HACK-A-THON

MOMENTUM 2016

DOCUMENTUM SPRING DATA LAB

Table of Contents

Overview

Welcome to the Documentum Spring Data Lab

The goal of the Documentum Spring Data API is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores. Documentum Spring Data abstracts many of the Documentum DFC

What is Spring Data?

Spring Data is part of the Spring Framework, one of the most widely used frameworks for software development today. There are many Spring projects including MVC, integration, batch, web, et al. Spring Data provides a data access layer - transactions, resources, abstraction, exceptions, et al. Spring Data is an umbrella project contains many subprojects that are specific to a given database. Examples include JPA, Neo4J, MongoDB and many more.

Spring Data for Documentum provides an abstraction for the Documentum platform. The goal of Documentum Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores. Uses the same terminology as Spring Data

- Repository, CrudRepository, et al

Based on the Documentum Foundation Classes (DFC). Spring Data for Documentum provides full CRUD (create, read, update and delete) operations.

- Based on Spring Boot
- Methods include –
  - findAll()
  - Count()
  - findOne()
  - delete()
  - save()
  - exists()

FAQ

- What does this mean for DFS, Core REST and other APIs?
  - Not much - DFS will continue to be supported, since we use it in many products today. REST will also be supported and maintained going forward. Spring Data for Documentum provides developers another set of tools for developing applications on the Documentum platform.
- In this release, what is not included?
  - Not all DFC API features are in Spring Data for Documentum today; workflow and lifecycles are not in the base release. As the community grows, we will add more features!
- Can I contribute to the project?
  - Yes

Introducing the Documentum Address Book Client Sample Application

In this Hackathon lab, we will build a simple Address Book Client application that persists Address Book Contacts within Documentum. The Address Book Client is a simple AngularJS web application that allows a user to perform the following functions:

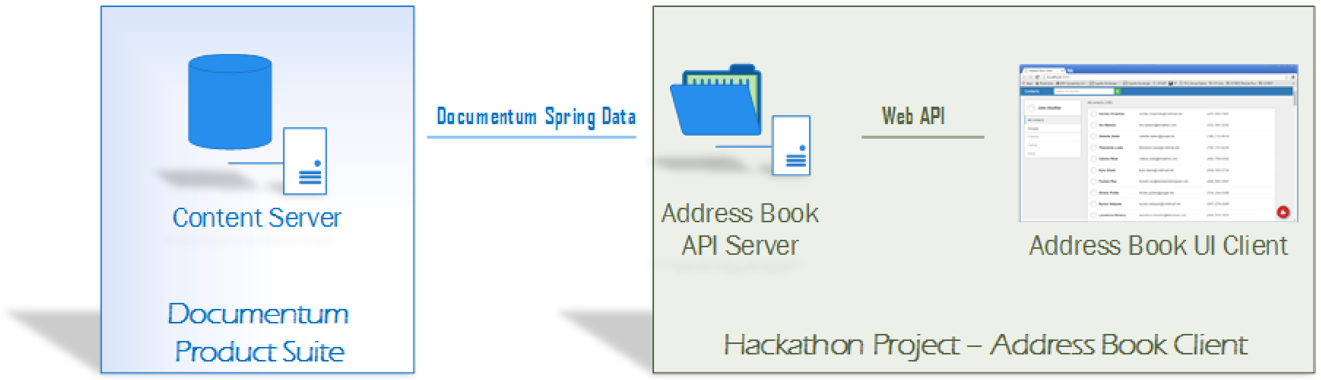
- View Contacts
- Add a Contact
- Edit a Contact
- Delete a Contact
- Upload a Contact Picture
- Filter Contacts by Group
- Search for Contacts

The goal of the Hackathon lab is to implement the backend services that the AngularJS UI Client communicates with via a simple REST API. The services will use Documentum Spring Data to perform all of the functions required by the UI Client Application.

