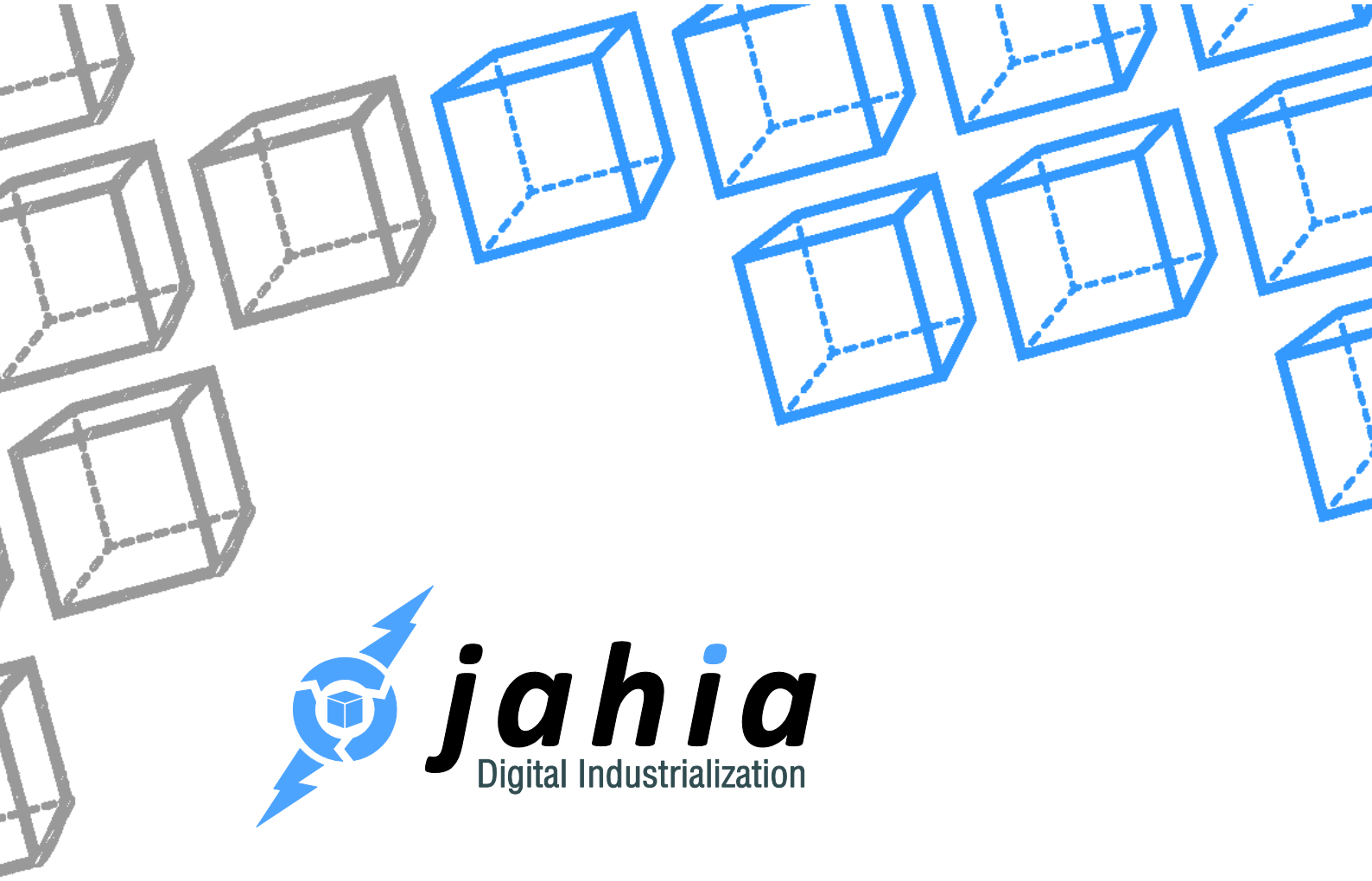


jahia
Digital Industrialization

Form Factory
Extending Actions

Rooted in Open Source CMS, Jahia's Digital Industrialization paradigm is about streamlining Enterprise digital projects across channels to truly control time-to-market and TCO, project after project.

Jahia Solutions Group SA

9 route des Jeunes,
CH-1227 Les acacias
Geneva, Switzerland

http://www.jahia.com

Summary

1 Introduction

The Form Center module allows users to create forms directly from their Digital Factory platform, letting them choose among input fields, validations and form actions. There are several ways to extend this module (fields, validations and actions). This document is a step-by-step guide on how to extend actions (definition, saving results, fetching and exploiting results).

1.1 The Concept of Action

An action is defined by:

- Its JCR definition
- Its view in the 'form creation' section
- The Java class implementing the action
- The API that fetches and returns the results saved by the action
- The views exploiting results and associated with this API

1.2 Purpose of This Document

This document describes all the steps required for the creation of a module defining an action that will save the results of a form to a MongoDB database, as well as the implementation of a JAX-RS

API that will return the results. Connection to the MongoDB database will be performed through the Database Connector module. Since the aim of this document is to explain in detail the method used to add an action to the Form Center, the code used will be limited to a basic backup of the JCR nodes to mongoDB—these data will not be fully usable independently from the JCR. Likewise, we will not cover code pertaining to the Spring Data framework or MongoDB.

Reference Documentation for MongoDB	http://docs.mongodb.org/manual/
Reference Documentation for the Spring Data MongoDB module, version 1.4.2	http://docs.spring.io/spring-data/data-mongo/docs/1.4.2.RELEASE/reference/html/

2 Creating the Module

The action for saving to a MongoDB database will be defined in a module created with Maven.

From a command line console, run the following command:

```
$: mvn archetype:generate -DarchetypeCatalog=https://devtools.jahia.com/nexus/content/repositories/jahia-snapshots
```

This command launches the creation of Digital Factory 7 components via Maven. The following options should be displayed:

```
Choose archetype:
1: https://devtools.jahia.com/nexus/content/repositories/jahia-
snapshots -> org.jahia.archetypes:jahia-templatesSet-archetype
(Archetype for creating a new template set project to be run on a
Digital Factory server)
2: https://devtools.jahia.com/nexus/content/repositories/jahia-
snapshots -> org.jahia.archetypes:jahia-profileModule-archetype
(Archetype for creating a new profile module project to be run on a
Digital Factory server)
3: https://devtools.jahia.com/nexus/content/repositories/jahia-
snapshots -> org.jahia.archetypes:jahia-module-archetype (Archetype
for creating a new module project to be run on a Digital Factory
server)
4: https://devtools.jahia.com/nexus/content/repositories/jahia-
snapshots -> org.jahia.archetypes:jahia-app-archetype (Archetype for
creating a new JahiApp module project to be run on a Digital Factory
server)
5: https://devtools.jahia.com/nexus/content/repositories/jahia-
snapshots -> org.jahia.archetypes:jahia-migration-archetype (This is
an archetype to generate a migration module)
Choose a number or apply filter (format: [groupId:]artifactId, case
sensitive contains): :
```

Since we want to create a module, we are going to choose number ‘3’ for *jahia-module-archetype*.

