OGMA Consulting Corp. – Summer 2021

Project Design Document

OGMA Generic Dashboard Builder (GDB)

August 27, 2021

Version 1.0

Team: Slytherin

Kevin Winther, Derrick Chow, Elija de Hoog

Contents

Part 1

Introduction

This document was made for OGMA Consulting Corp. as reference documentation for the Generic Dashboard Builder developed by Team Slytherin over the summer of 2021.

This document describes the design of the Generic Dashboard Builder, the specifications of which are detailed in Appendix A.

1.1 Implementation Plans

We will use Python to implement this. Visualizations are built using Dash by Plotly hosted on a Flask web server. Connections to a SQL database where data frames and visual reference data are stored is required. This is to emulate OGMAs backend environment.

1.2 Document History

This is the only version of this document (version 1.0), as of Aug. 27, 2021

1.3 Github Links and Contribution History

The link to the github as of 2021/08/27 is the following:

https://github.com/MiniTitanGett/OGMA2021_Dashboard

All contribution history, insights on branch history, and git blame history to see who has contributed to what code last can be found there. This was used to track coding commits and coder contributions in lieu of verbal or written reporting.

1.4 Github Ownership and Project History

This project began in its first version the Summer of 2020 with Team Slytherin: Kevin Winther, Elija de Hoog, Kaceja Calder, Callum Curtis and Mike Gruber. The github was maintained by Kevin Winther until leaving where ownership was passed to Callum Curtis who passed it to Mike Gruber.

The project was worked on by Mike Gruber until the Summer of 2021 where Slytherin was brought back as Kevin Winther, Elija de Hoog, Derrick Chow, and Mike Gruber. The new github for the project was maintained by Kevin Winther until leaving where ownership was passed to Mike Gruber.

Part 2

Design Elements

This part lists nouns and facts surrounding the code. The part following this one has more detailed design for each noun.

2.1 List of Nouns

These nouns have been sorted alphabetically and made singular.

- Callbacks
- Dash
- Dash Storage
- Data (Button)
- Dataset
- Data Fitting
- Date Filter
- Delete (Button)
- Edit Menu
- Flask
- Graph
- Hierarchy Filter
- Javascript
- Link/Unlink
- Load Menu
- Load Screen
- Popup Menu
- Prompt
- Resizable Graph
- SQL Server
- Save (Button)
- Server Side Session
- Sidemenu
- Stored Proc.
- Tab
- Tab Title
- Tile

Additional Nouns that haven't been sorted appear as follows:

- Axis Title
- Data Fitting
- Add Tile (Button)
- Reset (Button)
- Close Tile/Tab
- Help Info
- Sessions
- Legend
-
-
-
-
-
-

2.2 List of Facts

These facts are in no particular order.

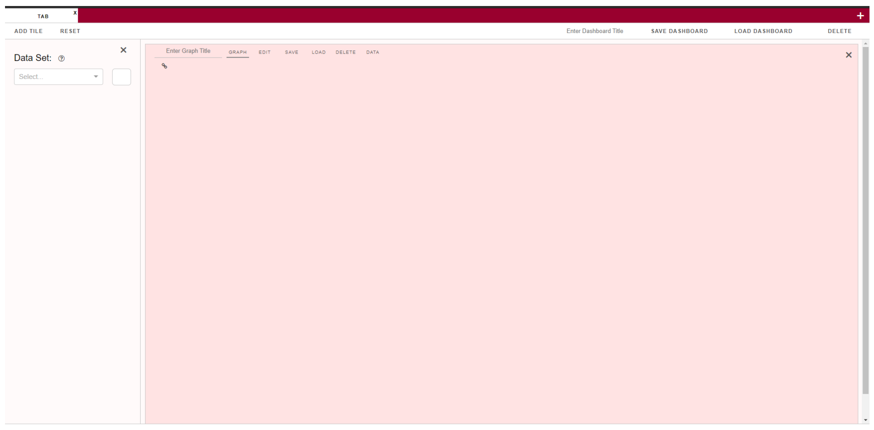
2.3 General Project Structure

The goal of the Generic Dashboard Builder is to allow users of the O&PEN System to visualize data from the system to be easily digested in the form of graphs and interactable dashboard interfaces. To do so the project uses Dash by Plotly and it’s usage of Callbacks, code that links inputs of dash or html components to functions that generate an output to similar dash or html components, to dynamically filter and display data.

The Dash server is hosted on a modified Flask Server that uses Server Side Sessions to handle user information. This Flask Servers GETS and POSTS are linked to a SQL Server which holds Datasets and reference information for sessionID’s and visual text.

On load, the sessionID is accessed and cross referenced in the database to establish user preferences, saved graphs, data access, and other session variables. Once the server side session is established the Dash frontend is requested and loaded and the user is brought to the Generic Dashboard Builder interface.

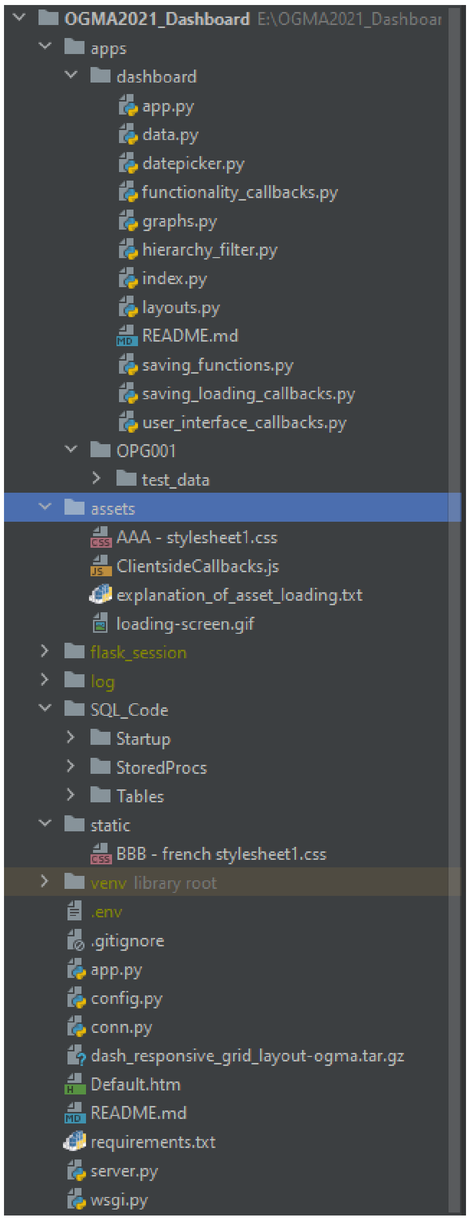
(http://127.0.0.1:8080/python/dashboard/?sessionID=101)



The program uses many different packages to accomplish its tasks. All can be found in requirements.txt. Notable packages are: Dash, Flask, Pandas, Numpy, PYMSSQL (sql integration), flask_session (server side sessions), visdcc (dash javascript runnable component), dash_responsive_grid_layout-ogma.tar.gz (resizable graphs), and scikit-learn (statistical analysis).

2.4 File Structure

The program is split over multiple folders and files. In the main folder OGMA2021_Dashboard there exists the following folders: [apps, assets, flask_session, log, SQL_Code, static, venv (personal)] and the following files: [.env, .gitignore, app.py, config.py, conn.py, dash_responsive_grid_layout-ogma.tar.gz, Default.htm, README.md, requirements.txt, server.py, wsgi.py] The tables next will explain what is in each file and its usage in the program. Italicized functions are client side callbacks.



```
OGMA2021_Dashboard E:\OGMA2021_Dashboar
  apps
    dashboard
      app.py
      data.py
      datepicker.py
      functionality_callbacks.py
      graphs.py
      hierarchy_filter.py
      index.py
      layouts.py
      README.md
      saving_functions.py
      saving_loading_callbacks.py
      user_interface_callbacks.py
    OPG001
      test_data
  assets
    AAA - stylesheet1.css
    ClientsideCallbacks.js
    explanation_of_asset_loading.txt
    loading-screen.gif
  flask_session
  log
  SQL_Code
    Startup
    StoredProcs
    Tables
  static
    BBB - french stylesheet1.css
  venv library root
  .env
  .gitignore
  app.py
  config.py
  conn.py
  dash_responsive_grid_layout-ogma.tar.gz
  Default.htm
  README.md
  requirements.txt
  server.py
  wsgi.py
```

File/Folder Name	Usage	Contains	Other
OGMA2021_Dashboard/app.py	Defines get_app function and external_scripts/stylesheets	get_app()	
.../config.py	Takes in .env file and initializes environmental variables	StreamToLogger(), CustomFileHandler(), logging information and init	
.../conn.py	Connections from Python to SQL Database	CursorByName, get_conn(), close_conn(), exec_storedproc(), exec_storedproc_results(), get_ref()	
.../dash_responsive_grid_layout-ogma.tar.gz	Used for installation of Dash_Responsive_Grid_Layout	bundle.js which can be edited to edit packages JS/REACT	
.../Default.htm	DEPRECATED		
.../README.md			
.../requirements.txt	Holds python package information to be able to run the program		
.../server.py	Initializes Flask server and contains flask request related functions	dict_to_string(), load_labels(), load_saved_graphs_from_db(), load_saved_dashboards_from_db(), validate_session(), load_dataset_list(), before request, before first request, after request, teardown requests handling	
.../wsgi.py	Contains the run function for the dashboard	if __name__ == '__main__' Core run function	
.../apps/dashboard/app.py	Inits app variable for Dash	app def	
.../apps/dashboard/data.py	Data manipulation functions and general use constants	Constants, dataset_to_df(), generate_constants(), data_filter(), customize_menu_filter(), create_categories(), get_label(),	

		linear_regression(), polynomial_regression(), confidence_intervals()	
.../apps/dashboard/datepicker.py	Stores layout and functions required for the datepicker	get_date_picker(), get_date_box(), get_secondary_data(), update_date_columns()	
.../apps/dashboard/functionality_callbacks.py	Stores all callbacks used for functionality (not purely front end components) Focus on datepicker, hierarchy filter, tables, menu. CALLBACK FILE 1/3	_update_graph(), _print_choice_to_display_and_modify_dropdown(), *hierarchy_menu_vis()*, _update_date_picker(), *datepicker_past_x_y_selections()*, split_filter_part(), _update_table(), *_update_data_fitting_options(), _hide_animated_bubble_options()*	Although almost all callbacks deal with some visual portion or output, these are core functionality callbacks.
.../apps/dashboard/graphs.py	Functions to build graphs	set_partial_periods(), get_empty_graph_subtitle(), get_hierarchy_col(), get_line_scatter_figure(), get_bubble_figure(), get_bar_figure(), get_box_figure(), get_table_figure(), get_sankey_figure(), __update_graph()	
.../apps/dashboard/hierarchy_filter.py	Stores layout and functions required for the hierarchy filter	generate_dropdown(), generate_history_button(), get_hierarchy_layout(),	
.../apps/dashboard/index.py	DEPRECATED		Used to be for running the program before wsgi.py
.../apps/dashboard/layouts.py	Stores all main layout information including Tabs, Tiles and menus. Also functions for shifting indices of tiles as it requires modifying the layouts JSON	change_index(), recursive_to_plotly_json(), get_data_set_picker(), get_data_menu(), get_div_body(), get_default_tab_content(), get_layout(), get_layout_graph(), get_dashboard_title_input(),	