The Address Book Client consists of three sub projects:

1. Address Book DocApp, this contains the contact object type that is installed in the repository.
2. Address Book API Server, this contains the REST API and the Documentum Spring Data Persistence logic
3. Address Book UI Client, this contains the simple AngularJS client application

The high level architecture of the application is shown below:



Section 1 Lab Introduction

Development Approach in This Lab

EMC Documentum @GitHub

GitHub is the world's largest code host. Built on the powerful Git version control system, GitHub is the ideal place to share code.

Code in GitHub lives in repositories. A repository contains all of the project files (including documentation), and stores each file's revision history. Repositories can have multiple collaborators and can be either public or private. The public GitHub repository for this hackathon is located at

https://github.com/Enterprise-Content-Management/documentum-springdata-lab

Branching is the duplication of source code under version control so that modifications can happen in parallel. Here we'll use branches a bit differently. Each round of coding has its own branch, so that you can start each round at the same stage as everybody else.

To get the complete working version of the lab you can clone the master branch.

This lab will be focused on Documentum Spring Data development so the AngularJS Client Code and the majority of the Server REST API code are already prepared and do not need to be modified. The rounds and steps below will highlight where code changes/additions need to be made.

Section 2 Address Book Client Development

Project Setup

We will start by checking the Documentum software set up and troubleshooting in the virtual machine. You will need to remote desktop connect to the virtual machine that you been assigned. All tasks in this lab shall be completed in your virtual machine. You don't need to install anything on your own laptop, unless you do not have the Remote Desktop software installed on your laptop yet.

Content Server Preparation

Content Server docbroker and repositories are started up by default. If not, please open run startAll.bat to start them. There is only one repository installed in this demo environment, whose name is repo1.

Documentum Administrator (DA) is deployed to the same Tomcat server, so now you are able to navigate Content Server repository by DA, too. It's root URL is: http://localhost:8083/da.

For any application that requires authentication to the Content Server repository, you can always use the username dmadmin and password D3m04doc!.

NOTE: the Address Book DocApp that contains the Contact object type is preinstalled in the image so you do not need to install it.

Get the code

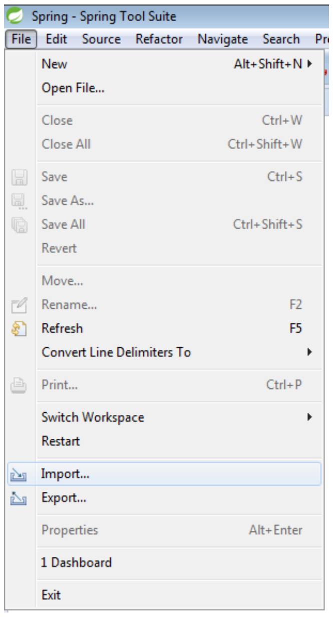
Each round in the lab you can either build on your code from the previous round or you can simply clone the branch for that round from GitHub. To get started you should clone the branch for the first round:

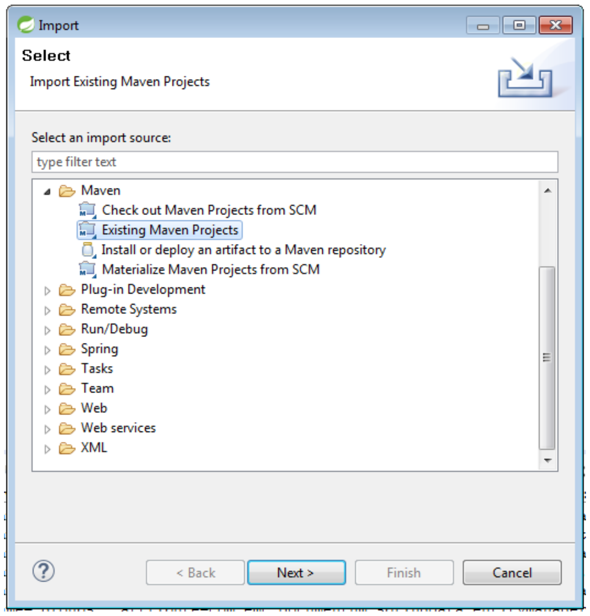
1. Open a command window
2. Browse to: **C:\momentum**
3. Run: **git clone –b round1 https://github.com/Enterprise-Content-Management/documentum-springdata-lab**