This triggers the generation of a standard module. The next screen asks us to validate the default values for the properties:

```
Confirm properties configuration:
groupId: org.jahia.modules
artifactId: sample
version: 1.0-SNAPSHOT
package: org.jahia.modules
digitalFactoryVersion: 7.0.0.0-SNAPSHOT
moduleName: Sample Module
 Y: :
```

We will need to redefine some of these values for our module.
Reply 'N'.

The interface will now display each property along with its default value. To keep the default value, simply validate with the Enter key. To define a new value, type it in and then validate with the Enter key.

Enter the following values:

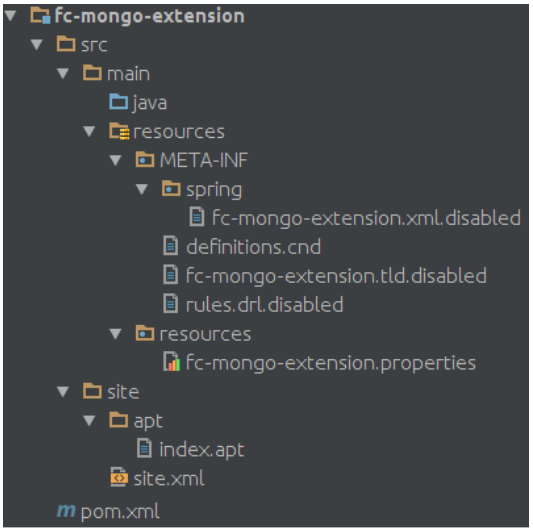
Property	Default Value	Final Value
groupId	org.jahia.modules	org.jahia.modules
artifactId	sample	fc-mongo-extension
version	1.0-SNAPSHOT	1.0-SNAPSHOT
package	org.jahia.module	org.jahia.module
digitalFactoryVersion	7.0.0.0-SNAPSHOT	7.0.0.2
moduleName	Sample Module	Form Center Mongo Extension

The interface asks to confirm the new values:

```
Confirm properties configuration:
groupId: org.jahia.modules
artifactId: fc-mongo-extension
version: 1.0-SNAPSHOT
package: org.jahia.modules
digitalFactoryVersion: 7.0.0.0-SNAPSHOT
moduleName: Form Center Mongo extension
 Y: :
```

Reply 'Y'.

We now have an empty module for Digital Factory 7:



3 Editing the *pom.xml* File

We are now going to edit the *pom.xml* file to adapt the module to our needs.

Add these tags in the *pom.xml* file:

```
<properties>
    <jackson.version>2.3.1</jackson.version>
    <jackson-annotations.version>2.3.0</jackson-annotations.version>
    <jersey.version>2.6</jersey.version>
</properties>
<dependencies>
   <!-- Modules -->
    <dependency>
        <groupId>org.jahia.modules</groupId>
        <artifactId>df-form-center</artifactId>
        <version>1.0-SNAPSHOT</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.jahia.modules</groupId>
        <artifactId>database-connector</artifactId>
        <version>2.0-SNAPSHOT</version>
        <scope>provided</scope>
    </dependency>
    <!-- Jackson And Jax-RS -->
    <dependency>
        <groupId>javax.ws.rs</groupId>
        <artifactId>javax.ws.rs-api</artifactId>
        <version>2.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.jaxrs</groupId>
        <artifactId>jackson-jaxrs-json-provider</artifactId>
        <version>${jackson.version}</version>
    </dependency>
```

```
    <dependency>
        <groupId>org.glassfish.jersey.bundles</groupId>
        <artifactId>jaxrs-ri</artifactId>
        <version>${jersey.version}</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.media</groupId>
        <artifactId>jersey-media-multipart</artifactId>
        <version>${jersey.version}</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>${jackson.version}</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>${jackson-annotations.version}</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>${jackson.version}</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.jaxrs</groupId>
        <artifactId>jackson-jaxrs-base</artifactId>
        <version>${jackson.version}</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.module</groupId>
        <artifactId>jackson-module-jaxb-annotations</artifactId>
        <version>${jackson.version}</version>
    </dependency>
</dependencies>
```

The module has dependencies to the *database-connector* and *df-form-center* modules, which will allow it to call their services or implement their interfaces.

All the following dependencies pertain to JAX-RS (results API) and Jackson (JSON object modeling).

Edit the *plugin* tag so that it matches the following code:

```
<plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <extensions>true</extensions>
    <configuration>
        <instructions>
            <Import-Package>
                javax.ws.rs;resolution:=optional;version="[1.1,3)",
                javax.ws.rs.ext;resolution:=optional;version="[1.1,3)",
                javax.ws.rs.core;resolution:=optional;version="[1.1,3)",
```

```
                org.springframework.data.mongodb.core.convert,
                ${jahia.plugin.projectPackageImport},
                *
            </Import-Package>
            <Jahia-Depends>default</Jahia-Depends>
            <JAX-RS-Alias>/mongo</JAX-RS-Alias>
            <JAX-RS-
Application>org.jahia.modules.fc.mongoextension.api.ResultApiApplication</JAX-RS-
Application>
        </instructions>
    </configuration>
</plugin>
```

- The *Import-Package* tag details the versions and resolution types of imports. Without this, OSGI will not be able to resolve the exact imports in the correct versions without any conflict.
- The *JAX-RS-Alias* tag defines the alias (or main entry point) for our API. Any request to the API will be done via the URL: *<SERVER-URL>/modules/mongo* .
- The *JAX-RS-Application* tag defines the JAX-RS application class, which defines and initializes the API (the entry points and functions for the API will be in a different class, declared as a component of the API).

4 Defining the New Action

Now that the module has been created, we are going to define the new action.

JCR Definition
In the file *definition.cnd*, add the following line:

```
[fcnt:saveToMongoAction] > jnt:content, fcmix:action, mix:title,
jmix:droppableContent, jmix:hiddenType
```

This definition tells the JCR that a node of the type *fcnt:saveToMongoAction* can be saved. As with all actions in the Form Center, the type must inherit from *fcmix:action* in order to be recognized. The other inheritances are used for displaying the action in the Form Builder.

