Social Networks and Archival Context OpenRefine Plugin

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Charles Michael Chang
Spring 2020.

Technical Project Team Members
Sandra Gould
Mark Jeong
John Perez
Victor Shen
Peter Tran
Grace Wu
Jessica Xu

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature [illegible] Date 04/01/2020
Charles Michael Chang

Approved Ahmed Ibrahim Date 04/29/2020
Dr. Ahmed Ibrahim, Department of Computer Science

Table of Contents

Abstract

Social Networks and Archival Context (SNAC) is an archival institution that works with a variety of other institutions to build an archival collection, but each of these institutions has a different structure for storing records. Relationships between different entities, labels for certain types of data, and the hierarchy of the data itself are inconsistent from each outside institution. SNAC needs to reconcile the differences between the outside data and its own data storage structure before importing the data into its database. It is extremely impractical and time consuming to clean up the data manually. The goal of the technical project was to develop a simple and fault-proof interface for reconciling that would be easy for users to learn and use.

The technical project consisted of the development of a standalone plugin for SNAC using OpenRefine, an existing open source project. The plugin has enabled participants to upload and consolidate large amounts of data on historical artifacts. With the extension, hundreds and thousands of entries can now be processed and pushed to SNAC in a couple minutes. The extension is able to import external data and match it to existing data within the SNAC server. Given that SNAC is an archival institution, it was very important that the information being uploaded would update the right fields for any given entry. The plugin effectively solved the problem of data mismatch when uploading information to the SNAC site. The project achieves simplicity without excessive functionality, and it creates a more efficient and streamlined process to import and update thousands of data entries. The plugin ensures that the reconciliation of data imported is both quick and accurate, which is crucial to an organization like SNAC where efficient and accurate querying are key.

List of Figures

1. Introduction

In the current day and age, data is abundant yet sometimes excessive, which can lead to information mismatch. How can this data be managed and archived so that related information regarding specific topics can be consolidated in an efficient manner? Our capstone group sought to solve this through data wrangling, or the mapping of raw data into another format, by developing a plugin for our project, Social Networks and Archival Context (SNAC).

SNAC, a website similar to Wikipedia, is a free, online resource that helps users discover biographical and historical information about people, families, and organizations that are created and are documented in historical resources (primary source documents) and their connections to one another (Social Networks and Archival Context [SNAC], n.d.). SNAC is used to locate archived collections as well as related resources held around the world. As an international cooperative, SNAC works to "build a corpus of reliable descriptions of people and artifacts that link to and "provide contextual understanding" of historical records (SNAC, n.d., para. 1). In order to create these contextual connections, SNAC sources its information from many different libraries and archival institutions. SNAC cooperates with over 4,000 institutions to gather and reconcile data (SNAC, n.d.)

The SNAC user base is extremely active: as of October 28, 2019, there have been almost 700 edits per week. There are currently 136 active users that contribute to the SNAC database and over 33,000 visitors per month on average. In total, there are over two million resources that have been created and edited since the creation of SNAC (Jeong, 2019). Our team's interest in maintaining and improving such a widely used resource is what originally drew us to pursue this project.

1.1 Problem Statement

While SNAC seeks to serve a variety of institutions and build an archival collection, each institution has a different structure for storing records. Relationships between different entities, labels for certain types of data, and the hierarchy of the data itself are inconsistent from each outside institution. SNAC needs to reconcile the differences between the outside data and its own data storage structure before importing the data into its database. It is extremely impractical to clean up the data manually or with simple tools (Ham, 2013). The reconciliation of this data is vital to the functionality of an archival organization such as SNAC because it is crucial for efficient and accurate querying (Park, 2008).

The technical project seeks to develop a standalone plugin for Social Networks and Archival Context (SNAC) using OpenRefine. OpenRefine is an open source software that is community-maintained and designed specifically for data normalization, transformation, and cleaning (Hill, 2016). It allows users to import and normalize data with a series of pre-existing default user interfaces after connecting to a target resource. OpenRefine provides a “powerful yet user-friendly interface” for experimenting with and querying data (Hill, 2016, p. 228).
With over 700 edits occurring to its data schema in a week, Social Networks and Archival Context (SNAC) is no small data archive (SNAC, n.d.). The current workflow for refining and updating data in SNAC is quite difficult and inaccessible to inexperienced users. It involves users hitting SNAC’s APIs for refining data on their server from the user’s local machine. The technical project aims to greatly simplify this process by creating a streamlined plugin that will have all the functionalities needed to refine and upload data in one location. The logical flow and components needed for the project are illustrated in Figure 1. The plugin will serve as a connection between the user’s local data and SNAC’s server. It will allow users to import

external data in the form of comma-separated values (CSV) files and make use of APIs provided by SNAC to reconcile and refine that data with SNAC’s unique JavaScript Object Notation (JSON) data structure. The plugin will have two main user groups: privileged and unprivileged users. Both types of users will be able to use the plugin to format any data ported in using SNAC’s organizational schema. Only privileged users will be able to then push the formatted data into SNAC’s own database utilizing the APIs provided by SNAC. The technical project will provide an easy way to reconcile outside data with SNAC’s existing data in addition with an improved user interface for an enhanced user experience.

1.2 Contributions

Over the course of eight months, the team was able to create a plugin extension using OpenRefine to import external data and match it to existing data within the SNAC server. With SNAC being a data archive, it was very important that the information being uploaded updated the right fields for any given entry. The plugin effectively solved the problem of data mismatch when uploading information to the SNAC site. The plugin achieves simplicity without excessive functionality, and it creates a more efficient and streamlined process. SNAC is easily able to import and update thousands of data entries. The

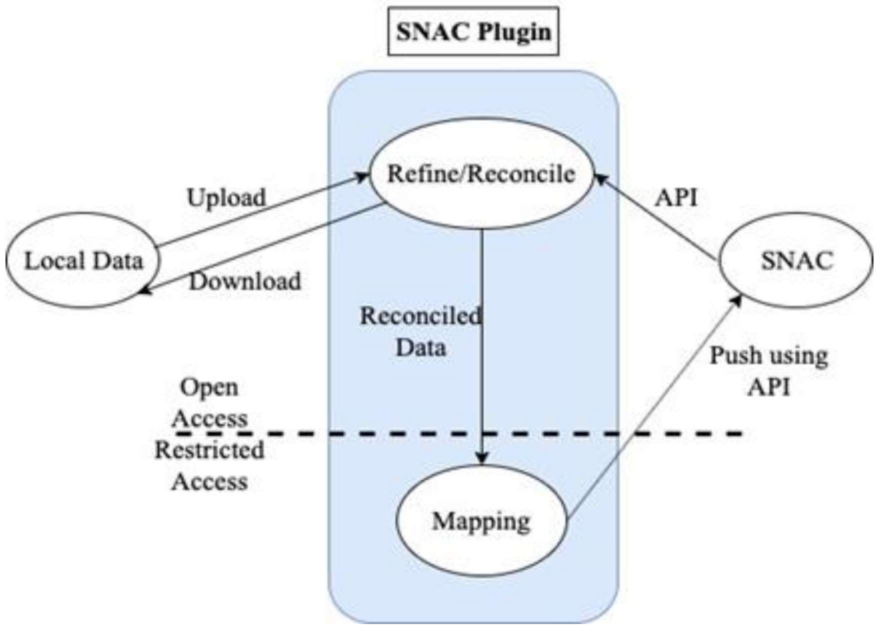


Figure 1: SNAC Plugin Model: An overview of the design of the plugin, depicting the different processes and functions that will be made available by the plugin (Xu, 2019).

features contributed to the final product and functionality offered by the plugin are expounded in the following sections of this thesis.

