ID2208 - Project
Programming Web Services

Weherage, Pradeep Peiris

February 27, 2016

eDocument Restful Service

1. Introduction

eDocument is a service module, which provides (Online/Cloud) services for managing electronic documents. The following are the main features of ‘eDocument’ ‘focused in this project work.

- Publish Document Templates
  eDocument provides a service API for publishing various document templates. The document template can be anything that is renderable with given data set and generates the final document output. Through this service API, users can publish their document templates publicly, personally or within selected user groups.

- List Document Templates
  This will list down all publically available document templates. Also templates that relate given user profile.

- Generate Documents
  There will be two variant of the service. The first one is the simplest form of the service, which generates document output for the given data set and document template. The next variant of the service may require only the data set as the input. It will select the document template that best matches (Text Matching) with the data set.

eDocument can be further improved with various other services, the following are few of them.

- Securely manage document templates
  Enable more secure options in sharing document templates

- Tagging Template
  Enable document templates to be tagged with predefined properties such as Domain (e.g Supply Chain, Financial, Procurement etc.), Type (e.g Order Request, Invoice etc) and with any free text.

- Improve template selection with Content based Filtering
  Use content based filtering in selecting document templates for given data set.
  E.g. The system may filter out templates tags with ‘Invoice’ for data set coming from Procurement domain.

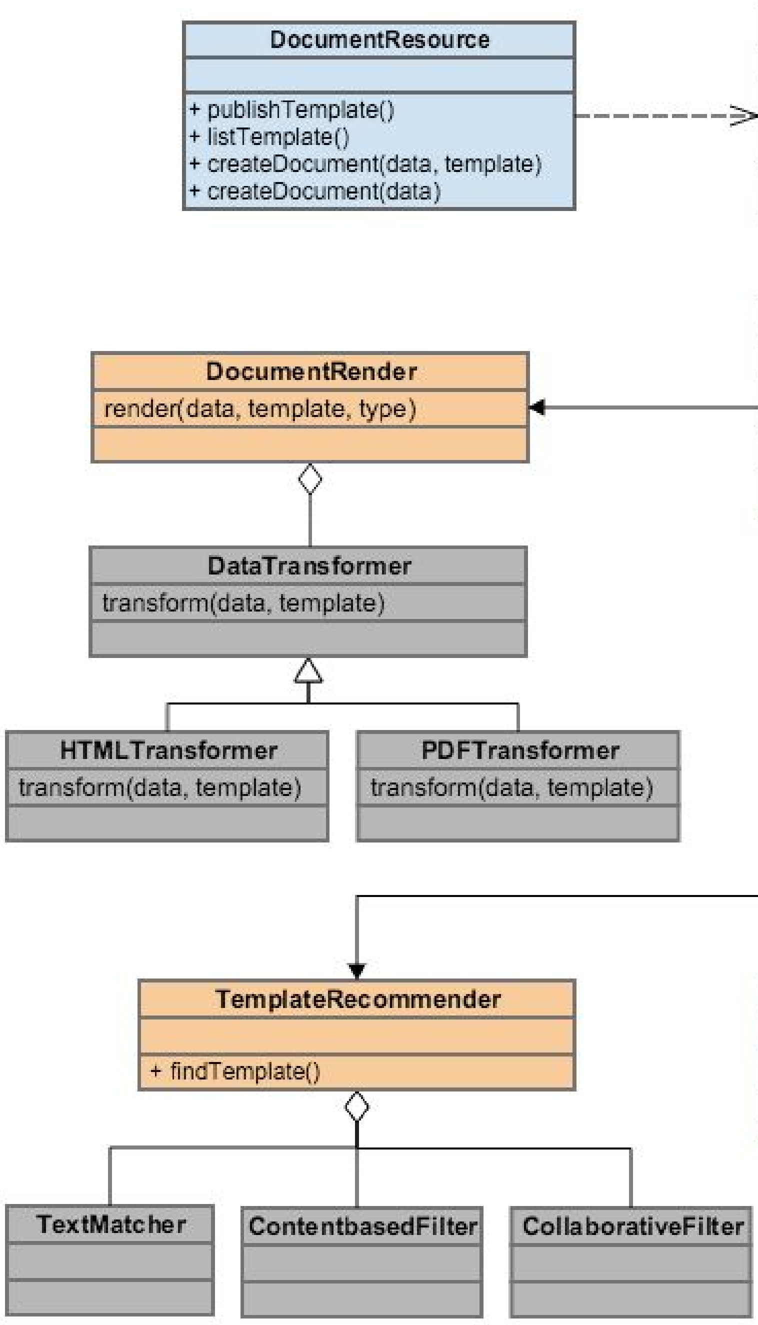
- Improve template selection with Collaborative Filtering
  Document generation API can be enabled to generate multiple document output that best matches with text matching and content based filtering, and let user to select his/her prefered document output. The system will use these user collaborative information in improving selection of document templates. That is, user’s preference information can be captured into a machine learning algorithm like ALS (Alternating Least Squares) and

