

humand

# Manual Web Api



# Introduction

Humand offers a set of integrations that we will refer to as the "WEB API." These integrations are available at no cost to all our clients and are primarily aimed at facilitating real-time and synchronous access to your information on our platform.

The main purpose of this API is to provide a user-friendly and secure interface that can be utilized by any programmer. Additionally, it is designed to simplify the integration of our cloud solution with your existing legacy systems, using a widely proven and currently popular architecture, such as REST API.

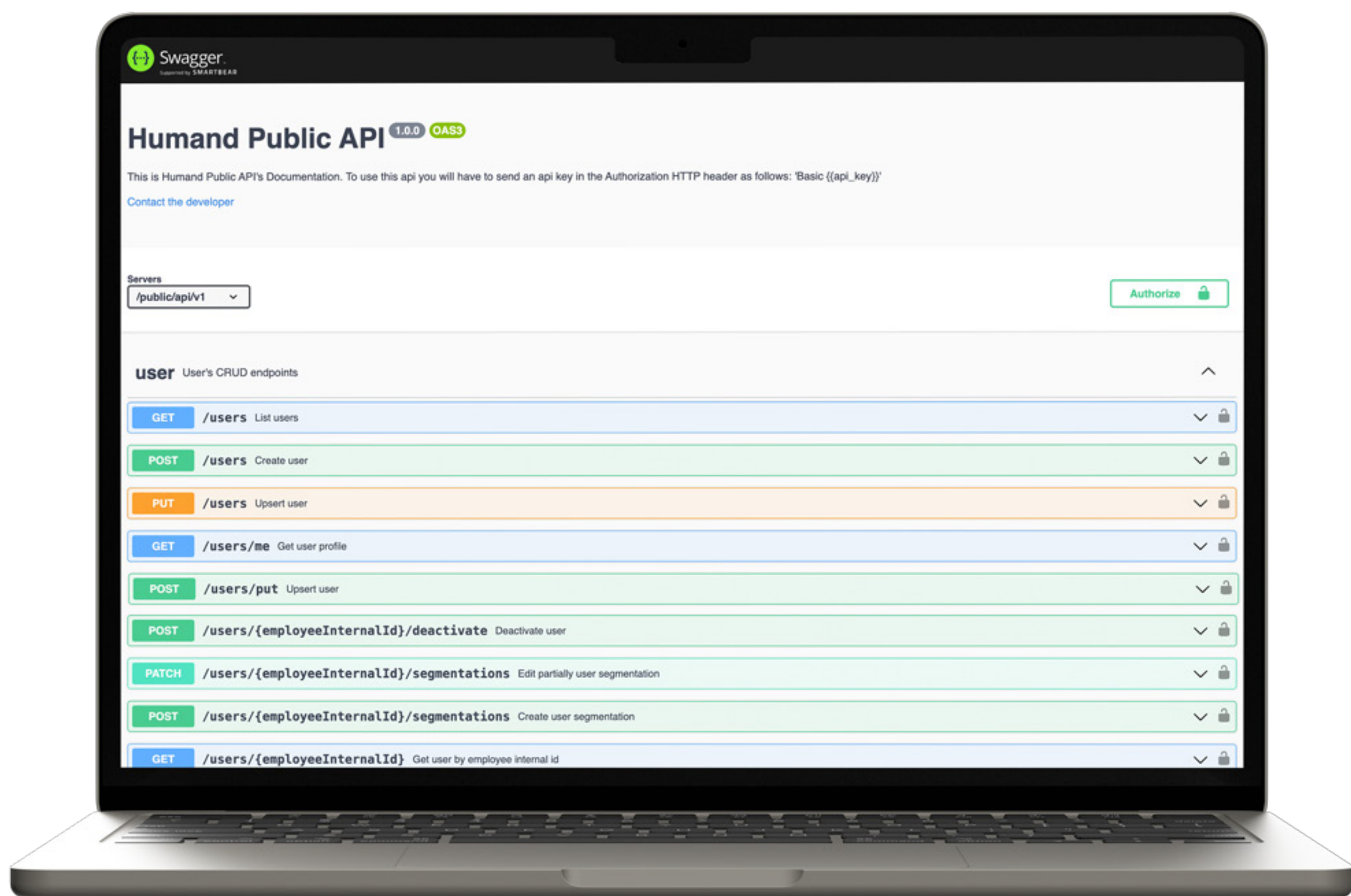
To achieve this goal, the WEB API provides a variety of standard services including employee creation and retrieval, organizational chart management, the ability to make posts, payroll receipt uploads, user segmentation into specific groups, and much more.

## Detailed Documentation

We have a site where you can find all the necessary information to create an integration with our "WEB API" system (invocation examples, responses, and much more). We use an industry-standard tool to detail all this information, such as Swagger.

The Swagger tool allows for documenting and testing APIs. This way, you can not only explore the list of services and their corresponding information (message headers, parameters, response structures, etc.) but also test the services directly through the browser without the need to write programming code.

[View site](#)



# General Integration Procedure

**A.** Currently, access to the testing platform is granted using an API Key. This specific access information for the testing platform will be provided by Humand once the decision to proceed with the integration has been made.

Once testing has been successfully completed in the test environment and the integration is ready for deployment in the production environment, the connection to the production community will be made using a different API Key that will be supplied by us.

