TopCoder Security Groups Frontend Part 3 Requirements Specification

1.Scope

1.1 Overview

TopCoder is a company which administers contests in computer programming and provides software development services to other companies using a component based development approach.

The development process is managed using a series of tools, known as the Topcoder competition platform. The tools empower customers and client employees to directly take advantage of the competition platform to develop quickly and efficiently their software projects.

From time to time Topcoder updates different applications composing the competition platform in order to meet client needs or to make the development process more efficient.

This project addresses one of these enhancement requests, namely management of client overall account in the entire platform. The client account will include all client assets as well as arbitrary groups of users. For example, a client has many projects that are created in Cockpit and each of those projects has multiple contests. Each customer, project and contest also has a set of users that need to be aware of them.

This front-end module will provide all the necessary pages and corresponding front-end actions for all the new user interfaces.

This component implements JSPs and Struts 2 Actions related to ARS 2.15, 2.16, 2.20, 2.24.

1.1.1 *Version*

1.0

1.2 Logic Requirements

This component implements JSPs and Struts 2 Actions related to ARS 2.15, 2.16, 2.20, 2.24.

Note that this component uses the BaseAction and SearchUserAction of the Frontend Part 1 component.

1.2.1 *SendInvitationAction*

This action is used to send group invitation.

1) input

   Front end page uses this method to get groups list. It does the following:

   // search groups of the client name

   GroupSearchCriteria c = new GroupSearchCriteria();

   c.setClientName(clientName);

   groups = groupService.search(c, 0, 0).getValues();

   // filter groups

   // get current user id

   long userId = getCurrentUserId();

```
// for admin, no need to filter
if (!authorizationService.isAdministrator(userId)) {
    // filter groups
    List<Client> clients =
customerAdministratorService.getCustomersForAdministrator(userId);
    List<Long> groupIds = authorizationService.getGroupIdsOfFullPermissionsUser(userId);
    for any group of groups {
        if (group.id is not in groupIds) and (group.client.id is not any of ids of clients) {
            the group should not be included;
        }
    }
}
```

2) execute

This methods sends invitations to the given handles. It does the following:

```
For each groupName in groupNames {
    Find group:Group of the groupName and clientName, using groupService.search method,
    there should be only one group found;
    For each handle of handles {
        Find groupMember:GroupMember of the handle from group.groupMembers;
        if (groupMember == null) {
            // create a new group member
            groupMember = new GroupMember();
            Find userId of the handle using userService;
            groupMember.setUserId(userId);
            groupMember.setUseGroupDefault(true);
            groupMember.setActive(false);
            groupMember.setGroup(group);
            group.groupMembers.add(groupMember);
            groupService.update(group);
        }
        // create invitation
        GroupInvitation invitation = new GroupInvitation();
        invitation.setGroupMember(groupMember);
        invitation.setStatus(InvitationStatus.PENDING);
        invitation.setSentOn(new Date());
```

```
            // save invitation
            groupInvitationService.add(invitation);
            // construt URLs
            String urlPrefix = acceptRejectUrlBase + "/?invitationId=" + invitation.getId()
              + "&accepted=";
            String acceptUrl = urlPrefix + "true";
            String rejectUrl = urlPrefix + "false";
            // send invitation
            groupInvitationService.sendInvitation(invitation, registrationUrl,
              acceptUrl, rejectUrl);
          }
        }
```

1.2.2 *CreateCustomerAdminAction*

This action is used to create customer administrator.

1) input

search all clients using the clientService, result is put to the clients output field.

```
// filter clients by user role
// get current user id
long userId = getCurrentUserId();

// for admin, no need to filter
if (!authorizationService.isAdministrator(userId)) {
  List<Client> possibleClients =
customerAdministratorService.getCustomersForAdministrator(userId);
  for any client of clients {
    if (client.id is not any of ids of possibleClients) {
      the client should not be included;
    }
  }
}
```

2) execute

```
// search user of the handle
List<UserDTO> users = userService.search(handle);
If users does not contain exactly one user, throw exception;
long userId = users.get(0).getId();
// get client
Client client = clientService.get(clientId);
```

```
// create customer admin
CustomerAdministrator ca = new CustomerAdministrator();
ca.setUserId(userId);
ca.setClient(client);
// save it
customerAdministratorService.add(ca);
```

1.2.3 *AccessAuditingInfoAction*

This action is used to access auditing information.

It does the following:

```
// filter historical data
// get current user id
long userId = getCurrentUserId();
criteria.setUserId(userId);
// search historical data
historicalData = groupMemberService.search(criteria, pageSize, page);

// get groups of the historical data
groups = new ArrayList<Group>();
for each data of historicalData {
   Group g = groupService.get(data.getGroupId());
   groups.add(g);
}
```

1.3 *Validation*

Validation is performed on server side using Struts 2 XML configuration based validators.

For complicated validation that can not be easily performed using the above mentioned validators, the action may override the validate method to perform custom validation.

Validation errors should be rendered back to the JSP pages.

Designers should refer to ARS corresponding sections for validation details.

1.3.1 *Prototype*

Prototype will be provided for this component. The pages related to ARS 2.15, 2.16, 2.20, 2.24 are in scope.

These pages should be converted to JSP pages, and properly bound to the above actions.

Note that in the above actions, result view names of the execution methods are not discussed, these are up to designers, also, sample Spring and Struts 2 configuration should be provided, so that page navigation works properly.

1.3.2 *Application Management*

See ADS 1.3 for application management details.

1.4 Required Algorithms

None.

1.5 Example of the Software Usage

This component implements JSPs and Struts 2 Actions related to ARS 2.15, 2.16, 2.20, 2.24. This component will be used for handling browser requests specific to group management, and for rendering results to be shown to the end user.

1.6 Enhancement policy

N/A - See details here:
http://apps.topcoder.com/forums/?module=Thread&threadID=728272&start=0

1.7 Future Component Direction

None

2. Interface Requirements

2.1.1 *Graphical User Interface Requirements*

None

2.1.2 *External Interfaces*

None

2.1.3 *Environment Requirements*

Development language: Java 1.6

Compile target: Java 1.6

2.1.4 *Package Structure*

com.topcoder.security.groups.actions

3. Software Requirements

3.1 Administration Requirements

3.1.1 *What elements of the application need to be configurable?*

See the actions in TCUML, fields marked with “config” should be configurable, they are injected by Spring IoC container.

3.2 Technical Constraints

3.2.1 *Are there particular frameworks or standards that are required?*

Spring 2.5.6

Struts 2.1.8

3.2.2 *Component Dependencies:*

Frontend Part 1 1.0

Its BaseAction and SearchUserAction are used in this component.

3.2.3 *Third Party Component, Library, or Product Dependencies:*

Spring 2.5.6: http://www.springsource.org/

Struts 2.1.8: http://struts.apache.org/

3.2.4 *QA Environment:*

JBoss 4

3.3 **Design Constraints**

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines.

3.4 **Required Documentation**

3.4.1 *Design Documentation*

Use-Case Diagram

Class Diagram

Sequence Diagram

Component Specification

3.4.2 *Help / User Documentation*

Design documents must clearly define intended component usage in the 'Documentation' tab of the TopCoder UML Tool.