		get_layout_dashboard(), get_customize_content(), get_tile(), get_tile_layout(), get_line_scatter_graph_menu(), get_bar_graph_menu(), get_bubble_graph_menu(), get_box_plot_menu(), get_table_graph_menu(), get_sankey_menu(), get_default_graph_options() empty_graph_menu()	
.../apps/dashboard/saving_functions.py	Contains functions used for manipulating graph and dashboard metadata (saving, loading, deleting, editing)	save_layout_state(), save_dashboard_state(), save_layout_to_db(), save_dashboard_to_db(), delete_layout(), delete_dashboard(), load_graph_menu(),	
.../apps/dashboard/saving_loading_callbacks.py	Saving, loading, deleting, editing tiles and dashboards are all dealt within the callbacks stored here. Also deals with sending out prompts, float menus CALLBACK FILE 2/3	*_load_tile_title(),* _update_tile_loading_dropdown_options(), _manage_tile_save_and_load_trigger(), _manage_tile_saves(), _manage_dashboard_saves_and_reset(), _load_select_range_inputs() , *null_reset_function_dropdown()*	
.../apps/dashboard/user_interface_callbacks.py	Callbacks regarding pure visuals or mostly visuals are stored here. Sidemenus, layout serving, new/delete, tabs, float menus, prompts, graph options, graph menus, highlighting, loadscreens	_generate_layout(), *resizeContentWrapper(),* _new_and_delete(), *enable_new_button(),* _change_tab(), *_update_num_tiles(),* _update_tab_title(), _serve_float_menu_and_take_result(), *_init_float_menu(),* _update_graph_type_options(), *_update_data_fitting(),* _update_graph_menu(), _manage_date_sidemenus(), *_resize_for_resizable_graphs(),* *_dataset_confirmation_visu*	

		als(), _highlight_tiles(), graphLoadScreen#(), graphRemoveLoadScreen#(), datasetLoadScreen(), datasetRemoveLoadScreen(), _serve_prompt(), _update_axes_titles(), _update_axes_titles() pt2	
.../assets	Holds visual assets to be loaded when dash is loaded	AAA-stylesheet1.css, ClientsideCallbacks.js, explanation_of_asset_loading.txt, loading-screen.gif	AAA is the main stylesheet
.../flask_session	Holds the server side session file system. DO NOT EDIT		
.../log	Holds the log file		
.../SQL_Code	Holds all SQL_Code to be run into the DB	Startup, StoredProcs, Tables	
.../static	Holds visual assets to be loaded after dash is loaded, ie) French styling	BBB - french stylesheet1.css	BBB is the secondary stylesheet. Similar loading style can be applied for specific client css
.../OPG001/test_data	DEPRECATED		

Part 3

Noun Entries

Each noun will have an entry detailing related facts. If more details are needed, a paragraph is provided. Attributes, behaviors and collaborations will be listed for nouns that may become a class.

3.1 Callbacks

3.1.1 Facts

- Callbacks come in two different flavours, **Server Side** and **Client Side** with **Server Side** callbacks being wrappers around Python functions that take in Inputs, States, and return to Outputs while **Client Side** callbacks are written in Javascript but function the same way.
- See https://dash.plotly.com/advanced-callbacks and https://dash.plotly.com/clientside-callbacks
- Daisy Chained Callbacks are allowed, Circular Callbacks are to be avoided (Circular within the same function is allowed though)
- Core to callbacks are **Pattern Matching Callback**, which is used extensively in the project to deal with objects that are similar or linked to a specific tile (generally 0-3 tiles, 4 for parent linked tiles) ex) {type: float_menu_result, index: 0}
- Callback execution order is random unless specifically chained together Output to Input
- Callbacks wrapped in for loops are done so since using ALL or MATCH wasn't possible. The issue with doing so is that when one callback in the loop in called, all callbacks with a lower value in the loop are also triggered ex) a callback wrapped in a loop with x = 2 is triggered, callbacks where x = 0, 1, 2 will be triggered in (decreasing order? Unable to remember exactly)

3.1.2 Summary

Callbacks are integral to Dash and the Generic Dashboard Builder. Client Side Callbacks can be found by inspecting the webpage and navigating to the (index) file in Sources.

3.1.3 Other Notes

Stored in ClientsideCallbacks.js, user_interface_callbacks.py, functionality_callbacks.py, saving_loading_callbacks.py

3.2 Dash

3.2.1 Facts

- Core package used in the project
- See https://dash.plotly.com/
- See https://community.plotly.com/c/dash/16

3.2.2 Summary

Dash by Plotly is the core package used in the project.

3.2.3 Other Notes

3.3 Dash Storage

3.3.1 Facts

- dcc.Stores used as client side session memory, exist until refreshed
- Used as hidden divs for chained callbacks as users can't interact with them using the inspect console
- See https://dash.plotly.com/dash-core-components/store

3.3.2 Summary

Dash Storage or dcc.Store are dash components used frequently in the GDB. They are used most commonly to store global variables since global instances can be edited and modified by every Dash instance. Examples are df_const (constants for loaded dataframes) and {type: float_menu_result, index: 0} (used to handle chain inputs from _manage_tile_save_load_trigger())

3.3.3 Other Notes

3.4 Data (Button)

3.4.1 Facts

- The data button resides in the tile header with the id = {'type': 'tile-data', 'index': tile}
- When clicked this brings up the data **Sidemenu** for the respective tile, if linked it will bring up **Sidemenu** 4

3.4.2 Summary

3.4.3 Other Notes

3.5 Dataset

3.5.1 Facts

- Comes in three flavours as of 2021/08/25: OPG001, OPG010, and OPG011
- Used in the Dataset picker to load a dataset for usage in respective tiles
- Ingested by stored proc and translated into Pandas dataframe for ease of modification. (Categories are applied to most columns to reduce strain on row manipulations)

3.5.2 OPG001

OPG001 is the first dataset ever used by the GDB. It is based on time related data, contains a core hierarchy and a secondary "variable" hierarchy, and date information (Fiscal, Gregorian, Julian). The columns in the database are as follows:

> [OPG Data Set], [Hierarchy One Name], [Hierarchy One Top], [Hierarchy One -1], [Hierarchy One -2], [Hierarchy One -3], [Hierarchy One -4], [Hierarchy One Leaf], [Variable Name], [Variable Name Qualifier], [Variable Name Sub Qualifier], [Date of Event], [Calendar Entry Type], [Year of Event], [Quarter], [Month of Event], [Week of Event], [Fiscal Year of Event], [Fiscal Quarter], [Fiscal Month of Event], [Fiscal Week of Event], [Julian Day], [Activity Event Id] , [Measure Value], [Measure Type], [Partial Period]

The allowed graphs to be built out of this dataset are:

> ['Line', 'Bar', 'Scatter', 'Bubble', 'Box_Plot', 'Table']

3.5.3 OPG010

OPG010 is the second dataset developed and is specifically used for building Sankey graphs. It requires two datasets, the core data and an auxiliary NodeData dataset. In the main dataset the columns are similar to OPG001:

> [OPG Data Set], [Hierarchy One Name], [Hierarchy One Top], [Hierarchy One -1], [Hierarchy One -2], [Hierarchy One -3], [Hierarchy One -4], [Hierarchy One Leaf], [Variable Name], [Variable Name Qualifier], [Variable Name Sub Qualifier],[Date of Event],[Calendar Entry Type],[Year of Event], [Quarter], [Month of Event], [Week of Event], [Fiscal Year of Event], [Fiscal Quarter] ,[Fiscal Month of Event], [Fiscal Week of Event], [Julian Day], [Activity Event Id] ,[Measure Value], [Measure Type], [Partial Period], [Measure Id] ,[Measure Fcn], [MeasQual1], [MeasQual2], [empty column], [Comment]

In the NodeData dataset the columns are for the sankey visuals and are:

[node_id], [x_coord], [y_coord], [colour]

The allowed graphs to be built out of this dataset are:

[‘Sankey’, ‘Table’]

3.5.4 OPG011

OPG011 is the third dataset that looks to condense the information found in the OPG001 and move some of the building of the dataset (hierarchies and aggregation) from the SQL server to the GDB. To do so, functions and stored procs were created to build the hierarchies (leaf to tree built by a reverse breadth first tree traversal). This dataset doesn’t look to add any new graph types but instead improve upon the efficiency of OPG001. The columns used in this dataset are:

[Hierarchy Value], [Hierarchy Level], [Variable Value], [Variable Level], [Date of Event], [Calendar Entry Type], [Activity Event Id] , **[Measure Value 1], [Measure Type 1], [Measure Value 2], [Measure Type 2], [Measure Value 3], [Measure Type 3], [Measure Value 4], [Measure Type 4], [Measure Value 5], [Measure Type 5],** [Partial Period]

NOTE: Bolded columns are not currently supported as of 2021/08/25 but will be in the future

The allowed graphs to be built out of this dataset are:

['Line', 'Bar', 'Scatter', 'Bubble', 'Box_Plot', 'Table']

3.5.3 Other Notes

It should be noted that the format of the datasets and the relations between each column is what matters to the program, what is stored in the columns is not. One could replace all organizations with species of Dinosaurs in OPG011 and the program will work the same since the data is just being filtered and manipulated by the program.

3.X Data Fitting

3.X.1 Facts

- The data fitting resides in the edit menu when a line or scatter graph type is selected.
- When a specific item is selected in the hierarchy filter the data fitting options will become visible and the user will be able to choose from: no fit, linear fit and a curve fit.
- Additionally when a linear fit or curve fit is selected the user will be presented with the option to display the confidence interval of the fit

3.X.2 Summary

- Both the linear and polynomial fits are created from a statistical model from the ordinary least squares method and the model returns the prediction of the best fit.
- The confidence interval calculates standard deviation and confidence interval for prediction and returns the upper and lower confidence intervals.

