**CQ Project**

Contents

[**Installations** 2](#_Toc64131777)

[**How to run the application** 4](#_Toc64131778)

[**Project Structure** 6](#_Toc64131779)

[**Development Questions** 17](#_Toc64131780)

[**Solutions** 18](#_Toc64131781)

# **Installations**

**NOTE:**  The installations guide referring to WINDOWS environment.

**Python:**

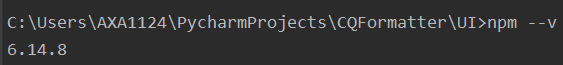
1. You should install python interpreter 3.x and verify the python installed in windows environment variable (to check if that works please open cmd and type “python” – if the python interpreter open – you are set!)

* How to download python interpreter: <https://www.python.org/downloads/>
* How to set python in environment variable: <https://stackoverflow.com/questions/3701646/how-to-add-to-the-pythonpath-in-windows-so-it-finds-my-modules-packages>)

1. Open server folder and run “install.bat”. this batch file will install all the python requirements for the project.

**Angular:**

1. you should download and install npm: (you can download it from here: <https://www.npmjs.com/get-npm> after install open cmd and type “npm --v” you should get the npm version – for example I have this version:

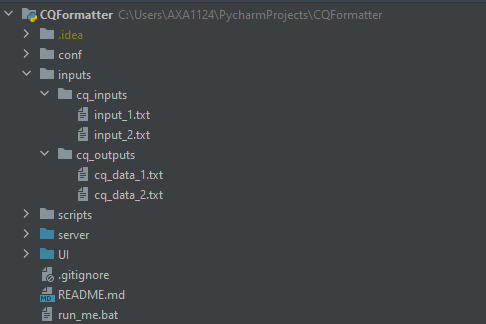
****

1. After installing npm, type “npm i” under the UI folder. It should install all the requirements for the frontend code.

# **How to run the application**

To run the application all you have to do:

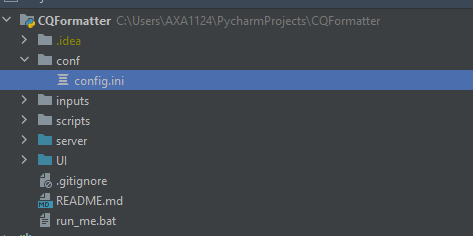
1. Go to the project folder and go to the inputs folder:



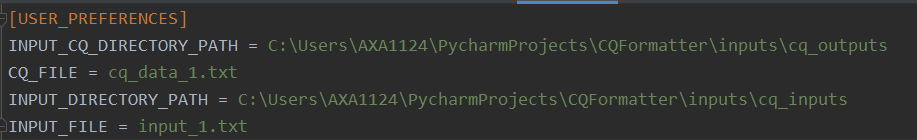
At this folder you have 2 folders: one for CQ inputs and the other one for CQ outputs.

Put your input and output file in those folders.

