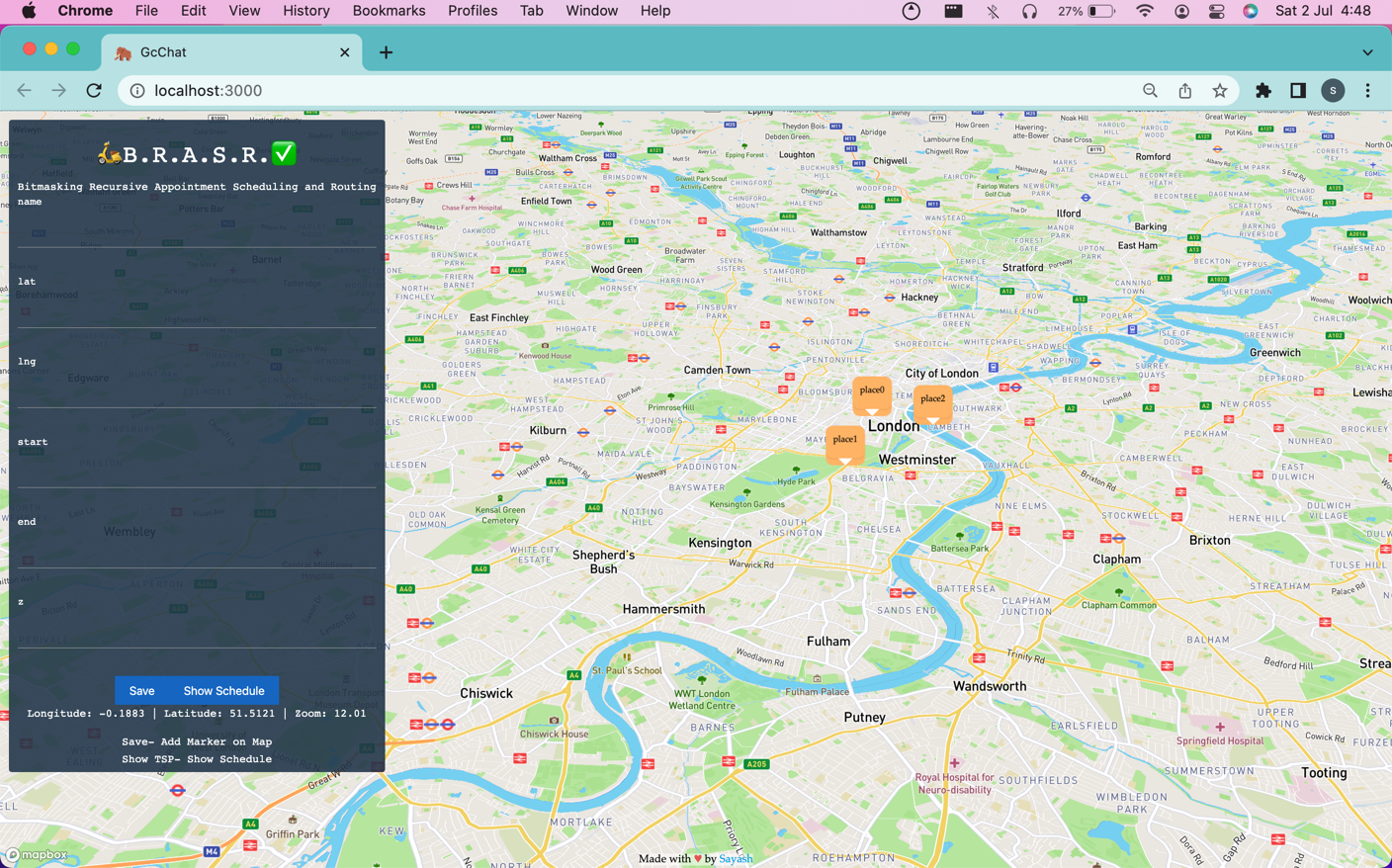
Team Sayash

🛵B.R.A.S.R.✅

Bitmasking Recursive Appointment Scheduling and Routing



# Documentation

* Deliverables
* Working
* Example JSON Request and JSON Response
* API model
* UI integration

EXL Hackathon- building solutions for larger challenges faced by insurance companies; solutions that fulfill the needs of consumers; solutions that improve customer experience