In the remaining of this thesis, section 2 presents related work. Section 3 elucidates the system design and required functionality of the plugin. Section 4 discusses the results of the technical project as it relates to the initial problem. Finally, Section 6 provides a conclusion to the work done in the past year, while section 7 gives way to future work on the project.

2. Related Work

According to their website, SNAC serves as an "online resource that helps users discover biographical and historical information about persons, families, and organizations that created or are documented in historical resources (primary documents) and their connections to one another" based on the idea of openly editable content (About SNAC, n.d.). Only members of the cooperative like select museums, universities, and libraries, and archives can edit data. This model is identical to that of Wikipedia. The problem with Wikipedia is that anyone on the Internet can edit any entry in their online encyclopedia. Inaccurate information, whether inserted accidentally or maliciously can persist on the site anywhere from a few hours to nearly two years (Criticism of Wikipedia, 2020). SNAC avoids this problem by allowing any user to browse the entries but only approved users can edit the information within the database. Though this essentially guarantees trustworthy information, the cost of this solution is that information will be entered into SNAC at a slower rate compared to Wikipedia.

A key component to SNAC is the OpenRefine (previously, a private endeavor called Google Refine) plugin. According to their homepage, OpenRefine is a great tool for "working with messy data; cleaning it; transforming it from one format into another; and extending it with web services and external data" (OpenRefine, 2010). After Google discontinued support of the project, it became open source and is now the primary interface that approved users interact with when editing data in SNAC. OpenRefine has a web user interface but is not hosted on the web. Rather, it must be downloaded and run locally. The myriad of editing tools available makes OpenRefine a very powerful tool, especially compared to the system that Wikipedia uses. Edits to Wikipedia include simply changing the text on a webpage. Any other instances of that same data on other webpages will not be changed in parallel. Seemingly the opposite of Wikipedia, SNAC is designed to be editable by reliable entities, always contain accurate information, and give editors powerful tools for working with large and varied datasets.

3. System Design

The goal of our project is to provide an interface for users to clean and manipulate historical data that will be then uploaded to the SNAC database. Our plugin was intended for members of cooperatives (such as that of institutions, universities, and libraries that are a part of the SNAC Cooperative) so that they could more easily upload data. While the average user (i.e. users who are not a part of the SNAC Cooperative) has access to the tool, as it is open source and available on GitHub, they will not be able to upload data without the proper credentials. Therefore, most users who intend to use SNAC will most likely use the website to look for information.

The backend was mainly implemented in Java while the frontend user interface was implemented in HTML/Javascript/CSS. There was not a choice of which programming languages to use because most of the plugin was already built out with these languages, and our project was mainly to add onto this existing project. Our code is licensed under the BSD3-Clause, a license allowing freedom in letting individuals use code without needing to disclose the source code or libraries as long as users address the copyright notice (The 3-Clause BSD License).

3.1 System Requirements

It is important to have system requirements so that the customer and the development team can have a clear understanding of what the project's long-term goals and timeline are. While there is potential for system requirements to change, having the system requirements set in the beginning will allow for the development team to have more productive sprints and accomplish what was set out more effectively.

Therefore, our system requirements (minimum and desired) are as follows:

Minimum requirements:

- Allowing users to import CSV data into the plugin
- Connect the data fields with different SNAC IDs
- Search for constellations in SNAC and match them to the imported data
- Allow a human editor to choose from several options to match for when the plugin is unsure
- Reconcile the imported changes based on the connection and matches
- Download the data that is now reconciled with SNAC's structure
- Users with privileges will be able to publish the data to SNAC

Desired requirements:

- Users will be able to reconcile more complex data items like relationships and geolocations
- Users will be able to edit already existing resources and constellations

So far, no optional requirements have been specified by the client.

3.2 Wireframes

Wireframes are an important tool for communication during development. It allowed the team to effectively understand and visualize the requirements requested by the client. Below are the initial wireframes for the project.

