/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Licensed to the Apache Software Foundation (ASF) under one

\* or more contributor license agreements. See the NOTICE file

\* distributed with this work for additional information

\* regarding copyright ownership. The ASF licenses this file

\* to you under the Apache License, Version 2.0 (the

\* "License"); you may not use this file except in compliance

\* with the License. You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing,

\* software distributed under the License is distributed on an

\* "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

\* KIND, either express or implied. See the License for the

\* specific language governing permissions and limitations

\* under the License.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

package org.apache.ofbiz.scrum;

/\*

\* https://www.versionone.com/scrum-project-management/ c'è una spiegazione sugli scrum

\* Scrum è un modo per fare progetti in modo agile.

\* In pratica il progetto va avanti per sprint in cui si sviluppa una funzionalità del

\* progetto alla volta. Alla fine di questo sprint si fa una sprint review in cui si

\* dicono le funzionalità aggiunte. Tre dei nostri metodi sono

\* retrieveMissingScrumRevision

\* removeDuplicateScrumRevision

\* viewScrumRevision

\* Per me revision alla fine sono le review. Io ho guardato

\* RemoveDuplicateScrumRevision e in pratica fa una query

\* (in cui però le condizioni non sono molto chiare) e poi elimina i duplicati

\* \*/

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.HashMap;

import java.util.HashSet;

import java.util.LinkedList;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Set;

import org.apache.ofbiz.base.util.Debug;

import org.apache.ofbiz.base.util.UtilMisc;

import org.apache.ofbiz.base.util.UtilProperties;

import org.apache.ofbiz.base.util.UtilValidate;

import org.apache.ofbiz.entity.Delegator;

import org.apache.ofbiz.entity.GenericEntityException;

import org.apache.ofbiz.entity.GenericValue;

import org.apache.ofbiz.entity.condition.EntityCondition;

import org.apache.ofbiz.entity.condition.EntityOperator;

import org.apache.ofbiz.entity.util.EntityQuery;

import org.apache.ofbiz.service.DispatchContext;

import org.apache.ofbiz.service.GenericServiceException;

import org.apache.ofbiz.service.LocalDispatcher;

import org.apache.ofbiz.service.ServiceUtil;

/\*\*

\* Scrum Services

\*/

public class ScrumServices {

public static final String module = ScrumServices.class.getName();

public static final String resource = "scrumUiLabels";

public static Map<String, Object> linkToProduct(DispatchContext ctx, Map<String, ? extends Object> context) {

Delegator delegator = ctx.getDelegator();

Locale locale = (Locale)context.get("locale");

LocalDispatcher dispatcher = ctx.getDispatcher();

String communicationEventId = (String) context.get("communicationEventId");

// Debug.logInfo("==== Processing Commevent: " + communicationEventId, module);

if (UtilValidate.isNotEmpty(communicationEventId)) {

try {

GenericValue communicationEvent = EntityQuery.use(delegator).from("CommunicationEvent").where("communicationEventId", communicationEventId).queryOne();

if (UtilValidate.isNotEmpty(communicationEvent)) {

String subject = communicationEvent.getString("subject");

if (UtilValidate.isNotEmpty(subject)) {

int pdLocation = subject.indexOf("PD#");

if (pdLocation > 0) {

// scan until the first non digit character

int nonDigitLocation = pdLocation + 3;

while (nonDigitLocation < subject.length() && Character.isDigit(subject.charAt(nonDigitLocation))) {

nonDigitLocation++;

}

String productId = subject.substring(pdLocation + 3, nonDigitLocation);

// Debug.logInfo("=======================Product id found in subject: >>" + custRequestId + "<<", module);

