IFS Network App Documentation  
  
**Set-up**

To start the latest version of the app  
1. Run 'git fetch' from the terminal

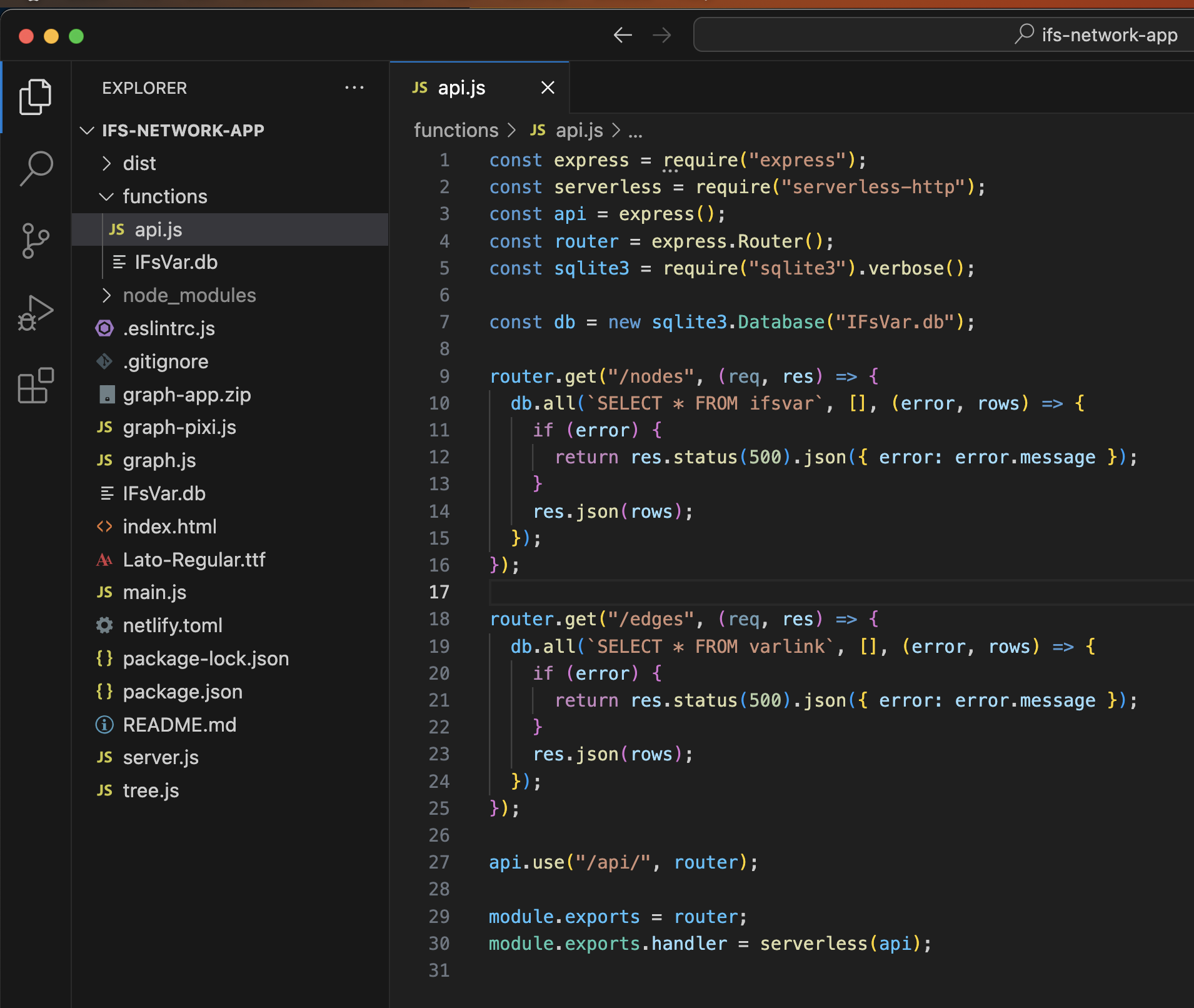
2. npm install

3. npm run autobuild

4. open a new tab in the terminal and run 'node server.js'

Open site at: <http://localhost:8080>  
  
**Technology**

This app uses Node.js and Express.js for the backend API, vanilla JavaScript, PixiJS and D3.js for the UI. There is a need for a local database file to be stored in the functions folder before running the server. API routes are created in **api.js** file to be consumed on the client-side.