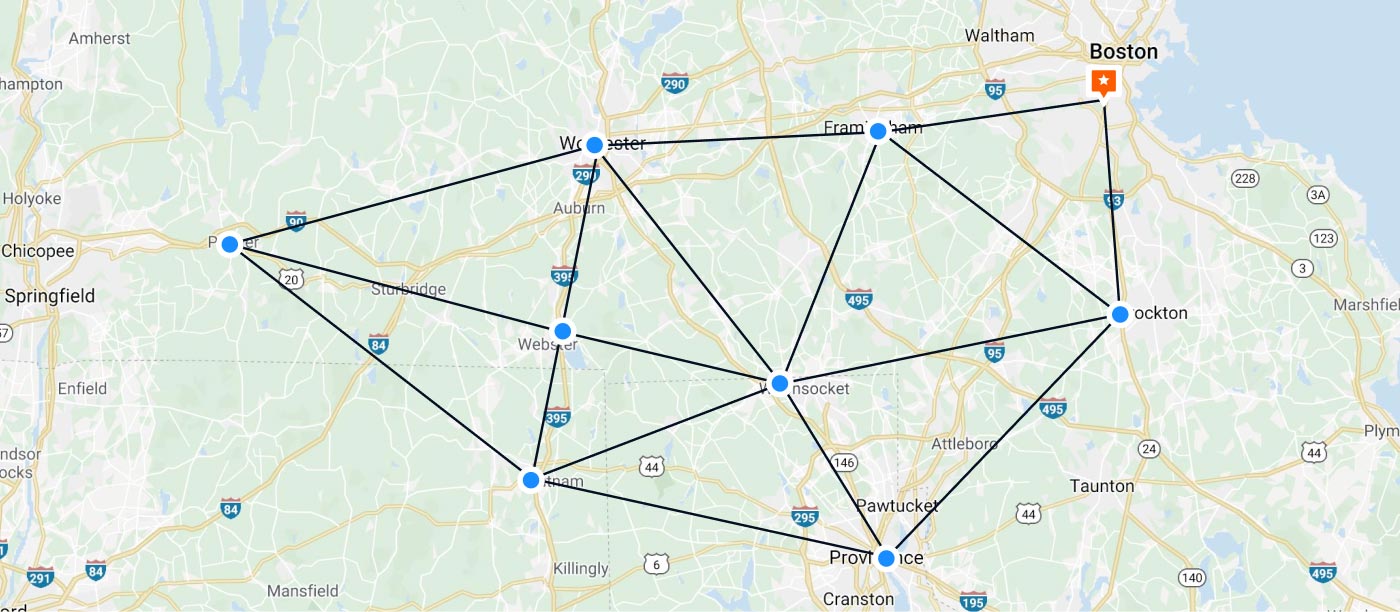
Theme- Create a solution for Appointment Scheduling and Routing

Appointment scheduling and routing solution helps in reducing manual, time-taking processes such as maintaining data. security, etc.

This solution helps in tracking locations and addresses. It has been integrated successfully with online maps to improve efficiency, which is not possible in a manual offline process.

Tech Stack

* REST API
* Frontend Framework
* Custom-built scheduling and routing algorithm
  + Uses a 32-bit Bitmasking Recursive approach using an NP-hard solution framework
  + Recursively finds the shortest path w.r.t time with given constraints
  + Uses graph-algorithmic approach
  + Uses bitmasking to store mid-execution state of algorithm



Visualising nodes and edges on the constructed graph

🛵B.R.A.S.R.✅ How it works-

Is the best and most efficient way of solving the problem.

Using custom-built modifications on the age-old original Travelling Salesman Problem (TSP), which is considerably the most famous NP-hard problem, we have built the 🛵B.R.A.S.R.✅ algorithm

Solution Flow

User enters locations to visit, entering information in a simple form- clicks Save button

Form response is appended to a global array consisting of locations added so far

User clicks Show Schedule button, global array is sent as a JSON array of objects as body request to the Cross-Platform server

[

{

"id": "place0",

"coords": {

"lat": 51.51198245486377,

"lng": -0.1278277598563

},

"start": 0,

"end": 1000,

"z": 0

},

{

"id": "place1",

"coords": {

"lat": 51.503120589264064,

"lng": -0.15282095066100

},

"start": 0,

"end": 1000,

"z": 0

},

{

"id": "place2",

"coords": {

"lat": 51.503341807681544,

"lng": -0.11952824596429

},

"start": 0,

"end": 1000,

"z": 0

}

]

example request.body sent to server

The Cross-Platform Modular Server parses the JSON array in the input parsing node