How to run the Address Book Server

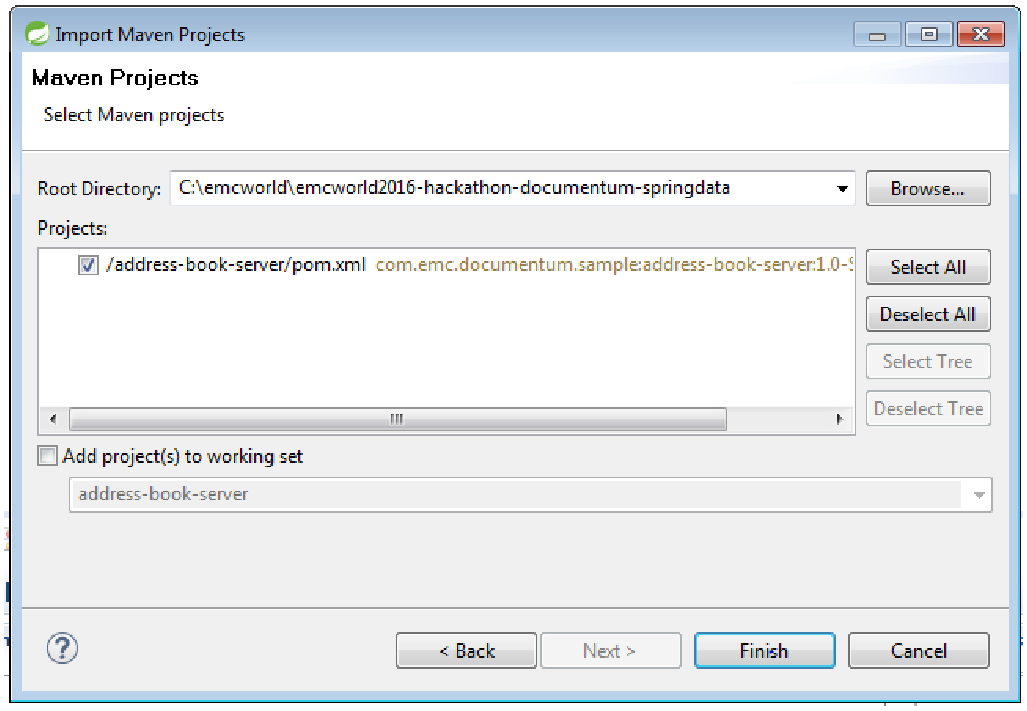
The Address Book API Server project is a Maven based Spring Boot WebApp project so you are free to open/edit/run in your IDE of choice. The instructions below are based on Spring Tool Suite which is pre-installed on the server.

1. Open Spring Tool Suite
2. Create a new Workspace under **C:\workspace\<your_name>**
3. Select **File => Import => Maven => Existing Maven Projects**





