Lab Domain Documentation

Lab Domain is the working name for a lab administration website. As of the writing of this document this project can be found on Github at <https://github.com/PhilipCollison-GDAIS/LabDb>. To have this project transferred to a new owner please contact [philip@umd.edu](mailto:philip@umd.edu). For any information on setting up Lab Domain with XAMPP please see the README on Github.

Schema:

The Lab Domain Schema holds information for racks, equipment, ports, patch panels, and connections.

Perhaps the most potentially confusing aspect of the schema is the affiliations table. Within the schema, every piece of equipment has an affiliation with an identical primary key. Their relationship is one-to-one. At the moment a piece of equipment can have two possible affiliations. This affiliation can be determined by looking at the affiliated char column in the affiliations table. If affiliated is ‘R’, then the piece of equipment is associated with a rack. In this case parent\_rack\_id and elevation should be set to the appropriate values. It is also possible for a piece of equipment to be affiliated with another piece of equipment. An example of this is a PCI card inside another piece of equipment. If one piece of equipment is affiliated with another, then the affiliated char will be set to ‘E’ and parent\_equipment\_id will be set.

In the future, there should also be an affiliation for a piece of equipment located in storage. This affiliation would be signified by a character ‘S’ as the affiliated char and its inclusion would necessitate that a column for storage be added to the affiliations table that stores storage location.

A patch panel is synonymous with an optical cassette. Physically, an optical cassette has two sides. On the front there are a sizeable number of ports. On the back-end there is a very small number of optical ports, typically just one or two.

A port can either be associated with a patch panel or with a piece of equipment. Each port also has an associated connector type. Besides just storing the name of each connector type (some example names are “RJ-45” and “Cat 5”), the connector\_types table also stores a character of either ‘E’ for Electrical or ‘O’ for Optical. The current name for this column is affiliated. This is not a good name and it should be changed to something more unique. There will eventually be more than just ‘E’ and ‘O’ as options for connector types.

A connection can currently be between any two ports. Eventually though, connections should only be between ports that have the same connector type affiliation where both ports exist on the same rack. If you wish to determine the location of a port, its rack and elevation, then there exists a stored procedure named elevation\_and\_rack\_id\_for\_port that does just that.

With Lab Domain there is also a table for projects. Both connections and pieces of equipment can be associated with projects. This is intended to give these items some measure of protection from accidental deletion or modification. Only lab administrators and people with project permissions should be able to affect these protected rows.

Besides those small items which have already been mentioned there are three which could or should be made to the schema. There should be a table for users with associated permissions. The rooms table should be changed into two different tables. A building/location table and a rooms table where each room is associated with some location. The MTP\_connects table should be split into two to match the format of the ports and connections tables. If this happens then there is perhaps no further purpose for the optical cassette table to store slot\_count or port\_count. A height column should be added to equipment to announce how many rack units it takes up vertically.

Website:

The Lab Domain website is currently hosted using XAMPP. For further setup instructions after installing XAMPP see the README on Github. You can then access the XAMPP control panel and phpMyAdmin on port 80. To access the LabDomain website visit port 8080. To change the port settings edit the appropriate lines in C:\xampp\apache\conf\extra\httpd-vhosts.conf. What should be added to the include\_path variable in C:\xampp\php\php.ini is not necessarily C:\Git\LabDb\Html. It will instead be C:\path\to\repo\LabDb\Html.

The website itself is written almost completely in PHP, although a small portion of the website is also written in JavaScript. Besides these two technologies, there are three additional libraries used.

The first of these is Bootstrap. Bootstrap is responsible for the styling of the navbar, the sidebar, and the jumbotron element holds the main page. Bootstrap is also used for styling the forms, tables, modals, and dropdowns. Bootstrap can be found at <http://getbootstrap.com>.

The second library that is used is jQuery, a common JavaScript library. jQuery is used to display all modals. It is also used to ask for conformation for conformation when deleting elements or manipulating or traversing the DOM.

The third item is DataTables, a plugin for jQuery that is used for most of the tables on the website. More information on DataTables can be found at <http://www.datatables.net>. To select a DataTables object for dynamic page editing use the “new $.fn.dataTable.Api( selector )” selector/constructor. This will return an API instance of the desired table. To modify the table in use the API methods instead of modifying the DOM. DataTables has its own Ajax implementation.

There are several places of importance within the website implementation in the Html folder.