rank the document templates.
(http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html#collaborative-filtering)

For the project it is considered only the above three main services, which are exposed as Restful web services. The document templates considered here is only XSL (Extensible Stylesheet Language). The document output the system generates are HTML or PDF, which depends on the given XSL template. It will apply a simple Text Matching in the XSL file to determine the expected document output.

2. System Overview

The following diagram shows the overall design and its components of eDocument Service.



As it shows in above diagram, Document Service associate with Document Render, Template Recommender and Document Data Access modules.

- Document Data Access Object
  Document DAO provides data access services, mainly storing and loading Templates.

- Document Render
  Document Render transforms the data into a document output with according the given template. There can be variant type of Renders. The design above allows to switch in between different renders at runtime.

- Template Recommender
  Template Recommender recommends the template that best matches with the given data. The recommendation module can be further extended with different recommendation algorithms such as Content based filter, Collaborative filtering.

3. System Implementation

3.1 Restful Services

eDocument System is a set of Restful services that can be accessed over HTTP.
The following are the main service interfaces.

3.1.1 Publish Document Template

The service expect the document template in XML format. (We considered only XSL templates in current implementation)

```
@POST
@Path("/template")
@Consumes(MediaType.APPLICATION_XML)
public void publishTemplate(String id, String templateContent) {

     doxService.createTemplate(id, templateContent);
}
```

3.1.2 List Document Template

The service simply list down all the available document template as JSON array list.

```
@GET
@Path("/template")
@Produces(MediaType.APPLICATION_JSON)
public List<Template> listTemplate() {
     return doxService.listTemplates();
}
```

3.1.3 Generate Document with Selected Template

This service generates the document according to the selected template over given data set.

```
@POST
@Path("/document/{templateid}")
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_JSON)
public DocumentOutput generateDocument(String templateid,String
data){
     return doxService.generateDocument(templateid, data);
}
```

