StepsAddFriends

Before we jump into the steps, i assume the reader has a basic working knowledge about Rave.

To add a person as your friend, an association must exist between you and the person whom you want to add as your friend.

To maintain this association we use the PERSON_ASSOCIATION table. The association table contains 3 columns viz. follower, followedBy and status.

Let A and B be 2 users and A wants to add B as his friend. To establish a relationship between these 2 users, user A must send a friend request to user B.

Sending Friend Requests:

This would be stored in the DB as follows.

Follower – A

Followedby – B

Status – Sent.

Follower – B

Followedby – A

Status – Received.

A snapshot of the data in the DB is shown below

SELECT * FROM PERSON_ASSOCIATION;

ENTITY_ID	STATUS	FOLLOWEDBY_ID	FOLLOWER_ID
1	SENT	2	1
2	RECEIVED	1	2

(2 rows, 5 ms)

Where

FOLLOWER_ID – 1 is the entity ID of USER A

FOLLOWEDBY_ID – 2 represents USER B who should be displayed in the friends list of USER A.

STATUS – Specifies the status of the relationship(whether a friend request is sent/ received /accepted)

One important thing to note here is that, adding a friend is mutual. i.e If A adds B as his friend and incase B accepts the request, then B should be in the friend list of A and A should be in friend list of B. That is why we have 2 entries in the DB.

In order to make this work, we would have to add a method in JpaPersonRepository.java

```
public boolean addFriend(String friendUsername, String username) {
        JpaPersonAssociation senderItem = new JpaPersonAssociation();                senderItem.setFollower
(personConverter.convert(findByUsername(username)));
        senderItem.setFollowedby(personConverter.convert(findByUsername(friendUsername)));
        senderItem.setStatus(FriendRequestStatus.SENT);
        senderItem = saveOrUpdate(senderItem.getEntityId(), manager, senderItem);

        JpaPersonAssociation receiverItem = new JpaPersonAssociation();
        receiverItem.setFollower(personConverter.convert(findByUsername(friendUsername)));
        receiverItem.setFollowedby(personConverter.convert(findByUsername(username)));
        receiverItem.setStatus(FriendRequestStatus.RECEIVED);
        receiverItem = saveOrUpdate(receiverItem.getEntityId(), manager, receiverItem);

        if(senderItem.getEntityId()!=null && receiverItem.getEntityId()!=null)
                return true;
        else
                return false;
}
```

The repository would be implementing an interface, so add a method declaration in PersonRepository.java. The same would be the case for any classes which implements any interface.

The repository would be called by a service, which in our case is the DefaultUserService.java

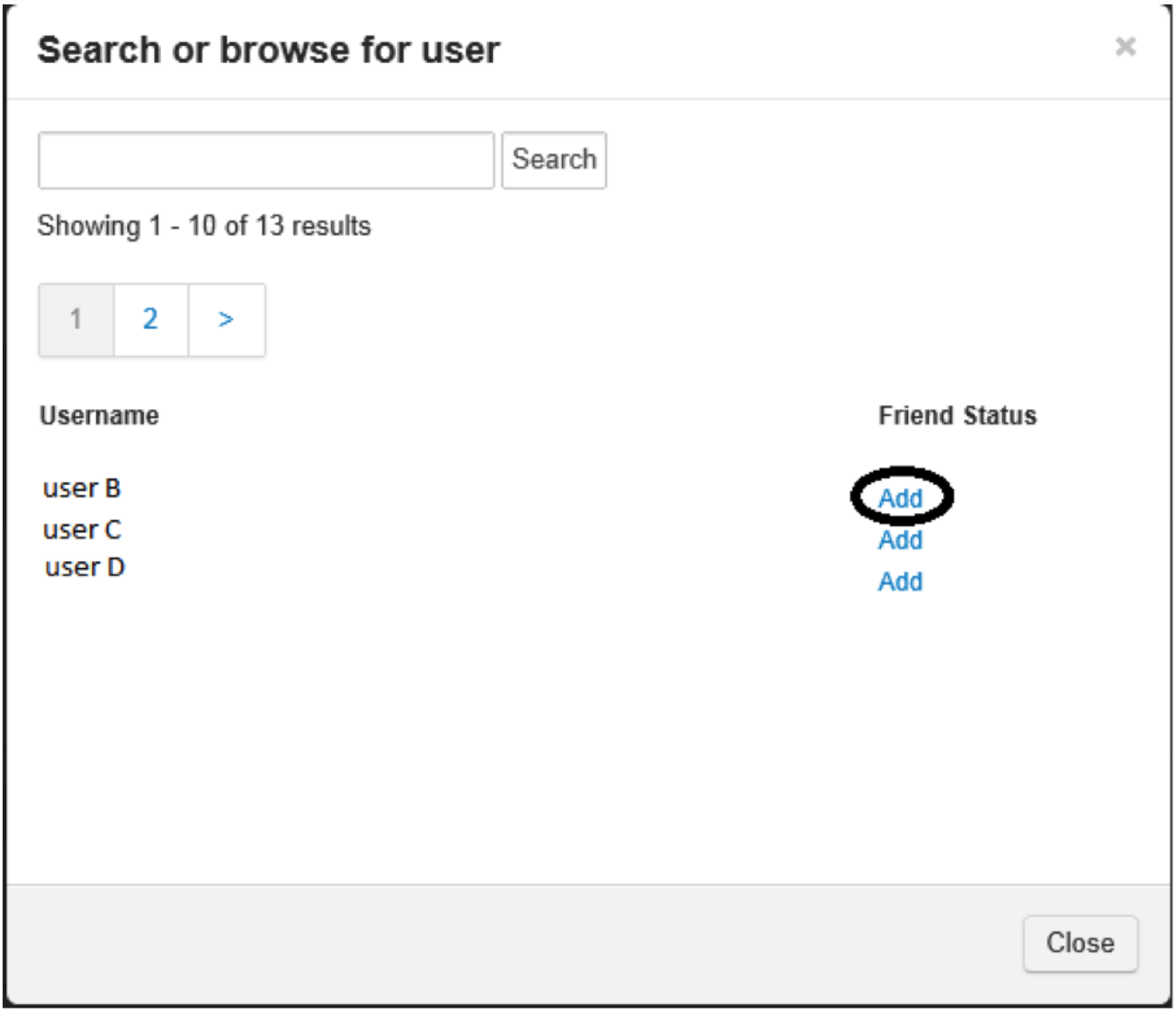
```
   public boolean addFriend(String friendUsername, String username) {
           return personRepository.addFriend(friendUsername,username);
   }
```

