Cab Allocation Service

Context: Target offers cab service for team members (TM) to drop them to drop-points nearest to their home. Each TM has to register to the travel portal to avail this service and select the drop point. Travel team staff can configure the number of cabs available along with their cost and capacity. They also configure the available drop points, and their distance from target headquarters and also the distance from one drop point to another. (Refer 3: **POST /drop\_points**)

You need to build 4 apis:

1. **POST /register**
2. **POST /cabs**
3. **POST /drop\_points**
4. **GET /routeplan**
5. **POST /register** -> Team members use this api to opt for a cab by providing their details.

**Example**:

**Accepts**: application/json

**Body**:

{

“team\_member\_id”: ”1”,

“gender”: ”M”

“drop\_point”: ”pointC”

}

**Response**: 201

NOTE: drop\_point should be verified against the available drop points posted to “/**drop\_points”** api below.

1. **POST /cabs** -> Travel team staff will use this api to set number of cabs available along with their cost and capacity.

**Example**:

**Accepts**: application/json

**Body**:

[

{

“cab\_id”:”1”,

“cost”: 2

“capacity”: 2

}

]

**Response**: 201

NOTE: The input should be validated such that all cabs together have enough capacity for all TM including any security personnel if required in case if there are all women TM’s. (Refer Rules : 2)

1. **POST /drop\_points** -> Travel team staff will use this api to set all available drop points and also the distance between them and target headquarter.

Example:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SLNO** | **DropPoints** | **Distance to**  **pointA** | **Distance to**  **pointB** | **Distance to**  **pointC** | **Distance to**  **pointD** | **Distance to**  **pointE** |
| 1 | target\_headquarter | 1 | 8 | 1 | 2 | 1 |
| 2 | pointA | 0 | 1 | 2 | 1 | 2 |
| 3 | pointB | 8 | 0 | 1 | 3 | 1 |
| 4 | pointC | 7 | 9 | 0 | 1 | 1 |
| 5 | pointD | 2 | 2 | 2 | 0 | 1 |
| 6 | pointE | 2 | 9 | 6 | 7 | 0 |

In above table

#1 : Distance between target\_headquarter to pointA, pointB… pointE

#2: Distance between pointA to pointA, pointB… pointE

#3: Distance between pointB to pointA, pointB… pointE

#4: Distance between pointC to pointA, pointB… pointE

#5: Distance between pointD to pointA, pointB… pointE

#5: Distance between pointE to pointA, pointB… pointE

This can be fed to the “**drop\_points**” api in the following way:

**Accepts**: application/json

**Body**:

{

“target\_headquarter”:”1,8,1,2,1”,

“pointA”: “0,1,2,1,2”,

“pointB”: “8,0,1,3,1”,

“pointC”: “7,9,0,1,1”,

“pointD”: “2,2,2,0,1”,

“pointE”: “2,9,6,7,0”

}

**Response**: 201

NOTE: It’s not necessary for the distance from “pointA to pointB” same as “pointB to PointA”, as there might be one-way traffic or other such constraints.

1. **GET /route\_plan ->** Travel team staff will use this api to see the overall cost of the travel including cab details, TM details, route, cost per cab per route.

**Example:**

**Response Body**:

{

“total\_cost”:”5”,

“routes”: [

{

“cab\_id”:”1”,

“team\_member\_ids”:”4,5”,

“route”: “target\_headquarter,pointE”,

“route\_cost”: 2

},

{…

..}

]

}

**Response Code**: 200

***NOTE:***

1. Here, “route\_cost” is determined by adding all the distance together in the route taken \* (multiplied by) cost of the cab. Example: Here Assuming both TM 4 and 5 are getting down at drop point **pointE**, and **pointE** is at distance 1 from target\_headquarter and cost of the cab is 2, so the overall cost of the cab is = 1 \* 2.
2. Here, “total\_cost” is just the summation of “cost” of all the “routes” in **GET /route\_plan**

**Rules:**

1. All team members should be dropped to their drop points
2. If any women team member is allocated to a cab, the system should ensure that she is not dropped last. If all team members that registered for cabs are women, target will provide a security personnel per cab, meaning the capacity per cab will reduce by 1.
3. The generated **route\_plan** should have minimum **total\_cost** and should adhere to all the above rules.

**Expectations:**

1. Implement as a **Micro service** using **TDD** approach, covering all the edge cases scenarios. Plus points for Documentation and clean code.
2. Using a suitable design pattern is highly recommended. Also, design should be efficient to handle modifications on the go, ex: increasing/decreasing number of cabs, variable number of team\_members, modifications to drop\_points, etc..
3. Exception handling, e.g verify the cabs with sufficient capacity is allocated, drop points are validated against the once provided in **drop\_points** api, etc..
4. In-memory cache/ database can be used for data storage.

**Example**

**Sample Input**

**1)POST /register**

**1)** **POST /register**

{

"team\_members": [

{

"tem\_member\_id": "1",

"gender": "M",

"drop\_point": "pointC"

},

{

"tem\_member\_id": "2",

"gender": "M",

"drop\_point": "pointB"

},

{

"tem\_member\_id": "3",

"gender": "M",

"drop\_point": "pointA"

},

{

"tem\_member\_id": "4",

"gender": "M",

"drop\_point": "pointE"

},

{

"tem\_member\_id": "5",

"gender": "M",

"drop\_point": "pointE"

}

]

}

**2) POST /cabs**

{

"cabs": [

{

"id": "cab1",

"cost": 2,

"capacity": 2

},

{

"id": "cab2",

"cost": 1,

"capacity": 3

}

]

}

**3) POST /drop\_points**

{

"target\_headquarter": "1,8,1,2,1",

"pointA": "0,1,2,1,2",

"pointB": "8,0,1,3,1",

"pointC": "7,9,0,1,1",

"pointD": "2,2,2,0,1",

"pointE": "2,9,6,7,0"

}

**Sample Output**

4) **GET /routeplan**

{

"total\_cost": "5",

"routes": [

{

"cab\_id": "cab1",

"team\_member\_ids": "4,5",

"route": "target\_headquarter,pointE",

"route\_cost": 2

},

{

"cab\_id": "cab2",

"team\_member\_ids": "3,2,1",

"route": "target\_headquarter,pointA,pointB,pointC",

"route\_cost": 3

}

]

}