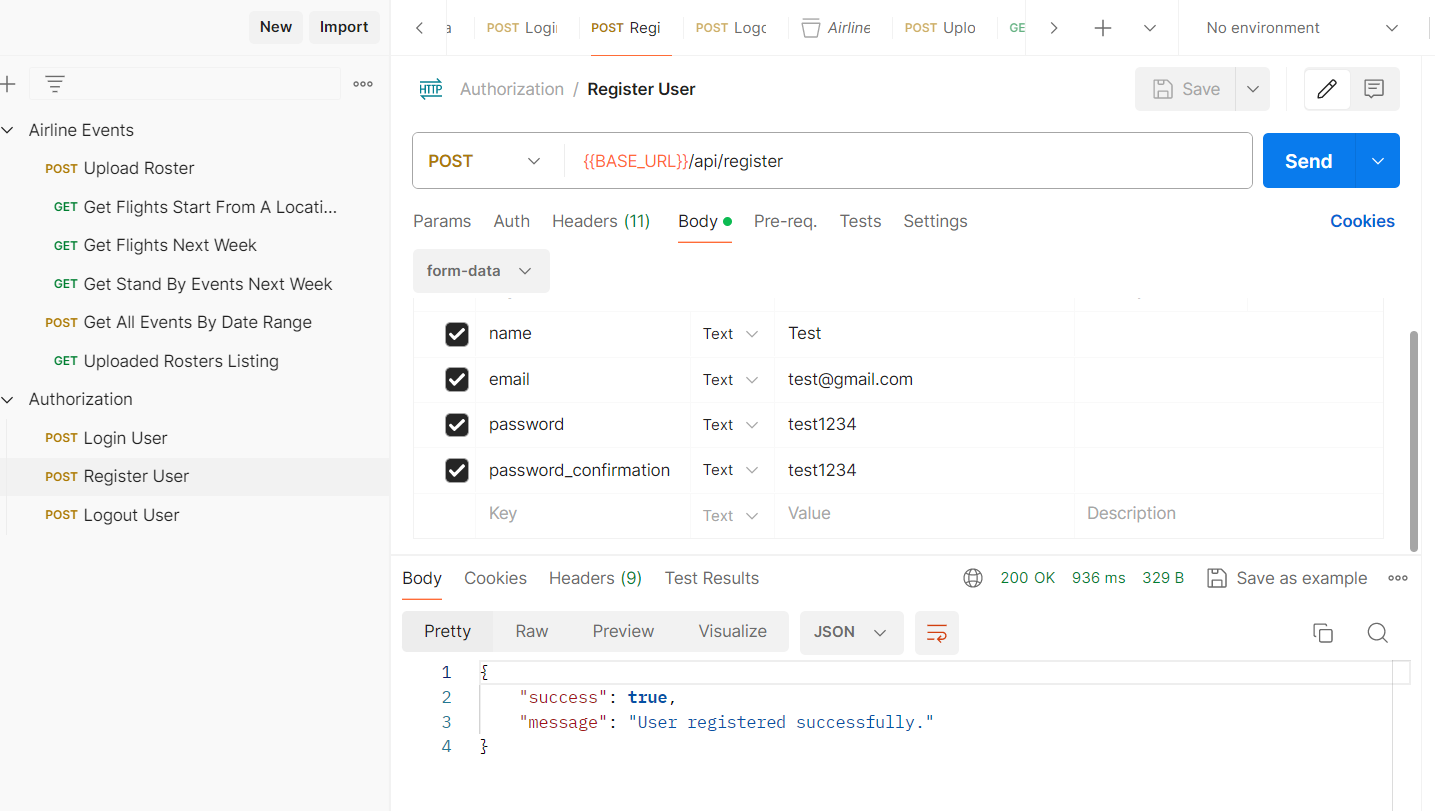
**API Document**

We have implemented token-based authentication for accessing API endpoints. So, to start with accessing APIs, first we need to register user using “Register User” API, then need to get the access token using “Login User” API with same credentials.

**API - Register User:**



**API Endpoint**: <http://localhost:8000/api/register>

**Method**: POST

**Headers**:

Accept: application/vnd.api+json

Content-Type: application/vnd.api+json

**Body Form Data:**

name: "Test"

email: "test@gmail.com"

password: "test1234"

password\_confirmation: "test1234"

**Response**:

{

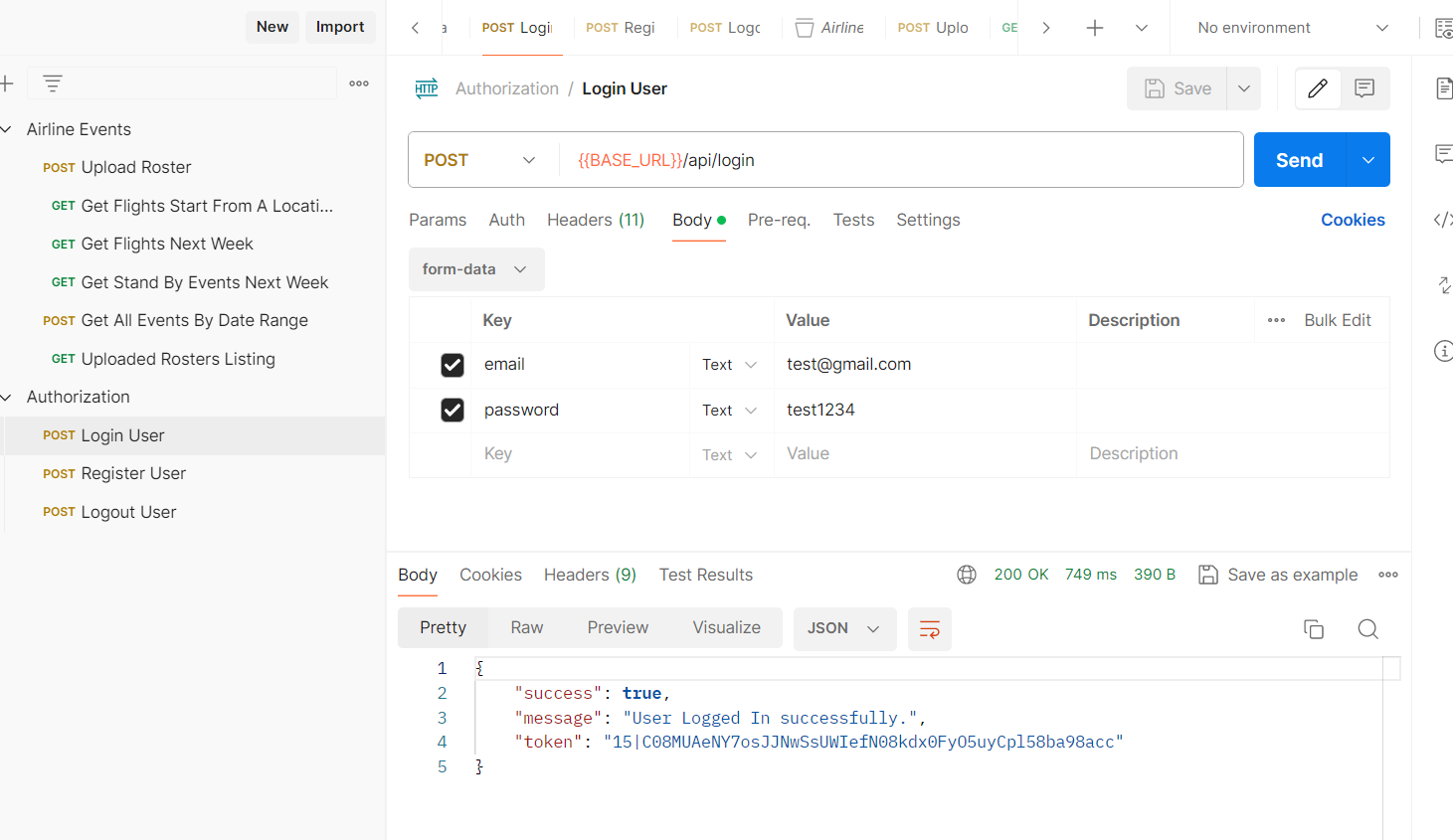
    "success": **true**,

    "message": "User registered successfully."

}

Once user get registered, we should use login API to get access token.

