Ah, I see! You're referring to the **Dijkstra Algorithm app** we worked on earlier, where I helped you set up the **grid-based pathfinding** with named locations (like "Alpha City", "Beta City", etc.) for your mock data. I remember now — you were simulating paths through a network of cities or nodes with weights (distances) and you wanted it to be visualized, using Dijkstra for the shortest path.

Here's a quick recap of what we did previously:

- We had **a grid** or **node-based graph** with cities like "Alpha City", "Beta City", etc.

- Each **node (city)** had **edges** with weights (representing distances or travel time).

- We implemented **Dijkstra's Algorithm** to calculate the shortest path from a starting node to a destination.

You're now looking to **incorporate TensorFlow.js** for dynamic node weight adjustments based on factors like **weather, accidents, or road closures**.

What to Do Next?

Since you've checked out the code, here's a **step-by-step** approach to integrate **TensorFlow.js** into the **Dijkstra app**:

1. Integrate TensorFlow.js for Dynamic Weight Adjustments

You can create a model (or a mock prediction for now) to adjust weights based on weather, road closures, and accidents. Here's how you can start:

1. **Create a TensorFlow service** (e.g., ai-weight.service.ts) that handles the prediction logic.

2. Pass in data (weather, accidents, etc.) as input to the model, and get the predicted weight adjustments for each node (road segment).

2. Modify Your Dijkstra Algorithm to Use These Predictions

Once you have dynamic weights coming from TensorFlow.js, **update your graph's edge weights** before running Dijkstra's algorithm. Here's what the process looks like:

- Get the **predicted weight** for each road/edge from the TensorFlow model.

- Use these **updated weights** to compute the shortest path using Dijkstra.

3. Mock Data for Testing (Reno Lake Tahoe)
Since you mentioned you have mock data for the **Reno Lake Tahoe region**, use that to feed into the TensorFlow model. Each segment between cities would have features like:

- Weather (e.g., snow, rain)
- Accidents (severity)
- Time of day (peak hours)

You can map these into a **feature vector** to pass into the model.
4. Set Up TensorFlow.js in Angular
If you're starting fresh with TensorFlow.js in this app, follow these steps:

1. **Install TensorFlow.js:**
   npm install @tensorflow/tfjs
2.
3. **Create the AI Service** (e.g., ai-weight.service.ts) where TensorFlow.js will be used for predictions.

Example of TensorFlow Service:

```
// src/app/services/ai-weight.service.ts
import { Injectable } from '@angular/core';
import * as tf from '@tensorflow/tfjs';

@Injectable({
  providedIn: 'root',
})
export class AiWeightService {
  private model: tf.LayersModel | null = null;

  constructor() {
    console.log('TensorFlow.js version:', tf.version.tfjs);
    this.loadModel();
  }

  // Load a pre-trained model or build one
  async loadModel() {
    this.model = await tf.loadLayersModel('assets/model.json'); // Assuming
you have a pre-trained model
    console.log('Model loaded');
  }
```

```
  // Simulate prediction (for now, mock data)
  predictWeights(weather: string, accidentSeverity: number, timeOfDay:
string): number {
    // Map input features to a tensor
    const inputTensor = tf.tensor2d([[this.weatherToValue(weather),
accidentSeverity, this.timeOfDayToValue(timeOfDay)]]);
    const prediction = this.model!.predict(inputTensor) as tf.Tensor;
    const predictedWeight = prediction.dataSync()[0]; // Get predicted
weight

    return predictedWeight;
  }

  private weatherToValue(weather: string): number {
    // Example mapping
    return weather === 'snow' ? 1.5 : weather === 'rain' ? 1.2 : 1;
  }

  private timeOfDayToValue(timeOfDay: string): number {
    // Example mapping: morning=0.5, evening=1.0, etc.
    return timeOfDay === 'morning' ? 0.5 : 1.0;
  }
}
```

This service:

- Loads a pre-trained TensorFlow model (you can either build or mock one).

- Takes in input features (like weather, accidents, and time of day) and returns a predicted weight.

Example of Using AI in Your Map/Graph:

In your **Dijkstra algorithm** implementation, you'd update the edge weights based on the predictions:

```
// src/app/components/map/map.component.ts
import { Component, OnInit } from '@angular/core';
import { AiWeightService } from '../../services/ai-weight.service';

@Component({
  selector: 'app-map',
  templateUrl: './map.component.html',
  styleUrls: ['./map.component.css']
})
export class MapComponent implements OnInit {
  constructor(private aiWeightService: AiWeightService) {}
```

```
  ngOnInit() {
    // Sample data: weather, accident severity, and time of day
    const weather = 'snow';
    const accidentSeverity = 3; // 1-5 scale
    const timeOfDay = 'morning';

    // Get predicted weight adjustment for the road segment
    const predictedWeight = this.aiWeightService.predictWeights(weather,
accidentSeverity, timeOfDay);

    console.log(`Predicted weight adjustment: ${predictedWeight}`);
    // Use this predicted weight in your graph or Dijkstra algorithm
  }
}
```

✅ Recap:

1. **Set up TensorFlow.js** in Angular (you've already done this with npm install @tensorflow/tfjs).

2. **Create an AI service** (ai-weight.service.ts) to handle dynamic weight prediction using TensorFlow.js.

3. **Update your Dijkstra algorithm** to use the predicted weights for dynamic pathfinding based on weather, accidents, etc.

4. **Use mock data** (like weather and accident severity) to test the integration with TensorFlow.js.

Next Step?

- **Want to dive into model training** (if you need a model)?

- **Help fine-tune the weights logic**?

- **Set up mock prediction** first to see how it works?

Let me know how you'd like to proceed, and I can guide you step by step through any part you want!