4.1 repository.xml

In the *main* folder, create an *import* subfolder and create the file *repository.xml* inside it, containing the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<content xmlns:j="http://www.jahia.org/jahia/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0">
    <modules jcr:primaryType="jnt:modules">
        <fc-mongo-extension j:dependencies="df-form-center database-connector"
                            j:modulePriority="0"
                            j:moduleType="module"
                            j:title="FC Mongo Extension"
                            jcr:primaryType="jnt:module">

            <portlets jcr:primaryType="jnt:portletFolder"/>
            <files jcr:primaryType="jnt:folder"/>
            <contents jcr:primaryType="jnt:contentFolder"/>
            <templates j:rootTemplatePath="/base"
jcr:primaryType="jnt:templatesFolder">
                <files jcr:primaryType="jnt:folder"/>
                <contents jcr:primaryType="jnt:contentFolder">
                    <form-center-actions jcr:primaryType="jnt:contentFolder">
                        <savetomongo jcr:primaryType="fcnt:saveToMongoAction"
tab="action">
                            <j:translation_en jcr:language="en"
                                              jcr:mixinTypes="mix:title"
                                              jcr:primaryType="jnt:translation"
                                              jcr:title="Save to MongoDB"
                                              label="ID / Name"/>
                            <label jcr:primaryType="fcnt:definitionOptions"
type="inputdisabled"
                                   jsonValue="Save to MongoDB">
                                <j:translation_en jcr:language="en"
                                                  jcr:primaryType="jnt:translation"
                                                  label="Label Action"/>
                            </label>
                            <actionname jcr:primaryType="fcnt:definitionOptions"
type="inputdisabled"
                                        jsonValue="savetomongo">
                                <j:translation_en jcr:language="en"
                                                  jcr:primaryType="jnt:translation"
                                                  label="Action Name"/>
                            </actionname>
                            <apiEntryPoint jcr:primaryType="fcnt:definitionOptions"
type="hidden"
                                           jsonValue="/modules/mongo">
                                <j:translation_en jcr:language="en"
                                                  jcr:primaryType="jnt:translation"
                                                  label="Api Entry Point"/>
```

```
                            </apiEntryPoint>
                            <databaseid jcr:primaryType="fcnt:definitionOptions"
type="input"
                                        jsonValue="mongo">
                                <j:translation_en jcr:language="en"
                                                  jcr:primaryType="jnt:translation"
                                                  label="Mongo Databse Id"/>
                            </databaseid>
                        </savetomongo>
                    </form-center-actions>
                </contents>
            </templates>
        </fc-mongo-extension>
    </modules>
</content>
```

This code makes it possible to write a node *form-center-actions* to the JCR, containing a node *savetomongo* of the type *fcnt:saveToMongoAction* with a *jcr:title* with value 'Save to MongoDB' in English language, as well as its subnodes, when deploying the module.

- A subnode *label* with value 'Save to MongoDB'
- A subnode *actionname* with value 'savetomongo'
- A subnode *apiEntryPoint* defining the API's entry point for this action

These nodes will exist under the path: */modules/fc-mongo-extension/1.0-SNAPSHOT/templates/contents* .

Important note: when the fields, validations and actions available in the form creation page are initialized, the *df-form-center* module scans all the nodes located in */modules* for definitions. The file *repository.xml* is therefore of critical importance for the visibility of our action in the Form Creation section. The value of the *actionname* node ('savetomongo') is extremely important for the rest of the code because it is the name of the Action class that will be executed by Webflow when submitting the form. Therefore, the action class of our new action must imperatively be *SaveToMongoAction.java* .

4.2 Image

Create an *img* folder in the module and inside place an image file containing the icon that will represent your action. For MongoDB, we will use the MongoDB logo:

4.3 Description

In the file *fc-mongo-extension.properties*, add the property corresponding to the action's description:

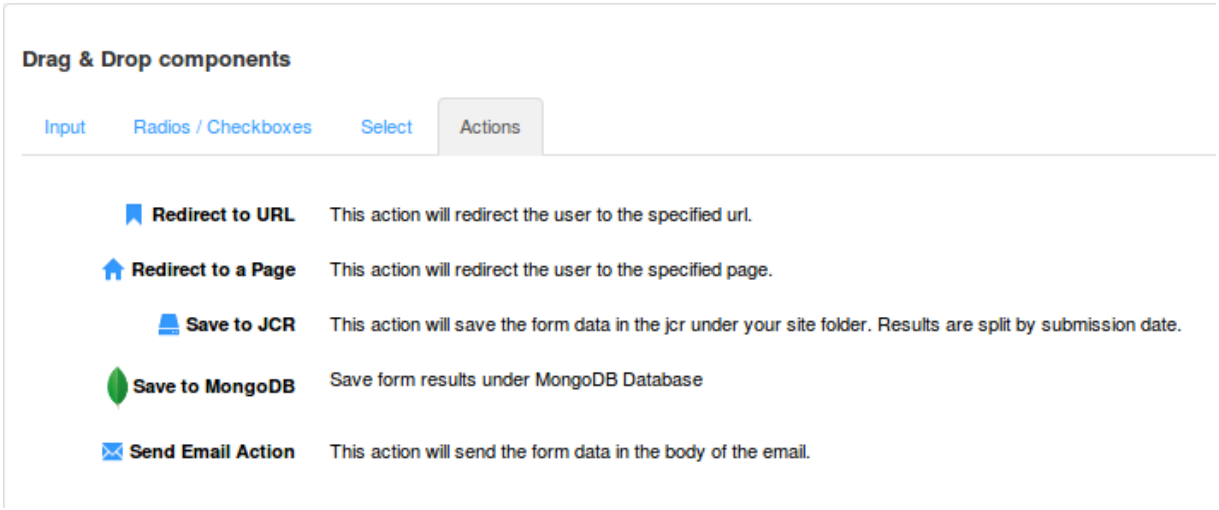
```
actions.savetomongo.description= Save form results under MongoDB Database
```

4.4 Underscore Template

Now that the action is defined in the JCR, under the *resources* folder create a folder *fcnt_saveToMongoAction* containing an *html* subfolder.
In the *html* subfolder, create a file named *saveToMongoAction.jsp* containing the following code:

```
<%@ taglib prefix="functions" uri="http://www.jahia.org/tags/functions" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<fmt:message var="description" key="actions.savetomongo.description"/>
<c:url var="imageUrl" value='${url.currentModule}/img/mongo_12_26.png'/>
<dl class="dl-horizontal">
    <dt><img src="${imageUrl}"> <@= label @></dt>
    <dd data-toggle="tooltip" title="${description}">
        ${functions:abbreviate(description,200,300,'...')}
    </dd>
</dl>
```

This code is used to define how the action will be displayed in the form creation view.



4.5 Action Classes

We can now create a new class *SaveToMongoAction.java* under the package *org.jahia.modules.fc.mongoextension.actions* . This class will be initialized with the following code:

```
package org.jahia.modules.fc.mongoextension.actions;

import org.jahia.bin.Action;
import org.jahia.bin.ActionResult;
import org.jahia.modules.formcenter.ApiBackendType;
import org.jahia.services.content.JCRSessionWrapper;
import org.jahia.services.render.RenderContext;
import org.jahia.services.render.Resource;
import org.jahia.services.render.URLResolver;
import org.json.JSONException;
import org.json.JSONObject;
import org.slf4j.Logger;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.List;
import java.util.Map;

import static org.slf4j.LoggerFactory.getLogger;

/**
 * This class will implement the Java Action execution that will save Form Results to MongoDB
 */
public class SaveToMongoAction extends Action implements ApiBackendType {

        private static final Logger logger = getLogger(SaveToMongoAction.class);

        public String getBackendType(){
            return "MONGO";
        }
        @Override
        public ActionResult doExecute(HttpServletRequest req, RenderContext renderContext,
Resource resource, JCRSessionWrapper session, Map<String, List<String>> parameters,
URLResolver urlResolver) throws JSONException {

            //TODO:Implement Action execution

            //Action execution result
            JSONObject jsonAnswer = new JSONObject();
            jsonAnswer.append("actionName", resource.getTemplate());
            jsonAnswer.append("status", "success");
            jsonAnswer.append("code", HttpServletResponse.SC_CREATED);
            jsonAnswer.append("redirectUrl", "");
            jsonAnswer.append("message", "The action Save to MongoDB has been called");

            return new ActionResult(HttpServletResponse.SC_OK, null, jsonAnswer);
        }
}
```

The action implements the *ApiBackendType* interface, which is used to declare the type of backend used by the action (*getBackendType()*). This must be a constant chosen by the developer (JCR, MONGO...). This value will be mapped to another constant (*apiName*) through an implementation of *ResultsProviderService*, in order to know the name of the file containing the entry points for the API that can query the results. The *doExecute* function must return an object of the *JSONObject* type with the following structure: *{actionname:<String>, status:<String>, code:<Integer>, redirectUrl:<String>, message:<String>}* .
In order for the Form Center to be able to know the result views corresponding to this new action, we need to write an implementation of the *ResultsProviderService* interface.
Create a package *impl* and add the file *SaveToMongoRawResultsImpl.java* with the following code:

```
package org.jahia.modules.fc.mongoextension.impl;

import org.jahia.modules.formcenter.ResultsProviderService;

/**
 * Created by rizak on 12/09/14.
 */
public class SaveToMongoRawResultsImpl implements ResultsProviderService {
    @Override
    public String getApiName() {
        return "rawResults";
    }

    @Override
    public String getBackendType() {
        return "MONGO";
    }
}
```

This implementation declares that the MONGO backend API uses the *rawResults* views (existing views in the formCenter: *dashboard*, *datatable* and *metadata*) to return the results.

4.6 API Classes

Create a new class *ResultApiApplication.java* under the package *org.jahia.modules.fc.mongoextension.api* . This class will be initialized with the following code:

```
public class ResultApiApplication extends ResourceConfig{
```

```
        public ResultApiApplication() {
        this(RepositoryFactory.class, DatabaseConnectorServiceFactory.class,
BundleContextFactory.class);
        }
        ResultApiApplication(final Class<? extends Factory<Repository>>
repositoryFactoryClass, final Class<? extends Factory<DatabaseConnectorOSGiService>>
databaseConnectorServiceFactoryClass, final Class<? extends Factory<BundleContext>>
bundleContextFactoryClass ) {
        super(MongoResultsApi.class, repositoryFactoryClass,
databaseConnectorServiceFactoryClass,bundleContextFactoryClass,
JacksonJaxbJsonProvider.class, HeadersResponseFilter.class, MultiPartFeature.class);
        register(new AbstractBinder() {
                @Override
                protected void configure() {
                bindFactory(repositoryFactoryClass).to(Repository.class);

        bindFactory(databaseConnectorServiceFactoryClass).to(DatabaseConnectorOSGiSe
rvice.class);
                bindFactory(bundleContextFactoryClass).to(BundleContext.class);
                }
        });
        }
}
```

Then create a class *SpringBeansAccess.java* with the following code:

```
/**
 * This class defines the bean that will inject objects to the API class
 */
public final class SpringBeansAccess {
    private final static SpringBeansAccess INSTANCE = new SpringBeansAccess();
    private Repository repository;
    private SpringBeansAccess() {
    }
    public static SpringBeansAccess getInstance() {
        return INSTANCE;
    }
    public Repository getRepository() {
        return repository;
    }
    public void setRepository(Repository repository) {
        this.repository = repository;
    }
}
```

Finally, create the class *MongoResultsApi.java* that implements the API:

```
/**
 * The main entry point to the Digital Factory Form Center Results API.
 * This class will provide all the needed methods to fetch results and filter them
 *
 * @author Rizak AHMED
 */
@Component
@Path(org.jahia.modules.fc.mongoextension.api.MongoResultsApi.API_PATH)
@Produces({"application/hal+json"})
public class MongoResultsApi {
    private static final Logger logger =
```

```
getLogger(org.jahia.modules.fc.mongoextension.api.MongoResultsApi.class);
    static final String API_PATH = "/mongo";

    //Services
    private Repository repository;

    //Attributes
    private String language;
    private String resource;

    public MongoResultsApi() {
    }

    @Inject
    public MongoResultsApi(Repository repository) {
        this.repository = repository;
    }