3.X.3 Other Notes

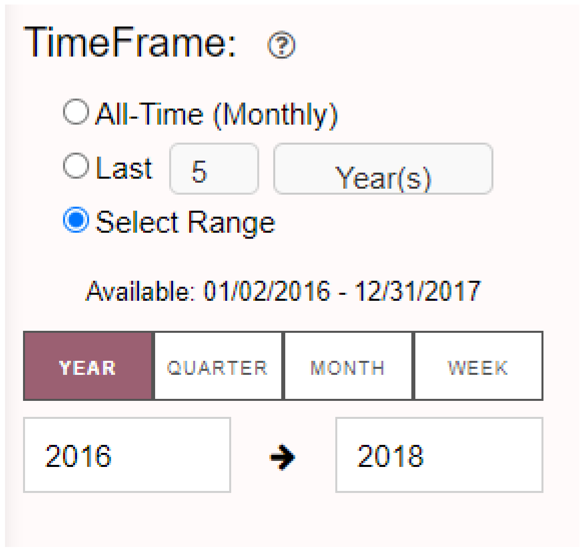
3.6 Date Filter

3.6.1 Facts

- The date filter, contained in datepicker.py, is located in the side menu and is a common filtering component.
- It uses df_const extensively to find limits for its dropdowns so users don't try to display data that doesn't exist
- 3 Modes (Gregorian, Fiscal, *Julian (not supported)*) with 3 selections (All-Time (Monthly), Last x y, Select Range)

3.6.2 Summary

The date filter, located under the "TimeFrame:" header in the side-menu along with the Hierarchy Filter make up the common filtering controls.



TimeFrame:

All-Time (Monthly)

Last 5 Year(s)

[✓ Select Range]

Available: 01/02/2016 - 12/31/2017

[✓ YEAR] QUARTER MONTH WEEK

2016 → 2018

3.6.3 Other Notes

3.7 Delete (Button)

3.7.1 Facts

- The delete button for each tile resides in the tile header with the id = {'type': 'delete-button', 'index': tile}
- The delete button for the dashboard resides under the parent nav bar with the id = ‘delete-dashboard’
- When clicked this calls a prompt allowing the user to delete the graph that has been currently loaded into that tile or dashboard, removing it from the session and the database.

3.7.2 Summary

3.7.3 Other Notes

3.7 Edit Menu

3.7.1 Facts

- The edit menu is contained in the tile but is hidden until the Edit button is selected which prompts the float menu to pull the edit menu from the tile and put it in the float menu via Javascript (Client Side Callback).
- The edit menu contains the graph type dropdown which allows the user to pick what graph they would like to build. A selection causes the graph menu contained in the edit menu to change to that of the requested layout. Defaults are set by callbacks.

3.7.2 Summary

The Edit Menu is the key way to personalize each tile and how to access graph specific options. The user can choose to accept the changes they have made or cancel/close to revert to what they previously had.

3.7.3 Other Notes

3.8 Flask

3.8.1 Facts

- Web server package used in GDB
- Session variable is client side unless Flask-Session is used which will eliminate client side session for server side sessions
- Flask-Caching (not flask-cache as it is not kept up) can be used to incorporate memoization and function caching
- Flask cookies (4kb max) act like dictionaries with set_cookie(), requests.cookie.get(key)
- See https://flask.palletsprojects.com/en/2.0.x/
- For Flask-Session, See https://flask-session.readthedocs.io/en/latest/

3.8.2 Summary

Flask is the web framework used for the Dash GDB design and is regarded as a microframework. It runs in a python instance and is easy to extend.

3.8.3 Other Notes

Located in server.py primarily and in wsgi.py

3.9 Graph

3.9.1 Facts

- Lives in the tile and should be kept visible while the user makes changes
- Comes in many flavours, exist as plotly figures in the visual div {type: tile-view-content, index: tile} -> {type: graph_display, index: tile}
- Should accurately display all information given through the appropriate filters

3.9.2 Line/Scatter

The line chart and scatter plot run off the same function as they are very similar when building each in plotly. The chart should have:

> [title, x legend position, y legend position, grid lines, show/hide legend, xaxis title, xaxis measure (forced to date), yaxis title, yaxis measure, graphed variables, display options (line, point, lines and points), data fitting options (if specific hierarchy)]

The traces should be grouped by colour with graphed variables being the colour if in specific hierarchy mode and trace titles being the colour if in level filter or specific-graph-children mode. Data fitting options may be turned off in level filter and specific-graph-children mode and on otherwise. Options include lines of best fit, curve fit lines, and upper/lower interval lines.

3.9.3 Bubble

The bubble chart is an animated plot (unless the xaxis is set to Time) where the bubbles move across the plot growing and shrinking to demonstrate changes over time. The chart should have:

> [title, x legend position, y legend position, grid lines, show/hide legend, xaxis title, xaxis type, xaxis measure, yaxis title, yaxis type, yaxis measure, size type, size measure]

Play and Pause buttons control the graph's time dimension and the scroll bar allows the user to select snapshots. This is disabled if the user has selected the xaxis to be time where the plot will be that of a scatter plot with varying sizes

3.9.4 Bar

The bar plot should have:

> [title, x legend position, y legend position, grid lines, show/hide legend, xaxis title, xaxis group function, yaxis measure, variable name selector, orientation selection, animation flag]

The grouping on the bar plot works as follows, if on hierarchy level filter or specific-graph-children mode then the graph will group by names in the level. If group by variable names is selected then colours will be applied to each name in the level and the xaxis will be variable names. Values per date will be stacked on top of each other with hoverable data explaining each date and its respective value. Grouping by specific item on the specific-item mode in the hierarchy will give the xaxis as date and time. If grouped by variable names the xaxis will be variable names and values per date will be stacked on top of each other with hoverable data explaining each date and its respective value. Colours are either used for hierarchy values or variable names.

Graph orientation can be swapped through the toggle. Selecting animate over time will translate the group by to that of variable names and will display the graph as it grows and shrinks to show the change of values overtime.

3.9.5 Box

The box plot should have:

> [title, x legend position, y legend position, grid lines, show/hide legend, xaxis title, yaxis title, axis measure, graphed variables, orientation selection, data point flag]

The box plot uses built in plotly stats to determine the max, min, quartile 1, quartile 3, and median.

3.9.6 Table

The table should have:

> [title, page size]

The table simply displays the raw data in filterable columns through the use of Dash-DataTable. (See https://dash.plotly.com/datatable) Since filtering and viewing such large datasets on the clients browser would take up significant space, filtering and displaying is handled server side through the _update_table() callback and its helper function split_filter_part() located in functionailty_callbacks.py under its own header. Page count is calculated in the return function. The server side rendering is why the table seems to render itself twice when built. That is because it is.

3.9.7 Sankey

The sankey plot should have:

> [title,graphed variables, variable selector, hierarchy level dropdown, hierarchy path, hierarchy type]

Sankey displays the work flow of an organization and the width of the arrow is proportional to the flow rate.

TODO: Add more info

3.9.3 Other Notes

3.10 Hierarchy Filter

3.10.1 Facts

- Has 2 main modes, Level Filter and Specific Item where Specific item has the option to "Graph All in Dropdown" or "Graph All Siblings" when at a leaf node. This causes the Specific Item mode to work similarly to the Level Filter for graphs
- Level Filter features a dropdown to select what hierarchy level to filter on (H0 - H5) and the Specific Item features a navigable list of buttons underneath the main selector. The list of buttons are clickable allowing the user to pop to that are in the hierarchy. To traverse downwards the user uses the drop down. To pop to the top the user can click the Top button. To go back a selection the user can click the back button.

3.10.2 Summary

The hierarchy filter, located under the "Hierarchy:" header in the sidemenu along with the DatePicker make up the common filtering controls.

3.10.3 Other Notes

3.11 Javascript

3.11.1 Facts

- Javascript is used in multiple areas in the GDB code. It is found in the assets folder as single run code to be run as the dash instance starts and is found in the callbacks as clients side callbacks.
- The form of JS found in callbacks are written as text making syntax errors hard to view. When debugging, checking the index of the web page under sources in a browser such as Chrome is recommended.
- Client Side Callbacks are used over Server Side Callbacks as they execute faster and do not require GETS and POSTS to the python server for code execution. This means python functions and SQL Server accesses are kept away from our JS
- Some REACT JS can be found in the bundle.js in dash_responsive_grid_layout-ogma.tar.gz.
- VISDCC Javascript components are used to catch JQUERY events set up by a Clientside callback (_update_axes_titles() functions in user_interface_callbacks). These specialty components are used to use JS to directly interact with the REACT DOM since the DOM used in REACT is different from the one used in base HTML.

3.11.2 Summary

3.11.3 Other Notes

See https://github.com/jimmybow/visdcc for VISDCC

See https://reactjs.org/ for REACT

See https://dash.plotly.com/clientside-callbacks for Client Side Callbacks

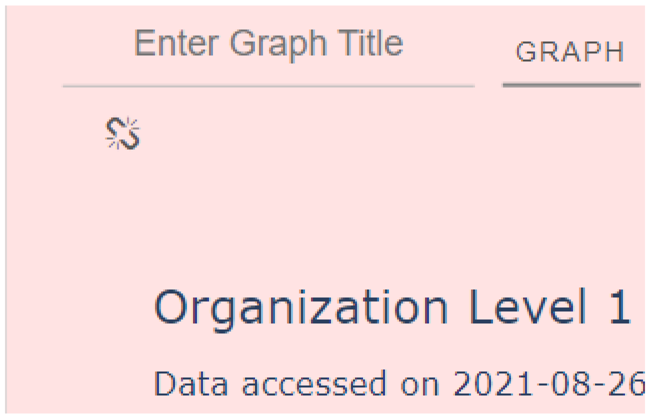
3.12 Linking/Unlinking

3.12.1 Facts

- fa fa-link is linked to the parent dataset which is tile index: 4 and all the tiles that are linked are manipulated by the parent dataset.
- fa fa-unlink is referencing the tiles that are not linked to the master and have their own data menu to manipulate the graph.
- When clicking the link button it will change from link to unlink and vice versa. Change the behaviour of the graph to depend of the parent dataset

3.12.2 Summary

The linking and unlinking icon can be found in the top left corner of the graph right under the tile title input box.



Organization Level 1

Data accessed on 2021-08-26

3.12.3 Other Notes

callback for linking and unlinking can be found in the saving_loading_callback.py

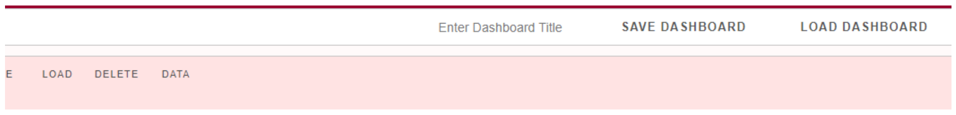
3.13 Load Menu

3.13.1 Facts

- Their two separate load menu one for just graph and one for dashboards
- Each individual user has a different load menu with the graph or dashboards they have saved as based on sessions.
- Other users will not have access to change graphs you made in your session.**This is going to be changed in the future when the GDB is incorporated into the O&PEN system so that some graphs will be accessible by multiple users**
- pop up load menu appear once the user click the load button
- once loading is confirm another prompt warning if you haven't save the graph or dashboard to either save the graph or overwrite the tile or dashboard

3.13.2 Summary

The load menu is a prompt menu which appears once the user clicks the load button for either the tile or load dashboard. The session users have only their saved graph/ dashboards.



