time they log in.
6.4	CSS	high	1. We apply the CSS technique to help develop a more feasible user interface for RUTO. 2. After using CSS, we shall easily provide a consistence format and style for all pages of RUTO. In addition, it make modifications in html more convenient.

High Level Architecture

The Business Process Flow

Figure 1 below shows the business process flow and logic of the RUTO system, in which we divide applications into four parts (Home Page, User Profile Page, Activity Profile Page, back-end Derby database). All the black arrows show the data flow directions. From Home Page, a user can create new account, log in to be a regular user or an administrator. On the user profile page, a user can fill up his/her personal information, check messages, search for both people and activities. On the activity profile page, RUTO displays basic information of an activity as well as its members. A user can join an activity or visit the members belonged to such an activity. All the three parts illustrated above need to access RUTO's back-end derby database in terms of retrieving data, to display or saving user data.

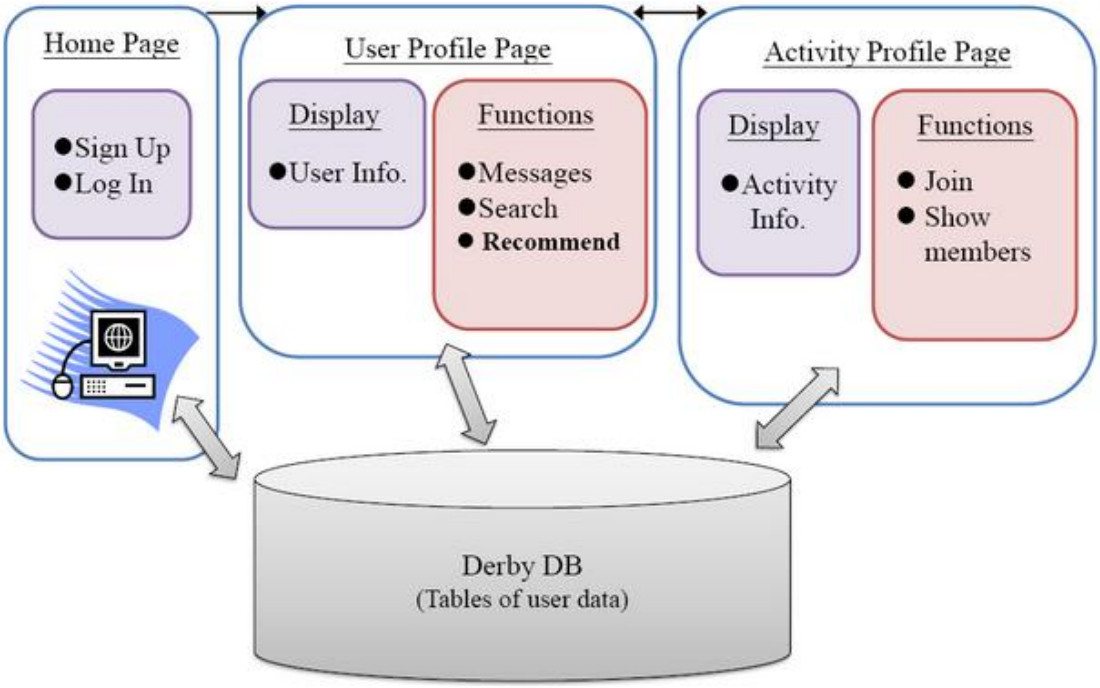


Figure 1 The business process flow of RUTO

EJB 3-tiered architecture

Figure 2 shows in details what component framework we choose and how we use it. We applies EJB 3-tiered component framework including the front-end tier (client, the web-browser, Controller and Views), mid-tier (the glassfish application server, Enterprise JavaBeans) and the

back-end database (derby database).

