[TopCoder]

Check Profile Completeness Servlet Filter 1.0 Component Specification

1. Design

The TopCoder registration process to become a member of TopCoder is going to be redesigned. The main goal is to allow for a minimal set of required info that the user needs to enter to sign up (handle, email, and password).

The existing TC web registration process will ask a new user to enter a lot of data when he/she tries to register to TC web-site. The registration process takes a lot of time, so many users can simply break the registration process and leave non-registered

Thus there is a need of an easy to use and user friendly web-application for supporting registration on the new users and management of registered user profiles at TC web-site.

The main goal of this application is to simplify registration of the new users on TC web-site by minimizing the count of mandatory data fields for new account registration, and to improve usability of user profile management for the registered users. Any additional profile information will be requested from the user by the system as it is needed. For example, if a user registers to compete in an assembly contest the site would prompt them for any required info that they have not yet entered.

This component is responsible for implementing a servlet filter to check the profile completeness.

1.1 Design Patterns

1.1.1 DAO and DTO

External DAO's and DTO's are used by this component.

1.1.2 Chain of responsibility

The defined servlet filter is one element in the chain of responsibility of servlet filters defined for the application.

1.1.3 Inversion of control (IoC)

The component will be used with Spring framework, which will inject configuration parameters into the beans provided by this component.

1.1.4 Strategy

This component provides a context for using external stuff (e.g. DAO's) in a pluggable manner.

CheckProfileCompletenessProcessor implementation is strategy in the context of CheckProfileCompletenessFilter

ProfileCompletenessChecker and UserIdRetriever implementations are strategies in the context of DefaultCheckProfileCompletenessFilter.

1.2 Industry Standards

HTTP, Servlet

1.3 Required Algorithms

1.3.1 Logging

The application will log activity and exceptions using the Logging Wrapper, and the Logging Wrapper should be configured to use Log4j.

It will log errors at Error level, and method entry/exit information at DEBUG level.

Specifically, logging will be performed as follows, if logging is turned on.

- Method entrance and exit will be logged with DEBUG level.

  - Entrance format: [Entering method {*className.methodName}*]
  - Exit format: [Exiting method {*className.methodName}]*. Only do this if there are no exceptions.
- Method request and response parameters will be logged with DEBUG level
  - Format for request parameters: [Input parameters[{request_parameter_name_1}:{ request_parameter_value_1}, {request_parameter_name_2}:{ request_parameter_value_2}, etc.)]]
  - Format for the response: [Output parameter {response_value}] . Only do this if there are no exceptions and the return value is not void.
  - If a request or response parameter is complex, use its toString() method. If that is not implemented, then print its value using the same kind of name:value notation as above. If a child parameter is also complex, repeat this process recursively.
- All exceptions will be logged at ERROR level, and automatically log inner exceptions as well.
  - Format: Simply log the text of exception: [Error in method {*className.methodName}*: Details {*error details*}]

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry
3. Log method input parameters
4. If error occurs, log it and skip to step7
5. Log method exit
6. If not void, log method output value
7. Method exit

The toString() method of the entity can be used for the logging.

1.3.2 @PostConstruct

The method that is marked with <<@PostConstruct>> in TCUML should be set as the value of the “init-method” attribute in the bean declaration of the Spring application context file. See the attached demo for example.

1.4 Component Class Overview

CheckProfileCompletenessFilter

This class is the server filter, which is used to check, whether the topcoder member has provided enough details to access some activity. The concrete details of the profile completeness checking are defined in the pluggable CheckProfileCompletenessProcessor, implementation of which will be inserted by spring.

CheckProfileCompletenessProcessor

This interface defines the contract for checking the completeness of the profile. It is inserted by spring to the CheckProfileCompletenessFilter.

DefaultCheckProfileCompletenessProcessor

This class is the default implementation of the CheckProfileCompletenessProcessor. It processes the check only if the input request uri matches the configurable pattern. Then this implementation retrieve the logged user and checks, whether he has provided enough profile fields for performing some action. This check is done by configurable ProfileCompletenessChecker. If these fields are not filled, the user will be redirected to page to complete data and request uri will be stored in the session under the key requestURISessionKey.

