Automating Database Uploads with Representational State Transfer Services

Christopher Keene and Matt Desimini
Seidenberg School of CSIS, Pace University, Pleasantville, *New York*

***Abstract*—The Pace University Library Catalog houses the dissertations of all Doctor of Professional Studies in computing. This work extends preceding work to organize the Doctor of Professional Studies dissertations. This system uses a database and a web interface to display the view to the end user. This system ensures the ease of use for the end user to organize and manipulate the documents in the database. The goal of this project is to add to the functionality of the dissertation database with a focus on automation. Autonomous document uploads and categorizations to the database through the use of custom built REST services and Google APIs will help streamline the process of uploaded dissertation to the database.**

***Index Terms*—RESTful services, REST, Database, MySQL, Spring, POI**

I. INTRODUCTION

THE DPS dissertation database grants users the ability to view information about the doctoral student dissertations as well as provide a provide a resource to view statistical information on the dissertation within the database. Last semester we focused on updating the user interface while decoupling the frontend load from the backend. We did that through the use of AngularJS. One thing that we did not focus on last semester was the admin section of the web application. That is why this semester leans on a focus on functionality; more specifically, automation. We will streamline the process of uploading documents to the database by doing it programmatically. To do so, we will extend the current applications functionality though the use of a representational state transfer application program interface or RESTful API (REST API). This allows the developer a greater ability to scale and maintain the project because minimal code will be necessary for the web application itself. The application will call the REST service and the requested data will be returned to the application.

Our current goals for the system are as follows:

- Build a REST service that will automate the process of uploading data to the database.
- Build a REST service with the responsibly of accessing Google's natural language processing (NLP) API.
- Automate categorizing dissertations within the database by the contents of the abstract.
- Fix any outstanding issues with the application
- Build a screen in the admin section that allows the user to interact with the "database upload" REST service.

The main goal of this project is to create a web service with two responsibilities.

1. Receive an abstract and return a category. Currently, Google's NLP API seems like the best fit to accomplish this goal. More research will determine if that is doable with our current resources or if we should be moving toward a homegrown solution.
2. Receive a Word document (and PDFs in the future) and populate the database based on the contents of that document.

Once we accomplish this, we can then configure DPS dissertation database application to allow the administrative user call upon the web service.

II. SYSTEM SPECIFICATIONS

A. Software Stack

To accomplish the goals of this project we had to choose a proper stack, or a group of programs and libraries used for this project. Our web service is built on Java and we used the Spring Framework to accomplish this, most notably, Spring REST. Spring REST. Spring REST allows to build a REST service through Java code and Java annotations. We also made use of Spring Boot. This allows the developer to test web applications on their local machine without the use an application server. Spring Boot web projects have a Java main class that embeds a tomcat server in a jar file instead of a war file. This allows the developer to run and rerun these projects with ease. It also makes it easier to deploy later by removing a lot of the necessary configurations that not Spring Boot applications are slave to. Spring is open source and is the central component of our stack.

Another key component to our project is the user needs to be able to upload a Microsoft Word document to the server. Apache Commons and Apache POI are an important open source library with regards to goal. In the case of this project, they both are responsible for enabling the upload of and extracting the text from the word document.

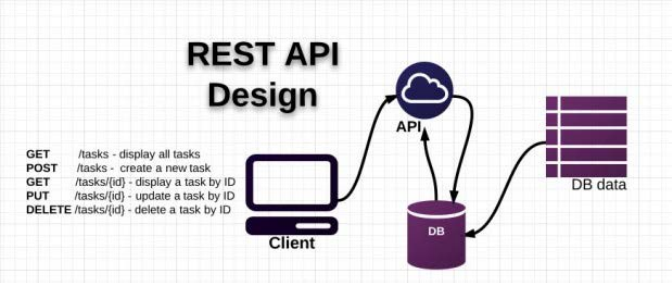

Figure 1. This is an example of the REST design architecture [12].

Since the user interface will be added to the original application, it will be composed of the same stack as well. That includes JQuery, AJAX, and BootStrap CSS. These are all open source and they are discussed further in last semester's paper.

