BIMSHIFT

HTML5 Report Creator

Colorado School of MInes
Field Session 2014
June 17, 2014

Tyler Lyons
Anthony Nguyen
Kolten Robison

Table of Contents

Client Background & Product Vision

BIMShift is a business based in Littleton, Colorado, founded by two Mines alumni, Mike Gioia and Justin Clark. The focus of the company is to provide efficient access to specialized data for members of construction projects. Through the usage of Building Information Modeling, *BIMShift* is able to provide clients with access to huge amounts of building metadata for people who may not be familiar with the underlying technologies in order to efficiently hasten construction projects while being within budget.

Their primary product, *DATAGate*, integrates building model information from AutoDesk® Revit® into a SQL database that is navigable through the use of their client-specialized website access. Because of its decentralized cloud design, *DATAGate* allows easy access to data from any web-enabled device. *DATAGate* allows the user to update and modify Revit® model information in the database with the *Manage_Elements* tool, view and export Revit® models with the *Model Viewer*, estimate building costs with the *Estimate Costs* tool, and even wirelessly access data from the construction site with the deployment of *TapTrack*™ enabled sites that leverage NFC (Near Field Communication) and QR Code (Quick Response Code) technologies.

Although *DATAGate* clients are able to easily access raw metadata, the interface lacks the ability to create detailed analytic reports or to visually display aggregate data. In order to add this functionality, our team was tasked with adding and integrating a functional report-generating system into the existing *BIMShift* development site.

We created a dynamic filtering system that allows a client to retrieve only specific elements whose parameters match certain values or ranges. The filtering capabilities also include a logical operator precedence system that allows reports to be based off of extremely intricate queries (chaining many AND and OR operators in SQL). This allows clients to quickly leverage powerful SQL calls to create tabular reports of building model elements and export them as Microsoft Excel compatible files. We also developed an effective grouping tool that allows users to create visual reports from their element data, such as a report that groups rooms by square foot area and their building departments.

Requirements

The overall task for our project during field session was to create a visual HTML5--compliant representation of the project data that *BIMShift* keeps for their clients. Because *BIMShift* is interacting in an existing and well-structured industry, a majority of the clients involved may have little technological experience. Because of this, we made our visual data representations friendly for even the most inexperienced users (e.g. large buttons, obvious control flow, all site information accessible within about two 'clicks', little visual clutter). Our project takes the form of a filterable tabular report generator as well as generated visuals to represent aggregated element data.

Functional Requirements

- Able to filter element data from the database.
    - There should be several ways to filter data by parameters.
        - Equals, NOT Equals, Contains, and Ranged.
            - Ranges must extract numbers from SQL String data.
            - Ranges must convert between feet inch format for lengths.
    - There must be a way to create multiple filters.
        - Filters can be in AND/OR groups to allow for building of logically operating queries.
    - Filters can be activated/deactivated.
    - Data can be grouped by particular fields.
        - Summable parameters (Area) must be calculated.
        - Number of elements per grouped parameter must be calculated.
- Able to create and view reports from filters.
    - Fields for column display may be selected and deleted.
    - Report can be saved and retrieved from the database.
    - Able to be exported to Microsoft Excel.
- Grouped reports must contain a graphical display.
    - Graphical display must leverage the Highcharts library.

Non--Functional

- Use HTML5-Compliant tags.
- Create a PHP, JS, and PHP_AJAX file for each section (View and Create Reports)
    - PHP
        - Manage page views and templates for initial loading.
        - Will pull any $_GET variables.
    - JS
        - Will control element bindings.
            - Keypress, click, change.
            - Used for interactivity without server requests.
        - Makes AJAX calls to update dynamic page portions.
        - Uses Highcharts to actually draw graphs.
    - PHP_AJAX
        - Handle server-side AJAX calls to replenish HTML.
        - Make SQL calls.
            - Filtering, displaying reports, pulling data for graphs, etc.

System Architecture

The project is designed to interact between SQL databases and HTML5-compliant GUIs. **Figure 1** below shows the general system architecture and relations between the two. Interactions between grey and blue indicate SQL calls to retrieve data, interactions between blue and red indicate pulled SQL data to create interactive GUIs, and interactions between red and green indicate the yielding of final report data through the use of GUIs.