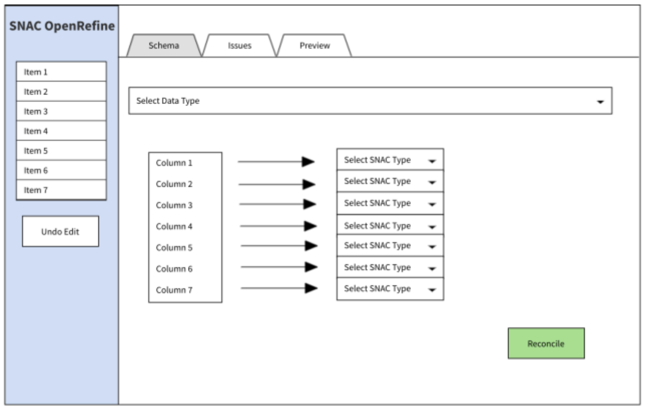


Figure 3.1.1: Schema Wireframe

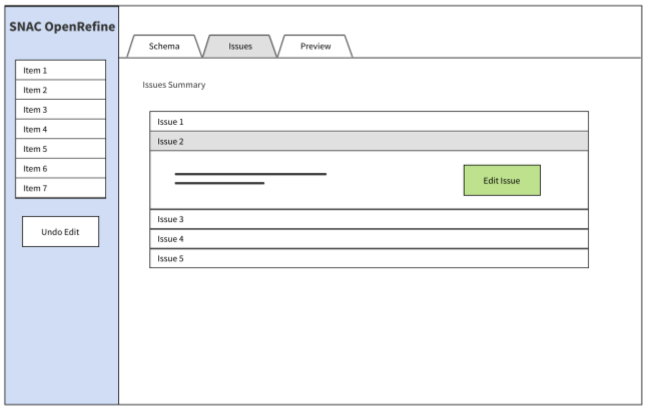


Figure 3.1.2 - Issues Wireframe

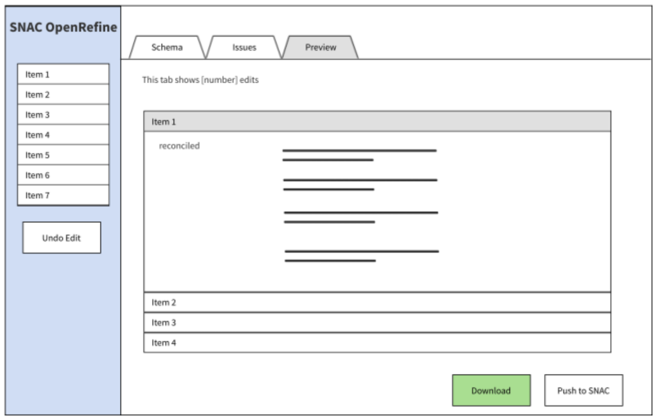
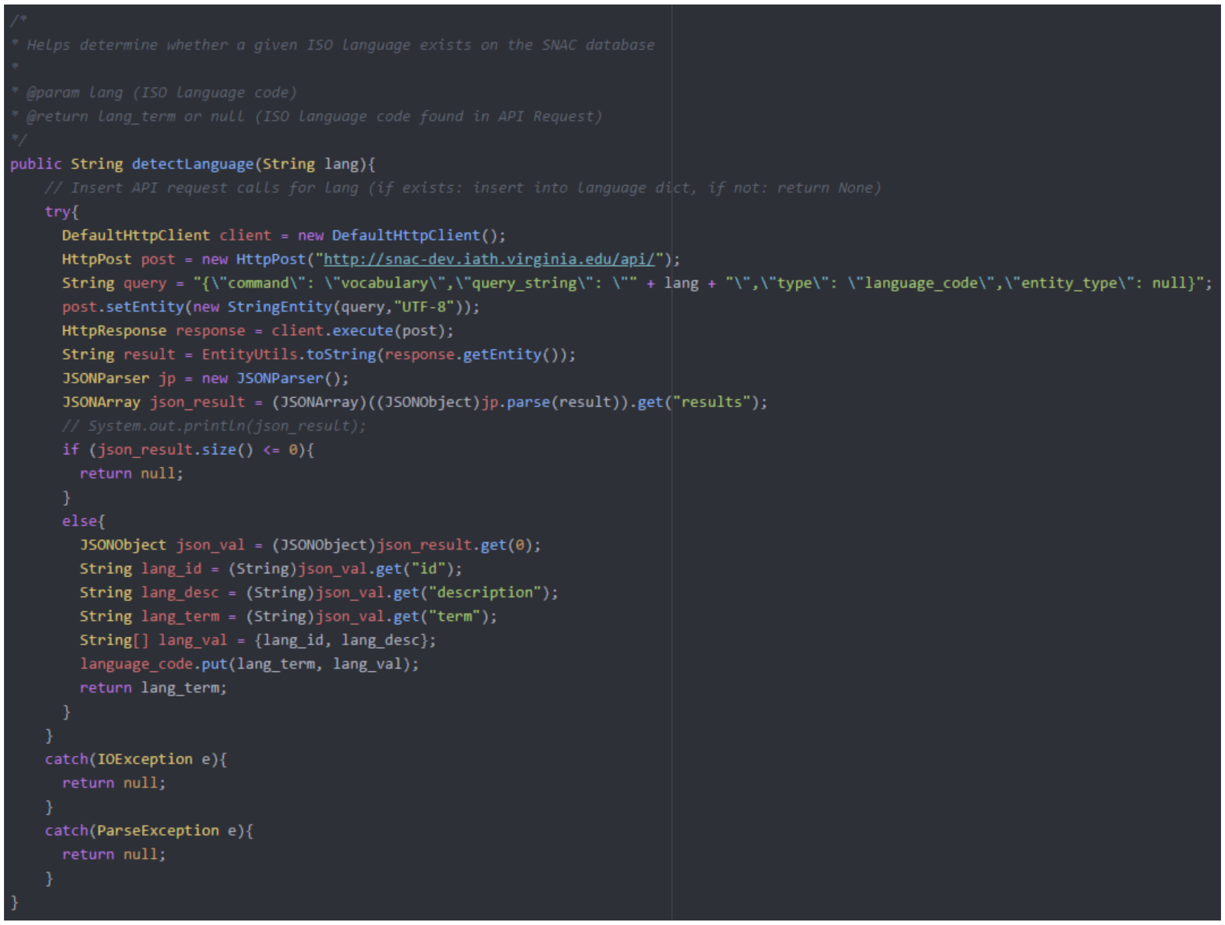


Figure 3.1.3 - Preview Wireframe

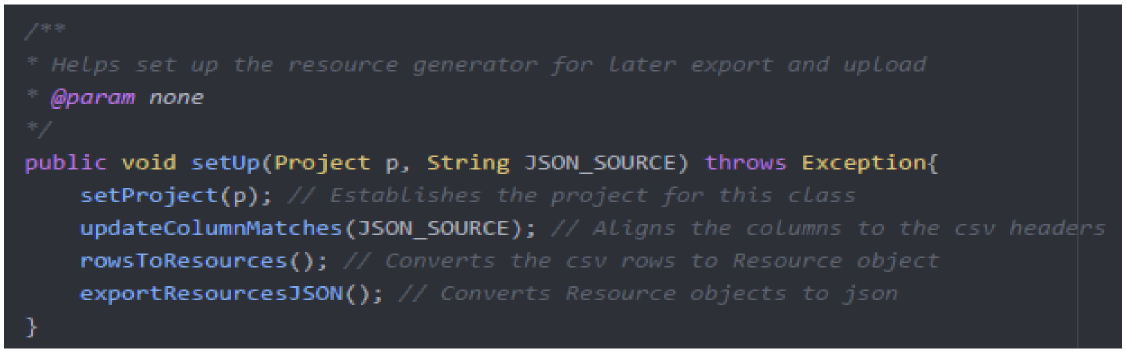
3.3 Sample Code



```java
/*
* Helps determine whether a given ISO language exists on the SNAC database
*
* @param lang (ISO language code)
* @return lang_term or null (ISO language code found in API Request)
*/
public String detectLanguage(String lang){
    // Insert API request calls for lang (if exists: insert into language dict, if not: return None)
    try{
      DefaultHttpClient client = new DefaultHttpClient();
      HttpPost post = new HttpPost("http://snac-dev.iath.virginia.edu/api/");
      String query = "{\"command\": \"vocabulary\",\"query_string\": \"" + lang + "\",\"type\": \"language_code\",\"entity_type\": null}";
      post.setEntity(new StringEntity(query,"UTF-8"));
      HttpResponse response = client.execute(post);
      String result = EntityUtils.toString(response.getEntity());
      JSONParser jp = new JSONParser();
      JSONArray json_result = (JSONArray)((JSONObject)jp.parse(result)).get("results");
      // System.out.println(json_result);
      if (json_result.size() <= 0){
        return null;
      }
      else{
        JSONObject json_val = (JSONObject)json_result.get(0);
        String lang_id = (String)json_val.get("id");
        String lang_desc = (String)json_val.get("description");
        String lang_term = (String)json_val.get("term");
        String[] lang_val = {lang_id, lang_desc};
        language_code.put(lang_term, lang_val);
        return lang_term;
      }
    }
    catch(IOException e){
      return null;
    }
    catch(ParseException e){
      return null;
    }
}
```

