DOCUMENTUM REST LAB

Table of Contents

Overview 5
Welcome to the Documentum REST Lab 5
Introducing the Documentum AngularJS File Manager application 5
Section 1 Lab Introduction 6
Documentum REST Services overview 6
Virtual machine 6
Technology stack 6
Spring Boot 6
Spring RestTemplate 6
Npm 7
Bower 7
Gulp 7
AngularJS 7
Development approach in this lab 7
EMC Documentum @GitHub 7
Section 2 File Manager Development 9
Round 1: Project setup 9
Documentum Content Server 9
Documentum REST Server 9
Documentum Administrator 9
Get the code 9
Build from CLI 10
Import into IDE 11
Troubleshooting 11
Summary 12
Round 2: Folder navigation 13
Overview 13
Get the code 13
Step 1: file-manager-ui 13

Step 2: file-manager-api 14
Step 3: test 18
Summary 19
Round 3: Create a new folder 20
Overview 20
Get the code 20
Step 1: file-manager-ui 20
Step 2: file-manager-api 21
Step 3: test 23
Summary 24
Round 4: Upload content 25
Overview 25
Get the code 25
file-manager-api 25
Step 3: test 27
Summary 27
Round 5: Edit content 28
Overview 28
Get the code 28
Step 1: file-manager-ui 29
Step 2: file-manager-api 29
Step 3: test 30
Summary 31
Round 6: Full-text search 32
Overview 32
Get the code 32
file-manager-api 32
Step 3: test 33
Summary 34
Round 7: Review the complete project 35
Overview 35

Get the code 35
Summary 35
Cheatsheet 36

Overview

Welcome to the Documentum REST Lab

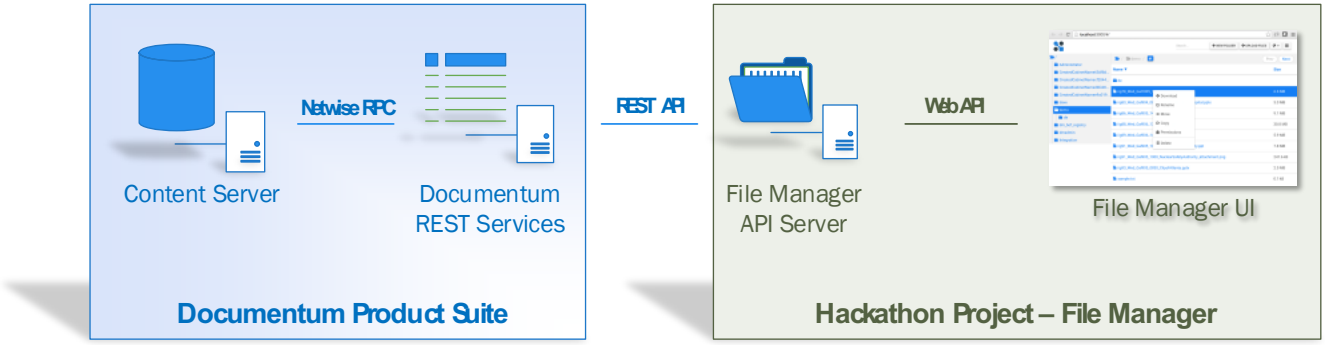
This EMC lab will walk you through some of the common challenges for developing a responsive front-end web application for common content management functionalities using the REST API at the backend.

In this lab you will learn how EMC Documentum REST Services quickly helps to model your common content management functions in an AngularJS application and gives you the easy access to Documentum content management system with the hypermedia driven API service.

Introducing the Documentum AngularJS File Manager application

In this Hackathon lab, we will buid a Documentum AngularJS File Manager application. Folder and document operations are the key abaility of any enterprise content managerment system. Documentum AngularJS File Manager is a very simple and intuitive web application which provides the functions to manage Documentum repository folders and contents through a modernized web application. As you get from its name, its front-end mainly leverages AngularJS to build the application. This project's goal is to demonstrate how to use Documentum REST Services to develop modern front-end Single Page Applications.

The Documentum AngularJS File Manager application consists of two sub projects, the back-end **File Manager API Server** and the front-end **File Manager UI**.



File Manager API Serer

File Manager API Server is a middleware Web API server built with Spring Boot. To the back-end, it interacts with Content Server repositories for folder and document operations using Documentum REST Services. To the front-end, it provides a set of File Manager APIs to the AngularJS application, in common HTTP endpoints and JSON format.

File Manager UI

File Manager UI is a Single Page Application built with AngularJS and Material Design. In this sample project, we leverage the GitHub open source project Angular File Manager to build the main UI components of the folder and document operations. We also make customizations for API calls, as well as some specific functions like full-text search and PDF view.

Section 1 Lab Introduction

Documentum REST Services overview

Documentum REST Services is a set of simple yet powerful hypermedia driven REST services which provide API access to Documentum Content Server repositories. Documentum REST Services supports a lot of service operations, while in this lab, we will only cover a number of common folder and document operations. Documentum REST Services also supports a rich set of authentication schemes, and in this lab, we only use the simplest one, the HTTP Basic authentication. Documentum REST Services supports both XML and JSON formats for its resource representations. JSON is simper in general usage so we consume Documentum REST Services with JSON format in this lab.

Virtual machine

Everyone will a virtual machine for this lab. The virtual machines contains all the softwares to develop and run the demo project.

Tip

You don't need to install anyting on your own laptop, unless you do not have the Remote Desktop software installed yet. All tasks in this lab shall be finished in the virtual machine.

Technology stack

Spring Boot

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run" it as a single application without explicit deployment. Spring Boot manages most of the 3rd party dependencies for you so that you need to do little with the configuration.

Spring RestTemplate

Spring RestTemplate makes it easy for you to consume RESTful Services where you can plugin in your customizations for message convertions, authentications, and so on.

Npm

Node.js is an asynchronous event driven framework designed to build scalable network applications. Node.js has its own package manager to install modules called npm. Npm makes it easy for JavaScript developers to share and reuse code, and it makes it easy to update the code that you're sharing.

A package can be downloaded with the command npm install <package>.

Bower

Bower is your front-end package manager which maintains a flat list of dependencies for your front-end applications, placing the burden on you to deal with version conflicts.

Bower is installed by npm with the command npm install –g bower. Then you can use bower to install your front-end packages, with the command bower install <package>.

Gulp

Gulp is a toolkit to build JavaScript (and other) applications with tasks. It's streaming and code-over-configuration.

Gulp is installed by npm with the command npm install –g gulp. Then you can use gulp to build your front-end application with tasks, using the command gulp <task>.

AngularJS

AngularJS is a JavaScript Model-View-Whatever (MVW) framework. The AngularJS framework works by first reading the HTML page, which has embedded into it additional custom tag attributes. Angular interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables, then can be managed by JavaScript codes.

Development approach in this lab

The AngularJS application is ready for you to use. The approach we take when building the API application is to start with interfaces and then gradually add implementations. We'll describe all the steps in some detail below.

The hackathon will consist of a number of rounds of coding. After each round, we'll stop coding and take some time to reflect on what we've done so far. If you haven't made as much progress as you would have liked in a round, you can catch up in between rounds, because we provide all the code in GitHub.

For tasks that require your coding actions, you will see this hint before the code "Your code here".

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/emcworld2016-hackathon-documentum-rest

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 on the same page as everybody else. You get the code in a branch using the command

git checkout -B <branch_name> origin/<branch_name> –f

Section 2 File Manager Development

Round 1: Project setup

In the first round, we will check the Documentum software set up and build the initial project in the virtual machine. You shall now need to remote desktop connect to your virtual machine.

Documentum Content Server

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

Documentum REST Server

Documentum REST Services 7.2 Patch 12 is deployed to the Tomcat 7 server at C:\Tomcat7i2\webapps\dctm-rest. It's started up by default. If not, please double click the Startup.bat on your VM's desktop or navitaget to C:\Tomcat7i2\bin\startup.bat to start it up. The HTTP port for the REST services is 8083, so the welcome page for Documentum REST Services is http://localhost:8083/dctm-rest. You can have a try in the web browser.

