***Insta Web API Documentation***

Contents

[Purpose 3](#_Toc48206117)

[Audience 3](#_Toc48206118)

[Notations 3](#_Toc48206119)

[General information of all the APIs 3](#_Toc48206120)

[API Documentation Structure 3](#_Toc48206121)

[Status Codes 5](#_Toc48206122)

[Section 1 – Authentication 5](#_Toc48206123)

[Login API 5](#_Toc48206124)

[URL 5](#_Toc48206125)

[Request Type 5](#_Toc48206126)

[Description 6](#_Toc48206127)

[Prerequisites 6](#_Toc48206128)

[Input 6](#_Toc48206129)

[Output 6](#_Toc48206130)

[Comments/Assumptions 7](#_Toc48206131)

[Status Codes 7](#_Toc48206132)

[Test Cases 7](#_Toc48206133)

[Refresh Token API 7](#_Toc48206134)

[URL 7](#_Toc48206135)

[Request Type 7](#_Toc48206136)

[Description 7](#_Toc48206137)

[Prerequisites 8](#_Toc48206138)

[Input 8](#_Toc48206139)

[Output 8](#_Toc48206140)

[Comments/Assumptions 8](#_Toc48206141)

[Status Codes 9](#_Toc48206142)

[Test Cases 9](#_Toc48206143)

[Section 2 – Hotel Inventory APIs 10](#_Toc48206144)

[Get Meeting Rooms API 10](#_Toc48206145)

[URL 10](#_Toc48206146)

[Request Type 10](#_Toc48206147)

[Request Parameters 10](#_Toc48206148)

[Prerequisites 10](#_Toc48206149)

[Description & Implementation 10](#_Toc48206150)

[Response 11](#_Toc48206151)

[Implementation Note - 11](#_Toc48206152)

[Status Codes 12](#_Toc48206153)

[Test Cases 12](#_Toc48206154)

[Example - **Request** 12](#_Toc48206155)

[Example – **Response** 13](#_Toc48206156)

[GET Hotel API 14](#_Toc48206157)

[URL 14](#_Toc48206158)

[Request Type 14](#_Toc48206159)

[Request Parameters 14](#_Toc48206160)

[Prerequisites 14](#_Toc48206161)

[Description and Implementation 14](#_Toc48206162)

[Status Codes 15](#_Toc48206163)

[Test Cases 16](#_Toc48206164)

[Events API 17](#_Toc48206165)

[Get Events 17](#_Toc48206166)

[URL 17](#_Toc48206167)

[Request Type 17](#_Toc48206168)

[Request Parameters 17](#_Toc48206169)

[Prerequisite 17](#_Toc48206170)

[Description and Implementation 17](#_Toc48206171)

[Status Codes 19](#_Toc48206172)

[Test cases 19](#_Toc48206173)

[Occupancy API 20](#_Toc48206174)

[Get Occupancy 20](#_Toc48206175)

[URL 20](#_Toc48206176)

[Request Type 20](#_Toc48206177)

[Request Parameters 20](#_Toc48206178)

[Prerequisite 20](#_Toc48206179)

[Description and Implementation 21](#_Toc48206180)

[Status Codes 22](#_Toc48206181)

[Test cases 22](#_Toc48206182)

[SET Occupancy 23](#_Toc48206183)

[URL 23](#_Toc48206184)

[Request Type 23](#_Toc48206185)

[Prerequisite 23](#_Toc48206186)

[Description and Implementation 23](#_Toc48206187)

[Post Fields 23](#_Toc48206188)

[Response Body 24](#_Toc48206189)

[Status Codes 24](#_Toc48206190)

[Test cases 25](#_Toc48206191)

# Purpose

The purpose of this documentation is to document all the APIs for instagathering and its input and output parameters plus the path to the API

# Audience

The audience for this document is

1. Client developers- Client developers can look at this documentation and figure out the details of the URL, input and output parameters
2. Reviewers of API – Reviewers can quickly look at this document to see if the design of a particular API is correct or not
3. API Developer – Helps the API developer understand what design we are following throughout and add additional ones following the design and send for review of the design either individually or may be after implementation.