3.13.3 Other Notes

callback for load menu can be found in the saving_loading_callback.py

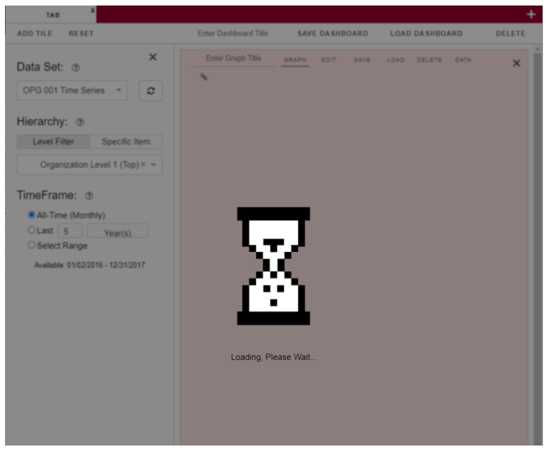
3.14 Load Screen

3.14.1 Facts

- The load screen, a faded grey overlay with a spinning hourglass gif, occurs when the user is loading or refreshing a dataset, loading a graph, or loading a dashboard.
- Load screens disappear when the graph_display updates meaning that a graph has been drawn. Dashboard Loadscreens (Fullscreen load screens) disappear when the graph_display is updated and contains the input in callback context that is equal to the number of tiles in the callback.
- The Input('num-tiles', 'data-num-tiles') is added to ensure the callback is called after the num-tiles callback to ensure the most current results.

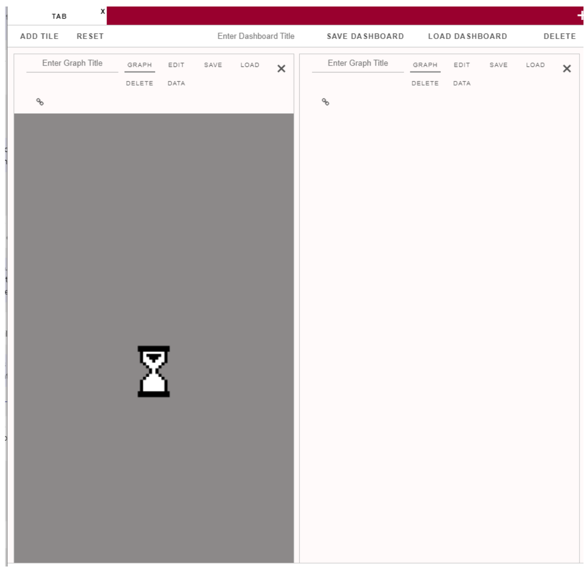
3.14.2 Summary

Here is the dashboard loadscreen:



Data Set:
OPG 001 Time Series ...
Hierarchy:
Level Filter | Specific Item
Organization Level 1 (Top)
TimeFrame:
(●) All-Time (Monthly)
( ) Last 5 Year(s)
( ) Select Range
Available: 01/02/2016 - 12/31/2017
Loading, Please Wait...

Here is the tile load-screen:



3.14.3 Other Notes

Callbacks regarding the loading screen can be found in assets/ClientsideCallbacks.JS and user_interface_callbacks.py

The gif of the hourglass can be changed in AAA - stylesheet1.css, it should be stored in assets for initial loading.

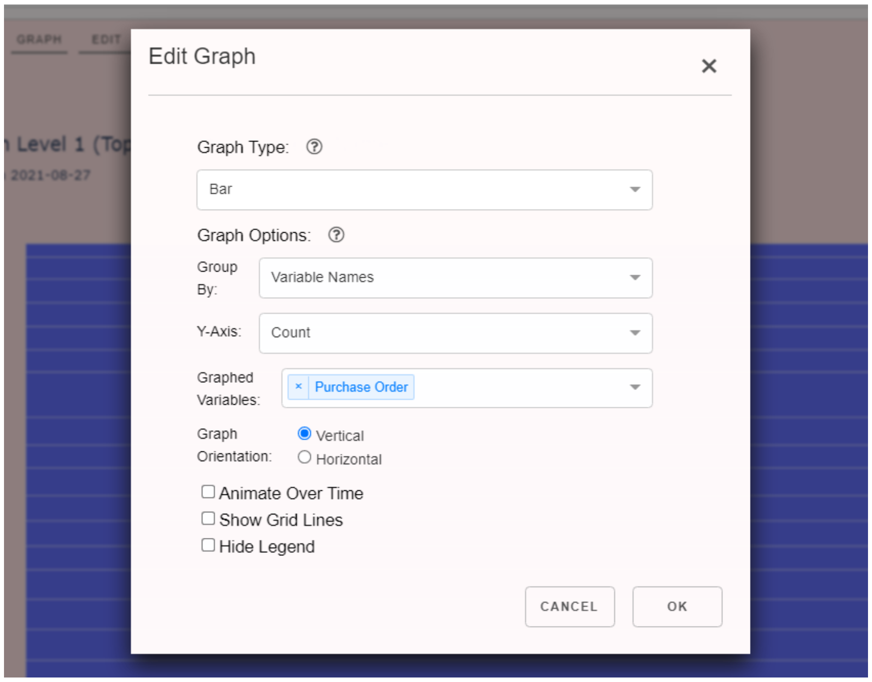
3.15 Popup Menu

3.15.1 Facts

- The pop up menu consists of edit and load, and load dashboard.
- floating menu and can see the graph be edited in the background
- grays out the dashboard to make the popup menu stand out

3.15.2 Summary

What the popup menu looks like:



Edit Graph ×

Graph Type: Bar

Graph Options:

Group By: Variable Names

Y-Axis: Count

Graphed Variables: × Purchase Order

Graph Orientation: [✓ Vertical] Horizontal

☐ Animate Over Time
☐ Show Grid Lines
☐ Hide Legend

CANCEL OK

3.15.3 Other Notes

callback for popup menu can be found in the saving_loading_callback.py

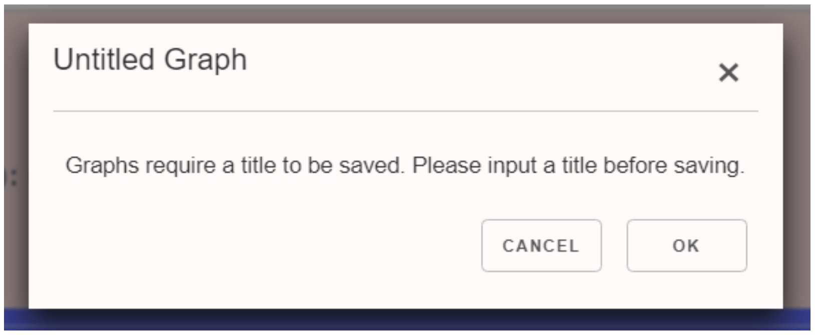
3.16 Prompt

3.16.1 Facts

- Popup that you can either click cancel or okay
- Special prompt for loading dataset the options are unlink, cancel, link but change when loading a dataset or swapping a dataset on a linked tile
- warning the user that they might lose the graph/ dashboard if they have not saved the graphs before loading or can cancel and save it before loading/changing datasets.
- An alert will for saving and loading indicating the changes have been made

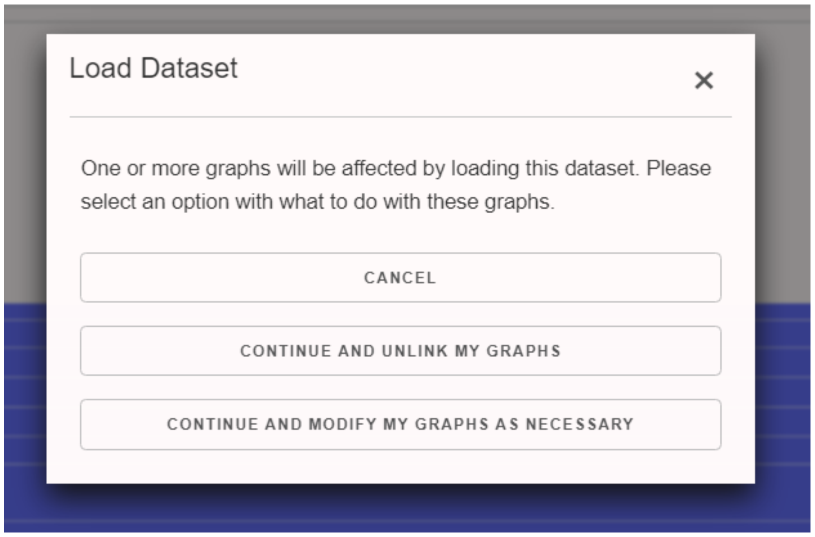
3.16.2 Summary

Regular prompt:


Untitled Graph

Graphs require a title to be saved. Please input a title before saving.

The special prompt:


Load Dataset

One or more graphs will be affected by loading this dataset. Please select an option with what to do with these graphs.

×
CANCEL
CONTINUE AND UNLINK MY GRAPHS
CONTINUE AND MODIFY MY GRAPHS AS NECESSARY

3.16.3 Other Notes

callback for popup menu can be found in the
saving_loading_callback.py

3.17 Resizable Graph

3.17.1 Facts

- See section [**4.4 grid_layout_Info]** for information regarding Resizable Graphs and the dash_responsive_grid_layout package
- Resizable Graphs were added to allow the user to modify the dashboard layout with even more customizability by allowing Tiles to grow, shrink, and be placed anywhere on the layout.
- Layout information, when saved as a dashboard, is stored in the dashboard dictionary under 'Dashboard Layout' in the main part of the dictionary.
- Base layout for when tiles are added are stored in the data.py under the constant LAYOUTS
- Tiles wrapped in the layout have a unique naming scheme ("tile-wrapper: " + str(tile)). This is done to avoid putting dictionaries directly in dictionaries which saves correctly, but when loading the program at start up causes errors.
- Tiles can be dragged by the header, resized by the little arrow on the bottom right corner

3.17.2 Summary

A feature request that allows users to resize and move their graphs. Uses the custom package dash_responsive_grid_layout to accomplish this. Install directions can be found in section **[4.3 Project Info]**

3.17.3 Other Notes

3.18 SQL Server

3.18.1 Facts

- The SQL Server is set to hooked up to the OGMA servers with stored procs for access.
- TODO: More info on SQL Server (Mike)

3.18.2 Summary

3.18.3 Other Notes

3.19 Save (Button)

3.19.1 Facts

- The save button will save the users graph or dashboard and is located in the header menu for the tile and dashboard.
- Saving will not prompt you unless the tile you are saving will be overwritten, the dashboard you are saving will be overwritten, or the tiles in the dashboard you are saving will be overwritten.
- Dashboard saving will save all the dashboards tiles and will write to the database pointers to the tiles in the dashboard.
- Once saving is completed an alert letting you know that the dashboard or tile has been saved for a few seconds.
- Saved tiles/dashboards are to be stored in the SQL database under a users sessionID. This makes it so users will not be able to access saved tiles/dashboards they did not make. **This is going to be changed in the future when the GDB is incorporated into the O&PEN system so that some graphs will be accessible by multiple users**

3.19.2 Summary

The save button is used to save graphs and dashboards to a user's sessionID for later use. It is used in conjunction with the Load (Button) and Load Menu to do so. Saved tiles/dashboards are stored in the SQL Database in OP_ref.