Documentum Administrator

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!.

Get the code

Each round in the lab 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:\emcworld**
3. Run command: **git clone https://github.com/Enterprise-Content-Management/emcworld2016-hackathon-documentum-rest.git -b round1**
4. Now the project shall be downloaded to location **C:\emcworld\emcworld2016-hackathon-documentum-rest**

Tip

Just for shortening in the below paragraph, we will use ***<root>*** *to stand for the file location* ***C:\emcworld\emcworld2016-hackathon-documentum-rest***.

Build from CLI

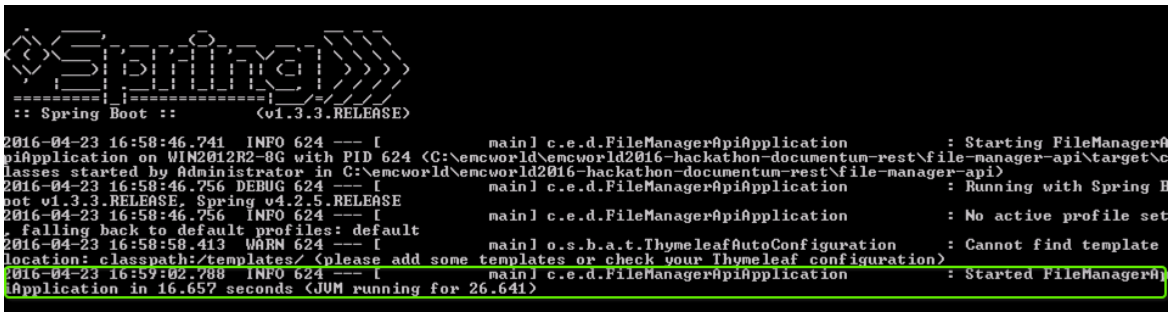
The project contains two sub projects, built with different technologies.

- file-manager-api, a spring-boot project built with Java and Maven
- file-manager-ui, an AngularJS project built with Npm, Bower and Gulp

The build environement has been set for you. Let's just go ahead to build the project.

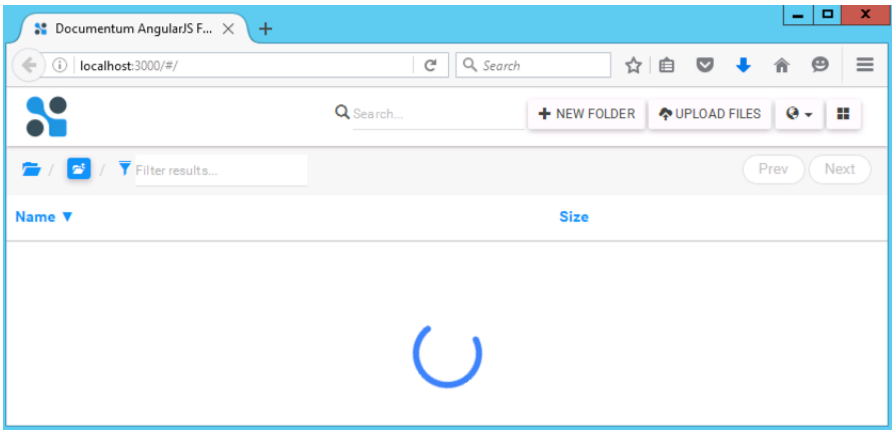
Firstly, we will build the file-manager-api project.

1. Open a Command window
2. Navigate to **<root>\file-manager-api**
3. Run command: **mvn spring-boot:run**
4. At the bottom, you will see the spring-boot project file-manager-api is up and running.



Next, we will build the file-manager-ui project.

1. Open a new Command window
2. Navigate to **<root>\file-manager-ui**
3. Run command: **npm install –save-dev**
   Tip**: If npm runs into errors, please try **npm cache clean** and install again. If it does not work, you can try to manually remove folder **C:\Users\dmadmin\AppData\Roaming\npm-cache
4. After it finishes, run command: **bower install**
5. After it finishes, run command: **gulp serve**
6. If everything goes well, the app will be promoted in the default web browser at location: **http://localhost:3000/#**



By above steps, the project File Manager has been built and run successfully. But you can do nothing with it because functions are not implemented yet.

Import into IDE

As later we will modify the project, we need to import the project into an IDE. In this lab, we will work on Spring Tool Suite (STS).

1. Open Spring Tool Suite. We will first import the project file-manager-api.
2. Click on **File** > **Import…** > **Existing Maven Projects** > **Next**
3. Browse to the location of the project: **<root>\file-manager-api**
4. Click on **Finish**.
   Spring Tool Suite recognizes this project as a Spring Boot project and compiles the code automatically.

Tip

The other option for you to import the project is to use Jetbrians Intellij, which is on your desktop.

5. Next, we will load the project file-manager-ui to Spring Tool Suite.
6. On Spring Tool Suite, **right click** on the **Project Explorer** at left side > **New** > **Project…** > **Web** > **Static Web Project**.
7. Input **Project name**: **file-manager-ui**
8. Uncheck **Use default location**
9. Browse to the location of the project: **<root>\file-manager-ui**
10. Click on **Finish**.
    You cannot build it from Spring Tool Suite because there are no plugins installed for Node.js and Gulp installed.

Tip

The other option for you to import the project is to use Jetbrians WebStorm, which is on your desktop.

Now let's take a few minutes to navigate the project structure. *<<live>>*

Troubleshooting

Documentum

To do parallel comparasion, you can always check the folder and document update from DA. Whatever you do with File Manager can be done by DA, too.

file-manager-api

For the file-manager-api project, you can configure its logging scope and level to include more logging messages, where the configuration is at **<root>\file-manager-ui\file-manager-api\src\main\resources\application.properties**. The default logging is:

```
logging.level.root=WARN
```

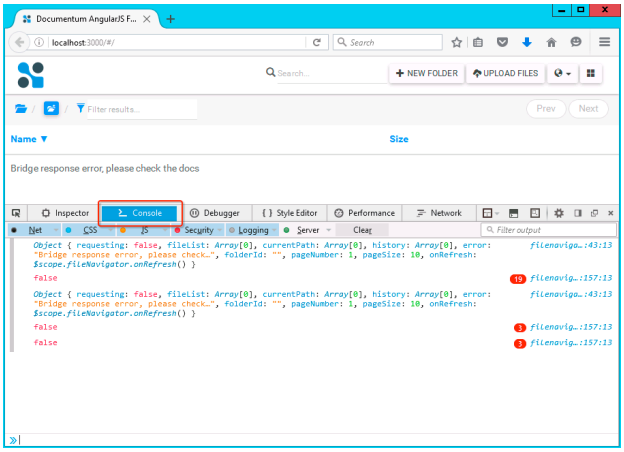
```
logging.level.com.emc.documentum=DEBUG
```

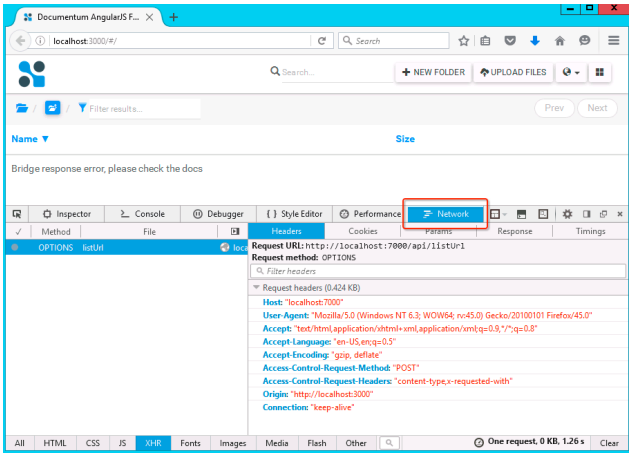
If you changed the application properties, you need to restart the spring-boot.

file-manager-ui

For the file-manager-ui project, you can use the browser's dev tool to debug the JS code. Here is the sample for Firefox web browser.