**API - Login User:**



**API Endpoint**: <http://localhost:8000/api/login>

**Method**: POST

**Headers**:

Accept: application/vnd.api+json

Content-Type: application/vnd.api+json

**Body Form Data:**

email: "test@gmail.com"

password: "test1234"

**Response**:

{

    "success": **true**,

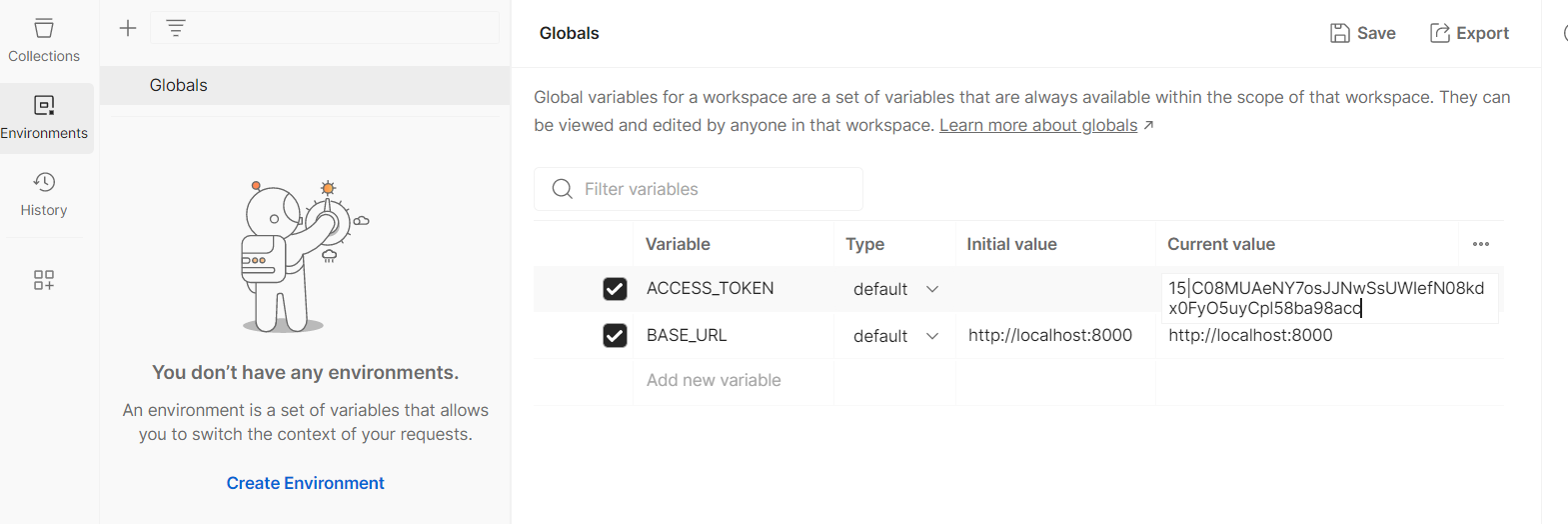
    "message": "User Logged In successfully.",

    "token": "15|C08MUAeNY7osJJNwSsUWIefN08kdx0FyO5uyCpl58ba98acc"

}

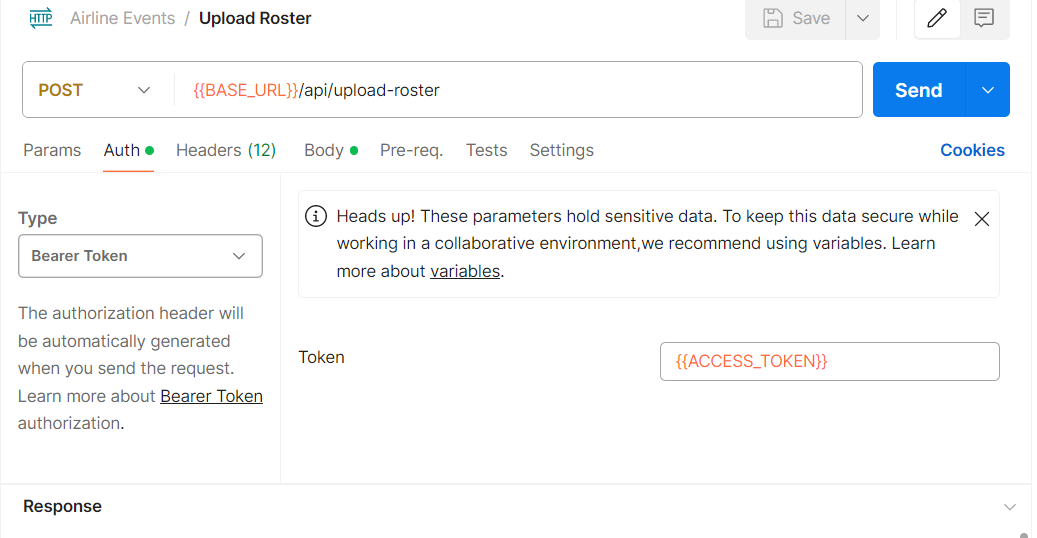
We need to use this yellow highlighted token to access other API endpoints.

**Note: If we are testing using postman, then we can set {{ACCESS\_TOKEN}} and {{BASE\_URL}} in global environment variables.**

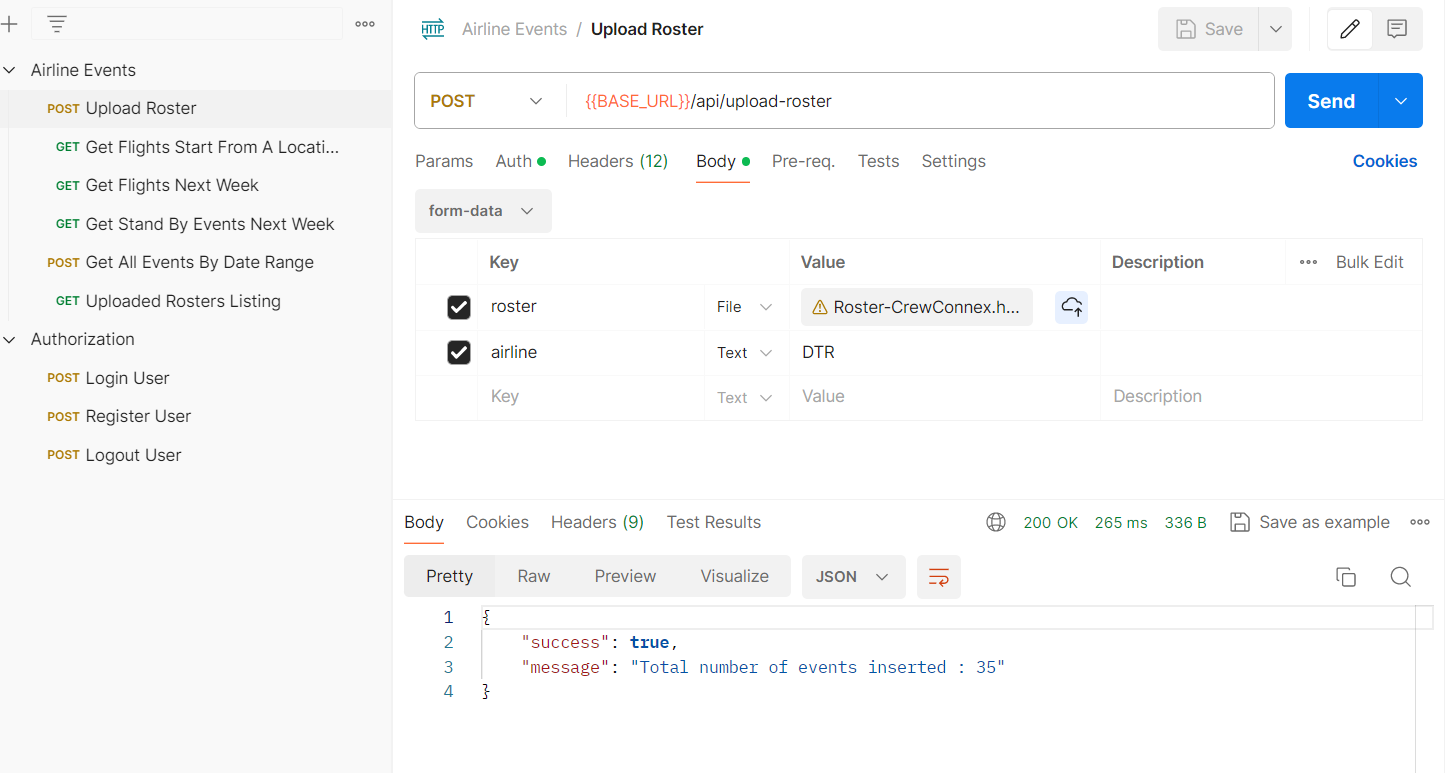


We need to set Auth as “Bearer token” in postman, and provide the value of token that we received from login API.

Note: We can manually put the access token for every API in Auth section, or we can create the environment variable named ‘ACCESS\_TOKEN’ for this token and can use that variable into API auth section.



