ARTICLE

Pure Client-Side Data Wrangling

Building a Strongly Single Page Web Application in an Academic Library for Production of a Catalog-Ready Patron File

Jason Bengtson

ABSTRACT

While many languages are used for data manipulation, it is unusual to see JavaScript put to this task. This paper describes a novel application built to manipulate catalog patron data using only JavaScript running in a browser. Further, it describes the approach of building and deploying "strongly single page web applications," a more extreme version of single page applications that have been condensed into a single HTML file. The paper discusses the application itself, how it is used, and the way that possessing web development and coding skills in an organization's systems department can help it flexibly respond to challenges using such novel solutions.

INTRODUCTION

For many years, libraries have absorbed increasingly technical tasks, supporting sophisticated platforms necessary to provide their patrons with superior information resources. In academic libraries, this has meant supporting platforms as diverse as catalogs, institutional repositories, digital collections platforms, acquisitions platforms, and platforms that combine these functions. Libraries are more than the sum of their parts—or the sum of their platforms, holdings, and access models—providing services that add value and enhance those various offerings. Academic libraries offer reference support, classes, academic research within the information science field, and proprietary technology enhancements. Of course, adding any technology-based enhancements to both patron-facing and internal services requires that libraries invest effectively in the technology subfield of librarianship, often by employing librarian-technologists or staff technologists with sufficient skill to write sophisticated code.

One way to add technology enhancements to web-based vended platforms is to employ a technique sometimes called "sideways web development." This methodology allows organizations to create applications that can be integrated into the "view" layer of vended web applications (the layer of the application composed of HTML, CSS, and JavaScript that is rendered in the browser) to add additional functionality or correct existing functionality for patrons.[1] However, in other cases proprietary applications built by the library that do not directly integrate with vended applications can still be employed to enhance the functionality of those applications. This case study describes one such application built by and for the New Mexico State University (NMSU) Library.

To operate correctly, library platforms require data of various kinds. Some of those platforms are limited in the type and format of data they can or will accept and lack any native provision for data

About the Authors

Jason Bengtson (bengtson@nmsu.edu) is Associate Professor and Head of Systems, New Mexico State University Library. © 2024.

Submitted: 19 February 2024. Accepted for publication: 18 September 2024. Published: 16 December 2024.

acquisition or conversion, requiring libraries to do the job of wrangling the data so that it is available and usable by those platforms. Such data are often heterogenous and vary wildly in quality and completeness. As illustrated in this case study, some campus systems do a poor job of applying accurate file extensions or outputting data in the correct format.

In a best-case scenario this data aggregation, shaping, and delivery is a function carried out for academic libraries by the central IT department at their university. Normally a request is submitted, and the central IT department builds a process under their aegis that outputs a data file conforming to the Data Standards Document for a particular application. These files can then be harvested and ingested into the target platform via an automated process. Such a scenario has been the majority of this author's experience. However, in some cases the central IT departments lack the facilities, expertise, or staffing to perform this function. In other cases, the campus culture has not been one where the central IT department has worked with the library in such a fashion, or one side has not engaged the other for cultural or perceived practical reasons. In such cases it falls to library technology departments to temporarily (or sometimes permanently) provide data wrangling services for their own platforms.

On the campus of NMSU, the central campus IT department (ICT) was highly resource constrained for many years. When the NMSU Library changed catalogs and adopted the Alma Library Service Platform from Ex Libris, they did so on a tight schedule that would have made a data integration request to ICT difficult, if not impossible, to carry out. Building a modality to synthesize campus staff and student data into files that could be used by Alma was also not a service that Ex Libris offered. As such, the NMSU Library Systems Department built its own process for collecting staff and student files and converting them into a patron file that Alma could ingest and use.

As previously noted, an arguably superior solution is to have campus central IT departments generate a data file on a regular basis (preferably daily) that conforms to the Data Standards Documents for Alma patron files. NMSU's ICT department now offers such services, but the backlog of requests is significant. While a request for centralized processing of the patron file is in progress at NMSU, in 2023 the NMSU Library Systems Department evaluated its legacy process and replaced it with a strongly single page web application, a term which will be discussed later in this paper. This case study will describe the original process and the application it was replaced with.

LITERATURE SEARCH

The literature contains very little of relevance in addressing the problem of wrangling data in the client (using browser rendered languages) by libraries.[2] The author initially searched the Library, Information Science & Technology Abstracts (LISTA) database with the query: ((data N3 science) OR (data N3 wrangling) OR (data N3 manipulation)) AND javascript. The author limited the query to peer-reviewed academic journal and received six results. Upon examination, none of them were relevant to the topic of using client-side web technologies to conduct data wrangling and manipulation.