1. **Right Click** on the page of File manager app, and **open Inspect Element**.
2. We can observe every piece of the HTML/JS elements from the inspector. In this lab, we mainly watch on **Console** and **Network** tabs. Please see samples below.





If necessary, we can add debug breakpoints on the project code to run code step by step, for Java and JS, respectively.

Note – you don't need to restart project file-manager-ui as **gulp** automatically monitors the local changes

Summary

In this round, we get the code for Documentum AngularJS File Manager, build it and run it in two separate ports.

- File Manager API Server
  - Command to start: <root>\file-manager-api\mvn spring-boot:run
  - Port: http://localhost:7000
- File Manager UI
  - Command to start <root>\file-manager-ui\gulp serve
  - Port: http://localhost:3000

Round 2: Folder navigation

Overview

In this round, we will learn how to list cabinets and folders under a repository.

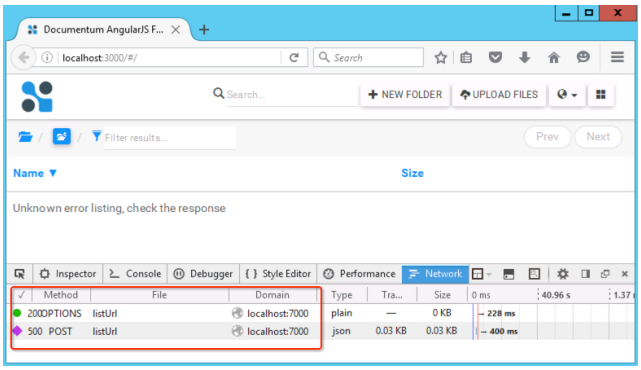
Get the code

We'll add code base on round 1. So at this moment, you don't need to refetch code from server. If you haven't downloaded the code, please refer to **Get the code** in Round 1.

Step 1: file-manager-ui

Now let's examine why the File Manager UI does not show anything on the page.

(1) Open the browser **Inspect Element/Network** as the section **Troubleshooting** shows you. You can refresh the page, and you will see that there are XHR (HTTP) requests sent to the File Manager API Server (URL pattern like **/api/...**) but the response were errors.



(2) Open source code **<root>/file-manager-ui/src/app/file/manager/controllers/filemanager.controller.js**
This is the main AngularJS controller for the this app. At the bottom you will see this code:

```
$scope.fileNavigator.refresh();
```

It indicates that an AngularJS service filenavigator takes responsibility to refresh the page.

(3) Open source code Open source code **<root>/file-manager-ui/src/app/filemanager/services/filenavigator.js**
Locate to the function **refresh**. You'll see that this function builds the folder tree based on the current path.

```
FileNavigator.prototype.refresh = function() {
    var self = this;
    var path = self.currentPath.join('/');
    return self.list().then(function(data) {
        self.fileList = (data.result || []).map(function(file) {
            return new Item(file, self.currentPath);
        });
        self.buildTree(path);
    });
};
```

a

b

Code Explanation

a. List folder children for current path (HTTP call)
b. Build folder hiearchy models and views in HTML

It depends on the function **list** to get data.

```
FileNavigator.prototype.list = function() {
    …

    var requestUrl = fileManagerConfig.listUrl;
    var data = {
        action: 'list',
        path: '/' + path,
        id: this.folderId,
        params: {
            onlyFolders: false,
            pageNumber: this.pageNumber,
            pageSize: this.pageSize
        }
    };

    $http.post(requestUrl , data).success(function(data) {…});
    return deferred.promise;
};
```

a
b

Code Explanation

a. listUrl is configured from **<root>/file-manager-ui/src/app/filemanager/providers/config.js**
b. The data is used as request payload in JSON format, with below info:

```
{
  “action”: “list”,
  “path”: “/”,
  “id”: “”,
  “params”: {
    “onlyFolders”: false,
    “pageNumber”: 1,
    “pageSize”: 20
  }
}
```

Now we get to know that the function **list** will send a **HTTP POST** method to the URI **<file-manager-api>/api/listUrl** with a JSON request body. It expects a list of cabinets, folders or objects to be returned in the response, based on the current path.

We also get to know that the server does not return the right response.

Step 2: file-manager-api

Now let’s examine where the API endpoints are in the project file-manager-api.

(1) Let's first look at a test sample. Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/filemanager/controller/FileManagerController.java** You'll see this is a typical Spring REST controller. This controller provide all HTTP endpoints for File Manager API. There has a sample endpoint defined in the class.

```
@Autowired
FileManagerApi fileManagerApi;

@RequestMapping(value = "/about",
        method = RequestMethod.GET,
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
public Item about() throws DocumentumException {..}
```

a b c

Code Explanation

a. URL endpoint for this service
b. HTTP method for this service
c. Response media type for this service

When your File Manager API Server is up, you can test this endpoint by URL **http://localhost:7000/api/about**

(2) There is another method **listObjects** in this controller. This endpoint provides the File Manager UI with the web API to list cabinets and objects.

```
@RequestMapping(value = "/listUrl",
        method = RequestMethod.POST,
        consumes = MediaType.APPLICATION_JSON_VALUE,
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
public Collection listObjects(@RequestBody BaseRequest request) {..}
```

a

Code Exaplantion

a. This end point accepts a POST method on the URL "/listUrl".

This controller itself is not Documentum REST API, but instead, it calls the fileManagerApi to get a page of cabinets or folder children. It's right – fileManagerAPI calls Documentum REST Services to get the objects.

Let's just do a quick overview of these classes as you will come back here several more times for next rounds of practices. *<<live>*

(3) Open package **<root>/file-manager-api/src/main/java/com/emc/documentum/restclient**. Please note the two classes in the **restclient** package.

- **DctmRestClient** – this is the Documentum REST Client where we will add codes to complete the REST calls. We will follow hypermedia representations of the REST Services to do operations.

- **DctmRestTemplate** – this is the lower level HTTP client to invoke HTTP requests by using Spring RestTemplate.

Now we are going to implement the class **DctmRestClient**. The first is to initialize the Spring RestTemplate clients and get Documentum repository in memory.

Your code here – complete the method afterProperties in DctmRestClient.java

```
@Override
public void afterPropertiesSet() throws Exception {
    restTemplate = new DctmRestTemplate(data.username, data.password, false);
    streamingTemplate = new DctmRestTemplate(data.username, data.password, true);

    // get home doc
    ResponseEntity<Map> homedoc = restTemplate.get(data.contextRootUri + "/services",
Map.class);
    Map rootResources = (Map) homedoc.getBody().get("resources");

    //TODO FOR ROUND 2 -- BEGIN
    //TODO FOR ROUND 2 -- RESOLVE 'repositoriesUri', FROM 'rootResources'
    String repositoriesUri = "";

    //TODO FOR ROUND 2 -- UNCOMMENT BELOW
//        // get repositories
//        ResponseEntity<JsonFeed> repositories = restTemplate
//                .get(repositoriesUri, JsonFeed.class, QueryParams.INLINE, "true");
//        for (JsonEntry repo : repositories.getBody().getEntries()) {
//            if (data.repo.equals(repo.getTitle())) {
//                repository = repo.getContentObject();
//                break;
//            }
//        }
    //TODO FOR ROUND 2 -- UNCOMMENT ABOVE
    //TODO FOR ROUND 2 -- END
}
```

a b c d e

Code Exaplantion

a. Initialize a Spring RestTemplate client used for REST calls to Documentum REST Server
b. Initialize a streaming client specifically used for content upload and download
c. Get home document resource from Documentum REST Server
   The only service that requires a hard-coded URI
d. Your task here – find the URL for repositories feed
e. Uncomment the code to get repositories feed

Answer

```
//CODE FOR ROUND 2 -- BEGIN
//CODE FOR ROUND 2 -- RESOLVE 'repositoriesUri', FROM 'rootResources'
Map repositoriesEntry = (Map) rootResources.get(LinkRelation.REPOSITORIES);
String repositoriesUri = (String) repositoriesEntry.get("href");
//CODE FOR ROUND 2 -- END
```

a

Code Explanation

a. Repositories resource can be discovered from the link relation "**http://http://identifiers.emc.com/linkrel/repositories**"