**Note – API Developers have to fill in the information properly here or else his code review will fail**

# Notations

|  |  |
| --- | --- |
| Notation | Meaning |
| HOST\_URL | This means the base url of web api hosting. This could easily change between QA and production environments so any API documentation would refer base url as HOST\_URL |
| {<<input>>} | This notation will be used to indicate input variables and depending on the call the values might be different as sent in by the user of the API |

# General information of all the APIs

The input and output of the API will be JSON based as this is a good format which is used by web and mobile apps.

# API Documentation Structure

The document will be divided into various sections of API like

1. Authentication
2. Hotel API
3. Event API etc

The API will have the following structure

1. URL of the API

This will give the URL of the API with notation like HOST\_API for base url of the API etc. If any developer wants to add a notation they can add it there and then add the notation in the notations table above

Also this part will describe any dynamic part of the URl like {{hotel\_id}} is the hotel id of an existing hotel

1. Request Type

POST/PUT/GET/PATCH/DELETE

Basically the type of request. To decide on which type of request it should be we can use the below link and try to fit it in the right method

<https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them>

1. Description of the API

This will basically describe the API in a very detailed manner. The main thing to discuss here is the functionality of the API and any nuances of the API plus any known issues

1. Pre requisites

This will describe what is the pre requisite for the API eg – user needs to be logged in and should have a token or may be hotel with id {{id}} should exists for an API which is inside some id etc etc

1. Input

The detailed description if inputs to an API . If the developer wants they can copy paste the structure from the code and explain all the individual properties. This will contain many subsections for each data type taken as input to the API

1. Output

The detailed description of the output of the API. Again developer can copy paste the structure maintaining the format od the document well

1. Comments/Assumptions

This can be easily empty but then if you want to add the right way to call the API you can do so here. Also Assumptions if any you are making.

1. Status codes

Status codes possibly returned by the API. The status codes will be described in the next section

1. Test cases

This will describe what and all unit tests you will apply to the API and make sure they run in the pipelines. Basically both success and failure test cases needs to be applied . This should also describe the return value expected etc

# Status Codes

Status codes from the API needs to be really consistent across API so that client can put in generic handlers for status codes in many cases. I think we should follow this documentation for the status codes returned

<https://restfulapi.net/http-status-codes/>

Remember not all status codes are valid for all APIs so we can describe the applicable ones in the status codes section in the API description

# Section 1 – Authentication

## Login API

### URL

https://{{HOST\_URL}}/api/login

### Request Type

POST – the request type is a POST method as the details will be in the message part of the body

### Description

The API communicates with identity server which is the authorization server and gets it authenticated. Claims are provided as part of the JWT token returned by the API which can be used in any further calls to Insta API.

Note about JWT token – InstaAPI has a particular clientid and secret key which is provided by the identity server for communication. Any API in InstaAPI will understand only tokens targeted for that client id and secret key as the JWT is signed with the secret key of the InstaAPI.

For example – If GRM API gets to identity server and gets a token for its functionality, the same token will not be recognized by the Insta API.

The result of the call will have three keys if the authentication/authorization is successful.

1. Access\_token – Contains the JWT token as provided by the identity server
2. Token\_type – This will always have the value “bearer” indicating this is a bearer token
3. Expires\_in – This will give out the expiry of the token. Every token will have expiry time.
4. Refresh\_token – This is the refresh token which can be provided to the refresh token api . This token can be passed as a token to refresh token api which is described next

### Prerequisites

User should be registered in the system prior to calling this API either through UI or through registration API

### Input

The input will have the following structure

LoginModel {

Username:<<user name>>

Password:<<password>>

}

User Name – Mandatory – user name of the user

Password – Mandatory- Encrypted password of the user (encryption algorithm to be updated)

Sample Json

{

    "UserName":"rammanohar",

    "Password":"Inncrewin@123"

}

### Output

The result of the call will have three keys if the authentication/authorization is successful. The Status code will be 200(OK).

1. Access\_token – Contains the JWT token as provided by the identity server
2. Token\_type – This will always have the value “bearer” indicating this is a bearer token
3. Expires\_in – This will give out the expiry of the token. Every token will have expiry time.