ProfileCompletenessChecker

This interfaces defines the contract for checking profile completeness for the specific user action. The concrete details of data retrieval is up to the implementation.

BaseProfileCompletenessChecker

This class is the base class for checking the user profile completeness. It provides default implementation of isProfileComplete() method, which checks mandatory fields for every action of the user. These mandatory fields are: user first name, last name, address(including address line, city and country) and phone. It also provides methods to check, whether the user is competitor or customer.

ForumProfileCompletenessChecker

This class is the class for checking the user profile completeness for forum participation.

DirectProfileCompletenessChecker

This class is the class for checking the user profile completeness for using Topcoder Direct.

JiraProfileCompletenessChecker

This class is the class for checking the user profile completeness for using jira.

VMProfileCompletenessChecker

This class is the class for checking the user profile completeness for using Virtual Machines(VM).

CopilotProfileCompletenessChecker

This class is the class for checking the user profile completeness for performing copilot activity.

WikiProfileCompletenessChecker

This class is the class for checking the user profile completeness for accessing Confluence wiki.

OnlineReviewProfileCompletenessChecker

This class is the class for checking the user profile completeness for accessing Online Review.

SVNProfileCompletenessChecker

This class is the class for checking the user profile completeness for accessing svn.

CompetitionProfileCompletenessChecker

This class is the class for checking the user profile completeness for taking part in competitions.

StudioProfileCompletenessChecker

This class is the class for checking the user profile completeness for accessing TCS Studio.

UserIdRetriever

This interface defines the contract for retrieving the user id from the request. The concrete details of data retrieval are up to the implementation.

BaseUserIdRetriever

This class is the base class of the UserIdRetrievers. It simply stores the logger and the session key if necessary for accessing by the subclasses.

ForumUserIdRetriever

This class is the implementation of UserIdRetriever for the topcoder forums.

OnlineReviewUserIdRetriever

This class is the implementation of UserIdRetriever for the online review.

StudioUserIdRetriever

This class is the implementation of UserIdRetriever for the topcoder studio.

TCSiteUserIdRetriever

This class is the implementation of UserIdRetriever for the topcoder site("/tc").

DirectUserIdRetriever

This class is the implementation of UserIdRetriever for the topcoder direct.

WikiUserIdRetriever

This class is the implementation of UserIdRetriever for the topcoder confluence wiki.

JiraUserIdRetriever

This class is the implementation of UserIdRetriever for the topcoder atlassian jira.

1.5 Component Exception Definitions

CheckProfileCompletenessProcessorException

This exception is thrown, when there's any problem with checking profile completeness. If this exception occurred, the user will be redirected to the error page. This is the base class for non - runtime exceptions in this component.

CheckProfileCompletenessConfigurationException

This exception is thrown, when the configuration for profile completeness was not properly set or invalid.

ProfileCompletenessCheckerException

This exception is thrown, when there's any problem with checking profile completeness.

1.6 Thread Safety

Technically, CheckProfileCompletenessFilter is not thread – safe. But the initialization of the filter will be done in a thread – safe manner by web container. All other non thread – safe implementations will be initialized in a thread – safe manner by Spring container and is used internally in a thread – safe manner as well. The implementations of UserIdRetriever and ProfileCompletenessCheckers are used in a thread – safe manner by DefaultCheckProfileCompletenessProcessor.

Thus, the component is thread-safe under these conditions.