    /**
     * Hello world function to check Result API is responding
     *
     * @return "Mongo API Hello World!"
     */
    @GET
    @Path("/hello")
    @Produces(MediaType.APPLICATION_JSON)
    public ApiResponse getHello(@PathParam("resource") String resource,
@PathParam("language") String language) {
        return new ApiResponse("success", HttpServletResponse.SC_OK, "Hello World!",
"Mongo API Hello World!");
    }
}
```

4.7 Spring Declarations

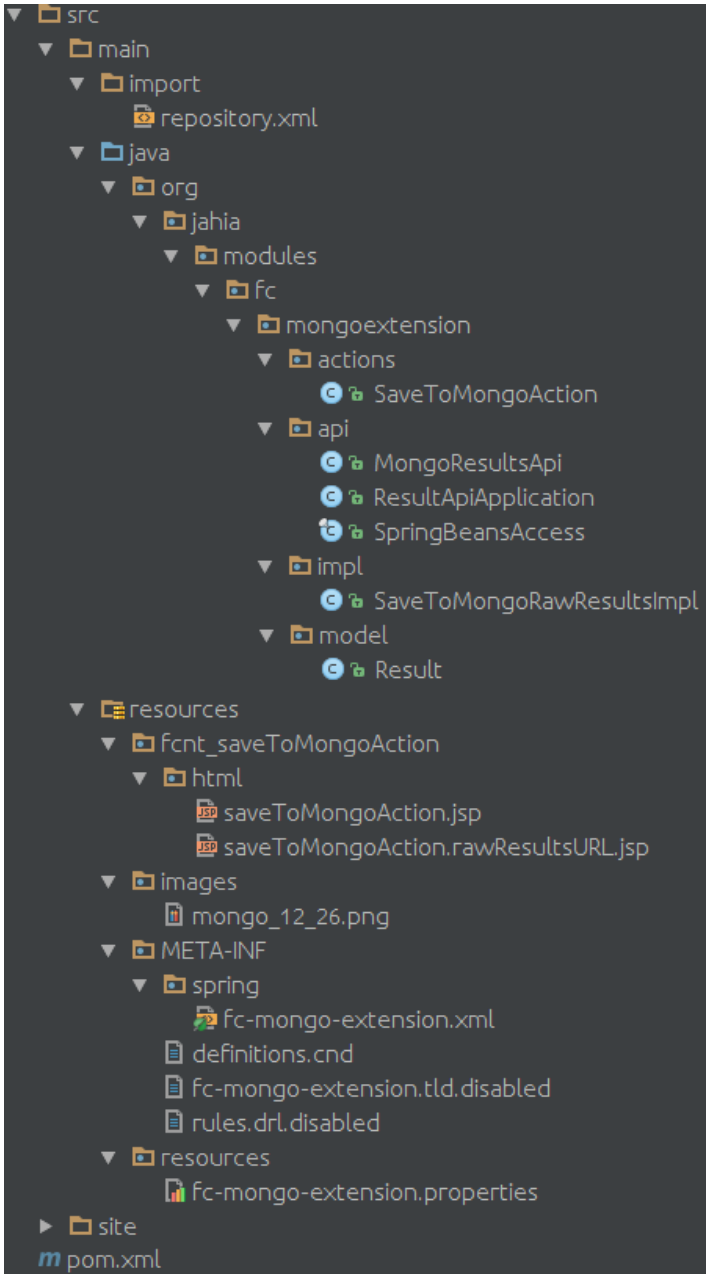
Now that the class of our action is defined, we need to declare the action in Spring. To do so, we will first rename the file *fc-mongo-extension.xml.disabled* located under the path *main/resources/META-INF/spring/* to *fc-mongo-extension.xml* and then edit it so that it contains the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id="SaveToMongoAction"
class="org.jahia.modules.fc.mongoextension.actions.SaveToMongoAction" />
    <bean id="springBeansAccess"
class="org.jahia.modules.fc.mongoextension.api.SpringBeansAccess" factory-method="getInstance">
        <property name="repository" ref="jcrSessionFactory"/>
    </bean>
    <osgi:service interface="org.jahia.modules.formcenter.ResultsProviderService">
        <bean id="SaveToMongoRawResults"
class="org.jahia.modules.fc.mongoextension.impl.SaveToMongoRawResultsImpl"/>
    </osgi:service>
</beans>
```

4.8 Testing the Action

4.8.1 Deploying the Module

Now that our action has been defined, our module has the following tree structure:



We can now deploy the module to test the action. Run the following Maven command in order to deploy the module:

```
$: mvn install jahia:deploy
```

4.8.2 Creating and Submitting a Form

In order to test our action, we are now going to create and publish a form that will only contain our new action in the action tab.
Once the form has been published, add it to a page and submit it.

The action is called successfully.

4.8.3 Testing the Results API

Call to the *hello* function:

```
$: curl http://localhost:8080/jahia701/modules/mongo/hello
```

```
{"status":"success","code":200,"message":"Hello World!","data":"Mongo API Hello
World!"}
```

5 Implementation

5.1 Saving the Result to MongoDB

In order to save the results to MongoDB, we will need to modify the action defined in the previous chapter:

5.1.1 Retrieving the *databaseId*

To start with, we need to retrieve the *databaseId* corresponding to the action:

5.1.2 Retrieving the MongoDB Connection

In order to retrieve the MongoDB connection, we will need to use the OSGI service exposed by *database-connector* and to implement the *BundleContextAware* interface.

- Injection of the service *DatabaseConnectorOSGiService* :

  Spring file (*fc-mongo-extension.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/osgi http://www.springframework.org/schema/osgi/spring-
osgi.xsd">
    <osgi:reference id="databaseConnectorOSGiService"
                    interface="org.jahia.modules.databaseConnector.DatabaseConnectorOSGiService"
                    availability="mandatory"/>
    <bean id="SaveToMongoAction"
class="org.jahia.modules.fc.mongoextension.actions.SaveToMongoAction">
        <property name="databaseConnectorService" ref="databaseConnectorOSGiService"/>
    </bean>
    <bean id="springBeansAccess"
class="org.jahia.modules.fc.mongoextension.api.SpringBeansAccess" factory-method="getInstance">
        <property name="repository" ref="jcrSessionFactory"/>
    </bean>

    <osgi:service interface="org.jahia.modules.formcenter.ResultsProviderService">
        <bean id="SaveToMongoRawResults"
class="org.jahia.modules.fc.mongoextension.impl.SaveToMongoRawResultsImpl"/>
    </osgi:service>
</beans>
```

Class *SaveToMongoAction.java* :

```
...
/**
 * This class will implement the Java Action execution that will save Form Results to MongoDB
 */
public class SaveToMongoAction extends Action implements ApiBackendType, BundleContextAware {

    private static final Logger logger = getLogger(SaveToMongoAction.class);
    private DatabaseConnectorOSGiService databaseConnectorService;

    private BundleContext bundleContext;
    private DatabaseConnectorManager connectorManager = DatabaseConnectorManager.getInstance();
    @Override
    public void setBundleContext(BundleContext bundleContext) {
        this.bundleContext = bundleContext;
    }

    public void setDatabaseConnectorService(DatabaseConnectorOSGiService
databaseConnectorService) {
        this.databaseConnectorService = databaseConnectorService;
    }

    public String getBackendType(){
        return "MONGO";
    }

    @Override
    public ActionResult doExecute(HttpServletRequest req, RenderContext renderContext, Resource
resource,
                                  JCRSessionWrapper session, Map<String, List<String>>
parameters,
                                  URLResolver urlResolver) throws JSONException,
InvalidSyntaxException {

        //Action execution result
        JSONObject jsonAnswer = new JSONObject();
        jsonAnswer.append("actionName", resource.getTemplate());
        jsonAnswer.append("redirectUrl", "");

