Struts Actions 4 Requirements Specification

1. Scope

1.1 Overview

As we progress in upgrading the Direct application, the next thing we are focusing on is the complete flow of launching a contest. This includes everything from entering the basic data for a contest to payment, spec review, provisioning, etc. We currently have a basic flow for launching a contest created in the service layer and being assembled for the html version of Direct. This contest will dig deeper into the process of launching a contest to address the requirements introduced and clarified by the requirements that are referenced in this contest.

This component provides the struts actions to repost contest, create new component version, delete contest, link contests, modify project budget, start specification review, and view specification review result.

1.1.1 Version

1.0

1.2 Logic Requirements

Each of the following actions will be designed as a Struts action with AJAX handling capability. It is up to the designer to make best use of Struts capabilities, including interceptors and base action classes. The only client of this application will be the JSPs, so there are no APIs or structural requirements. The designer is free to refactor any interceptor or action as long as requirements are met.

This component is responsible for several actions that are used to perform requests from the Frontend.

1.2.1 Struts Actions

Here we have some general information about the actions:

- All actions will extend the AbstractAction class from the struts framework component.
- Javabean properties (i.e. getXXX/setXXX) should be used for all action data. The input data will be provided as request parameters and will need to be mapped to these setters using JavaBean conventions.
- The design may rely on some data conversion to take place before the action, such as converting dates, but the designer must specify what those conversions must be.
- The execute method will place all result data in the AggregateDataModel then it will set it to the action to make it available in the ValueStack. Essentially, we expect return parameters to be sent back. Use simple “return” key for the model.

1.2.2 Validation

Validation here will comprise the user input as specified in the use cases provided in the ARS for each relevant action. Validation errors would be packaged in the ValidationErrors entity with each property that fails validation placed in the ValidationErrorRecord entity. ValidationErrorRecord contains a messages array field that allows multiple validation errors to be provided per field (for example, a field could be too long and contain incorrect format).

1.2.3 Repost Contest

The user can repost previously cancelled, failed or completed contest. The copy of the contest will be created and placed to the Draft state.

Input:
contestId – software contest id

Process:
The DirectServiceFacade.repostSoftwareContest method is called to repost the contest.

Result:
The id of the newly reposted contest.

1.2.4 *Create New Component Version*

The user can create new version of the Software Design/Development component, which was already completed.

Input:
contestId – software contest id

Process:
The DirectServiceFacade.createNewVersionForDesignDevContest method is called to create the new version for design/dev contest.

Result:
The id of the newly created contest.

1.2.5 *Delete Contest*

The user can remove any of his/her contests. First, the user can break active contest and it will move to the “cancelled” state. Second, it will remove draft/cancelled/failed contest from the project.

Input:
contestId – the contest id.
studio – flag indicating it’s a software or studio competition

Process:
The DirectServiceFacade.deleteContest method is called to delete the contest.

Result:
None.

1.2.6 *Get Contest Links*

The action will get contest’s parent and child links for display. It only applies to software contest.

Input
contestId – the contest id.

Process
The DirectServiceFacade’s getParentProjects and getChildProjects methods should be called to get the parent and child projects for display.

Result
Retrieved parent and child projects.

1.2.7 *Link Contests*

The user can assign relationships between any contests in his/her project. The user can provide links to one or many contest from any contest within the same project.

Input:
contestId – the current contest id.
parentContestIds – zero or more ids of the parent contests.
childContestIds – zero or more ids of child contests

Process
The DirectServiceFacade's updateProjectLinks method should be called to update the project link.

Result
None

1.2.8 *Modify Project Budget*

The action will modify the billing project's budget.

Input:
billingProjectId – the id of the selected billing project
changedAmount – the amount of change to the budget. It must be positive.
action – increase or decrease the budget

Process
The DirectServiceFacade.updateProjectBudget method should be called to update the project's budget. If the action is decrease, the -changedAmount should be passed to the façade method.

Result
The retrieved project budget change data.

1.2.9 *Start Specification Review*

The action will start the specification review.

Input:
contestId – the contest id.

Process
The ContestServiceFacade.createSpecReview method should be called to create the spec review. And then its markSoftwareContestReadyForReview method should be called to start the spec review.

Result:
None

1.2.10 *View Specification Review Result*

The action will get specification result for display.

Input:
contestId – the contest id

Process
The ContestServiceFacade.getScorecardAndReview method should be called to get the scorecard and specification review for display.

Result
Retrieved scorecard and review results.

1.2.11 *Struts action mapping*

The designer does not need to provide a mapping file for struts actions. All struts and spring files will be provided by the assembly.

1.2.12 *Services*

The actions will directly use façade classes. These will be injected.

1.2.13 *Logging and Error Handling*

The actions will not perform logging and any error handing. If any errors occur, they actions will simply put the Exceptions into the model as a “result”. There will an interceptor that will process the logging of this.

1.2.14 *Transaction handling*

Any transactions will be handled externally by the container at the level of the façade. Nothing needs to be done in this component.

1.2.15 *Thread-safety*

Since we are operating in a servlet container, threading is not an issue.

1.2.16 *Configuration*

All configurations will be done via method injection. Each item being injected via a setter will have a corresponding getter.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The actions will handle contest launching requests.

1.5 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: Java1.5, J2EE 1.5
- Compile target: Java1.5, J2EE 1.5
- Application Server: JBoss 4.0.2

- Informix 11

2.1.4 *Package Structure*

com.topcoder.service.actions

3. Software Requirements

3.1 Administration Requirements

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

- User session key
- Login page name
- Service Facades

3.2 Technical Constraints

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

- Struts 2.1.1
- Spring 3.0
- EJB 3.0

3.2.2 *TopCoder Software Component Dependencies:*

- Struts Framework 1.0
- Contest Service Façade 1.0
- Direct Service Façade 1.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

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

None

3.2.4 *QA Environment:*

- Java 1.5/J2EE 1.5
- JBoss 4.0.2
- Informix 11
- MySQL 5.1
- Struts 2.1.8.1
- Spring 3.0
- Javascript 1.8
- Mozilla Firefox 2.0/3.0
- IE 6.0/7.0
- Google Chrome
- Safari 3/4

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.