1. Getting Started
Access the distribution of “trip-sorter” and open “index.html”. There is no need to have an app-server or internet connection.

2. External Javascript Dependencies
The external dependencies are included in the project. Locate them at js/ext-lib.

- **RequireJS**, helps to modularize the project, define dependencies between components and load the required libraries
- **Bootstrap**, to create web-interfaces quickly with responsive designs
- **AngularJS**, to have a (simple) MVC pattern application
- **Graph.JS**, provides graph algorithms, particularly the Dijkstra algorithm to find the shortest path between two nodes

3. General overview

- **index.html** holds the html code
- **js/app-init.js** loads the dependencies and creates the angular application
- **js/app-controllers** contains the application controllers. In this case there is just one, “trip-sorter-controller”
- **js/app-controllers/object** contains the object prototypes used by the controllers
- **js/app-validators** contains angularJS directives to be used as validators
- **js/responseJSON/response.js** contains the test data delivered in response.json. I needed to convert the json file to js to be able to load it without server-side. The cross-origin resource sharing security requires server-side.

4. The Algorithm

- The algorithm used for both “cheapest” and “fastest” options is the Dijkstra algorithm to find the shortest path between two nodes
- The cost of the edge is either the price (“cheapest”) or the time in minutes (“fastest”)
- The implementation of the Dijkstra algorithm expects at maximum a single directed edge between 2 nodes in each direction.
  In certain cases, there can be 2 or more connections between the same pair of cities (e.g. one by train, another by bus)
  This is solved picking up the one with lower edge cost, prior to call the Dijkstra shortest path algorithm.
- I have done an improvement to optimize the cases when there are trips with the same cost (either price or time). The second criteria is “cheapest” or “fastest”, the opposite of the selection.
  This is done adding the “second criteria” to the edge cost as a very small fraction. For example, if the selection is “cheapest”, the edge cost is calculated:

  ```
  edgeCost = priceCostWithDiscount + (timeMinutesCost / VERY_LARGE_NUMBER)
  ```

  This is just for the algorithm to decide for the best option, in case of “draw”.

5. Constraints
Without server-side API it was not possible to include the raw response.json in the project due **to** cross-origin resource sharing security constraints.
To replace the data, please do it in **js/responseJSON/response.js**.