**Task-13 CODE**

QUESTION

1. Create a scheduler. When I provide any cron expression for 3 minutes, after 3 minutes, whatever page has the Expirydate property with the current date and time should be published, and those with previous dates and times should be unpublished.

**Answer:**

**BY USING SCHEDULER PUBLISHING AND UNPUBLISHING THE PAGE**

**package** com.adobe.aem.guides.demo.core.schedulers;

**import** org.osgi.service.metatype.annotations.AttributeDefinition;

**import** org.osgi.service.metatype.annotations.AttributeType;

**import** org.osgi.service.metatype.annotations.ObjectClassDefinition;

@ObjectClassDefinition(name = "PreviousDatePublish",description = "scheduler is created for publish")

**public** **@interface** PresentDatePublish {

@AttributeDefinition(

name="service name",

type = AttributeType.***STRING***,

description = "enter the service name"

)

**public** String getservice\_name() **default** "practise";

@AttributeDefinition

(

name="can run concurrently",

type=AttributeType.***BOOLEAN***,

description="can run concurrently"

)

**public** **boolean** canrunconcurrently() **default** **false**;

@AttributeDefinition

(

name="Enabled scheduler",

type=AttributeType.***BOOLEAN***,

description="Enable the scheduler"

)

**public** **boolean** Enabledscheduler() **default** **true**;

@AttributeDefinition(

name="Expression",

type = AttributeType.***STRING***,

description = "enter the Expression"

)

**public** String getExpressi();

}

**SECONDSCHEDULER**

**package** com.adobe.aem.guides.demo.core.schedulers;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.Map;

**import** javax.jcr.Node;

**import** javax.jcr.RepositoryException;

**import** javax.jcr.Session;

**import** javax.jcr.query.Query;

**import** javax.jcr.query.QueryManager;

**import** javax.jcr.query.QueryResult;

**import** org.apache.sling.api.resource.LoginException;

**import** org.apache.sling.api.resource.ResourceResolver;

**import** org.apache.sling.api.resource.ResourceResolverFactory;

**import** org.apache.sling.commons.scheduler.ScheduleOptions;

**import** org.apache.sling.commons.scheduler.Scheduler;

**import** org.osgi.service.component.annotations.Activate;

**import** org.osgi.service.component.annotations.Component;

**import** org.osgi.service.component.annotations.Deactivate;

**import** org.osgi.service.component.annotations.Modified;

**import** org.osgi.service.component.annotations.Reference;

**import** org.osgi.service.metatype.annotations.Designate;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** com.day.cq.replication.ReplicationActionType;

**import** com.day.cq.replication.ReplicationException;

**import** com.day.cq.replication.Replicator;

**import** com.day.cq.wcm.api.Page;

**import** com.day.cq.wcm.api.PageManager;

@Designate(ocd = PresentDatePublish.**class**)

@Component(service = Runnable.**class**, immediate = **true**)

**public** **class** PresentDatePublishsecond **implements** Runnable {

**private** **static** **final** Logger ***log*** = LoggerFactory.*getLogger*(PresentDatePublishsecond.**class**);

@Reference

**private** Scheduler scheduler;

@Reference

**private** ResourceResolverFactory resourceResolverFactory;

@Reference

**private** Replicator replicator;

**private** **static** **final** String ***SERVICE\_USER*** = "hemanth";

**private** String cronExpression;

@Modified

**public** **void** modify(PresentDatePublish sch) {

**this**.cronExpression = sch.getExpressi();

addscheduler(sch); // Re-schedule with the new cron expression

}

@Activate

**public** **void** activation(PresentDatePublish sch) {

**this**.cronExpression = sch.getExpressi();

addscheduler(sch);

}

**public** **void** addscheduler(PresentDatePublish sch) {

***log***.info("Scheduler is created");

**if** (sch.Enabledscheduler()) {

ScheduleOptions scheduleOptions = scheduler.EXPR(cronExpression);

scheduleOptions.canRunConcurrently(sch.canrunconcurrently());

scheduleOptions.name(sch.getservice\_name());

scheduler.schedule(**this**, scheduleOptions);

}

}

@Deactivate

**public** **void** deactivate(PresentDatePublish sch) {

removescheduler(sch);

}

**public** **void** removescheduler(PresentDatePublish sch) {

***log***.info("Job is unscheduled");

scheduler.unschedule(sch.getservice\_name());

}

@Override

**public** **void** run() {

***log***.info("Scheduler is running in present date");

***log***.info("My cron expression in present date: " + cronExpression);

ResourceResolver resourceResolver = **null**;

**try** {

resourceResolver = getServiceResourceResolver();

**if** (resourceResolver != **null**) {

handlePageReplication(resourceResolver);

}

} **catch** (LoginException e) {

***log***.error("Failed to get service resource resolver", e);

} **catch** (ReplicationException e) {

***log***.error("ReplicationException occurred", e);

} **finally** {

**if** (resourceResolver != **null**) {

resourceResolver.close();

}

}

}

**private** ResourceResolver getServiceResourceResolver() **throws** LoginException {

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

param.put(ResourceResolverFactory.***SUBSERVICE***, ***SERVICE\_USER***);

**return** resourceResolverFactory.getServiceResourceResolver(param);

}

**private** **void** handlePageReplication(ResourceResolver resourceResolver) **throws** ReplicationException {

**try** {

Session session = resourceResolver.adaptTo(Session.**class**);

**if** (session == **null**) {

***log***.error("Could not obtain a JCR session.");

**return**;

}

String queryString = "SELECT \* FROM [cq:PageContent] AS content WHERE ISDESCENDANTNODE(content, '/content/Demo/us/en') AND content.[expirydate] IS NOT NULL";

QueryManager queryManager = session.getWorkspace().getQueryManager();

Query query = queryManager.createQuery(queryString, Query.***JCR\_SQL2***);

QueryResult result = query.execute();

javax.jcr.NodeIterator nodes = result.getNodes();

SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSXXX");

String currentDateStr = sdf.format(**new** Date());

**while** (nodes.hasNext()) {

Node node = nodes.nextNode();

**if** (node.hasProperty("expirydate")) {

String expiryDate = node.getProperty("expirydate").getString();

PageManager pageManager = resourceResolver.adaptTo(PageManager.**class**);

Page page = pageManager.getContainingPage(node.getPath());

**if** (page != **null**) {

**if** (expiryDate.startsWith(currentDateStr.substring(0, 10))) {

// Publish the page

replicator.replicate(session, ReplicationActionType.***ACTIVATE***, page.getPath());

***log***.info("Published page: " + page.getPath());

} **else** **if** (expiryDate.compareTo(currentDateStr) < 0) {

// Unpublish the page

replicator.replicate(session, ReplicationActionType.***DEACTIVATE***, page.getPath());

***log***.info("Unpublished page: " + page.getPath());

}

}

}

}

} **catch** (RepositoryException e) {

***log***.error("Error while querying and replicating pages", e);

}

}

}