The service can be called by many controllers and APIs. In our case, the service would be called by PersonApi. The reason is, URL hits would be mapped by controllers but RPC and REST calls would be mapped by APIs and adding a friend is done by an RPC call from javascript.

The PersonApi should have a method like this for adding a friend.

```
   @ResponseBody
   @RequestMapping(value = "{friendUsername}/addfriend", method = RequestMethod.POST)
   public RpcResult<Boolean> addFriend(@PathVariable final String friendUsername) {
           return new RpcOperation<Boolean>() {
                   @Override
                   public Boolean execute() {
                        try {
                        String name = URLDecoder.decode(friendUsername, "UTF-8");
                          boolean result = userService.addFriend(name, userService.getAuthenticatedUser().
getUsername());
                           return result;
                        } catch (UnsupportedEncodingException e) {
                               return false;
                        }
                   }
           }.getResult();
   }
```

Now let's come to the UI part of it. Assume that the list of users in Rave is displayed to the user A and a link is available nearby each user to add him as a friend. So now to add user B as a friend, user A would click on 'Add' option near user B.



This would call the addFriend method in rave_person_profile.js which would in turn call a method in rave_api.js which would trigger an RPC call captured by the PersonApi.java

```
#!javascript
function addFriend(userId, username){
        $('#friendStatusButtonHolder'+userId).hide();
        rave.api.rpc.addFriend({friendUsername : username,
            successCallback: function(result) {
                rave.personprofile.addFriendRequestUI(username);
                $('#friendStatusButtonHolder'+userId).empty();
                $('#friendStatusButtonHolder'+userId)
                    .append(
                            $("<a/>")
                            .attr("href", "#")
                            .attr("id", userId)
                            .attr("onclick", "rave.personprofile.removeFriendRequestSent(" +
                               userId+", '" + username+"');")
                            .text(rave.getClientMessage("common.cancel.request"))
                    );
                $('#friendStatusButtonHolder'+userId).show();
            }
        });
}
```

The rave_api.js would contain this method.

```
#!javascript
   function addFriend(args) {
          var user = encodeURIComponent(encodeURIComponent(args.friendUsername));
          $.post(rave.getContext() + path + "person/" + user + "/addfriend",
          function(result) {
              if (result.error) {
                  handleRpcError(result);
              }
              else {
                  if (typeof args.successCallback == 'function') {
                       args.successCallback(result);
                  }
              }
         }).error(handleError);
   }
```

Receiving Friend Request:

Now that user A has sent his friend Request to user B, user B should be made aware of the friend request from A.