Failure output will return an error http status code and body most of the times will contain a error and an error description as follows

"{\"error\":\"invalid\_grant\",\"error\_description\":\"The user name or password is incorrect.\"}"

### Comments/Assumptions

### Status Codes

The status codes and its meaning for this API is as follows

* 200 – Means credentials provided are valid and return values will have the token and expiry as documented above
* 400 -In case the input does not contain the required parameters. Message in the failure output would be “Please provide all mandatory inputs”

Eg- "Message": "Please provide all mandatory inputs"

* 401 – Unauthorized if the credentials are invalid. Message will contain the error and error description
* 500 – In case of internal failures. This is a case of exception in login and message would contain the exception message

### Test Cases

Test cases needs to be written for following scenarios

* Valid credentials
* Invalid credentials
* Input not properly sent to the API

## Refresh Token API

### URL

https://{{HOST\_URL}}/api/Login/Refresh

### Request Type

PUT

### Description

The API takes the refresh token as the only form data and communicates with identity server to provide the access token to the user. Refresh tokens are required if you want to give a short lived token to the client and then we will need to provide refresh token so that we can call further API by not logging in

Client would do the following

Client would call the login API and get an access token and get also a refresh token. This will be stored on client side to call the resource API. Now any call fails with 401, client will take the refresh token and send across to this API to see if that works and if that works, this API will again return access token and a new refresh token along with access token expiry. If the refresh token call also does not work, client will redirect to the login page which means refresh token also has expired and this is the right time to login again to get a new access token.

Refresh tokens are implemented in a way that once for a client and a user. Once a refresh token expires for a client and a user then it cannot work well.

### Prerequisites

User should have called the login API described above and got the refresh token so that you can refresh the access token using this API

### Input

Input will be the refresh token which you got from the login API described above. Client needs to set this in the body. Given below is the request

PUT /InstaAPI/api/login/Refresh HTTP/1.1

Host: localhost

Authorization: Basic OTJhODU4Y2UtNWVkYi00YTYyLWFjOGUtYzk4MTAzZDE3ZWExOkNpVUc0dHJvTWw5QVdiWEkzeVExeUlaSGtDN1laOVpLVGYvWkMzUFlMaDA9

Content-Type: application/json

Cookie: ASP.NET\_SessionId=gtmfl5f2kofk0lsjgr43yeop

{

"refreshToken":"a33a7aa9035c4359a1fd5996723954a3"

}

Refresh\_token is the token returned by login API

### Output

The output is exactly similar to Login API and it is as follows

1. Access\_token – The fresh access token API
2. Token\_type- this will always be “bearer”
3. Expires\_in- Gives the value of token expiry
4. Refresh\_token – refresh token for future updated access tokens

## Comments/Assumptions

The API will be useful for short lived access tokens. Usually refresh tokens are used for the following

1. Updating access token content: as you know the access tokens are self contained tokens, they contain all the claims (Information) about the authenticated user once they are generated, now if we issue a long lived token (1 month for example) for a user named “Alex” and enrolled him in role “Users” then this information get contained on the token which the Authorization server generated. If you decided later on (2 days after he obtained the token) to add him to the “Admin” role then there is no way to update this information contained in the token generated, you need to ask him to re-authenticate him self again so the Authorization server add this information to this newly generated access token, and this not feasible on most of the cases. You might not be able to reach users who obtained long lived access tokens. So to overcome this issue we need to issue short lived access tokens (30 minutes for example) and use the refresh token to obtain new access token, once you obtain the new access token, the Authorization Server will be able to add new claim for user “Alex” which assigns him to “Admin” role once the new access token being generated.
2. Revoking access from authenticated users: Once the user obtains long lived access token he’ll be able to access the server resources as long as his access token is not expired, there is no standard way to revoke access tokens unless the Authorization Server implements custom logic which forces you to store generated access token in database and do database checks with each request. But with refresh tokens, a system admin can revoke access by simply deleting the refresh token identifier from the database so once the system requests new access token using the deleted refresh token, the Authorization Server will reject this request because the refresh token is no longer available (we’ll come into this with more details).
3. No need to store or ask for username and password: Using refresh tokens allows you to ask the user for his username and password only one time once he authenticates for the first time, then Authorization Server can issue very long lived refresh token (1 year for example) and the user will stay logged in all this period unless system admin tries to revoke the refresh token. You can think of this as a way to do offline access to server resources, this can be useful if you are building an API which will be consumed by front end application where it is not feasible to keep asking for username/password frequently.

