Lucas Kitaev

Mr. Combs

Senior Projects

11 March 2018

Journal 5

Goals

- Finish implementing servlets
- Research and integrate Apache Ivy for Java dependencies
- Research existing recommendation algorithms and their implementations
- Create package and abstract class for recommendation engine
- Sketch diagram of application and algorithm logic
- Implement AJAX to send to and receive data from the server
- Link user like/dislike lists to group pages

Research

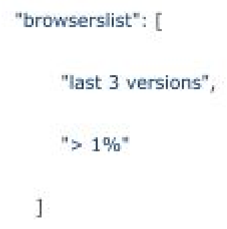
The first thing I researched was a Java development tool by the Apache Foundation, called Ivy, which is a sister project to Apache Ant, a build tool (executes tasks such as compiling source code, running the project, and cleaning up unneeded files) I had already been using. Ivy adds on to the functionality of Ant by allowing for dependency management similar to npm, which was covered in a previous journal. Any external resources a project is dependent on can be declared in an Ivy file, and then when Ivy is run, it will download anything not already installed locally to the project's library folder ("Ivy Documentation"). Another development tool I researched was Browserslist, a JavaScript program (installed with npm), that is used to automatically handle browser compatibility. Browsers that a web application wants to support can be declared in the Browserslist configuration, which then passes on those requirements to other programs such as Babel (for JS compatibility) and Autoprefixer (for CSS compatibility), and allows them to make more efficient decisions on which compatibility options to use and which ones to not (Ai). Both tools make the development process much simpler.

For the next part of my research, I took a brief look into recommendation algorithms. For this project, I'm not trying to do anything overly complicated, since the focus is more on just learning how to make a web application itself. The simplest solution to building a recommendation engine appears to consist of basic operations on sets (which have been covered before), which are performed among groups of users (Ridwan). My current implementation uses the local storage of browsers to keep track of likes and dislikes; hopefully I'll be able to implement it as an external database soon. The main method upon which recommendations will be generated is by taking a number of users with high similarity coefficients, calculated based on the number of shared titles in like/dislike lists, and then intersecting (picking out unique items in) the grouped sets to determine a new set of possible recommendations for each user.

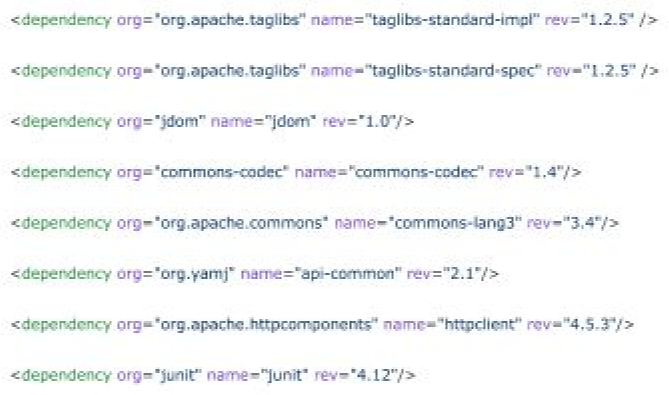
The last section of my research focused on extending my knowledge of JavaScript. This included the jQuery library, which I mentioned in another journal and got slightly acquainted with, as well as the Knockout library. For the jQuery part, I wanted to learn how to use its AJAX (Asynchronous Javascript And XML) functions. The purpose of AJAX is to send and receive data from servers without the need to refresh the browser page. Traditionally AJAX was used with XML data (covered previously), but AJAX can be used to send any type of data, most commonly encoded URL's with query parameters, multipart form data submissions possibly including file uploads, and JSON (also covered previously), in addition to XML. AJAX is greatly simplified in jQuery, with GET (retrieve data from server) and POST (send data to server) requests able to be executed only with the `$.get()`[1] and `$.post()` respectively ("Category: Ajax"). My application uses AJAX to send vote (like/dislike) data to a Java servlet located on the server. To supplement this, I also learned how to use Knockout, which uses a MVVM (Model, View, View-Model) pattern to handle web user interfaces. The model is consisted of stored application data either externally or internally, the view of HTML `data-bind` declarations that link to the view model, a JavaScript object that specifies how bound elements are to behave. All three components are interconnected, so when the model updates, so does the view, and vice-versa ("Knockout Introduction"). Knockout is used in my application to update users on the status of their vote on a particular group. If they haven't voted, the vote form will be present, and once the form has been submitted, it will disappear in place of a confirmation dialog, which will remain present as long as the vote is not reset.

