**Get Cab Search System**

1. **Register a driver**

In this API, you need to save the driver's details in the database. Please ensure that the email, phone\_number, license\_number, and car\_number are unique fields. Also, validate the request object and throw appropriate exceptions and response codes as mentioned below.

**POST** /api/v1/driver/register/

Request Header:

{  
 "Content-Type": "application/json"  
}

Request Body:

 {

"name": "", ----------------------> string, required

"email": "", ----------------------> string, required, should be unique

"phone\_number": ----------------------> number, required, should be unique and equal to 10 digits

"license\_number": "", ----------------------> string, required, should be unique

"car\_number": "" ----------------------> string, required, should be unique

}

Response Code: **201**

Response Body:

{

"id": , ----------------------> integer

"name": "",

"email": "",

"phone\_number":

"license\_number": "",

"car\_number": ""

}

In case of **Bad Request**, like for instance, the email field is missing, then an exception must be thrown with the following details:

Response Code: **400**

Response Body:

{

"status": "failure",

"reason": "explanation" ---------> string, explanation can be any message

}

1. **Share Driver Location**

In this API, you need to save the location of the driver. The location constitutes of the latitude and longitude of the driver. The ":id" parameter field in the request URL(as shown below) is the driver id that is generated when we onboard a driver.

**POST** /api/v1/driver/:id/sendLocation/

Request Header:

{  
 "Content-Type": "application/json"  
}

Request Body:

{

"latitude": 12.972442, ----------------------> double, required

"longitude": 77.580643 ----------------------> double, required   
}

Response Code: **202**

Response Body:

{  
 "status": "success"  
}

In case of a **Bad Request**, like for instance, if latitude field is missing, then an exception must be thrown with the following details:

Response Code: **400**

Response Body:

{   
  "status": "failure",  
 "reason": "explanation" ---------> string, explanation can be any message  
}

1. **Get Nearby Cabs**

In this API, you need to return the list of cabs that are present within 4 km(<=4 km) range of the passenger location using [**Haversine**](https://en.wikipedia.org/wiki/Haversine_formula)distance formula. The passenger's location(latitude and longitude) are provided in the request object. You need to search for all the drivers whose location is within a 4 km range of the passenger's location and return them In the response.

**POST** /api/v1/passenger/available\_cabs/

Request Header:

{  
 "Content-Type": "application/json"  
}

Request Body:

{

"latitude": 12.972442, ----------------------> double, required

"longitude": 77.580643, ----------------------> double, required

}

Response Code: **200**

Response Body:

{

"available\_cabs": [

{

"name": "",

"car\_number": "",

"phone\_number":

},

{

"name": "",

"car\_number": "",

"phone\_number":

},

{...}

]

}

If **no cabs are found**, then the response details should be as follows. Also please make sure that the message should be exactly as shown below.

Response Code: 200

Response Body:

{

"message": "No cabs available!"

}

In case of a **Bad Request**, like for instance, if the latitude field is missing, then the response details should be as follows:

Response Code: **400**

Response Body:

{

"status": "failure",

"reason": "explanation" ---------> string, explanation can be any message

}

Important Notes

* In case you need to store data in a database, then please use an in-memory relational/non-relational database like HSQLDB, SQLite, in-memory MongoDB, etc for performing operations on the data.
* Please make sure that before submitting your solution (in Step 3.), the port of your solution must be set as **8080**. Also, ensure that you have committed and pushed your changes before submitting the solution.
* Please clean your local database before running any test case (in Step 2. Execute Test Cases).
* Please note that every test case has a few sub-test cases inside. These sub-test cases are run in sequential order. So, when the "Run" button (in Step 2.) is clicked for executing a test case, then "if one of the sub-test cases fail, then the subsequent sub-test cases might also fail", as a result of which templatised fields like "$id", "$id2" can be seen when debugging the code.
* Please note that you can submit the solution multiple times.