The author followed up their initial search by searching the NMSU Primo discovery layer with the query: "data wrangling" AND javascript. The author received a disappointing five hits, with the only potentially relevant item being the book *Data Wrangling with JavaScript* by Ashley Davis.[3] Unfortunately, NMSU did not have a copy of the book and perusal of the table of contents from Amazon.com indicated the book would be focused on the use of Node.js, which is to say JavaScript on the server rather than in the client.[4]

It seems that client-side data wrangling and manipulation in libraries is a topic with very limited treatment in the literature and one that is ripe for additional work.

The author then conducted a multi-vector phrase search of the LISTA database for "single page web applications" OR "single page web apps" OR "single page applications" OR "single page apps" limited to academic journals. This returned one result, which the author determined was not relevant to the subject of this paper (the paper in the search results described exposing website data to web crawlers).[5]

Based on these searches the author determined that this case study dealt with novel enough material that a further review of the literature was not likely to be useful in the context of this paper.

While not the focus of this paper, it is worth providing some context for the general process described here, which falls generally under the paradigm of "extract-transform-load" (ETL). This is a process many systems librarians will be familiar with as a methodology to extract data and transform that data (both by cleaning it and by re-serializing it) before delivering it to a data warehouse. In the case of this application, the data warehouse is the library service platform Alma. A search of the LISTA database and the NMSU Primo interface provided several sources suitable to ground the reader in this process.

In his book *Integrations of Data Warehousing, Data Mining and Database Technologies: Innovative Approaches*, Panos Vassiliadis includes a chapter that serves as a survey of ETL technology.[6] In this chapter Vassiliadis describes the problems that must be overcome in this process, including differing data schemata, data quality issues, and the need to create these workflows in ways that provide for refreshing the data frequently. The survey goes on to cover topics such as the use of the Uniform Modeling Language (UML) for conceptual ETL and data warehouse modeling, practical problems and solutions for cleaning data, challenges in data extraction, issues with pivoting in tabular data, and a variety of additional developments in this area.

In their paper for the *Journal of Computing and Information Technology*, Rahman, Marz, and Akhter demonstrated that by using an ETL metadata model (mapped out via UML) they were able to standardize their data warehouse refresh operations.[7] They used stored procedures specific to the databases they interacted with, all coordinated with a metadata model to achieve standardization in their ETL pipeline and efficient load behaviors.

Tom Adamich provided an overview of the ETL process as a means to transform MARC data into Linked Data.[8] They note that in the library context ETL is often a process used for various types of MARC record. In the article they ask several questions about the use of ETL processes in libraries and how they might scale into the future.

In this case study MARC records are not the data source, and the specific solution illustrated remains a partially manual process due to current data source limitations. However, the utility of the general ETL process and the search for greater efficiencies within it provide some common ground between the work described here and efforts described in these ETL articles.

METHODS

The First Data Wrangling Process

The initial process was an ingenious one assembled on the fly by the senior systems analyst at the time of the catalog migration: Christopher Landt. It used an approach based on a variety of technologies. These technologies were applied via two separate processes for the staff and student files. These files are used to create and update patron records in Alma.

The process used to provide data to Alma for the NMSU campus had two inputs: one report containing staff that was deposited on a regular basis into an SFTP directory controlled by the library and one student report that was extracted from the IBM Cognos system used by the campus. The staff report was delivered as a Tab Separated Values (TSV) file (despite arriving with the .txt extension) and the Cognos report could be outputted in multiple ways, including as an Excel file or what Cognos called a Comma Separated Values (CSV) file (this report was actually also in the TSV format). For the first process this staff file was extracted as an Excel file.

The student and staff files went through two distinct processes, producing two files that had to be loaded into Alma in a particular order (student then staff) to avoid incorrect data overwrites for purposes of checkout permissions. The end product for both was an XML file in the format required by Ex Libris for Alma ingest.[9] The XML file then had to be compressed for upload.

The staff process was the simpler of the two. The staff file was converted to an Excel spreadsheet and a macro devised by the senior systems analyst was run against the report, making various formatting and data changes. This macro needed to be updated each semester to account for date changes that populated in the final XML data. The report was then exported to the CSV format and a Ruby script that the senior systems analyst had devised converted the CSV into the correct XML format.[10] At that point, the file could be compressed and was ready for ingest by Alma via an import profile.

The student process was more complex, due in part to the less usable and duplicative data derived from the Cognos report. The data were extracted as an Excel file, which was then imported as a table to an Access database containing an existing table of “Crimson Scholars” who were to be granted special checkout privileges. A query devised by the senior systems analyst then added the “CRIM” user type to the Crimson Scholars in the students table and adjusted other data accordingly. The Crimson Scholars table had to be updated every semester due to changing student membership. The student table was then extracted as an Excel spreadsheet.