        //Mongo Template
        MongoTemplate mongoTemplate = null;
        try{
            // Get the node Results Folder Node
            QueryManager qm = session.getWorkspace().getQueryManager();
            Query q = qm.createQuery("SELECT * FROM [fcnt:formResults] AS child WHERE
child.[parentForm] = '" + parameters.get("formId").get(0) + "'", Query.JCR_SQL2);
            JCRNodeWrapper resultsFolder = (JCRNodeWrapper) q.execute().getNodes().nextNode();

            //Get labels Map
            JCRNodeWrapper labelsNode = resultsFolder.getNode("labels");
            Map<String, String> labelIds = new LinkedHashMap<String, String>();
            JCRNodeIteratorWrapper labels = labelsNode.getNodes();
            while(labels.hasNext()){
                JCRNodeWrapper label = (JCRNodeWrapper)labels.next();
                labelIds.put(label.getName().split("-")[0],label.getIdentifier());
            }
            //Create fields Map
            Map<String,Object> fields= new LinkedHashMap<String, Object>();
```

```
            for(Map.Entry<String,String> label : labelIds.entrySet()){
                String labelID = label.getValue();
                List<String> values = parameters.get(label.getKey());
                Object resultValues=null;
                if(values!= null && CollectionUtils.isNotEmpty(values)){
                    resultValues=values.size()>1?values:values.get(0);
                }
                fields.put(labelID,resultValues);
            }
            //Create Result Object from form result
            Result mongoResult = new Result( fields, resultsFolder.getIdentifier(),
session.getUser().getUsername(), req.getRequestURL().toString());

            //Get DatabaseID from action
            String databaseIdRequest = Messages.format("SELECT * from [fcnt:saveToMongoAction]
as action WHERE ISDESCENDANTNODE(action,'{0}')",resultsFolder.getPath()+"/actions");
            Query databaseIdQuery = qm.createQuery(databaseIdRequest,Query.JCR_SQL2);
            JCRNodeWrapper actionNode =
(JCRNodeWrapper)databaseIdQuery.execute().getNodes().nextNode();
            String databaseId =
actionNode.getNode("databaseid").getProperty("jsonValue").getString();
            if (databaseConnectorService.registerDatabase(databaseId, MONGO)) {
                ServiceReference[] serviceReferences = bundleContext.getAllServiceReferences(
                        MongoTemplate.class.getName(), createFilter(MONGO, databaseId));
                if (serviceReferences != null) {
                    mongoTemplate =  (MongoTemplate)
bundleContext.getService(serviceReferences[0]);
                }
            }
            //Write result into MongoDB Database
            if(mongoTemplate!=null){
                mongoTemplate.insert(mapper.writeValueAsString(mongoResult),"results");
            }
            //Set Json Success
            jsonAnswer.append("status", "success");
            jsonAnswer.append("code", HttpServletResponse.SC_CREATED);
            jsonAnswer.append("message", Messages.get("resources.fc-mongo-extension",
"actions.resultSaved", renderContext.getUILocale()));
        }
        catch(Exception e){
            //Set Json Error
            jsonAnswer.append("status", "error");
            jsonAnswer.append("code", HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            jsonAnswer.append("message", Messages.get("resources.fc-mongo-extension",
"actions.resultNotSaved", renderContext.getUILocale()));
        }
        //Return Action Json Status
        return new ActionResult(HttpServletResponse.SC_OK, null, jsonAnswer);
    }
}
```

Class *Result.java* of the package *org.jahia.modules.fc.mongoextension.model* :

```
/**
 * Created by rizak on 24/10/14.
 */
public class Result {
    Map<String,Object> fields;
    private Date creationDate;
    private String folderId;
    private String savedBy;
    private String formPageURL;

    public Result(){
        this.fields = new LinkedHashMap<String, Object>();
        this.creationDate = new Date();
    }

    public Result(String folderId, String savedBy, String formPageURL){
        this.fields = new LinkedHashMap<String, Object>();
        this.creationDate = new Date();
        this.folderId = folderId;
        this.savedBy = savedBy;
        this.formPageURL = formPageURL;
    }

    public Result(Map<String, Object> fields, String folderId, String savedBy, String
formPageURL){
        this.fields = fields;
        this.creationDate = new Date();
        this.folderId = folderId;
        this.savedBy = savedBy;
        this.formPageURL = formPageURL;
    }

    public Map<String, Object> getFields() {
        return fields;
    }

    public void setFields(Map<String, Object> fields) {
        this.fields = fields;
    }

    public Date getCreationDate() {
        return creationDate;
    }

    public void setCreationDate(Date creationDate) {
        this.creationDate = creationDate;
    }

    public String getFolderId() {
        return folderId;
    }

    public void setFolderId(String folderId) {
        this.folderId = folderId;
    }

    public String getSavedBy() {
        return savedBy;
    }
```

```
    public void setSavedBy(String savedBy) {
        this.savedBy = savedBy;
    }
}
```

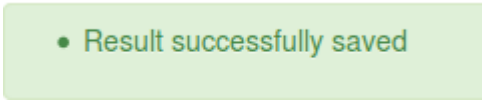
ResourceBundle file:

```
actions.savetomongo.description= Save form results under MongoDB Database
actions.resultSaved=Result successfully saved
actions.resultNotSaved=Result could not be saved
```

Testing the Action

5.1.3 Submitting a Form

Let's submit our form again:



Now let's check what is in our MongoDB database.

```
> db.results.find()
{ "_id" : ObjectId("54528c71ccf207f8120d5345"), "fields" : { "bdadd703-8b95-41b0-
83c6-38d23bdae9f9" : "Rizak Test" }, "creationDate" : NumberLong("1414696049757"),
"folderId" : "a65c1bd2-f48b-4e0b-bb19-3d55851603ab", "savedBy" : "root" }
>
```

The result has indeed been written to the MongoDB database.

6 Results API

Now that we have a form action that can save results to a MongoDB collection, we are going to add the necessary entry points in the API in order to display the results.

6.1 URL Mapping JSP

In the *html* folder of the node *fcnt_saveToMongoAction*, add the file *saveToMongoAction.rawResultsURL.jsp* containing the following code:

```
<template:addResources type="inlinejavascript">
        <script>
        var apiName = "rawResults";
        var apiURLBase =
"${url.context}/modules/mongo/${currentResource.locale}/";