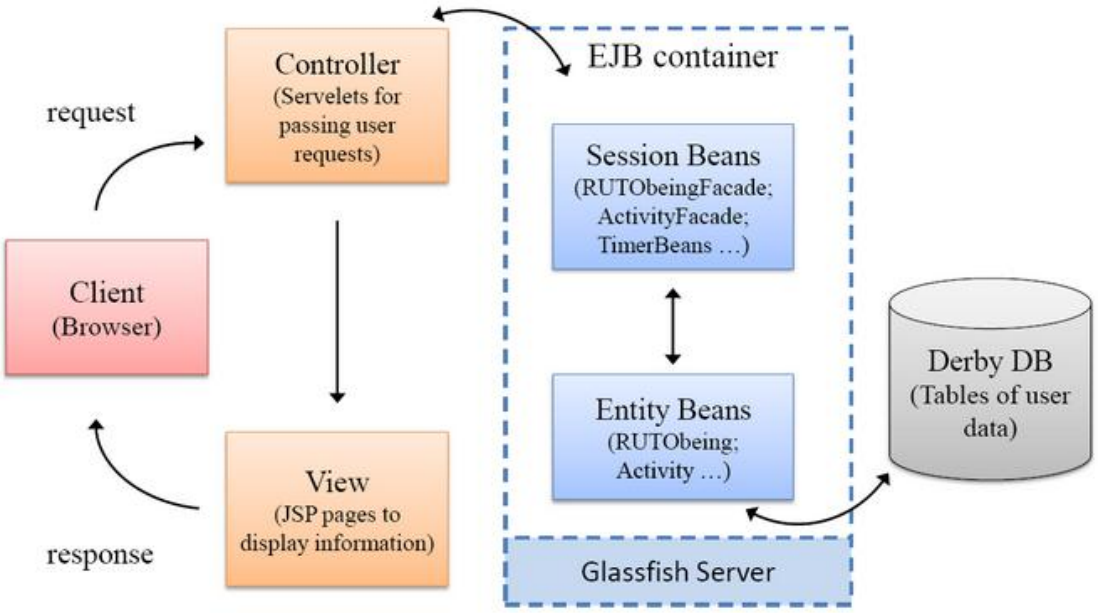


Figure 2 EJB 3-tiered architecture

Client Tier

Our clients send request to our Controller(a series of Servlets that process and pass user data), for instance, we introduce a simple use scenario here, if user1@gmail.com wants to sign up in our RUTO system, he/she need to input email address, set password and click 'sign up' button. We refer to this kind of process as a user request. After that, a specific CreateAccount servlet will be in charge of parsing and saving user's input data and interact with the mid-tier Enterprise JavaBeans.

Server Tier

In the Enterprise JavaBeans container on glassfish server, we created entities, such as RUTObeing.java to describe a registered user, Activity.java to present a posted social activity. And further, corresponding session beans for these entities are generated automatically via Netbeans IDE, which are named as RUTObeingFacade and ActivityFacade, etc. Servlets can pass user data to session beans, and we write user's personal information into back-end databases via the persistence service provided by entities related to those session beans for entities. Further more to retrieve data from back-end database and present it back to client-end(client gets the response

from our system), we also depend on EJBs to retrieve data and display results on Views(a series of jsp files).

Database Tier

We use derby database in our first iteration. We have created several tables to save user data, including table RUTObeing for saving basic user information, table Location for saving location information, table Label for saving users' hobbies and also tags of activities, table Activity for storing basic activity information, table Message for saving user messages, table reservation for recording users' reservations for different activities.

Component Level Design

Database Design

We have designed the following 6 tables:

1) **RUTObeing**(+ID:+Long, birthday:String, education_background:String, email:String, first_name:String, last_name:String, gender:String, password:String, username:String, relationship_status:String);
2) **Activity**(+ID:+Long, description:String, organizer:String, theme:String, state:int, start_date:String, end_date:String, start_time:String, end_time:String);
3) **Location**(ID:Long, useremail:String, activityID:Long, state:String, city:String, address1:String, address2:String, zipcode:String);
4) **Label**(+id:+Long, useremail:String, activityID:Long,mainclass:String, subclass:String, other:String);
5) **Message**(+id:+Long, from_email:String, to_email:String, message:String, state:int);
6) **Reservation**(+id:+Long, useremail:String, activityID:Long);

We establish tables in database by using EJB @Entity, and we use ID number as primary key

for each table, the ID number can be generated automatically by EJB using @GeneratedValue (strategy = GenerationType.AUTO). For convenience, we do not apply any referential relationships between two tables(foreign keys). However, we use the following strategy to make tables interacting with each other.

- RUTObeing(email)=Activity(organizer)=Location(useremail)=Label(useremail)=Message(from_email)=Message(to_email)=Reservation(useremail)

In fact we use user email address as the attribute that can uniquely identify each user, and pass this email to attribute organize in table Activity, attribute useremail in table Location, and so on. For example, in table activity, if we want to know more about the organizer, we can search this organizer string(organizer email address) further in RUTObeing(email) in order to find out this user (a row in RUTObeing).

- Activity(ID)=Location(activityID)=Label(activityID)=Reservation(activityID)

We pass the ID attribute in Activity to attribute activityID in table Label, attribute activityID in table Reservation, and attribute activityID in table Reservation. For example, if we want to find out the location and tag about a specific activity whose ID number is 100, we can search column activityID in table Location and column activityID in table Label respectively, pick up the rows with activityID=100 and retrieve further information, such as zipcode in table Location and mainclass in table Label.

Besides the back-end derby database, our system can be divided into five sub-systems which are Login, UserProfile, Be_friend_of, Activity and Recommend.

Login Design

The first sub-system Login is shown in Figure 3-a and Figure 3-b below. As shown in Figure 3-a, RutoBeing is an entity bean used to store the information of RUTO users. RUTObeingFacade is the session bean associates with RutoBeing that connects the front-end and the database. The RUTObeingFacade extends AbstractFacade. CreatedAccount class (Servlet) is used to set up user accounts and utilize RUTObeingFacade to persist users' registration data into database (table RUTObeing). Login class (Servlet) mainly realize the function of user login, it search the

RUTObeing table in database via session bean RUTObeingFacade to see if the login email and password can match a specific row. CreateAccount and Login Servlets are both extended from HTTPServlet.