At that point, the new student spreadsheet needed to have two cleanup and data wrangling macros (both devised by the senior systems analyst) run against it in a specific succession. These macros had to be updated each semester. When the macros were finished with their work there were a series of manual steps that had to be run against the spreadsheet to remove the numerous duplicate entries. Then the file was exported to CSV and a Ruby script was run against it to convert it into the appropriate XML format before it was compressed to a zip file for upload into Alma.

While clever and well documented, the process had a number of shortcomings, including the large number of steps, the requirement that a variety of only partially compatible technologies be used in concert with each other (including an outdated version of Ruby that needed to be installed on the machines of those technologists using the process), and the need to update macro commands and the Access database every semester. Any missed step would result in a file that would not be ingested properly into Alma, and the entire process was extremely time consuming.

The New Data Wrangling Process

Systems Department leadership changed in 2023, and the new department head began making a variety of transformations within the department. One of the processes they assessed as potentially benefitting from an update was the Alma patron load process.

The new process performs all data wrangling using a strongly single page web application, employing JavaScript on the client-side. A single page web application is a software application built with web code that does not require more than one webpage be used for its operation, generally meaning the content of the webpage that serves as its user interface employs highly dynamic changes to its Document Object Model to do work.[11]

The author uses the term "strongly single page web application" to differentiate between applications such as the one described in this case study and the broader category of single page web applications. A strongly single page web application is built specifically to not only employ only one webpage to do its work, but also consists of only a single HTML file. It has no server-side components, and all JavaScript, CSS, and images are internal to the single HTML file, requiring no external components.

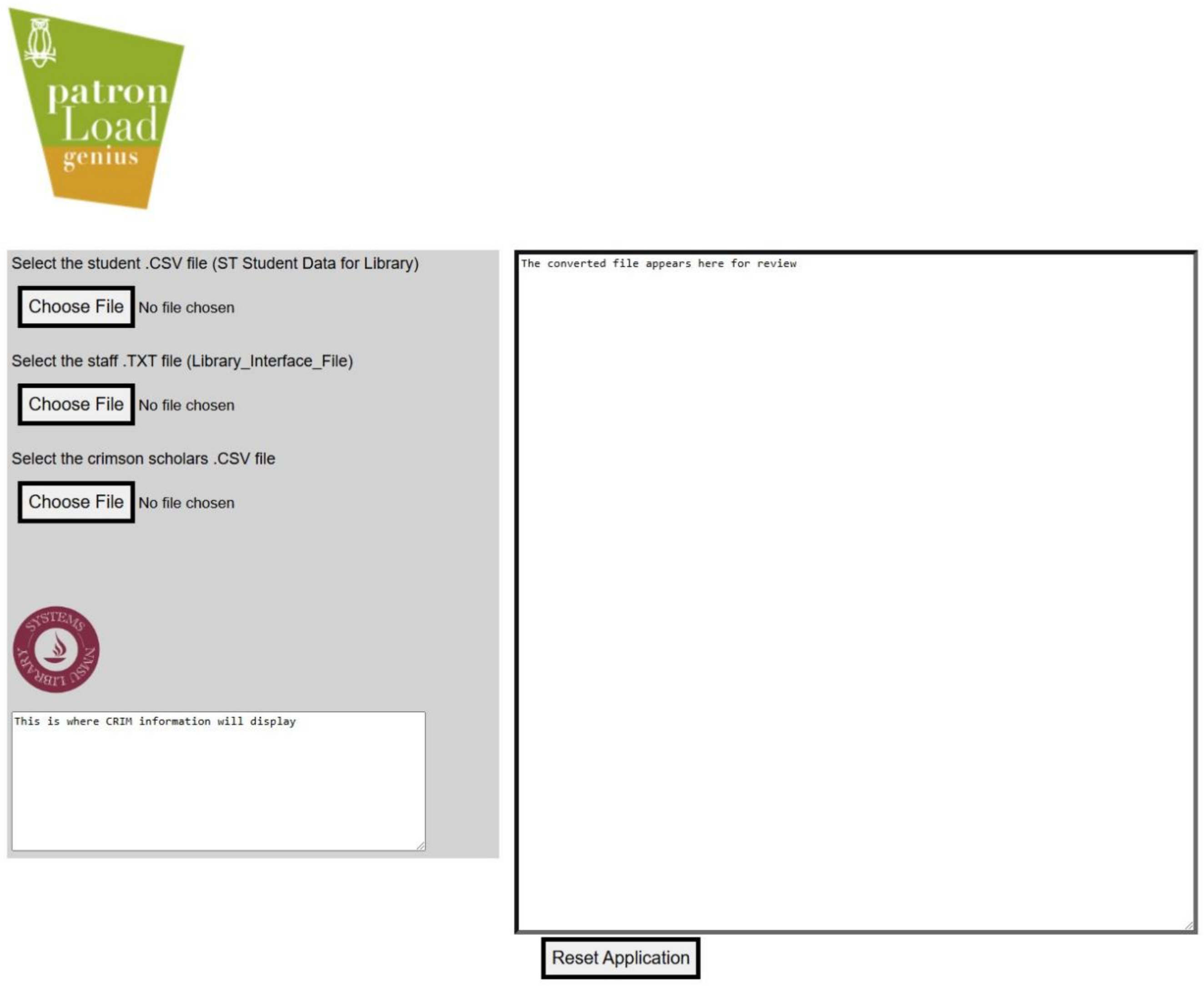
This format confers several advantages for situations like the one described in this case study. A single HTML file that opens in a browser, without other dependencies that must follow it within a directory, is compact, highly portable, and easy for users who are not web developers or coders to understand and use. If it is well constructed and formatted it is still relatively easy to navigate, update, and maintain. It uses only client-side technologies that don't require the broader knowledge base needed to work on full stack web applications with a server-side component and it does not require an internet connection to work. All CSS and JavaScript can be situated with `<script>` and `<style>` tags while images can be encoded as base64 (in the form of a data URL) and added to the source attributes of image tags.[12] Base64 images make the file more challenging to navigate than it might otherwise be and are larger than their bitstream counterparts (by approximately one third), but the trade-off is lower loading times over some network configurations and thriftiness with file portability.[13]

