Automating Data Collection and Location Tagging in Pansophy

Nicolas Crespo, Tabb High School and The Governor's School for Science and Technology
Mentors: Valerie Bookwalter (Computer Scientist) & Mike Dickey (Sr. Production Support Technician)

Introduction

The superconducting radio frequency (SRF) cavities that line the linear particle accelerator at Jefferson Lab allow for near zero energy loss and enable particles to travel at close to the speed of light. However, the highly specialized nature of SRF technology means Jefferson Lab must design, construct and test SRF cavities in house. Though this allows the Lab to produce cavities for accelerator facilities around the world, it also presents an exceedingly difficult challenge in inventory and data management. Pansophy, an internal data management system is an all-encompassing solution to this problem.

Pansophy is an internal website that is only accessible inside of Jefferson Lab's firewall. Its front end is written in ColdFusion, JavaScript and CSS, with ColdFusion being used mainly due to its ability to seamlessly embed SQL (Structured Query Language) onto a webpage. ColdFusion is an HTML-like language with its own plethora of custom tags and a built-in scripting language, CFScript.

SQL is used to query the two main relation Oracle databases: PRIMeS (Production and Research Inventory Management System) and Travelers. Each database houses a collection of tables each with a primary key and various foreign keys. By cross referencing the correct tables, relevant information about a part or traveler can be discerned even when provided limited information.

Objective

As parts make their way around the SRF Test Lab, they go undergo various actions at different locations, or work centers. At each step, an engineer is tasked with creating a Traveler for the part they are dealing with. To streamline the creation of travelers into the Pansophy system and minimize human errors, it is in Jefferson Lab's best interest to automate the entry of part information as much as possible. As such, stickers containing unique bar codes (Figure 2) have been placed on most parts. These barcodes, which produce Transaction IDs, can prove various important characteristics of a part. Combined with location information stored in cookies that is pre-set on every engineer's machine, a proper implementation of this system should instantly redirect an engineer to a Traveler page (Figure 7) with just one scan of a barcode (Figure 1).

Figure 1: A barcode scanner to be used in the lab

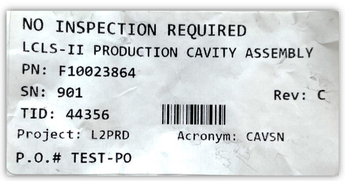

Figure 2: A sticker containing the barcode to be scanned by an engineer

Methodology

Before the scanning of a barcode and thus introduction of a transaction ID, some preliminary information is required. By allowing an admin user to set session-persistent cookies on every computer on the floor of the Test Lab a single time (Figure 3), this system can retain the work center, action, and location ID of a user's computer. The work center variable contains the location of the user's machine, the action variable defines the procedure the user is doing at that location, and the location ID is used to more precisely identify the current location. Rather than redundantly entering this information for every new traveler, these cookies can be set once per computer.

Since the barcode scanner (Figure 1) inserts the transaction ID as if it was typed manually, a single text entry box (Figure 4) is all that is needed for acquiring the Transaction ID. Once entered, the program creates a traveler ID in the following format:

`{PROJECT}-{WORK CENTER}-{PART NAME}-{ACTION}-{REVISION NUMBER}`

Using this Traveler ID, a corresponding URL can be built that will redirect the user to the appropriate traveler page.

Lastly, proper error checking must be implemented to account for bad user input, unexpected query results and undefined cookies. For example, if multiple travelers fit the criteria prescribed by the user, the system should present the user with a list of possible Traveler ID's and allow the user to search for and select one (Figure 6).

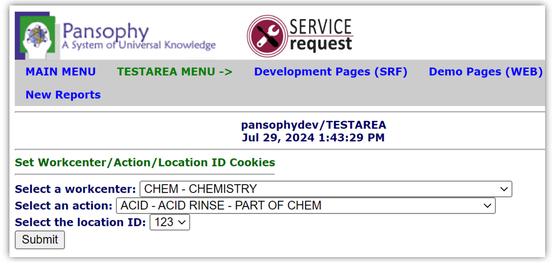

Figure 3: The admin restricted page used for setting computer specific cookies.

Figure 4: The Transaction ID entry box. Redirects used to traveler on submit.

Results

On the page to set cookies (Figure 3), two queries to the `TRAV_ACRONYMS` table in the Travelers database return details about all possible work centers and actions in a dash-separated format. By specifying the fields `OBS`, for obsolete, `CLOSED`, and `RETIRED` as `null`, any work centers or actions that are no longer in use are not returned. A third query to the `INV_LOCATIONS` table in the `PRIMeS` database (Figure 5) collects all valid location IDs. A `NULL` option was also hard coded into the Location ID drop-down due to certain travelers omitting that field. These three queries populate three drop downs with their results. On submit, a separate script sets cookies to their selected values by invoking a method in the custom `CookieBuilder` component.