1. After that, go to the configuration file:

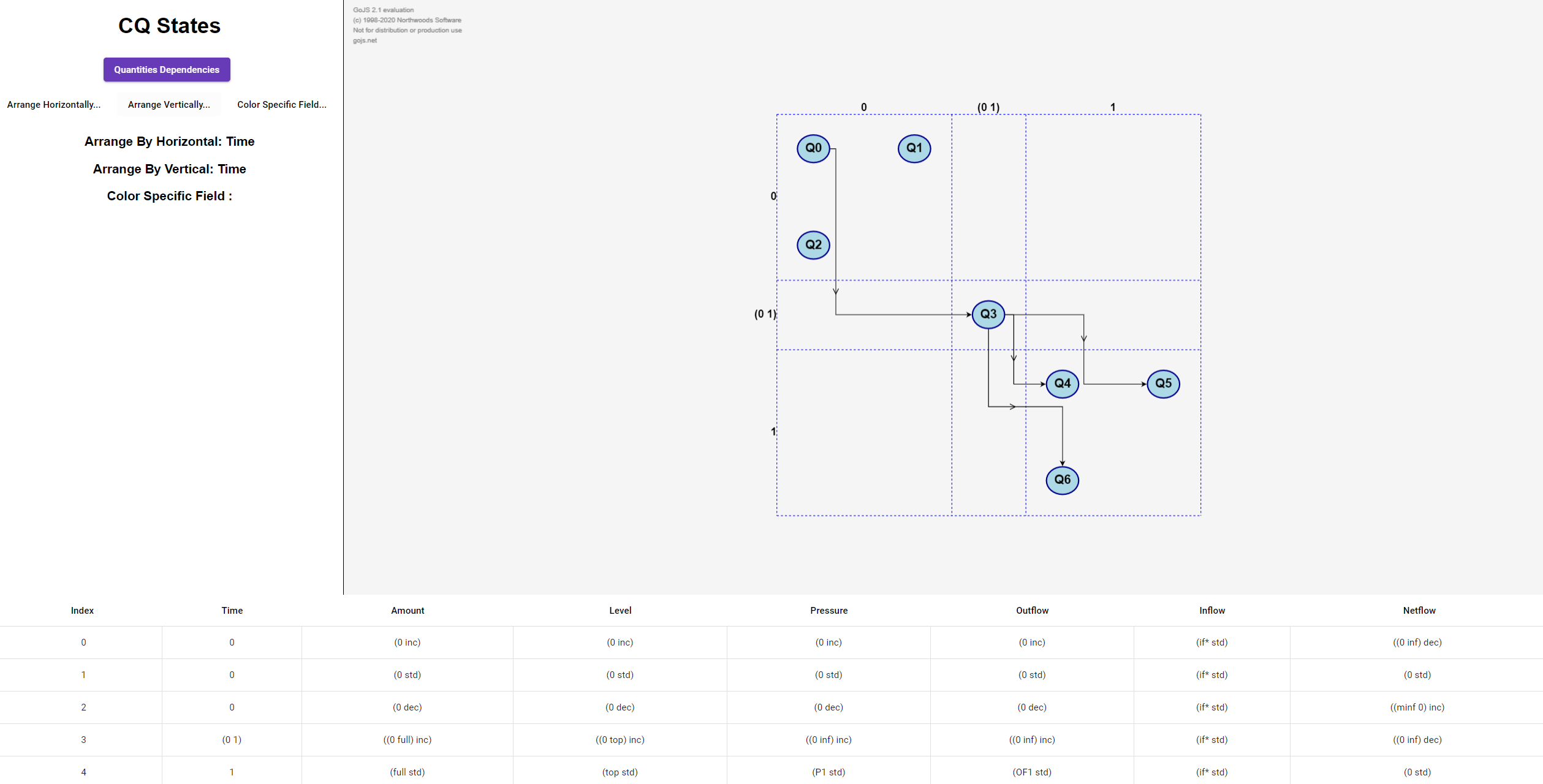


And set the inputs and outputs folder and the files names. For example, I used this configuration file:



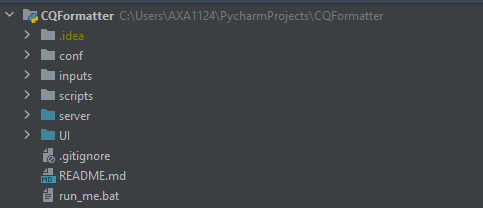
1. THAT’S IT! Run the file “run\_me.bat”. go to: <http://localhost:4200>

And there you go:



# **Project Structure**

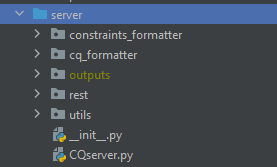
The project contains several folders. Let’s take a look each folder:



1. **Conf** – configuration folder – this folder has one file that called “conf.ini” – please look at the last page.
2. **Inputs** – inputs folder – contains the inputs file for the application.
3. **Scripts** – the “run\_me” script calls to the scripts in this folder. Scripts folder contains 2 scripts: run\_server and run\_client.
4. **Sever** – the server code – written in Python.
5. **UI** – the angular code – written in TypeScript, JavaScript, HTML and CSS.

**Server:**

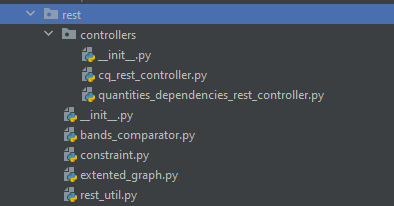
Let’s take a look on the **server** module:



**CQServer.py** – the server of the application. Listening on port 8080.

**Utils** – contains general\_utils functions.

**Rest**:



**Controllers**:

For each screen – home page and dependencies page, each one has a rest controller. The CQServer calls to the controller to get/post the data.

**Bands\_comparator**: sorting all the quantities values according to the magnitudes and directions.

**Constraint**: represent a constraint.

**Extended\_graph**: graph that contains:

* Nodes
* Edges
* User preferences (sorting and coloring)
* Quantities data

**Rest\_util**: for creating data (according to configuration file) and generate colors palette for coloring the nodes.

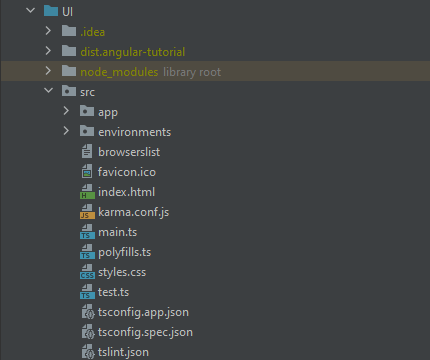
**Constrains\_formatter & cq\_formatter**:

Those modules responsible to format the CQ input & output files.