**API - Upload Roster:**



**API Endpoint**: <http://localhost:8000/api/upload-roster>

**Method**: POST

**Headers**:

Accept: application/vnd.api+json

Content-Type: application/vnd.api+json

Authorization: Bearer {{ACCESS\_TOKEN}}

**Body Form Data:**

**roster**: Roster-CrewConnex.html

**airline**: DTR

// \*\*\***roster**: roster is file field, where we need to upload roster \*\*\*

// \*\*\* **airline**: This is added so that we can parse HTML file according to given airline layout. For now, we are assuming that the given roster file is belongs to “DTR” airline, so for testing we need to give “DTR” in **airline** parameter with upload file, otherwise it won’t parse the file assuming we are not supporting other airlines layout for now. Once we provide the correct parameters, we will see the number of events inserted into database in the response. \*\*\*

**Response**:

{

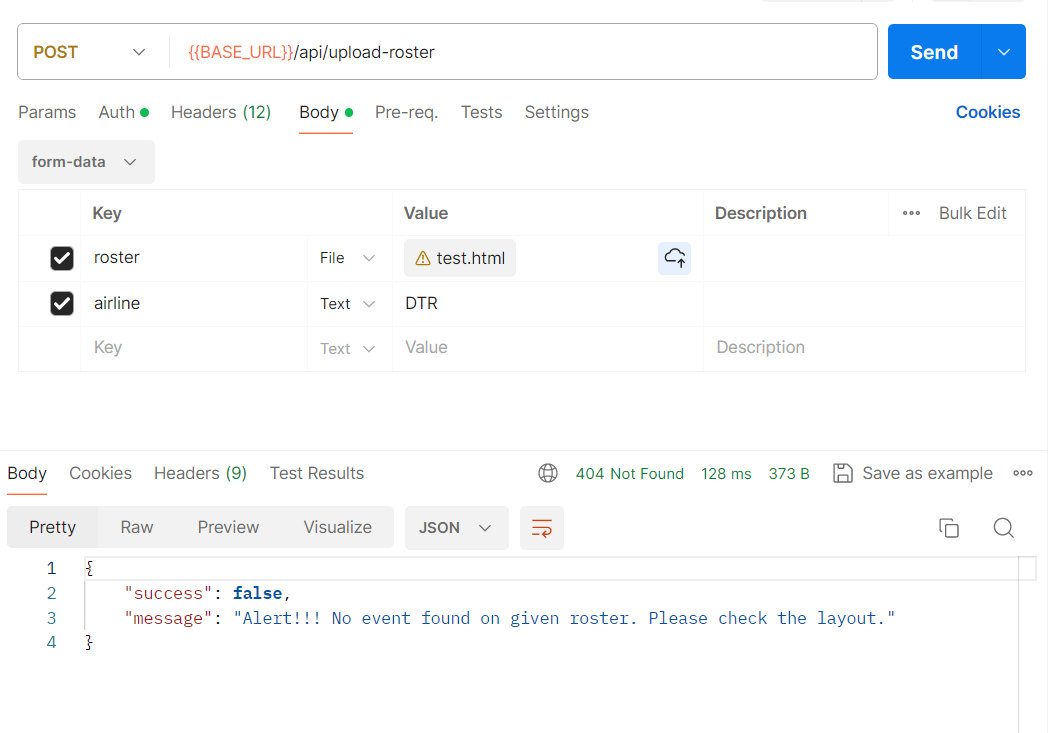
    "success": **true**,

    "message": "Total number of events inserted: 35"

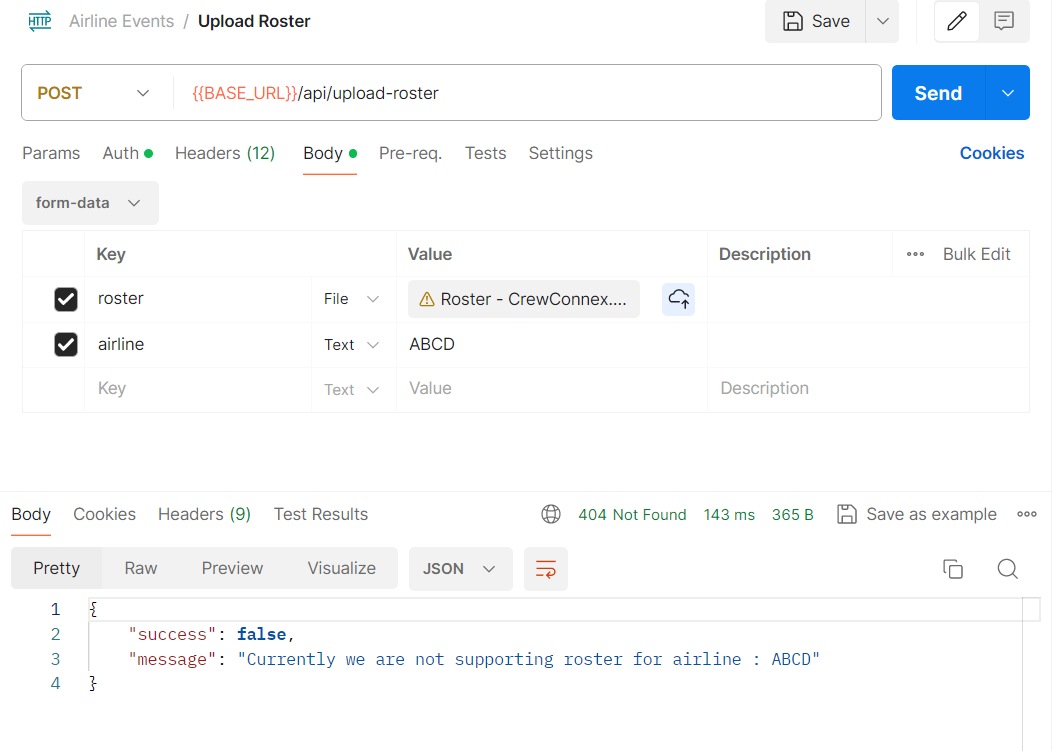
}

**Case: HTML file uploaded is not as per the given format (Invalid HTML file).**

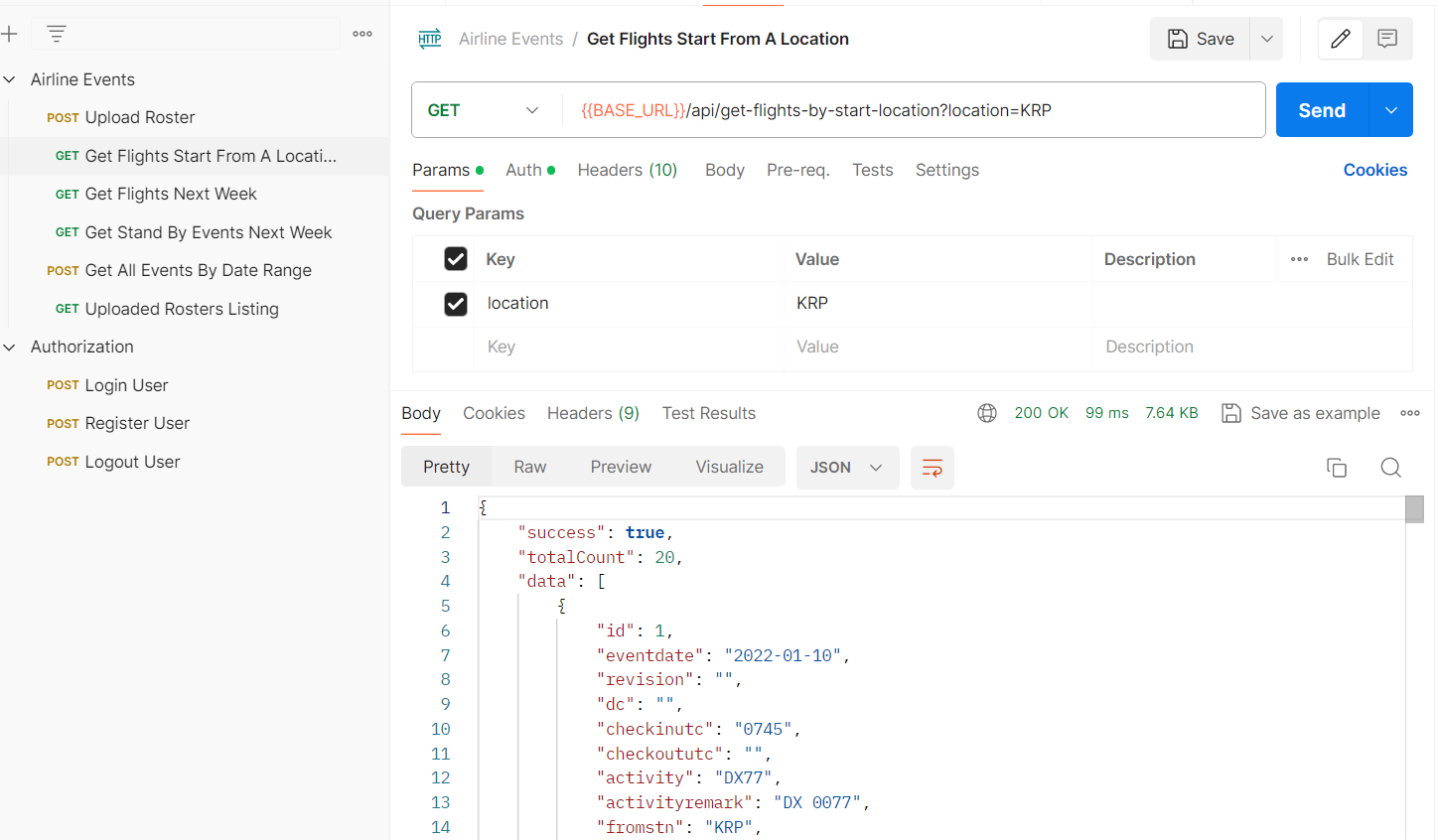
**If someone upload the incorrect HTML file, then we won’t find the activities/events on the uploaded file. So, we will provide the alert response as follows:**