4. Browse to and select the address-book-server project from **C:\momentum\documentum-springdata-lab\**



Root Directory: C:\emcworld\emcworld2016-hackathon-documentum-springdata

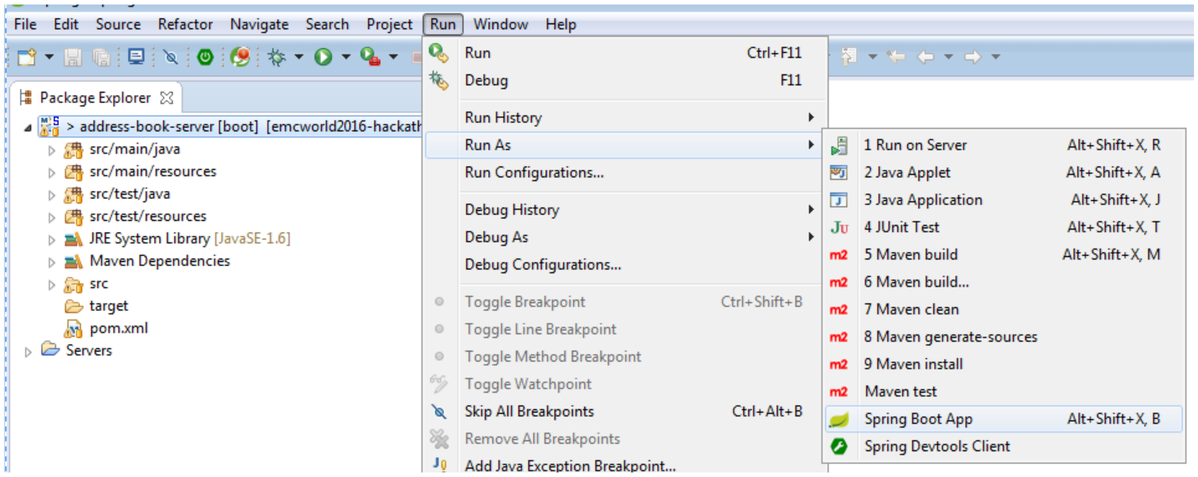
Projects:

☑ /address-book-server/pom.xml com.emc.documentum.sample:address-book-server:1.0-

☐ Add project(s) to working set

address-book-server

5. Press **Finish** (wait for the project to import)
6. Select the project
7. Select **Run Menu => Run As => Spring Boot App**


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
> address-book-server [boot] [emcworld2016-hackath
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-1.6]
Maven Dependencies
src
target
pom.xml
Servers
Run Ctrl+F11
Debug F11
Run History
Run As
Run Configurations...
Debug History
Debug As
Debug Configurations...
Toggle Breakpoint Ctrl+Shift+B
Toggle Line Breakpoint
Toggle Method Breakpoint
Toggle Watchpoint
Skip All Breakpoints Ctrl+Alt+B
Remove All Breakpoints
Add Java Exception Breakpoint...
1 Run on Server Alt+Shift+X, R
2 Java Applet Alt+Shift+X, A
3 Java Application Alt+Shift+X, J
4 JUnit Test Alt+Shift+X, T
5 Maven build Alt+Shift+X, M
6 Maven build...
7 Maven clean
8 Maven generate-sources
9 Maven install
Maven test
Spring Boot App Alt+Shift+X, B
Spring Devtools Client

How to run the Address Book Client

We will not be making any changes to the AngularJS Address Book Client during the lab so we just need to run the client:

1. Open a command window
2. Browse to: **C:\momentum\documentum-springdata-lab\address-book-client**
3. Enter the following commands:
    a. **npm install**
    b. **bower install**
    c. **gulp default**
4. Open a browser and browse to **http://localhost:4000**

The Address Book Client will not yet be able to perform any actions (create, edit, etc), we will implement these features in the upcoming rounds.

Note: if you are interested in the code then take a look in **C:\momentum\documentum-springdata-lab\address-book-client\src**.

If you make any changes ☺ then as long as gulp is running you just need to refresh your browser to see your change in action.

Round 1: Create Contact Repository

Overview

In this round, we will get the basic Spring Data Repository defined that will allow for performing basic actions on our Contact objects (Save, Read, Delete, Set Content, etc).

We already have a Contact domain class defined (**com.emc.documentum.sample.domain.Contact**) defined, this Contact class is used to represent our repository contact object by mapping fields to Documentum attributes by name and through the use of Documentum Spring Data annotations. The mapping of the Class to Documentum Object Type is shown below:

Contact Class fields	contact Documentum Object Type properties
String id	r_object_id (ID)
String name	object_name (String 255)
String email	email (String 255)
String telephone	telephone (String 32)
List<String> groups	groups (Repeating String 255)
Integer contentSize	r_content_size (Integer)

Note how we are mapping the Class Name to the object type using the following annotation:

```
=====================================CODE BLOCK====================================
@DctmEntity(repository = “contact”)
==================================================================================
```

Also note how we are mapping fields in the class where its name does not match the Documentum object type property name and the ID attribute:

```
=====================================CODE BLOCK====================================
@Id
protected String id;
@DctmAttribute(value="object_name")
private String name;
==================================================================================
```

Get the code:

If you have followed the Project Setup instructions then you are all set.