3.19.3 Other Notes

3.20 Server Side Sessions

3.20.1 Facts

- In order to have server side sessions one must install Flask-Session. This will replace Flasks normal Session variable with a new server-side session.
- The server side session or Session from here on out is using a file system server as of (2021/08/27)
- The Session server can be configured to many different variables such as Redis, Memcached, MongoDB, or SQLAlchemy
- The file system is stored in the folder flask_session. To clear all sessions, delete the contents of this folder. BE CAREFUL IN DOING SO
- For Flask-Session, See https://flask-session.readthedocs.io/en/latest/

3.20.2 Summary

Flask-Session is used as a package to initialize the Session variable used by Flask to a File System driven Server Side Session. Info on the package can be found on it's website: https://flask-session.readthedocs.io/en/latest/

3.20.3 Other Notes

Session is used in many places as a dictionary. Session is initialized in server.py

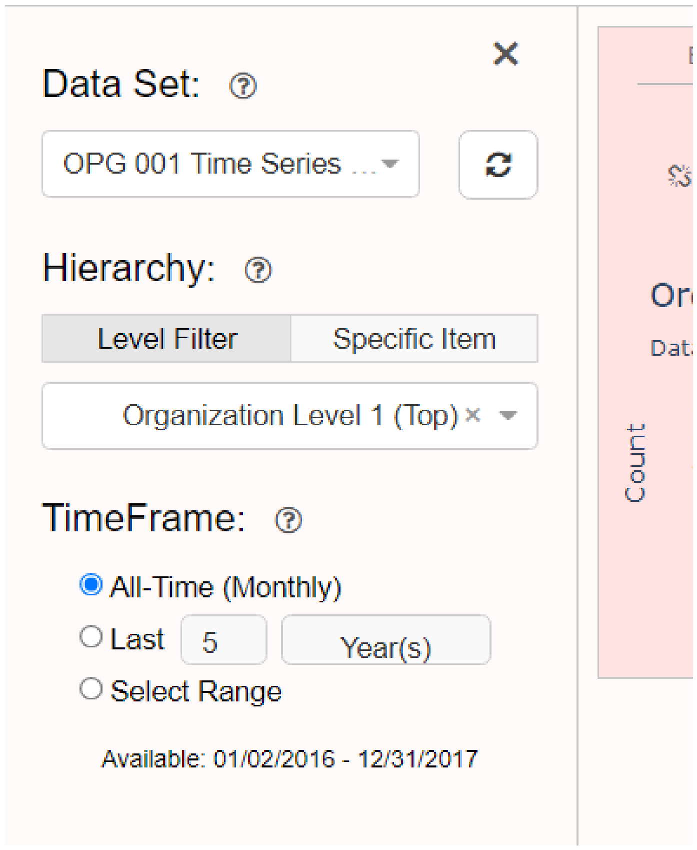
3.21 Sidemenu

3.21.1 Facts

- Consists of the dataset, hierarchy, date
- To learn more about the datasets see section [**3.5 Dataset**] and hierarchy [**3.10 Hierarchy Filter**] and date filter [**3.6 Date Filter**]
- If the side menu is closed it can be accessed by clicking the data button in the tile.

3.21.2 Summary

Give the user the ability to filter the data and choose what data to look at. Can be found next to the tiles.



3.21.3 Other Notes

callback can be found under the user_interface_callback.py

3.22 Stored Procs

3.22.1 Facts

- group of transact-SQL statements compiled into a single execution plan.
- Preserves data integrity
- TODO: More info on Stored Procs (Mike)

3.22.2 Summary

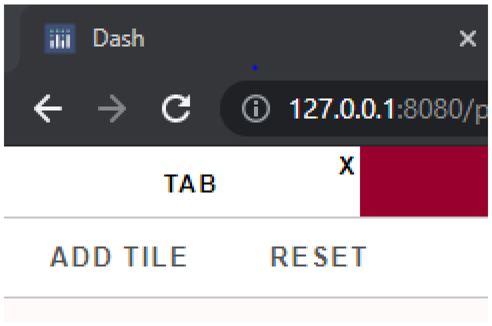
3.22.3 Other Notes

3.23 Tab

3.23.1 Facts

- The graphical engine can have up to 12 tabs and each tab can have 4 tiles.
- tabs are dashboards and are not connected to other dashboards.
- Tabs are at the top of the application right under the address bar
- the look mimics chrome tabs
- the tab that is selected is highlighted white and the text colour is inverted
- new tab will appear beside the tab

3.23.2 Summary



3.23.3 Other Notes

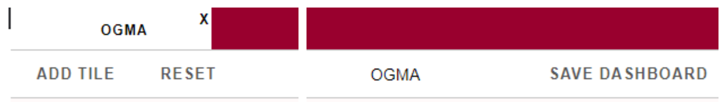
3.24 Tab Title

3.24.1 Facts

- Input box next to the save dashboard button
- The title over writes the default title

3.24.2 Summary

Example of a user setting a title for the tab.



3.24.3 Other Notes

We did not implement the title to auto adjust to the size of the button.

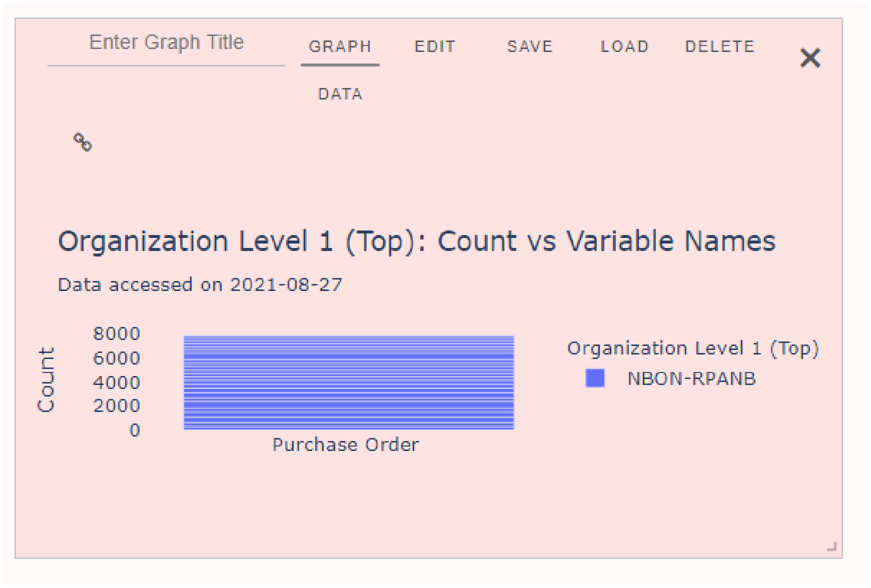
3.25 Tile

3.25.1 Facts

- At the moment you are only able to have 4 tiles per tab
- Depending if the tile is linked to the parent the tile will either have its own data menu or connect to the parent.
- Tiles can be moved around and resize to make the graphs bigger or smaller. See section[**3.17 Resizable Graph**]
- tiles that are highlighted are linked to parent dataset

3.25.2 Summary

View of one tile:


Enter Graph Title
GRAPH
EDIT
SAVE
LOAD
DELETE
DATA
Organization Level 1 (Top): Count vs Variable Names
Data accessed on 2021-08-27
Count
8000
6000
4000
2000
0
Purchase Order
Organization Level 1 (Top)
NBON-RPANB

3.25.3 Other Notes

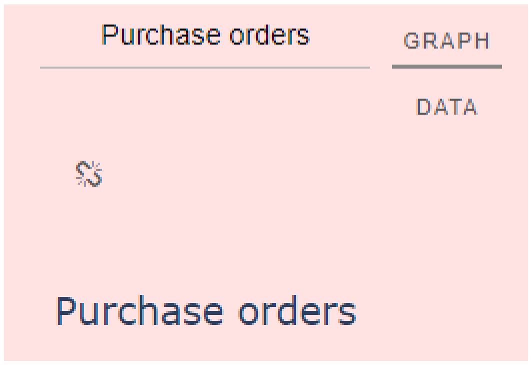
3.26 Tile Title

3.26.1 Facts

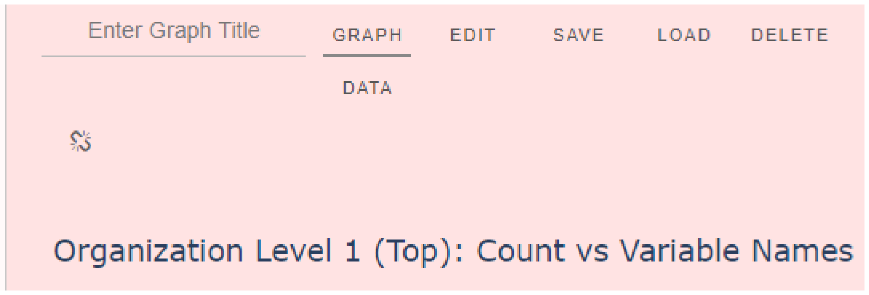
- The input box for tile title is right beside the graph button
- input box is used to pass the title to build the graph with the title given by the user.
- The default takes the hierarchy and the parameters creating the graph

3.26.2 Summary

Given by the user:



Default Title:



3.26.3 Other Notes

3.27 Axis Title

3.27.1 Facts

- Axis titles exist on bar, line, scatter, bubble, and box plots. They allow the x-axis and the y-axis titles to be edited in the graph view.
- The edit capability is set in graphs.py on the config value of the graphs component with the following code:

```
edits=dict(annotationPosition=False,
annotationTail=False,annotationText=False,colourbarPo
sition=False,ColorbarTitleText=False,
legendPosition=True, legendText=False,
shapePosition=False, titleText=False,
axisTitleText=True))
```

- Axis titles are saved to the customize menu (graph menu) in hidden inputs. In order to do so, one must catch the “plotly_relayout” event thrown by the React component when an edit or click has been received. To do so two client side callbacks have been made. One to apply a JQUERY listener to the graph wrapper which will run code to a VISDCC Javascript Dash Object, and another callback to catch that event and parse the information from it to the outputs (hidden inputs). This information is stored and saved with the graph information during saving and accessed during loading. (The titles will not be reset) The same callbacks handle legend location in the same way.
- Titles will not reset unless an entirely new graph is made overtop of the graph. Axis measure title changes are caught if the user has a custom title and an alert will show up to the user notifying them of the change.