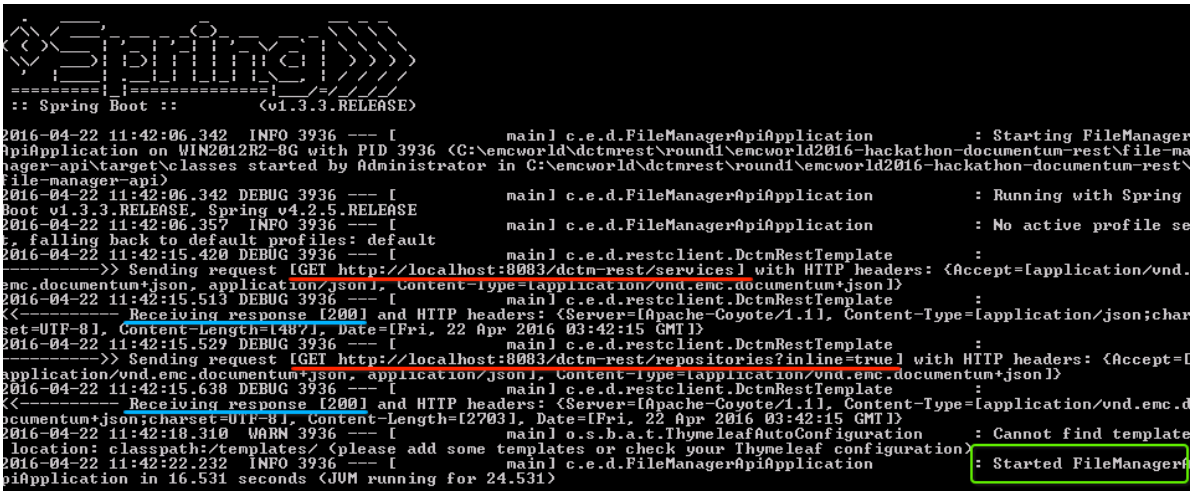
Tip:

*Let Spring Tool Suite to resolve the Java packages for unknow classes. Shortcut **Ctrl + Shift + O**.*

(4) Now let's restart the File Manager API Server, you will see some difference from the console.

1. Press **ctrl + c** on the Command window where project **file-manager-api** is running
2. Press **y** to terminate the batch
3. Run command: **mvn spring-boot:run**

You will see outputs from the Command window console.



But wait – we want to implement the function to navigate cabinets and objects, right? Yes – its our next step.

(5) On the source code **DctmRestClient.java**, there are two methods **getAllCabinets()** and **getChildren()** waiting for your compeletion.

Your code here – complete methods getAllCabinets() and getChildren() in DtmRestClient.java

```
public List<JsonEntry> getAllCabinets(int pageNumber, int pageSize) {
    //TODO FOR ROUND 2 -- BEGIN
    //TODO FOR ROUND 2 -- RESOLVE 'cabinetsUrl', FROM 'repository'
    String cabinetsUrl = "";
    //TODO FOR ROUND 2 -- END

    return getJsonEntriesByUrl(pageNumber, pageSize, cabinetsUrl);
}

public List<JsonEntry> getChildren(String path, int pageNumber, int pageSize) {
    JsonObject folder = getObjectByPath(path);

    //TODO FOR ROUND 2 -- BEGIN
    //TODO FOR ROUND 2 -- RESOLVE 'childrenUrl', FROM 'folder'
    String childrenUrl = "";
    //TODO FOR ROUND 2 -- END

    return getJsonEntriesByUrl(pageNumber, pageSize, childrenUrl);
}
```

Answer

```
//CODE FOR ROUND 2 -- BEGIN
//CODE FOR ROUND 2 -- RESOLVE 'cabinetsUrl', FROM 'repository'
String cabinetsUrl = repository.getHref(LinkRelation.CABINETS);   a
//CODE FOR ROUND 2 -- END

//CODE FOR ROUND 2 -- BEGIN
//CODE FOR ROUND 2 -- RESOLVE 'childrenUrl', FROM 'folder'
String childrenUrl = folder.getHref(LinkRelation.OBJECTS);   b
//CODE FOR ROUND 2 -- END
```

Code Explanation

a. Cabinets resource can be discovered from the link relation "**http://http://identifiers.emc.com/linkrel/repositories**" of repository

b. Folder children resource can be discovered from the link relation "**http://http://identifiers.emc.com/linkrel/objects**" of folder

Step 3: test

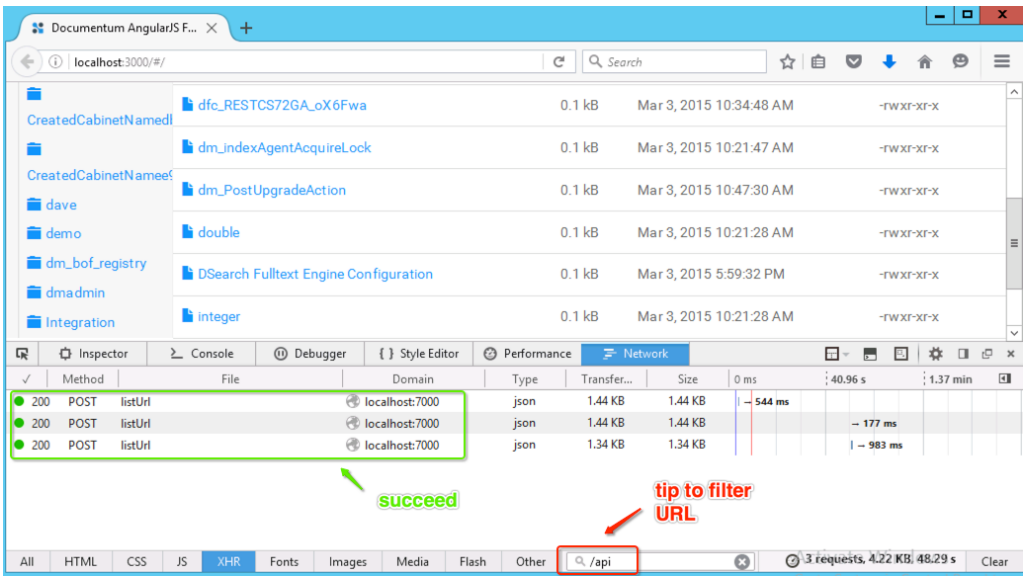
With all above work, we are done with the task Folder Navigaiton. Let's restart the File Manager API Server and see the result.

(1) Press **ctrl + c** on the Command window where project **file-manager-api** is running

(2) Press **y** to terminate the batch

(3) Run command: **mvn spring-boot:run**

(4) Open Firefox web browser, refresh the page **http://localhost:3000**, you will see folders listed in the UI page.



Congratulation! You've completed the most tough task in this lab!

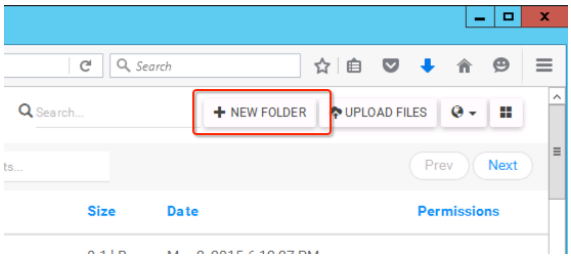
Summary

- File Manager UI sends XHR requests to File Manager API Server to get cabinets and folders with the pattern: POST /listUrl
- File Manager API Server starts from Home Document to discover resources in Documentum REST Services
- In Documentum REST Sevices, link relations are defined to discover resources.
    - home doc --> repositories: http://identifiers.emc.com/linkrel/repositories
    - repository --> cabinets: http://identifiers.emc.com/linkrel/cabinets
    - folder --> child objects: http://identifiers.emc.com/linkrel/objects

Round 3: Create a new folder

Overview

As you see on the UI, there is a button to create new folders on the right top of the page. We will implement this function in this round.