### Status Codes

The status codes and its meaning for this API is as follows

* 200 – Means credentials provided are valid and return values will have the token and expiry as documented above
* 400 -In case the input does not contain the required parameters or is wrong. Message would contain an appropriate message

Eg- "Message": "Please provide all mandatory inputs"

* 500 – In case of internal failures. This is a case of exception in login and message would contain the exception message

### Test Cases

Test cases needs to be written for following scenarios

* Invalid grant\_type
* Invalid refresh\_token
* Valid refresh token

# Section 2 – Hotel Inventory APIs

## Get Meeting Rooms API

### URL

https://{{HOST\_URL}}/api/Hotels/{Id}/MeetingRooms

### Request Type

GET – the request type is a Get method

### Request Parameters

1. {Id} – Hotel Id is required
2. {name} – optional
3. {sort} – optional (field\_name:asc or field\_name:desc)
4. {page} – page number
5. {page\_size} – page size

**Request URL Example**

### Prerequisites

Token needs to set in header.

### Description & Implementation

The API is used to return a list of all the conference rooms which will be filtered by request parameters.

So, code structure will be

1. Routing Setup –   
   As per API URL, need to configure route in *WebApiConfig.cs*   
   Or

Can set on API method like

**[Route("{hotelId:int}/MeetingRooms")]**  
  
This API URL will redirect on *MeetingRoomController* and call ***GetMeetingRooms*** method, which will accept *HotelId, name, sort, page & page\_size* parameters.

1. Create *MeetingRoom* model with the following properties
   1. ConferenceRoomID
   2. RoomName
   3. Description
   4. IsPrimaryRoom
   5. MaxAttendees
   6. NoOfRooms
   7. SetupCost
   8. … **Can Add more properties**
2. HotelModel – It is already exists and constructor is taking *HotelId* to get Hotel data.
3. Create *GetMeetingRooms* function in *HotelModel* which will return the list of *MeetingRoom* model.
4. And, to get meeting rooms, Call *HotelModel*’s *GetMeetingRooms* function from *MeetingRoomsController*

*return new HotelModel(id).GetMeetingRooms(filters);*Note: - filters is an object of FiltersParameters model

1. *HotelModel(id).GetMeetingRooms(filters)* – will call to HotelBO2.GetConferenceRooms() and it will return the list of all the ConferenceRoomBO2 and then will prepare list of *MeetingRoom* model.

### Response

Create ApiResponseModel for generic API response with following properties

1. data – this will contain a list or object of generic type
2. total\_count – this will only be available in json response when client requested for pageable data.This gives the total count of meeting rooms

**Json Example**

{

"data": [

{

"ConferenceRoomID": 489,

"RoomName": "P1",

"Description": "",

"IsPrimaryRoom": true,

"MaxAttendees": 100,

"NoOfRooms": 0,

"SetupCost": 2500

},

{

"ConferenceRoomID": 494,

"RoomName": "B1",

"Description": "",

"IsPrimaryRoom": false,

"MaxAttendees": 50,

"NoOfRooms": 0,

"SetupCost": 100

}

]

}

### Implementation Note -

GetMeetingRoom API is a Get request accepting multiple parameters, from those parameters will create FiltersParameters model and pass it for further operations.

**FiltersParameters** **model** **structure & Properties**

1. search - SearchParameters: - Optional
   1. id (number)
   2. name (string)
2. sorting – SortingParameters class model - optional
   1. sort\_field (string)
   2. direction (string (asc/desc) – default is asc)
3. paging – PagingParameters class model – optional
   1. page\_num
   2. page\_size

