Data Dashboard - Project Spec

Concept

The goal of the “Data Dashboard” is to make an accessible web application using Fluid’s Infusion framework that supports the exploration of data sources and the rapid prototyping of visualizations and sonifications of data.

It is becoming evermore critical that internet users are able to be literate in the use and comprehension of data. To meet this growing need, a panoply of tools has been created meant for the developer or data science; whereas user-focused tools have mostly been limited to expensive enterprise softwares. Moreover, as the process of creating data representations expands, so does the burning need for accessible sharing formats such as sonification. Motivated to create something end-user focused, but also simple, free, and accessible, the Data Dashboard is a first-step goal.

The project will entail a visualization/sonification canvas that uses pre-structured representations for data such as pie charts, bar charts, line graphs, and DAGs. The user will be able to establish a session with the Data Dashboard by loading a data source through either a text box, a file uploader, or a stable URL. Once the data source is loaded, a “fields” box will be populated, establishing a browsable filter that displays the variable names available for projecting to a graph.

The user will be *required* to select a graph **structure** prior to selecting any “fields”. Since graph dimensionality is yoked to a notion of graph representation, we will enforce this order of operations to be sure the data being selected is compatible with the representation; thus, the application having a valid, renderable state. An example of this would be a user selecting a line graph, then selecting two fields, which would each appear in the Representation Component with a selection box to choose *x* or *y* axis for that field.

The Data Dashboard will also allow for some simple customization such as colors, scales, and projections. Using the above line graph example, this would allow a user to make their line graph **red**, with the x-axis showing only **[0, 100]** using a **logarithmic** projection. These simple customizations have the opportunity to be expanded upon successful completion of the first version and user feedback.

Finally, the project will allow for a simple export to common filetypes such as SVG or PNG. The long-term goal will be to allow a session to also connect to a GitHub profile which then live commits your visualization/sonification to a repo. This will make spreading accessible sonifications easier and put more ownership into the users hands.

Dependencies

The creation of this project requires several JavaScript dependencies:

- Infusion
- JQuery
- D3
- Flocking

CSS Libraries are flexible, but I prefer BassCSS. If Infusion has a preferred CSS libary, please send over the relevant links ASAP.

The final product will be able to be built and minified for hosting using a single Grunt task. Once the core functionality has been established, tests may be added to our Grunt file.

Server-side dependencies will not be included in this spec and will be dependent on the final habitat of the web application, after initial development.

Application Structure

As a DOM-rendered, client-side application using Infusion, the structure of the application will be a relationship of components, each responsible for different parts of a hierarchical data model and set of rendering tasks utilized to achieve functional application states such as **establishing a session**, **rendering a graph**, and **rendering custom features**.

Component Hierarchy

- Dashboard Component
  - Representation Component
    - Graph Component
    - Mappings Selector
    - Features Selector
    - Projections Selector
  - Fields Component
  - DataSource Component

Dashboard Component

This will be the parent component of the entire application. The goal is to encapsulate all functionality under a single dashboard component so that, down the line, multiple dashboards could be a mounted to a single DOM to allow for side-by-side graph comparisons.

Model:

The data model of the Dashboard will make available:

- Raw, Parsed Data Stream/Source
- Current Representation Selection
- Field Selections

View:

The Dashboard Component's view will solely be the hierarchical layout for the renderable child components.

Representation Component

This will be the parent of the final graph representation and all selectors used for customization.

Model:

The representation component will contain:

- Current graph structure being built (eg, pie chart, bar chart, etc)
- Final data snippets selected for graphing (ie, field selections)
- Relationships between data snippets and graph representation (eg, *field A is bound to x-axis*)
- Current values of all customization options
- Selected projections to trigger correct transformations upon rendering

View:

The representation component will be a parent to the final d3 graph (**Graph Component**), a dropdown selection for axis projections (**Projections Selector**), customizable selectors (**Features Selector**), and data <-> graph binding selectors (**Mappings Selector**).

There will also be a "Represent!" button that takes all your current selections and builds the D3 graph in the **Graph Component**. Eventually, once a graph is built, this component will be responsible for having an export selector and button for saving a the final representation to disk.

Fields Component

The fields component is responsible for displaying a browsable data structure that tells a user what variables are available for graphing, how much data a given variable has, and revealing the structure of the data (eg, a parent-child structure for a DAG vs. time-series events vs. field quantities).

Model:

The fields component is responsible for:

- List of selectable fields from data source
- Amount of data in each field (eg, you may have 10 JSON blobs and only 3 have a field called “time”, thus the fields component should communicate that ‘3 entries are available for the time field.’)
- Which fields are currently selected

View:

This should be a scrollable and clickable box that contains one element per field. The fields are selectable, but should use information about the current representation (held by the Dashboard Component) to determine if the current selection is valid (ie, it should know if too many fields are being selected)

DataSource Component

The datasource component is the where the initial raw data should be added into the dashboard. Binding this component to some data for parsing is what counts as “establishing a session” and is the first chronological state of the application.