Get the code

1. Open a Command window
2. Browse to **C:\emcworld\emcworld2016-hackathon-documentum-rest**
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: file-manager-ui

We will start with the UI part to see how this is implemented in HTML/JS and what it expects from the server.

(1) Open source code **<root>/file-manager-ui/src/app/filemanager/templates/navbar.html**
You will notice the HTML tag to create the button for "New Folder".

```
<button class="btn btn-default btn-sm" ng-click="modal('newfolder') &&
prepareNewFolder()">
    <i class="glyphicon glyphicon-plus"></i> {{"new_folder" | translate}}
</button>
```

It indicates that the click on the button triggers to a new modal (template) "newfolder".

(2) Open source file **<root>/file-manager-ui/src/app/filemanager/templates/modals.html**
You will notice this HTML tag with the id "newfolder".

```
<div class="modal animated fadeIn" id="newfolder">
  <div class="modal-dialog">
    <div class="modal-content">
        <form ng-submit="createFolder()">
            …
```

There are 3 implementations of function **createFolder()**. They are called in this sequence: **filemanager.controller.js → apimiddleware.js → apihandler.js**.
Our task is to implement the XHR request in apihandler.js.

(3) Open source file **<root>/file-manager-ui/src/app/filemanager/services/apihander.js**, and navigate to this method. The function createFolder sends an XHR request to the server with a JSON payload.

```
ApiHandler.prototype.createFolder = function(apiUrl, item) {
    …

    var requestUrl = apiUrl;   a
    var data = {
        action: 'createFolder',   b
        newPath: item.path,
        name: item.tempModel.name,
        parentId: item.tempModel.id
    };

    $http.post(requestUrl, data).success(function(data) {…});

    return deferred.promise;
};
```

Code Explanation

a. apiUrl is configured from **<root>/file-manager-ui/src/app/filemanager/providers/config.js**

b. The data is used as request payload in JSON format, with below info:

```
{
  "action": "createFolder",
  "newPath": "",
  "name": "hackathon-demo",
  "parentId": ""
}
```

Step 2: file-manager-api

In File Manager API Server, we will need to complete the controller endpoint and the REST client method for folder creation.

(1) Open source code Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/filemanager/controller/FileManagerController.java** and you will see the method **createFolder**.

Your code here – complete method **createFolder** in **FileManagerController.java**

```
@RequestMapping(value = "/createFolderUrl",
        method = RequestMethod.POST,
        consumes = MediaType.APPLICATION_JSON_VALUE,
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
public CommonResult createFolder(@RequestBody CreateObjectRequest request) throws
DocumentumException {
     //TODO FOR ROUND 3 -- BEGIN
     //TODO FOR ROUND 3 -- CALL 'fileManagerApi' to create folder.
     //TODO FOR ROUND 3 -- END
     return successResponse();
}
```

Answer

```
//CODE FOR ROUND 3 -- BEGIN
//CODE FOR ROUND 3 -- CALL 'fileManagerApi' to create folder.
fileManagerApi.createFolderByParentId(request.getParentId(), request.getName());  a
//CODE FOR ROUND 3 -- END
```

Code Explantion

a. All folders must be created under a parent cabinet or folder, so the parent folder ID cannot be null. fileManagerApi finally calls DctmRestClient to create the folder.

(2) Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/filemanager/restclient/DctmRestClient.java** and implement the method **createFolder**.

Your code here – complete method **createFolder** in **DctmRestClient.java**

```
public JsonObject createFolder(String parentId, String folderName) {
    JsonObject parentFolder = getObjectById(parentId);  a

    //TODO FOR ROUND 3 -- BEGIN
    //TODO FOR ROUND 3 -- RESOLVE 'childFoldersUrl', FROM 'parentFolder'  b
    String childFoldersUrl = "";
    //TODO FOR ROUND3 -- END

    ResponseEntity<JsonObject> result = restTemplate.post(childFoldersUrl,…  c
}
```

Code Explanation

a. Get the folder resource first by id
b. Your task - Find the link relation for child object creation under the folder
c. Make a POST method to the URL for folder creation

Answer

```
//CODE FOR ROUND 3 -- BEGIN
//CODE FOR ROUND 3 -- RESOLVE 'childFoldersUrl', FROM 'parentFolder'
String childFoldersUrl = parentFolder.getHref(LinkRelation.FOLDERS);  a
//CODE FOR ROUND3 -- END
```

a. Folder child folders resource can be discovered from the link relation "**http://http://identifiers.emc.com/linkrel/folders**" of folder.

But not all set yet. Since this is a POST request with a request body, the request must specifiy a **Content-Type** header. We need to make a modification to **DctmRestClient**.

(3) Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/restclient/DctmRestTemplate.java** and modify the method **defaultHeaders** to add request Content-Type.

Your code here – modify method **defaultHeaders** in **DctmRestTemplate.java**

```
protected HttpHeaders defaultHttpHeaders(Object requestBody) {
    HttpHeaders headers = new HttpHeaders();
    headers.setAccept(Arrays.asList(DCTM_VND_JSON_TYPE, MediaType.APPLICATION_JSON));

    //TODO FOR ROUND 3 -- BEGIN
    //TODO FOR ROUND 3 -- SET REQUEST CONTENT TYPE FOR JSON
    //TODO FOR ROUND 3 -- END

    return headers;
}
```

Answer

```
//CODE FOR ROUND 3 -- BEGIN
//CODE FOR ROUND 3 -- SET REQUEST CONTENT TYPE FOR JSON
headers.setContentType(DCTM_VND_JSON_TYPE);
//CODE FOR ROUND 3 -- END
```

a

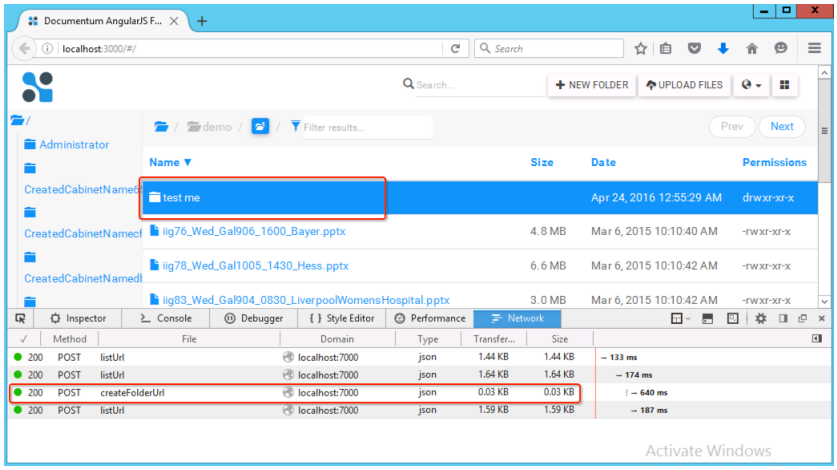
a. The JSON request Content-Type for Documentum REST Services is to **application/vnd.emc.documentum+json**.

Step 3: test

With all above work, we are done with the task Create Folder. Let's restart the File Manager API Server and see the result.

(1) Press **ctrl + c** on the Command window where project **file-manager-api** is running
(2) Press **y** to terminate the batch
(3) Run command: **mvn spring-boot:run**

Open Firefox web browser, refresh the page **http://localhost:3000**, now you can create a new folder.



Note that you must first click on a folder on the sidebar, then click the button to create new folders. It's because folders in a repository cannot be orphans.

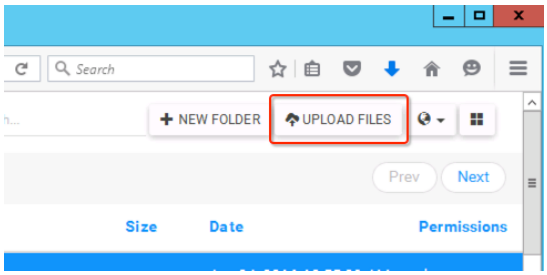
Summary

- File Manager UI sends XHR requests to File Manager API Server to get cabinets and folders with the pattern: POST /createFolderUrl
- File Manager API Server needs to find the folder child folders resource in Documentum REST Services to make the POST request to create a folder
- In Documentum REST Sevices, a link relation is defined to discover this resource.
    - folder --> child folders: http://identifiers.emc.com/linkrel/folders

Round 4: Upload content

Overview

Next, let's implement the function to upload files to Documentum repository.