Step 1:

Create an Interface (right click **src/main/java** => New => Interface):

Interface Name	Package	Extends
ContactRepository	com.emc.documentum.sample.repositories	DctmRepositoryWithContent<Contact, String>

Answer:

```
package com.emc.documentum.sample.repositories;

import com.emc.documentum.sample.domain.Contact;
import com.emc.documentum.springdata.repository.DctmRepositoryWithContent;

public interface ContactRepository extends DctmRepositoryWithContent<Contact, String> {
}
```

Step 2:

Tell the Spring Boot Application that we are going to be using Documentum Spring Data:

1. Add the following annotation to the **com.emc.documentum.sample.Application** class:

```
====================================CODE BLOCK=====================================

@EnableDctmRepositories

==================================================================================
```

Answer:

```
@Configuration
@ComponentScan(basePackages = {"com.emc.documentum.springdata", "com.emc.documentum.sample"})
@EnableAutoConfiguration
@EnableDctmRepositories
@EnableWebMvc
public class Application {
```

Step 3:

Enable the predefined junit **ContactRepositoryTest** Integration Test

1. Browse to and open **com.emc.documentum.sample.repositories.ContactRepositoryTest**
2. Add the following code:

```
====================================CODEBLOCK=====================================

@Autowired

private ContactRepository contactRepository;

==================================================================================
```

Answer:

```
@Configuration
@ComponentScan(basePackages = {"com.emc.documentum.springdata", "com.emc.documentum.sample"})
@EnableAutoConfiguration
@EnableDctmRepositories
@EnableWebMvc
public class Application {
```

3. Uncomment the following test methods:
   a. createContact()
   b. findContactById()
   c. deleteContact()
   d. updateContact()
   e. setAndRetrieveContactPicture()

Hint:

Select the lines that are commented and press CTRL + /

Step 4:

Run the predefined junit **ContactRepositoryTest** Integration Test class to check everything is working:

1. Browse to **com.emc.documentum.sample.repositories.ContactRepositoryTest**
2. Right click => Run As => Junit Test

Round 2: Display Contacts

Overview:

In this round, we will update Contact Spring Data Repository to allow us to get all of our contacts, we will then wire up the REST controller so that the Client Application can display all of the contacts.

Get the code (optional):

1. Open a command window
2. Browse to **C:\momentum\documentum-springdata-lab**
3. Run: **git checkout –B round2 origin/round2 –f** **(note: this will override any changes you have already made)**
4. Refresh your project in Spring Tool Suite

Step 1:

Enable retrieval of all contacts via the **ContactRepository**:

1. Add the following method declaration to the **com.emc.documentum.sample.repositories.ContactRepository** interface

```
====================================CODE BLOCK====================================
@Override

@Query("select r_object_id, object_name, email, telephone, groups, r_content_size
from contact")

public Iterable<Contact> findAll();

==================================================================================
```

Note: You might wonder why we are having to override the OOTB findAll() method with our own? This is due to a bug at time of writing this lab. More information for those who are interested: because we are retrieving properties that would not normally be fetched by a **select *** DQL query we need to explicitly define the query to be executed otherwise the properties we need to create our Contact class will not be returned (specifically groups and r_content_size). This issue can be resolved by dynamically building the query using reflection of the domain class.