(NOTE: the cq\_formatter has a main from his own and he can generate a gml file that located in the outputs result. The folder will be created if it wasn’t existing)

**Client:**

Let’s take a look on the **UI** module:

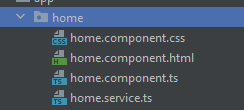


The main folder here is the src/app

As you can see there is 2 folders: home and plot-b.

1. Home is a component for the home page – for the state graph and the state’s table.
2. Plot B is a component for the quantities dependencies.

**Home:**

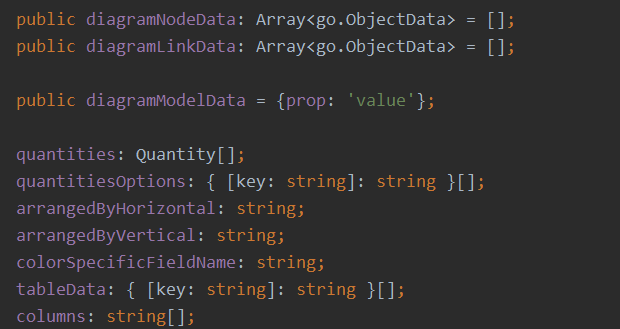


**CSS** – css file for the view adjustment

**HTML – the Html file**

**Component.ts** – the main file! This file represent how the component works! Let’s take a look:

The component fields:

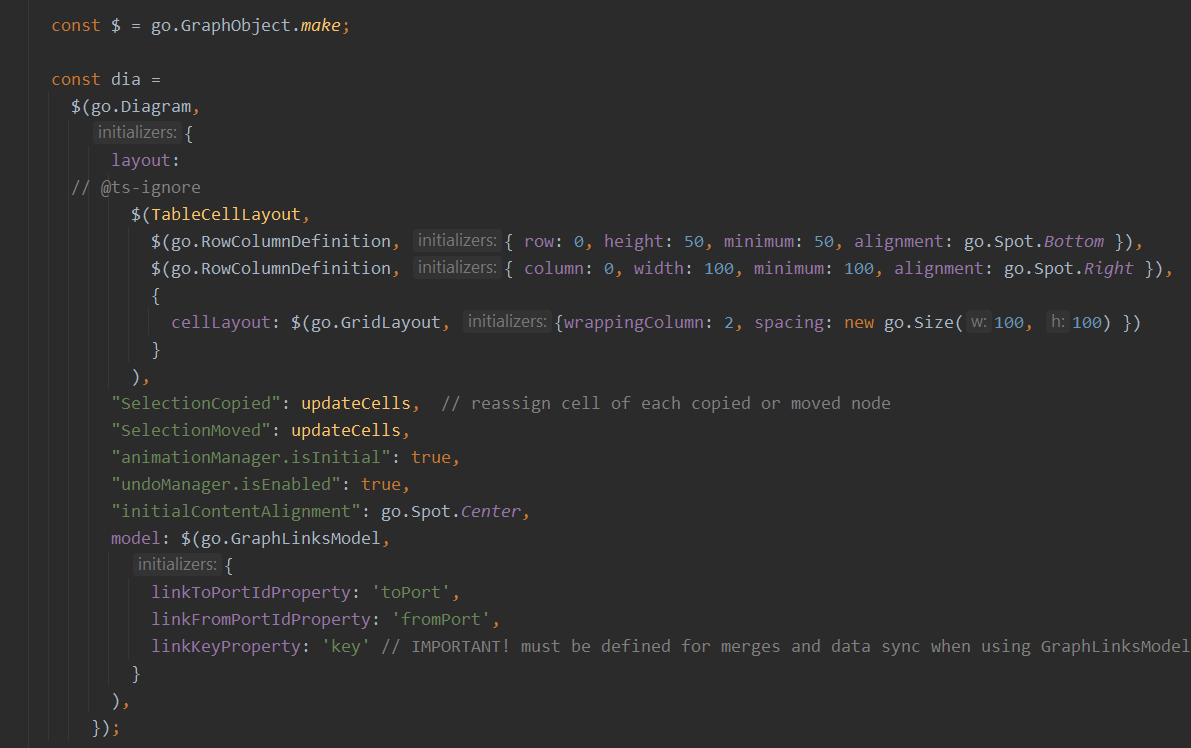


Here for example we have the nodes, links, quantities, and sorting preferences, etc.