Get the code

1. Open a Command window
2. Browse to **C:\emcworld\emcworld2016-hackathon-documentum-rest**
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

file-manager-api

In File Manager API Server, we will implement the REST client method for file upload. But let's first check the controller endpoint.

(1) Open source code Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/filemanager/controller/FileManagerController.java**.

```
@RequestMapping(value = "/uploadUrl",
        method = RequestMethod.POST,
        consumes = MediaType.MULTIPART_FORM_DATA_VALUE,
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
public CommonResult uploadContent(MultipartHttpServletRequest request) throws
DocumentumException {…}
```

a

Code Explanation

a. The file upload endpoint accepts HTTP multipart request. The first part contains the destination folder, the remaining parts contain the document names and contents.

Your are not required to make changes to the controller as it is already completed.

(2) Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/restclient/DctmRestClient.java** and implement the method **createContentfulDocument**.

Your code here – complete method **createContentfulDocument** in **DctmRestClient.java**

```
public JsonObject createContentfulDocument(JsonObject folder, byte[] data, String filename,
String mime) {
    MultiValueMap<String, Object> parts = new LinkedMultiValueMap<>();

    //TODO FOR ROUND 4 -- BEGIN
    //TODO FOR ROUND 4 -- RESOLVE 'part1'
    HttpEntity part1 = null;
    //TODO FOR ROUND4 -- END
    parts.add("metadata", part1);

    //TODO FOR ROUND 4 -- BEGIN
    //TODO FOR ROUND 4 -- RESOLVE 'part2'
    HttpEntity part2 = null;
    //TODO FOR ROUND4 -- END
    parts.add("binary", part2);

    ResponseEntity<JsonObject> result =
streamingTemplate.post(folder.getHref(LinkRelation.DOCUMENTS),
            parts,
            JsonObject.class);
    return result.getBody();
}
```

b

c

a

Code Explanation

a. When creating a contentful document (properties + binary), Documentum REST Services accepts the HTTP multipart request, too. On the target folder's link relation "**http://http://identifiers.emc.com/linkrel/documents**", the client makes a POST multipart request.

b. The first part should be the JSON object of the document, where in this case, it only specifies the document name.

c. The second part should be the binary (byte array) of the document content, where its mime type is known.

Answer

```
//CODE FOR ROUND 4 -- BEGIN
//CODE FOR ROUND 4 -- RESOLVE 'part1'
PlainRestObject doc = new PlainRestObject("dm_document",
        singleProperty(DocumentumProperties.OBJECT_NAME, filename));
MultiValueMap<String, String> part1Headers = new LinkedMultiValueMap<>();
part1Headers.set("Content-Type", DctmRestTemplate.DCTM_VND_JSON_TYPE.toString());
HttpEntity part1 = new HttpEntity<>(doc, part1Headers);
//CODE FOR ROUND4 -- END
parts.add("metadata", part1);

//CODE FOR ROUND 4 -- BEGIN
//CODE FOR ROUND 4 -- RESOLVE 'part2'
MultiValueMap<String, String> part2Headers = new LinkedMultiValueMap<>();
part2Headers.set("Content-Type", mime);
HttpEntity part2 = new HttpEntity<>(data, part2Headers);
//CODE FOR ROUND4 -- END
parts.add("binary", part2);
```

a

b

o

de Explanation

a. Add the Java POJO to the first part. Spring RestTemplate will auto detect the data type and convert it to the JSON message.

b. Add the byte array to the second part. Spring RestTemplate will auto detect the data type and send it as the raw data.

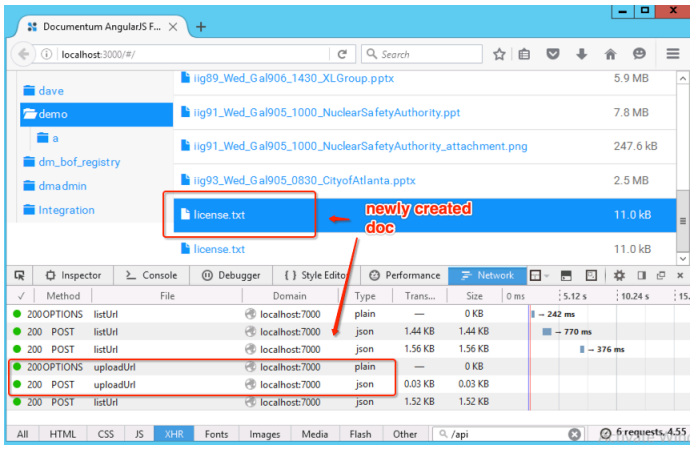
Please also note **defaultHttpHeaders** changes to **DctmRestTemplate**.

Step 3: test

With all above work, we are done with the task Upload Content. Let's restart the File Manager API Server and see the result.

(1) Press **ctrl + c** on the Command window where project **file-manager-api** is running
(2) Press **y** to terminate the batch
(3) Run command: **mvn spring-boot:run**

Open Firefox web browser, refresh the page **http://localhost:3000,** now you can upload a file to a selected folder.



Note that you must first click on a folder on the sidebar, then click the button to create new folders. It's because folders in a repository cannot be orphans.

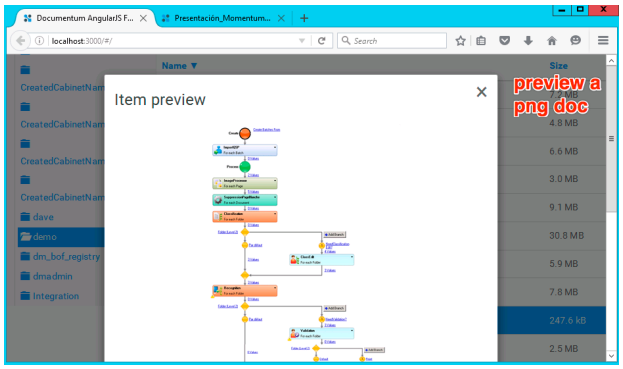
Summary

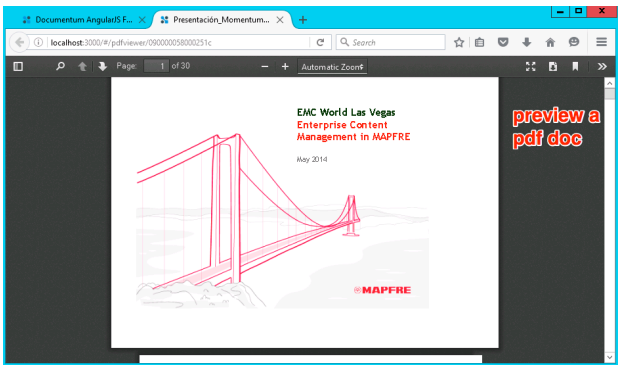
- File Manager UI sends XHR requests to File Manager API Server to upload files with the pattern: POST /uploadUrl
- File Manager API Server needs to find the folder child documents resource to make the POST request to create a contentful document
- In Documentum REST Sevices, a link relation is defined to discover this resource.
  - folder --> child documents: http://identifiers.emc.com/linkrel/documents
- When creating a contentful document, Documentum REST Services accepts HTTP multipart request, where the 1st part is the JSON (or XML of course) object, and the 2nd part is the content binary.