There may be some confusion over the nature of a web application and whether a strongly single page web application still qualifies as one in those cases when it is not run from a web server. Especially given that applications exist (such as OpenRefine) that are not web applications but which express client-side web code so that they can have an interface that runs in a browser. A strongly single page web application is composed solely of web coding languages and must be run entirely within the run-time environment of a web browser to operate. The building blocks of such an application are web technologies (and only web technologies), whether run over a network or not. Perhaps most telling, such an application *can* be placed on a web server and loaded over a network via the HTTP protocol—it simply does not have to be. An apt comparison is that of a pumping truck parked near a construction site and used to pump water. The truck may be there for weeks, months, or even years. The owners of the truck might decide to never move it at all once it is in place. However, this does not make it a fixed pumping station, despite its capability in that capacity. Hypothetically the owners could, at any time, choose to move the truck. In the same way, a strongly single page web application, which could run in good order from a web server, may be tasked to run without one while remaining a web application. Moreover, there are other reasons to use strongly single page web applications, such as leveraging less than optimal networks with an application that employs only one file download instead of many, or to serve web content through firewalls that may block or provide slow throughput for bitstream files, while letting base64 files pass unhindered. However, those use cases are not the subject of this paper.

The new application requires three inputs: the staff and student files and the original Crimson Scholars Excel file the Systems Department received each semester. The Crimson Scholars file is exported to the CSV format prior to use. The student file is extracted from Cognos as an ostensible CSV file, although the application handles it as a TSV file. The application employs the File API to allow users to easily load the files from their local computer into the application.[14] It runs a series of functions and data wrangling operation to produce an XML file properly formed and formatted in a way that Alma could use.

Particular attention was paid to the user interface to make it intuitive. The button to begin the data processing does not appear until all files are uploaded, for instance. Similarly, the buttons for working with the resultant XML do not appear until processing is complete. The finished file is fully scrollable and reviewable in a `textarea` element on the webpage, making it easy to spot check data and formatting without having to download the file and open it in an editor, or copy and paste it into another application.

Figure 1. The user interface of the data wrangling application


patron Load genius
Select the student .CSV file (ST Student Data for Library)
Choose File No file chosen
Select the staff .TXT file (Library_Interface_File)
Choose File No file chosen
Select the crimson scholars .CSV file
Choose File No file chosen
SYSTEMS NMSU LIBRARY
This is where CRIM information will display
The converted file appears here for review
Reset Application

Data Wrangling in the Client

As I note in my book on library web development, one of the most effective strategies for wrangling data and reserializing it into a new format is to first convert the data into programmatic objects in memory.[15] Programmatic objects are quite easy to work with, and if you are fortunate in that your data are modest enough that they can fit into the portion of your computer's memory available to the web browser (as was the case with these patron data), you can effectively work on all of your data at once. If not, you will likely need to process your data on the server-side where you can employ strategies such as chunking and work with your dataset one piece at a time from the source file. While the File API is an excellent way to add application elements for selecting and loading files into memory, it lacks some of the flexibility of programmatic file access possible through server-side language libraries and modules. In addition, the browser by necessity constrains the amount of access the client-side application has to system resources, and these restrictions cannot be flexibly modified as is generally possible with languages running on the server. This places practical limits on the amount of data that can be reasonably processed via client-side scripting techniques via a strongly single page web application.