2. Environment Requirements

2.1 Environment

Development language: Java

Compile Target: J2SE 1.6, J2EE 1.5

2.2 TopCoder Software Components

2.2.1 Generic

Logging Wrapper 2.0 – used for logging

Base Exception 2.0 – provides base classes for custom exceptions

2.2.2 Custom

User Profile and Audit Back End 1.0 – used to retrieve user information and make audit records

2.2.3 Existing codebase

Topcoder Web Module - https://coder.topcoder.com/tcs/internal/web_module/trunk

Topcoder Online Review - https://coder.topcoder.com/tcs/clients/cronos/applications/online_review/trunk/

Topcoder Direct - https://coder.topcoder.com/tcs/clients/cronos/applications/direct/trunk/

2.3 Third Party Components

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

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.topcoder.reg.profilecompleteness.filter

com.topcoder.reg.profilecompleteness.filter.impl

com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers

com.topcoder.reg.profilecompleteness.filter.impl.retrievers

3.2 Configuration Parameters

3.2.1 CheckProfileCompletenessFilter

Parameter	Description	Value
checkProfileCompletenessProcessor	CheckProfileCompletenessProcessor instance used to check the completeness of the profile	CheckProfileCompletenessProcessor. **Required.** Not null.
errorPageUri	The error page of the uri, if any exception occurred while processing the request.	String. **Required.** Not null, not empty.
log	Logger used to perform logging. If null, then logging should not be performed.	Log. Optional**.**
configurationFileName	The spring configuration file name. Note, that this is the FilterConfig configuration parameter	String. **Required.** Not null, not empty.

3.2.2 DefaultCheckProfileCompletenessProcessor

Parameter	Description	Value
matchUriPatterns	String uri patterns, used to check whether the given page should be verified for profile completeness.	List<String>. **Required.** Not null, not contains null/empty elements.
userIdRetriever	the UserIdRetriever instance, used to retrieve the user id.	UserIdRetriever. **Required.** Not null.
profileCompletenessCheckers	ProfileCompletenessChecker instances, used to verify profile completeness for the specific user instance	List<ProfileCompletenessChecker>. **Required.** Not null, not contains null elements.
completeDataURIs	Complete data uri strings, used to redirect user to the page, if the profile is incomplete	List<String>. **Required.** Not null, not contains null/empty elements.
userDAO	The user dao, used to retrieve the user by the id	UserDAO. **Required.** Not null.
requestURISessionKey	String key, under which the requestURI will be stored in http session, if the profile was not completed.	String. **Required.** Not null, not empty.
log	Logger used to perform logging. If null, then logging should not be performed.	Log. Optional**.**

3.2.3 ForumUserIdRetriever, OnlineReviewUserIdRetriever, StudioUserIdRetriever, TCSiteUserIdRetriever, JiraUserIdRetriever

Parameter	Description	Value
userIdSessionKey	String key, under which the user will be stored in http session.	String. **Required.** Not null, not empty.
log	Logger used to perform logging. If null, then logging should not be performed.	Log. Optional**.**

3.2.4 DirectUserIdRetriever, WikiUserIdRetriever

Parameter	Description	Value
log	Logger used to perform logging. If null, then logging should not be performed.	Log. Optional**.**

3.2.5 ForumProfileCompletenessChecker, DirectProfileCompletenessChecker, JiraProfileCompletenessChecker, VMProfileCompletenessChecker, CopilotProfileCompletenessChecker,WikiProfileCompletenessChecker, OnlineReviewProfileCompletenessChecker,SVNProfileCompletenessChecker, CompetitionProfileCompletenessChecker

Parameter	Description	Value
log	Logger used to perform logging. If null, then logging should not be performed.	Log. Optional**.**

Some user id retrievers require to state the session key for authentication token. Below is the defined keys as per current existing code:

ForumUserIdRetriever – "authToken"

OnlineReviewUserIdRetriever – "userId"

StudioUserIdRetrieve – "userid"

TCSiteUserIdRetriever – "userid"

JiraUserIdRetriever – "user"

Other user id retrievers will obtain the credentials by SessionPersistor.

3.3 Dependencies Configuration

See demo for configuration sample.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Extract the component distribution.