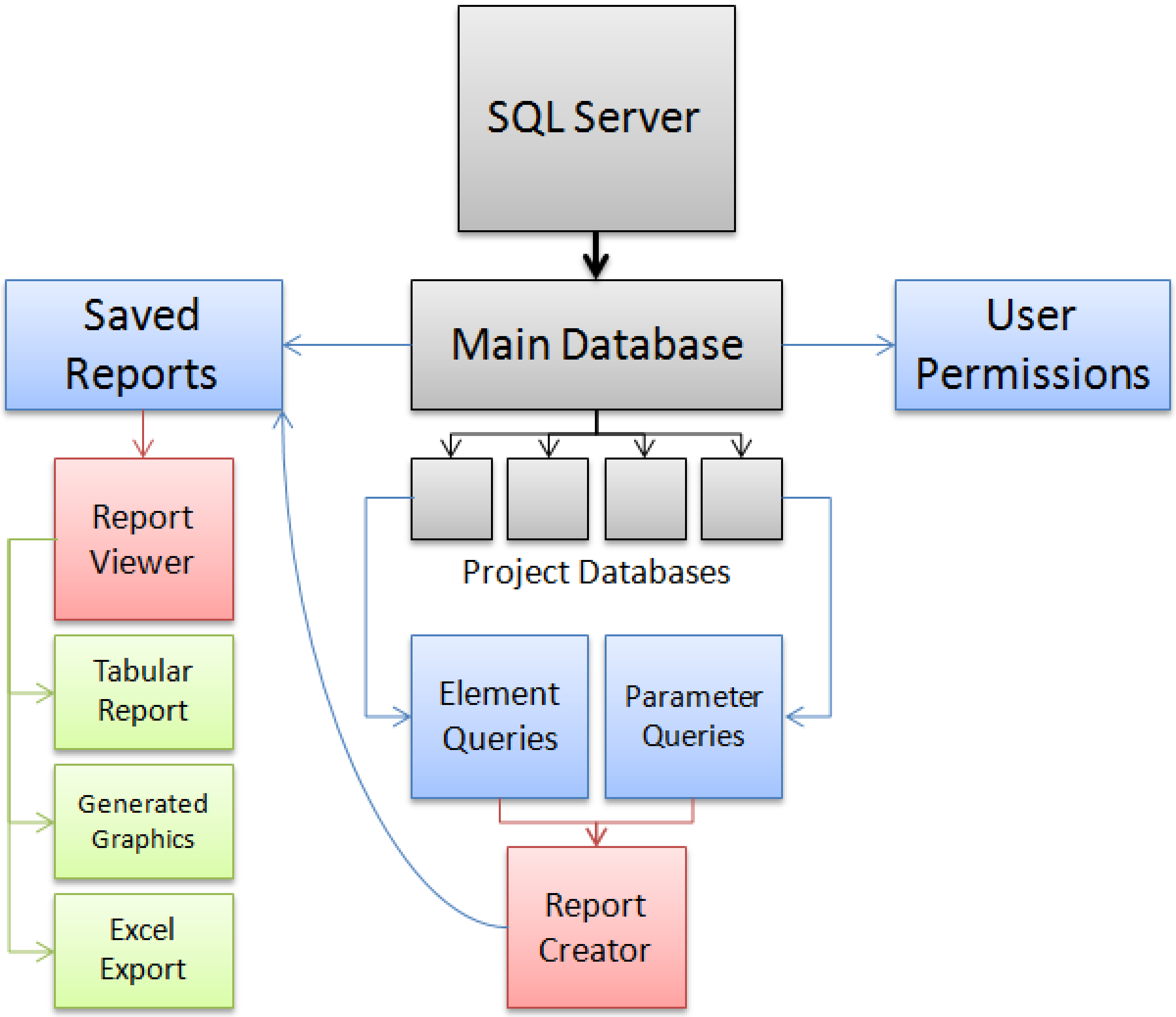


Figure 1: System Architecture

Technical Design

Our program is split into two basic modules: one to create reports and one to view saved reports. Each module, in order to comply with existing site standards used by *BIMShift*, is comprised of three main files: a main PHP file, an AJAX PHP file, and a JavaScript file. The main PHP file controls the general templating of the page's html code and provides div tags with ids that can be dynamically loaded via AJAX. The AJAX PHP file is where the majority of program logic occurs, such as SQL requests, handling AJAX calls from JavaScript and yielding proper html results, storing and modifying html for filtering templates, and performing user permission checks. The JavaScript file handles element event bindings so that buttons and fields are interactive and can use AJAX requests to update html without refreshing the page as well as block user input when waiting on AJAX returns. The general website flow can be seen below in **figure 2**.