The first of these is the inc folder. The inc folder holds most of the commonly repeated visual elements, the navbar, the sidebar, the footer and the header, as well as the code used to connect to the server and form persistent user sessions. Of special interest inside the inc folder are the prototypes, reports, and forms files. Each form and report adheres to the appropriate prototype, and more will be said about that soon. It should also be known that database.php in /inc contains a number of common functions that are used throughout the website. Most of these are functions for populating dropdown options but some are also data validation functions.

The next two places of interest are the forms file and the reports file. Each file in the reports folder defines an object that implements the report prototype. The variable $report is then assigned this object and at the end of the file /inc/reports.php is then included. A similar process occurs for forms. One form of interest is the connections form which is submitted by Ajax instead of as an Html form with method=”post”. Also with the equipment form there will need to be Ajax that adds and removes form fields based on the results of dropdown. If a new piece of equipment being created is supposed to be affiliated with another piece of equipment then the user should select that on a dropdown and the appropriate fields should appear. If the user then decided that the piece of equipment should instead be affiliated with a rack, then they would select another option on the dropdown. When they select the new option the fields that appear would change into the appropriate form fields for the given option.

Next of interest is /js/java.js, the file containing all of the common JavaScript throughout the website. Java.js is included in header.php and it is thus present on every page. There are four functions of interest in java.js. The first of these is the constructor for all DataTables. Every DataTable is a table with a class of “.data-table”. On $(document).ready() each of these DataTables is constructed with the options given in the constructor. Because “scrollY” is not false, when the DataTables are constructed each table is actually split into two tables. What this means is that *the constructor cannot safely be called again despite what the documentation says*. The reason that you would want to call the constructor again is that it returns the jQuery/DataTables API object which you could then safely modify with DataTable methods. To manipulate DataTables instead use “new $.fn.dataTable.Api( selector )”.

Most tables have buttons associated with them. For example every rack page has a table for equipment and a table for optical cassettes. At each table there are buttons for adding, editing, copying, and deleting. Not all tables have all buttons. There is the problem though that buttons and tables need to be linked. The edit button for example should always be disabled unless there is exactly one selected row in the table. When the edit button is clicked it also needs to be able to get the values of the rows in the selected table. To accomplish this linking, every table/button group is surrounded by a div with class “table-n-buttons”.

The next function in java.js is a JavaScript function called redirectUser() which takes two parameters the element it is being called on. And a url that the user is being redirected to. redirectUser() is intended to be a function called onclick for buttons within table-n-button divs. Those buttons with a class of “oneSelected” should have a url parameter formatted like “/forms/racks.php?copy\_id=”. The user would then be redirected to “/forms/racks.php?copy\_id=val”, where val is the value of the one selected row in the associated DataTable. Buttons that do not have a class of “oneSelected” are simly redirected to the supplied url.

The next function in java.js ensures that the buttons in a table-n-buttons are only enabled/disabled when the appropriate number of rows are selected in the associated table. When a data-table row is clicked it is given the class “selected”. When it is clicked again the class is removed. On each click those buttons with a class of “oneSelected” become enabled if there is exactly one row with the class selected in the associated table. Those buttons with a class of “notNoneSelected” become enabled if and only if there are rows selected in the appropriate table.

The last function in java.js has to deal with the delete buttons in table-n-button divs. This function has three parts and two prerequisites. The first prerequisite is the inclusion of the appropriate “/delete/something.php” file. The second is that each table-n-button in “/reports/something.php” must have a unique name. The first thing that this function does is that each button with a class of “delete-button” is given its own confirm\_delete div. Next, each of these divs becomes a jQuery modal dialog. Finally jQuery registers a trigger such that when any delete-button is clicked the appropriate modal, by name, is opened. When the user presses “Ok” on the modal, confirming that they would like to delete, a post request is then sent to the current page. There key of the request is the table’s unique name with “-delete” appended. The value of the request is an array storing the ids of the selected rows. On success of the operation the selected rows are visually removed from the table.

Also of importance are the modal buttons for racks and equipment that are used to add ports and equipment, and the fact that each row holds the id of the item that it represents.

The majority of the TODO’s left in the code have to do with unimplemented data validation. Be aware that when input is valid, true is returned. If there is some error in the input, then instead a string is returned that informs the user of the error that occurred.

Be aware that edit and copy functionality is, for the most part, unimplemented.