        //Listing API entries in an object
        var apiMethods = {
        "results":function(parameters){
                if(parameters.fromDate!=null && parameters.toDate!=null){
                return apiURLBase + parameters.formId + "/from/" +
parameters.fromDate + "/to/" + parameters.toDate + "/results";
                }
                else{
                return apiURLBase + parameters.formId + "/results";
                }
        },
        "results_submissiontime":function(parameters){
                return apiURLBase + parameters.formId + "/submissiontime";
        },
        "results_submissionpage":function(parameters){
                return apiURLBase + parameters.formId + "/submissionpage";
        },
        "results_submissionempty":function(parameters){
                return apiURLBase + parameters.formId + "/submissionempty";
        },
        "results_choice":function(parameters){
                return apiURLBase +
parameters.formId+"/choice/"+parameters.choiceId;
        },
        "total":function(parameters){
                return
"${url.context}/modules/mongo/total/"+parameters.formId;
        },
        "lastDays":function(parameters){
                return
"${url.context}/modules/mongo/${renderContext.UILocale.language}/totallastdays
/"+parameters.formId;
        },
        "groupTotal":function(parameters){
```

```
                return
"${url.context}/modules/mongo/${renderContext.UILocale.language}/groupTotal/"+
parameters.formId;
        },
        "results_choicelabel":function(parameters){
                return
"${url.context}/modules/formcenter/results/${currentResource.locale}/"+paramet
ers.formId+"/results/choicelabel";
        },
        "results_label":function(parameters){
                return
"${url.context}/modules/formcenter/results/${currentResource.locale}/"+paramet
ers.formId+"/results/labels";
        }
        };

        //Generate a URL from parameters
        function getFetchURL(urlParameters) {
                return apiMethods[urlParameters.apiMethodName](urlParameters);
        }
        </script>
</template:addResources>
```

The *apimethods* object is used to map the entry points for the API with keys of the *String* type.
In this jsp, we notice that the keys ***results_choicelabel*** and ***results_label*** are mapped with entry points of the *formCenter* module (indeed, whichever action is called, some of the form's metadata are saved to the JCR and the API of the formCenter is thus capable of returning them). Each function of the API returns the same Json object, with the following structure:

{
   "status" : <Status>,
   "code" : 200,
   "message" : "",
   "data" : <Data Object>
}

Below is the list of the keys to be mapped for the *rawResult* interface and the expected 'data' objects:

Key	Functionality	Returned data object

results	Returns the list of results for a Backbone Pageable model. Populates the columns of the *datatable* view.	"data" : [{
		"submissionDate" : <Date (in long format)>,
		"folderId" : <Folder JCR Id>,
		"username" : <Username>,
		"origin" : <Page URL>,
		"id" : <JCR Id>,
		"results" : [{
		"label" : <Label JCR Id>,
		"htmlId" : <HTML Id>,

		"values" : [<String values>] }] }]

results_submissiontime	Returns the number of submissions per time unit (months, years etc). Populates a graph in the *dashboard* view (line graph).	"data":{
		"groupingPattern" : "<GroupingPattern>",
		"timeTotal":
		[{"timekey" : <TimeGroup>,
		"count":<Count>}]
		}

results_submissionpage	Returns the number of submissions per page and per time unit (months, years etc). Populates a graph in the *dashboard* view (bar graph).	"data" : {
		"groupingPattern" :
		"<GroupingPattern>",
		"groupedCount" : [
		{
		"counts" :
		[{"timekey":<TimeGroup>,
		"count" : <Count>}],
		"page" : <PageURL>
		}
		]
		}

results_submissionempty	Returns the number of submissions per form input field, as well as the form's total number of submissions (the computation of empty fields will be done in the view). Populates a table in the *dashboard* view.	"data" : {
		"countByField" : [{
		"field" : "<FieldLabel>"
		"count" : <Count>
		}]
		}

results_choice	Returns the number of submissions per choice of a multiple value field. Populates a graph in the *dashboard* view (pie chart).	"data" : [{ "key" : "<choiceKey>" , "value" : <Count>, "label" : "<ChoiceValue>" }]
total	Returns the total number of submissions for a form since its creation (result view count).	"data":<Count>

lastdays	Returns the total number of submissions for a form over the last *n* days (result view count).	`"data" : { "total" : <Count>, "label" : <Label String>}`
grouptotal	Returns the total number of submissions for a form since its creation. Results will be grouped following a parameter (day, week, month, year) (result view count).	`"data" : { "total" : { <TimeGroup>:<Count>}, "label" : <Label String> }`
results_choicelabel	Returns the list of multiple-choice fields along with their labels.	`"data" : { "choiceFields" : [{id:<ChoiceFieldJCRId>, label : <LabelValue>}] }`
results_label	Returns the labels of a form's fields (Name of the columns in the *datatable* view).	`"data" : { <Field HTML Id> : <Field Label Value>, }`

6.2 Function Implementation Example

The views of the *RawResult* interface (*metadata*, *dashboard* and *datatable*) can now make Ajax calls to the entry points of the JAX-RS API for MongoDB. The last step is to implement the functions that will return the results. Below is an example of the implementation of the *getSubmissionPage* function.

In the class *MongoResultsApi*, add the *getSubmissionPage* function with the following code:

```
        /**
         * This function Get the number of submissions through time.
         * The time grouping is handled by the groupBy parameter
         * @param asyncResponse : Make the method results asynchronous
         * @param resource : "submissions" node of the form results in JCR
         * @param language : Language to use (To return counts label from
bundle properties)
         * @param groupBy : Grouping optional parameter
         * @param fromDate : Date filter
         * @param toDate : Date filter
         */
         @GET
```

```
        @Path("/{language}/{resource}/submissiontime")
        @Produces({MediaType.APPLICATION_JSON})
        public void getSubmissionTime(@Suspended final AsyncResponse
asyncResponse,@PathParam("resource") final String resource,
                                      @PathParam("language") final String
language,@QueryParam("group_by") final List<String> groupBy,
@QueryParam("fromDate") final String fromDate, @QueryParam("toDate") final
String toDate) {
        new Thread(new Runnable() {
                @Override
                public void run() {
                Response result = null;
                try {
                        result = queryResultsThroughTime(resource, groupBy);
                } catch (InterruptedException e) {
                        logger.error(e.getMessage(), e);
                }
                asyncResponse.resume(result);
                }

                private Response queryResultsThroughTime(final String id,
final List<String> groupByList) throws InterruptedException {
                try {
                        return
JCRTemplate.getInstance().doExecuteWithSystemSession(null, "live", new
JCRCallback<Response>() {
                        @Override
                        public Response doInJCR(JCRSessionWrapper session)
throws RepositoryException {
                                Map<String, Object> map = new HashMap<String,
Object>();
                                JCRNodeWrapper resource =
session.getNodeByIdentifier(id);
                                JCRNodeWrapper jcrFormResults =
resource.getParent();
                                String buildingLang =
jcrFormResults.getProperty("buildingLang").getString();

        session.setFallbackLocale(LanguageCodeConverters.languageCodeToLocale(
buildingLang));
                                try{
                                MongoTemplate mongoTemplate =
getMongoTemplate(jcrFormResults,session);
                                Map<String, Integer> groupTotal = new
LinkedHashMap<String, Integer>();