B. Backend and MySQL Database

As mentioned earlier, the backend of this section of the project is going to be completely different from the backend of the dissertation database. Where that one has a PHP backend, this one will leverage REST built on Java. It will bypass the PHP code altogether as you can see in the image above in figure 1.

C. The Frontend

This application will utilize one screen for the user to interact with. The screen will be in the admin section of the dissertation database and will be responsible for document uploads and calling the rest service via an AJAX http call.

```
$.ajax({
        type: "POST",
        enctype: 'multipart/form-data',
        url: "/api/upload/",
        data: data,
        processData: false,
        contentType: false,
        cache: false,
        timeout: 600000,
        success: function (data) {

        $("table").show();
        $("#result").text(data.message);
        $("#result").addClass("alert alert-success");
        $("#btnSubmit").prop("disabled", false);

        // …

});
```

Notice in the code you see the URL value "/api/upload/". That is the URI that identifies our REST service. The document will get transferred to that endpoint and return a success or error. Everything else will be managed on the backend.

D. Going Forward

We are going to build a REST service and a user interface to automate the process of uploading documents and categorizing them accordingly. We will be fortunate to complete it all due to time constraints. That does not take away from what needs to be done going forward. Once this is complete the focus should move toward a more secure application. Last semester, we salted the passwords so that the passwords were no longer stored in plain text in case the database were compromised. There will need to be some form of token exchange to ensure that only the DPS dissertation database system has access to the REST service. We do not want to leave it vulnerable to cross-site scripting and SQL injection by hackers.

III. Enhancements

This section will discuss the enhancements that were made to the DPS dissertation system. It is based on a separate architecture from the original system but will be simple to integrate. It will be just another page in the admin section of the site but, as stated earlier, it will utilize a different backend via REST services.

A. Leveraging a REST API

A REST API is a way of exchanging text-based data between computer systems on the internet. These resources are available on the internet through a URL that is used to identify them, usually through the form of an HTTP. Because of this, they encompass a series of CRUD (create, read, update, and delete) operations as well. REST APIs can return data to the client in the form of JSON, HTML, or XML. We will be using JSON for the purposes of our application. The frontend of the application leverages an AngularJS framework and JSON is the type of data that it prefers.

REST is more of an architectural style or idea. Thus, there are guidelines to REST that should be followed. In practice, all of these guidelines are seldom followed to a tee but it is not REST unless it adheres to some of the key guidelines or constraints.

1. Client-Server – the separation of the client and server is a key aspect with RESTful services. Separating the frontend logic from the backend provides portability. This logic will also enable the application to be more scalable. Any changes on the frontend will have no effect to backend changes and vice versa. The frontend will call the data that it needs and move on. The backend is in no way responsible for manipulating that data in any way.
2. Stateless – another key factor regarding RESTful services is that it is stateless. When the client calls on the server it must request all of the data that is necessary to understand the request. Sessions are stored on the client because true RESTful services are not responsible for managing sessions. This also makes the application more reliable. If sections of an application go down for any reason, should continue to function (minus the portions that are down). Scalability is enhanced because the server no longer has to worry about session states.

That will free server resources by default since the server is no longer working after it satisfies the client's request.

3. Cache – cache is a means of improving network efficiency. It can be argued that all one needs to implement a RESTful API are the first two constraints but cache is necessary to be a true REST service. Another note, it can be easy to forget about these other guidelines because there are a lot of libraries available that streamlines the process of building a RESTful API. They usually encompass these other guidelines by default and can be adjusted by the developer. Our RESTful API leverages Spring Boot to develop this web service for the DPS Dissertation Database. RESTful services mandate that all request be explicitly deemed cacheable or non-cacheable. By default, Spring requests are non-cacheable. That is a layer of abstraction that these libraries provide which explains why it is stated that the first two constraints are the most important.
4. Uniform Interface – "The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components. By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved" [11]. A uniform interface simplifies the architecture, allowing each part to evolve independently. This also comes with its own set of constraints:
    a. Identification of resources – an example of this would be that each REST resource is identified by its uniform resource identifier (URI).
    b. Hypermedia as the engine of application state (HATEOAS) – A REST client should be able to see all of the available resources and actions it needs.
    c. Self-descriptive messages – "Each message includes enough information to describe how to process the message" [11].
5. Code on Demand – the functionality of a client can be extended through a REST service through the transfer of executable code, such as a java applet or client-side scripts.

This list of constraints is what makes a REST service what it is. It is an important part of the application because these services will grant the application more functionality while making little changes to the DPS dissertation database application itself.

B. Automatic Document Uploads

DPS dissertations are currently uploaded to the database via the admin screen by filling out a form. The database has several columns causing it to be tedious when having to upload multiple dissertations. We decided to automate the process of filling out the form in the database by creating a REST service, built on Java, that the DPS dissertation database application can call. The REST service will return values from the dissertation that will automatically populate the form. Another function would be to access this REST service in the form of a UI, allowing the user to directly upload the document, that will automatically populate the form or directly update the database as long as all the required information is available, and if the user chooses to do so.

An example of this would be in the UI. the user would choose the documents to upload, press the submit button, and the document will either populate the form or upload the information directly to the database. It grants the administrator the ability to upload documents to the database. The administrator may choose a .docx file to upload and then once submitted, will get the opportunity to verify that the contents of the dissertation are in the correct category before submitting it to the database. The response will be shown on a table for the user to see a visual representation of their document upload.

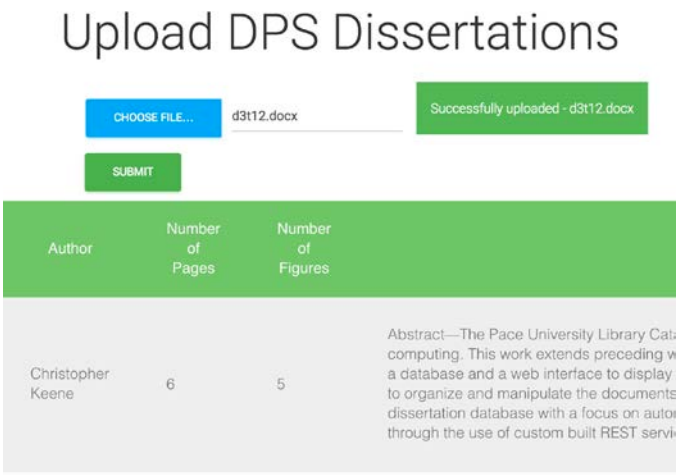

Figure 2. This is an example of the UI. The user can see the results of the upload and choose to submit it to the database.

The document payload is submitted to the REST API via an AJAX call to the endpoint "/api/upload". The rest of the work is done in the backend.

One of the API's that was used with the functionality of our API is Apache POI. Apache POI provides a series of Java libraries that is responsible for reading and writing files in the Microsoft Office formats, such as Microsoft Word and Excel. We just need it to read Word documents for our purposes.