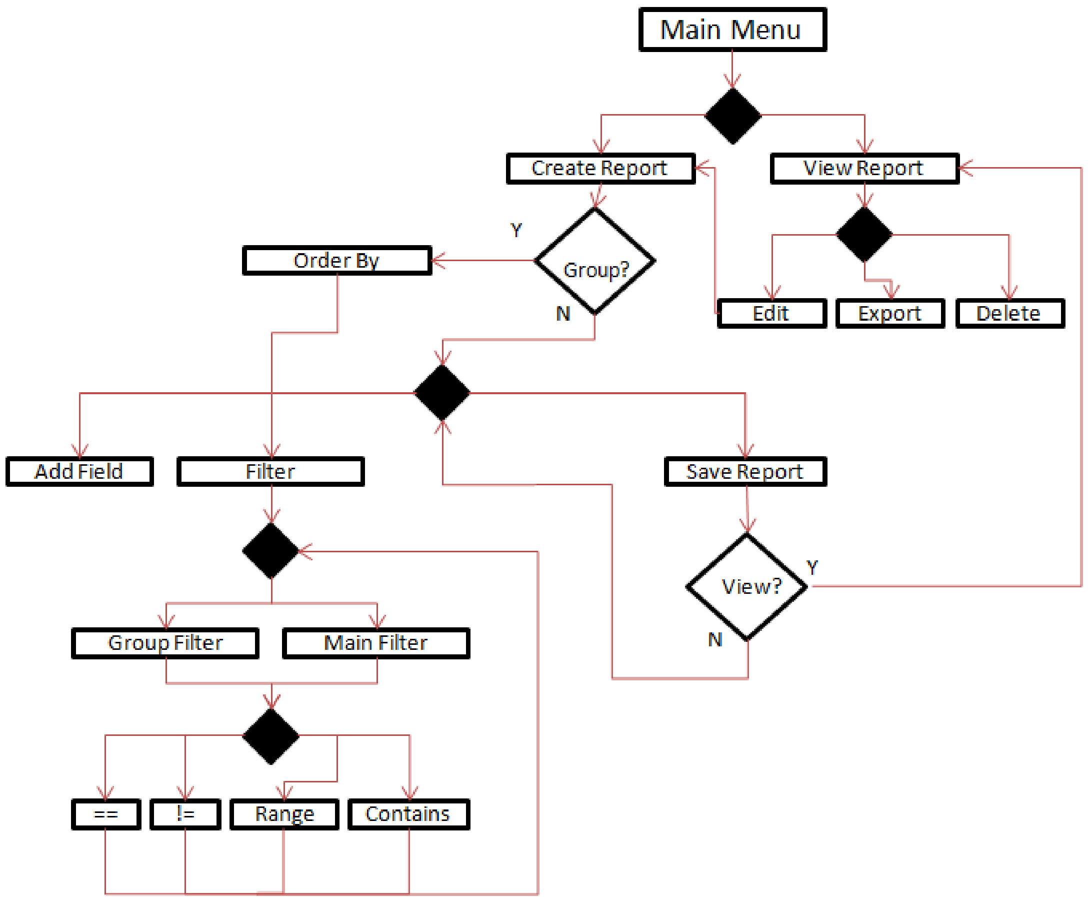