4.3 Demo

4.3.1 *web.xml file*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>Demo</display-name>
    <filter>
        <filter-name>completenessFilter1</filter-name>
        <filter-class>com.topcoder.reg.profilecompleteness.filter.
CheckProfileCompletenessFilter</filter-class>
         <init-param>
               <param-name>configurationFileName</param-name>
               <param-value>applicationContext1.xml</param-value>
         </ init-param >
    </filter>

    <filter>
        <filter-name>completenessFilter2</filter-name>
        <filter-class>com.topcoder.reg.profilecompleteness.filter.
CheckProfileCompletenessFilter</filter-class>
         <init-param>
               <param-name>configurationFileName</param-name>
               <param-value>applicationContext2.xml</param-value>
         </ init-param >
    </filter>

    <!—- Other filters with spring application contexts may go here -->

    <filter-mapping>
        <filter-name>completenessFilter1</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>completenessFilter2</filter-name>
        <url-pattern>/forums</url-pattern>
    </filter-mapping>



    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

4.3.2 *Spring configuration file*

Note, that following configuration application contexts should be stored in separate configuration files, as defined in 4.3.1. They are coupled in the below – mentioned example for simplicity.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
    "http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

  <!-- Create logger -->
  <bean id="log" class="com.topcoder.util.log.LogManager" factory-method="getLog">
    <constructor-arg value=" com.topcoder.reg.profilecompleteness.filter"/>
  </bean>

  <!-- Create user DAO -->
  <bean id="userDAO" class="com.topcoder.web.common.dao.impl.UserDAOImpl"/>

  <!-- Create user id retrievers -->
  <bean id="forumUserIdRetriever" class="
com.topcoder.reg.profilecompleteness.filter.impl.retrievers.ForumUserIdRetriever"
init-method="checkInitialization">
    <property name="userSessionKey" value="authToken"/>
    <property name="log" ref="log"/>
```

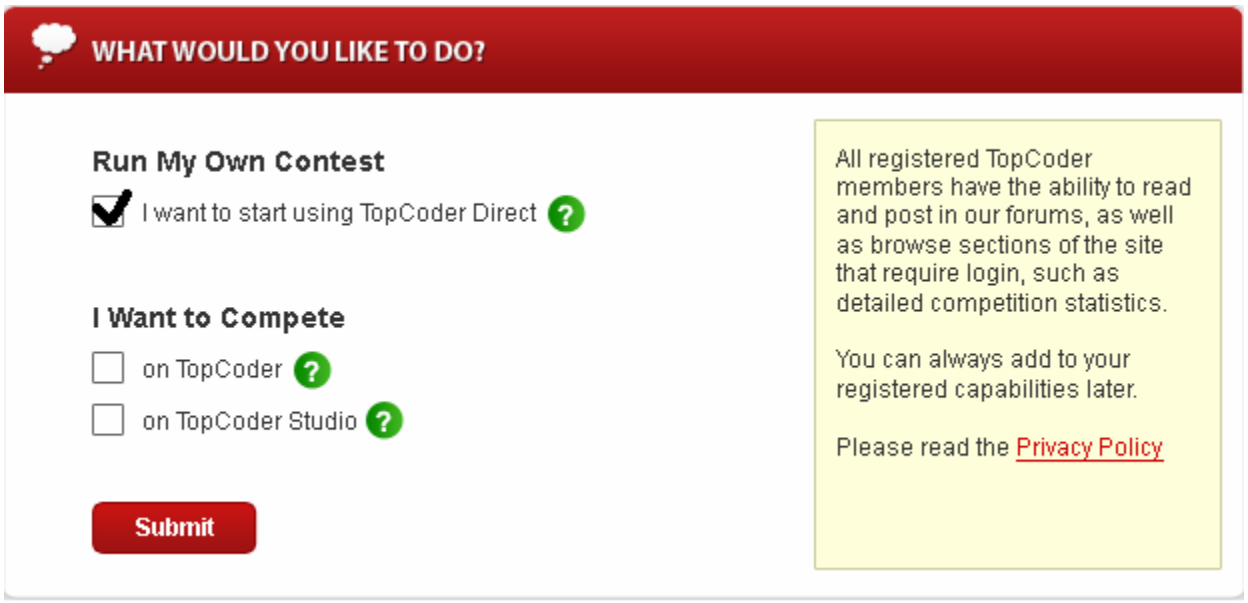
```
  </bean>