3.27.2 Summary

Axis titles seem simple at the start but the amount of intricate code to catch events and interface with the Dash React components is quite large. All the user sees though is that they can edit the graph's axis titles by simply clicking on the axis title and typing what they would like.

3.27.3 Other Notes

Plans for a “Reset Titles to Default” button were discussed but have not been implemented as of 2021/08/27

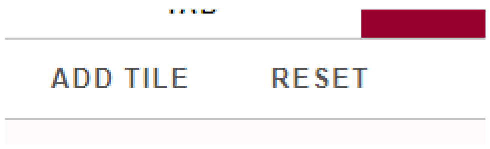
3.28 Add tile (Button)

3.28.1 Facts

- look at how many tiles are built. If the number is less than 4; a new tile will be added.
- The maximum tile number cannot exceed 4
- Add tile button is right beside the reset button for the dashboard
- An outline of the button will appear if hovered over

3.28.2 Summary

The Add Tile button is above the side menu.



3.28.3 Other Notes

Only create one tile at a time as the button is disable until it is finished adding a new tile. callback can be found in the user_interface_callback.py

3.29 Reset (Button)

3.29.1 Facts

- reset the entire dashboard. The side menu is clear and the settings the user has chosen are set back to when you open up the graphical engine.
- The dataset that has been already loaded will not have to be loaded in from the database again as the user did not restart their session.
- deletes all the tiles except for one
- located right next to the Add tile Button
- given two options CANCEL or OK when prompted

3.29.2 Summary

3.29.3 Other Notes

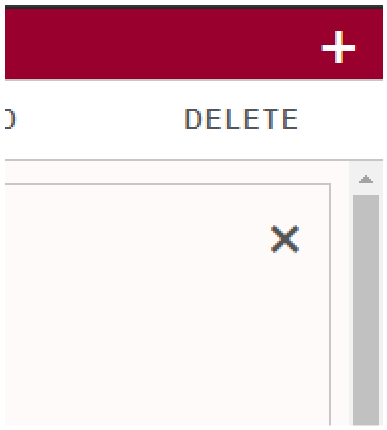
3.30 Add Tab(+) (Button)

3.30.1 Facts

- The maximum tabs that can be add is 12
- button is disabled until the tab is created.
- the add tab button can be found in the top right corner of the application
- An outline of the button will appear if hovered over

3.30.2 Summary

The add tab can be found in the top right corner:



3.30.3 Other Notes

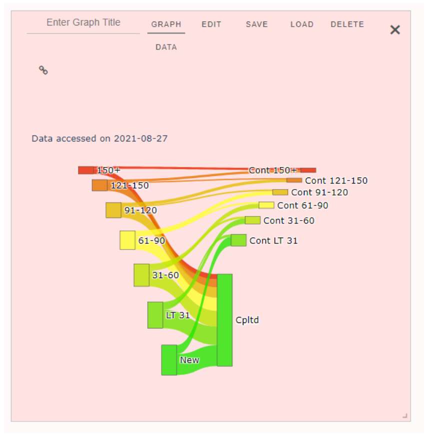
3.31 Close tile (X)

3.31.1 Facts

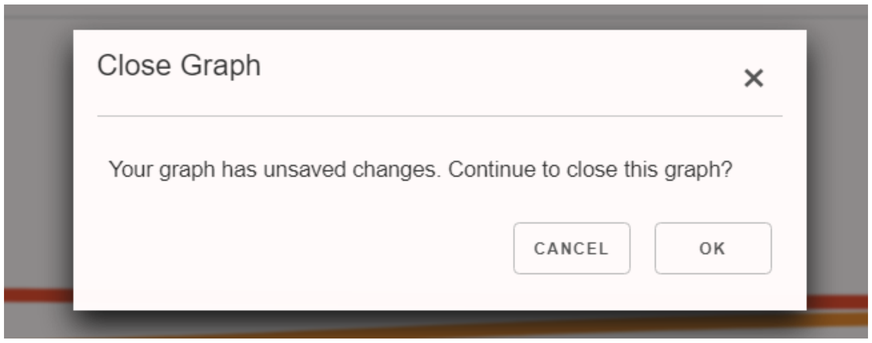
- Will warn the user when closing a tile if it has not been saved.
- Prompt will appear making the user choose if they want to close their graph without saving.
- gray out the background of the float menu
- given two options CANCEL or OK
- close tile button can be found in the top right corner of the tile

3.31.2 Summary

Close tile button top right corner:



prompt for warning:



Your graph has unsaved changes. Continue to close this graph?

3.31.3 Other Notes

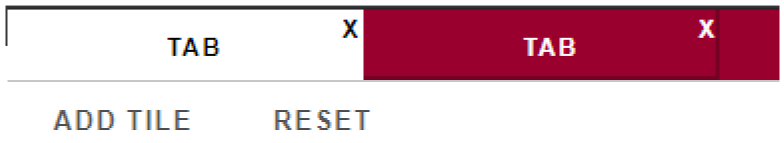
can be found in the saving loading callback.py

3.32 Close tab (X)

3.32.1 Facts

- close tab button can be found in the top right corner of the tab
- once the user click the close the tab is delete if the one selected is deleted the next tile will be selected

3.32.2 Summary



3.32.3 Other Notes

callback can be found in the user interface callback.py

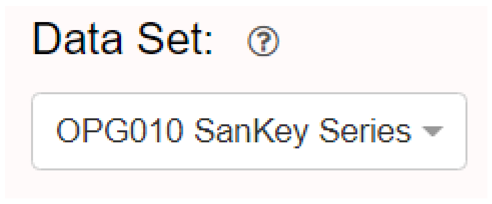
3.33 Help Info

3.33.1 Facts

- Gives the user some information on how to use the feature
- On the right side of the feature
- The Help info is a tooltip and is code in the /assests/ AAA-stylesheet1.css
- the label for the tooltips are in OP_Ref.sql so it can be changed to reflect what info the user might require

3.33.2 Summary

The help info tooltip looks like:



The contents of the help info can be modified in. They are found next to complex features.

3.33.3 Other Notes

tooltips can be found in the layout.py

labels can be found in OP_Ref.sql

3.34 Sessions

3.34.1 Facts

- See 3.20 Server Side Sessions and 3.3 Dash Storage for more

3.34.2 Summary

3.34.3 Other Notes

3.35 Legend

3.27.1 Facts

- Legends exist on bar, line, scatter, bubble, and box plots.
- Legends are to be sorted in alphabetical order.
- The edit capability is set in graphs.py on the config value of the graphs component with the following code:

  ```
  edits=dict(annotationPosition=False,
  annotationTail=False,annotationText=False,colourbarPo
  sition=False,ColorbarTitleText=False,
  legendPosition=True, legendText=False,
  shapePosition=False, titleText=False,
  axisTitleText=True))
  ```

- Legend positions (x and y) are saved to the customize menu (graph menu) in hidden inputs. In order to do so, one must catch the "plotly_relayout" event thrown by the React component when an edit or click has been received. To do so two client side callbacks have been made. One to apply a JQUERY listener to the graph wrapper which will run code to a VISDCC Javascript Dash Object, and another callback to catch that event and parse the information from it to the outputs (hidden inputs). This information is stored and saved with the graph information during saving and accessed during loading. (The titles will not be reset) The same callbacks handle Axis Titles in the same way.
- Legend position will not reset unless an entirely new graph is made overtop of the graph.

3.27.2 Summary

The legend position is extremely similar to that of the Axis Titles. See **3.27 Axis Title** for more information

3.27.3 Other Notes

Part 4

FAQ / Other

Here are some frequently asked questions regarding the GDB along with additional misc. information.

4.1 **Question:** How does asset loading work in Dash and why do we have a static folder and an assets folder?

Answer: Asset loading works as follows:

There are a few things to keep in mind when including assets automatically:

1 - The following file types will automatically be included:

A - CSS files suffixed with .css

B - JavaScript files suffixed with .js

C - A single file named favicon.ico (the page tab's icon)

2 - Dash will include the files in alphanumerical order by filename. So, we recommend prefixing your filenames with numbers if you need to ensure their order (e.g. 10_typography.css, 20_header.css)

3 - You can ignore certain files in your assets folder with a regex filter using app = dash.Dash(assets_ignore='.*ignored.*'). This will prevent Dash from loading files which contain the above pattern.

4 - If you want to include CSS from a remote URL: If you duplicate the file structure of your local assets folder to a folder hosted externally to your Dash app, you can use assets_external_path='http://your-external-assets-folder-url' in the Dash constructor to load the files from there instead of locally. Dash will index your local assets folder to find all of your assets, map their relative path onto assets_external_path and then request the resources from there. app.scripts.config.serve_locally = False must also be set in order for this to work.

5 - Your custom CSS will be included after the Dash component CSS

6 - It is recommended to add __name__ to the dash init to ensure the resources in the assets folder are loaded, eg: app = dash.Dash(__name__, meta_tags=[...]). When you run your application through some other command line (like the flask command or gunicorn/waitress), the __main__ module will no longer be located where app.py is. By explicitly setting __name__, Dash will be able to locate the relative assets folder correctly.

As for why we have a static folder and an assets folder is that if we stored the BBB - french stylesheet1.css in the assets folder the css would be loaded on build of the Dash instance. We want to make it based on whether or not the user has a french sessionID or bilingual flag so we need to store it elsewhere. The static folder is where we would store client specific css to be loaded the same way as the french css.

4.2 Questions: What is dash_responsive_grid_layout-ogma.tar.gz and what does it do?

Answer: That particular file is the package for the import line: import dash_responsive_grid_layout as drgl. It is a custom component developed by plotly that was forgotten for about 2 years and has been modified to remove some buggy code that was left in. To remove JS code from this package modify the bundle.js file.

4.3 Project Info:

The event # for this project is 12976.