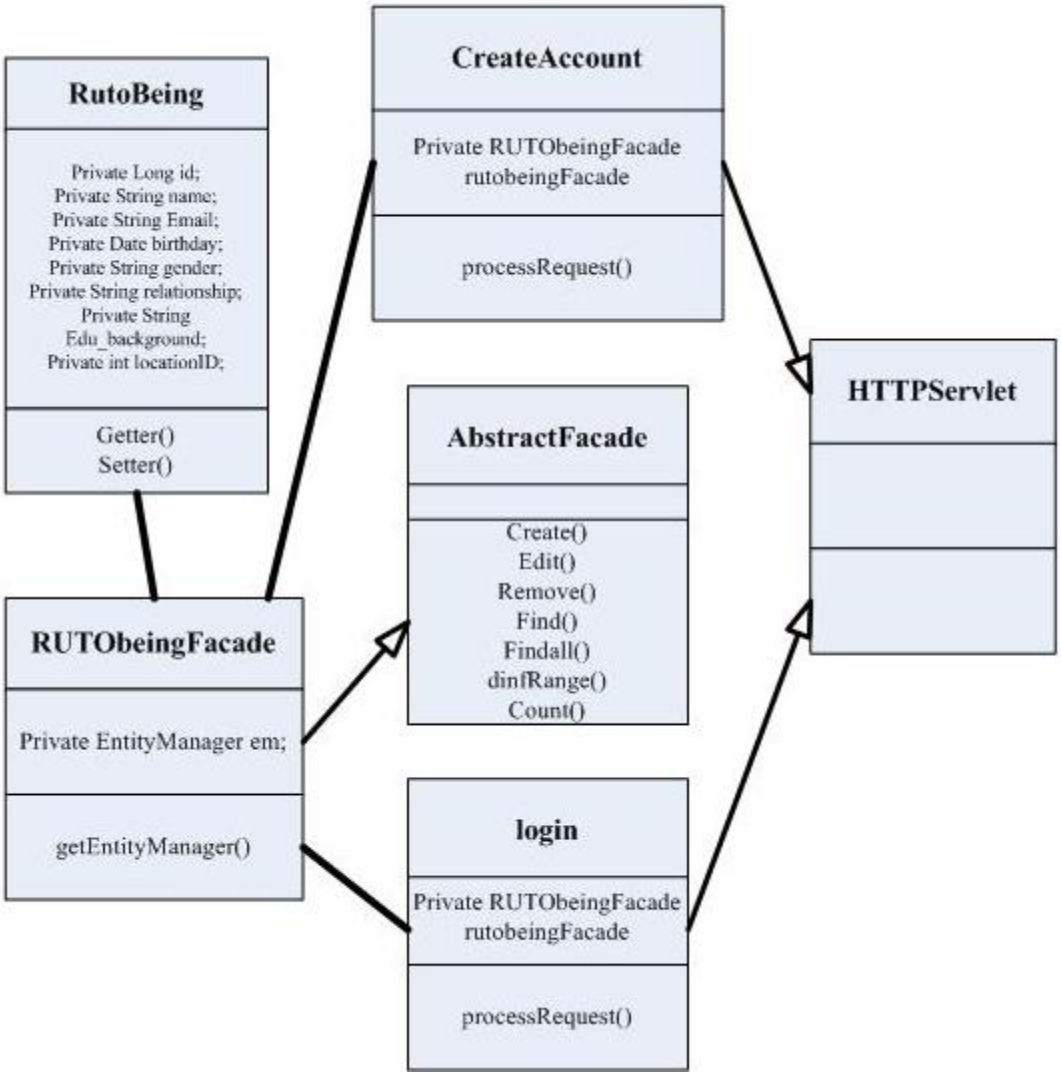


Figure 3-a. Regular User Login Design

As shown in Figure 3-b, an administrator is defined as a specific user who enjoys the email as "admin" and password as "admin". This administrator boasts the two privileges as deleting any user from our system and deleting any activity from our system. When we delete a specific user, we also delete all the related information(basic details in RUTObeing table, hobbies in Label table and address in Location table) of this user from back-end database. Similarly, when we delete an activity, we also delete all the related information(basic details in Activity table, tags in Label table and address in Location table) of this activity from back-end database.