Step 2:

Enable the predefined junit **ContactRepositoryTest** Integration Test

1. Browse to and open **com.emc.documentum.sample.repositories.ContactRepositoryTest**
2. Uncomment the following test methods:
    a. findAllContacts()

Hint:

Select the lines that are commented and press CTRL + /

Step 3:

Run the predefined junit **ContactRepositoryTest** Integration Test class to check everything is working:

1. Browse to **com.emc.documentum.sample.repositories.ContactRepositoryTest**
2. Right click => Run ContactRepositoryTest

Step 4:

Give the REST controller access to the ContactRepository:

1. Browse to and open **com.emc.documentum.sample.controller.ContactController**
2. Add the following member variable code:

```
====================================CODEBLOCK====================================

@Autowired

private ContactRepository contactRepository;

=================================================================================
```

Step 5:

Complete the **getAllContacts()** method in **com.emc.documentum.sample.controller.ContactController** to allow the Web API to return all contacts to the UI. HINT:.

Hint:

Take a look at the **findAllContacts()** test code.

Answer:

```
Iterable<Contact> contacts = contactRepository.findAll();
return contacts;
```

Step 5:

Ensure your Address Book Client UI application and API server are running (see section Project Setup) and refresh http://localhost:4000 to see all of the contacts in the repository.

Round 3: Create, Edit and Delete a Contact

Overview:

In this round, we will then wire up the REST controller so that the Client Application can create, edit and delete a contact.

Get the code (optional):

1. Open a command window
2. Browse to **C:\momentum\documentum-springdata-lab**
3. Run: **git checkout –B round3 origin/round3 –f** **(note: this will override any changes you have already made)**
4. Refresh your project in Spring Tool Suite

Step 1:

Complete the **createContact()** method in **com.emc.documentum.sample.controller.ContactController** to allow the Web API to create a new contact.

Hint:

Take a look at the **createContact()** test.

Answer:

```
// make sure no ID is set so that new object is created
contact.setId(null);

// save the new contact
contactRepository.save(contact);

return contact;
```

Step 2:

Complete the **updateContact()** method in **com.emc.documentum.sample.controller.ContactController** to allow the Web API to edit an existing contact.

Hint:

Take a look at the **updateContact()** test code.

Answer:

```
// ensure the contact ID is correctly set
contact.setId(id);

// update the contact
return contactRepository.save(contact);
```

Step 3:

Complete the **deleteContact()** method in **com.emc.documentum.sample.controller.ContactController** to allow the Web API to delete an existing contact.

Hint:

Take a look at the **deleteContact()** test code.

Answer:

```
// delete the contact
contactRepository.delete(id);
```

Step 4:

Ensure your Address Book Client UI application and API server are running (see section Project Setup) and refresh http://localhost:4000 to test create, edit and delete contacts.

Round 4: Set and Retrieve a Contact Picture

Overview:

In this round, we will then wire up the REST controller so that the Client Application can upload and display a contact png picture. Behind the scenes Documentum Spring Data will be storing the picture as the content of the contact sys object.

NOTE: a sample png can be found in **C:\momentum**

Get the code (optional):

1. Open a command window
2. Browse to **C:\momentum\documentum-springdata-lab**
3. Run: **git checkout –B round4 origin/round4 –f** **(note: this will override any changes you have already made)**
4. Refresh your project in Spring Tool Suite

Step 1:

Complete the **setContactPicture()** method in **com.emc.documentum.sample.controller.ContactController** to allow the Web API to upload a contact picture.

Hint:

Take a look at the **setAndRetrieveContactPicture ()** test code.