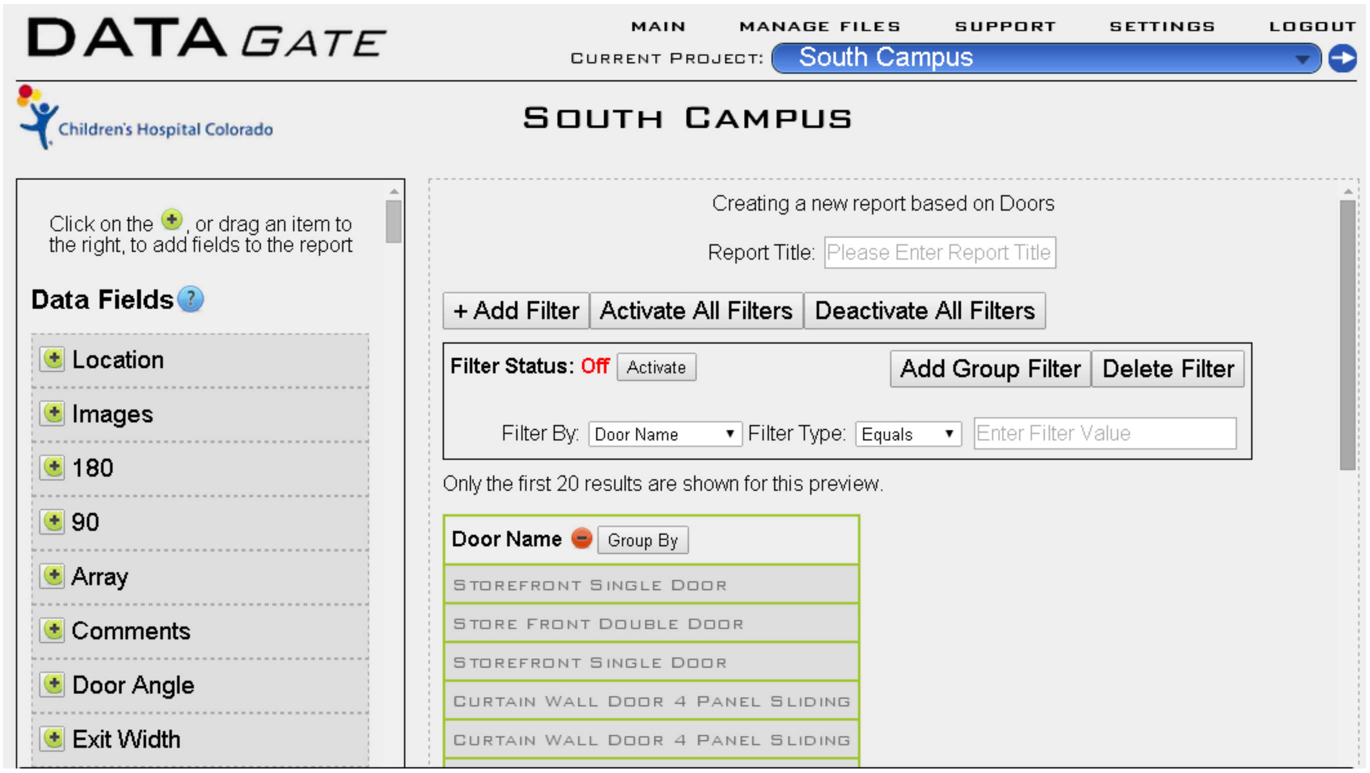