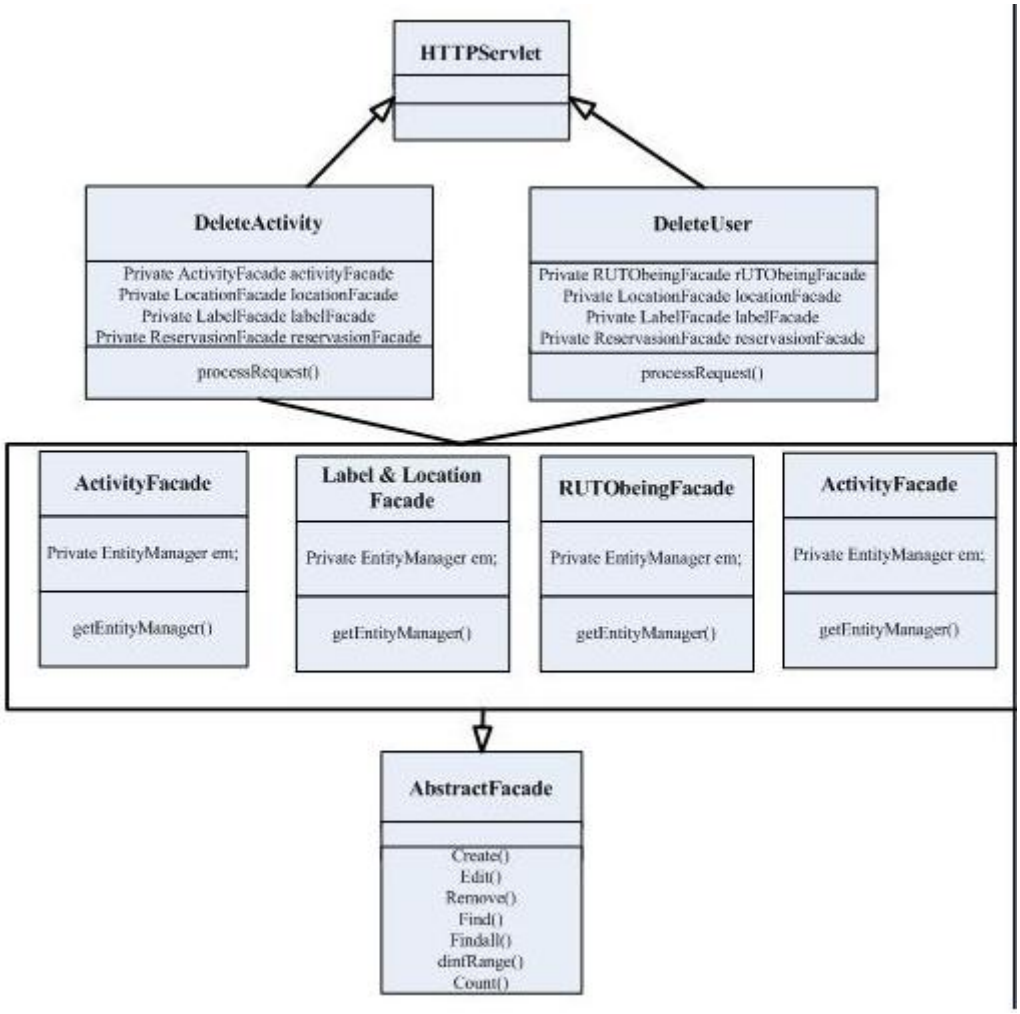


Figure 3-b. Admin Login Design

User Profile Design

The second sub-system is User Profile as shown in Figure 4. In this sub-system, we build the main frame and functions of the user profile page. We focus on 2 aspects. One is that user can edit or complete information of profile and save. Extended from HTTPServlet, SaveUserInfo Servlet can acquire updated user data from jsp files and then depend on RUTObeingFacade to edit or save information into database. The other aspect is to implement simple communication among all

users. Search class (Servlet) is designed to find user via email address. When user A wants to leave a message on user B, firstly, user A need to search for user B by inputting user B's email userB@ and visit user B's profile. SendingMessage (Servlet) records from whom, to whom and the message content and depends on session bean MessageFacade to save messages into table Message. When a user checks the messagebox, we retrieve from table Message and pick out all messages whose from whom email is the current user or to whom email the current user.

