Lucas Kitaev

Mr. Combs

Senior Projects

17 February 2018

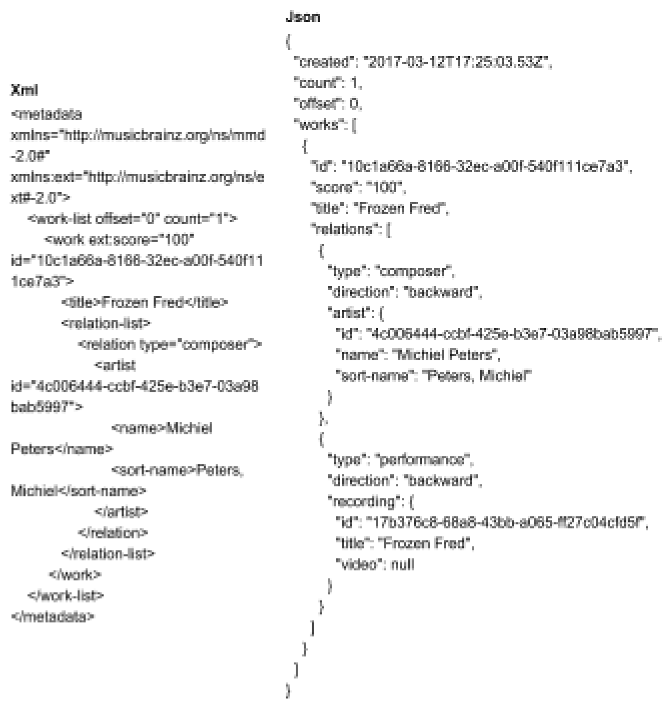
Journal 4

Goals

- Research how to use the OMDb and MusicBrainz API's
- Implement search functionality using Java beans
- Implement scriptlet functionality to output search results and switch between query types
- Create servlets to generate group pages, implement movie servlet
- Implement scriptlet functionality to link to the appropriate servlet from search results
- Add documentation to the source code, and package the current version for release
- Implement music servlets

Research

My research for this phase of the project was mostly focused on learning how to implement the OMDb and MusicBrainz API's (application programming interfaces) to gather data for search results and result (group) pages. Both use API's their respective website's "web service" to query a database containing all information relevant to the particular search type. OMDb is used for both television shows and films, and sends responses in JSON (JavaScript Object Notation, a very simple format to represent data, see below for an example). MusicBrainz (MB) is primarily used for artists, albums (and EP's, singles, etc.), and songs, but has information in many more areas. It does offer JSON responses like OMDb, but the Java API for MB uses XML (Extensible Markup Language, similar to HTML) responses. To illustrate the differences between the two, here's a comparison:



Xml

```
<metadata
xmlns="http://musicbrainz.org/ns/mmd
-2.0#"
xmlns:ext="http://musicbrainz.org/ns/e
xt#-2.0">
   <work-list offset="0" count="1">
      <work ext:score="100"
id="10c1a66a-8166-32ec-a00f-540f11
1ce7a3">
         <title>Frozen Fred</title>
         <relation-list>
            <relation type="composer">
               <artist
id="4c006444-ccbf-425e-b3e7-03a98
bab5997">
                  <name>Michiel
Peters</name>
                  <sort-name>Peters,
Michiel</sort-name>
               </artist>
            </relation>
         </relation-list>
      </work>
   </work-list>
</metadata>
```

Json

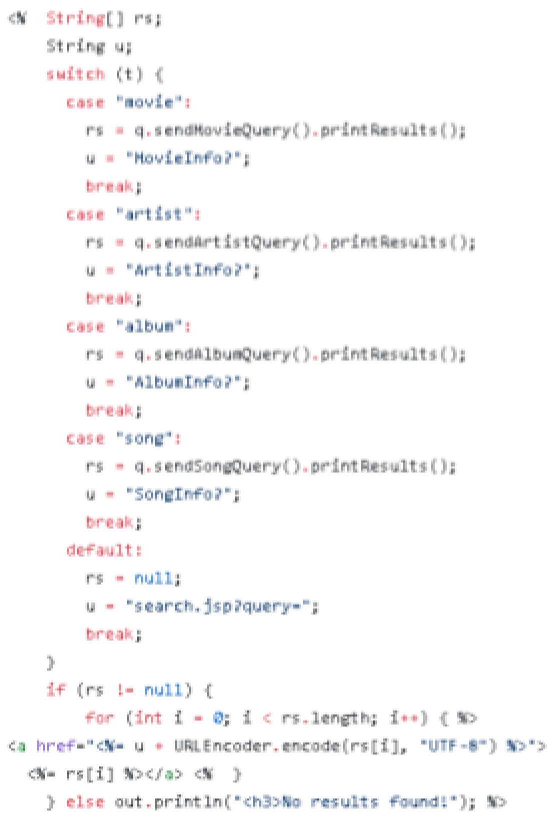
```
{
 "created": "2017-03-12T17:25:03.53Z",
 "count": 1,
 "offset": 0,
 "works": [
  {
   "id": "10c1a66a-8166-32ec-a00f-540f111ce7a3",
   "score": "100",
   "title": "Frozen Fred",
   "relations": [
    {
     "type": "composer",
     "direction": "backward",
     "artist": {
      "id": "4c006444-ccbf-425e-b3e7-03a98bab5997",
      "name": "Michiel Peters",
      "sort-name": "Peters, Michiel"
     }
    },
    {
     "type": "performance",
     "direction": "backward",
     "recording": {
      "id": "17b376c8-68a8-43bb-a065-ff27c04cfd5f",
      "title": "Frozen Fred",
      "video": null
     }
    }
   ]
  }
 ]
}
```