To display the lists in the profile page, first we have to set the data into the model so that it can be displayed in the JSP. To set the data in the model, we should make the following changes in the ProfileController.java

```
final NavigationMenu topMenu = new NavigationMenu("topnav");

NavigationItem friendRequestItems = new NavigationItem("page.profile.friend.requests", String.valueOf
(friendRequests.size()) , "#");
for(Person request : userService.getFriendRequestsReceived(username)) {
        NavigationItem childItem = new NavigationItem((request.getDisplayName()!=null && !request.
getDisplayName().isEmpty())? request.getDisplayName() : request.getUsername(), request.getUsername(), "#");
        friendRequestItems.addChildNavigationItem(childItem);
}
topMenu.addNavigationItem(friendRequestItems);
topMenu.getNavigationItems().addAll((ControllerUtils.getTopMenu(view, refPageId, user, false).
getNavigationItems()));
model.addAttribute(topMenu.getName(), topMenu);
```

The userService should now add a method to get the list of friend requests received from the DB by calling the repository.

The below method should be added to JpaPersonRepository.java

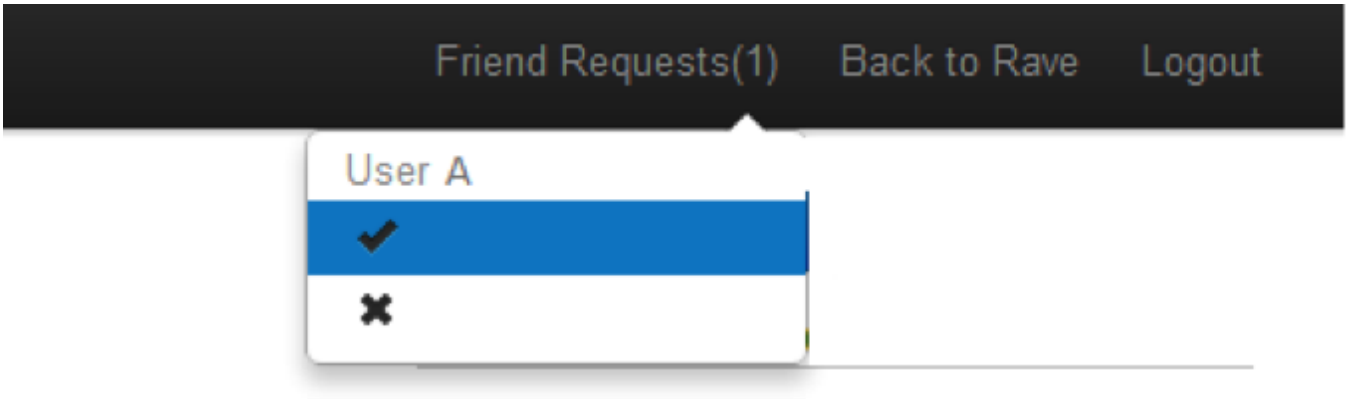
```
public List<Person> findFriendRequestsReceived(String username) {
   TypedQuery<JpaPerson> friends = manager.createNamedQuery(JpaPerson.FIND_FRIENDS_BY_USERNAME, JpaPerson.
class);
   friends.setParameter(JpaPerson.USERNAME_PARAM, username);
   friends.setParameter(JpaPerson.STATUS_PARAM, FriendRequestStatus.RECEIVED);
   return CollectionUtils.<Person>toBaseTypedList(friends.getResultList());
}
```

```
#!javascript
The query is “select a.followedby from JpaPersonAssociation a where a.follower.username = :username and a.
status = :status”
```

So once the list of friend requests is set to the model, we can display it in the jsp as we want. Below implementation displays it as a dropdown box with accepting and declining options. This is added to the navbar.tag so that it gets displayed in the top navigation menu.

```
#!jsp
<c:when test="${navItem.name=='page.profile.friend.requests'}">
  <li class="dropdown"><a href="#" data-toggle="dropdown" class="dropdown-toggle friendRequestDropdownLink"
><fmt:message key="${navItem.name}"/>(${navItem.nameParam})</a>
  <c:choose>
  <c:when test="${navItem.hasChildren}">
        <ul class="dropdown-menu friendRequestDropdown">
<c:forEach items="${navItem.childNavigationItems}" var="childItem">
                <li class="requestItem">${childItem.name}
                        <a class="acceptFriendRequest" id="${childItem.nameParam}" href="#"><i class="icon-ok"><
/i></a>
                        <a class="declineFriendRequest" id="${childItem.nameParam}" href="#"><i class="icon-
remove"></i></a>
                </li>
        </c:forEach>
        </ul>
  </c:when>
  <c:otherwise>
<ul class="dropdown-menu">
                <li>No Friend Requests</li>
        </ul>
  </c:otherwise>
  </c:choose>
  </li>
</c:when>
```

The dropdown for friend request would look like this.