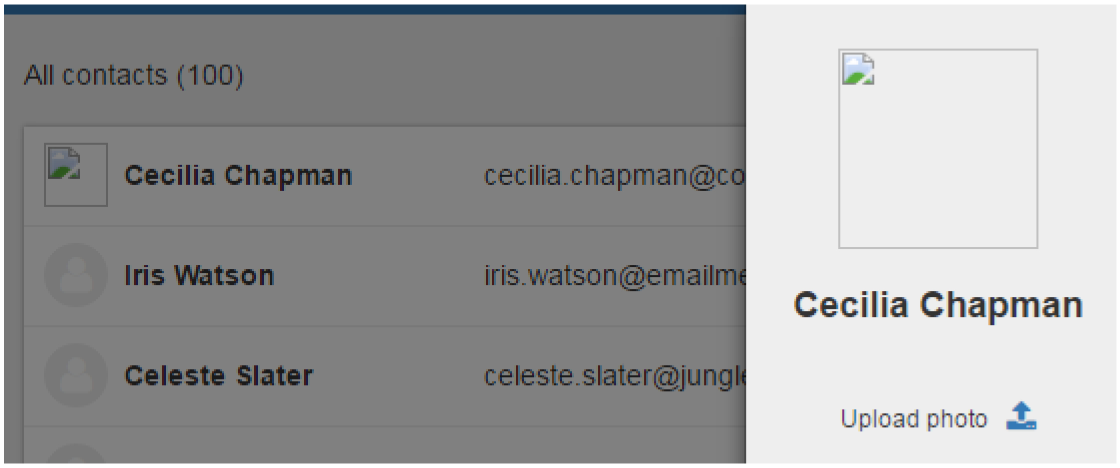
Answer:

```
Path tempFile = null;
try {
        // create a temp file to hold the picture
        tempFile = Files.createTempFile(null, null);

        // copy the picture to the temp file
        file.transferTo(tempFile.toFile());

        // get the contact
        Contact contact = contactRepository.findOne(id);

        // get the file extension
        String fileExtension = StringUtils.getFilenameExtension(file.getOriginalFilename());

        // set the picture as the contact content
        contactRepository.setContent(contact, fileExtension, tempFile.toString());

} finally {
        // clean up the temp file
        FileSystemUtils.deleteRecursively(tempFile.toFile());
}
```

Step 2:

Ensure your Address Book Client UI application and API server are running (see section Project Setup) and refresh http://localhost:4000 to test upload a contact png picture.

You will know that the upload has been successful when you see a broken picture, we fill fix this in the next step.



You can also browse to your contact using DA to see that the content of the object has been set. You will find the contacts in your users cabinet (dmadmin).

Step 3:

Complete the **getContactPicture()** method in **com.emc.documentum.sample.controller.ContactController** to allow the Web API to retrieve a contact picture.

Hint:

Take a look at the **setAndRetrieveContactPicture ()** test code.

Answer:

```
Path tempDir = null;

try {
        // create a temp dir to hold the picture
         tempDir = Files.createTempDirectory(null);

        // get the contact
        Contact contact = contactRepository.findOne(id);

        // create a temp file name
        String tempFileName = tempDir.toString() + File.separator + UUID.randomUUID();

        // read the contact picture into the temp dir
        String picturePath = contactRepository.getContent(contact, tempFileName);

        // copy the file content to the response output stream
        FileInputStream fileInputStream = new FileInputStream(picturePath);
        IOUtils.copy(fileInputStream, response.getOutputStream());
        fileInputStream.close();

} finally {
        // clean up the temp file and dir
        FileSystemUtils.deleteRecursively(tempDir.toFile());
}
```

Step 4:

Ensure your Address Book Client UI application and API server are running (see section Project Setup) and refresh http://localhost:4000 to check that your contact pictures are now being retrieved.

Round 5: Search for Contacts

Overview:

In this round, we will add a new query to the Contact Spring Data Repository to allow us to search for contacts where the name contains a given value. We will also modify the REST controller so that we can search for contacts by name as well as finding all contacts.

Get the code (optional):

1. Open a command window
2. Browse to **C:\momentum\documentum-springdata-lab**
3. Run: **git checkout –B round5 origin/round5 –f** **(note: this will override any changes you have already made)**
4. Refresh your project in Spring Tool Suite

Step 1:

Add a new findByNameContaining(String) method to the **ContactRepository** that will enable querying for any contact where the object_name is like the value provided.

Hint:

To escape the percentage character you use another percentage character (%%).