**Case: If someone try to upload the file for different airline that we are not supporting (other than DTR for now), then we will show message that we are not parsing for the given airline. Once we update parsing feature for other airlines, then we will validate/process roster file accordingly.**

****

**API - Get Flights Start From A Location:**



**API Endpoint**: <http://localhost:8000/api/get-flights-by-start-location>

**Method**: GET

**Headers**:

Accept: application/vnd.api+json

Content-Type: application/vnd.api+json

Authorization: Bearer {{ACCESS\_TOKEN}}

**Body Form Data:**

location: "KRP"

**Response**:

{

"success": true,

"totalCount": 1,

"data": [

{

"id": 1,

"eventdate": "2022-01-10", // we can return date format accordingly

"revision": "",

"dc": "",

"checkinutc": "0745", // we can return time format accordingly

"checkoututc": "",

"activity": "DX77",

"activityremark": "DX 0077",

"fromstn": "KRP",

"stdutc": "0845",

"tostn": "CPH",

"stautc": "0935",

"achotel": "DO4",

"blockhours": "",

"flighttime": "",

"nighttime": "",

"duration": "",

"ext": "",

"paxbooked": "",

"acreg": "OYJRY",

"user\_id": 1, // this is the id who uploaded the roster

"roster\_id": "1", // this is id for uploaded roster (saved in rosters table)

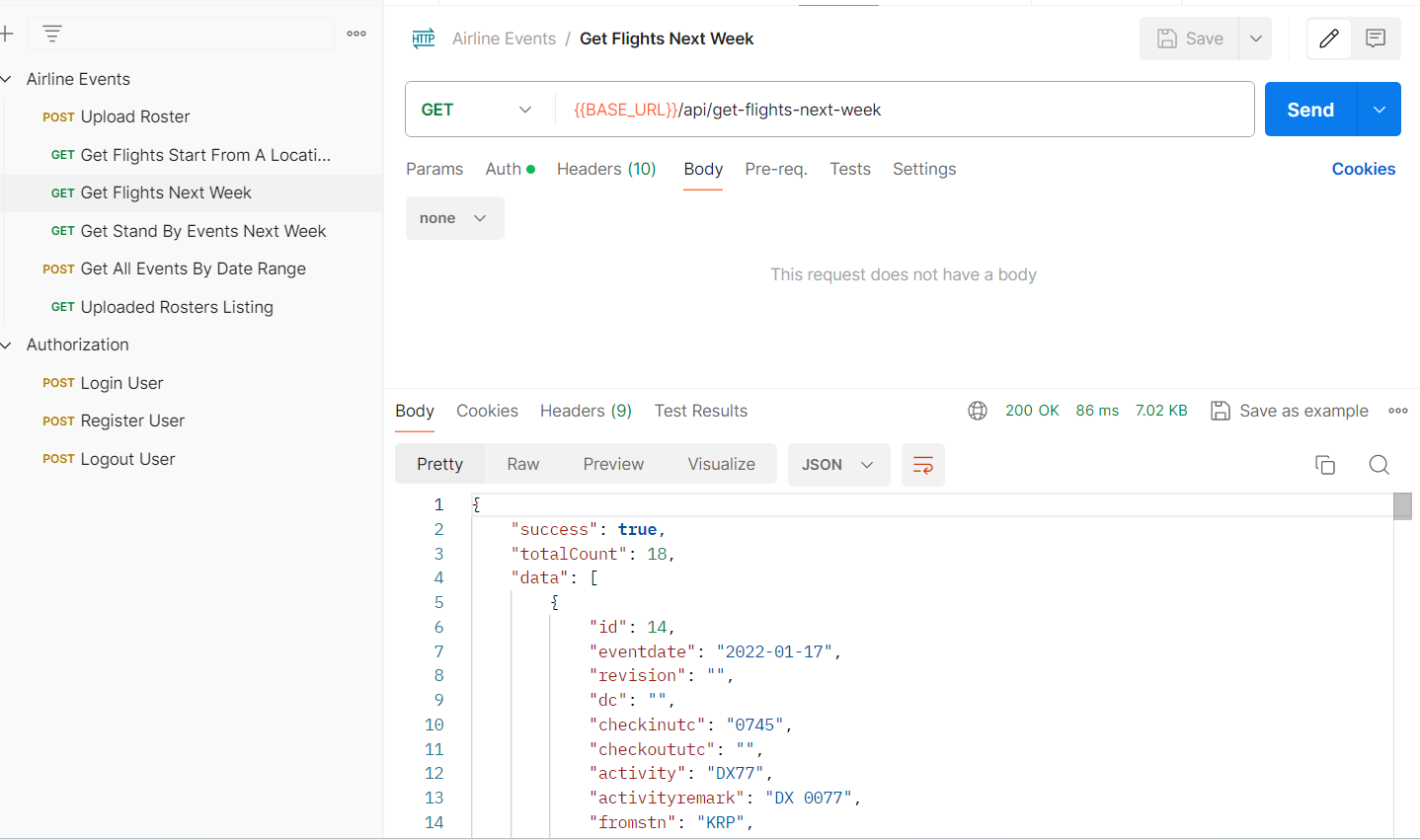
"created\_at": "2024-04-05 15:58:43"

}

]

}

**API - Get Flights Next Week:**



**API Endpoint**: <http://localhost:8000/api/get-flights-next-week>

**Method**: GET

**Headers**:

Accept: application/vnd.api+json

Content-Type: application/vnd.api+json

Authorization: Bearer {{ACCESS\_TOKEN}}

**Body Form Data:**

NA

**Response**:

{

"success": true,

"totalCount": 18,

"data": [

{

"id": 14,

"eventdate": "2022-01-17",

"revision": "",

"dc": "",

"checkinutc": "0745",

"checkoututc": "",

"activity": "DX77",

"activityremark": "DX 0077",

"fromstn": "KRP",

"stdutc": "0845",

"tostn": "CPH",

"stautc": "0935",

"achotel": "DO4",

"blockhours": "",

"flighttime": "",

"nighttime": "",

"duration": "",

"ext": "",

"paxbooked": "",

"acreg": "OYJRY",

"user\_id": 1,

"roster\_id": "1",

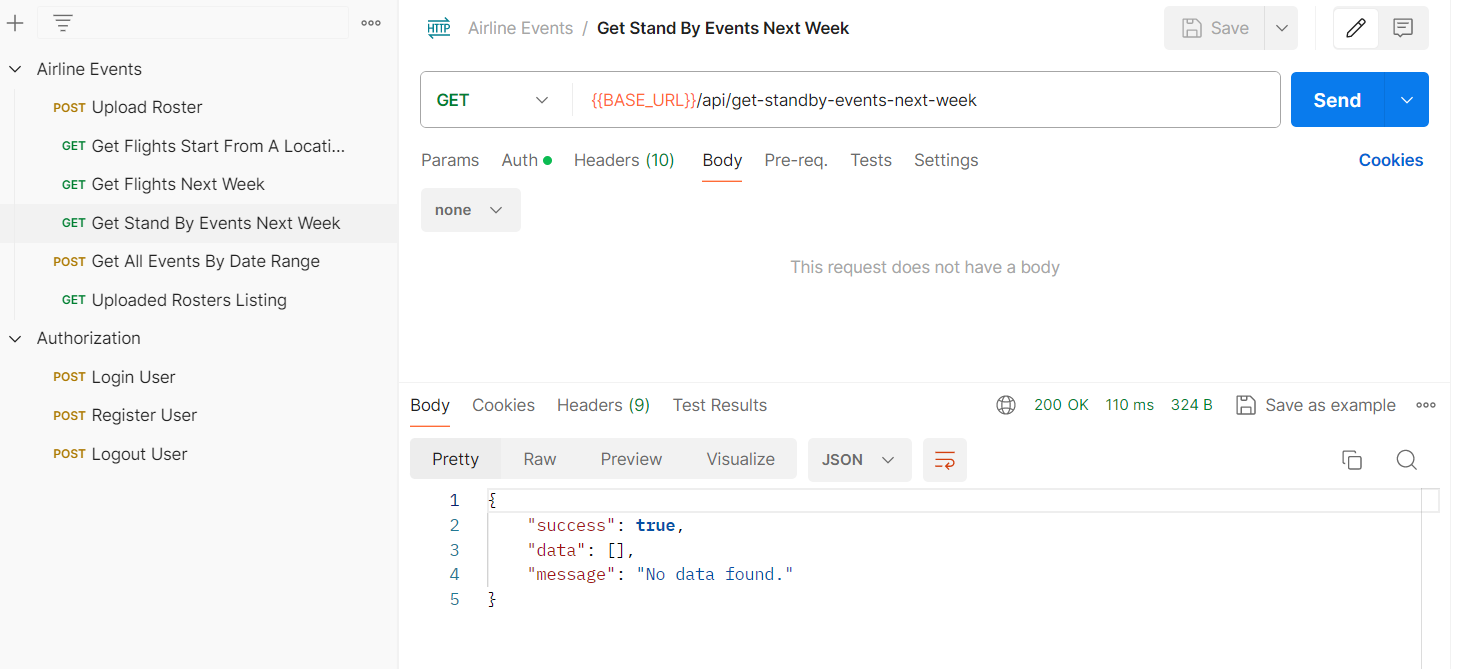
"created\_at": "2024-04-05 15:58:43"

