Struts Actions 2 Requirements Specification

1. Scope

1.1 Overview

We are going to move Direct(www.topcoder.com/direct) from flex to HTML based platform. Release 1 contains Launch Contest page(s). Clients can use this page to create/update studio (nonsoftware) and software contests. Existing back end services will be used.

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 API 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. The ARS sections mapping to these actions are stated.

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 ARS will state the names of the parameters, and we will assume camel case equivalents. In essence, this component will drive how the parameters are sent to the action as well as what resultant data is sent back
- 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 return data in 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 Get contest technologies action (ARS 2.6.1.4, same for below sections)

This action will return all available technologies using ContestServiceFacade‘s getActiveTechnologies.

1.2.4 Get contest categories action (2.6.1.5)

This action will return all available technologies using ContestServiceFacade‘s getActiveCategories.

1.2.5 *File Upload and Attach contest file action*

Since Struts already provides an interceptor for file uploading the assemblers will simply need to configure it for the specific actions. Please consult the following class documentation for Struts: `FileUploadInterceptor` (`org.apache.struts2.interceptor.FileUploadInterceptor`)
The bean needed for the data transfer will be defined here.

An action will persist contest files that will be made available directly with the use of the above File Upload interceptor using the data laid out in ARS 2.8.1.3. Execute the save using the ContestServiceFacade's uploadDocumentForContest.

1.2.6 *Get documents of contest action (2.8.1.8)*

This action will return all documents of the contest by getting the contest itself. Use ContestServiceFacade's getContest or getSoftwareContestByProjectId based on whether this is a studio or non-studio contest.

The studio documents are available in the StudioContest.ContestData entity, in the documentationUploads field. The software documents are available in the SoftwareCompetition.AssetDTO, in the documentation field.

Studio documents are in UploadedDocument instances, and Software documents are in the CompDocument instances.

1.2.7 *Delete document of contest action (2.8.1.10)*

This action will delete the document. It will first remove the document from the contest then delete the document itself. This is done using the ContestServiceFacade's removeDocumentFromContest then removeDocument methods, respectively.

1.2.8 *Pay by credit card action*

This action will make a payment for user using a credit card, as per ARS 2.11.1.2-3. This will use the processContestCreditCardPayment method of ContestServiceFacade.

It will also send an email to the user using the email address in the principal (can be gotten from session's Principal). Use Email Engine component. Use a configurable template for subject and body, where the template parameters will be the user email and billing parameters plus the payment result parameters.

1.2.9 *Pay by billing account action*

This action will make a payment for user using an existing billing account, as per ARS 2.11.1.2-3. This will use the processContestPurchaseOrderPayment method of ContestServiceFacade.

It will also send an email to the user using the email address in the principal (can be gotten from session's Principal). Use Email Engine component. Use a configurable template for subject and body, where the template parameters will be the user email and billing parameters plus the payment result parameters.

1.2.10 *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.

1.2.11 *Services*

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

1.2.12 *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.

1.2.13 *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.

1.2.14 *Thread-safety*

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

1.2.15 *Configuration*

All configuration 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
- Pipeline Service Façade 1.0
- Project Service Façade 1.0
- Email Engine 3.1

**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.