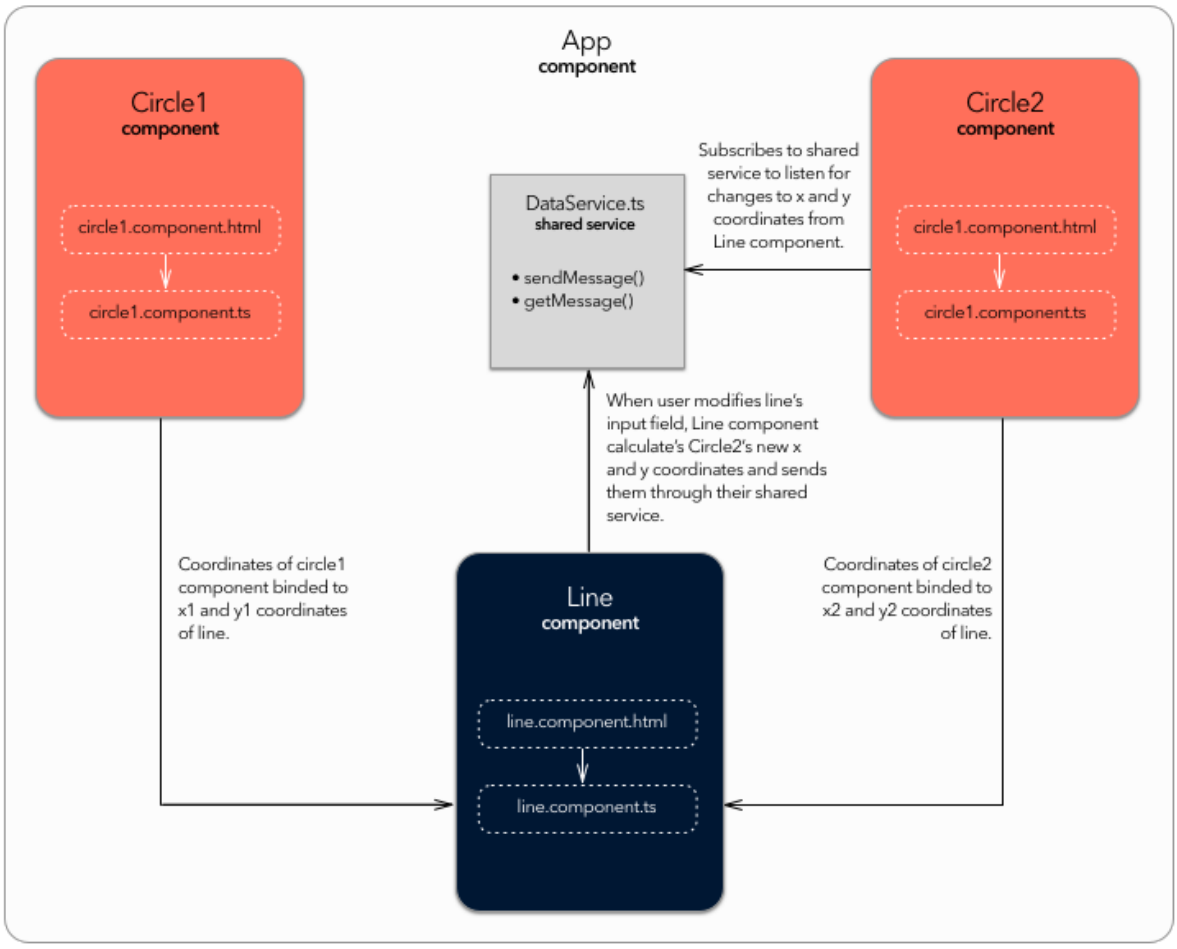
Vik Kalkat
June 24, 2019

Challenge Submission Available at:
Github: https://github.com/vskalkat/vev-challenge
Heroku: https://vev-challenge.herokuapp.com/

Overview
The challenge was completed using AngularJS, particularly because of my comfort with the framework through experience. Also, the data binding played a very useful role for the required functionality.



This diagram visually summarizes the structure and interaction of components in the app.