}

]

}

**API - Get Stand By Events Next Week:**

****

**API Endpoint**: <http://localhost:8000/api/get-standby-events-next-week>

**Method**: GET

**Headers**:

Accept: application/vnd.api+json

Content-Type: application/vnd.api+json

Authorization: Bearer {{ACCESS\_TOKEN}}

**Body Form Data:**

NA

**Response**:

{

    "success": **true**,

    "data": [],

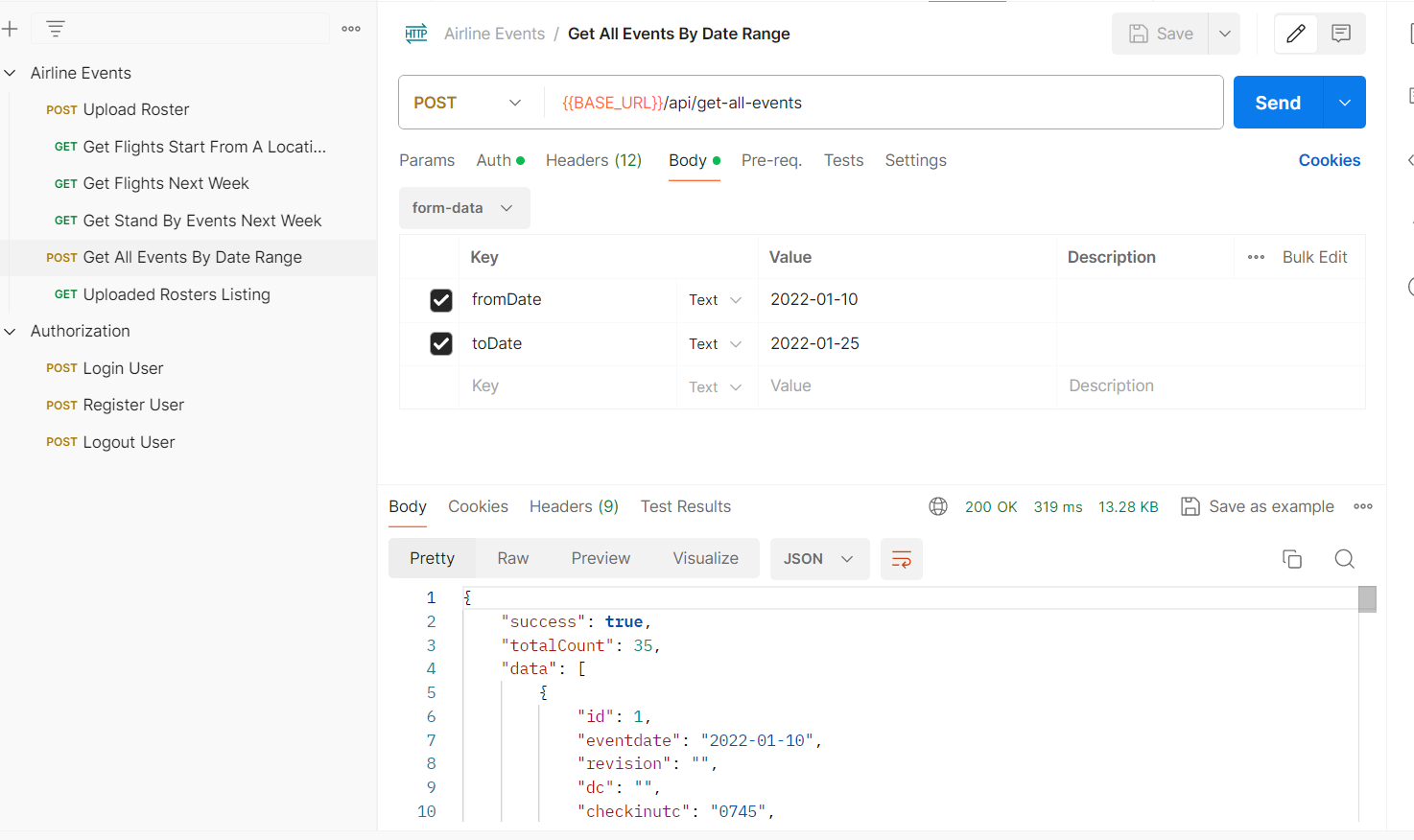
    "message": "No data found."

}

**\*\*\* Note: Since we are taking ’14 Jan 2022’ as current date, so next week (17 Jan – 23 Jan), we don’t have any stand by event. We are assuming next week as starting from next Monday. \*\*\***

**Also, since in activity table only day is showing in Date field like ‘Mon 10’, we are assuming month as “Jan” and Year as ‘2022’ from current date to store the event date in db.**