Figure 3.3.1: ***detectLanguage***

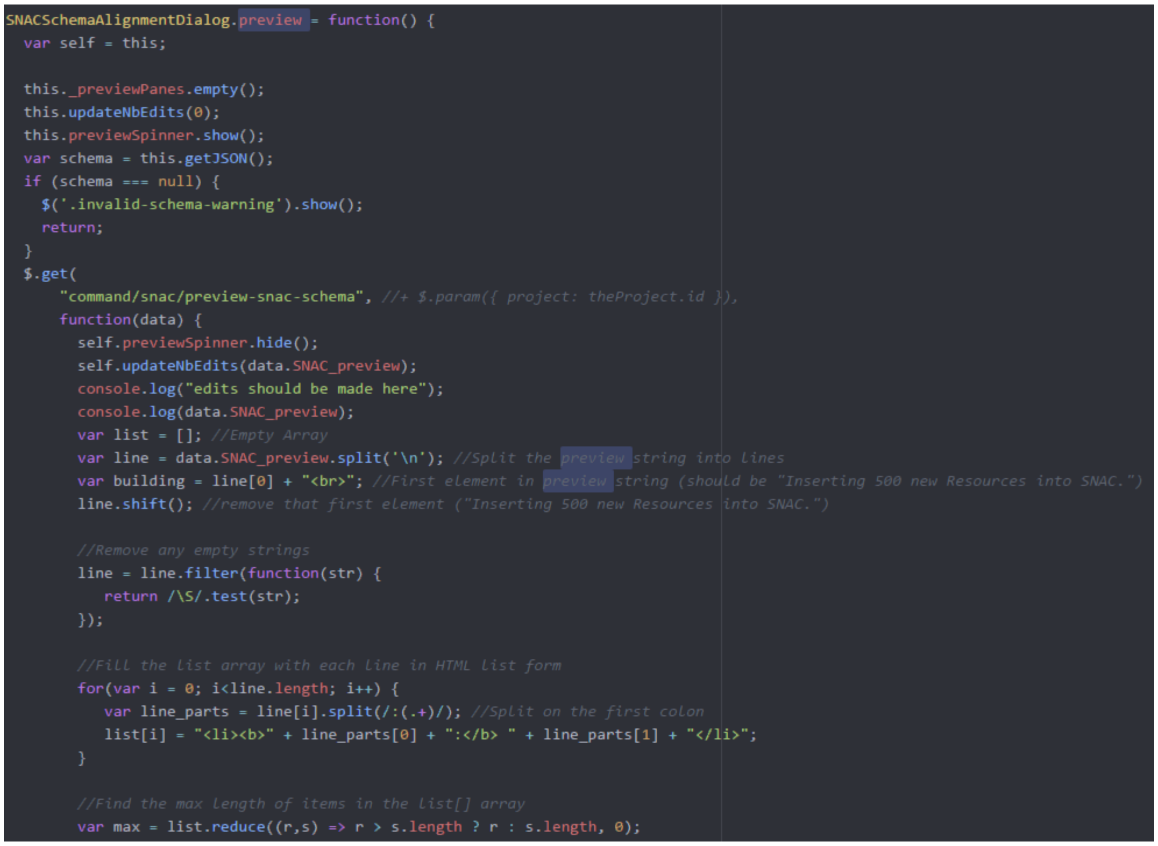
A function used as a validator for languages associated with a given resource object. It takes a given ISO code from the resource and runs it through SNAC's database to make sure such ISO code is supported before inserting it into SNAC. If it exists, return the result string (which contains language description, name, and id) and store them into an array for later purposes, such as displaying them in the preview tab. If it does NOT exist, then proceed onto the next resource object.



```
/**
* Helps set up the resource generator for later export and upload
* @param none
*/
public void setUp(Project p, String JSON_SOURCE) throws Exception{
    setProject(p); // Establishes the project for this class
    updateColumnMatches(JSON_SOURCE); // Aligns the columns to the csv headers
    rowsToResources(); // Converts the csv rows to Resource object
    exportResourcesJSON(); // Converts Resource objects to json
}
```

Figure 3.3.2: ***setUp***

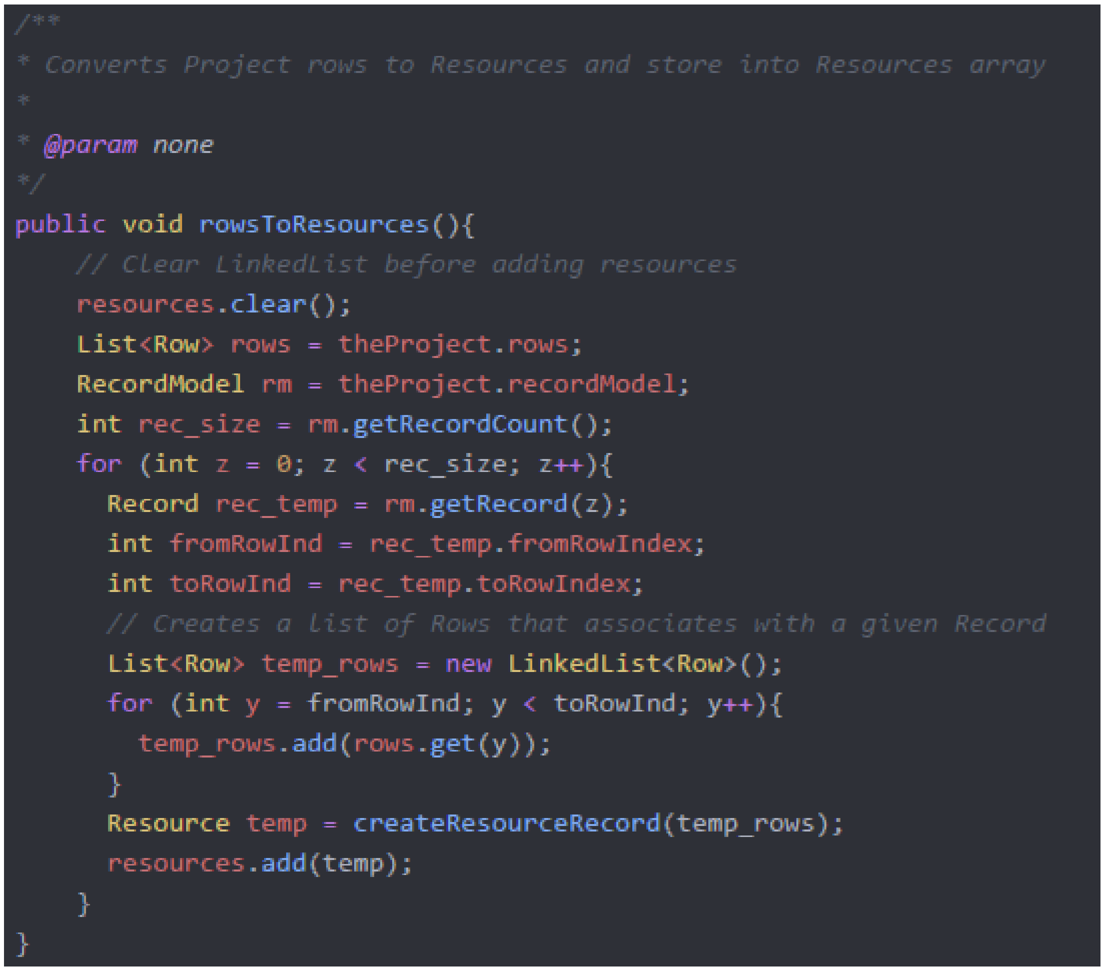
Essentially the brain of the whole resource building operation. There are four components to each operation; ***1. setProject:*** *gives the backend a reference to the "project" object which contains all the data from the imported csv.* ***2. updateColumnMatches:*** *Transfers the column headers that were aligned by the user to the backend to help with locating cell values within the "project" object.* ***3. rowsToResources:*** *Converts row objects within project into resource objects.* ***4. exportResourcesJSON:*** *Converts our resources into a json format that can later be exported by our users.*



```
SNACSchemaAlignmentDialog.preview = function() {
  var self = this;

  this._previewPanes.empty();
  this.updateNbEdits(0);
  this.previewSpinner.show();
  var schema = this.getJSON();
  if (schema === null) {
    $('.invalid-schema-warning').show();
    return;
  }
  $.get(
      "command/snac/preview-snac-schema", //+ $.param({ project: theProject.id }),
      function(data) {
        self.previewSpinner.hide();
        self.updateNbEdits(data.SNAC_preview);
        console.log("edits should be made here");
        console.log(data.SNAC_preview);
        var list = []; //Empty Array
        var line = data.SNAC_preview.split('\n'); //Split the preview string into lines
        var building = line[0] + "<br>"; //First element in preview string (should be "Inserting 500 new Resources into SNAC.")
        line.shift(); //remove that first element ("Inserting 500 new Resources into SNAC.")

        //Remove any empty strings
        line = line.filter(function(str) {
           return /\S/.test(str);
        });

        //Fill the list array with each line in HTML list form
        for(var i = 0; i<line.length; i++) {
           var line_parts = line[i].split(/:(.+)/); //Split on the first colon
           list[i] = "<li><b>" + line_parts[0] + ":</b> " + line_parts[1] + "</li>";
        }

        //Find the max length of items in the list[] array
        var max = list.reduce((r,s) => r > s.length ? r : s.length, 0);
```