This section:



Defines the diagram layout – don’t touch this part. The layout responsible for setting the rows and columns and placing nodes in their place according to the row and col indices.



This part defines our diagram! Her layout is the TableCellLayout.

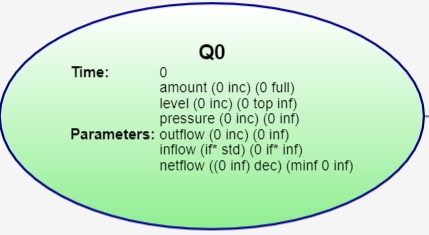
NOTE: I suggest at this part to see a simple example of state graph in goJS: <https://gojs.net/latest/intro/index.html>

Nodes Templates:

In the application when the user clicks a node the node become bigger with more data. How this is works? So, there are 2 types of nodes:

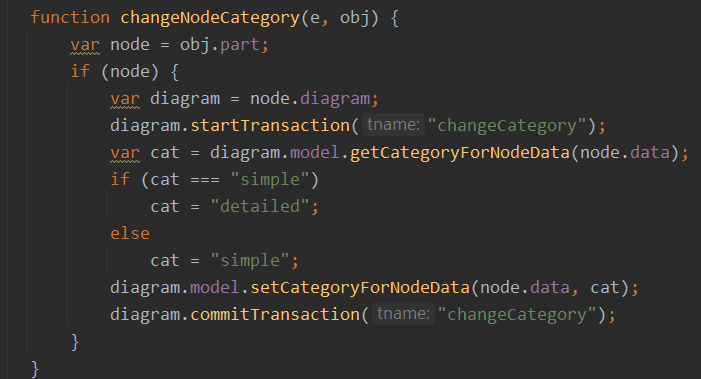
simpleNodeTemplate – simple nodes with only name and color:

detailedNodeTemplate – highlighted node with time and all his parameters values:



**What we do when we clicking a node?**

1. When we click on simpleNode we change his category to detailedNode.



1. Moreover, when we click a node we want to highlight his row in the table.

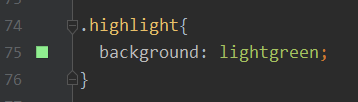


node.key.slice example:

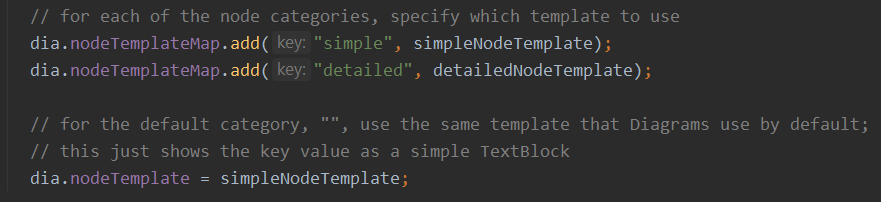
“Q12”.slice(1) = “12” // it’s like s[1:] in Python

+index – convert the string object into integer.

After that we are extracting the row with the state “12” according to html classes, adding this row “highlight” class which defined in the CSS file:



1. We highlight all his links

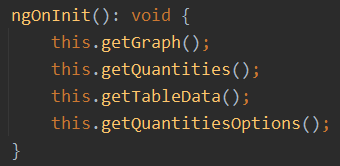


This section defines the nodeTemplatedMap (please see example: <https://gojs.net/latest/intro/templateMaps.html> (search: “Changing category of a Part”)

The last line in the picture: “dia.nodeTemplate = simpleNodeTemplate” defines that the default template is simpleNodeTemplate.

As we have two types of node, we have 3 type of links:

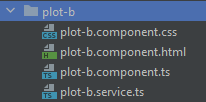
* Simple – simple link from u to v
* Detailed – with all the changes\_quantities as label
* Highlighted – when clicking a node the link become red



Init the data from the server.