The application discussed here converts the various TSV and CSV files into arrays of JavaScript objects to effectively perform operations on the patron data.[16] Once the data are successfully transformed into arrays of programmatic objects (one array for each file), the data are run through a series of functions that remove duplicates and provide basic cleaning for the data. A simple, initial set conversion is adequate to remove exact duplicates, but a more sophisticated use of JS filters in a second function is necessary to remove the objects that are exact duplicate records from the student file but not exact object matches (entries resulting in changes to student status or additional programs that for some reason are not cleaned up in Cognos). These functions also perform basic data cleaning (removing white space and reserved characters, for instance) and reserialize the properties in the objects of both arrays to reformat them with the tag names required by Alma. In many cases the values must also be reformatted so that they match the allowable values in the Alma patron file template.

Expiry and purge dates that determine checkout periods are algorithmically derived by testing the current system Date object to find the relevant semester (Spring, Fall, or Summer).[17] Once this is determined, the user type is tested against conditionals that determine the final dates to be added to the patron file. This method is vital to the new process as it removes the need to update code by semester as must be done in the original process for the three Excel macros and the Access database query.

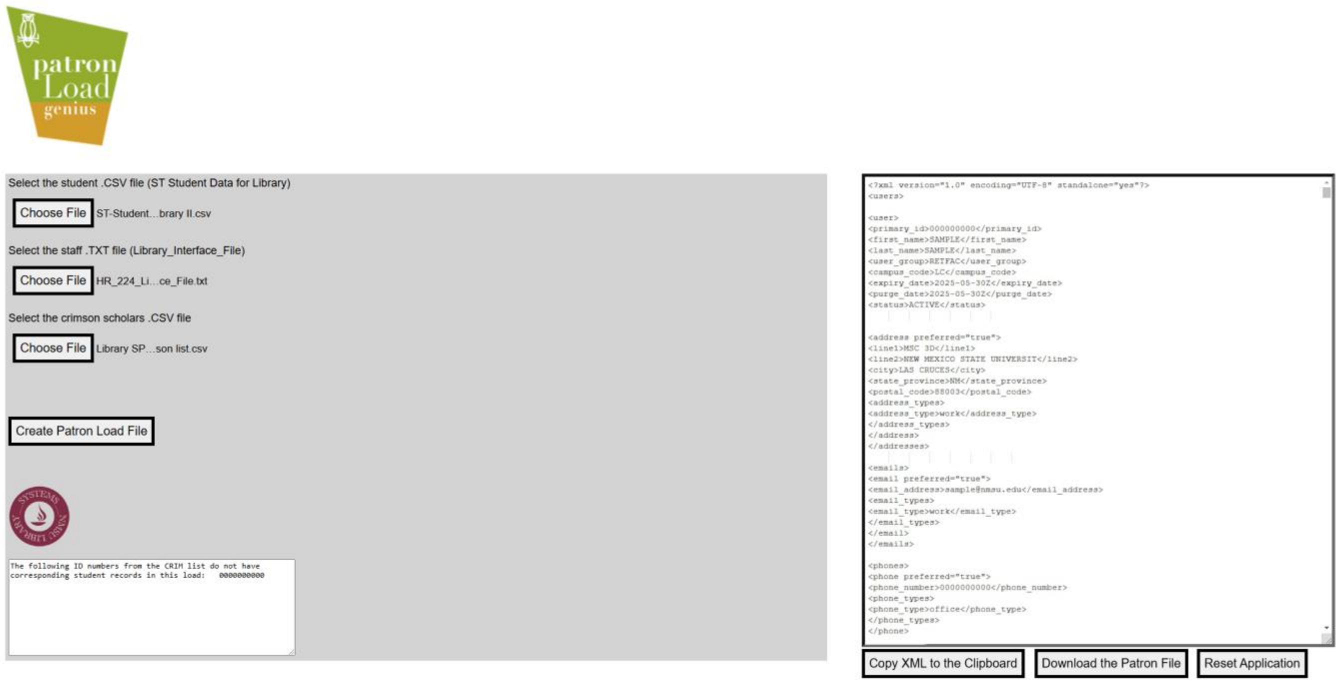
The Crimson Scholar array is then iterated over; student objects with matching IDs have their patron type set to "CRIM" and their purge and expiry dates are modified to the more generous dates provided to Crimson Scholars. This operation utilizes the Crimson Scholar file and performs the same basic function as the operation with the Access database in the original process.

