Logical Glue Front-End Coding Exercise

The Problem

We have a REST API service that is returning an array of “membership function” objects from our business layer. Each JSON object in the array represents a Type 2 Fuzzy Logic fuzzy set.

Your task is to create a small component/library to visualise these membership functions within a browser.

Data

An example of the JSON produced by the REST service is as follows:


```
[{
  "name": "Low",
  "lowerStart": 4000.0,
  "lowerTop1": 4000.0,
  "lowerTop2": 33600.0,
  "lowerEnd": 43800.0,
  "upperStart": 4000.0,
  "upperTop1": 4000.0,
  "upperTop2": 33600.0,
  "upperEnd": 54000.0
}, {
  "name": "Medium",
  "lowerStart": 43800.0,
  "lowerTop1": 54000.0,
  "lowerTop2": 65000.0,
  "lowerEnd": 79000.0,
  "upperStart": 33600.0,
  "upperTop1": 54000.0,
  "upperTop2": 65000.0,
  "upperEnd": 100997.14285714287
}, {
  "name": "High",
  "lowerStart": 79000.0,
  "lowerTop1": 100997.14285714287,
  "lowerTop2": 200000.0,
  "lowerEnd": 200000.0,
  "upperStart": 65000.0,
  "upperTop1": 100997.14285714287,
  "upperTop2": 200000.0,
  "upperEnd": 200000.0
}]
```


The same data has also been supplied in an “example.json” file along with this document.

> ***Note:*** *For the purposes of this exercise, you can supply the data to the component as a plain JS object. You do not need to create a back-end REST service.*

Our data scientist had the following to say about this data:

> *'Each membership function has a name and describes two overlaid trapeziums. If you consider the projection of each trapezium's footprint to a 2D plane, then each has a "core" area, which is the range between its "top1" and "top2" values. And it also has a "fuzzy" area, which is the range between its "start" and "end" values.*
>
> *The fuzzy areas of adjacent membership functions should overlap, but their core ranges will not. The first membership function will have no fuzzy area on its left, and the final membership function will have no fuzzy area on its right.*
>
> *For the purposes of a simple visualisation, we should ignore the "lower" trapezium and only use the "upper" one.'*

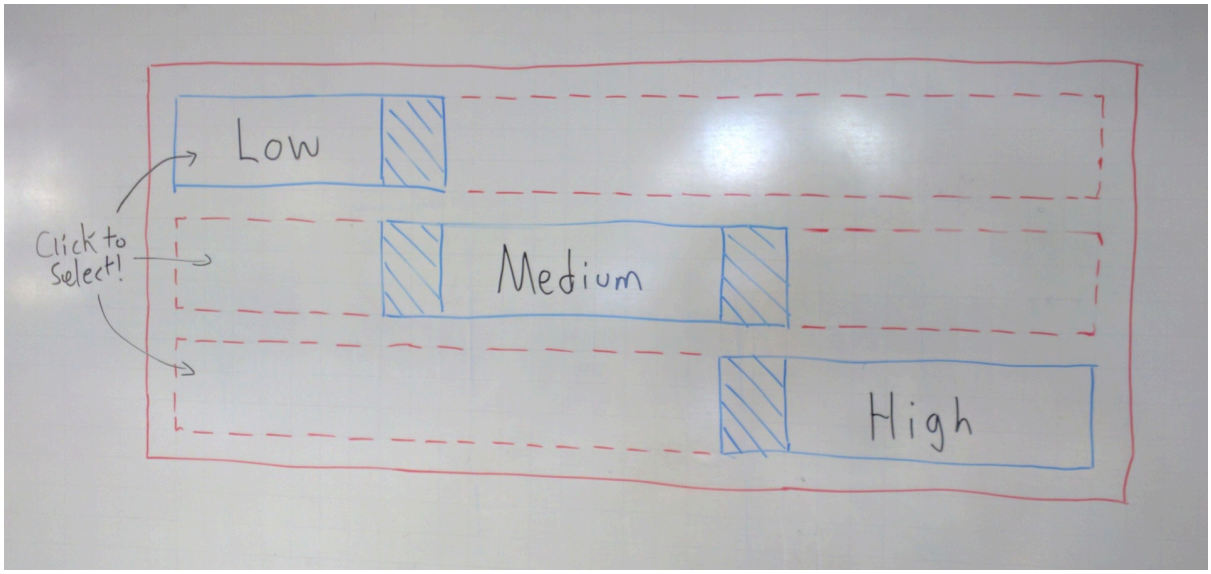
The membership functions will be supplied in left-to-right order, without overlaps between their core ranges, as in the above example.

The component should be flexible enough to handle variations of the data. E.g.

- Different numbers of membership functions (within a reasonable range of $1 < x < 11$)
- Different numeric values for membership functions.

Design

During a design discussion, someone sketched out the following idea for how to visualise the data:



Functional Requirements

1. The component should accept input data as a JS object (e.g. a JS representation of the JSON given above).
2. Each membership function should be displayed as a subrange of the entire range (e.g. that covered by all the membership functions)
3. The width of the displayed membership function should correspond to its size relative to the other membership functions.
4. The “fuzzy” range should be visually distinguished from the “core” range.
5. Each entire range should be clickable. The component should provide some event or callback that the application code can hook into to receive notifications of clicks, along with the data of the corresponding membership function.
6. The control should expand to fit its container.

Non-Functional Requirements

The component should be flexible enough to be re-used in different places throughout the application, wherever we need to visualise membership function data.

The code should follow industry best practices with regards to encapsulation, performance, reusability, testability, and so on.

Technology

At Logical Glue, we currently use Angular 1. If you are experienced with that framework, we would be interested in seeing how you tackle the problem with it. However, we are also happy to receive code using other technologies such as React, Ember, Knockout, etc., or without the use of any framework or library at all.

You may use other libraries and compilers as you see fit. In an exercise like this, some over-engineering is acceptable, since the purpose is to demonstrate skills, not produce the minimum viable solution.

Your submission should be supplied either as .zip archive via email, or as a link to git repository. Please provide complete instructions on how to run it.

If your submission requires any compilation, minification or bundling build steps, please run this locally and include the output in your submission, and supply the corresponding sources in the form in which they were authored.

The use of standard package managers such as npm for installing dependencies is allowed. Please provide complete instructions on how to perform the install. The installation must succeed on both Ubuntu and macOS. If you are in any doubt, include the dependencies within your submission archive as well.

Good luck!