Creating custom visualizations

Document author mikko.koskinen@tyks.fi

Overview

Building custom visualizations is needed when none of the platforms available tools are able to create the required visualization. Building custom visualizations can be done using a variety of librarys, but when actual custom work is required it is usually beneficial to use a fairly low level libraries such as d3.js in javascript.

Links

- d3.js data visualization library
- Python flask backend
- pentaho CDE
- Building custom visualizations in Pentaho

Workflow

It is a good practice to start from simple building blocks and add more complexity as you advance. This workflow takes the visualization design as a starting point

1 Draw the visualization

This is the most important part of the workflow. This determines the design of your visualization.

It is a good practice to write your visualization code straight into a minimal html template. That way is is easy to debug in the browser. You can launch a simple python webserver from the folder containing you visualization html file with `$python3 -m http.server` and navigating to localhost, port 8000 `127.0.0.1:8000` on your browser.

When building your visualization code do not forget to save the changes into the code and make a complete refresh of the page with `ctrl + shift + r`. The browser cache has a tendency to cause problems otherwise.

Make your data entry as easy as possible at this stage by using csv, json etc. files locally. This data can be real or mock data as long as the data format and complexity are comparable. Concentrate only on getting the visualization as you want it to be.

If making custom visualizations is a recurring practice it saves time to build some boilerplate templates containing the html atleast. This saves time when lauching new projects.

2 Add backend

If the data is updated from the server during the use of the visualization, at some point a backend API should be implemented.

At this time it is usually good to have a simple database copy or data files that the backend has access to.

The aim is to fine tune the functionality of the frontend. A good method is to build a minimal backend server with python flask and create the required API endpoints there. Both the webserver and the backend server can still be run locally (and data stored locally).

At this stage you can fine tune the calls to the backend and the following functionality of the visualization. You will also determine exactly the responses that you need from the backend APIs. You should not need to do any data manipulation in the frontend. All that should be done in the backend.

If mock data is still used by the backend server, it can now be compared to the real data and the specs for the backend (and source system) processing become clear. Do not change the frontend code to suit the backend. The frontend visualization code should be completely separate and only dictate the format and shape of the data it receives from the backend. This will also retain modularity as either the backend solution or the visualization can be changed independently as long as the API specs remain the same.

3 Connect to the framework

After the API calls have been fully developed and tested as well as the visualization code, it will probably be deployed into a larger system. This could be a web application built with a Javascript (ReactJS, Angular, etc.), or any other system used to display the visualization. It does not really matter, whether the backend uses relation-object transfer, direct sql queries or some other, more complicated, data pipeline. That should not affect your visualization (everything should work as long as the API specs are correct).

At this stage the visualization code should be moved into the deployment repo and tested still. If there are some minor changes to be made into the code considering data import (best practice is to not have any) they should be implemented here. If a testing mode is available it should be used to see everything works in the context. When everything is fine the visualization can be put into production.

4 Custom visualizations in Pentaho

Pentaho offers a way to create custom javascript visualization to their dashboard application in a sandbox environment. This differs somewhat from other frameworks we have used since data is retrieved directly in pentaho.

A good starting point is still to follow at least step one locally. Once the visualization is, done clone the repo:

https://github.com/pentaho/pentaho-engineering-samples

And follow the instructions in this tutorial:

http://pentaho.github.io/pentaho-platform-plugin-common-ui/platform/visual/samples/
bar-d3-sandbox/step1-environment-preparation

One of the biggest hurdles in transferring the visualization into pentaho is the fact that data comes from pentahos own relation-object transformation so you'll have to make a model file and and introduce the model into the visualization (this is the data import into the visualization). Anyone doing the visualizations should work closely with the ETL/data managers to ensure that the data is available in the right form through the pentaho platform.

An overview of the pentho visualization API is found in:

http://pentaho.github.io/pentaho-platform-plugin-common-ui/platform/visual/

When everythiong works in the sandbox environment and you proceed to deployment the instruction of recursive find and replace is an overkill. You'll need to change the name of the visualization in three different pom.xml files that are easily found and edited manually. Remember to be carefull with the visualization name and version number. Otherwise just follow along the instructions and everything should be fine.