On the transaction ID entry page (Figure 4), a text entry passes user input to the `getTravID` function from the `TravIDBuilder` component. This function calls the `getTravData` function, which includes a query that returns important data about the transaction ID. The transaction ID is introduced into this query using a `<cfqueryparam>` tag. This tag specifies what type the input value should be so that a malicious actor cannot inject SQL code into the query and potentially compromise the database. The query results are then returned and used alongside preset cookie values in a series of string concatenations to build a valid traveler ID.

Next, the `shouldUpdateLogs` function is used to query the `INV_TRANSACTIONS` and `INV_INVENTORY` tables (Figure 5) to determine if a part is still in inventory. The `INV_TRANSACTIONS` table is a history of all travelers, while the `INV_INVENTORY` table is a list of all parts in inventory. If the part is in the inventory and is not already in the current user's location, a new record is inserted in the `INV_TRANSACTIONS` table and a corresponding record in the `INV_INVENTORY` table is updated with the new location ID.

It is possible for a traveler to have multiple revisions added to it as certain actions are performed on it multiple times. A call to the `getRevisionNumber` method returns the total number of revisions a traveler has received and, if valid, this number is appended to the traveler ID. Finally, all information is concatenated to a URL the user is redirected to. In the case of the `getRevisionNumber` function returning an empty string, meaning the Traveler ID is invalid, the `getTravIDExtended` function is called. By attempting to match the traveler ID with a filtered set of options with the `LIKE` keyword in SQL, this function returns a list of possible traveler IDs that the user can search through and select (Figure 6). As the user types in the provided text input box, the results are filtered with JavaScript to contain only those that match the user's input. When a valid option is selected, the `submitTravID` script is executed, which redirects the user to the correct traveler page (Figure 7) with the now adequate amount of information.

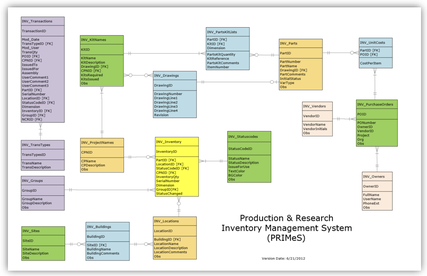

Figure 5: An entity relational diagram describing the PRIMeS database.

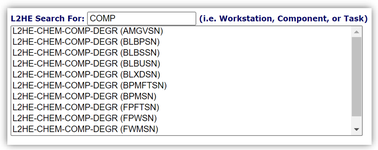

Figure 6: A search and select page that filters valid traveler IDs. Redirects user on click.

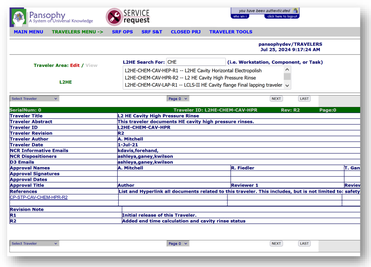

Figure 7: The traveler page a user would reach after scanning a part.

Challenges

The disconnect between the PRIMeS and Travelers database, which is bridged by the construction of a Traveler ID, made it difficult to test the code. Often times a query had to be reverse-engineered to ensure its validity, which took a considerable amount of time. Working in the development side of Pansophy's databases also proved problematic as all the dev data was years old and adhered to old standards. The development database had to be updated to allow for proper testing. Lastly, discrepancies in the format for traveler IDs required edge cases and special functionality to be implemented to account for exceptions to the standard format.

Future Work

This work is currently being implemented in the production Pansophy site where engineers can use it. Real world testing in the Test Lab is needed to gain feedback from real users and potentially change functionality to better fit their needs. As new projects, parts and processes come about at Jefferson Lab, this system may also need to be extended or altered to accommodate for any future changes. The modularization of the current system with custom components and functions should facilitate this process.

Acknowledgments

I would like to acknowledge Mike Dickey and Valerie Bookwalter for their excellent mentorship and guidance throughout this project, as well as Megan McDonald, Matthew Menia and the rest of the Pansophy Team for their support. Additionally, I would like to acknowledge Carol McKisson and Jalyn Dio for making the program possible, and my fellow interns for making my experience even more enjoyable.

References

Bookwalter, V., Dickey, M., McEwen, A., & Salvador, M. (2017). PANSOPHY, a JLAB SRF engineering data management system, supporting data collection, retrieval and analysis utilized by LCLS-II. *OSTI OAI (U.S. Department of Energy Office of Scientific and Technical Information)*. https://doi.org/10.18429/jacow-srf2017-mopb043

CFML reference. (2024, May 30). Adobe. Retrieved July 30, 2024, from https://helpx.adobe.com/coldfusion/cfml-tags-functions-reference-user-guide.html

Contributors, M. O. J. T. a. B. (n.d.). *Learn CF in a week*. https://learncfinaweek.com/

Pansophy — Process and data management. (n.d.). Thomas Jefferson National Accelerator Facility. https://www.jlab.org/accelerator/srf/pansophy