Some ideas on SimCSS online interface:

There are two libraries worthy of considerations:
D3.js (https://d3js.org) is a JavaScript library for manipulating documents based on data. Some examples: https://github.com/d3/d3/wiki/Gallery

Leaflet (https://leafletjs.com) is the leading open-source JavaScript library for mobile-friendly interactive maps. Some examples: https://leafletjs.com/examples.html

D3.js will create an interface very similar to desktop client. On the other hand, Leaflet will put everything on the map interface, it is easy to support the scenarios on different geographic locations.
Suggestions: try D3.js first, or divide into two teams, one works on D3.js, the other on Leaflet.

Data: PEARC Hackathon/Business_CCUS/Scenarios/S1R42
Sources/**Sources.txt** and Sinks/**Sinks.txt** have been formatted as GeoJSON;
GeoJSON is a format for encoding a variety of geographic data structures in JSON.
Sources.geojson

```
{"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name":
"urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "ID": 1, "costFix ($m)": 0.0,
"fixO&M ($m\/y)": 0.0, "varO&M ($\/tCO2)": 65.0, "capMax (MtCO2\/y)":
15.0, "capFctr": 1.0, "LON": -87.7659, "LAT": 38.3689, "NAME": 1,
"CREDIT": 0.0, "#GenUnts": 1, "ID_1": 1, "ID_2": 1 }, "geometry": {
"type": "Point", "coordinates": [ -87.7659, 38.3689 ] } }
]}
```

Sinks.txt

```
{"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name":
"urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "field_1": 1, "field_2": 1, "field_3":
88.49, "field_4": 0.0, "field_5": 0.0, "field_6": 88.493, "field_7": 0.0,
"field_8": 0.0, "field_9": -6.72, "field_10": 0, "LON": -88.3261, "LAT":
38.5645, "field_13": 0, "field_14": 0, "field_15": 0, "field_16": 0,
"field_17": "Clay City Consol.", "ID": 3, "field_19": 384466, "field_20":
4269288, "field_21": -6.72, "field_22": -1.09, "field_23": 18.32,
"field_24": 88.49, "field_25": 57.08, "field_26": 25.66 }, "geometry": {
"type": "Point", "coordinates": [ -88.3261, 38.5645 ] } }, ...
```

GeoJSON support reference: **d3-geo** (https://github.com/d3/d3-geo), GeoJSON on leaflet (https://leafletjs.com/examples/geojson/)

Scenarios/S1R42/Results/PEARC_Hackathon.1531846206038/shapeFiles/Networks.shp is also converted into **Network.geojson**

Networks: PEARC Hackathon/Business_CCUS/Scenarios/S1R42/Network
Delaunay Network is defined in **DelaunayPaths.txt**

```
#  Selected node pairs
SINK  20    SINK  17    29329 32623
SINK  40    SINK  37    26944 32449
SINK  13    SINK  41    24231 24922
SINK  25    SINK  41    24488 24922
SINK  19    SINK  37    22341 32449
SINK  10    SINK  2     13502 19017
SINK  35    SINK  20    28024 29329
SINK  5     SINK  9     30187 31519
SINK  34    SINK  17    32416 32623
SINK  32    SINK  23    21356 32165
SINK  25    SINK  8     24488 24937
SINK  32    SINK  13    21356 24231
SINK  23    SINK  17    32165 32623
SINK  12    SINK  9     28661 31519
SINK  4     SINK  18    9275  14809
SINK  7     SINK  41    22715 24922
SINK  23    SINK  39    32165 35954
SINK  35    SINK  34    28024 32416
SINK  33    SINK  30    17680 20752
SINK  26    SINK  23    18045 32165
SINK  16    SOURCE      1     18380 19031
```

Sink ID is defined in Sinks.geojson, you will able to get coordinates and draw a line to connect two points.