GenericValue product = EntityQuery.use(delegator).from("Product").where("productId", productId).queryOne();

if (product != null) {

GenericValue communicationEventProductMap = EntityQuery.use(delegator).from("CommunicationEventProduct").where("productId", productId, "communicationEventId", communicationEventId).queryOne();

if (UtilValidate.isEmpty(communicationEventProductMap)) {

GenericValue communicationEventProduct = delegator.makeValue("CommunicationEventProduct", UtilMisc.toMap("productId", productId, "communicationEventId", communicationEventId));

communicationEventProduct.create();

}

try {

GenericValue productRoleMap = EntityQuery.use(delegator).from("ProductRole").where("productId",productId, "partyId", communicationEvent.getString("partyIdFrom"), "roleTypeId","PRODUCT\_OWNER").queryFirst();

GenericValue userLogin = (GenericValue) context.get("userLogin");

// also close the incoming communication event

if (UtilValidate.isNotEmpty(productRoleMap)) {

dispatcher.runSync("setCommunicationEventStatus", UtilMisc.<String, Object>toMap("communicationEventId", communicationEvent.getString("communicationEventId"), "statusId", "COM\_COMPLETE", "userLogin", userLogin));

}

} catch (GenericServiceException e1) {

Debug.logError(e1, "Error calling updating commevent status", module);

return ServiceUtil.returnError(UtilProperties.getMessage(resource, "ScrumErrorCallingUpdatingCommeventStatus", locale) + e1.toString());

}

} else {

Debug.logInfo("Product id " + productId + " found in subject but not in database", module);

}

}

}

}

} catch (GenericEntityException e) {

return ServiceUtil.returnError(UtilProperties.getMessage(resource, "ScrumFindByPrimaryKeyError", locale) + e.toString());

}

Map<String, Object> result = ServiceUtil.returnSuccess();

return result;

} else {

Map<String, Object> result = ServiceUtil.returnError(UtilProperties.getMessage(resource, "ScrumCommunicationEventIdRequired", locale));

return result;

}

}

/\*\*

\* viewScrumRevision

\* <p>

\* Use for view Scrum Revision

\*

\* @param ctx The DispatchContext that this service is operating in

\* @param context Map containing the input parameters

\* @return Map with the result of the service, the output parameters.

\*/

public static Map<String, Object> viewScrumRevision(DispatchContext ctx, Map<String, ? extends Object> context) {

Delegator delegator = ctx.getDelegator();

LocalDispatcher dispatcher = ctx.getDispatcher();

String revision = (String) context.get("revision");

String repository = (String) context.get("repository");

Map<String, Object> result = ServiceUtil.returnSuccess();

StringBuilder logMessage = new StringBuilder();

StringBuilder diffMessage = new StringBuilder();

try {

if (UtilValidate.isNotEmpty(repository) && UtilValidate.isNotEmpty(revision)) {

String logline = null;

String logCommand = "svn log -r" + revision + " " + repository;

Process logProcess = Runtime.getRuntime().exec(logCommand);

BufferedReader logIn = new BufferedReader(new InputStreamReader(logProcess.getInputStream()));

while ((logline = logIn.readLine()) != null) {

logMessage.append(logline).append("\n");

}

String diffline = null;

String diffCommand = "svn diff -r" + Integer.toString((Integer.parseInt(revision.trim()) - 1)) + ":" + revision + " " + repository;

Process diffProcess = Runtime.getRuntime().exec(diffCommand);

BufferedReader diffIn = new BufferedReader(new InputStreamReader(diffProcess.getInputStream()));

while ((diffline = diffIn.readLine()) != null) {

diffMessage.append(diffline).append("\n");

}

}

result.put("revision", revision);

result.put("repository", repository);

result.put("logMessage", logMessage.toString());

result.put("diffMessage", diffMessage.toString());

} catch (IOException e) {

e.printStackTrace();

return ServiceUtil.returnError(e.getMessage());

}

return result;

}

/\*\*

\* retrieveMissingScrumRevision

\* <p>

\* Use for retrieve the missing data of the Revision

\*

\* @param ctx The DispatchContext that this service is operating in

\* @param context Map containing the input parameters

\* @return Map with the result of the service, the output parameters.