JSON and XML both get the job done: XML being more concise, and JSON being more readable. For a server processing requests, and especially at low volumes, it could hardly matter which one is used. The API methods by which these responses can be accessed are fairly straightforward. The general process I've implemented is to create a "client" that is used to interface with the web service. The client is then fed the query supplied by the user on the search page, a search is executed using the client, and the results are assigned to their own variable. The results can then be printed to the search page by grabbing the associated title of each result. Gathering data for group pages works in a similar fashion, but instead a query is executed on only a single result, after which, properties like year and description can be printed onto a webpage using a servlet.

The other piece of research I've done for this journal is on JavaServer Page "scriptlets." Scriptlets are Java code embedded directly into the JSP markup, and there are three different types, each with a slightly different notation. A full scriptlet uses `<% %>` tags, and can contain all the same code as what one would find in a regular Java class, but with the ability to reference implicit objects that are inherent to the web page itself, like `PrintWriter` and `HttpSession`, as well as bean objects that have already been instantiated. Scriptlet declarations are defined by `<%! %>` and are used to define variables that can be used elsewhere in the JSP. Scriptlet expressions are used to evaluate variables and output the result to the webpage, and are denoted by `<%= %>` tags. Scriptlet comments can be put in `<%-- --%>` tags, differing from regular HTML comments (`<!-- -->`) by not being able to be viewed in a web browser (“JSP Syntax”). The main advantage of scriptlets is to perform quick functions that don’t warrant being put in a separate Java class. One important thing to note is that the default JSP compiler doesn’t use the latest Java version, so in order to take advantage of the same features that are available to regular servlets, it’s necessary to configure the server to use the proper Java version.