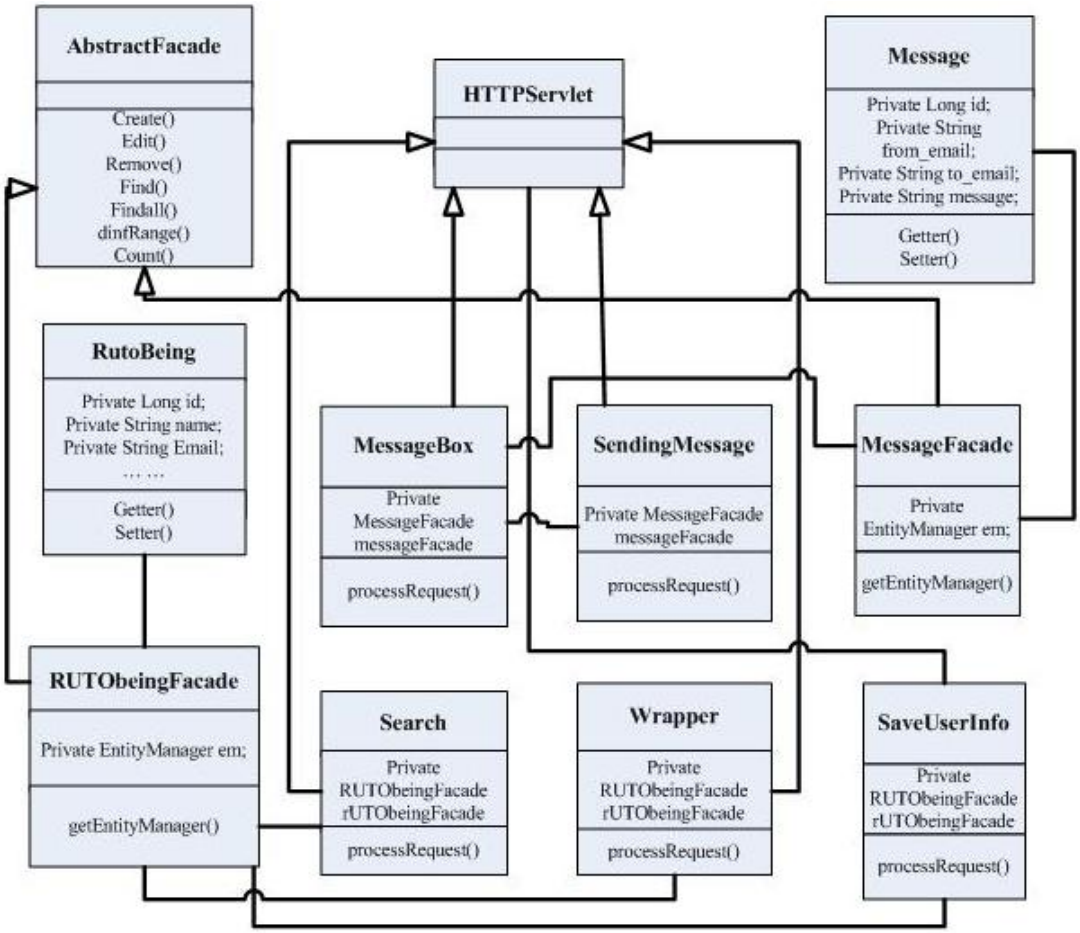


Figure 4. User Profile

Activity

Figure 5 shows the activity sub-system.Here, all classes are used to implement activity related functions. Activity is the entity bean which save the activities posted by users and post to the public. SaveActivity Servlet depends on ActivityFacade to save all information about an activity when it is created by some user. MyActivity is used to display the activities posted by the user. SearchActivity implements the function of searching activites which are posted by all the

users, in our application, we can also find out all the activities a user has joined or search a specific activity by location or tag.