Figure 3.3.3: ***rowsToResources***

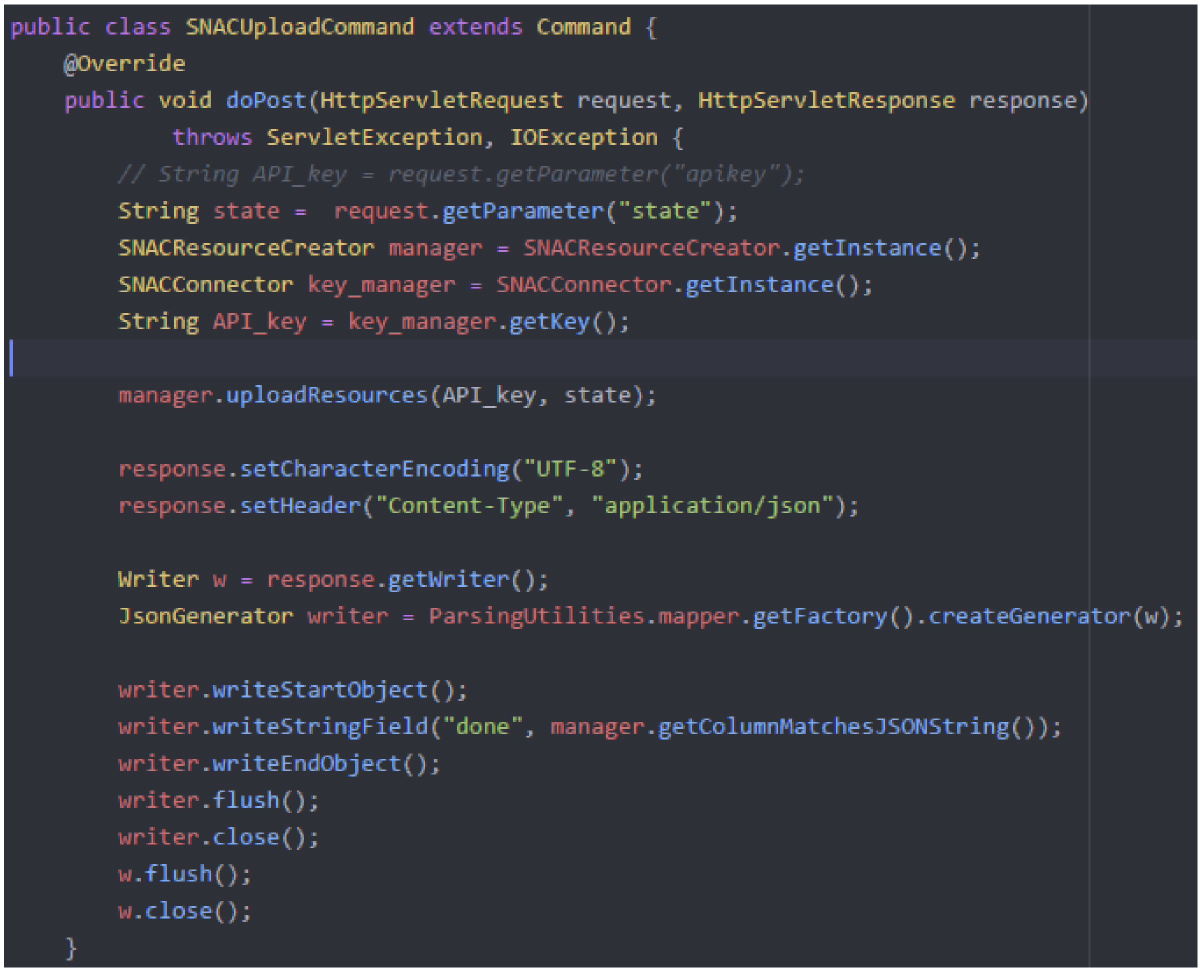
*Extracts rows from the given "project" and converts them into resource objects based on the cell values within those rows. Specifically, one resource can span more than one row if they have multiple values for a given column. OpenRefine calls these "****records****". For a given record, we know the starting and ending row index thus allowing us to group all the rows associated with an entity and return a list of rows used to build out one resource object.*



```
/**
* Converts Project rows to Resources and store into Resources array
*
* @param none
*/
public void rowsToResources(){
    // Clear LinkedList before adding resources
    resources.clear();
    List<Row> rows = theProject.rows;
    RecordModel rm = theProject.recordModel;
    int rec_size = rm.getRecordCount();
    for (int z = 0; z < rec_size; z++){
      Record rec_temp = rm.getRecord(z);
      int fromRowInd = rec_temp.fromRowIndex;
      int toRowInd = rec_temp.toRowIndex;
      // Creates a list of Rows that associates with a given Record
      List<Row> temp_rows = new LinkedList<Row>();
      for (int y = fromRowInd; y < toRowInd; y++){
        temp_rows.add(rows.get(y));
      }
      Resource temp = createResourceRecord(temp_rows);
      resources.add(temp);
    }
}
```

Figure 3.3.4: Front-end example

This is an example function you'll find on the javascript (front-end) portion of our project within ***SNACSchemaAlignmentDialog.js****. Everything found on this js file is what users see when using our plugin. For this specific* ***preview*** *function, it calls onto the backend using* ***"get"*** *to obtain one or two resources/constellations as a string that will be displayed on the preview tab for users to see before pushing all objects onto the SNAC database.*