\*/

public static Map<String, Object> retrieveMissingScrumRevision(DispatchContext ctx, Map<String, ? extends Object> context) {

//prende dal dispatch contesto il delegator che usa per fare le query e il dispatcher per far partire un altro servizio alla fine

Delegator delegator = ctx.getDelegator();

LocalDispatcher dispatcher = ctx.getDispatcher();

//ottiene dal contesto

//userLogin:

//latestRevision: ha il numero di revision che devono essere trovate

//repositoryRoot: serve per trovare il revision link

GenericValue userLogin = (GenericValue) context.get("userLogin");

String latestRevision = (String) context.get("latestRevision");

String repositoryRoot = (String) context.get("repositoryRoot");

//crea result che è quello che ritorna quando il metodo va a buon fine

Map<String, Object> result = ServiceUtil.returnSuccess();

try {

//controlla che le due stringhe prese prima dai parametri non siano null

if (UtilValidate.isNotEmpty(repositoryRoot) && UtilValidate.isNotEmpty(latestRevision))

{

//prende latest revision, togli gli spazi in eccesso con trim e poi lo trasforma in intero. E' il numero di revision che deve trovare

Integer revision = Integer.parseInt(latestRevision.trim());

//cicla tante volte quante il numero di revision da trovare

for (int i = 1; i <= revision; i++)

{

String logline = null;

List<String> logMessageList = new LinkedList<String>();

String logCommand = "svn log -r" + i + " " + repositoryRoot;

//crea questo processo eseguendo il comando appena definito

Process logProcess = Runtime.getRuntime().exec(logCommand);

//crea questo buffered reader che ha preso l'input stream dal processo appena creato

BufferedReader logIn = new BufferedReader(new InputStreamReader(logProcess.getInputStream()));

//continua a leggere dal reader finchè non è vuoto. Quello che legge lo salva in logMessageList come stringa togliendo gli spazi in eccesso

while ((logline = logIn.readLine()) != null) {

logMessageList.add(logline.toString().trim());

}

//controlla se ci sono stringhe nella lista appena fatta

if (UtilValidate.isNotEmpty(logMessageList)) {

//qui ottiene le informazioni sull'utente e sul task

String userInfo = logMessageList.get(1).replace(" | ", ",");

String taskInfo = logMessageList.get(3);

//------------------------------------------------------------------------------------------------------------------ get user information

String[] versionInfoTemp = userInfo.split(",");

String user = versionInfoTemp[1];

//------------------------------------------------------------------------------------------------------------------ get task information

String taskId = null;

//cicla su taskInfo e cerca una sequenza di 5 numeri consecutivi che costituiscono l'id che viene salvato

char[] taskInfoList = taskInfo.toCharArray();

int count = 0;

for (int j = 0; j < taskInfoList.length; j++) {

if (Character.isDigit(taskInfoList[j])) {

count = count + 1;

} else {

count = 0;

}

if (count == 5) {

taskId = taskInfo.substring(j - 4, j + 1);

}

}

//prende il link in cui c'è la revision

String revisionLink = repositoryRoot.substring(repositoryRoot.lastIndexOf("svn/") + 4, repositoryRoot.length()) + "&revision=" + i;

//stampa il link trovato sul log

Debug.logInfo("Revision Link ============== >>>>>>>>>>> "+ revisionLink, module);

//controlla che prima abbia trovato un id

if (UtilValidate.isNotEmpty(taskId)) {

//crea una stringa con la versione corrente che usa nella query

String version = "R" + i;

//query su WorkEffortAndContentDataResource

//Work Effort

//A Work Effort can be one of many things including a task, project, project phase, to-do item,

//calendar item, or even a Workflow Activity.