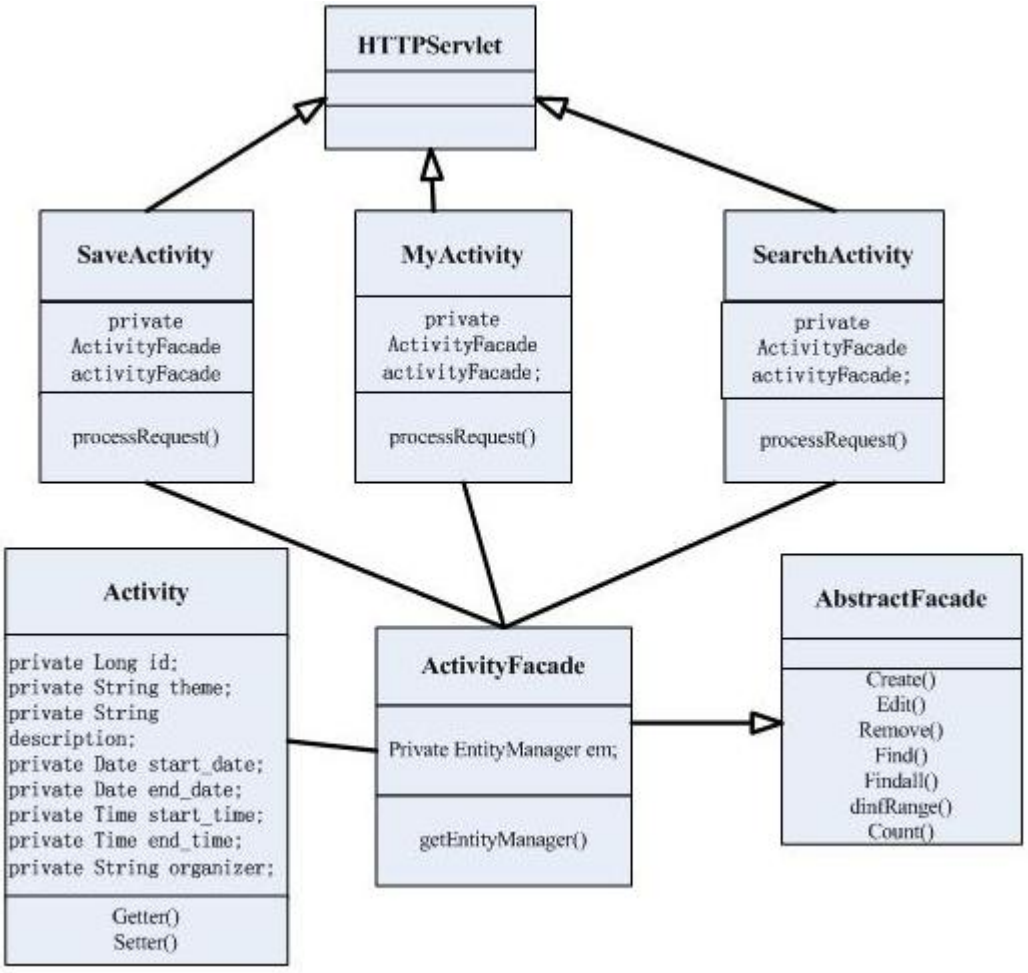


Figure 5. Activity

Recommendation

In the recommendation sub-system as shown in figure 6, there are mainly three components, friends recommendation, activity recommendation and show recommendation. In friends recommendation, with the aid of RUTObeingFacade, LocationFacade and LabelFacade, we can retrieve data from the corresponding tables in database and compute similarities between every

two users by matching related attributes in these three tables. We define the similarity rate (percentage) as # of shared fields divided by total # of attribute fields o f one user. In activity recommendation, with the aid of LocationFacade, LabelFacade and ActivityFacade, we can retrieve data from the corresponding tables in database and compute the similarity rate between an activity and a specific user. To print out the recommendation results, we give a decreasing order in similarity rate. Users can easily select friends and activities based on our recommendation.

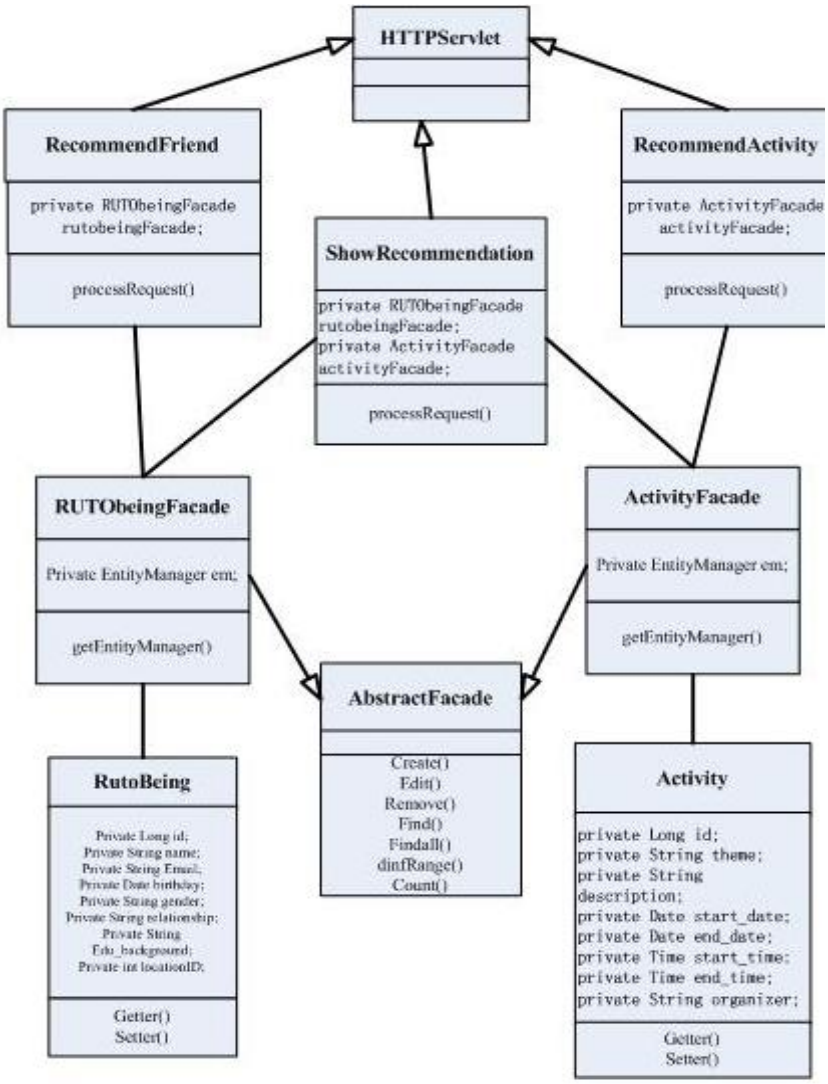


Figure 6 Recommendation

EJB Timer Service

As shown in the following figure 7, we create two EJB timer session beans, one is called Timerbean to tell if an activity is undergoing. We call the start( ) function of Timerbean when a user has created a new activity and asigned specific start date and start time to that activity. When

the start date and start time comes, @Timeout handleTimerout() function of Timerbean will be executed. In this handleTimeout() function, we change the state attribute of activity entity to "1" via calling ActivityFacade and the function edit() in the abstractFacade, which means that the activity is undergoing. Similarly, the other EJB Timer is declared as Timerfinishbean to tell if an activity is over. We call the start() function of Timerfinishbean when a user has created a new activity and filled in specific end date and end time of that activity. When the end date and end time comes, @Timeout handleTimeout() function of Timerfinishbean will be executed. In this handleTimeout() function, we change the state attribute of activity entity to "0" via calling ActivityFacade and the function edit() in the abstractFacade, which means that the activity is already over.