### Status Codes

The status codes and its meaning for this API is as follows

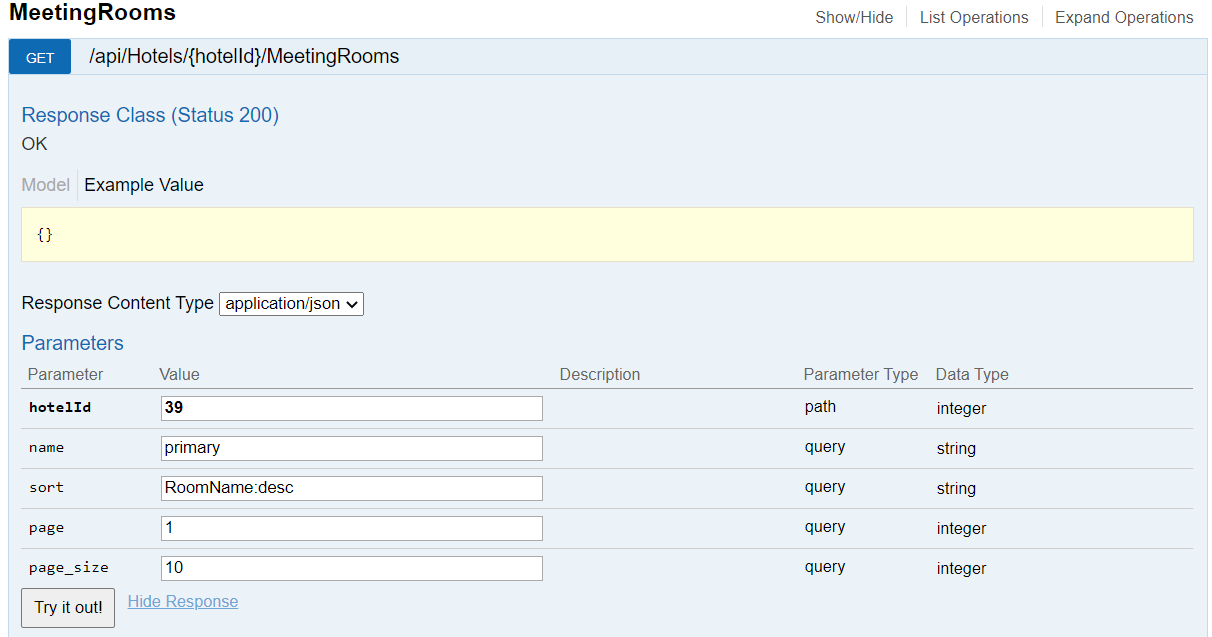
* 200 – Valid response with JSON data
* 400 - Input mismatched from client side.
* 401 – Unauthorized if the credentials are invalid.
* 404 – In case of internal failures.
  + Hotel not found
* 500 – Any internal error/unknown errors
  + Code error
  + DB connectivity errors
  + Or any other error which is not handled in API

### Test Cases

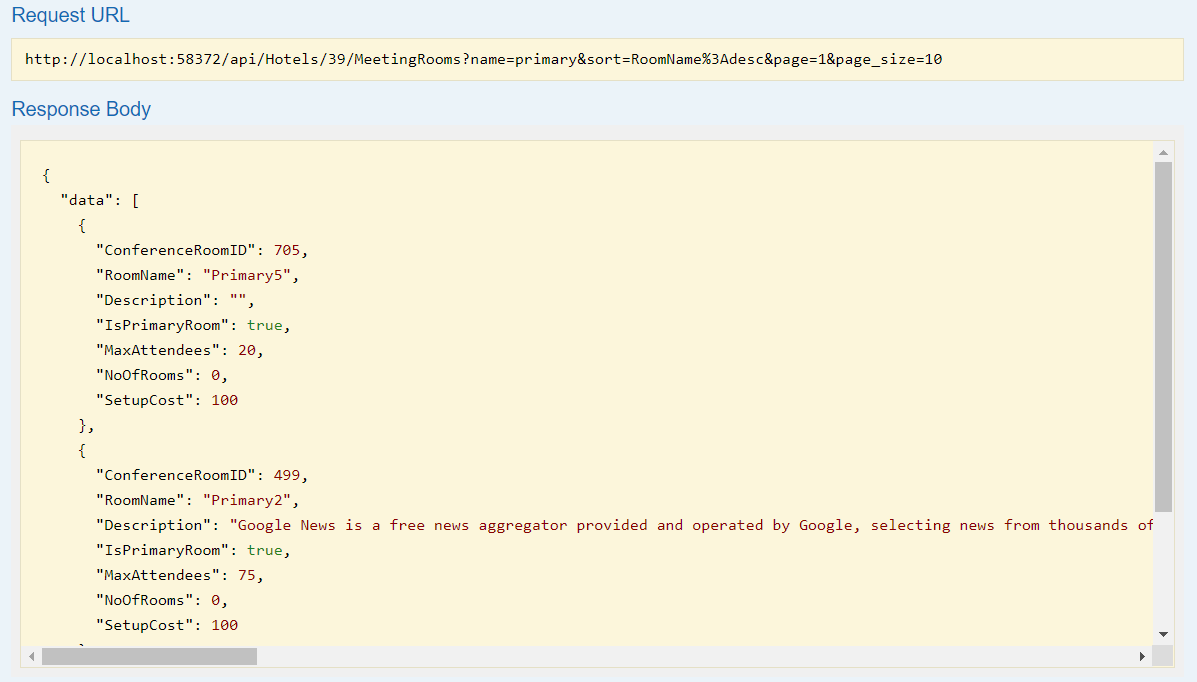
Test cases needs to be written for following scenarios