**Data  
IFsVar.db can be switched out for newer versions by replacing IFsVar.db with the new file. Change the file name accordingly in api.js.** The graph can only be rendered with a ‘flat’ dataset that only contains Variables and links from Variable to Variable. *The dataset should not contain Submodule/Segment nodes and links from Variable -> Submodule, Submodules -> Submodules etc. Extraction of the Submodule and Segment nodes and links is handled within the graph rendering function.*  




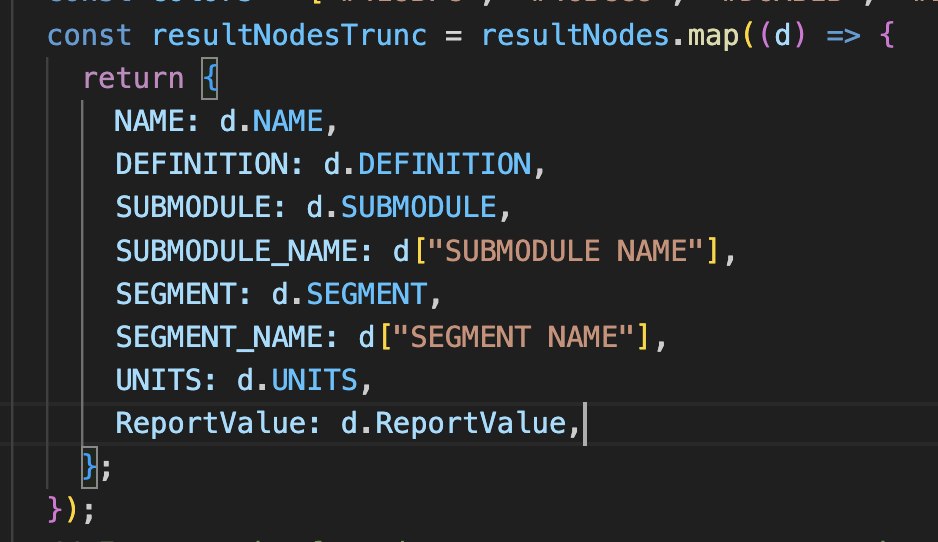


**Data Transformations (main.js)**

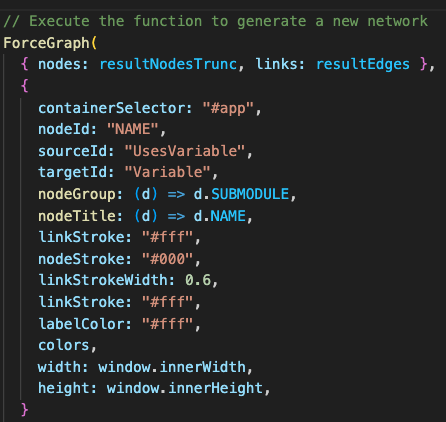
Once the data is retrieved from the API, there is actually little data processing work needed.  
1) I minimized the amount of attributes to show in each node data object to a select important few for performance optimization.

2) I relabeled some attributes to form a single string for easy access within the graph function.

**If key names in the dataset change, they should only be updated here.** There is no need to edit the graph rendering function itself.



**Graph initialization**

The function to execute the graph is called ForceGraph, which accepts many input parameters. Only the parameters within the red box may need to be changed based on the dataset. The rest are style-based parameters and are optional with default settings as explained below.  
  
The existing dataset has this data structure, which is why I specified **nodeId, sourceId and targetId** as such.  
links = [{**UsesVariable**: "FDEM", **Variable**: "AGDEM"}, {**UsesVariable**: "FEDDEM", **Variable**: "AGDEM"}, …]  
nodes = [{NAME: XXX, ….}]

* The container to render the graph in is **#app** because this div with such an ID is specified in index.html.  
  **<div id="app"></div>**
* **Colors can be changed by listing them out in an array and input as a function parameter. Colors are assigned based on the numerical value of submodule ID**