Round 5: Edit content

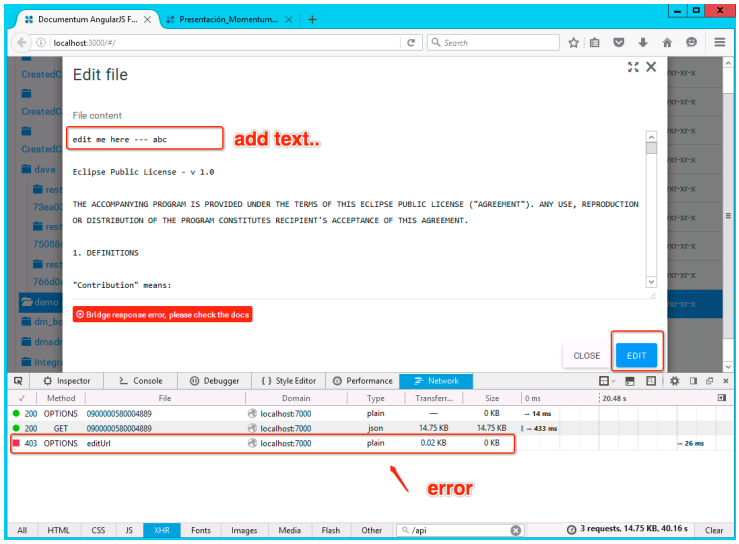
Overview

In this section, we will take a look how to edit a textural content of the document. In this round, we provide the complete code for content download. So now, you can play with content download & view.





But edit does not work. We will fix it in this round.



Get the code

1. Open a Command window
2. Browse to **C:\emcworld\emcworld2016-hackathon-documentum-rest**
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: file-manager-ui

Although we do not need to modify the UI part, it is still helpful if we take a look at the code to understand what file formats are set as editable.

(1) Open source code **<root>/file-manager-ui/src/app/filemanager/providers/config.js**. You can find below code snippet which defines which file formats are vieweable or editable.

```
isEditableFilePattern:
/\.(txt|html?|aspx?|ini|pl|py|md|css|js|log|htaccess|htpasswd|json|sql|xml|xslt?|sh|rb
|as|bat|cmd|coffee|php[3-6]?|java|c|cbl|go|h|scala|vb)$/i,
isImageFilePattern: /\.(jpe?g|gif|bmp|png|svg|tiff?)$/i,
isExtractableFilePattern: /\.(gz|tar|rar|g?zip)$/i,
isPdfFilePattern: /\.(pdf)$/i,
```

AngularJS models and views read this config to determine whether specific events are applicable for a selected item.

Step 2: file-manager-api

In File Manager API Server, the controller already defins a method for edit content, but it's not set as a HTTP endpoint yet. You tasks start from here.

(1) Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/filemanager/controller/FileManagerController.java**.

You action here – complete the method **editContent** in **FileManagerController.java**

```
//TODO FOR ROUND 5 -- BEGIN
//TODO FOR ROUND 5 -- RESOLVE @RequestMapping
//TODO FOR ROUND 5 -- END
public CommonResult editContent(@RequestBody CreateObjectRequest request) throws
DocumentumException {
    fileManagerApi.updateContent(request.getId(), request.getContent());
    return successResponse();
}
```

a

Code Explanation

a. The endpoint accepts a POST method on URL '/editUrl', and returns the JSON message.

Answer

```
//CODE FOR ROUND 5 -- BEGIN
//CODE FOR ROUND 5 -- RESOLVE @RequestMapping
@RequestMapping(value = "/editUrl", method = RequestMethod.POST, produces =
MediaType.APPLICATION_JSON_UTF8_VALUE)
//CODE FOR ROUND 5 -- END
```

(2) The next is to implement the REST client method **updateContent**. Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/restclient/DctmRestClient.java**.

Your action here – comlete method **updateContent** in **DctmRestClient.java**

```
public JsonObject updateContent(JsonObject doc, byte[] data) {
    //TODO FOR ROUND 5 -- BEGIN
    //TODO FOR ROUND 5 -- IMPLEMENT
    return new JsonObject();
    //TODO FOR ROUND 5 -- END
}
```

a

Code Explanation

a. There are two ways to update a content for a document. (1) checkout and check in the new content; (2) overwrite the primary content directly. For simplicity, we use approach (2).

Answer

```
//CODE FOR ROUND 5 -- BEGIN
//CODE FOR ROUND 5 -- IMPLEMENT
String format = (String) doc.getPropertyByName(DocumentumProperties.CONTENT_TYPE);
ResponseEntity<JsonObject> result =
streamingTemplate.post(doc.getHref(LinkRelation.CONTENTS),
        data,
        JsonObject.class,
        QueryParams.OVERWRITE, "true",
        QueryParams.FORMAT, format);
return result.getBody();
//CODE FOR ROUND 5 -- END
```

b

a

c

d

Code Explanation

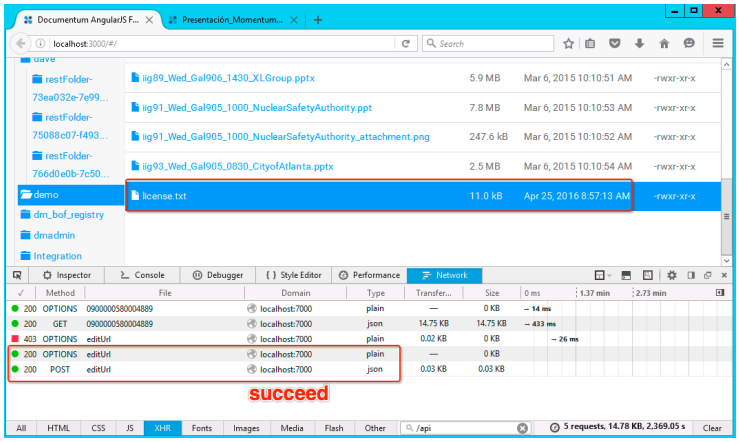
a. Use streaming RestTemplate as it does not buffer content in memory.
b. Make a POST method to contents feed of the document by link relation "**contents**".
c. Specify overwrite.
d. Specify the source content format. For an overwrite, it must match the original content format.

Step 3: test

With all above work, we are done with the task Edit Content. Let's restart the File Manager API Server and see the result.

(1) Press **ctrl + c** on the Command window where project **file-manager-api** is running
(2) Press **y** to terminate the batch
(3) Run command: **mvn spring-boot:run**

Open Firefox web browser, refresh the page **http://localhost:3000**, you can now edit the content.



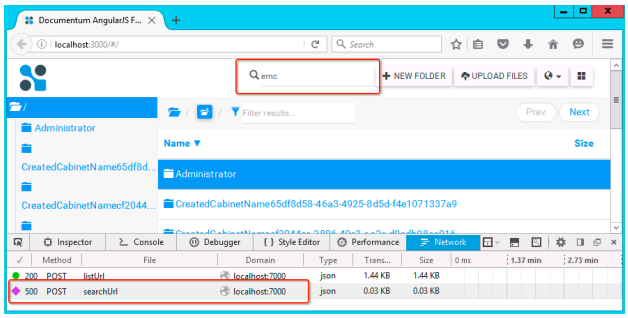
Summary

- File Manager UI sends XHR requests to File Manager API Server to update content with the pattern: POST /editUrl
- In Documentum REST Sevices, a link relation is defined to discover the document contents resource.
    - document --> contents: contents
- When overwriting a primary content, the REST client should specify both **overwrite** and **format** query parameters.

Round 6: Full-text search

Overview

The last round for code practice is full-text search. You can see from the UI page's context bar, there is an input field for **Search**. But it does not work.



Get the code

1. Open a Command window
2. Browse to **C:\emcworld\emcworld2016-hackathon-documentum-rest**
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

file-manager-api

The task for you is to complete the search URL and parameter for the Documentum REST request.

(1) Open source code **<root>/file-manager-api/src/main/java/com/emc/documentum/restclient/DctmRestClient.java**.

Your action here – complete the method **simpleSearch** in **DctmRestClient.java**

```
public List<JsonEntry> simpleSearch(String terms, String path, int page, int
itemsPerPage) {
    //TODO FOR ROUND 6 -- BEGIN
    //TODO FOR ROUND 6 -- IMPLEMENT
    String searchUrl = "";                                  a
    String[] queryParams = new String[0];                   b
    //TODO FOR ROUND 6 -- END
    ResponseEntity<JsonFeed> result = restTemplate.get(searchUrl,
            JsonFeed.class,
            queryParams);
    return result.getBody().getEntries();
}
```

Code Explanation

a. The search link relation can be found from repository
b. The query should contain the **q** parameter for simple search expression.