Answer:

```
@Query("select r_object_id, object_name, email, telephone, groups, r_content_size from contact
where object_name like \'%%%s%%\'")
public Iterable<Contact> findByNameContaining(String value);
```

Step 2:

Enable the predefined junit **ContactRepositoryTest** Integration Test

1. Browse to and open **com.emc.documentum.sample.repositories.ContactRepositoryTest**
2. Uncomment the following test methods:
   a. findContactByNameContaining()

Hint:

Select the lines that are commented and press CTRL + /

Step 3:

Run the predefined junit **ContactRepositoryTest** Integration Test class to check everything is working:

1. Browse to **com.emc.documentum.sample.repositories.ContactRepositoryTest**
2. Right click => Run ContactRepositoryTest

Step 4:

Modify the **getAllContacts()** method in **com.emc.documentum.sample.controller.ContactController** to allow the Web API to find contacts by name.

Hint:

Take a look at the **findContactByNameContaining ()** test code.

Answer:

```
Iterable<Contact> contacts = null;

/*
 * if the name URI param is supplied then only get contacts where the name contains the
 * value in the URI param
 */
if(StringUtils.hasText(name)) {
        contacts = contactRepository.findByNameContaining(name);
} else {
        contacts = contactRepository.findAll();
}

return contacts;
```

Step 5:

Ensure your Address Book Client UI application and API server are running (see section Project Setup) and refresh http://localhost:4000 to check that you can search for contacts.

Round 6: Filter Contacts by group

Overview:

In this round, we will add a new query to the Contact Spring Data Repository to allow us to filter for contacts by group. We will also modify the REST controller so that we can find contacts by group as well as finding all contacts and searching contacts by name.

Get the code (optional):

1. Open a command window
2. Browse to **C:\momentum\documentum-springdata-lab**
3. Run: **git checkout –B round6 origin/round6 –f** **(note: this will override any changes you have already made)**
4. Refresh your project in Spring Tool Suite

Step 1:

Add a new findByGroups(String) method to the **ContactRepository** that will enable querying for any contact where any group is equal to the value provided.

Answer:

```
@Query("select r_object_id, object_name, email, telephone, groups, r_content_size from contact
where any groups = \'%s\'")
public Iterable<Contact> findByGroups(String value);
```

Step 2:

Enable the predefined junit **ContactRepositoryTest** Integration Test

1. Browse to and open **com.emc.documentum.sample.repositories.ContactRepositoryTest**
2. Uncomment the following test methods:
   a. findContactByGroup()

Hint:

Select the lines that are commented and press CTRL + /

Step 3:

Run the predefined junit **ContactRepositoryTest** Integration Test class to check everything is working:

1. Browse to **com.emc.documentum.sample.repositories.ContactRepositoryTest**
2. Right click => Run ContactRepositoryTest

Step 4:

Modify the **getAllContacts()** method in **com.emc.documentum.sample.controller.ContactController** to allow the Web API to find contacts by group.

Hint:

Take a look at the **findContactByGroup ()** test code.

Answer:

```
Iterable<Contact> contacts = null;

/*
 * if the name URI param is supplied then only get contacts where the name contains the
 * value in the URI param, else if the group URI param is supplied then only get contacts
 * that are part of the group specified by the group URI param
 */
if(StringUtils.hasText(name)) {
	contacts = contactRepository.findByNameContaining(name);
} else if(StringUtils.hasText(group)) {
	contacts = contactRepository.findByGroups(group);
} else {
	contacts = contactRepository.findAll();
}

return contacts;
```

Step 5:

Ensure your Address Book Client UI application and API server are running (see section Project Setup) and refresh http://localhost:4000 to check that you can filter contacts by group.

Round 7: Review the complete project

Overview

That's all for your tasks. Now let's get the complete project so you can compare what you did.

Get the code:

1. Open a command window
2. Browse to **C:\momentum\documentum-springdata-lab**
3. Run: **git checkout –B master origin/master –f** **(note: this will override any changes you have already made)**
4. Refresh your project in Spring Tool Suite

Summary

This sample project and its tutorials is available on GitHub: https://github.com/Enterprise-Content-Management/documentum-springdata-lab. To learn more about Documentum Spring Data, please reach out to **Documentum@GitHub** and **Documentum@EMC Develop Network**.