Figure 2: User Control Flow

The create reports page allows the user to leverage complex SQL calls to retrieve element data without knowing about the underlying technical details. Below in **figure 3**



Click on the ⊕, or drag an item to the right, to add fields to the report

Creating a new report based on Doors

Report Title: Please Enter Report Title

Filter Status: Off

Filter By: Door Name Filter Type: Equals Enter Filter Value

Only the first 20 results are shown for this preview.

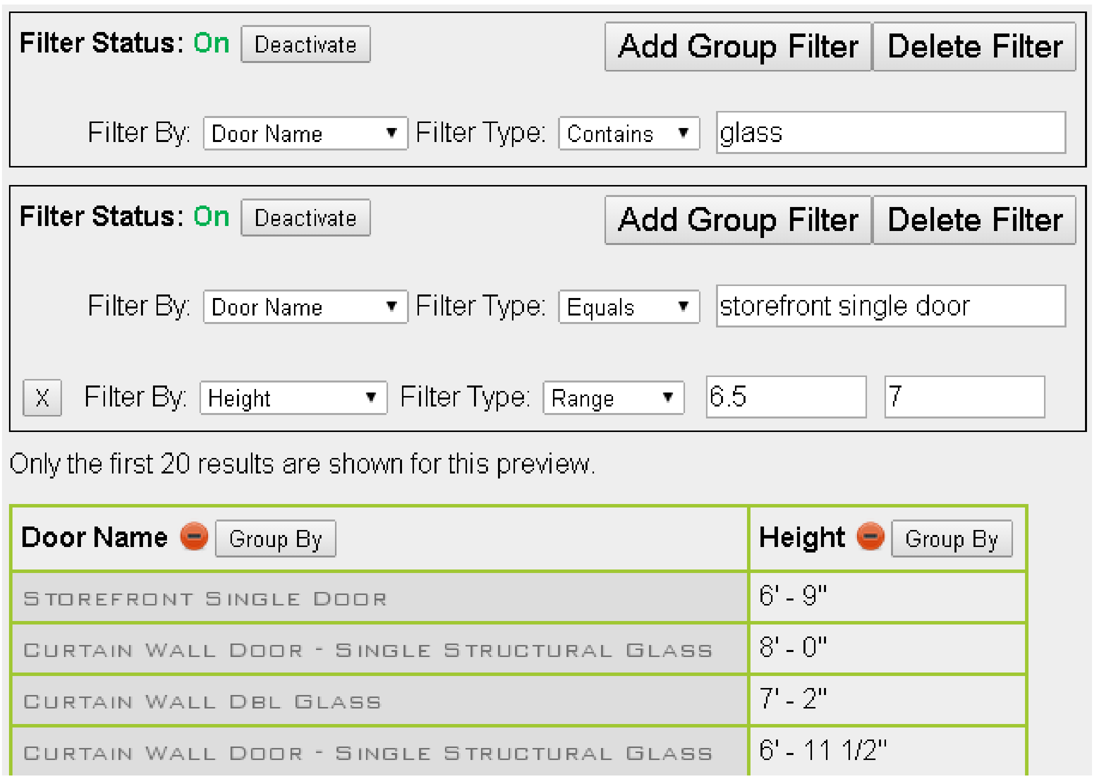
Door Name
STOREFRONT SINGLE DOOR
STORE FRONT DOUBLE DOOR
STOREFRONT SINGLE DOOR
CURTAIN WALL DOOR 4 PANEL SLIDING
CURTAIN WALL DOOR 4 PANEL SLIDING

is the create report page layout.

Figure 3: The Create Report page

The left side of the page contains a list of all of the parameters that a certain type of element, in this case doors, might have. They may be added as columns to the report table below by clicking on the green buttons next to the names or by dragging them into the table. They are retrieved by a SQL call to the parameters tables to retrieve all unique parameters with a category that matches the element category you are creating a report from.

On the right side are the filtering controls, the preview table containing the first 20 results, and a save button (out of frame) at the bottom of the table. The *add filter* button at the top of the page allows you to add a whole new group filter, while the *add group filter* button allows you to add a sub filter to an existing filter group. Reports are generated by matching elements that meet all of the sub filters for *at least one* of the group filters. **Figure 4** shows an example of a filter set that matches a storefront door with a particular height, or all doors that contain glass in their name.