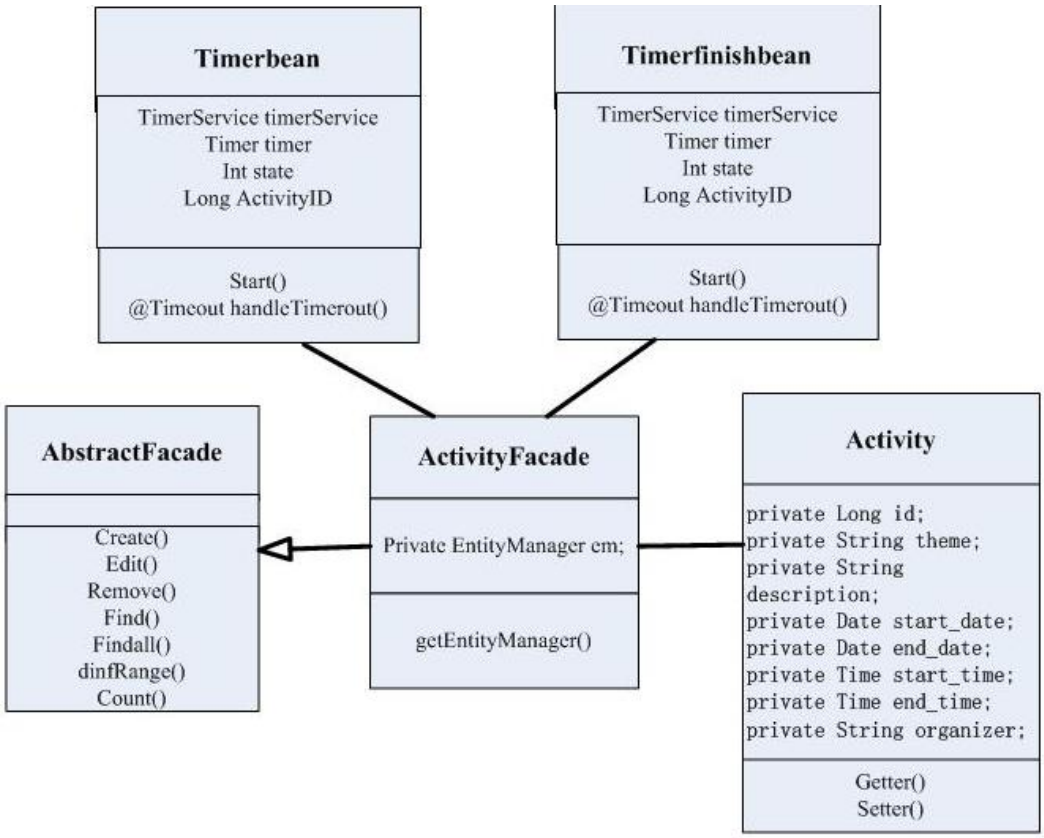
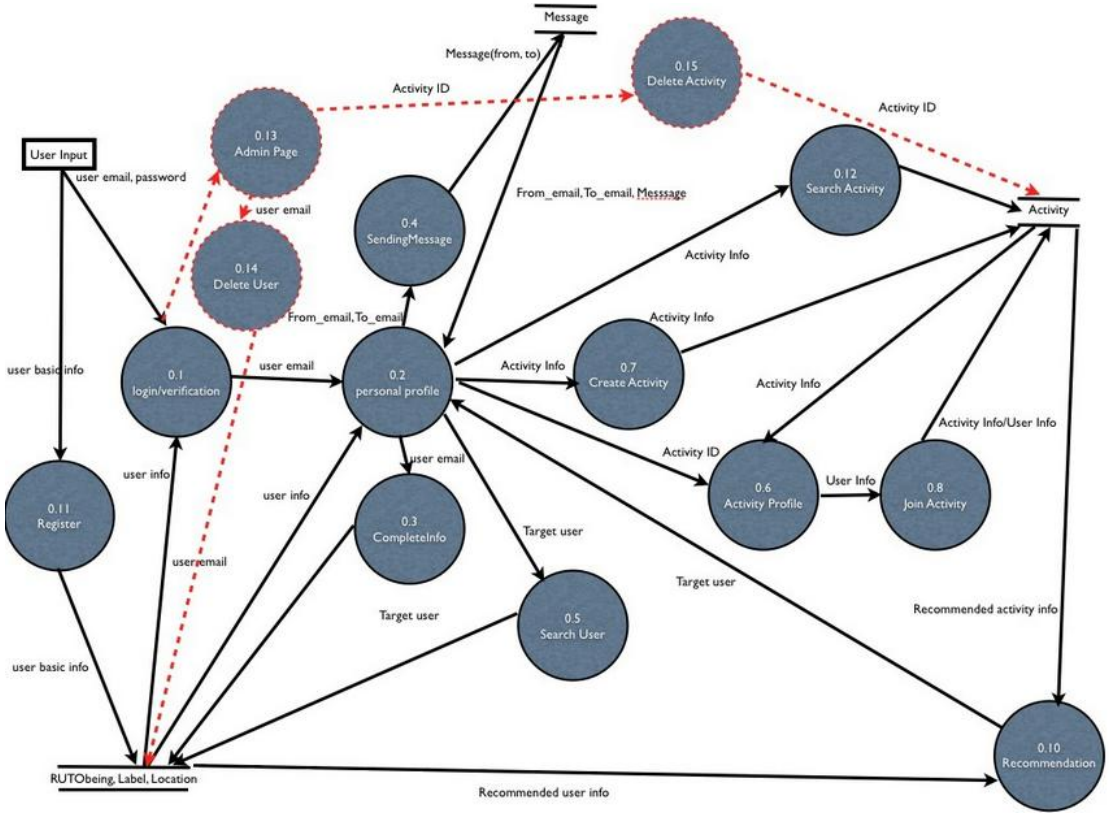