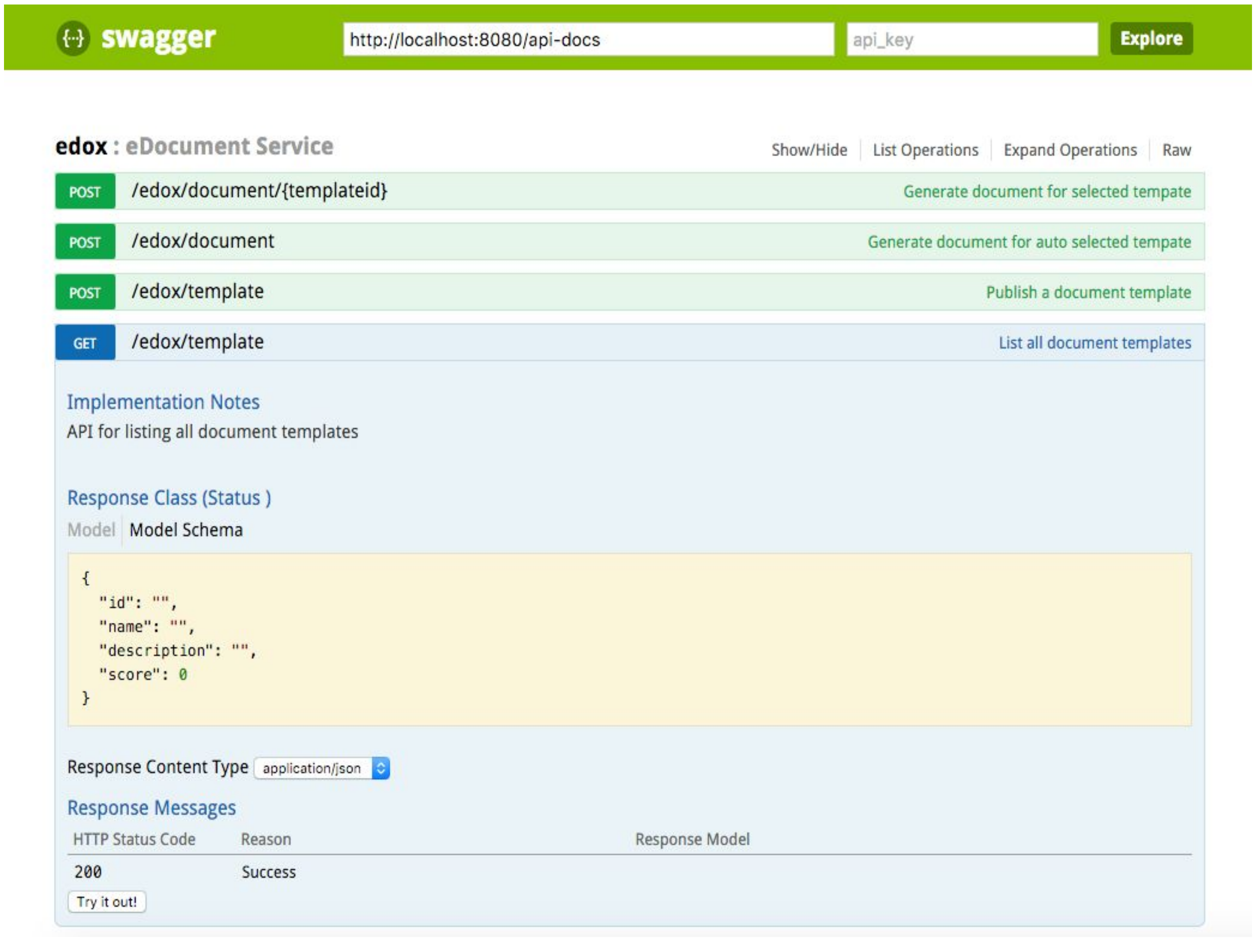
3.1.4 Generate Document with System Recommended Template

This service generates the document with system recommended template. The recommendation in current implementation is based on Levenshtein text matching. That is, the template that best matches with given data set is chosen for Document rendering.

```
@POST
@Path("/document")
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_JSON)
public DocumentOutput generateDocument(String data) {
      return doxService.generateDocument(data);
}
```

3.2 Service Representation with Swagger (http://swagger.io)

Swagger is an Open Source specification for describing and visualizing RESTful Web Services. With Swagger enabled RESTful APIs, we get interactive documentation, service discoverability even with client SDK generation tools.
eDocuement Service is build with Swagger framework, so that all the available services are expressed appropriately.


swagger
http://localhost:8080/api-docs
api_key
Explore

edox : eDocument Service

Show/Hide | List Operations | Expand Operations | Raw

POST	/edox/document/{templateid}	Generate document for selected tempate
POST	/edox/document	Generate document for auto selected tempate
POST	/edox/template	Publish a document template
GET	/edox/template	List all document templates

Implementation Notes

API for listing all document templates

Response Class (Status)

Model | Model Schema

```
{
  "id": "",
  "name": "",
  "description": "",
  "score": 0
}
```

Response Content Type application/json

Response Messages

HTTP Status Code	Reason	Response Model
200	Success	

3.3 Document Formating with Apache FOP (https://xmlgraphics.apache.org/fop/)

Apache FOP, Formatting Object Processor is framework under Apache Software Foundation, which read formatting objects (XSL with special FOP namespace values) file and render the resulting pages to specified out. The output format currently supported in the framework are PDF, PCL RTF and TXT.
In eDocument Service implementation, XSL-FO templates are identified by the special namespace value and rendered into only PDF format. Otherwise, the standard document output format is HTML, which is rendered from XSL.

3.3 Template Recommendation with Levenshtein

Levenshtein Distance of text matching is used in auto selecting the Template that best matches for the given data set. Template Recommender module in the system, first extract the all possible XPaths of input data file. XPaths are then matches with the available XSL templates using Levenshtein distance.

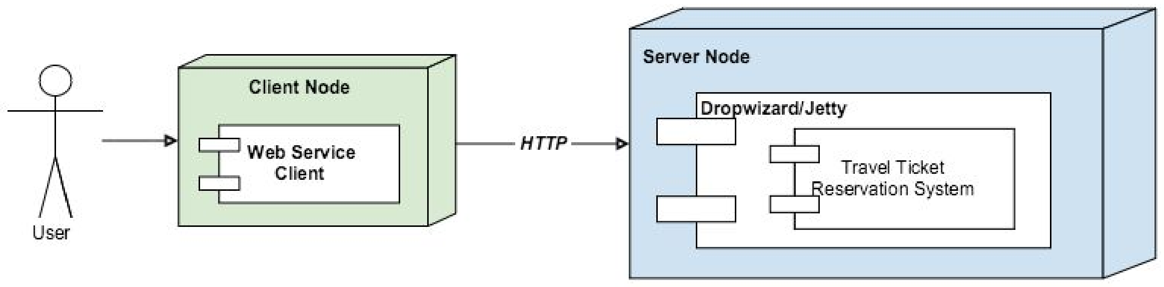
As an example, the XPath of given data xml file could be, */Order/Packages/Package/Items.*
In a XSL file this XPath might may have used with additional expression such as count(*/Order/Packages/Package/Items).* Levenshtein distance gives a good measure on such text matches.

Also, the system can be further improved other recommendation algorithm, which will be briefly explained in later section.

4. System Deployment

4. 1 Dropwizard framework with embedded Jetty http container.

Dropwizard is a framework for web application that bundle everything into deployment ready one artifact and hence improve the maintenance and productivity of application development. Dropwizard RESTful service is the based framework for eDocument Service, which simplifies the service implementation and deployment.
eDocument Service application is bundled into one single artifact (jar) with use of Dropwizard framework (Jetty Web Server is embedded within the application) and let it be run as a single micro service.



5. Further Improvements

5.1 Template Recommendation with Content based Filtering and Collaborative Filtering

Suppose our eDocument Service was used by users in large scale, more relevant information can be gathered for improving the Template Recommendation.

5.1.1 Content based Filtering

Traditionally Content-based filtering is used in recommending items with comparison between the content of a item and a user profile. The content of each item is represented as a set of terms, and user's profile represented with the same terms, which was built up by analysing the content of the items that have been referred by the user. This perfectly maps with the recommendation requirement of eDocument Service.

The term vector for users can be generated from the user’s profile information and document templates that user has be accessed. The term vector of the input data, can be parsing the document itself.

5.1.2 Collaborative Filtering

Collaborative filtering is based on analyzing a large amount information to identify user’s behaviours, activities and preferences. It is based the assumption that users who agreed in past on something will agree in the future, and that they will like similar kinds of items. This is quite appropriate with our Document Services. As users who used a document template before might have high chance of using it again for his/her future work, or any similar template. Also Collaborative filtering leads to identify similar users and templates.
There are many algorithm have been used in measuring the user similarity or item similarity. For example K-Nearest Neighbor Pearson Correlation.
Also, Machine Learning algorithm like Alternative Least Squares (ALS) build with Apache Spark Machine Learning Library (http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html) is a better options to handle with large data set.

Reference

http://swagger.io/
http://www.dropwizard.io/
https://xmlgraphics.apache.org/fop/
http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

Technologies

- Dropwizard web service framework (http://www.dropwizard.io)
- Swagger (http://swagger.io)
- Apache FOP (https://xmlgraphics.apache.org/fop/)
- Maven (https://maven.apache.org/)

Prerequisite

- Maven 3.x as the build tool
- Java 7

Build the application

- mvn package

Run the application

- java -jar target/id2208-1.0.0.jar server app.yml

Access REST API Reference with Swagger

- http://localhost:8080/swagger