Filter By: Door Name Filter Type: Contains glass

Filter By: Door Name Filter Type: Equals storefront single door

X Filter By: Height Filter Type: Range 6.5 7

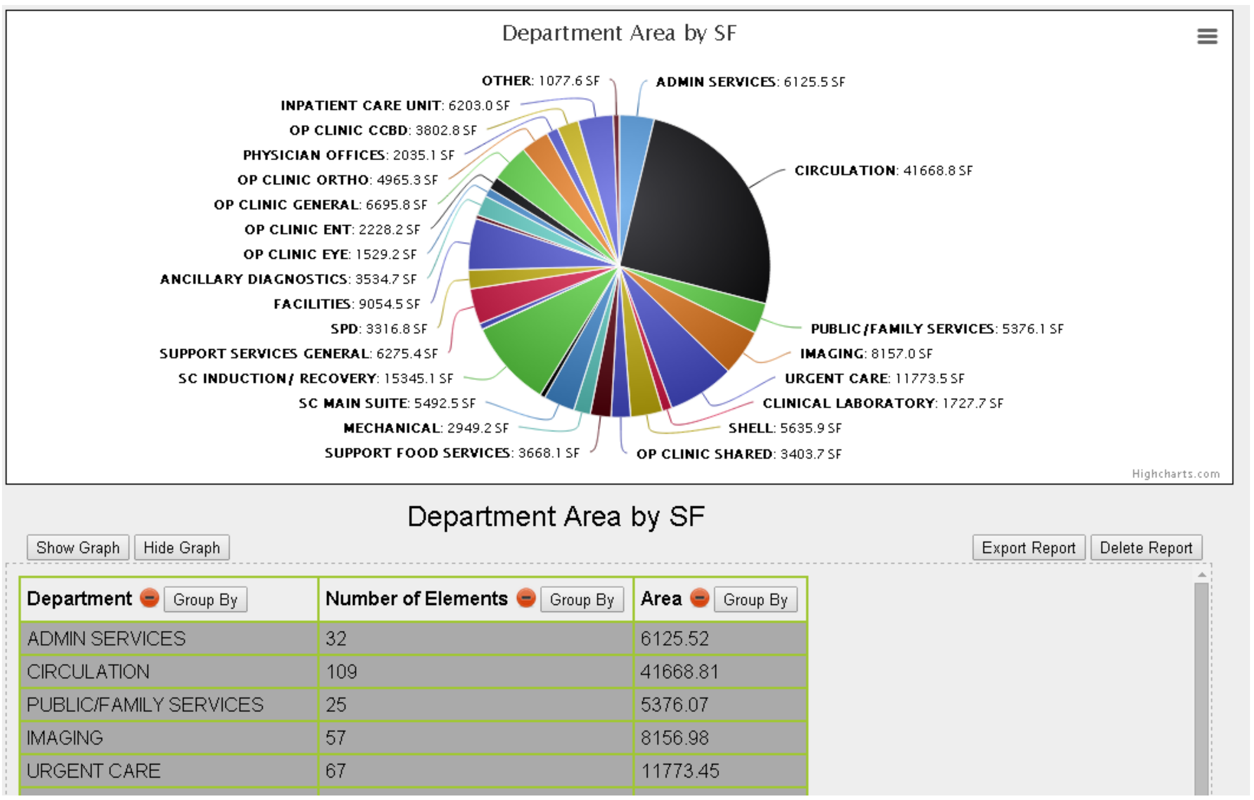
Only the first 20 results are shown for this preview.

Door Name	Height
STOREFRONT SINGLE DOOR	6' - 9"
CURTAIN WALL DOOR - SINGLE STRUCTURAL GLASS	8' - 0"
CURTAIN WALL DBL GLASS	7' - 2"
CURTAIN WALL DOOR - SINGLE STRUCTURAL GLASS	6' - 11 1/2"

Figure 4: An example filtering set to generate a report.

The filter sets generate SQL calls to the database to pull all elements that match the element category of the filtering. The elements are then filtered out in a large PHP foreach loop that checks the elements parameters against the existing filters. An ajax call returns the preview table for the resulting elements that meet the filter requirements.

In the column header for the preview table, there are also *group by* buttons that allow a user to create a grouped report by any field. Grouped reports are unique in that they automatically sum element number and area fields by a certain parameter field and they generate graphical report visuals when viewing the saved graphical report. **Figure 5** shows an example of a grouped report of building room usage by department, along with a pie chart of building usage.


Department Area by SF
OTHER: 1077.6 SF
ADMIN SERVICES: 6125.5 SF
INPATIENT CARE UNIT: 6203.0 SF
OP CLINIC CCBD: 3802.8 SF
PHYSICIAN OFFICES: 2035.1 SF
CIRCULATION: 41668.8 SF
OP CLINIC ORTHO: 4965.3 SF
OP CLINIC GENERAL: 6695.8 SF
OP CLINIC ENT: 2228.2 SF
OP CLINIC EYE: 1529.2 SF
ANCILLARY DIAGNOSTICS: 3534.7 SF
FACILITIES: 9054.5 SF
SPD: 3316.8 SF
PUBLIC/FAMILY SERVICES: 5376.1 SF
SUPPORT SERVICES GENERAL: 6275.4 SF
IMAGING: 8157.0 SF
SC INDUCTION/ RECOVERY: 15345.1 SF
URGENT CARE: 11773.5 SF
SC MAIN SUITE: 5492.5 SF
CLINICAL LABORATORY: 1727.7 SF
MECHANICAL: 2949.2 SF
SHELL: 5635.9 SF
SUPPORT FOOD SERVICES: 3668.1 SF
OP CLINIC SHARED: 3403.7 SF
Highcharts.com
Department Area by SF
Show Graph
Hide Graph
Export Report
Delete Report
Group By
Group By
Group By

Department	Number of Elements	Area
ADMIN SERVICES	32	6125.52
CIRCULATION	109	41668.81
PUBLIC/FAMILY SERVICES	25	5376.07
IMAGING	57	8156.98
URGENT CARE	67	11773.45

Figure 5: Department Rooms by Sq. Ft. Area.

