Jacob J. Zhang

May 14th, 2017

Software Design Patterns

Spring Semester

Final Project: JabRef Clone

Objective

For the Software Design Patterns final project, I created a bibliography management tool that allows users to add, edit, upload, and sort and manage references for bibliographies. The idea was based on JabRef, a self-proclaimed “graphical Java application for managing bibtex (.bib) databases.” However, whereas JabRef runs on the operating system, due to the shift in recent years from desktop applications to web-based ones, we were encouraged to build a web application with a rich featureset.

The final product has all the capabilities you would expect as a user: being able to upload .bib files to import them to the database, easy adding and editing of the loaded references, ordering of any column in ascending or descending order, searching for a specific reference, deletion of one or many references at once, and easy export as a .bib file.

Framework Technology

To add interesting features to the program, I utilized a wide array of technologies, including several libraries written by forward thinking contributors in Java and Javascript. For the backend of the application, I used Spring Boot-- a web framework for stand-alone projects-- and Maven to compile, build, and run it on a server. For data storage, the built-in H2 database was used for persistence (only between sessions), and Spring Data REST provided easy generation of a resource definition and REST API. Spring Data REST was one of the key parts of the architecture, enabling us to expose a discoverable REST API with collection, item, and association resources in a JSON schema that communicated very simply with our frontend.

Perhaps the most innovative aspect of our JabRef clone is the use of React, a Facebook-backed frontend framework. There has been a gradual shift in the past decade to move more of the business logic of a web application from the backend to the

frontend. In the mid-to-late 2000s, the standard web framework was Ruby on Rails, who claimed to "optimize for programmer happiness with convention over configuration". Rails and other MVC frameworks like CakePHP and Django would receive a user action event, run through logic on its own servers to determine what the rendered page would look like, and serve final rendered HTML to the user.

Newer frameworks like React allow programmers to build single page applications, which are defined as applications where "all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions." The page is loaded once, and anything in the Document Object Model that needs to change will do so without a reload (usually via an XHR call). The end result is that because everything needed is on the initial load, responses to user actions feel significantly faster. This approach also takes the load off the server, having to not render multiple page and only concerning itself with the web service. Critics of this approach will point out the futility of SEO, being that an entire application could technically only have one link, and the potentially long initial load of all the resources.

Implementation

In the case of our JabRef clone, the vast majority of functionality was written in an app.js file. In the vast majority of JS implementations, best practice is to only include one React component per file, but Webpack/bundle integration proved difficult with Spring on a Tomcat instance.

React has great event handling built into the framework, so I was able to trigger method calls with onChange and onClick user actions in a clean and easy manner. Each component was responsible for its own state. As an example, each Reference component would have state on what reference it was holding, whether or not it was checked, and how to handle its own deletion. On the other hand, the ReferenceTable kept a list of all references it had initially loading, the current list of references (due to search filtering or deletion), and several attributes for reordering. The idea of components managing their own states and event handling helped decouple the application for easier maintenance down the line.

I wrote the search and modal services from scratch as they were both easy extensions from the React framework. Search simply loops through the references currently in state within ReferenceTable, and checks to see if any of the fields match what the user typed

into the input field. The modal service was done in CSS, and were triggered on user clicks.

I used several external libraries for uploading and parsing BibTex files. For file upload, I used a storage service found via the Spring Guides that already contained the service, error handling, and file properties set up. For the parser, after doing some research, I was very impressed with JBibTex, a bibTeX and LaTeX parser and formatter library, as it handled many differents versions of .bib files. It was also available through a Maven plugin, something the other alternatives did not provide. Exportation proved to be more of a challenge, and in perhaps a relatively unclean way, I did it through string concatenation and triggered the download of the file through a JS-generated file.

Finally, for thumbnail images, I used the Google Books API due to its scalability and breadth of resources for all books. On every initial load, AJAX calls to the API passes it information about the reference.

Summary

The coupling of React and Spring with their associated technologies made the development process surprisingly smooth. The frameworks also provided endless posibilities in terms of features due to how well both handled data.

References:

https://projects.spring.io/spring-boot/

http://projects.spring.io/spring-data-rest/

http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php

http://rubyonrails.org/

https://facebook.github.io/react/

https://github.com/spring-guides/gs-uploading-files

https://github.com/jbibtex/jbibtex