List <GenericValue> workeffContentList = EntityQuery.use(delegator).from("WorkEffortAndContentDataResource").where("contentName",version.trim() ,"drObjectInfo", revisionLink.trim()).queryList();

//crea questa lista di condizioni che usa nella query successiva (ci sono anche condizioni in OR)

List<EntityCondition> exprsAnd = new LinkedList<EntityCondition>();

exprsAnd.add(EntityCondition.makeCondition("workEffortId", EntityOperator.EQUALS, taskId));

List<EntityCondition> exprsOr = new LinkedList<EntityCondition>();

exprsOr.add(EntityCondition.makeCondition("workEffortTypeId", EntityOperator.EQUALS, "SCRUM\_TASK\_ERROR"));

exprsOr.add(EntityCondition.makeCondition("workEffortTypeId", EntityOperator.EQUALS, "SCRUM\_TASK\_TEST"));

exprsOr.add(EntityCondition.makeCondition("workEffortTypeId", EntityOperator.EQUALS, "SCRUM\_TASK\_IMPL"));

exprsOr.add(EntityCondition.makeCondition("workEffortTypeId", EntityOperator.EQUALS, "SCRUM\_TASK\_INST"));

exprsAnd.add(EntityCondition.makeCondition(exprsOr, EntityOperator.OR));

//nuova query da WorkEffort in cui mette tutte le condizioni di prima

List<GenericValue> workEffortList = EntityQuery.use(delegator).from("WorkEffort").where(exprsAnd).queryList();

//controlla che i risultati delle due query siano diversi da null

if (UtilValidate.isEmpty(workeffContentList) && UtilValidate.isNotEmpty(workEffortList)) {

//crea una mappa stringa-oggetto in cui mette diverse cose

//Le due query vengono usate solo in questo punto per controllare che siano non null

Map<String, Object> inputMap = new HashMap<String, Object>();

inputMap.put("taskId", taskId);

inputMap.put("user", user);

inputMap.put("revisionNumber", Integer.toString(i));

inputMap.put("revisionLink", revisionLink);

inputMap.put("revisionDescription", taskInfo);

inputMap.put("userLogin", userLogin);

//stampa inputMap sul log

Debug.logInfo("inputMap ============== >>>>>>>>>>> "+ inputMap, module);

//da la map al dispatcher che fa partire il servizio "updateScrumRevision" dando come parametro inputMap

//in inputMap ci sono le informazioni sulle missing Scrum Revision

dispatcher.runSync("updateScrumRevision", inputMap);

}

}

}

}

}

//cattura tutte le eccezioni, stampa il messaggio dell'eccezione e ritorna error

} catch (IOException e) {

e.printStackTrace();

return ServiceUtil.returnError(e.getMessage());

} catch (GenericEntityException entityEx) {

entityEx.printStackTrace();

return ServiceUtil.returnError(entityEx.getMessage());

} catch (GenericServiceException serviceEx) {

serviceEx.printStackTrace();

return ServiceUtil.returnError(serviceEx.getMessage());

}

return result;

}

/\*\*

\* removeDuplicateScrumRevision

\* <p>

\* Use for remove duplicate scrum revision

\*

\* @param ctx The DispatchContext that this service is operating in

\* @param context Map containing the input parameters

\* @return Map with the result of the service.

\*/

public static Map<String, Object> removeDuplicateScrumRevision(DispatchContext ctx, Map<String, ? extends Object> context)

{

//delegator è un interfaccia che usa solo quando deve fare delle query

//il dispatcher non viene mai usato

Delegator delegator = ctx.getDelegator();

LocalDispatcher dispatcher = ctx.getDispatcher();

//prende dalla map un oggetto che è una string e ha come chiave repository root. Viene usata solo sotto per calcolare revision link che è una stringa usata nella query

String repositoryRoot = (String) context.get("repositoryRoot");

//crea quest result che in pratica viene restituito dalla funzione e in questo caso indica che la funzione è andata a buon fine. Se ci sono eccezioni entra nel catch e ritorna errore

Map<String, Object> result = ServiceUtil.returnSuccess();

try {

//qui crea questa nuova lista di entityConditions che sono le condizioni della query che farà

List<EntityCondition> exprsAnd = new LinkedList<EntityCondition>();

//prende la stringa che a preso sopra dalla mappa(repositoryRoot) e prende la parte dopo svn/ fino alla fine poi la concatena con &revision=

String revisionLink = repositoryRoot.substring(repositoryRoot.lastIndexOf("svn/") + 4, repositoryRoot.length()) + "&revision=";

//aggiunge queste tre condizioni alla lista delle condizioni creata sopra

exprsAnd.add(EntityCondition.makeCondition("workEffortContentTypeId", EntityOperator.EQUALS, "TASK\_SUB\_INFO"));

exprsAnd.add(EntityCondition.makeCondition("contentTypeId", EntityOperator.EQUALS, "DOCUMENT"));

exprsAnd.add(EntityCondition.makeCondition("drObjectInfo", EntityOperator.LIKE, revisionLink + "%"));

//crea questa lista di GenericValue che è il risultato della query che viene fatta dal delegator dalla tabella "workEffort..." in cui ci sono le condizioni messe sopra

List<GenericValue> workEffortDataResourceList = EntityQuery.use(delegator).from("WorkEffortAndContentDataResource").where(exprsAnd).queryList();

//controlla che il risultato della query sia non vuoto

if (UtilValidate.isNotEmpty(workEffortDataResourceList))

{

//stampa la dimensione della query. module è il nome della classe

Debug.logInfo("Total Content Size ============== >>>>>>>>>>> "+ workEffortDataResourceList.size(), module);

//crea un insieme di chiavi in cui metterà i valori non duplicati(keys) e una in cui mette i duplicati da cancellare (exclusions)

Set<String> keys = new HashSet<String>();

Set<GenericValue> exclusions = new HashSet<GenericValue>();

//for che cicla su ogni valore nel risultato della query

for (GenericValue workEffort : workEffortDataResourceList)

{

//qui aggiunge tutti gli elementi a keys mentre i duplicati li mette in exclusions.

String drObjectInfo = workEffort.getString("drObjectInfo");

if (keys.contains(drObjectInfo)) {

exclusions.add(workEffort);

} else {

keys.add(drObjectInfo);

}

}

//----------------------------------------------------------------------------------------remove the duplicate entry

//stampa la dimensione delle cose che rimuove

Debug.logInfo("Remove size ============== >>>>>>>>>>> "+ exclusions.size(), module);

//controlla se deve rimuovere qualcosa

if (UtilValidate.isNotEmpty(exclusions)) {

//cicla sugli elementi da rimuovere

for (GenericValue contentResourceMap : exclusions)

{

//stampa id dell'elemento da rimuovere

Debug.logInfo("Remove contentId ============== >>>>>>>>>>> "+ contentResourceMap.getString("contentId"), module);

//cerca in dataResource e in content gli elementi con id uguale a quello da eliminare

GenericValue dataResourceMap = EntityQuery.use(delegator).from("DataResource").where("dataResourceId", contentResourceMap.getString("dataResourceId")).queryOne();

GenericValue contentMap = EntityQuery.use(delegator).from("Content").where("contentId", contentResourceMap.getString("contentId")).queryOne();

//rimuove questi due elementi

contentMap.removeRelated("WorkEffortContent");

contentMap.removeRelated("ContentRole");

contentMap.remove();

dataResourceMap.removeRelated("DataResourceRole");

dataResourceMap.remove();

}

}

}

}

//nulla di strano se ci sono le eccezioni la cattura stampa qualcosa

catch (GenericEntityException entityEx) {

entityEx.printStackTrace();

//ritorna un errore chiamando un metodo statico a cui passa il messaggio contenuto nell'eccezione

return ServiceUtil.returnError(entityEx.getMessage());

}

//alla fine ritorna il risultato

return result;

}

}