  <bean id="onlineReviewUserIdRetriever" class="
com.topcoder.reg.profilecompleteness.filter.impl.retrievers.OnlineReviewUserIdRetriever"
init-method="checkInitialization">
    <property name="userSessionKey" value="userId"/>
    <property name="log" ref="log"/>
  </bean>

  <bean id="studioUserIdRetriever" class="
com.topcoder.reg.profilecompleteness.filter.impl.retrievers.StudioUserIdRetriever"
init-method="checkInitialization">
    <property name="userSessionKey" value="userid"/>
    <property name="log" ref="log"/>
  </bean>

  <bean id="tcSiteUserIdRetriever" class="
com.topcoder.reg.profilecompleteness.filter.impl.retrievers.TCSiteUserIdRetriever"
init-method="checkInitialization">
    <property name="userSessionKey" value="userid"/>
    <property name="log" ref="log"/>
  </bean>

  <bean id="directUserIdRetriever" class="
com.topcoder.reg.profilecompleteness.filter.impl.retrievers.DirectUserIdRetriever"
init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>

  <bean id="wikiUserIdRetriever" class="
com.topcoder.reg.profilecompleteness.filter.impl.retrievers.WikiUserIdRetriever"
init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>

  <bean id="jiraIdRetriever" class="
com.topcoder.reg.profilecompleteness.filter.impl.retrievers.JiraUserIdRetriever"
init-method="checkInitialization">
    <property name="userSessionKey" value="user"/>
    <property name="log" ref="log"/>
  </bean>

  <!-- Create profile completeness checkers -->
  <bean id="ForumProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.ForumProfileCompletenes
sChecker"  init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>
  <bean id="DirectProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.DirectProfileCompletene
ssChecker"  init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>

  <bean id="JiraProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.JiraProfileCompleteness
Checker"  init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>

  <bean id="VMProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.VMProfileCompletenessCh
ecker"  init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>

  <bean id="CopilotProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.CopilotProfileCompleten
essChecker"  init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>
```

```
  <bean id="WikiProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.WikiProfileCompleteness
Checker"  init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>

  <bean id="OnlineReviewProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.OnlineReviewProfileComp
letenessChecker"  init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>

  <bean id="SVNProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.SVNProfileCompletenessC
hecker"  init-method="checkInitialization">
    <property name="log" ref="log"/>
  </bean>

  <bean id="CompetitionProfileCompletenessChecker" class="
com.topcoder.reg.profilecompleteness.filter.impl.completenesscheckers.
CompetitionProfileCompletenessChecker"  init-method="checkInitialization">
    <property name="log" ref="logger"/>
  </bean>