The original patron load process produced a student and a staff file, which had to be loaded into Alma student first so that any overlapping entries would be overwritten by the staff file. The reason for this is that staff permissions are more generous than student permissions and several patrons were affiliated with the university in both roles. The new application produces a unified file by mapping the staff and student arrays into new associative arrays in which the campus unique ID number serves as the index. The new student array is then iterated over, with staff records pushed into the array, thus overwriting any existing objects in the student array with a matching index.

The final array is now syntactically correct with data as accurate as can be culled from the campus reports. At this point, the application then writes the array of objects into an XML string that can be downloaded as a file, copied to the computer's clipboard, or examined in a textbox in the application's user interface. The ability to examine the file contents in a text box makes it easy to spot check the file for any formatting issues or other potential areas of concern. The XML file can then be compressed, added to the upload SFTP site, and ingested into Alma to update existing patron records. The Alma ingest operation usually takes several minutes, depending on the size of the patron file.

The application's web user interface also generates and displays a list of user IDs from the Crimson Scholar file that have no corresponding IDs in the student file (meaning that those user IDs are not present in the final patron load file). The existence of this disparity is a consequence of changes to enrollment during the semester and the outputted list provides the user with a convenient reference in case a problem should arise in that area of the data.

Figure 2. The application interface when the operation is complete



Select the student .CSV file (ST Student Data for Library)

Choose File ST-Student...brary II.csv

Select the staff .TXT file (Library_Interface_File)

Choose File HR_224_Li...ce_File.txt

Select the crimson scholars .CSV file

Choose File Library SP...son list.csv

Create Patron Load File

```
The following ID numbers from the CRIM list do not have
corresponding student records in this load:   0000000000
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<users>

<user>
<primary_id>000000000</primary_id>
<first_name>SAMPLE</first_name>
<last_name>SAMPLE</last_name>
<user_group>RETFAC</user_group>
<campus_code>LC</campus_code>
<expiry_date>2025-05-30Z</expiry_date>
<purge_date>2025-05-30Z</purge_date>
<status>ACTIVE</status>


<address preferred="true">
<line1>MSC 3D</line1>
<line2>NEW MEXICO STATE UNIVERSIT</line2>
<city>LAS CRUCES</city>
<state_province>NM</state_province>
<postal_code>88003</postal_code>
<address_types>
<address_type>work</address_type>
</address_types>
</address>
</addresses>

<emails>
<email preferred="true">
<email_address>sample@nmsu.edu</email_address>
<email_types>
<email_type>work</email_type>
</email_types>
</email>
</emails>

<phones>
<phone preferred="true">
<phone_number>0000000000</phone_number>
<phone_types>
<phone_type>office</phone_type>
</phone_types>
</phone>
```

RESULTS

The original data wrangling phase of the process took this author, on average, between twenty-five and forty-five minutes, depending on if issues arose. The data wrangling with the new application takes between two and five minutes. The new application does not require any updates to the underlying structure due to changes in semester since the date modification is done automatically through comparisons with the system Date object (thus data and code are kept separate), reducing upkeep compared to the former process. The original method required three separate and lengthy text files to document the process for wrangling data and the process for making semester by semester changes. The new application is fully documented on a single confluence wiki page. This same documentation page holds the HTML file for the application and the Crimson Scholar CSV file for each semester. Such centralization would not have been possible with the old process, which required a large number of very different files, including Ruby files that would not work without the local installation of an older version of the Ruby interpreter.

The new application is run once a weekday during the first five to seven weeks of the semester, depending on the rate of change in the data observed by the Systems Department. It is run once a week later in the semester as the changes slow down.

Consequently, the new application has proven more efficient, more straightforward, more portable, and more homogenous in approach while requiring far less knowledge on the part of the person performing the operation. It is also far less vulnerable to external changes (such as changes in Excel, Access, or Ruby), given that a web browser serves as its run time environment, and the scripting is entirely in vanilla JavaScript. As a strongly single page web application it is extremely portable, easy to understand, and easy to use. The author's colleagues in the NMSU Library Systems Department have expressed satisfaction and even relief with the new process. The bar for maintenance is reduced, as the application is written in languages needed for the rest of the library's web presence and that are more commonly part of a web developer's toolkit.

DISCUSSION

This case study elucidates two points in particular: (1) the potential for JavaScript to be used in applications that require data manipulation and (2) the utility to be derived from libraries maintaining the local capability to perform sophisticated coding development.