Once the REST service is called on the backend, POI then receives the payload and has to get an input stream from it. POI has many built-in classes and methods that makes the process a little easier to code and read later for maintenance behind a layer of abstraction. After the document is extracted, the information in the document has to be parsed and stored individually in memory. This part is made slightly easier because all of the documents must be in the IEEE format. For example, the title should be the first line – or lines – of text in the document. The title should be followed by the authors, then Abstract, etc. The parsed data can then be sent directly to the database or used to populate the form on the admin screen. Theoretically, whatever can be entered into the database from the documents can be uploaded to the database autonomously.

```
XWPFDocument doc = new XWPFDocument(fileUpload.getInputStream());
XWPFWordExtractor ex = new XWPFWordExtractor(doc);

String text = ex.getText();
System.out.println(text);

List<XWPFParagraph> paragraphs = doc.getParagraphs();
for (XWPFParagraph p : paragraphs) {
    int index = paragraphs.indexOf(p);
    String paragraph = p.getParagraphText();
    System.out.println("Para: " + p.getParagraphText() + " index " + paragrap

    if(!paragraph.isEmpty() && index == 1) {
        String title = paragraph;
        System.out.println("TITLE: " + title);
    }

    TODO fix this to  multiple authors
    if(!paragraph.isEmpty() && index == 2) {
        String author = paragraph;
        System.out.println("AUTHOR: " + author);
        if(author.contains("and") || author.contains("&")) {
            Pattern r = Pattern.compile("([\\w\\s]+) and ([\\s\\w]+)");
            Matcher m = r.matcher(author);
            if (m.find( )) {
                System.out.println("Found value: " + m.group(0) );
                System.out.println("Found value: " + m.group(1) );
                System.out.println("Found value: " + m.group(2) );
              }else {
                System.out.println("NO MATCH");
              }
            System.out.println("IT DOES!!! " + author);
        }
    }

    TODO search ABSTRACT until the INTRO is found
    if(paragraph.contains("Abstract") || paragraph.contains("ABSTRACT")) {
        String abs = paragraph;
        System.out.println("ABSTRACT: " + abs);
    }
}
```

Figure 3. Early stages of the REST API code.

Going back to adding all the database columns individually, the code in figure 3 is an example showing how it will work. The first piece checks for the title and attempts to ensure that it is where it should be. XWPFParagraph is a POI class that distinguishes between a new line and a new paragraph. So we are iterating through that. The author is a little more complicated because there can be multiple authors. Currently, we only check for two (and it is incomplete) but this is to display what the API will be capable of. Once these are sorted, it will send the information to the database, or to the form so that the admin can verify the information before sending. Using this document produces an output of:

NUMBER OF FIGURES: 6
TITLE: Automating Database Uploads with Representational State Transfer Services
AUTHOR: Christopher Keene
ABSTRACT: Abstract—The Pace University Library Catalog houses the dissertations of all Doctor of Professional Studies in computing…
NUMBER OF PAGES: 6
KEYWORDS: RESTful services, REST, Database, MySQL, Spring, POI

C. *Document Classification*

Document classification is a subsection of machine learning (ML) with relation to natural language processing (NLP). The goal of document classification is to assign a category or class to a document. In the context of this project, we will be using it to autonomously categorize our dissertations based on the contents of the abstract. Automatic document classification can be split into three types: supervised document classification – which uses human intervention to provide information on the correct classification for the documents, unsupervised document classification – is when the classification is done without any external influence, and semi-supervised document classification – which is a little of both. We will need to use a supervised document classification, meaning we must have a training data set for this to work. We must make sure that we categorize our dissertations and have at least 10 abstracts per category to start.

Some of the ML techniques that are being considered for this task is the naïve Bayes classifier, support vector machines, or going the NLP approach through a third party service such as Google or Microsoft. The goal is to utilize the training set in an attempt to return reliable categories automatically for the dissertations.

D. *Automating The Dissertation Categories*

We will be categorizing the dissertations in the database programmatically based on the contents of the abstract. To accomplish this, we will leverage Google's Could NLP API. Google's Could NLP is a REST API that we can call and This is very useful because it will analyze the text from the abstract and parse information from the text into entities. This entity identification is what we will use to sort our categories. For example, when we send an abstract to Google's API, it will return a list of entities, or relevant text that it decides is related to the abstract.

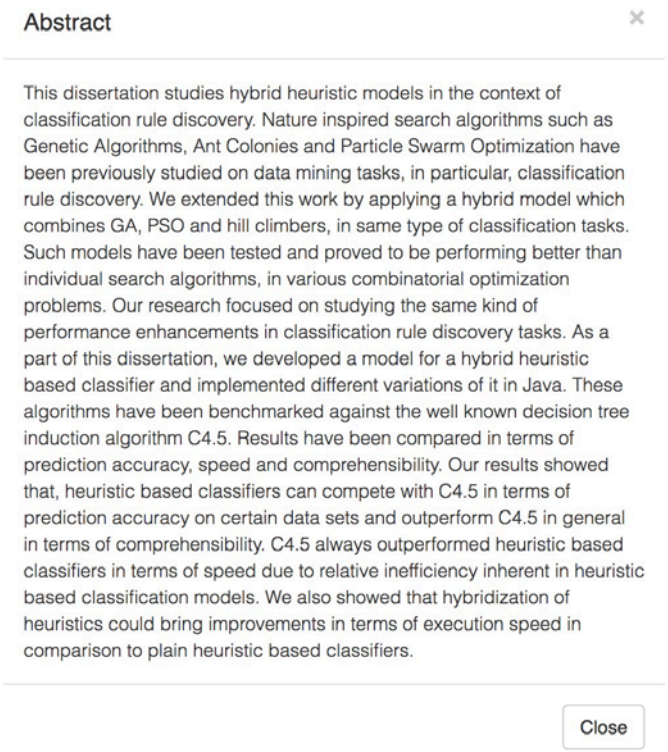

Figure 4. The abstract of a dissertation

Below, is a list of returned entities from the submitted abstract. The entity with the "1" subtext is the one with the highest salience, or importance relative to the document. Salience is measured from 0 – 1 with 1 representing the highest.

"Model" has the highest salience in the case of this example with a salience of 0.23. Model is followed by "classification of discovery," and that has a salience of 0.07. These entities are what we will try to analyze to determine categories for our dissertations.

1. Models, Consumer Good, Salience: 0.23
2. Classification rule discovery, Other, Salience: 0.07
3. Context, Other, Salience: 0.06

The examples above are listed with the word found in the abstract, the category, and the salience respectively.

E. Leveraging Google's Natural Language Processing API

To sort the dissertations into their respective categories we must utilize Google's NLP API. For now, we are using a personal API key. Pace University will need its own API key once the application is ready for production. Once you have signed up and registered with Google, they will send you a json file to download. This contains the authentication information needed to use their API. Here are the steps needed to make sure that everything is running correctly after signing up:

1. Add GOOGLE_APPLICATION_CREDENTIALS to the environmental variables in eclipse with a value of <path to the downloaded authentication json file>.
2. Create a json request and save it into a json file (entity-request.json).

```
{
  "document":{
    "type":"PLAIN_TEXT",
    "content":"Michelangelo Caravaggio, Italian painter, is known for 'The Calling of Saint Matthew'."
  },
  "encodingType":"UTF8"
}
```

3. Then authenticate the service account using this command: gcloud auth activate-service-account --key-file=***service-account-key-file***
4. You must then get the authorization token by inputing this command: gcloud auth print-access-token ***access_token***
5. Use that access token and place it in this curl command: curl -s -k -H "Content-Type: application/json" \ -H "Authorization: Bearer ***access_token***" \https://language.googleapis.com/v1/documents:analyzeEntities \ -d @entity-request.json

If a json response is returned after step 5, then everything should be running fine. That response contains the entities needed to sort the dissertation into a category. It is time to see this running through the application. This project utilizes maven, which is a dependency management tool. Maven makes it a lot easier than having to manage all of the jar files yourself. To get started with the NLP API, the Cloud Natural Language API Client Libraries must be installed. So instead of looking for yet another jar file and adding it to you class path, use simply add

```
<dependency>
        <groupId>com.google.cloud</groupId>
    <artifactId>google-cloud-language</artifactId>
        <version>0.11.0-alpha</version>
</dependency>
```

to the pom.xml file. Once this is complete, the application should be ready to request entities based on the dissertation abstract.

F. Future Work

The values that Google returns is not the end of the road. This gives you all the useful words from the abstract which can then be used to place the dissertation into a specific category. Due to time constraints, this part will not be complete. Though, the abstract, along with every database column that can be taken from looking at a dissertation, will be sorted and categorized autonomously. All the user has to do is upload the document via the UI and the rest should be handled on the backend.

One suggestion for handling the returned entities from Google would be to create a word bank with a list of associated words from each category. A word association website like http://www.wordassociations.net/ can be used to research associated words in every category. For example, one of the categories is computing. When computing is entered in the website, it returns a series of nouns, adjectives, and verbs that are associated with that word; words such as, machinery, informatics, grid, and around 30 more. That can be one way of categorizing the dissertations without any user input.

IV. Conclusion

The DPS dissertation database allows users to view the information about the DPS student's dissertations. This time we focused on streamlining the process for administrative users through the use of web services and a new user interface. These web services are providing a way to autonomously upload dissertation documents to the database by grouping sections of the dissertations into relevant columns while also categorizing the documents based on the contents of the abstract. This will give the user the option to fill out the available data in the form or upload directly to the database via the user interface. This new level of functionality should save time for the administrator and in turn save money for the school.

REFERENCES

[1] Downie, Nick. "Chart. js Documentation." *Dostopno na: http://www.chartjs.org/docs (marec 2014)* 65 (2014): 66.

[2] "PHP: PHP Manual - Manual", *Php.net*, 2016. [Online]. Available: http://php.net/manual/en/. [Accessed: 21- Feb- 2017].

[3] "AngularJS", *Docs.angularjs.org*, 2016. [Online]. Available: https://docs.angularjs.org/guide. [Accessed: 21- Feb- 2017].

[4] jquery.org, "jQuery API Documentation", *Api.jquery.com*, 2016. [Online]. Available: http://api.jquery.com/. [Accessed: 27- Feb- 2017].

[5] "MySQL : MySQL Documentation", *Dev.mysql.com*, 2016. [Online]. Available: http://dev.mysql.com/doc/. [Accessed: 27- Feb- 2017].

[6]"PDO Tutorial for MySQL Developers - Hashphp.org", *Wiki.hashphp.org*, 2016. [Online]. Available: http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers. [Accessed: 27- Feb- 2016].

[7]"Using The $http Service In AngularJS To Make AJAX Requests", *Bennadel.com*, 2016. [Online]. Available: https://www.bennadel.com/blog/2612-using-the-http-service-in-angularjs-to-make-ajax-requests.htm. [Accessed: 27- Feb- 2016].

[8]"Design Guidelines for Secure Web Applications", *Msdn.microsoft.com*, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/ff648647.aspx. [Accessed: 27- Feb- 2017].

[9] Rennie, J.; Shih, L.; Teevan, J.; Karger, D. (2003). *Tackling the poor assumptions of Naive Bayes classifiers* (PDF). ICML. [Accessed: 10- Feb- 2017].

[10] Santini, Marina; Rosso, Mark (2008), *Testing a Genre-Enabled Application: A Preliminary Assessment* (PDF), BCS IRSG Symposium: Future Directions in Information Access, London, UK
[Accessed: 10- Feb- 2017].

[11] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures* (Ph.D.). University of California, Irvine.
[Accessed: 21- Feb- 2017]

[12] T. II, "Principles of good RESTful API Design - Code Planet", *Code Planet*, 2017. [Online]. Available: https://codeplanet.io/principles-good-restful-api-design/. [Accessed: 06- Mar- 2017].

[13]"Apache POI - the Java API for Microsoft Documents", *Poi.apache.org*, 2017. [Online]. Available: https://poi.apache.org/. [Accessed: 08- Mar- 2017].

[14]"36. Cache Abstraction", *Docs.spring.io*, 2017. [Online]. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/cache.html. [Accessed: 08- Mar- 2017].

[15]"Cloud Natural Language API | Google Cloud Platform", *Google Cloud Platform*, 2017. [Online]. Available: https://cloud.google.com/natural-language/. [Accessed: 08- Mar- 2017].

[16]"Spring Boot file upload example – Ajax and REST", *Mkyong.com*, 2017. [Online]. Available: https://www.mkyong.com/spring-boot/spring-boot-file-upload-example-ajax-and-rest/. [Accessed: 08- Mar- 2017].

[17]"Quickstart | Google Cloud Natural Language API Documentation | Google Cloud Platform", *Google Cloud Platform*, 2017. [Online]. Available: https://cloud.google.com/natural-language/docs/getting-started. [Accessed: 02- Apr- 2017].