Github Repo Link

https://github.com/MiniTitanGett/OGMA2021_Dashboard

Coding Information

To access the project use the following link while the flask server is running: (http://127.0.0.1:8080/python/dashboard/?sessionsID=ID#)

ID# is the session ID created in OP_Curr_Sessions.sql

To run the flask server run the wsgi.py file

AS OF [2021-05-10] THE WORKING DATASETS ARE:

- 'OPG001_2016-17_Week_v3.csv'
- 'OPG010 Sankey Data.xlsx'
- 'OPG001 Graph Data.xlsx'

Example .env File

```
SECRET_KEY='123456789123456789'

CONNECTION_STRING='DRIVER={SQL Server Native Client
11.0};SERVER=localhost;DATABASE=OGMA_Test;Trusted_Conne
ction=yes'

# optional settings (will default if not specified -
see config.py)

BASE_PATHNAME='/python/'

LOG_LEVEL='DEBUG'

LOG_FILE='C:\\Users\\derek\\work\\OGMA2021_Dashboard\\l
og\\python.log'

DATA_SETS = '["OPG001.sql","OPG010.sql"]'

SESSIONLESS=FALSE
```

To Build Your Own Local Database:

1. Download MSSQL and SSMS using these links.
   a. For MSSQL download the Free Developer Version
   b. Use defaults

    c. Go through the steps until you reach the final screen of the MSSQL download and make note your connection_string. This will be used in your .env file to connect to your SQL server

2. Once both are installed and configured, you will want to go to the left hand side and right click on Databases. Click New Database…
3. Write a name such as OGMA_Test and press OK
4. Right click your database and go down to Tasks -> Import Data
5. Click Next and change the Data source to Microsoft Excel
6. Change the Excel file path to the excel file you would like to upload
7. Click Next and set your destination to SQL Server Native Client 11.0
8. Select "Copy data from one or more tables of views"
9. Next -> Run Immediately -> Finish
10. Feel free to right click and rename your table to OPG001, etc.
11. If you encounter any more issues refer to the following links:
    a. https://docs.microsoft.com/en-us/sql/relational-databases/import-export/import-data-from-excel-to-sql?view=sql-server-ver15#wiz
    b. https://stackoverflow.com/questions/9943065/the-microsoft-ace-oledb-12-0-provider-is-not-registered-on-the-local-machine
    c. https://stackoverflow.com/questions/37902257/sql-server-error-connection-string-property-has-not-been-initialized

Snyk Advisor

https://snyk.io/advisor/python/

Rates python packages based on popularity, maintenance, security and community.

Testing Criteria

1. Tile saving and loading
2. Dashboard saving and loading
    1. saving and loading default layout
    2. saving and loading moving around the tile

3. Adding tiles
    1. with no dataset selected
    2. with a dataset
4. Resetting tiles
5. Deleting tiles
6. Adding dashboard tabs
7. Deleting dashboard tabs (delete the tab you are on as well as another)
8. Swapping between dashboard tabs
9. Loading in data sets
    1. loading in data set when a tile has an incompatible graph type selected
    2. loading in data set when multiple tiles has an incompatible graph types selected
    3. loading in a different data set with a compatible graph type (table is our only option as of now)
    4. Opposite order of loading data sets
    5. Repeat with loaded in data set
    6. Load with select range option for datepicker
10. Hierarchy picker
    1. level filter
    2. specific item
11. Date picker
    1. All-Time
    2. Last
    3. Select range
12. Linking tile back to master
    1. compatible graph type
    2. incompatible graph type
13. Graph edit selections
14. Ensure prompts/pop outs work
15. miss match of the axes labels
16. resize able graphs

Installing Requirements

At the moment visdcc and dash_responsive_grid_layout aren't in the requirements. Install them as follows:

1. **visdcc** In venv terminal type this: *pip install visdcc*
2. **dash_responsive_grid_layout** In venv terminal type this: *pip install dash_responsive_grid_layout-ogma.tar.gz*
3.

If you get a **ModuleNotFoundError** like the one **below**. The way to overcome this error is to pip install the modules on to your physical machine before Pycharm will recognizes the module.

```
[2021-08-09 14:17:11,520] DEBUG in config.py
(fn:<module> ln:118): Configuration complete.

[2021-08-09 14:17:13,241] ERROR in config.py
(fn:write ln:74): Traceback (most recent call
last):

[2021-08-09 14:17:13,241] ERROR in config.py
(fn:write ln:74):   File "<frozen
importlib._bootstrap>", line 1007, in
_find_and_load

[2021-08-09 14:17:13,242] ERROR in config.py
(fn:write ln:74):   File "<frozen
importlib._bootstrap>", line 986, in
_find_and_load_unlocked

[2021-08-09 14:17:13,242] ERROR in config.py
(fn:write ln:74):   File "<frozen
importlib._bootstrap>", line 680, in _load_unlocked

[2021-08-09 14:17:13,243] ERROR in config.py
(fn:write ln:74):   File "<frozen
importlib._bootstrap_external>", line 790, in
exec_module

[2021-08-09 14:17:13,243] ERROR in config.py
(fn:write ln:74):   File "<frozen
importlib._bootstrap>", line 228, in
_call_with_frames_removed

[2021-08-09 14:17:13,244] ERROR in config.py
(fn:write ln:74):   File
"C:\Users\derek\work\OGMA2021_Dashboard\apps\dashbo
ard\app.py", line 11, in <module>

[2021-08-09 14:17:13,244] ERROR in config.py
(fn:write ln:74): from apps.dashboard.layouts
import get_layout

[2021-08-09 14:17:13,244] ERROR in config.py
(fn:write ln:74):   File
"C:\Users\derek\work\OGMA2021_Dashboard\apps\dashbo
ard\layouts.py", line 13, in <module>

[2021-08-09 14:17:13,245] ERROR in config.py
(fn:write ln:74): import visdcc as visdcc
```

```
[2021-08-09 14:17:13,245] ERROR in config.py
(fn:write ln:74): ModuleNotFoundError
```

```
[2021-08-09 14:17:13,245] ERROR in config.py
(fn:write ln:74): :
```

```
[2021-08-09 14:17:13,245] ERROR in config.py
(fn:write ln:74): No module named 'visdcc'
```

Warning about not finding reference of these modules will appear in layouts.py, but will not effect how the program will run.

```
Cannot find reference 'ResponsiveGridLayout' in '__init__.py'
Cannot find reference 'Run_js' in '__init__.py'
```

4.4 grid_layout_Info:

GridLayout Info Table

description
displayName
methods

name	docblock	modifiers	params	returns	description
onLayoutChange	Callback for the onLayoutChange		name layout		Callback for the onLayoutChange

props

id

type name | string	
required	false
description	The ID used to identify the component in Dash callbacks

children

type name | node	
required	false
description	A list of renderable child elements, that will be placed inside the grid

rowHeight

type name | number	
required	false

description	The height, in pixels of a row in the grid

cols

type name: number	
required	false
description	The number of columns to display on the grid

width

type name: number	
required	false
description	The width, in pixels, of the grid

autoSize

type name: bool	
required	false

description	If true, containers will automatically resize to fit the content

draggableCancel

type name: string	
required	false
description	CSS selector for tags that will not be draggable. Requires a leading '.'

draggableHandle

type name: string	
required	false
description	CSS selector for tags that will act as the draggable handle. Requires a leading '.'

verticalCompact

type name: bool	
required	false
description	If true, the layout will compact vertically

compactType

type name value	
required	false
description	Compaction type. One of 'vertical' and 'horizontal'

value	computed
'vertical'	false
'horizontal'	false

layout

type

name

value

name	sh
value	

x

name	number
required	true

y

name	number
required	true

w

name	number
required	true

h

name	number

required	true

i

name	custom
raw	PropTypes.Number
required	false

required	false
description	Array of objects with the format: { x: number, y: number, w: number, h: number } If custom keys are used, then an optional \`i\` parameter can be added that matches the key `x` = the x position of the object with key `i` `y` = the y

	position of the object with key `i` `w` = the width of the object with key `i` in terms of columns `h` = the height of the object with key `i` in terms of rows `i` = the key of the object, must be a string

margin

type

name	arrayOf
value | name | number |	

required	false
description	Margin between items [x, y] in px

containerPadding

type name: arrayOf value: (name: number)	
required	false
description	Padding inside the container [x, y] in px

isDraggable

type name: bool	
required	false
description	Elements can be dragged

isResizable

type name: bool	
required	false

description	Elements can be resized

useCSSTransforms

type name: bool	
required	false
description	Use CSS transforms instead of Position. Improves performance if switched on

preventCollision

type name: bool	
required	false
description	If true, grid items won't change position when being dragged over

onLayoutChange

type name | func	
required	false
description	Callback upon the layout changed @param layout: the layout

onDragStart

type name | func	
required	false
description	Callback when dragging is started

onDrag

type name | func	
required	false
description	Callback upon each drag movement

onDragStop

type name | func	
required	false
description	Callback upon drag completion

onResizeStart

type name | func	
required	false
description	Calls when resize starts

onResize

type name | func	
required	false
description	Calls when resize movement happens

onResizeStop

type name | func	
required	false
description	Calls when resize is complete

setProps

type name | func	
required	false
description	Dash-assigned callback that should be called whenever any of the properties change

Example of setting up a layout

A few things that aren't ever said since there is no documentation on this object forgotten by the Dash devs:

1. The object is called by typing `drgl.ResponsiveGridLayout()` after importing like

below ("pip install dash_responsive_grid_layout")
`import dash_responsive_grid_layout as drgl`
2. An object added to the children of this element is wrapped in a gridItem automatically
3. When creating a ResponsiveGridLayout it looks like one should:
    1. Define an ID
    2. Define the cols
    3. Define the layouts
    4. Define the breakpoints
    5. If using draggable, define the draggableHandle

```
drgl.ResponsiveGridLayout(

    [...],

    id='grid-layout',

    cols=cols,

    layouts=layouts,

    breakpoints=breakpoints,

    draggableHandle='.dragbar'

)
```

    6.
4. Each of these should look similar to this:

```
 layouts = {

    'lg': [

        {'i': 'a', 'x': 0, 'y': 0, 'w': 1, 'h': 2,
'static': False},

        {'i': 'b', 'x': 1, 'y': 0, 'w': 1, 'h': 2,
'minW': 1, 'maxW': 2},

        {'i': 'c', 'x': 2, 'y': 0, 'w': 1, 'h': 2}

    ],

    'sm': [

        {'i': 'a', 'x': 0, 'y': 0, 'w': 1, 'h': 2,
'static': False},

        {'i': 'b', 'x': 1, 'y': 0, 'w': 1, 'h': 2,
'minW': 1, 'maxW': 2},

        {'i': 'c', 'x': 2, 'y': 0, 'w': 1, 'h': 2}
    ]
}
cols = { 'lg': 3, 'sm': 4 }
breakpoints = { 'lg': 1200, 'sm': 460}
```

1.

5. Note how the layout sets out an index for each item, it's default position and size. There are additional vars (static, minW, maxW) further testing needs to be done to see if they actually do anything.
6. If a layout has been defined it will only apply to the children of the ResponsiveGridLayout, adding a new child or removing one will reset the layout. This means if you want to add a tile to a specific area, you must change the children *and* modify the layout.
7. Also see how the layouts have a default for 'lg' (>1200px due to breakpoint) and 'sm' (1200px> size >460) A breakpoint must be defined as a dictionary with its default being None. One can set the layout to a single item ({'lg':{...}}) and that will be fine as long as the breakpoint is set to that item with an arbitrary input

```
layouts = {
    'lg': [
        {'i': '1', 'x': 0, 'y': 0, 'w': 1, 'h': 1},
        {'i': '2', 'x': 1, 'y': 0, 'w': 1, 'h': 1},
        {'i': '3', 'x': 0, 'y': 1, 'w': 1, 'h': 1},
        {'i': '4', 'x': 1, 'y': 1, 'w': 1, 'h': 1}
    ]
}
cols = {'lg': 2} # default is none
breakpoints = {'lg': 1200} # default is none
```

1.