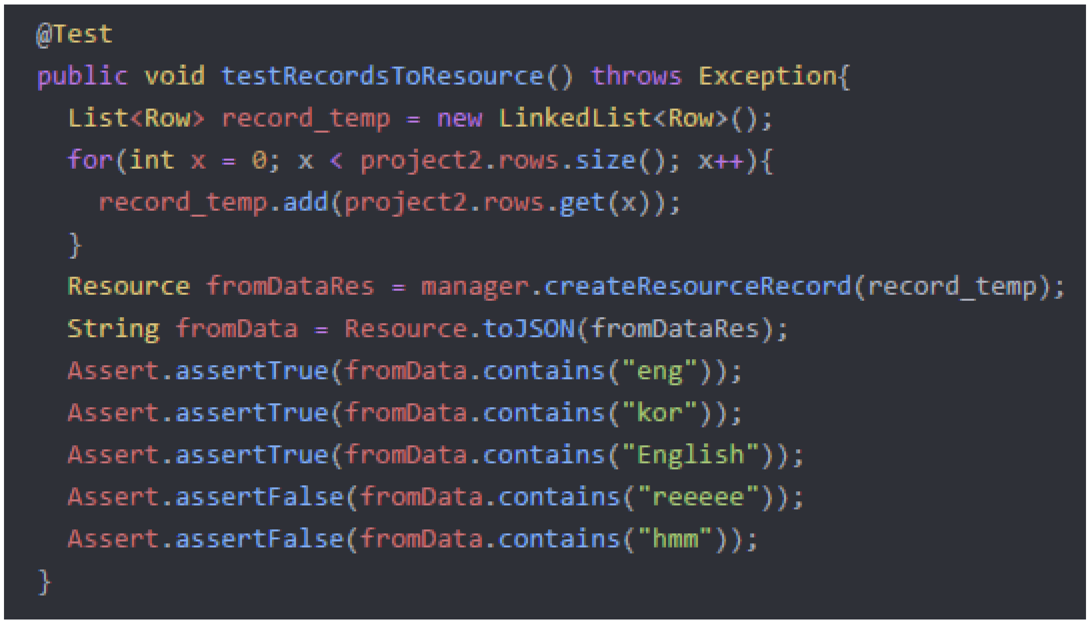
```java
public class SNACUploadCommand extends Command {
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // String API_key = request.getParameter("apikey");
        String state =  request.getParameter("state");
        SNACResourceCreator manager = SNACResourceCreator.getInstance();
        SNACConnector key_manager = SNACConnector.getInstance();
        String API_key = key_manager.getKey();

        manager.uploadResources(API_key, state);

        response.setCharacterEncoding("UTF-8");
        response.setHeader("Content-Type", "application/json");

        Writer w = response.getWriter();
        JsonGenerator writer = ParsingUtilities.mapper.getFactory().createGenerator(w);

        writer.writeStartObject();
        writer.writeStringField("done", manager.getColumnMatchesJSONString());
        writer.writeEndObject();
        writer.flush();
        writer.close();
        w.flush();
        w.close();
    }
```

*Figure 3.3.5: **doPost** example*

*A key function in allowing the plugin to transfer information from the frontend to the backend. Anytime the front end uses the function **post**, doPost will activate. This specific doPost is used for uploading resources into SNAC. When the user clicks on "Upload to SNAC" in the plugin, the frontend will call post which tells the backend to take all the resources it generated and upload them onto SNAC.*

3.4 Sample Tests

While implementing unit tests may seem redundant at first, eventually these tests will prove to be extremely impactful while developing new features. Testing allows developers to know if the product fails due to a new feature or an edit on an existing feature. As more tests are being developed, the easier it becomes to hone in on the portion of the code that may have caused a build failure thus effectively improves debugging without the need to manually test every functionality.

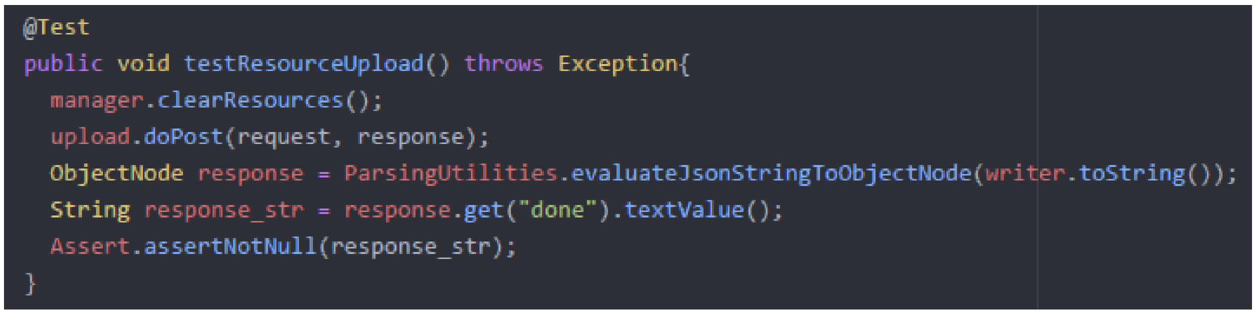


```java
@Test
public void testRecordsToResource() throws Exception{
  List<Row> record_temp = new LinkedList<Row>();
  for(int x = 0; x < project2.rows.size(); x++){
    record_temp.add(project2.rows.get(x));
  }
  Resource fromDataRes = manager.createResourceRecord(record_temp);
  String fromData = Resource.toJSON(fromDataRes);
  Assert.assertTrue(fromData.contains("eng"));
  Assert.assertTrue(fromData.contains("kor"));
  Assert.assertTrue(fromData.contains("English"));
  Assert.assertFalse(fromData.contains("reeeee"));
  Assert.assertFalse(fromData.contains("hmm"));
}
```

Figure 3.4.1: Tests for the converting functionality

Records from the imported csv into SNAC Resource objects that are to be uploaded to the SNAC Database. Specifically it checks for values that are and aren't in a created Resource object.