Server send an internal request to perform the calculative task using the 🛵B.R.A.S.R.✅ algorithm

* + Calculative function sends an external POST request to Time Travel Matrix API, obtaining information on least distance and time between each node
  + Parses external API response into an n x n matrix, n = number of nodes
  + Constructs a dense network as opposed to a sparse network of nodes
    - Number of edges = nC2 = n(n-1)/2 nodes
  + 🛵B.R.A.S.R.✅ algorithm runs on constructed dense network, returns object

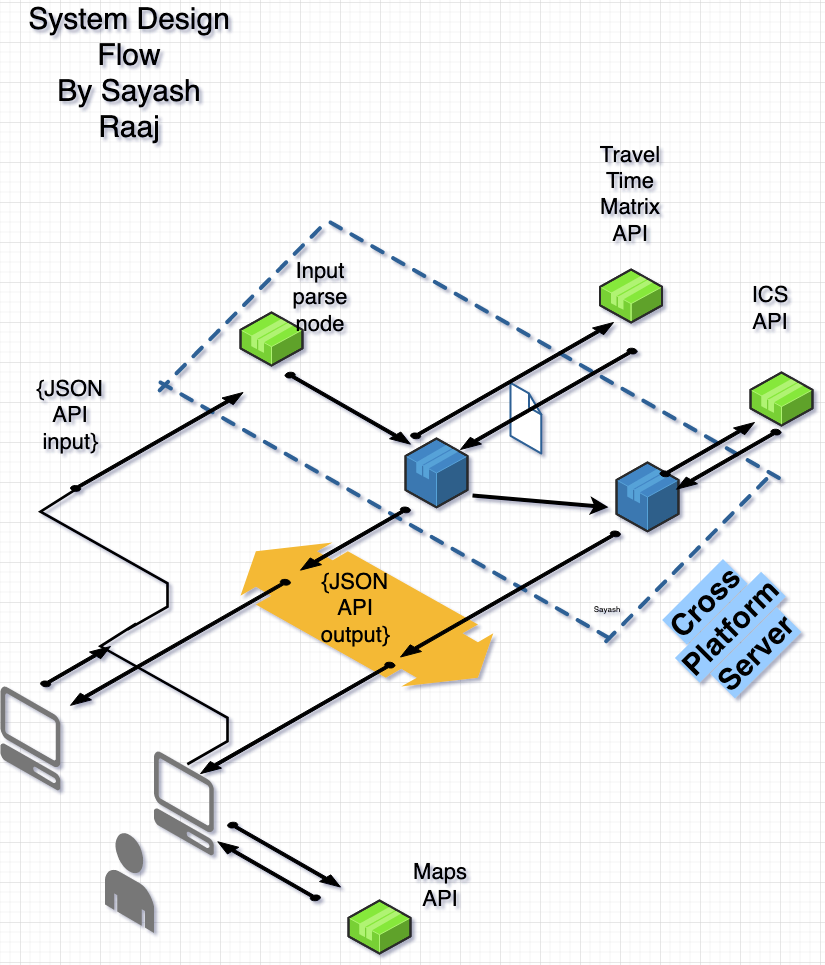
Server reformats output as a union of request.body parameters and 🛵B.R.A.S.R.✅ output

JSON output

Server sends an internal request to ICS generating function

* External POST request sent, response obtained with necessary ICS configuration
* ICS calendar file generated

Server reformats output for ICS file obtained



{

"ans": 3108,

"vec": [

{

"day": 0,

"name": "place0",

"start": "2022-07-02T13:18:09.862Z",

"end": "2022-07-02T13:18:09.862Z"

},

{

"day": 0,

"name": "place1",

"start": "2022-07-02T13:32:50.862Z",

"end": "2022-07-02T13:32:50.862Z"

},

{

"day": 1,

"name": "place0",

"start": "2022-07-03T13:18:09.862Z",

"end": "2022-07-03T13:18:09.862Z"

},

{

"day": 1,

"name": "place2",

"start": "2022-07-03T13:29:22.862Z",

"end": "2022-07-03T13:29:22.862Z"

}

]

}

example response