**Home-service file**: defines how to access the server and ask for the data / send him data.

**Plot-B:**

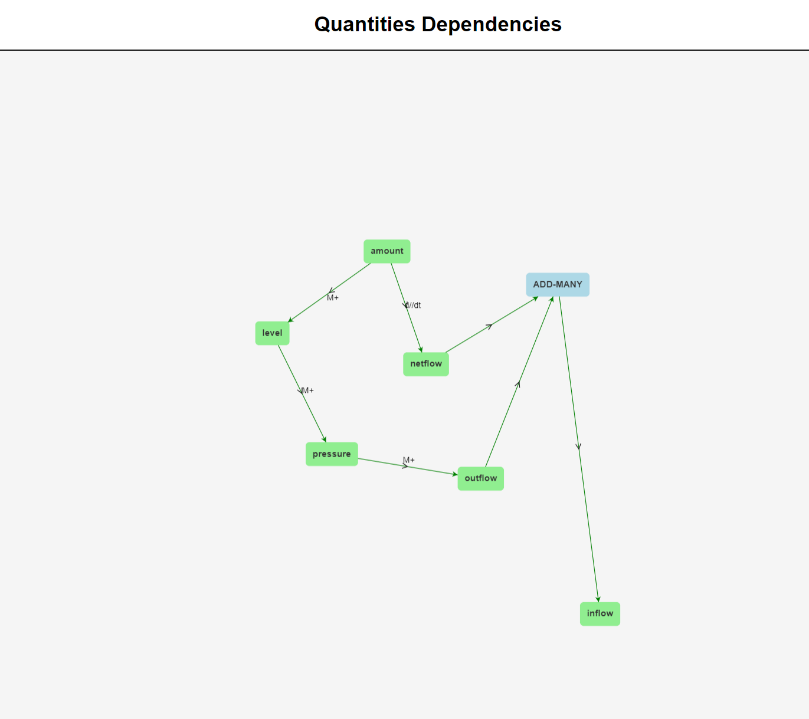


**CSS** – css file for the view adjustment

**HTML** – the Html file

**Component.ts** – the main file! This file represents how the component works! Let’s take a look:

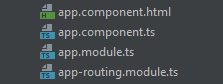
This component is smaller than the home page component. It only should display the quantities graph:



This diagram using layout: ContinuousForceDirectedLayout.

The node template in this case is very simple – just display the key and color (that sends from the server)

**Plot-b-service**: defines how to access the server and ask for the data / send him data.



Those files define the routing in the application. Do not touch them, unless you want to create a new component/ new routing.

**Types.ts** – defines types (it’s like structs in C)

# **Development Questions**

Dear engineer,

If you read this section, you probably want to learn how to change some features in my application. That’s great! I am suggesting trying to solve every question by your own and after that read the solution in the Solutions section. Good luck!

Algorithms:

1. How the sorting (Horizontal and vertical) algorithm works?
2. Explain the parsing algorithm for the CQ input file – how the quantities dependencies graph works?
3. Explain how the parsing algorithm for the CQ output file – how to parse the CQ output into GML format?

Appearances:

1. How the colors of the nodes are generated when the user clicks “color by time”?
2. Which part (server/client) is responsible to give color to each node?
3. How to display the detailed node template as default instead of the simple node template?
4. Explain how to change the colors of the quantities dependencies nodes graph?
5. Explain how to change the labels of the quantities dependencies edges graph?
6. Explain how you will change the color of the highlight color in the states table?

New features:

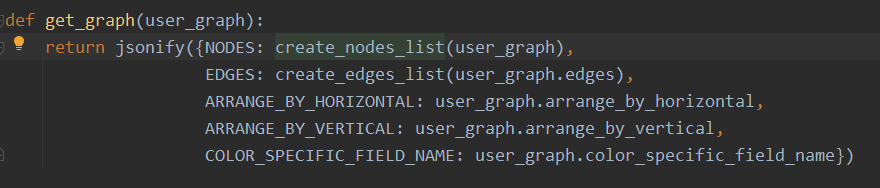
1. Add a new button that called “my new Button”. The button will be routing to a new tab that called “/newRouting”

Connections settings:

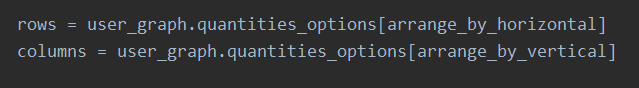
1. Find the server port and change him to 9090.

# **Solutions**

1. Let’s take a look on CQServer.get\_graph and we search for the node creation function. In the node creation function we will see this block of code:



In Create\_node\_list:



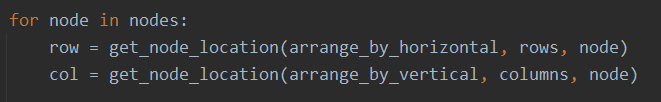
if we want to sort horizontal by time (Arrange\_by\_horizontal = ‘Time) and sort vertical by level (Arrange\_by\_vertical = ‘level’):

Rows will be [0, (0 1), 1, (1 2), 2],

and columns will be: [(0 dec), (0 s std), (0 inc), ((0 top) inc), (L1 std),, (top std), (top inc)]