* Hotel existing and return meeting rooms
* Hotel does not exist (negative scenario)
* Check with meeting rooms data
* Meeting rooms does not exist (negative scenario)
* Search for meeting room (negative test, should be failed)
* Paging testing – (check some pages)
* Sorting testing – (check both asc/desc)

### Example - **Request**



### Example – **Response**



## GET Hotel API

## URL

https://{{HOST\_URL}}/api/Hotels/{Id}/

## Request Type

GET

## Request Parameters

{Id} => HotelId of the hotel

## Prerequisites

* Token to be set on header
* User should have access permission to the hotel
* Hotel should be valid

## Description and Implementation

Gets a hotel with the id represented by the id of the hotel. If the id does not exist then it will return a not found error.

The API uses the following models

* HotelModel
* AddressModel
* CountryModel
* LocationModel
* LatLongModel
* HoldPeriodModel

Example Request

<http://localhost/InstaAPI/api/Hotels/39>

Example response

{

    "data": {

        "HoldPeriods": [

            {

                "id": 5632,

                "StartTime": "08:00:00",

                "EndTime": "12:00:00",

                "Name": "AM"

            },

            {

                "id": 5633,

                "StartTime": "12:00:00",

                "EndTime": "17:00:00",

                "Name": "PM"

            },

            {

                "id": 5634,

                "StartTime": "17:00:00",

                "EndTime": "22:00:00",

                "Name": "Dinner"

            },

            {

                "id": 5633,

                "StartTime": "12:00:00",

                "EndTime": "22:00:00",

                "Name": "PM,Dinner"

            },

            {

                "id": 5632,

                "StartTime": "08:00:00",

                "EndTime": "22:00:00",

                "Name": "AM,PM,Dinner"

            },

            {

                "id": 5632,

                "StartTime": "08:00:00",

                "EndTime": "17:00:00",

                "Name": "AM,PM"

            }

        ],

        "HotelId": 39,

        "HotelCode": null,

        "HotelURL": "Knights-Inn-NAIROBI",

        "StarRating": 7,

        "location": {

            "Address": {

                "Address1": "VILLA ROSA KEMPINSKI NAIROBI, CHIROMO ROAD",

                "Address2": "",

                "City": "New York",

                "Country": {

                    "CountryId": 99,

                    "CountryName": "INDIA"

                },

                "ZipCode": "00800",

                "StateId": 0

            },

            "LatLong": {

                "Latitude": -1.271252,

                "Longitude": 36.809015

            }

        }

    }

}

## Status Codes

* 200 – Valid response with JSON data
* 400 - Input mismatched from client side.
* 401 – Unauthorized if the credentials are invalid.
* 404 – In case of not found resources
  + Hotel not found
* 500 – Any internal error/unknown errors
  + Code error
  + DB connectivity errors
  + Or any other error which is not handled in API

## Test Cases

Following test cases will be written

* API with non existing hotel id
* API with existing hotelid

# Events API

## Get Events

### URL

https://{{HOST\_URL}}/api/Hotels/{Id}/Events

### Request Type

GET – Request will be GET type

### Request Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Meaning | Type | Required |
| Id | Hotelid | Comes in the URL as a parameter | Yes |
| Sd | Start Date in MM/DD/YYYY format | Date | Yes |
| Ed | End Date in MM/DD/YYYY format | Date | Yes |
| Name | Name of the event to filter with . This will work like contains | String | No |
| Sort | Sorting by one field like fieldname:asc(desc) | String | No |
| Page | Page number requested | Integer | No. If not provided it returns all the results without paging |
| Page\_size | Page size of a single page. Valid only if Page is given . | Integer | No |

Request URL example

<https://api.instagathering.com/hotels/39/Events?Sd=10/12/2020&ed=10/22/2020>

### Prerequisite

* Token must be set in the header
* User in the token should have appropriate permissions to access events
* Hotel must exist
* Hotel must have events enabled

### Description and Implementation

The API will return only the basic information of the event. The children for the same can be got by calling children API only.