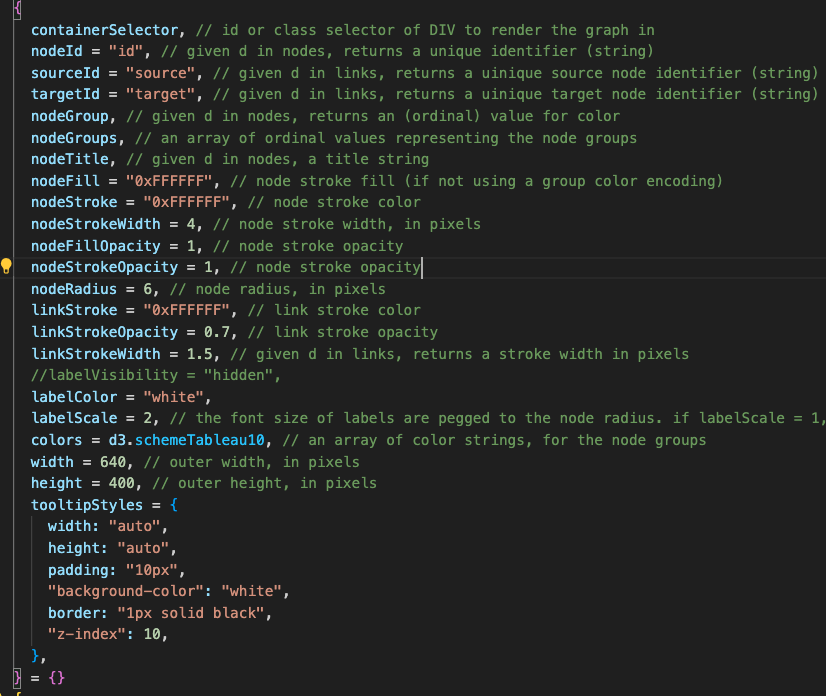
const colors = ["#418BFC", "#46BCC8", "#D6AB1B", "#EB5E68", "#B6BE1C", "#F64D1A", "#BA6DE4", "#EA6BCB", "#B9AAC8", "#F08519"];

To color a Variable based on it’s submodule ID, nodeGroup is given a function (d) => d.SUBMODULE

* To allow dynamic assignment of values to each node, which affects the style of the node, input parameters can be javascript functions, instead of strings.
* The **label font size** are pegged to the radius assigned to each node. if labelScale = 1, the font size is the same number of pixels as the radius.

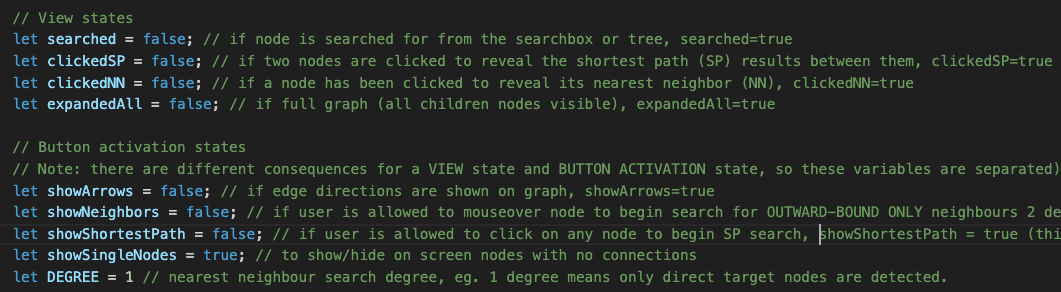
**Graph function parameters (graph-pixi.js)**

The function to generate the graph resides in graph-pixi.js When a graph is initialized, these are the **default parameters**. They can be overwritten if need be only if specified during the ForceGraph function call in **main.js**, as demonstrated above.

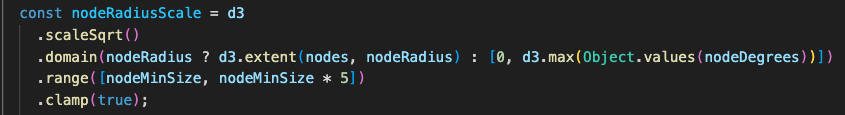


**In-built Graph features (graph-pixi.js)**

These initial states are in-built to the ForceGraph function, and can be changed depending on how you would like the initial state of features to behave. Example, the choice to have a full expanded or collapsed graph is specific with the ‘expandedAll’ variable. The choice to show arrows to indicate direction on page load.