The user can either accept the friend request or decline the friend request.

Accepting Friend Requests:

When user B accepts the friend request sent by user A, then we should do another RPC call to persist the relationship.

The accept friend request method in rave_person_profile.js looks like this

```
#!javascript
function acceptFriendRequest(userId, username){
   $('#friendStatusButtonHolder'+userId).hide();
   rave.api.rpc.acceptFriendRequest({friendUsername : username,
       successCallback: function(result) {
           rave.personprofile.removeFriendRequestReceivedUI(username);
           $('#friendStatusButtonHolder'+userId).empty();
           $('#friendStatusButtonHolder'+userId)
               .append(
                       $("<a/>")
                       .attr("href", "#")
                       .attr("id", userId)
                       .attr("onclick", "rave.personprofile.removeFriend(" +
                           userId+", '" + username+"');")
                       .text(rave.getClientMessage("common.remove"))
               );
           $('#friendStatusButtonHolder'+userId).show();
       }
    });
}
```

The RPC function call in rave_api.js is

```
#!javascript
   function acceptFriendRequest(args) {
   var user =  encodeURIComponent(encodeURIComponent(args.friendUsername));
   $.post(rave.getContext() + path + "person/" + user + "/acceptfriendrequest",
        function(result) {
             if (result.error) {
                    handleRpcError(result);
             }
             else {
                    if (typeof args.successCallback == 'function') {
                        args.successCallback(result);
                    }
             }
        }).error(handleError);
   }
```

You may note that we have encoded the username before we perform the RPC calls. The reason for encoding the username is Rave supports OpenId login and OpenId usernames contains special characters and URL like usernames which when passed without encoding might cause problems. So we encode it and decode it at the receiving end in the PersonApi.java

The PersonApi.java method which captures the RPC call is

```
@ResponseBody
@RequestMapping(value = "{friendUsername}/acceptfriendrequest", method = RequestMethod.POST)
public RpcResult<Boolean> acceptFriendRequest(@PathVariable final String friendUsername) {
    return new RpcOperation<Boolean>() {
        @Override
        public Boolean execute() {
                try{
                        boolean result = userService.acceptFriendRequest(URLDecoder.decode(friendUsername, "UTF-
8"), userService.getAuthenticatedUser().getUsername());
                        return result;
                }catch (UnsupportedEncodingException e) {
                    return false;
}
}
    }.getResult();
}
```

The Repository method would retrieve the old values, change the status to accepted and then store it back to the DB.

```
public boolean acceptFriendRequest(String friendUsername, String username) {
        TypedQuery<JpaPersonAssociation> query = manager.createNamedQuery(JpaPersonAssociation.
FIND_ASSOCIATION_ITEM_BY_USERNAMES, JpaPersonAssociation.class);
       query.setParameter(JpaPersonAssociation.FOLLOWER_USERNAME, username);
       query.setParameter(JpaPersonAssociation.FOLLOWEDBY_USERNAME, friendUsername);
       JpaPersonAssociation receiverItem = getSingleResult(query.getResultList());
       receiverItem.setStatus(FriendRequestStatus.ACCEPTED);
       receiverItem = saveOrUpdate(receiverItem.getEntityId(), manager, receiverItem);

        query = manager.createNamedQuery(JpaPersonAssociation.FIND_ASSOCIATION_ITEM_BY_USERNAMES,
JpaPersonAssociation.class);
       query.setParameter(JpaPersonAssociation.FOLLOWER_USERNAME, friendUsername);
       query.setParameter(JpaPersonAssociation.FOLLOWEDBY_USERNAME, username);
       JpaPersonAssociation senderItem = getSingleResult(query.getResultList());
       senderItem.setStatus(FriendRequestStatus.ACCEPTED);
       senderItem = saveOrUpdate(senderItem.getEntityId(), manager, senderItem);

        if(receiverItem.getEntityId()!=null && senderItem.getEntityId()!=null)
                return true;
        else
                return false;
}
```

DB snapshot after friend request is accepted.

SELECT * FROM PERSON_ASSOCIATION;

ENTITY_ID	STATUS	FOLLOWEDBY_ID	FOLLOWER_ID
1	ACCEPTED	2	1
2	ACCEPTED	1	2

(2 rows, 5 ms)

Removing Friends/Friend Requests:

For Removing friends or friend request, the RPC calls are similar to Adding friends but the repository would search for records from the DB and remove them instead of inserting new ones. For further more detailed implementation, download the Rave source code and start Raving 🙂