API will return the array of the following structure. Clients can use the following structure to deserialize the json into the object

public class EventModel

{

public int EventId { get; set; }

public string EventName { get; set; }

public DateTime EventDate { get; set; }

public DateTime StartTime { get; set; }

public DateTime EndTime { get; set; }

public int ConferenceRoomId { get; set; }

}

The data will be sorted and paged according to the request parameters.

**Note- There is server side paging available for start date and end date only. Rest of the filtering happens in the service layer as of now**

The API will have start date and end date as required parameters and will not work without those parameters. This is done so that client does not ask server details for too many details and clog up the server.

Sample json for the return data is as follows

{

"data": [

{

"EventId": 23073,

"EventName": "RamManohar",

"EventDate": "2020-07-22T00:00:00",

"StartTime": "1900-01-01T10:00:00",

"EndTime": "1900-01-01T12:00:00",

"ConferenceRoomId": 498

},

{

"EventId": 23074,

"EventName": "RamManohar",

"EventDate": "2020-07-23T00:00:00",

"StartTime": "1900-01-01T10:00:00",

"EndTime": "1900-01-01T12:00:00",

"ConferenceRoomId": 498

},

{

"EventId": 23075,

"EventName": "RamManohar",

"EventDate": "2020-07-24T00:00:00",

"StartTime": "1900-01-01T10:00:00",

"EndTime": "1900-01-01T12:00:00",

"ConferenceRoomId": 498

},

],

"total\_count": 25

}

Total\_count will be returned only if paging is asked for or else total\_count will be empty. That is both page\_num and page\_size needs to be filled out to get this in return json.

### Status Codes

The status codes and its meaning for this API is as follows

* 200 – Valid response with JSON data
* 400 - Input mismatched from client side.
* 401 – Unauthorized if the credentials are invalid.
* 404 – In case of internal failures.
  + Hotel not found
* 500 – Any internal error/unknown errors
  + Code error
  + DB connectivity errors
  + Or any other error which is not handled in API

### Test cases

1. Calling the API with basic parameters of start date and end date
2. Calling the API with basic parameters and sorting
3. Calling API with basic parameters and paging
4. Calling API with basic, sorting and paging parameter
5. Calling API with basic and search string event name
6. Calling the API with wrong hotel id
7. Calling API with wrong types of input
8. Calling API with wrong type of parameters (like invalid date, end date)

# Occupancy API

## Get Occupancy

### URL

https://{{HOST\_URL}}/api/Hotels/{Id}/Occupancy

### Request Type

GET – Request will be GET type

### Request Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Meaning | Type | Required |
| Id | Hotelid | Comes in the URL as a parameter | Yes |
| startdate | Start Date in MM/DD/YYYY format | Date | Yes |
| enddate | End Date in MM/DD/YYYY format | Date | Yes |
| Name | Name of the event to filter with . This will work like contains | String | No |
| Sort | Sorting by one field like fieldname:asc(desc) | String | No |
| Page | Page number requested | Integer | No. If not provided it returns all the results without paging |
| Page\_size | Page size of a single page. Valid only if Page is given . | Integer | No |
| type | The Occupancy of what type of Resource | string | Yes |

Request URL example

<https://api.instagathering.com/Hotels/39/occupancy?type=meetingroom&startdate=2020-06-01&enddate=2020-09-05>

### Prerequisite

* Token must be set in the header
* User in the token should have appropriate permissions to access events
* Hotel must exist
* Hotel must have events enabled

### Description and Implementation

The API will return only the basic information of the event. The children for the same can be got by calling children API only. The OccupancyController instantiates a HotelModel by passing the hotel id, the HotelModel has a method to get the occupancy. In the GetOccupancy method we make use of the HotelBO’s getOccupancyMethod. The HotelBO communicates with HotelDO which in turn invokes an SP that will get the occupancy. The SP for fetching the occupancy will get the events from the eventDetails table and do a union with the events from the dataimportconferenceroom table.

API will return the array of the following structure. Clients can use the following structure to deserialize the json into the object

public class EventModel

{

public int EventId { get; set; }

public string EventName { get; set; }

public DateTime EventDate { get; set; }

public DateTime StartTime { get; set; }

public DateTime EndTime { get; set; }

public int ConferenceRoomId { get; set; }

}

The data will be sorted and paged according to the request parameters.