Model:

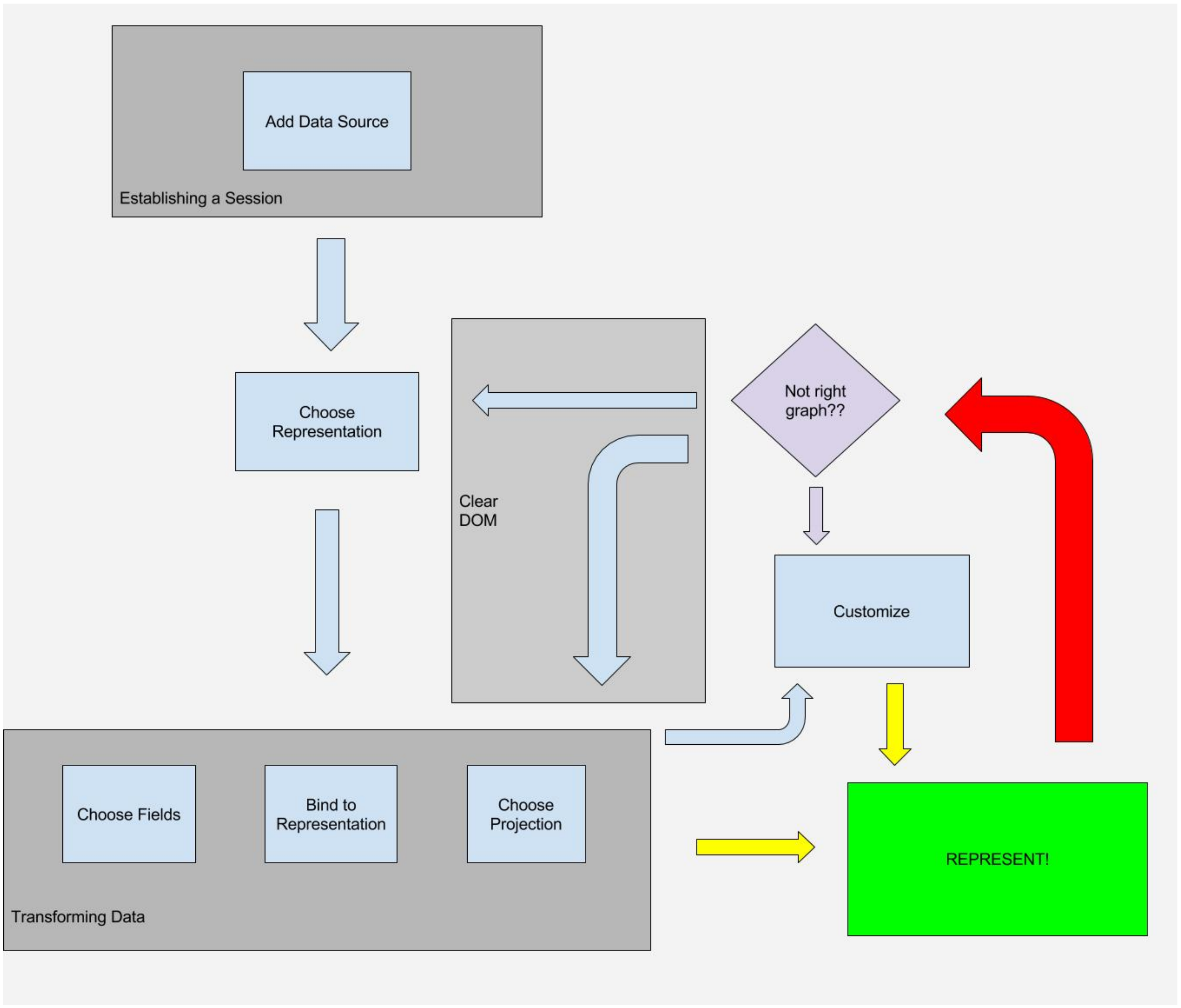
- Data format to be used (informs which parser to use as a transformation to the dashboard)
- Raw Data (ie, file path, text field, stream)
- Boolean of whether current data selection is valid and parseable

View:

The DataSource Component will be a text box with a paired drop down selector. The selector will allow the user to choose what format their data is in and the text box will be available for copy/paste of particular data snippets. This will eventually be expanded to handle a file uploader and/or URL upload.

There will also be a button “Parse” to be clicked when the data is ready to be parsed. The text box should be highlighted in red or green along with a caption box beneath to let the user know if the current data selection is OK to parse.

User Flow


Add Data Source
Establishing a Session
Choose Representation
Clear DOM
Not right graph??
Customize
Choose Fields
Bind to Representation
Choose Projection
Transforming Data
REPRESENT!