Answer

```
//CODE FOR ROUND 6 -- BEGIN
//CODE FOR ROUND 6 -- IMPLEMENT
String searchUrl = repository.getHref(LinkRelation.SEARCH);
String[] queryParams = new String[0];
if (("/".equals(path) || Strings.isNullOrEmpty(path))) {
    queryParams = new String[] {
            QueryParams.Q, urlEncodeQueryParam(terms),
            QueryParams.VIEW, DEFAULT_VIEW,
            QueryParams.PAGE, String.valueOf(page),
            QueryParams.ITEMS_PER_PAGE, String.valueOf(itemsPerPage)
    };
} else {
    queryParams = new String[] {
            QueryParams.Q, urlEncodeQueryParam(terms),
            QueryParams.VIEW, DEFAULT_VIEW,
            QueryParams.PAGE, String.valueOf(page),
            QueryParams.ITEMS_PER_PAGE, String.valueOf(itemsPerPage),
            QueryParams.LOCATIONS, urlEncodePathParam(path)
    };
}
//CODE FOR ROUND 6 -- END
```

a, b, c, d

Code Explanation

a. The simple search expression should be URL encoded
b. You can specify which attributes to return for the search result item
c. You can specify the pagination
d. You can also specify the search location

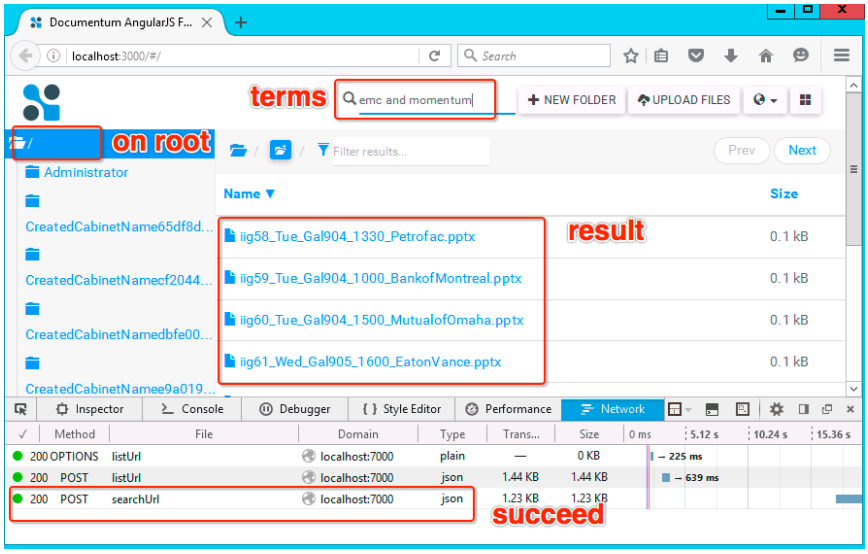
Step 3: test

With all above work, we are done with the task Full-text Search. Let's restart the File Manager API Server and see the result.

(1) Press **ctrl + c** on the Command window where project **file-manager-api** is running
(2) Press **y** to terminate the batch
(3) Run command: **mvn spring-boot:run**

Open Firefox web browser, refresh the page **http://localhost:3000**, you can now search documents by terms.

- You can use simple search language like 'emc' and ('documentum' or 'test')
- You can select a folder first, then perform the search



Summary

- In Documentum REST Sevices, a link relation is defined to discover the full-text search resource.
  - repository --> search: http://identifiers.emc.com/linkrel/search
- We can execute simple search language on Documentum REST Services 7.2
- Full-text search can be based on locations

Round 7: Review the complete project

Overview

That's all for your tasks. Now let's get the complete project and view more functions like copy, move, rename, etc.

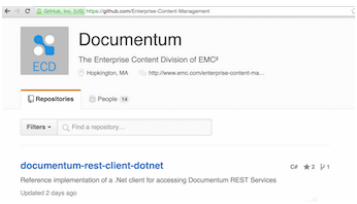
Get the code

(1) Open a Command window
5. Browse to **C:\emcworld\emcworld2016-hackathon-documentum-rest**
6. Run: **git checkout –b last origin/last –f** (note: this will override any changes you have already made)
7. Refresh your project in Spring Tool Suite

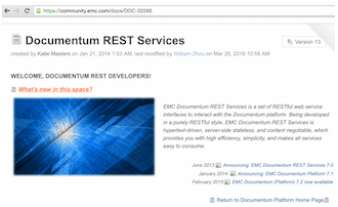
Summary

This sample project and its tutorials is available on GitHub: https://github.com/Enterprise-Content-Management/emcworld2016-hackathon-documentum-rest. To learn more about Documentum REST Services, please reach out to **Documentum@GitHub** and **Documentum@EMC Develop Network**.

https://github.com/Enterprise-Content-Management



https://community.emc.com/docs/DOC-32266



Please also stay tuned for **Documentum REST Bedrock release**, which brings you more exiciting features.

Cheatsheet

```
##############
##    get code   ##
##############

# get code branch for round 1
# open a new command window (1)
> cd C:\emcworld
> git clone https://github.com/Enterprise-Content-Management/emcworld2016-hackathon-documentum-rest.git -b round1
> cd emcworld2016-hackathon-documentum-rest

# get code branch for <round x> for the 1st time
# reuse command window (1)
> git checkout -b <round x> origin/<round x> -f

# switch to existng local code branch <round y>
# reuse command window (1)
> git checkout <round y>

########################
##    build file-manager-api    ##
########################

# build & run file-manager-api
# open a new command window (2)
> cd C:\emcworld\emcworld2016-hackathon-documentum-rest\file-manager-api
> mvn spring-boot:run

# re-build & re-run file-manager-api
# reuse command window (2)
> # press keys ctrl + 'c' to terminate
> # press key 'y' to confirm
> mvn spring-boot:run

########################
##    build file-manager-ui    ##
########################

# build & run file-manager-ui
# open a new command window (3)
> cd C:\emcworld\emcworld2016-hackathon-documentum-rest\file-manager-ui
> npm install --save-dev
> bower install
> gulp serve

##############################################
##    part of link relations for documentum rest services    ##
##############################################

home doc   --> repositories:        http://identifiers.emc.com/linkrel/repositories
```

```
repository  --> cabinets:              http://identifiers.emc.com/linkrel/cabinets
repository  --> search:                http://identifiers.emc.com/linkrel/search
repository  --> dql:                   http://identifiers.emc.com/linkrel/dql

folder      --> child folders:         http://identifiers.emc.com/linkrel/folders
folder      --> child objects:         http://identifiers.emc.com/linkrel/objects
folder      --> child docs:            http://identifiers.emc.com/linkrel/documents

document    --> contents:              contents
document    --> primary content:       http://identifiers.emc.com/linkrel/primary-content
document    --> parent links:          http://identifiers.emc.com/linkrel/parent-links

object      --> object:                self
object      --> object (edit):         edit
object      --> object (delete):       http://identifiers.emc.com/linkrel/delete

content     --> content media:         http://identifiers.emc.com/linkrel/content-media

##########################################
##    json media type for documentum rest services     ##
##########################################

application/vnd.emc.documentum+json

########################################
##    http methods for documentum rest services    ##
########################################

GET         # fetch
POST        # create or partial update
PUT         # complete update
DELETE    # remove

##################################################
##    typical json representation for documentum rest services    ##
##################################################

{
   "properties": {
      "object_name": "readme.txt",
      "title": "Read me file for beginners"
   },
   "links": [
      {
         "href": "self",
         "rel": "http://localhost:8083/dctm-rest/repositories/repo1/documents/090000123456789a"
      },
      {
         "href": "contents",
         "rel": "http://localhost:8083/dctm-rest/repositories/repo1/objects/090000123456789a/contents"
      }
   ]
}
```