Accomplishments

- Ivy dependencies
- Configured Browserslist



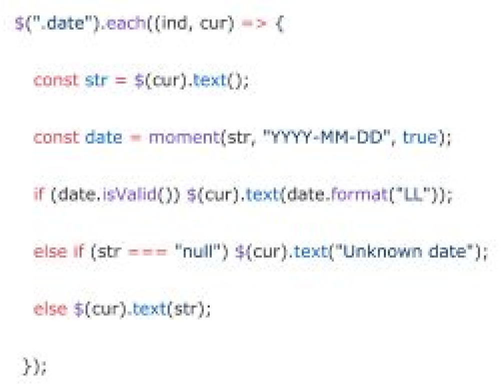
```
"browserslist": [
    "last 3 versions",
    "> 1%"
  ]
```



```
<dependency org="org.apache.taglibs" name="taglibs-standard-impl" rev="1.2.5" />
<dependency org="org.apache.taglibs" name="taglibs-standard-spec" rev="1.2.5" />
<dependency org="jdom" name="jdom" rev="1.0"/>
<dependency org="commons-codec" name="commons-codec" rev="1.4"/>
<dependency org="org.apache.commons" name="commons-lang3" rev="3.4"/>
<dependency org="org.yamj" name="api-common" rev="2.1"/>
<dependency org="org.apache.httpcomponents" name="httpclient" rev="4.5.3"/>
<dependency org="junit" name="junit" rev="4.12"/>
```

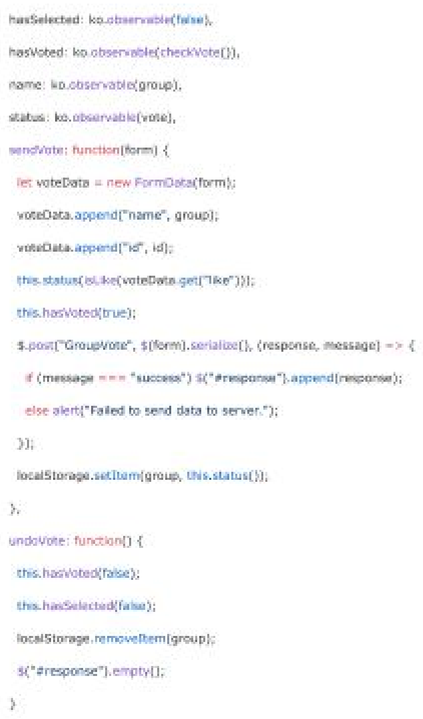
[1] The dollar sign ($) is used to refer to a jQuery instance.

- Redone JavaScript code using jQuery



```
$(".date").each((ind, cur) => {
  const str = $(cur).text();
  const date = moment(str, "YYYY-MM-DD", true);
  if (date.isValid()) $(cur).text(date.format("LL"));
  else if (str === "null") $(cur).text("Unknown date");
  else $(cur).text(str);
});
```

- Vote form using jQuery and Knockout



```
hasSelected: ko.observable(false),
hasVoted: ko.observable(checkVote()),
name: ko.observable(group),
status: ko.observable(vote),
sendVote: function(form) {
 let voteData = new FormData(form);
 voteData.append("name", group);
 voteData.append("id", id);
 this.status(isLike(voteData.get("like")));
 this.hasVoted(true);
 $.post("GroupVote", $(form).serialize(), (response, message) => {
  if (message === "success") $("#response").append(response);
  else alert("Failed to send data to server.");
 });
 localStorage.setItem(group, this.status());
},
undoVote: function() {
 this.hasVoted(false);
 this.hasSelected(false);
 localStorage.removeItem(group);
 $("#response").empty();
}
```

Reflection

Still somewhat behind on the schedule, considering I haven't got much done for the recommendation engine itself. I have, however, resolved the database issue by switching from a GlassFish Server to an Apache Tomcat Server. Tomcat also provides advantages in being more lightweight than GlassFish; it implements the Web Profile of the Java EE standard, which is a subset that focuses on only the core components of what Java EE offers, which is great, because those are the only features that I need. For my next journal, I'll need to research OAuth 2.0 in order to use it for authenticating users, and more research on code testing, especially for JavaScript. Since I'm running low on time, it might be a good idea to do some research on the Bootstrap CSS framework as well, which will allow me to focus on developing the core application, and not have to worry about the user interface as much, since that can be handled for me automatically.

Works Cited

Ai. “Ai/Browserslist.” *GitHub*, Evil Martians, 8 Mar. 2018, github.com/ai/browserslist.

“Category: Ajax.” *JQuery API Documentation*, The JQuery Foundation, api.jquery.com/category/ajax/.

“Ivy Documentation.” *The Apache Ant Project*, The Apache Software Foundation, ant.apache.org/ivy/history/latest-milestone/index.html.

“Knockout Introduction.” *KnockoutJS*, knockoutjs.com/documentation/introduction.html.

Ridwan, Mahmud. “Predicting Likes: Inside A Simple Recommendation Engine's Algorithms.” *Toptal Engineering Blog*, www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine.