Description | Submission

View Submission Instructions

Background

There are several CRM that helps manage inbound customer leads. Let us say you are supposed to build a system that stores all the lead information and allows you to edit and update the conversation of the agents with leads. The Schema of the Lead should be as following:

```
{
    "id": ,                     ---------> integer, primary key, unique, non-null
    "first_name": "",           ---------> string, non-null
    "last_name": "",            ---------> string, non-null
    "mobile": ,                 ---------> string, non-null, length:10, unique
    "email": "",                ---------> string, non-null, unique
    "location_type": ,          ---------> enum ["Country","City","Zip"], non-null
    "location_string": "",      ---------> string, non-null
    "status": ,                 ---------> enum["Created","Contacted"], non-null
    "communication": ""         ---------> string, default: null
}
```

Exercises

1. Fetch a lead

   Create a get API to fetch the details about a lead using the id of the lead("lead_id" path parameter) from the database. "lead_id" path parameter is a mandatory parameter.

   GET /api/leads/:lead_id

   Response Code: **200**

   Response body:

   ```
   {
       "id": ,
       "first_name": "",
       "last_name": "",
       "mobile": ,
       "email": "",
       "location_type": ,
       "location_string": "",
       "status": ,
       "communication": ""
   }
   ```

   In case a **Lead is not present**, then you need to return the following response details:

   Response Code: **404**

   Response Body:

   {} ---------> empty response

   In case of a **Bad Request**, like for instance, if the "lead_id" path parameter is missing, then you need to return the following response details:

   Response Code **400**

   Response body:

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

2. Generate a lead

   In this API, you need to add a lead to the database. Ensure that you validate the request object and ensure that it is consistent with the schema of the Lead object.

   POST /api/leads/

   Request Header:

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

   Request Body:

   ```
   {
       "first_name": "",           ---------> required
       "last_name": "",            ---------> required
       "mobile": ,                 ---------> required
       "email": "",                ---------> required
       "location_type": "",        ---------> required
       "location_string": ""       ---------> required
   }
   ```

   Response Code: **201**

   Response body:

   ```
   {
       "id": "",
       "first_name": "",
       "last_name": "",
       "mobile": ,
       "email": "",
       "location_type": "",
       "location_string": ""
       "status": "Created"
   }
   ```

   In case of a Bad Request, like for instance, if the "email" is not present or "email" is duplicate, then you need to return the following response details:

   Response Code 400 (failure): Expected response should be of the form:

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

3. Update a lead

   In this API, you need to update the lead data. The ":lead_id" path parameter is mandatory and it is the id of the lead already saved in the database. Ensure that you validate the request object and ensure that it is consistent with the schema of the Lead object.

   PUT /api/leads/:lead_id

   Request Header:

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

   Request body:

   ```
   {
       "first_name": "",
       "last_name": "",
       "mobile": ,
       "email": "",
       "location_type": "",
       "location_string": ""
   }
   ```

   Response Code: **202**

   Response body:

   ```
   {
       "status": "success"         ---------> string
   }
   ```

   In case of a **Bad Request**, like for instance, if the "email" is not present or "email" is duplicate, then you need to return the following response details:

   Response Code: **400**

   Response body:

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

4. Remove a lead

   In this API, you need to remove a lead. The ":lead_id" path parameter is mandatory and it is the id of the lead already saved in the database.

   DELETE /api/leads/:lead_id

   Response Code: **200**

   Response body:

   ```
   {
       "status": "success"         ---------> string
   }
   ```

   In case of a **Bad Request**, like for instance, if the ":lead_id" path parameter is not present, then you need to return the following response details:

   Response Code: **400**

   Response Body:

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

5. Mark a lead

   In this API, you need to update the status for the lead from `Created` to `Contacted` and also add the communication details in the request object. The ":lead_id" path parameter is mandatory and it is the id of the lead already saved in the database.

   PUT /api/mark_lead/:lead_id

   Request Header:

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

   Request body:

   ```
   {
       "communication": ""         ---------> required
   }
   ```

   Response Code: **202**

   Response body:

   ```
   {
       "status": "Contacted",
       "communication": ""         ---------> string(the communication that was present in the request object)
   }
   ```

   In case of a **Bad Request**, like for instance, if the ":lead_id" path parameter is not present, then you need to return the following response details:

   Response Code: **400**

   Response body:

   ```
   {
       "status": "failure",        ---------> string
       "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.

Submission Instructions

View My Submissions | Scroll to top

Step 1 Get Repo | Step 2(optional) Run Test Cases (Beta) | Step 3 Submission