Description | Submission | View Submission Instructions

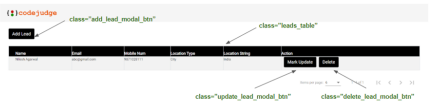
Exercises

1. **Display the leads**

   Display the leads in a table(element class name: **"leads_table"**) wherein each row will show the details of the lead (first_name, last_name, mobile, email, location_type (3 types: City, Zip and Country), location_string and two action buttons: "Mark Update"(element class name: **"update_lead_modal_btn"**) (see element details in the milestone 4) and "Delete"(element class name: **"delete_lead_modal_btn"**) (see element details in the milestone 3)

   Page Endpoint: /

   UI Elements:



   The request to get the leads is as follows:

   - REST Method: GET
   - API: /api/leads/?location_string=India
   - Response: List of Lead Objects

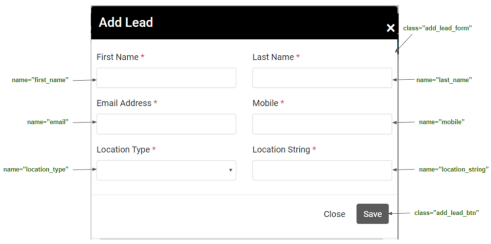
```
[
    {
        "id": 1,
        "updated_at": "2019-06-12T12:11:39.127842Z",
        "created_at": "2019-06-12T12:11:39.127901Z",
        "first_name": "Nilesh",
        "last_name": "Agarwal",
        "mobile": "9871828111",
        "email": "abc@gmail.com",
        "location_type": "City",
        "location_string": "India",
        "status": "Created",
        "communication": null,
        "tags": null
    }
]
```

2. **Add a Lead Using Modal Window**

   Create Add Lead Button (element class name: **"add_lead_modal_btn"**). On click of it, open a popup/modal with a form (element class name: **"add_lead_form"**) and capture the details about a Lead. The form will comprise the fields as shown below. Put form validation in the form wherein the "Save" button (element class name: **"add_lead_btn"**) should not be enabled unless all the fields are filled. Once all the fields are filled, the "Save" button should be enabled. **Please note that the "Location Type" field must be a component having "select" tag(element name :"location_type").**

   Page Endpoint: /

   UI Elements:



   On click of the "Save" button(element class name: **"add_lead_btn"**), make a backend request to save the details. The request should be as follows:

   - REST Method: POST
   - API: api/leads/
   - Request Object:

```
{
    "first_name": "Sal",
    "last_name": "Sasadsa",
    "mobile": "9871828333",
    "email": "nag@gmail.com",
    "location_type": "Country",
    "location_string": "India"
}
```

   Once the API returns a successful response, the popup/modal should get closed. Also, refresh the data in the table so that the lead that was just added gets displayed in the table.

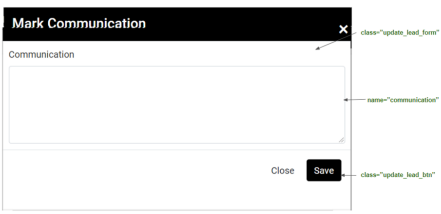
   NOTE: The data must be displayed in the same order as returned by the API response.

3. **Mark Update Communication for the Lead**

   On click of the "Mark Update" button, open a popup/modal with a form(element class name: **"update_lead_form"**). In that form, add a communication field (element name: **"communication"**) to record the communication that happened with the lead. Also, once it is saved, close the popup/modal and refresh the data in the table so that next time when the "Mark Update" button is clicked, the communication text that was saved gets displayed in the communication field.

   Page Endpoint: /

   UI Elements:



   On click of the "Save" button(element class name: **"update_lead_btn"**), you need to make a backend request in order to save the communication information. The request to update the communication for the lead is given below. The "id" parameter should be replaced with the actual id of the lead for which the communication is being updated:

   - REST Method: PUT
   - API: api/mark_lead/id (Ex: api/mark_lead/1)
   - Request Object:

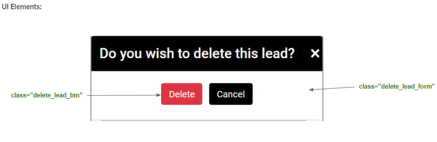
```
{
    "communication": "Good Lead"
}
```

4. **Delete a Lead**

   On click of the "Delete" button, open a popup/modal with a div container(element class name: **"delete_lead_form"**) which will be asking for confirmation before deleting the lead. The confirmation text will be: "Do you wish to delete this lead?" and the container will also have two buttons: "Delete" and "Cancel" buttons.

   Page Endpoint: /

   UI Elements:



   On click of the "Delete" button (element class name: **"delete_lead_btn"**), the lead must be deleted by making a backend request. The request to delete the lead is given below. The ":id" parameter should be replaced with the actual id of the lead for which the communication is being updated:

   - REST Method: DELETE
   - API: api/leads/:id/ (Ex. api/leads/1/)

   Once the API returns a successful response, the popup/modal should get closed.

Important Notes

- Please make sure that the UI and all the components are visible clearly in a 1280 X 800 resolution screen. If elements/components are not visible clearly (like they are getting cut from the screen, etc) then the test case might fail.
- Please start the Backend Service if the problem statement requires Backend Service API Integration. Use the endpoint URL(generated once the Backend Service gets deployed) in your project for local testing and API Integration. For instance, if the API given in the exercise is **"/leads/api/"**, then you need to make an API call at **"[endpoint URL]/api/leads/"** to get the data. Make sure that you go through the README.md file(that you will get after cloning the Git repository - mentioned below in the Submission Instructions) to understand how to use this endpoint url in your project.
- You can reset the Backend Data(i.e. bringing the Backend Data to the original state - when the Backend Service got deployed initially) by making a backend request to the endpoint URL(generated once the Backend Service gets deployed) as shown below:
  - REST Method: POST
  - API: /api/reset-db/
  - Request: {}
  - Response Code: 202 (on success).

  This will come in handy in cases wherein you added some data in the Backend as part of an Exercise and wish to remove that or reset the data to the original state
- Please note that when you execute a test case (mentioned under Step 2. Execute Test Cases section) by clicking on the "Run" button, we reset the Backend data first(provided the problem statement requires Backend Service API Integration) and then run the test case to ensure that the test case runs smoothly and there are no data-related issues.
- Also, after starting the live run session via the Utility(Desktop App), you would see a URL(ex.24jksnf.ngrok.io, etc) at the bottom bar of the Utility, kindly ignore that URL.
- Please find below some sample examples for how the elements should look after adding the class names, names, ids, etc. This is required so that the data can be put inside or extracted from these elements correctly:
  - *"Table" with class name as "leads_table":*
    `▼<table _ngcontent-scu-c2 class="table table-striped table-bordered leads_table mat-table" mat-table matsort matsortactive="name" role="grid" ng-reflect-data-source="[object Object]" ng-reflect-active="name"> == $0`
  - *"Select" box with name as "location_type":*
    `▼<select _ngcontent-gsx-c13 class="form-control ng-pristine ng-invalid ng-touched" name="location_type" required ng-reflect-required ng-reflect-name="location_type">…</select> == $0`
  - *"Input" text box with name as "email":*
    `<input _ngcontent-gsx-c13 class="form-control ng-pristine ng-invalid ng-touched" name="email" required type="email" ng-reflect-required ng-reflect-name="email"> == $0`
- Please note that you can submit the solution multiple times.

Submission Instructions

View My Submissions | Scroll to top

Step 1 Get Repo | Step 2 Start Backend Server | Step 3 Connect Backend Server | Step 4(optional) Run Test Cases Beta | Step 5 Submission