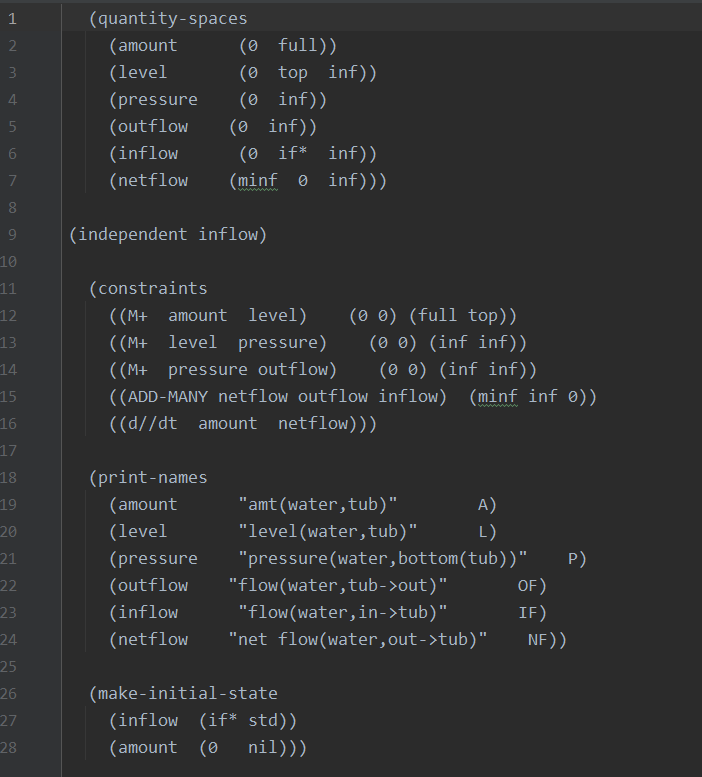
after that, we are setting each node according to this time value and level value. We ask:

“node where your time value located in rows and where your level value located in columns?”



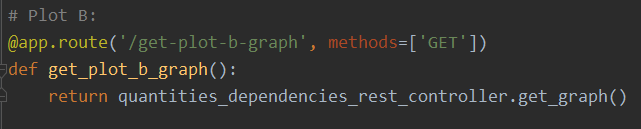
After that, we are setting the row and col values in node object that sent to the UI component.

1. This question is a pure parsing question. Let’s see the input\_file of this case:

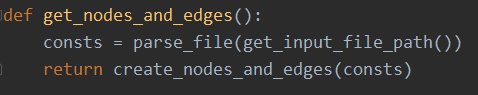


And we want to take only the constrains.

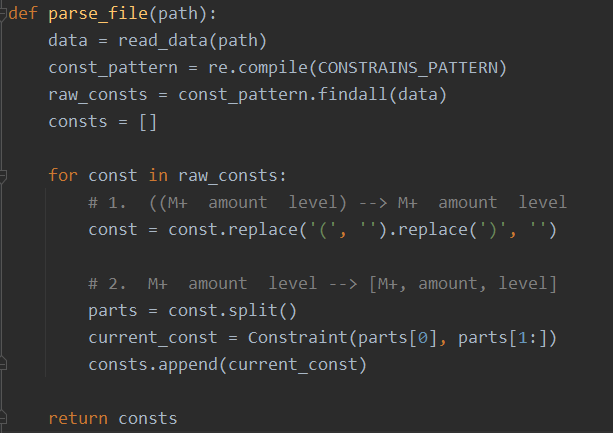
Before I explain the algorithm, let’s find the parsing section in the code. Go to CQServer (why? Because the UI gets the data from this REST controller) and we will see a function that called:



Sounds good!



There is a method that called “parse\_file” this is the parsing algorithm function!

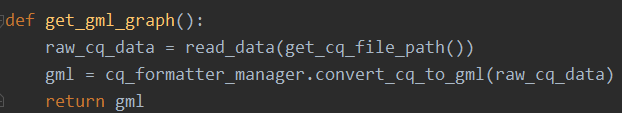


So, it’s very simple parsing! First, we run regular expression to extract only the constrains section. And after that, I am parsing the raw strings in two steps as we can see.

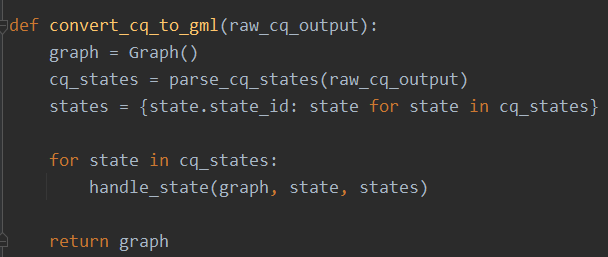
1. This parsing question is quite complicated. As we saw on question 2, let’s find the parsing function from CQServer.py:



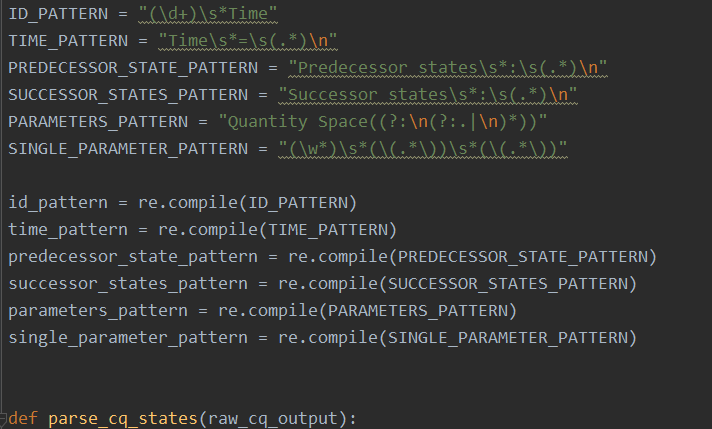
In this function we call to:



Let’s see the convert\_cq\_to\_gml – this is the main function here.



parse\_cq\_states using a lot of regex to extract the data. Let’s see that:

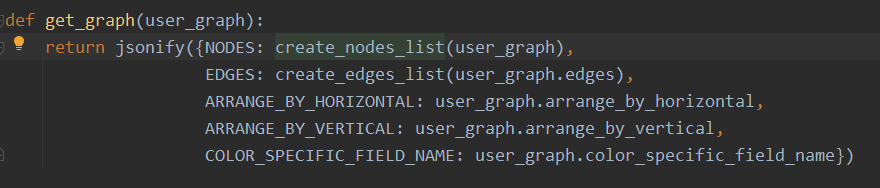


I am suggesting taking the cq\_output and put it here: <https://regex101.com/>

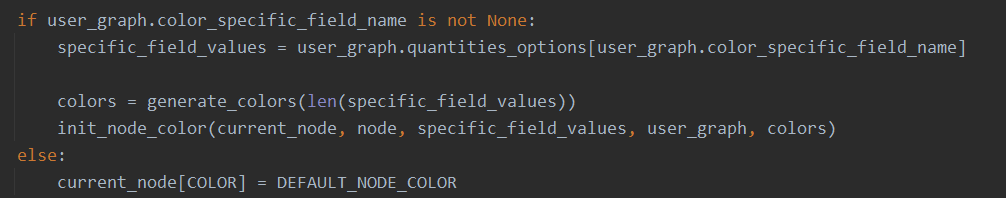
And put every pattern here and see how the regex recognize each section.

1. To generate colors to all the nodes according by time (time can be replaced with any field. I chose time just for the example) we have to find how many times exists., after we know that we have N times we need to generate N colors and decide the #1 color belongs to the first time, and the #2 color belongs to the second time and so on…

Let’s take a look at CQServer.get\_graph. In get\_graph we will search the solution in:



create\_node\_list:



If the user didn’t choose any color – we set the default color.

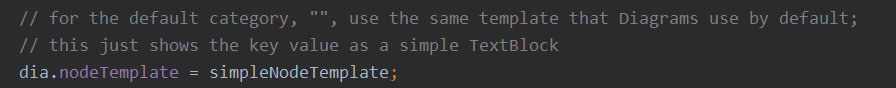
In case the user did choose a color – we ask: “give me all the possible values of this field” and after that generate len() colors. After we have array of N elements (of colors), we give each value a unique color – see: init\_node\_color()

1. As we saw on solution 4, the color is defined in the server side. Moreover, if you take a look on the node’s templates, you can see there is a dynamic color:



The field color comes from the node properties.

1. As we saw on the “Nodes templates” part, we can define what is the default node template to all the nodes. Go to [home.component.ts](http://home.component.ts):



If we want to display the detailed node template, we will write:

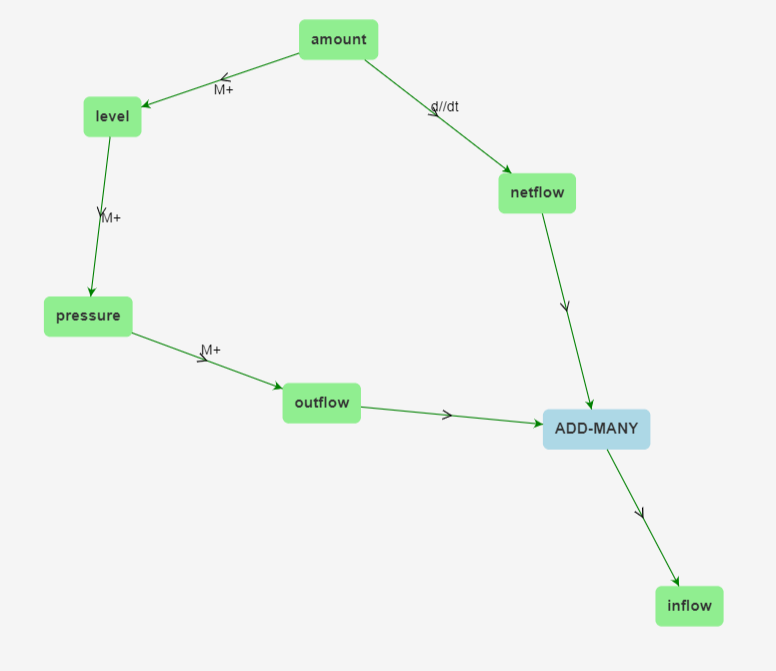


1. We need to change the nodes color in the quantities dependencies, so go to:

plot-b.component.ts and search the node template:



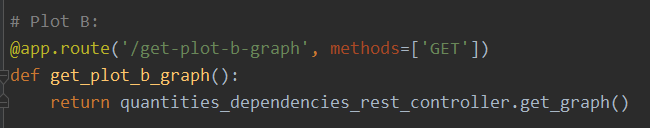
As we can see there is a dynamic color for each node:



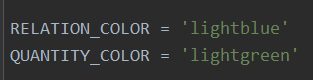
The quantities colored by “lightgreen” and the relations colored by “lightblue”.

So, if we want to change the colors, we need to change it in the **server side.**

Go to CQServer:



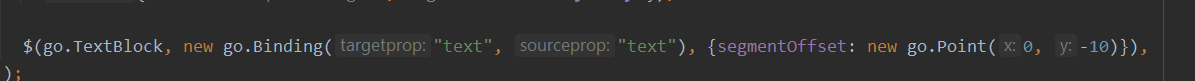
In get\_graph:



There you go!

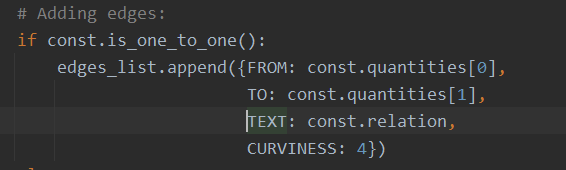
1. We need to change the edges labels in the quantities dependencies, so go to:

plot-b.component.ts and search the link template:



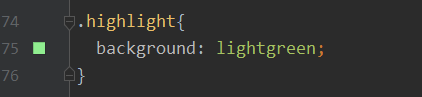
And we see there is a dynamic text that arrives from the server.

We need to change the text in the quantities\_dependencies\_rest\_controller.py



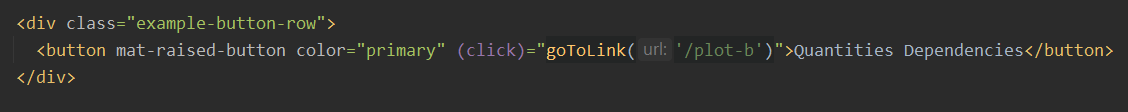
Here we put in the text the relation name (M+, M-, etc..) change it to your new text 😊

1. That is a very simple question. The highlight in the table happens because of a CSS class. Go to: [home.component.css](http://home.component.css) and change the class .highlight:

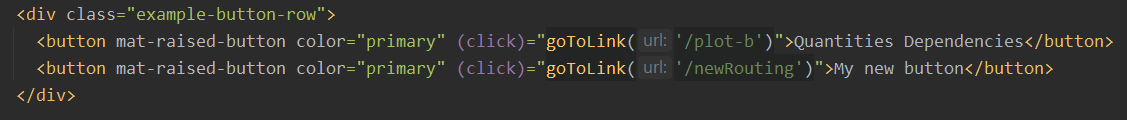


Here you can provide a color or HEX color (if you are using IntelliJ or another idea there is a autocomplete of colors)

1. To create a new button, go to the HTML file at the home component:



Here can you add the new button:



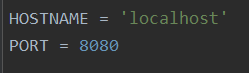
But we want to support the new routing. So, go to: app-routing.module.ts

This file defines how the routing works.



Now we need to create a new component. I suggest copying an existing component (like home or plot-b and change his name 😊)

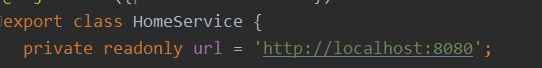
1. The server port defined in CQServer.py:



BUT this is not enough. We need to change the rest calls from the angular part to ask the data from: localhost:9090 and not location:8080.

Where we defined this?

That’s right. In the services files:



And here:



Change the 8080 to 9090. Run “run\_me.bat” file and that’s it 😊