Figure 7 EJB Timer Service

Data Flow of User Scenario

This is our data flow diagram. It indicates the data flow and interaction between different actions. We have three main categories of database. The category RUTObeing is used to store users' information. Message is for storing message related information. And the activity information is kept in database. Data are transferred between these databases.



Component Model Services

This section involves details about which component model services planned to use and how they do in RUTO.

1. A set of ***Entities*** represents persistent business data, which is stored in each row of a database table. We may add behavior specific to that data (the methods here are just getters, setters). Note that the id for each entity is automatically generated by RUTO system, so we use @GeneratedValue (strategy = GenerationType.AUTO); EntityManager.find method can be used to look up entities by ids.
2. ***Session beans*** automatically generated by Netbeans IDE from corresponding entities : implement business processes and interact with clients;
3. ***Persistence Service***: a). to put a new record into database, we can use public function create() defined in class 'AbstractFacade' in Netbeans; b). to edit a existing tuple in database, we can use public function edit() defined in class 'AbstractFacade' in Netbeans; c) to remove a specific tuple from database, we can use public function remove() defiend in class 'AbstractFacade' in Netbeans;
4. ***Transaction***: transaction means an atomic sequences of actions(reads/writes). The state of persistent entities automatically synchronized to the database when the associated transaction commits. To realize transaction service, we can deploy Java Transaction API. JTA supplies begin, commit and rollback methods.
5. ***Queries***: Java Persistence Query Language, EntityManager.createQuery method used to create dynamic queries defined within business logic. For example, in our RUTO system, we can use EntityManager.createNamedQuery method to create static queries defined in annotation metadata.

   @NamedQuery(name="findAllRUTObeingsWithEmail",query="SELECT r FROM RUTObeing r WHERE r.name LIKE :SomeEmail").
6. ***EJB Timer Service***: we might use EJB timer service and timeout service in our system. For example, we can build a timerbean class, and define a starttimer() function before @Timeout and a processtimeout() function behind. Let's say that we assign a timer to each activity on our system, when an activity starts, the timer begin to time, the duration time depends on the

finishing time of this activity. We might code in the @Timeout section to inactivate the corresponding activity.

7. ***JAAS***: We use EJB security service to make our administrator login more safely. We developed a login module ourselves, and called it using our own callback handler. We also configured the glassfish server to enable this feature.
8. ***Ajax***: We use Asynchronous JavaScript to deal with our asynchronous message passing. For example, while users stay on their personal page, they will be informed if a new message is coming. Or, it a new message comes when a user is off-line, he or she will be informed when log in .

Response to Demo Concerns

In demo, Jon asked us several questions, Here we will give brief answers.

Q1, can you show me how to change user password?

A1, we didn't implement that part by demo, But now we have implemented this function. User can change password from user's personal page by clicking on change password button.

Q2, can you show me how to change the privacy of user profile?

A2, so far we haven't implemented that function. We plan to do that on the second iteration.

Q3, can you show me how to change activity informations.

A3, so far we haven't implemented that function. We plan to do that on the second iteration.

Also, we want to explain a little bit about the unexpected popup windows and missing buttons. The fact is that some students entered Java script in our text box and thus change our webpages' function and behavior. Our problem is that we did not verify the input carefully, leaving this chance to hackers. We will focus more on security issue on second iteration.