The report graphics are created in JavaScript using a library called HighCharts. First a SQL call is made to loop through the elements and add their area department, storing them in an associative array that is indexed by the other parameter that the user is filtering by. For example, in the above chart the SQL call sums all of the room area amounts and increments them in an array that is indexed by the department name. So, *array_name['ADMIN SERVICES]'* would contain the value *6125.52*. The resulting graph is interactive, animated, and automatically sized and colored from the generated element data.

Design & Implementation Decisions

Instead of writing a report generation page from scratch, we decided to port over a partially finished version that *BIMShift* gave us access to. Although this also required us to change a lot of the code to integrate into the *BIMShift* site which had changed file and database structures since the original version, we felt that this would still be more efficient than creating the page entirely from scratch.

Now that the project is complete we regard this decision with mixed feelings. Although it helped us to initially create a functional filtering system, it became harder and harder to restructure the existing system to fit into the constraints of the rest of the website and the rest of our required functionality. If we had started on the filtering page from scratch, it likely would have become less of a hassle to add functionality and to make the code modular enough to support expanding client requirements and expectations.

We initially decided to use a combination of SQL queries and PHP to filter data rather than just using PHP to post-filter data. This was to make loading times quicker and more efficient, as well as to simplify our code and build off of existing queries. Although this was great for our first iteration of our filtering system, this system fell short of being able to comply with updated client requirements towards the end of the project. We then decided to restructure the project entirely to use only PHP to perform filtering of data so that we could filter by parameters that did not explicitly appear in the report table. For example, if we wanted a table that listed door name and height for doors that had a certain door finish on them, but we don't want the column to display the door finish which is the same across the report.

After examining how the client used a SQL table to store reports in the database, we decided to implement 8 relational tables to store data in a very manageable way. The reports table stores the name of a report and the category of elements it is generating, the filters table stores the information in each user--specified filter such as filter type and value, the filter- groups table stores an ID for relating entire groups of toggle-able filters to perform logical filtering, and the report-fields table stores the different columns used in the report generation. There are also relational tables to relate users who can view a certain report.

Finally we decided to use pre-existing libraries to integrate both chart display, as well as to export tabular data to Microsoft Excel. These libraries, HighCharts and PHPExcel, both contain a lot of unused functionality that bloats the codebase, but we found this acceptable because *BIMShift* already has a structure that supports simple PHP and JavaScript library including.

In working on this project, we have all learned lessons regarding the overall structure of web-site design. We have learned that when working on a project that is not built on a web framework (raw PHP) it is helpful to extract code into included PHP files, otherwise files grow too large and become difficult to navigate. We have also learned how to create carefully balanced interactions between JavaScript and PHP via AJAX calls as well as to prevent sequential AJAX calls from becoming out of sync with the JavaScript.

Results

Our goal was to design a web--based system to create reports based off of existing client data, to integrate this system with their existing website, and to make this system simple enough to use that people who are not familiar with the underlying technology can generate useful reports from the data. The reports are comprised of a mixture of tabular data and an optional graphical display. The data that *BIMShift* records is construction data that is pulled from architectural files for large- scale construction projects; the main project that we use to test is a Children's Hospital construction project. In order to create the tabular reports, we were required to implement a system that allows users to filter through many interrelated SQL tables of hundreds of thousands of elements.

Our final product includes all of the core requirements specified by the client. We have created a system to allow the filtering of data, complex filtering with logical operations, saving and viewing of reports, grouping of fields in reports, inclusion and exclusion of different element parameters, generation of visual reports from grouped data, export of report graphics in several formats, and exporting of tabular report data to Microsoft Excel.

Appendix

Information about some of the libraries and languages we used can be found at the following links:

Highcharts:	*http://www.highcharts.com/*
PHP:	http://www.php.net/
PHPExcel:	https://phpexcel.codeplex.com/
SQL:	http://www.mysql.com/