This page last changed on Aug 17, 2010 by hohosky.

Copilot Pool and Profile DAO Requirements Specification

Scope

Overview

TopCoder is revamping the current copilot selection process to build a more structural one. One big change is that we will bring in the idea of the copilot pool which is consisted of authorized pre-approved members; only members in the copilot pool will have access to the new copilot opportunities (the new copilot opportunities will be posted as copilot selection contests by client/PM). And each copilot will have copilot profile pages which show the statistics of his past copilot performance, such as number of projects managed, number of contests managed, the number of current working on projects and past project histories etc. The copilot profile will be used by the client/PM as a reference when choosing a copilot.
This component is responsible for providing the DAO layer for the copilot pool and profile module.

Version

1.0

Logic Requirements

The architecture TCUML will be provided, and this component is responsible for the classes/interfaces defined in the com.topcoder.direct.services.copilot.dao and com.topcoder.direct.services.copilot.model package.

GenericDAO

This interface defines the generic CRUD methods to manage the entity into the persistence. It contains the following methods:

- create - create a new record into persistence for the entity and return the generated entity id.
- update - update the entity back to the persistence. It should throw EntityNotFoundException if the entity doesn't exist in persistence.
- delete - delete the entity from the persistence by entity id. It should throw EntityNotFoundException if the entity doesn't exist in persistence.
- retrieve - retrieve the entity from the persistence by entity id. It should return null if the entity doesn't exist in persistence.
- retrieveAll - retrieve all the entities from the persistence. Return an empty list if there are no such entities.

CopilotProfileDAO

This interface simply extends the GenericDAO interface to manage the CopilotProfile entity.
All methods from GenericDAO should be implemented.
It contains the following additional method:

- getCopilotProfile - get copilot profile by the user id. Null should be returned if it doesn't exist.

CopilotProjectDAO

This interface simply extends the GenericDAO interface to manage the CopilotProject entity.
All methods from GenericDAO should be implemented.
It contains the following additional method:

- getCopilotProjects - get copilot projects associated with the given copilot-profile-id. Return an empty list if there are no associated copilot projects.

CopilotProjectPlanDAO

This interface simply extends the GenericDAO interface to manage the CopilotProjectPlan entity.
All methods from GenericDAO should be implemented.
It contains the following additional method:

- getCopilotProjectPlan - get copilot project plan associated with the given copilot-project-id. Return null if there is no associated copilot project plan.

LookupDAO

This interface defines the following 3 methods:

- getAllCopilotProfileStatuses - get all the CopilotProfileStatus entities from persistence.
- getAllCopilotProjectStatuses - get all the CopilotProjectStatus entities from persistence.
- getAllCopilotTypes - get all the CopilotType entities from persistence

UtilityDAO

An implementation should be provided to query the database tables to get the desired values (Native Hibernate Query should be used here).
1. getContestBugCount

This method is used to get the number of bugs for the given contest (software contest only for now) from the JIRA database directly.
The bugs are stored in the jiraissue database table as described here.
The software contest bugs will have a "ProjectID" custom field defined in the customfield database table (in its cfname column). And the associated software contest id for the bug is stored in the customfieldvalue database table. The database association is described here.

2. getContestLatestBugResolutionDate

This method is used to get the latest resolution date of software contest's bugs.
The resolution date will be stored into the "Resolution Date" custom field of the software contest bug.

3. getCopilotEarnings

This method is used to get the copilot earnings from the payment & user_payment db tables defined in topcoder_dw database.
The SQL is like: select sum(gross_amount) from payment p, user_payment up where p.payment_id = up.payment_id and p.payment_type_id = ? and up.user_id = ?
The paymentTypeId should be configurable (for the Copilot), and the userId will be passed in.

4. getCopilotProjectContests

This method is used to get the ids of online review contests associated with the given copilot and tc-direct-project.
The online review contests will have copilot resource role for the given copilot.

Hibernate Implementations

Hibernate implementations should be provided for all the DAO interfaces described above.

The Hibernate session factory should be injected by the Spring as explained in this link, and the SessionFactory.getCurrentSession() should be used to retrieve the session in order to use the Spring managed transaction.
Designer is also responsible for providing the Hibernate ORM files for the entities in the model package.

Entities

Here gives a brief description of the involved entities:

- CopilotProfile - represents the profile for a copilot
- CopilotProfileInfo - represents the additional profile information
- CopilotProfileInfoType - represents the type of the additional profile information
- CopilotProfileStatus - represents the copilot profile status

- CopilotProject - represents the copilot project
- CopilotProjectInfo - represents the additional project information
- CopilotProjectInfoType - represents the type of the additional project information
- CopilotProjectStatus - represents the copilot project status
- CopilotType - represents the copilot types

- CopilotProjectPlan - represents the plan for a copilot project
- PlannedContest - represents the planned contests for a copilot project

Configuration

Spring framework should be used to load configuration.

Transaction

Any method that modifies (creates, deletes or updates) the persistence data should be transactional.
Spring declarative transaction should be used, and the designer should use the @Transactional annotation for the transactional methods.

Logging

The method entry and exit need to be logged with DEBUG level, and the exception needs to be logged with ERROR level.
The TCS Logging Wrapper should be used to do the logging.

Exception

A base exception (CopilotDAOException) is already provided for this component, and the EntityNotFoundException mentioned in GenericDAO requirement is also provided. Designers are free to add any custom exception extending the base exception.

Auditing

The database tables contain audit fields (create_user, create_date, modify_user and modify_date columns), they need to be updated when inserting and updating the data record.

Thread-safety

This component should work thread-safely.

Required Algorithms

None

Example of the Software Usage

This component provides the DAO implementations for the copilot pool and profile module.

Future Component Direction

None

Interface Requirements

Graphical User Interface Requirements

None

External Interfaces

None

Environment Requirements

- Development language: Java1.5
- Compile target: Java1.5
- JBoss 4.0.2
- Informix 11.0

Package Structure

com.topcoder.direct.services.copilot.dao
com.topcoder.direct.services.copilot.model

Software Requirements

Administration Requirements

What elements of the application need to be configurable?

- Copilot Payment Type Id
- Hibernate Session Factory

Technical Constraints

Are there particular frameworks or standards that are required?

None

TopCoder Software Component Dependencies:

Logging Wrapper 2.0
Base Exception 2.0

Third Party Component, Library, or Product Dependencies:

Spring 2.5.6
Hibernate 3.5

QA Environment:

- Java 1.5
- JBoss 4.0.2
- Informix 11.0

Design Constraints

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

Required Documentation

Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

Help / User Documentation

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