8. The cols or columns are set based on the size and show how many columns the grid will have
9. DraggableHandle must use a `.` in front of its title since it is based on the class of the object.
10. Children of ResponsiveGridLayout must be wrapped in a div that is just a string, no dictionaries (wrap dicts with str())

11. Children of the ResponsiveGridLayout should be wrapped in a div with an id that will match to the layout given to its parent:

```
drgl.ResponsiveGridLayout([
      html.Div(
            [...],
            id='a'),
      html.Div(
            [..],
            id='b'),
      html.Div(
            [...],
            id='c')
      ], ... )
```

1.

To get the json value of the layout once modifed for saving etc.

```
@app.callback(
    Output('data-node', 'children'),
    [Input('grid-layout', 'layouts')])
def myfunc(lay):
    print(lay)
    return json.dumps(lay)
```

Should return something like:

{'lg': [{'w': 1, 'h': 2, 'x': 0, 'y': 0, 'i': 'a', 'moved': False, 'static': False}, {'w': 1, 'h': 1, 'x': 0, 'y': 2, 'i': 'b-div', 'moved': False, 'static': False}], 'sm': [{'w': 1, 'h': 2, 'x': 0, 'y': 0, 'i': 'a', 'moved': False, 'static': False}, {'w': 1, 'h': 1, 'x': 0, 'y': 2, 'i': 'b-div', 'moved': False, 'static': False}]}

Example included in project (Python)

```
import dash_responsive_grid_layout as drgl
import dash
from dash.dependencies import Input, Output
import dash_core_components as dcc
import dash_html_components as html
import json

app = dash.Dash('')

app.scripts.config.serve_locally = True

layouts = {
    'lg': [
        {'i': 'a', 'x': 0, 'y': 0, 'w': 1, 'h': 2,
'static': False},
        {'i': 'b', 'x': 1, 'y': 0, 'w': 1, 'h': 2, 'minW':
1, 'maxW': 2},
        {'i': 'c', 'x': 2, 'y': 0, 'w': 1, 'h': 2}
    ],
    'sm': [
        {'i': 'a', 'x': 0, 'y': 0, 'w': 1, 'h': 2,
'static': False},
        {'i': 'b', 'x': 1, 'y': 0, 'w': 1, 'h': 2, 'minW':
1, 'maxW': 2},
        {'i': 'c', 'x': 2, 'y': 0, 'w': 1, 'h': 2}
    ]
}
```

```
style = { 'borderStyle': 'solid', 'height': '100%'}
cols = { 'lg': 3, 'sm': 4 }
breakpoints = { 'lg': 1200, 'sm': 460}



# app.layout = html.Div([drgl.GridLayout([
#     html.Div('a', key='a', style=style),
#     html.Div('b', key='b', style=style),
#     html.Div('c', key='c', style=style),
# ], layout=layout, rowHeight=30, width=1200
# )])



# app.layout = html.Div([drgl.ResponsiveGridLayout([
#     html.Div('a', key='a', style=style),
#     html.Div('b', key='b', style=style),
#     html.Div('c', key='c', style=style),
# ], cols=cols, layout=layout
# )])

app.layout = html.Div([
    drgl.ResponsiveGridLayout(
        [
            dcc.Graph(
                id='a',
                figure={
                    'data': [
                        {
                            'x': [1, 2, 3, 4],
                            'y': [4, 1, 3, 5],
                            'text': ['a', 'b', 'c', 'd'],
                            'customdata': ['c.a', 'c.b',
'c.c', 'c.d'],
                            'name': 'Trace 1',
                            'mode': 'markers',
                            'marker': {'size': 12}
                            },
                        {
                            'x': [1, 2, 3, 4],
                            'y': [9, 4, 1, 4],
                            'text': ['w', 'x', 'y', 'z'],
                            'customdata': ['c.w', 'c.x',
'c.y', 'c.z'],
                            'name': 'Trace 2',
                            'mode': 'markers',
                            'marker': {'size': 12}
                            }
                        ]
                    },
                config={
                    'autosizable': True
                    },
                style=style
            ),
            html.Div([
                html.H1('HELLO WORLD', id='b-h1',
className='dragbar'),
                html.Div([
                dcc.Graph(
                    id='b-graph',
                    figure={
                        'data':
[
                            {'x': [1, 2, 3], 'y': [4, 1,
2], 'type': 'bar', 'name': 'SF'},
                            {'x': [1, 2, 3], 'y': [2, 4,
5], 'type': 'bar', 'name': u'Montréal'},
                            ],
                        'layout': {
                            'title': 'Dash Data
Visualization'
```

```
                                }
                            },
                        config={
                            'autosizable': True,
                            'doubleClick': 'autosize',
                            'frameMargins': 0,
                            },
                    )], style={'height': '50%'}
                    )
                ], key='b', id='b-div', style=style)
            ],
            id='grid-layout',
            cols=cols,
            layouts=layouts,
            breakpoints=breakpoints,
            draggableHandle='.dragbar'
    ), #drgl.GridLayout
    html.Div('hi', id='data-node')
]) #html.Div

@app.callback(
    Output('data-node', 'children'),
    [Input('grid-layout', 'layouts')])
def myfunc(lay):
    print(lay)
    return json.dumps(lay)

if __name__ == '__main__':
    app.run_server(debug=True)
```

Link to GitHub

Base:

AlgorithmHub/dash-responsive-grid-layout:
dash-responsive-grid-layout

Edited:

https://github.com/MiniTitanGett/dash-responsive-grid-layout

4.5 Old Tables with Info (2021/08/26):

TO DO: Orange = OPG011, Yellow = MAIN

TASK	PRIORITY	DIFFICULTY	STATUS	DUE
Documentation	HIGH	HARD	WIP Kevin	Aug. 27
Bug Testing	HIGH	HARD	WIP Derrick	Aug. 27
Speed up Program - Use wrapping to reduce the size of callback chains to improve init load times	HIGH	MEDIUM		Aug. 27
~~Axis Fix~~	~~HIGH~~	~~HARD~~	~~COMPLETED (Aug 09)~~	~~Aug. 20~~
Atomized Data - (atomize new hierarchy, test new filtering)	HIGH	HARD	WIP Elija	Aug. 20
Multiple Value and Measure Types	HIGH	HARD	**MUST BE DONE AFTER**	Aug. 27

			OPG011 AGG.	
Adding Links to the Table	HIGH	EASY (Should be written already)	~~Completed~~	Aug. 27
Edit OPG011 to build variable hierarchy	HIGH	EASY		Aug. 27
~~Clientside.js warning clean up~~	~~MEDIUM~~	~~LOW~~	~~Completed~~	~~AUG. 27~~
~~Re-sizable Graphs~~	~~MEDIUM~~	~~HARD~~	~~Completed~~	~~Aug. 20~~
Code Cleanup	MEDIUM	MEDIUM		Aug. 27
~~Adding a dataset that is of the same type but different information (OPG001 PT.2)~~	~~MEDIUM~~	~~MEDIUM~~	~~Completed (Aug 12)~~	~~Aug. 13~~
~~Bubble Time Option~~	~~MEDIUM~~	~~MEDIUM/EASY~~	~~Completed~~	~~Aug. 13~~
Handle Multiple Hierarchies	DEFERRED	HARD	DEFERRED	N.A.
Terrys New Graph Type	DEFERRED	UKNOWN	DEFERRED	N.A
More than four tiles (no limit to graphs?) - Somehow remove loops?	LOW	HARD+		N.A.
Icons instead of tiles	LOW	LOW		N.A.

Client CSS	LOW	MEDIUM		N.A.

BUG LIST:

BUG	HOW TO REPRODUCE	STATUS
When deleting a unlinked graph the datamenu and graph menu are out of sync	Load in a dataset -> (switch to another dataset and unlink -> add tile)x2 -> close first graph unlinked graph	Complete
Click close data sidemenu on startup, throws error	Start program > access > click x on sidemenu	Complete
clicking save Button, Infinite loading screen for that tile	load in a data set > switch to another dataset > unlink graph option > add tile > save new tile	Complete
Swapping tabs resetting menus to defaults		Complete
Err Throw when saving tile w/ no data	Start program, give tile a title, save title	Complete
data fitting error when linked	Load in a dataset OPG001-> then switch to OPG001C unlink > create new tile > switch to line graph > switch to specific and select LADWP > open edit menu > datafiting not displaying	

infinite loading screen after loading in OPG001C after sankey incosistent behaviour as it comes and goes has to do with when the callback fires	load in dataset OPG010 > swap to OPG001C	Complete

4.6 Current Bug Table After OPG011 Merge (2021/08/27):

For the commit before the merge of OPG011 where these bugs weren't a problem the commit ID is: `1 parent b507c62 commit 53a4427ef634eda96cc5f643988bbf72830e5ddb`

BUG	**HOW TO REPRODUCE**	**STATUS**
Infinite Loading Screen	Load OPG001, add a new tile, unlink new tile. load OPG011 into that tile **OR** swap the order of the datasets (seems the unlinked prompt is to blame)	
No graph being displayed	Load OPG001, add a new tile, unlink new tile. load OPG011 into that tile, the graph displays no data **OR** swap the order of the datasets (seems the unlink is to blame)	
Hover data on animated bar graph group by variable names in OPG011 showing H0=xyz instead of bolding xyz	Load OPG011, Bar graph, animate over time, group by variable name	
Box Plot on OPG011 is not filtering by graphed variables	Load OPG011, Box plot	Fixed
Bubble Plot on OPG011 not working	Crashes with err on build. Load OPG011, Set to bubble	

Lines of best fit when multiple traces are on the graph (through graphed variables) are indeciferable from each other	Load dataset, Line graph, specific filtering, datafitting on, see multiple traces no differentiation	
data fitting showing up on show all	After the above bug loading OPG011, from here on out OPG011 will show data fitting, might be linked with OPG011 as OPG001 hasn't been tested	
empty specific hierarchy isn't returning empty graphs		Fixed
loading a saved OPG011 graph will not load a graph	save a graph made with OPG011. Reset, immediately attempt to load graph to tile, any type of loading it in will fail - this means dashboard loading will also be affected	Fixed
sidemenu isn't being updated as tiles are unlinked	load dataset (OPG001), add tile, unlink tile, data menu button (could be the speed of which to click the data menu)	
graph isn't updating after being relinked to main if the one of the graphs either the main dataset or not main dataset is OPG011	load dataset (OPG001), add tile, unlink tile, load dataset into unlinked tile (OPG011), relink to main dataset	