                                //Create Aggregations List
                                List<AggregationOperation> aggList = new
ArrayList<AggregationOperation>();

                                //1St Aggregation step (Select fields)

        aggList.add(project("submissionDate").and("folderId").as("folderId").a
ndExpression("dayOfMonth(submissionDate)").as("day").andExpression("week(submi
ssionDate)").as("week").andExpression("month(submissionDate)").as("month").and
Expression("year(submissionDate)").as("year"));

        aggList.add(match(Criteria.where("folderId").is(jcrFormResults.getIden
tifier())));
                                //2nd Aggregation step (apply date filter)
                                Criteria whereCriteria =
getTimeCriteria(fromDate,toDate);
                                if(whereCriteria!=null){
```

```
                                        aggList.add(match(whereCriteria));
                                }

                                //3rd Aggregation step (apply grouping)
                                String groupBy = groupByList.get(0);
                                DateTimeFormatter countformat =
FormatEnum.FormatKey.getPatternFromKey(groupByList.get(0));
                                AggregationOperation groupOperation;
                                if(groupBy.equals("year"))
                                {
                                        groupOperation =
group(fields().and("submissionDate").and("year")).count().as("count");
                                }
                                else if(groupBy.equals("week")){
                                        groupOperation =
group(fields().and("year").and("week")).count().as("count");
                                }
                                else if(groupBy.equals("day")){
                                        groupOperation =
group(fields().and("year").and("month").and("day")).count().as("count");
                                }
                                else{
                                        groupOperation =
group(fields().and("year").and("month")).count().as("count");
                                }
                                aggList.add(groupOperation);

                                //Create Aggregation from list
                                Aggregation agg = newAggregation(aggList);

                                //Run Aggregation on database
                                AggregationResults<ResultCount> counts =
mongoTemplate.aggregate(agg, "results", ResultCount.class);

                                //Put results in Map
                                for(ResultCount result : counts){

        groupTotal.put(result.getGroup(countformat),result.getCount());
                                }

                                map.put("timeTotal",groupTotal);

        map.put("groupingPattern",FormatEnum.FormatKey.valueOf(groupByList.get
(0)).toString());
                                }
                                catch (Exception e) {
                                logger.error("getSubmissionTime Exception -
formId=" + resource + " - language=" + language + " - from date : " + fromDate
+ " - to Date : " + toDate);
                                logger.error(e.getMessage(), e);
                                return
Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity( new
ApiResponse("error", HttpServletResponse.SC_INTERNAL_SERVER_ERROR, "MongoDb
Exception", null)).build();
                                }
                                return
Response.status(Response.Status.OK).entity( new ApiResponse("success",
HttpServletResponse.SC_OK, "results per "+groupByList.get(0), map)).build();
                        }
                        });
                } catch (RepositoryException e) {
                        logger.error(e.getMessage(), e);
                }
```

```
                return Response.status(Response.Status.OK).entity(new
ApiResponse("error", HttpServletResponse.SC_BAD_REQUEST, "MongoDb Exception",
"no result")).build();
                }
        }).start();
        }
```

The annotations in the header of this function define its JAX-RS properties:

@GET: the type of HTTP method.

@Path("/{language}/{resource}/submissiontime"): the entry point of this function into the API (parameters are placed in the URL).

@Produces({MediaType.APPLICATION_JSON}): the return type of the function in the API. We can also notice that the parameters of the GET call are passed as arguments thanks to the @QueryParam annotation.

This function makes an asynchronous call to the sub-function *queryResultsThroughTime* that declares an object of the *MongoTemplate* type (Spring Data Object) and then builds an aggregation with a pipeline containing:

- a projection (selection and computation of the fields)
- two criteria (date and folder)
- a grouping (function of the *groupBy* parameter)

This aggregation returns objects of the type *ResultCount*, a class of the package *org.jahia.modules.fc.mongoextension.model* .
As for the function, it returns a Json Object with the following structure:

```
{"status":"success",
 "code":200,
 "message":"results per month",
 "data":{
    "groupingPattern":"YYYYMM",
    "timeTotal":{
        "201307":897,
        "201405":882,
     }
 }
}
```

That is a total of 897 submissions in July 2013 and 882 in May 2014. (These data depend on the contents of the MongoDB database.)

Class *ResultCount* :

```
/**
 * This Object represents the result from Mongo DB count.
 * It contains separate date fields because of the grouping options
 */
public class ResultCount {
        private int month;
        private int year;
        private int week;
        private int day;
        private int count;

        public String getOrigin() {return origin; }

        public void setOrigin(String origin) {
        this.origin = origin;
        }

        private String origin;

        public int getMonth() {
        return month;
        }

        public void setMonth(int month) {
        this.month = month;
        }

        public int getYear() {
        return year;
        }

        public void setYear(int year) {
        this.year = year;
        }

        public int getWeek() {
        return week;
        }

        public void setWeek(int week) {
        this.week = week;
        }

        public int getDay() {
        return day;
        }

        public void setDay(int day) {
        this.day = day;
        }

        public int getCount() {
        return count;
        }
```

```
        public void setCount(int count) {
        this.count = count;
        }

        /**
        * This method returns the result group String (formatted)
        * @param formatter the format for the date
        * @return a formatted string built with the result's date field
        */
        public String getGroup(DateTimeFormatter formatter){
        if(formatter!=null){
                DateMidnight submission = null;

                if(year>0 && month>0 && day>0){
                submission = new
DateMidnight().withYear(year).withMonthOfYear(month).withDayOfMonth(day);
                }
                else{
                if(year>0 && month>0){
                        submission = new
DateMidnight().withYear(year).withMonthOfYear(month);
                }
                else{
                        if(year>0 && week>0){
                        submission = new
DateMidnight().withYear(year).withWeekOfWeekyear(week);
                        }
                        else{
                                submission = new
DateMidnight().withYear(year);
                        }
                }
                }
                return formatter.print(submission);
        }
        return null;
        }
}
```

The *getGroup* function returns the character string containing the result group used to group the results.

Similarly, all entry points must be implemented respecting the definition of the data object contained at the end of the URL Mapping JSP section.

7 Summary

We have seen in this document that some elements were crucial when defining a new action:

- The mapping of the entry points for the API
- The name of the action class
- The Spring definition of the action
- The structure of the Json object to be returned

If the implementation guidelines are not respected for these elements, the action or the API will not work correctly.

Jahia Solutions Group SA

9 route des Jeunes,
CH-1227 Les acacias
Geneva, Switzerland

http://www.jahia.com