As noted in the literature search section, there is currently a dearth of information on data wrangling with JavaScript in the client. This is unfortunate, given that in cases where an application is being built to do such work that requires a user interface, it is often more convenient and much more portable to build a web application that functions entirely in the client (assuming modest quantities of data). While JavaScript may not be a language optimized for data work, its data objects are similar to those of other C-based languages, making it adequate for data cleaning, wrangling, manipulation, and re-serialization. As the engine for dynamism in the client it is well appointed for data visualization, especially when using libraries like D3. In addition to this application, the author built a single page web application for a previous employer that converted another type of CSV data into client-side web code (filterable data lists) that could be dropped into the web content management system editor to add fully functional content to the appropriate webpage. In both cases data transformation using JavaScript was effective.

However, use cases must be treated judiciously and client-side JavaScript would not be the best choice for very large datasets, or for use cases in which the intention was to fully automate a data process. For this case study the application is producing combined data sets with between 25,000 and 30,000 users. While large, the amount of data held in memory when it runs is still far lower than would be the case for a similar combined dataset from one of the many largest institutions of higher education and is still sufficient that it requires a workstation of at least middling power and 16GB of memory. In this case, the application is serving as a bridge between a more challenging user-run process and a fully automated process using a derived file generated by the campus central IT department. In many cases data operations can be fully automated and run on a server using a server-side language and scripts regularly automated by the Scheduler (also known as the Cron).[18] In this instance, since one of the files must be extracted manually (and regenerated for each day's run), such a more complete automation level is not an option.

JavaScript's objects also employ a prototypal inheritance model that can, at times, be challenging to work with for coders primarily experienced with server-side coding.[19] Depending on the task and the familiarity of the developer with JavaScript, this can make building operations that require duplicating and modifying objects a frustrating process, perhaps offsetting the advantages of object-based data manipulation for some developers. Clearly in this, as with any major project, the costs and benefits must be weighed against each other before deciding if a strongly single page web application is the most useful approach for a particular situation.

There are other means of working with data that do not require coding, although they are subject to notable limitations. The author briefly experimented with Easy Data Transform (https://www.easydatatransform.com/) when it was brought to his attention and found it an excellent tool for working with tabular data. They were unable to find a way, however, to use it to construct an XML file from the data that conformed to the particulars required by Alma for ingest. While data files could be exported as XML, they found no mechanism allowing them to map the data to a specific structure, something easily done for more complex data formats like JSON or XML using a suitable programming language. Similarly, while Easy Data Transform possessed a number of powerful ways to remove duplicates from a tabular input file, the author was unable to find one that could handle the specific issue of duplicates in the student data in which up to eight duplicate lines with different student types per user in an unpredictable order required conditionals beyond what Easy Data Transform offered to collapse into a single user. Additionally, the author did not see a way to generate and insert items like Expiry date according to the library requirements. Due to its license requirement the author was unable at the time to experiment with FME (https://fme.safe.com/) but feels confident that some of the same challenges would be in evidence.

The ability of NMSU Library to produce both versions of the patron load process is a direct consequence of the fact that it retained the coding skills in its Systems Department necessary for such work. The author has experienced situations at several libraries where the possession of this capability has been a key element in being able to effectively meet challenges and respond to unique opportunities. This has included data wrangling (catalog migrations, custom data transformations, rapid utilization of the HathiTrust emergency access program during the COVID-19 pandemic), interface improvement (custom improvements to discovery layers, custom improvements to the view layer of an institutional repository), or the creation of original, sophisticated, proprietary web applications (building database-driven applications for the library). Some libraries have recognized the utility of this skill capacity, while others have been less enthusiastic about embracing technology in this way, preferring instead to direct resources toward what may be perceived as more traditional (and to the thinking of some, more core) library functions. As this example shows, however, possessing such development capacity can be game-changing for challenging situations like the one NMSU Library found itself in when it migrated catalogs.

While services that involve physical resources are likely to remain a part of many library portfolios for a very long time, improving access to digital resources and using technological building blocks to improve current or imagine new services has become a vital facet of modern library presence. This is, however, sometimes a matter of contention. On one memorable occasion the author was confronted by a senior and more traditionally focused librarian during a planning session for a new emerging technology space with the forceful declaration that they would refuse to support anything that did not support the mission of the library. The other librarian did not specify what they considered the mission of the library to be, however they seemed to feel that it was not aligned with technology-centered efforts. The author has sometimes encountered similar, if less animated, cultural difficulties at other libraries.

As the environment continues to change for libraries, a deep technological bench will become more essential for those libraries that seek to capitalize on that change and even lead the way in some areas of the information landscape. Moreover, information science provides unique insights to the work of computer science and information technology, addressing areas such as user needs and search in ways that can be both different and complementary to those sibling disciplines.[20] The technology subfield of librarianship offers significant opportunities to drive improvements in the profession and research in the discipline of information science. It serves, in many ways, as a surface area in which information science can connect with many other disciplines in a meaningful way.