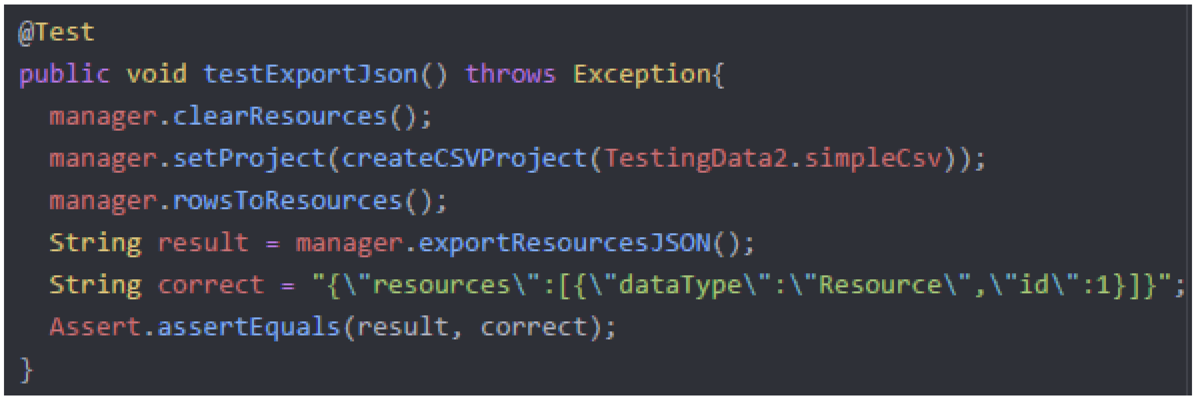
3.5 Code Coverage



```
@Test
public void testResourceUpload() throws Exception{
  manager.clearResources();
  upload.doPost(request, response);
  ObjectNode response = ParsingUtilities.evaluateJsonStringToObjectNode(writer.toString());
  String response_str = response.get("done").textValue();
  Assert.assertNotNull(response_str);
}
```

Figure 3.4.2: Tests for the upload functionality

Checks our plugin to see if we are able to connect to the SNAC's server through SNAC AP.I



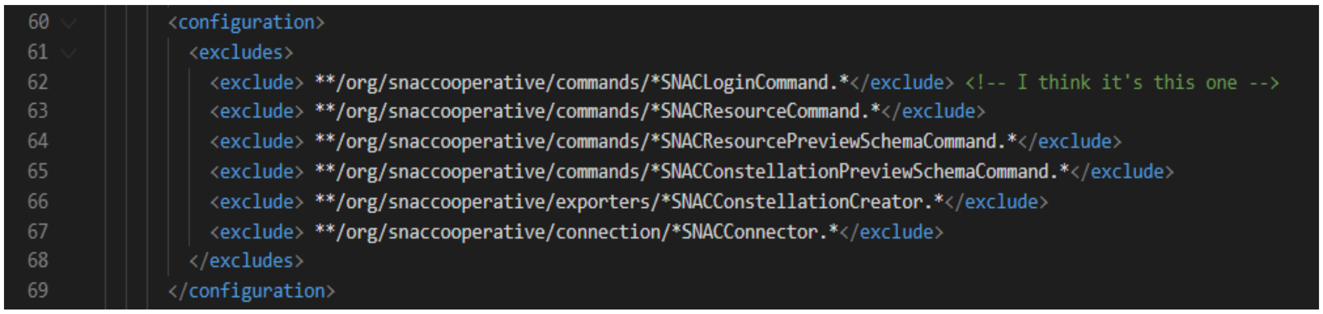
```
@Test
public void testExportJson() throws Exception{
  manager.clearResources();
  manager.setProject(createCSVProject(TestingData2.simpleCsv));
  manager.rowsToResources();
  String result = manager.exportResourcesJSON();
  String correct = "{\"resources\":[{\"dataType\":\"Resource\",\"id\":1}]}";
  Assert.assertEquals(result, correct);
}
```

Figure 3.4.3: Tests for the export functionality

Export the Resources that users generate. Specifically, this test checks whether the given exported json is correctly formatted.

Our chosen code coverage tool, Coveralls.io, is a web service that allows developers to track code coverage over time and ensures that all new developed code is fully covered. We chose Coveralls.io because the original OpenRefine repository also used Coveralls.io as their code coverage tool, allowing for easy integration between the code coverage tool and the code base.

Setting up Coveralls.io was relatively simple. Since Coveralls.io is a web based tool, we simply had to connect the Github repository to Coveralls.io. Coveralls.io was configured such that code coverage runs whenever branch "capstone-master" (our main branch) was updated and allowed for each developer to independently do work on their own branch before merging into "capstone-master". In addition, we also configured the main pom.xml (/pom.xml) such that Coveralls.io was recognized as a package by the repository. In order to ensure that code coverage would be 100%, we had to exclude certain files within the SNAC extension via /extensions/pom.xml. Since these files cannot be tested, we excluded them from code coverage.



```
<configuration>
  <excludes>
    <exclude> **/org/snaccooperative/commands/*SNACLoginCommand.*</exclude> <!-- I think it's this one -->
    <exclude> **/org/snaccooperative/commands/*SNACResourceCommand.*</exclude>
    <exclude> **/org/snaccooperative/commands/*SNACResourcePreviewSchemaCommand.*</exclude>
    <exclude> **/org/snaccooperative/commands/*SNACConstellationPreviewSchemaCommand.*</exclude>
    <exclude> **/org/snaccooperative/exporters/*SNACConstellationCreator.*</exclude>
    <exclude> **/org/snaccooperative/connection/*SNACConnector.*</exclude>
  </excludes>
</configuration>
```

Figure 3.5.1: Excluded files

Screenshot of files that were excluded from code coverage in /extensions/pom.xml.

Our total code coverage value can be found directly on our GitHub repository (https://github.com/snac-cooperative/OpenRefine/tree/capstone-master) in the README.md as a badge titled "coverage". More information can be displayed once clicking on the badge or going to https://coveralls.io/github/snac-cooperative/OpenRefine?branch=capstone-master. Note that the total coverage is around 45%. This is because there are other extensions within the OpenRefine repository such as database, gdata, jython, etc. that were developed prior to our team starting development on the SNAC extension. Since we are not working on the other extensions, we are not responsible for the code coverage of those extensions. The code coverage within the SNAC extension, therefore, is much higher than the total coverage. By viewing the source files on Coveralls.io, it is possible to view the code coverage of each individual file and even the lines that are covered within each file.

3.6 Installation Instructions

For Mac OS/Linux users:

1. Before proceeding, take note of the required packages/tools you'll need to run the program. (See general troubleshooting for required packages/tools).
2. Navigate to https://github.com/snac-cooperative/OpenRefine/tree/capstone-master.
3. Click on the "Clone or download" button and then copy the URL under "Clone with HTTPS" then run git clone https://github.com/snac-cooperative/OpenRefine.git (the link that you copied from "Clone with HTTPS").
4. Navigate into the root directory of the repository you cloned.
5. Switch to the capstone-master branch: git checkout capstone-master
6. Run git pull origin capstone-master
7. Run ./refine build in the terminal.
8. Run ./refine in the terminal.
9. A local build should open in your web browser that you should be able to access.

For Windows users:

1. Before proceeding, take note of the required packages/tools you'll need to run the program. (See general troubleshooting for required packages/tools).
2. Navigate to https://github.com/snac-cooperative/OpenRefine/tree/capstone-master.
3. Click on the "Clone or download" button and then copy the URL under "Clone with HTTPS", then run git clone https://github.com/snac-cooperative/OpenRefine.git (the link that you copied from "Clone with HTTPS").
4. Navigate into the root directory of the repository you cloned.
5. Switch to the capstone-master branch: git checkout capstone-master
6. Run git pull origin capstone-master
7. Run ./refine build in Git Bash or any Linux-based terminal. This should build dependencies for the project (it may take a while).
8. Run ./refine.bat to open a localhost version of the project.
9. A local build should open in your web browser that you should be able to access.