**API - Get All Events By Date Range:**



**API Endpoint**: <http://localhost:8000/api/get-all-events>

**Method**: POST

**Headers**:

Accept: application/vnd.api+json

Content-Type: application/vnd.api+json

Authorization: Bearer {{ACCESS\_TOKEN}}

**Body Form Data:**

**fromDate**: "2022-01-10"

**toDate**: "2022-01-25"

**Response**:

{

"success": true,

"totalCount": 35,

"data": [

{

"id": 1,

"eventdate": "2022-01-10",

"revision": "",

"dc": "",

"checkinutc": "0745",

"checkoututc": "",

"activity": "DX77",

"activityremark": "DX 0077",

"fromstn": "KRP",

"stdutc": "0845",

"tostn": "CPH",

"stautc": "0935",

"achotel": "DO4",

"blockhours": "",

"flighttime": "",

"nighttime": "",

"duration": "",

"ext": "",

"paxbooked": "",

"acreg": "OYJRY",

"user\_id": 1,

"roster\_id": "1",

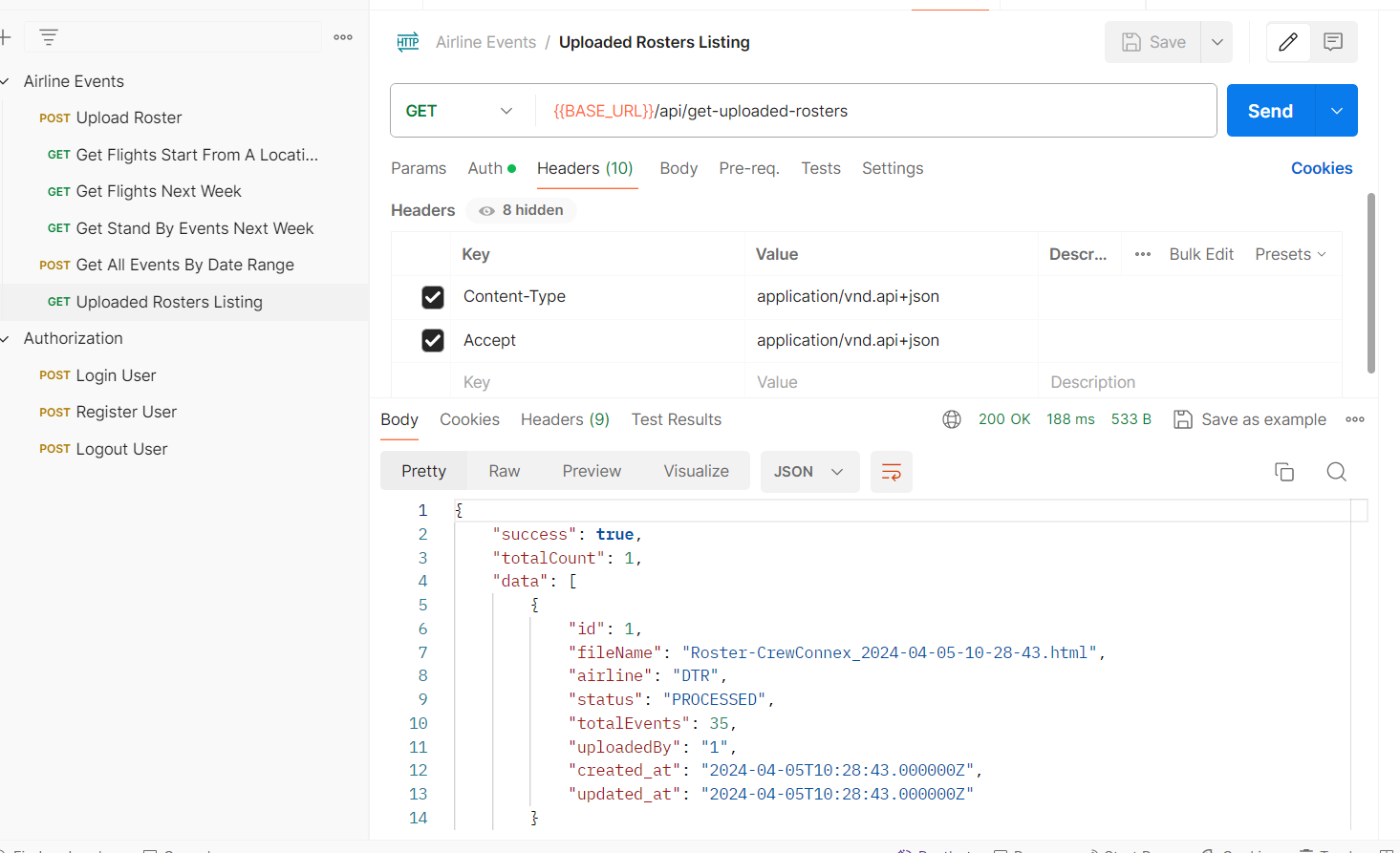
"created\_at": "2024-04-05 13:21:30"

}

]}

**API - Uploaded Rosters Listing:**

**Note: The purpose of this API is to track the uploaded files. If a file is got uploaded, then It’s status will be ‘UPLOADED’. If a file is parsed successfully and event get saved into database, then it’s status will become ‘PROCESSED’. With the help of this API, we can track the uploaded file record.**



**API Endpoint**: <http://localhost:8000/api/get-uploaded-rosters>

**Method**: GET

**Headers**:

Accept: application/vnd.api+json

Content-Type: application/vnd.api+json

Authorization: Bearer {{ACCESS\_TOKEN}}

**Body Form Data:**

NA

**Response**:

{

"success": true,

"totalCount": 1,

"data": [

{

"id": 1,

"fileName": "Roster-CrewConnex\_2024-04-05-10-28-43.html",

"airline": "DTR",

"status": "PROCESSED",

"totalEvents": 35,

"uploadedBy": "1",

"created\_at": "2024-04-05T10:28:43.000000Z",

"updated\_at": "2024-04-05T10:28:43.000000Z"

}

]

}

Note:

**filename**: Uploaded file name with timestamp.

**airline**: Since we will be having roster from different airlines, we can pass airline code or id to segregate the records.

**Status**: Uploaded file status (UPLOADED/PROCESSED).

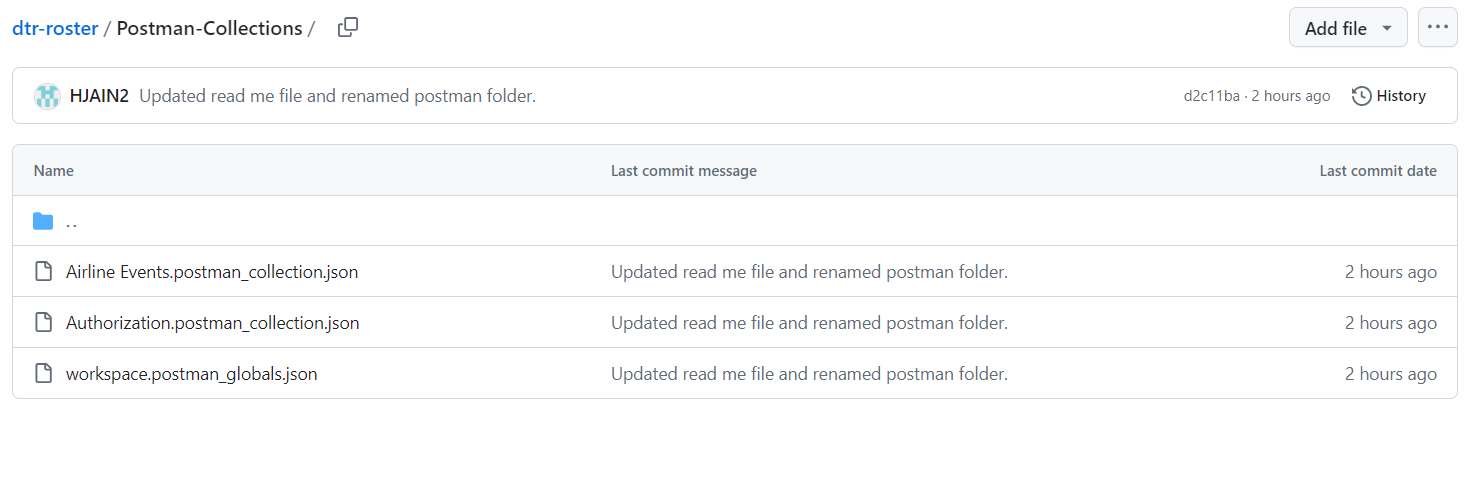
**totalEvents**: No of activities/events inserted by the uploaded roster.

**uploadedBy**: User id from authentication, who uploaded the roster.

**Guideline**:

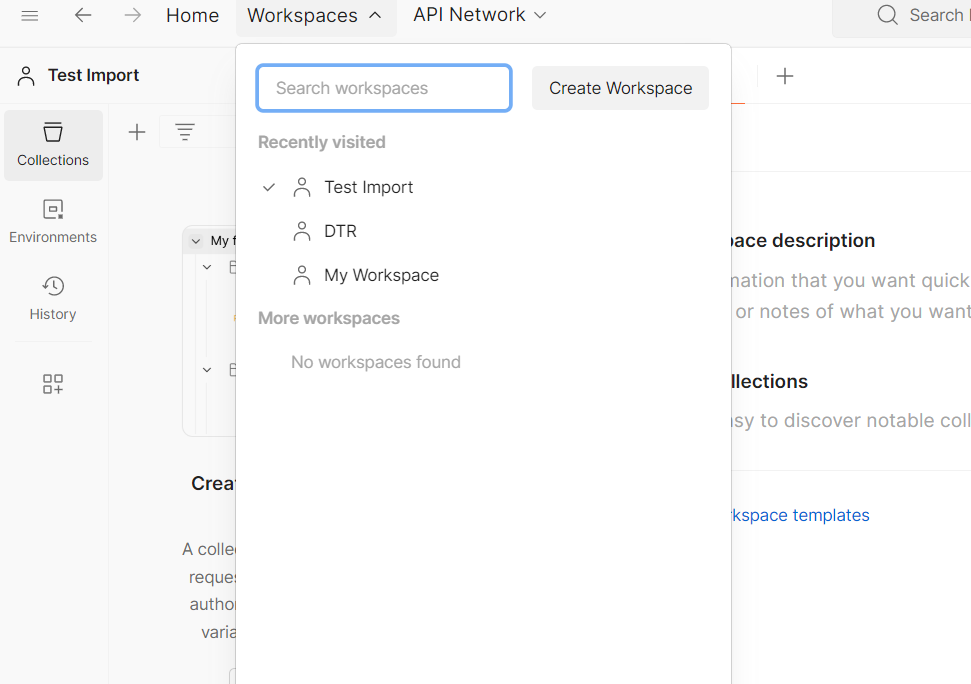
For testing the APIs, we have shared the API collections here:

<https://github.com/harshitjaindev/dtr-roster/tree/main/Postman-Collections>

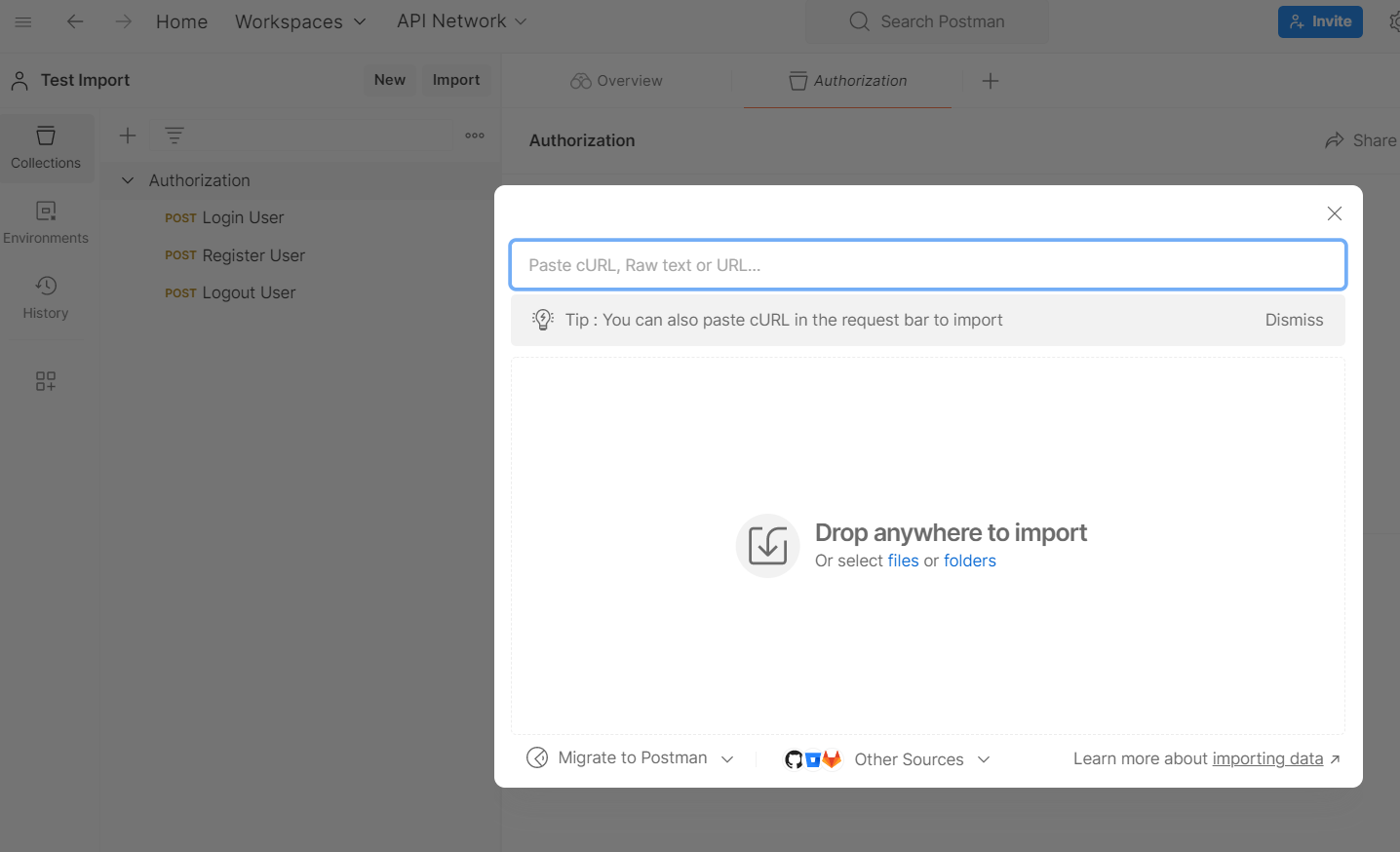


We need to import the collections and globals files in postman.

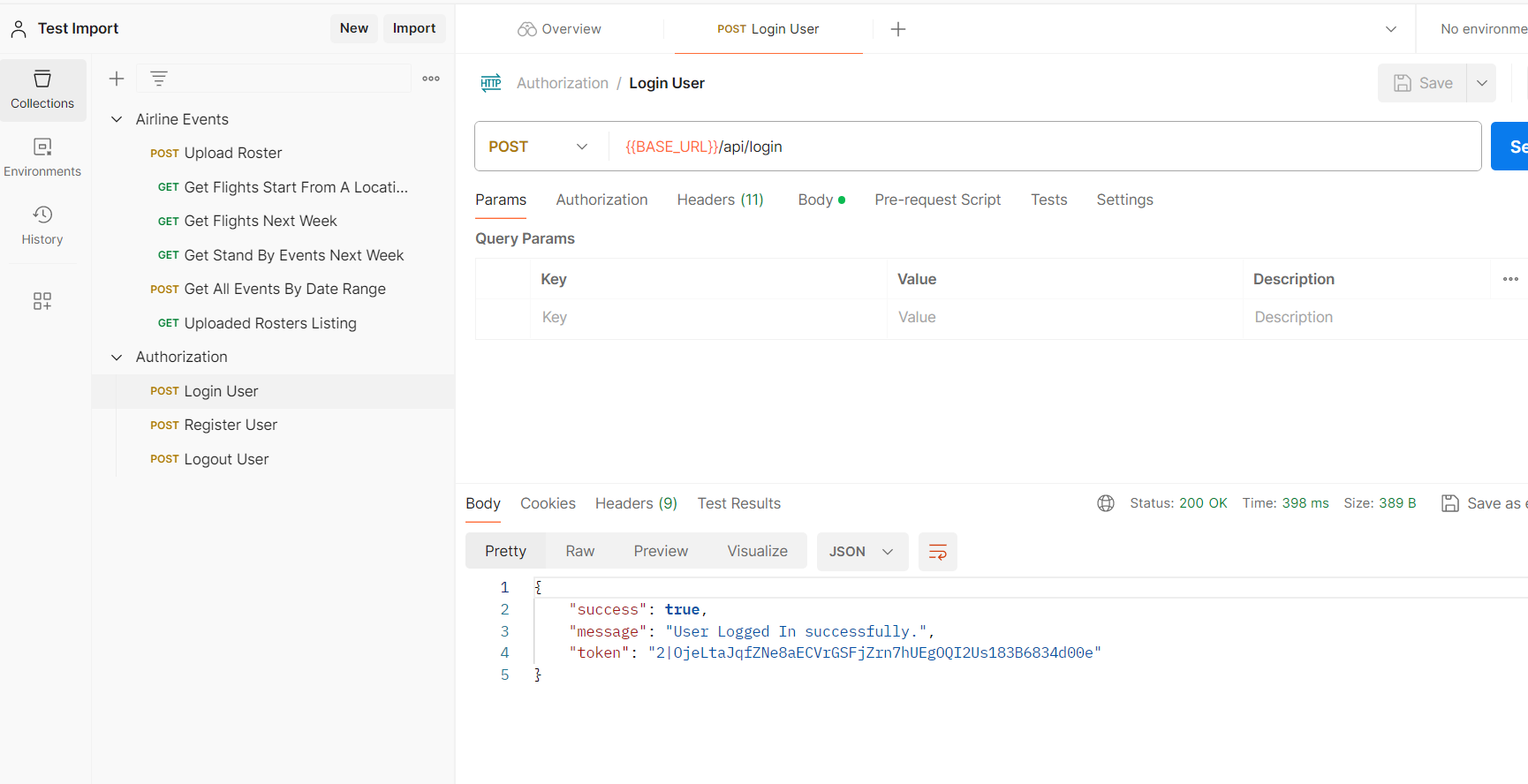
Step 1. Create a New workspace in postman using “**Create Workspace**”.