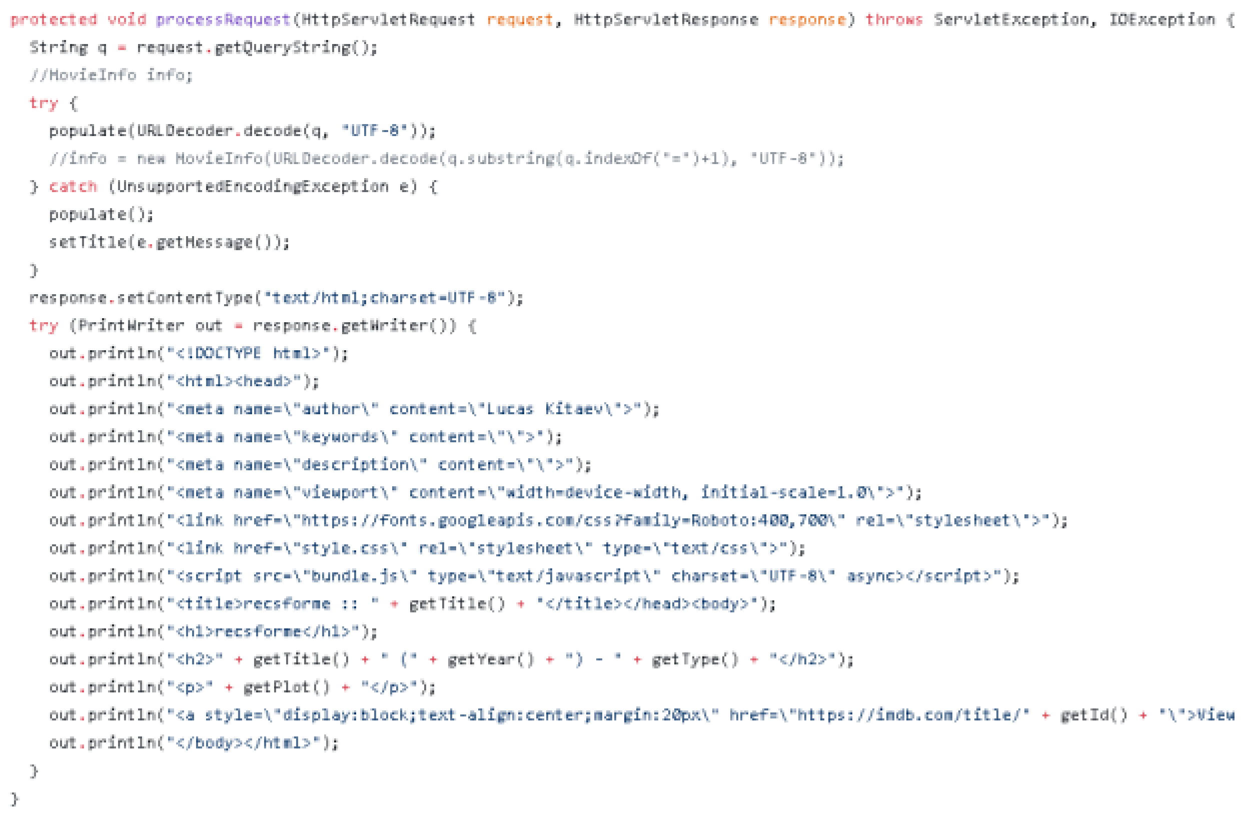
Accomplishments

- Working search for all types
  - This encompases a number of classes, viewable here
  - Searches are only designed to return the most relevant results, and are limited to 25 entries
- Scriptlet on search page to dynamically generate the output



```
<%  String[] rs;
    String u;
    switch (t) {
      case "movie":
        rs = q.sendMovieQuery().printResults();
        u = "MovieInfo?";
        break;
      case "artist":
        rs = q.sendArtistQuery().printResults();
        u = "ArtistInfo?";
        break;
      case "album":
        rs = q.sendAlbumQuery().printResults();
        u = "AlbumInfo?";
        break;
      case "song":
        rs = q.sendSongQuery().printResults();
        u = "SongInfo?";
        break;
      default:
        rs = null;
        u = "search.jsp?query=";
        break;
    }
    if (rs != null) {
        for (int i = 0; i < rs.length; i++) { %>
<a href="<%= u + URLEncoder.encode(rs[i], "UTF-8") %>">
  <%= rs[i] %></a> <%  }
    } else out.println("<h3>No results found!"); %>
```

- Working movie servlet



```
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  String q = request.getQueryString();
  //MovieInfo info;
  try {
    populate(URLDecoder.decode(q, "UTF-8"));
    //info = new MovieInfo(URLDecoder.decode(q.substring(q.indexOf("=")+1), "UTF-8"));
  } catch (UnsupportedEncodingException e) {
    populate();
    setTitle(e.getMessage());
  }
  response.setContentType("text/html;charset=UTF-8");
  try (PrintWriter out = response.getWriter()) {
    out.println("<!DOCTYPE html>");
    out.println("<html><head>");
    out.println("<meta name=\"author\" content=\"Lucas Kitaev\">");
    out.println("<meta name=\"keywords\" content=\"\">");
    out.println("<meta name=\"description\" content=\"\">");
    out.println("<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">");
    out.println("<link href=\"https://fonts.googleapis.com/css?family=Roboto:400,700\" rel=\"stylesheet\">");
    out.println("<link href=\"style.css\" rel=\"stylesheet\" type=\"text/css\">");
    out.println("<script src=\"bundle.js\" type=\"text/javascript\" charset=\"UTF-8\" async></script>");
    out.println("<title>recsforme :: " + getTitle() + "</title></head><body>");
    out.println("<h1>recsforme</h1>");
    out.println("<h2>" + getTitle() + " (" + getYear() + ") - " + getType() + "</h2>");
    out.println("<p>" + getPlot() + "</p>");
    out.println("<a style=\"display:block;text-align:center;margin:20px\" href=\"https://imdb.com/title/" + getId() + "\">View
    out.println("</body></html>");
  }
}
```

- First release on GitHub, with continuous integration (automated testing) and code documentation

Reflection

I'm still behind schedule (supposed to be working on the actual recommendation engine now), but once I overcome the current challenge of getting the music servlets to function, I should be able to finish. There's a strange bug with the MB lookup function not including the properties I want (namely, the list of albums an artist has). It worked once, but it hasn't since, so I'll be looking into the issue. On the bright side, I've managed to get my GitHub repository looking rather professional (if I do say so myself); I'll consider starting to promote it soon. Currently, the first beta release is planned for the sixth of March, however, before that time, I should write some code tests, to ensure that everything is working properly. Doing so will require research on using the JUnit library, which I’m not very very familiar with yet.

Works Cited

"JSP Syntax." *Tutorials Point*, 8 Jan. 2018, www.tutorialspoint.com/jsp/jsp_syntax.htm.