 <bean id="messageSource"
       class="org.springframework.context.support.ResourceBundleMessageSource">
              <property name="basenames">
                     <list>
                            <value>messages</value>
                     </list>
              </property>
       </bean>
</beans>
```

4.3.3 *Check profile completeness demo*

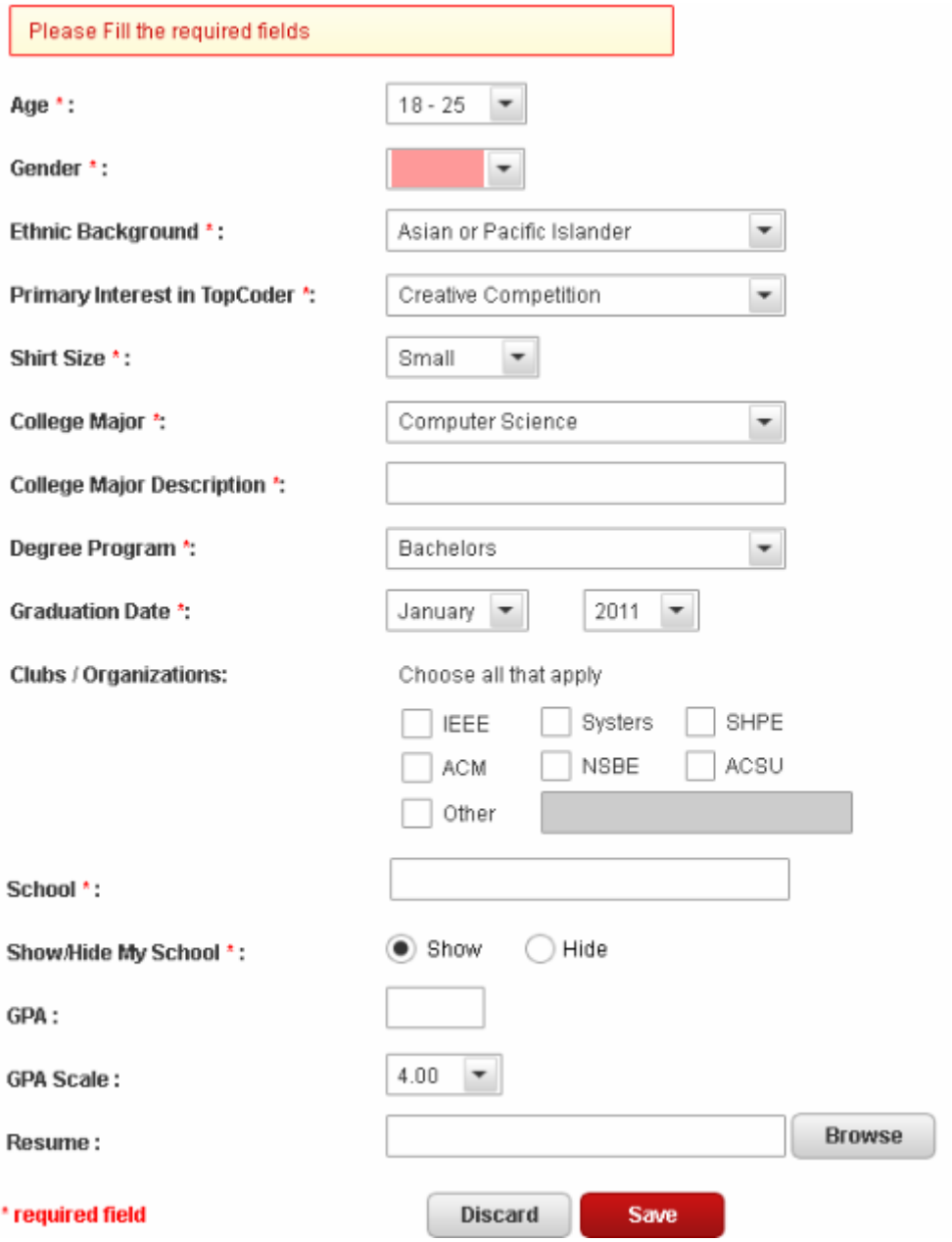
Following are not provided in the demo, they are just sketchs to help people understand this component.

Consider the user with id = 7 was registered at the topcoder web site. During the registration he was prompted to fill out only required fields first name/last name/email. Consider user also specified he is a customer and wanted to use the topcoder direct.



The user is trying to access the topcoder jira at the address https://apps.topcoder.com/bugs/ . The ProfileCompletenessCheckerFilter verifies, whether necessary data is filled. The registered user has

not provided/her his gender. The request uri is stored under configurable variable and the user is redirected to the configurable web – page to complete profile data.



The completion of the profile is handled by the Complete Profile Struts Actions component. After the successful completion of the data the user will be redirected to the stored web page http://apps.topcoder.com/bugs .

Other profile completeness check is done in similar manner.

5. Future Enhancements

None mentioned.