A further interesting area for exploration is the utility of the strongly single page web application approach. While not suitable for many applications, it may provide a highly portable, future scalable, and easily maintainable (given that it is built with exclusively client-side technologies requiring more common developer skills than approaches with more obscure or broader portfolios of languages) format for solving problems. This type of application can be sent to users as a single file, rather than a folder full of confusing dependencies. In the modern context such an application has an analog with Java-based applications, in that the web browser functions as a run time environment in the same way that a Java run time environment allows Java to run on machines with different processor architectures. A key difference, however, is the natural ubiquity of web browsers on computers compared to the sometimes poorly updated or entirely absent Java run time environment. This makes strongly single page web applications even more portable than Java applications in most circumstances.

One area in which such applications experience limitations is in the use of JavaScript frameworks and libraries. The client-side of applications are increasingly being authored entirely inside of frameworks such as Angular.js or React. However, many such frameworks require a server-side component, making them infeasible for an application of this type. Even frameworks or libraries that are entirely client-side add a significant amount of overhead to the application file and must be handled judiciously. In this application, for instance, the author chose not to employ even the commonly used jQuery library to reduce overall file overhead.

Frameworks also can increase the complexity of an application, making even a relatively simple app difficult to improve or maintain. Fortunately, “vanilla” JavaScript proved sufficient to efficiently code all the dynamic portions of the application. In future strongly single page web applications, however, JavaScript libraries such as jQuery, JQuery UI, D3.js, etc. will likely be employed and the overhead they add behind `<script>` tags in the file will simply be a trade-off intrinsic to this approach. Despite these trade-offs, however, the author considers the strongly single page web application to be a valuable part of their toolkit.

CONCLUSIONS

The results of this case study indicate that when confronting certain types of data wrangling tasks, a client-side solution may be a viable option. Additionally, for certain situations the extreme strongly single page web application may have benefits that make it a superior choice. Further, the process of development and application seems to indicate that academic libraries possessing systems departments with strong development and coding abilities have significant advantages in flexibility when confronting challenges involving digital resources.

ENDNOTES

[1] Jason Bengtson, *Library Web Development: Beyond Tips and Tricks* (Chicago: ALA Editions, 2019).

[2] MDN, "Client-Server Overview—Learn Web Development," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview.

[3] Ashley Davis, *Data Wrangling with JavaScript* (New York: Manning, 2018).

[4] Amazon.com, "Data Wrangling with JavaScript: Davis, Ashley: 9781617294846," https://www.amazon.com/Data-Wrangling-JavaScript-Ashley-Davis/dp/1617294845; Davis, *Data Wrangling with JavaScript*.

[5] Robert Fox, "Information Economy," *Digital Library Perspectives* 34, no. 2 (2018): 78, https://doi.org/10.1108/DLP-10-2017-0040.

[6] Panos Vassiliadis, "A Survey of Extract-Transform-Load Technology," *International Journal of Data Warehousing and Mining* 5, no. 3 (2009): 171, https://doi.org/10.4018/jdwm.2009070101.

[7] Nayem Rahmen, Jessica Marz, and Shameem Akhter, "An ETL Metadata Model for Data Warehousing," *Journal of Computing & Information Technology*, 20, 2 (2012): 95, https://doi.org/10.2498/cit.1002046.

[8] Tom Adamich, "ETL: Extract-Transform-Load," *Technicalities* 40, no. 1 (January/February 2020): 16.

[9] MDN, "XML Introduction—XML," accessed December 13, 2023, https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction.

[10] MDN, "Ruby—MDN Web Docs Glossary," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Glossary/Ruby.

[11] MDN, "SPA (Single-Page Application)—MDN Web Docs Glossary," accessed January 5, 2023. https://developer.mozilla.org/en-US/docs/Glossary/SPA.

[12] MDN, "<script>: The Script Element," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script; MDN, "<style>: The Style Information Element," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Web/HTML/Element/style; MDN, "Data URLs," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Web/URI/Schemes/data.

[13] MDN, "Base64—MDN Web Docs Glossary," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Glossary/Base64.

[14] MDN, "File API—Web APIs," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Web/API/File_API.

[15] Bengtson, *Library Web Development*.

[16] MDN, "Array—JavaScript," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array.

[17] MDN, "Date—JavaScript," accessed December 13, 2023, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date.

[18] MDN, "Scheduler—Web APIs," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Web/API/Scheduler.

[19] MDN, "Inheritance and the Prototype Chain—JavaScript," accessed January 5, 2023, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain.

[20] Bengtson, *Library Web Development*.