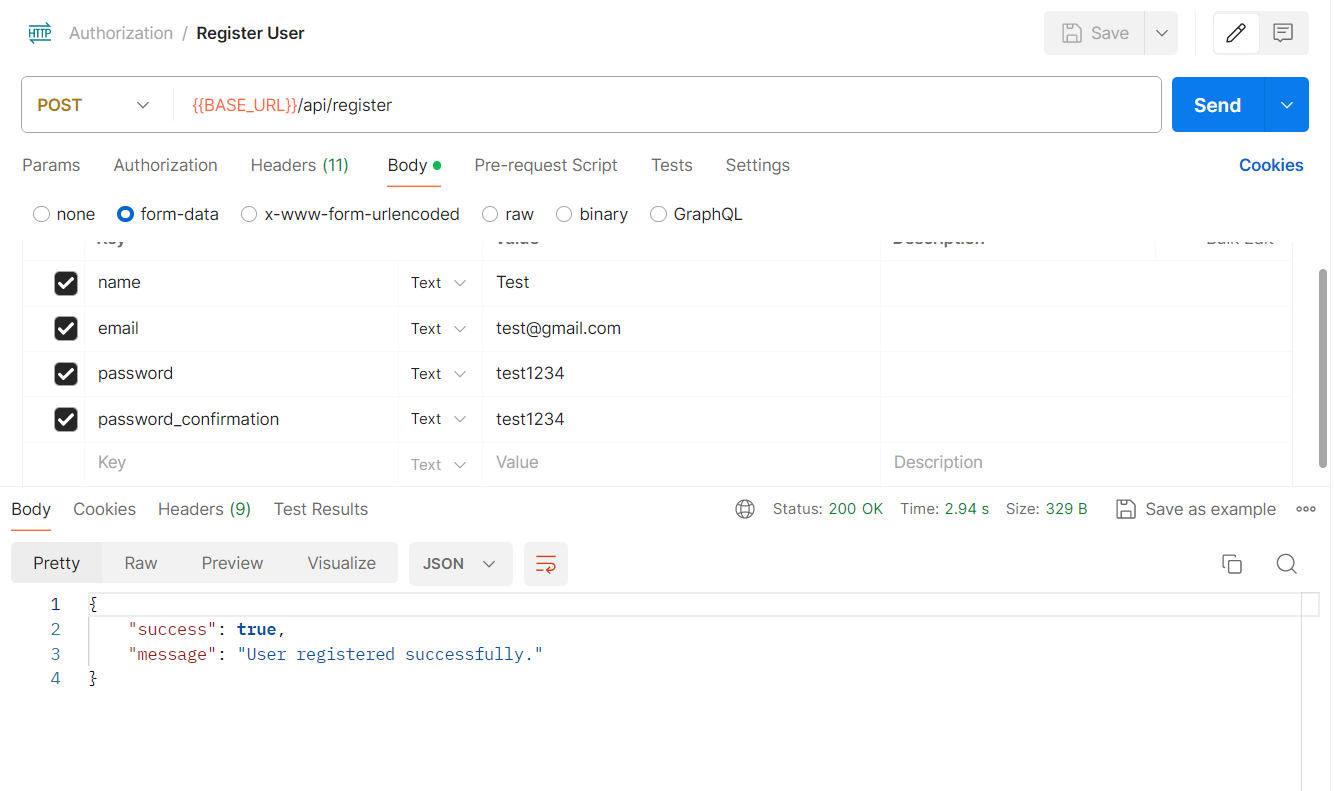
2. Once workspace is being created, kindly click on “Import button”. It will ask to select the file for import, we need to provide all the 3 files one by one.



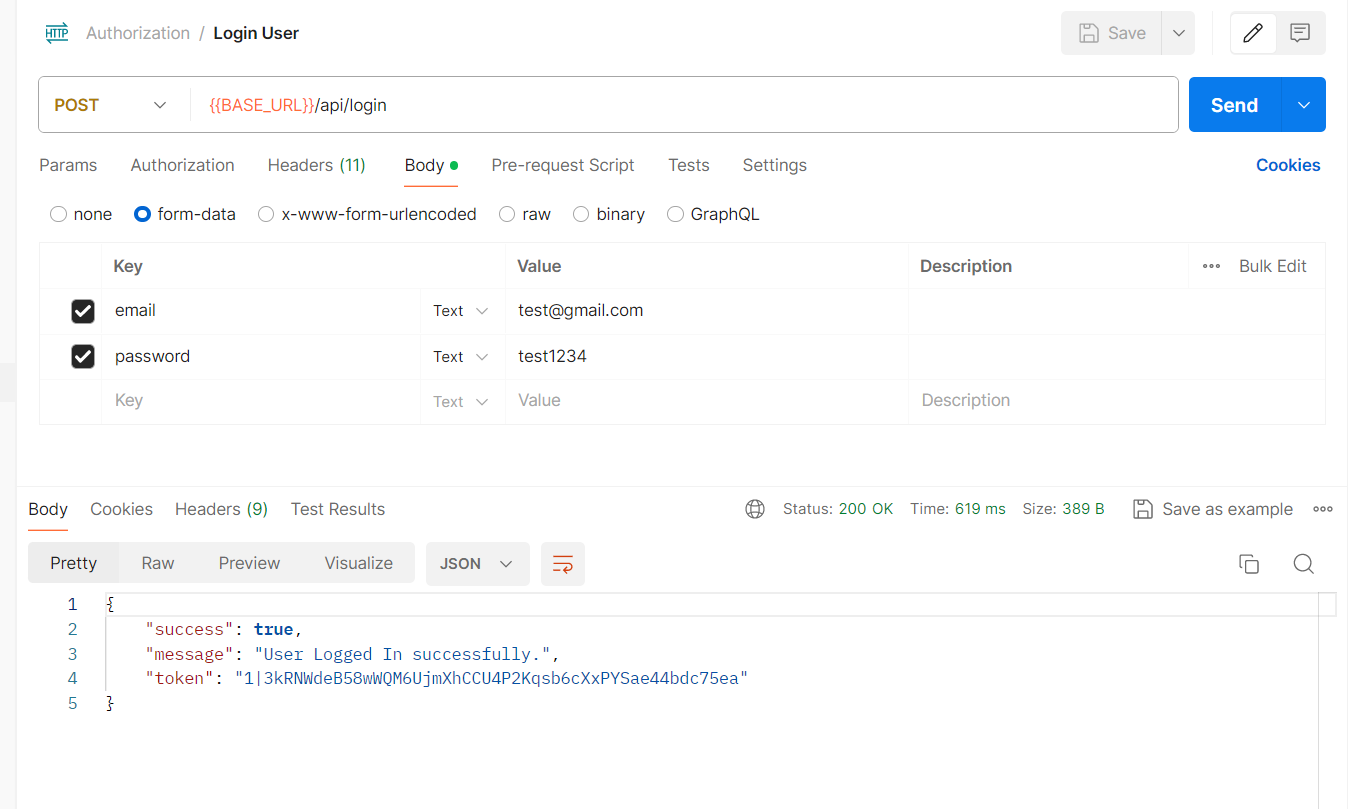
Step 3. After successful import, we can see all the APIs inside workspace.



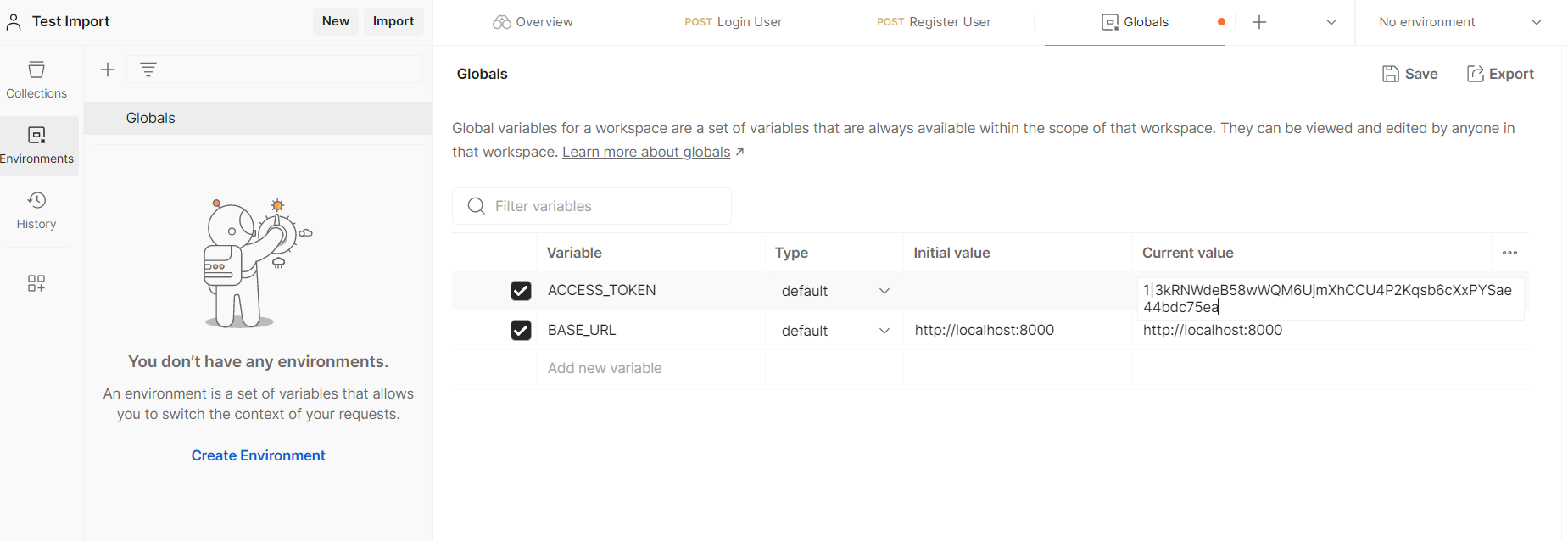
4. We need to go to “Register User” API to register a user.



5. Once we got registered, we need to go to “Login User” API and we need to trigger the API to get the token:



6. Last step is to go to Environment > Global and update the variables (token, url) accordingly.



**ACCESS\_TOKEN** : We need to place the generated token here.

**BASE\_URL** : We need to update our base URL here. (Ex: <http://localhost:8000>)

Now we can test all the APIs.

**Note: Make sure that we are connected to Database and already run the database migration.**