**Node sizes are assigned through a scale**. The smallest node size value is assigned through the ‘nodeMinSize’ variable, and the scale adjusts the radius value based on a property value, which is the node degree (number of incoming connections to a node)

Nodes can also be sized based on other parameters besides node degree, if a nodeRadius accessor is specified on graph initialization.

**To adjust label positioning**, go to Line 550 of graph-pixi.js. Access the nodeData variable to adjust any node/label on a per node basis. By default, labels are positioned beside all nodes.

**Simulation parameters**

The ‘collide’ force prevents nodes from overlapping by creating a radius buffer around each node: https://d3js.org/d3-force/collide

simulation.force(

"collide",

d3

.forceCollide()

.radius((d) => (d.type === "tier1" || d.type === "tier2" ? d.radius \* 2 : d.radius))

.iterations(3)

)

The ‘cluster’ force is custom. It forces nodes to be grouped (ie. positioned together) based on their submodule. The strength value increases the ‘pull’ strength to group nodes together.

simulation.force("cluster", forceCluster().strength(0.45));

The ‘charge’ force repels nodes away from each other. distanceMin is the minimum distance between nodes over which this force is considered. Helps to void an infinitely-strong force if two nodes are exactly coincident.

simulation.force(

"charge",

d3

.forceManyBody()

.strength(expandedAll ? -100 : -150)

.distanceMin(100)

);

*Do not set such a high charge as new nodes added to screen on click of a parent node seem to get pushed further and further away from their original position.*

**Tooltips**

updateTooltip (Line 1386 in graph-pixi.js) is the function to update tooltip content within a DIV.

To specify which key-value pairs within the raw data to render inside the tooltip, update the **TOOLTIP\_KEYS** variable.

**User Interface Theming**

**Import font from Google Font.** Select name of font you want to use from Google font, retrieve the embed code and replace the URL.

<link href="https://fonts.googleapis.com/css?family=Montserrat:400,800&display=swap" rel="stylesheet" />

<link href="https://fonts.googleapis.com/css?family=Lato:300,400,700&display=swap" rel="stylesheet" />

**Using locally stored fonts:**

In the @font-face rule, define a name for the font (MyCustomFont in the example) and the path to the font file. The path should be relative to the CSS file. Then, use the custom font by setting the font-family property to the name defined. Replace 'path-to-your-font-file/font-file.ttf' with the actual path to your font file and 'MyCustomFont' with the name you want to give to your font.

Note that different browsers support different font formats, so provide multiple font formats (e.g., .woff, .woff2, .eot) to ensure compatibility across all browsers. You can do this by adding multiple src descriptors in your @font-face rule. For example:

@font-face {

font-family: 'MyCustomFont';

src: url('path-to-your-font-file/font-file.ttf');

src: url('font-file.eot'); /\* IE9 \*/

src: url('font-file.eot?#iefix') format('embedded-opentype'), /\* IE6-IE8 \*/

url('font-file.woff2') format('woff2'), /\* Super modern browsers \*/

url('font-file.woff') format('woff'), /\* Modern browsers \*/

url('font-file.ttf') format('truetype'); /\* Safari, Android, iOS \*/

}

p {

font-family: 'MyCustomFont';

}

**UI colors.** The existing specs gives a black background with white text

:root {

--main-bg-color: black;

--text-color: white;

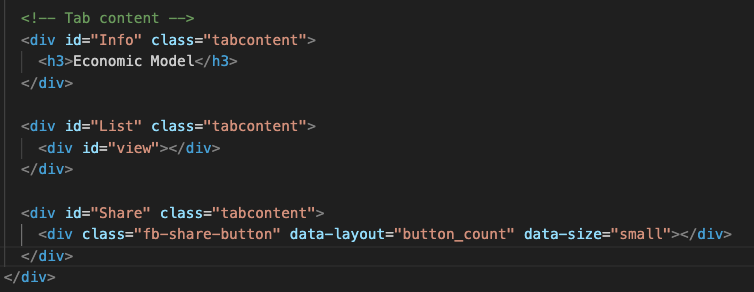
--text-hover-color: #ddd;

}

**Side panel**

**Locate the div with id called ‘Info’ to alter the content displayed in it.**

Example, remove the existing text called ‘Economic Model’ and change it to the desired text.



**Debugging**  
Open Chrome dev tools and check the console.log statement, which prints out the array of node and link objects currently on screen after graph updates.