General troubleshooting:

- The application should still run fine despite warnings from the tool about having too new of a Java version.
- You should have JDK 8 and Apache Maven (primarily used for testing of the project) installed.
    - Failing ./refine test for reasons other than failed test cases may be caused by not having Maven properly installed. Please have a Maven version of at least 3.6.3 and set the location of the unzipped package (ex. ../apache-maven-3.6.3) to an environment variable called MVN_HOME.
    - If you run ./refine test and it gives you the error of “No such file or folder”, it may be because there’s a space in your file path, i.e. “Desktop/**My Files**/Openrefine”. The workaround solution right now is to change your file path so that it doesn’t have spaces either by moving the project around or changing the name of the directory with a space in it. i.e. “Desktop/**My_Files**/Openrefine”.
- Sometimes, running mvn clean install rebuilds a clean version of the dependencies and can resolve “weird” issues.

Troubleshooting for Windows users:

- If an error similar to “not able to find Google.Refine.refine” appears, attempt to run ./refine build followed by ./refine.bat in terminal. Another option is to click the “refine.bat” file in the directory itself, which runs the script to open the localhost.
- If there is an error that complains about your JAVA_HOME environment not being configured correctly, check to see that echo %JAVA_HOME% works within the terminal to see if the correct Java file path appears.
    - If you need to change your environment variables, go to System Properties → Environment Variables. Under “System variables”, there should be a variable called “JAVA_HOME” with a value of something like “C:\Program Files\Java\jdk1.8.0_131” or wherever you have your jdk located on your local computer. If you do not have this configured, you should set it up.

Troubleshooting for Mac users:

- If an error similar to "not able to find Google.Refine.refine" appears, attempt to run ./refine build followed by ./refine.bat in terminal. Another option is to click the "refine.bat" file in the directory itself, which runs the script to open the localhost.
- If there is an error that complains about your JAVA_HOME environment not being configured correctly, check to see that echo $JAVA_HOME works within the terminal to see if the correct Java file path appears.
    - If it does not show a Java version, then you will need a Java version past 8 and set the JAVA_HOME environment variable for MacOS. See Aggarwal (2018) or "How to set JAVA_HOME environment variable on Mac OS X 10.9?" (2014) for help with this.

Other Resources:

- *Installation Instructions* (OpenRefine, 2019)
- *OpenRefine Installing Java* (OpenRefine, 2019)

4. Results

The plugin has enabled participating institutions and individuals to upload and consolidate certain large tracts of data on historical artifacts. As originally intended, the project produced a stable product in the form of an extension to an existing open source project that solves most of the problems presented by the client. The client is pleased with the progress that had been made with the intention of the project being assigned to future developers for a later date for continued development. The client in particular is also a developer in the same associated space. The developers work on the API and web-server side that our tool communicates with.

This tool helps guide the development on their side to be more compatible with our work so that it may be readable and passes on without trouble. As for the actual customer base this will eventually be shuttled to, the customer can use this tool by downloading and installing dependencies as stated in the installation instructions. Once done, they can upload their data through the tool, consolidate changes, and push them onto the official SNAC database. Usually, inserting records into the database via the website interface takes a few minutes every entry, which leads hundreds of entries, which takes hours in a day. With the tool, hundreds of entries can now be processed and pushed to SNAC in a couple minutes. With 34 participating institutions around the world and counting, 2 million resources and counting, and 33,000 SNAC users on average, the OpenRefine Capstone team is hoping to streamline working with SNAC (About SNAC, n.d.).

5. Conclusions

In the end, we were able to successfully build a stable plugin that allows participating organizations and individuals to streamline bulk data insertion to SNAC. Compared to the website interface, this significantly decreases the amount of time spent trying to insert resources from hours to mere minutes. This will ultimately allow the institutions already using SNAC to easily pursue the goals of creating accessible information while enticing others to follow suit. Academic institutions and universities, more than ever, are seeking ways to efficiently access and organize data online. Our work contributes to an open source code base that will be used by many in the future. This shows the impact and importance of creating tools that seek to increase the efficiency of working and manipulating large amounts of data.

6. Future Work

A remedial action to take is to think of ways to make this plugin as user-friendly as possible for non-technical people. It is safe to assume that archivists, the ones that will be using this plugin, are non-technical in terms of figuring out how to use the tool initially. One way to take action to make the interface as user-friendly as possible is to provide tips for users. This can be done by having pop-ups that guide you through the tool the first time you use it that explains what the possible functionalities are and how to use them. That way, archivists will become well-versed with the tool and know its capabilities and functionalities when testing and pushing their own data.

Directions for future investigations can go towards optimizing the performance of the plugin through a couple ways. One way could be to possibly implement a machine learning algorithm that could potentially be trained for column name detection so there would be no need for drop downs. Users can then validate the column names after they have been assigned headers if needed. Another way could be to have the ability to upload multiple resources through the API calls to help reduce the network latency or possibly the processing time needed to upload each resource individually. Cloud is something viable for this plugin. This would enable more possibilities such as automatic updates so users would always have the most updated version and not have to download a new version every time. It would also be less maintenance for the users and less resources to manage on their local machines.

7. References

About SNAC. (n.d.). Retrieved from https://portal.snaccooperative.org/about

Aggarwal, H. (2018, January 28). Setting up Environment Variables in MacOS Sierra. Retrieved April 25, 2020, from https://medium.com/@himanshuagarwal1395/setting-up-environment-variables-in-macos-sierra-f5978369b255

Criticism of Wikipedia. (2020, March 20). Retrieved from https://en.wikipedia.org/wiki/Criticism_of_Wikipedia#Accuracy_of_information

Ham, K. (2013). Free, Open-source Tool for Cleaning and Transforming Data. *Journal of the*

Hill, K. M. (2016). In Search of Useful Collection Metadata: Using OpenRefine to Create Accurate, Complete, and Clean Title-Level Collection Information. *Serials Review*, 42(3), 222-228. Retrieved from https://www.sciencedirect.com/journal/serials-review

How to set JAVA_HOME environment variable on Mac OS X 10.9? (2014). Retrieved April 25, 2020, from https://stackoverflow.com/questions/22842743/how-to-set-java-home-environment-variable-on-mac-os-x-10-9

Jeong, M (2019, October 28). Personal Interview with J Glass.

Medical Library Association. Retrieved from https://www.ncbi.nlm.nih.gov/

OpenRefine. (2010). *Introduction to OpenRefine*. Retrieved from http://openrefine.org/

OpenRefine. (2019). *Installation Instructions*. Retrieved April 25, 2020, from https://github.com/OpenRefine/OpenRefine/wiki/Installation-Instructions

OpenRefine. (2019). *OpenRefine Installing Java*. Retrieved April 25, 2020, from https://github.com/OpenRefine/OpenRefine/wiki/Setup-JAVA

Park, J-R. (2008). Metadata Quality in Repositories: a Survey of the Current State of the Art. *Cataloging & Classification Quarterly*, 47(3), 213-228. doi:10.1080/01639370902737240

The 3-Clause BSD License. (n.d.). Retrieved April 25, 2020, from https://opensource.org/licenses/BSD-3-Clause