**Note- There is server side paging available for start date and end date only. Rest of the filtering happens in the service layer as of now**

{

"data": [

{

"EventId": 23073,

"EventName": "RamManohar",

"EventDate": "2020-07-22T00:00:00",

"StartTime": "1900-01-01T10:00:00",

"EndTime": "1900-01-01T12:00:00",

"ConferenceRoomId": 498

},

{

"EventId": 23074,

"EventName": "RamManohar",

"EventDate": "2020-07-23T00:00:00",

"StartTime": "1900-01-01T10:00:00",

"EndTime": "1900-01-01T12:00:00",

"ConferenceRoomId": 498

},

{

"EventId": 23075,

"EventName": "RamManohar",

"EventDate": "2020-07-24T00:00:00",

"StartTime": "1900-01-01T10:00:00",

"EndTime": "1900-01-01T12:00:00",

"ConferenceRoomId": 498

},

],

"total\_count": 25

}

### Status Codes

The status codes and its meaning for this API is as follows

* 200 – Valid response with JSON data
* 400 - Input mismatched from client side.
* 401 – Unauthorized if the credentials are invalid.
* 404 – In case of internal failures.
  + Hotel not found
* 500 – Any internal error/unknown errors
  + Code error
  + DB connectivity errors
  + Or any other error which is not handled in API

### Test cases

1. Calling the API with basic parameters of start date and end date
2. Calling the API with basic parameters and sorting
3. Calling API with basic parameters and paging
4. Calling API with basic, sorting and paging parameter
5. Calling API with basic and search string event name
6. Calling the API with wrong hotel id
7. Calling API with wrong types of input
8. Calling API with wrong type of parameters (like invalid date, end date)

## SET Occupancy

### URL

https://{{HOST\_URL}}/api/Hotels/{Id}/Occupancy

### Request Type

POST – Request will be of POST type

Request URL example

<https://api.instagathering.com/hotels/39/occupancy>

### Prerequisite

* Token must be set in the header
* User in the token should have appropriate permissions to create event
* Hotel must exist
* Hotel must have events enabled
* Meeting room must exist

### Description and Implementation

This is used to set occupancy of a resource for a period. The occupancy will be saved as an event in the eventDetails table. The request reaches the EventController and it instantiates the EventModel. In the EventModel, the EventBO is instantiated using newly created event entity. Save method of EventBO is used to save the event details. The ISP\_CREATE\_EVENT\_DETAILS Is called by the eventDO for db insertion.

### Post Fields

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter Name | Meaning | Type | Required |
| startTime | Start Date in (MM/DD/YYYY HH:mm:ss) format | Date | Yes |
| endTime | End Date in (MM/DD/YYYY HH:mm:ss) format | Date | Yes |
| name | Name of the event | String | Yes |
| conferenceRoomId | Id of the Conference room where the event will happen | Integer | Yes |
| eventDate | Date on which event is supposed to happen | Date | Yes |

Sample JSON

{

“startTime”: ”2020/08/01T12:00:00”,

“endTime”: ”2020/08/01T13:00:00”,

“name”: “Annual Day”,

“conferenceRoomId”: 500,

“eventDate”: “2020/08/01”,

}

### Response Body

{

“id”: 201,

“startTime”: ”2020/08/01T12:00:00”,

“endTime”: ”2020/08/01T13:00:00”,

“name”: “Annual Day”,

“conferenceRoomId”: 500,

“eventDate”: “2020/08/01”,

}

The response body will be like the request body with id being the extra field. The id will be unique identifier of the newly created event.

### Status Codes

The status codes and its meaning for this API is as follows

* 201 - Created
* 200 – Valid response with JSON data
* 400 - Input Invalid.
* 401 – Unauthorized if the credentials are invalid.
* 404 – In case of internal failures.
  + Hotel not found
  + Meeting room not found
* 500 – Any internal error/unknown errors
  + Code error
  + DB connectivity errors
  + Or any other error which is not handled in API

### Test cases

1. Calling with Valid fields should return 201 status.
2. Calling without body should return 400 status.
3. Calling with invalid fields should return 400 status with proper validation message.
4. Calling the API with wrong hotel id
5. Calling the API with wrong meeting room id