Structure

The project follows a typical MVC structure, where the model represents the data in the app, in this case the coordinates of various elements. The view represents the user interface, and the controller represents the logic used for handling data and events. For organization, individual elements on the screen are organized into their own components, providing clean separation between each of their implementations. The three separate components are:

- Circle1 - the first circle, starting at coordinates (x,y) = (0,0).
- Circle2 - the second circle, starting at coordinates (x,y) = (400,400).
- Line - the line connecting the Circle1 and Circle2.

Process

1. Send X and Y coordinates from Circle1 and Circle2 to Line.

At first, Circle1 and Circle2 were developed with draggable functionality. The Line component was created last. All three of these components are placed as children inside app.component.html. Through use of template reference variables, Line is easily able to receive Cricle1 and Circle2's individual x and y coordinates. Having these values allows Line to know where it should start and end.

2. Handle Input from Line Component

When the Line component's input changes, Circle2's position is expected to change appropriately. This required various trigonometric calculations. Since Line has access to Circle1's x and y coordinates, which we will call x1 and y1, and it also has access to Circle2's x and y coordinates, which we will call x2 and y2, it made most sense to perform the calculations within the Line component.

To calculate the new (x2, y2) coordinates, an angle must first be calculated using sine rule. To do this, I used the length between Circle1 and Circle2 BEFORE the input was changed.

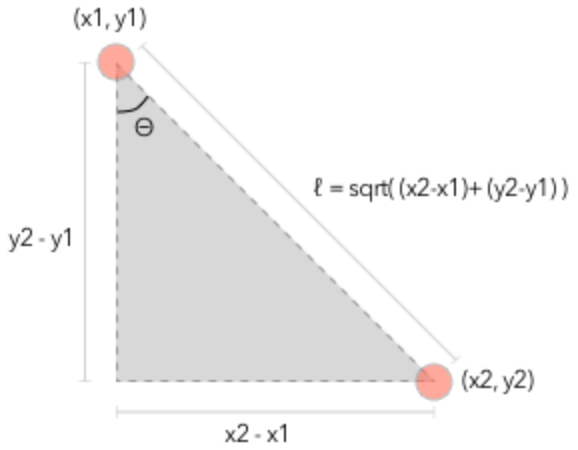


Diagram shows the trigonometric illustration drawn between Circle1 and Circle2.

Using this setup, Θ was calculated as follows:

$$\Theta = \arcsin\left(\frac{x2 - x1}{\ell}\right)$$

Then, the new x2 and y2 position can be calculated using Sine rule and Cosine rule, this time using the NEW line length.

$$X2 = \ell\sin\Theta + x1$$
$$Y2 = \ell\cos\Theta + y1.$$

3. Update Circle2's Position.

Circle2's position was updated upon change in Line's input value. This was accomplished using a shared service. A shared service allows any component within the app to subscribe to its events. The subscribed component actively listens for those events to be triggered and responds when they do so.

After Line calculates (x2, y2), it sends the data into the shared service. When this happens, Circle2 is actively listening for incoming messages and thus received the new data and updates its x and y coordinates.

Next Steps

Create Interface for a Position Model

An important part of all 3 of the elements are their positions, namely their x and y coordinates. Since these position attributes are used so commonly across the app, it may be cleaner to implement a Position model with attributes 'x' and 'y'. This way, each component can simply use one position model to handle the x and y positions rather than declaring two separate x and y variables every time.

Reduce shared calculation

When the input value on the Line component is changed, circle2's position is modified as a result. Currently, the Line component calculates what circle2's new X and Y position should be when the Line's input value is changed. This may be messy since a variable used in Circle2 is calculated in Line.

Refactor Repeated Code into a Function

Various calculations are performed multiple times in Line.component. Firstly, the Pythagorean theorem is performed during component initialization in order to calculate the starting value that should be placed within the input field. Also, the sine rule is performed to calculate the angle upon initialization as well because this angle is required in order to rotate the input field for styling purposes. Both of these calculations are performed again every time the input value is changed by the user. Any time a line of code appears more than once, it is cleaner and more efficient to turn it into a function call.