This process ensures a secure and controlled transition from the testing environment to the production environment, maintaining data security and integrity at all times.

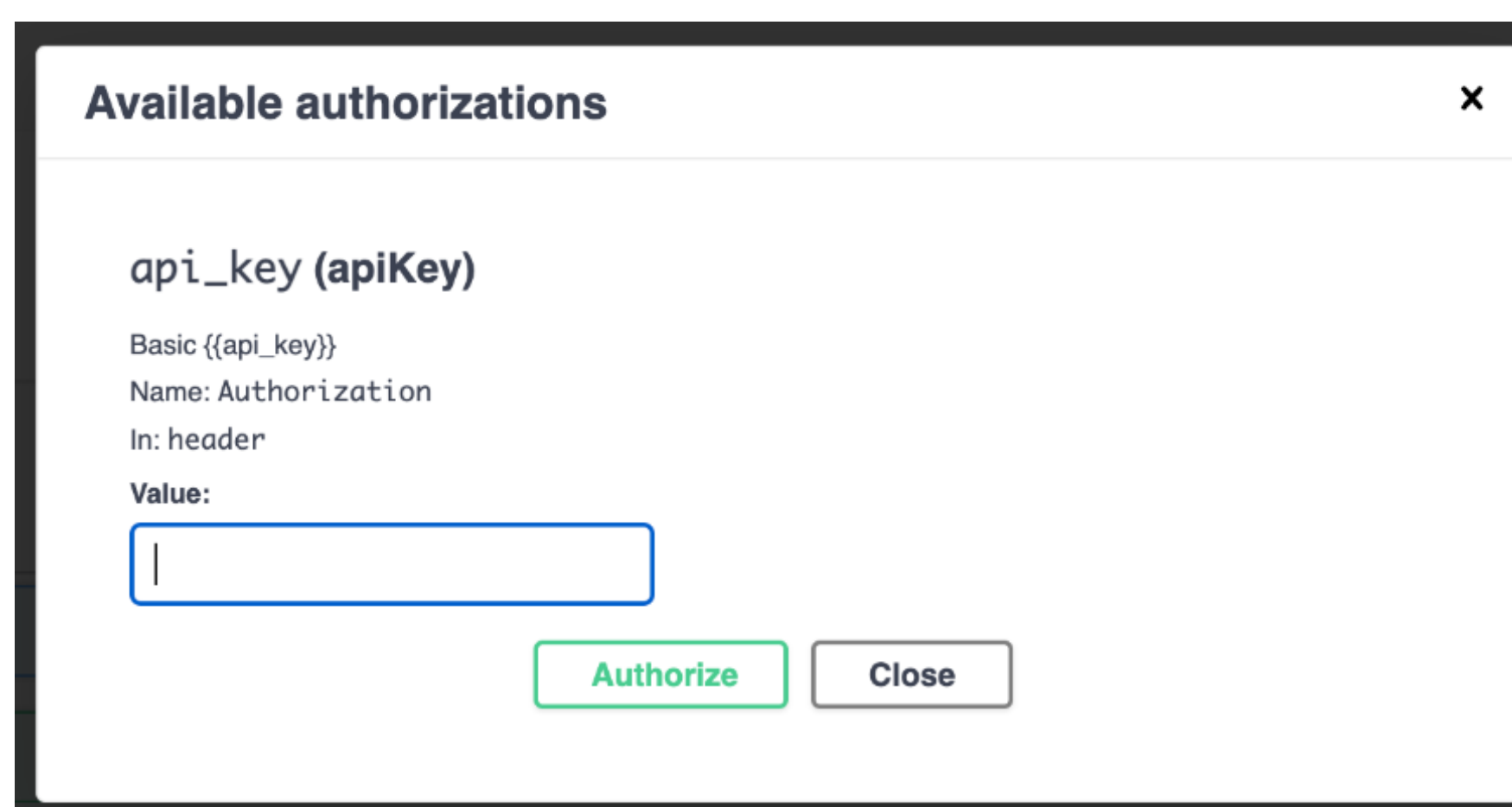
Currently, access granted through an API Key is to the testing platform (this information will be provided by Humand once the decision to proceed with the integration is made). Once testing is complete, the connection will be made to the production community with a different API Key that we will provide.

**Important:** this API Key must be associated with a user who has administrator permissions. It is important that this User is NOT deleted to ensure permanent integration.

**B.** To perform the tests, the API Key should be entered in the following format:

Basic JyBpaEH7YMppAQjqyGMSV53PHk9-Pd\_T

In this example, "JyBpaEH7YMppAQjqyGMSV53PHk9-Pd\_T" is the associated API Key (Humand's team will provide the corresponding key for each community). Note that you must enter the word Basic, followed by a space, and then the associated API Key.



The screenshot shows a dialog box titled "Available authorizations" with a close button (X) in the top right corner. The main content area displays the following information:

- api\_key (apiKey)**
- Basic {{api\_key}}
- Name: Authorization
- In: header
- Value:

Below the "Value:" label is an empty text input field. At the bottom of the dialog, there are two buttons: "Authorize" (highlighted in green) and "Close".

# Scope Details of Integrations with Humand:

**Users in Humand have three types of pertinent information that make up their profile:**

## **A. Basic Fields of Personal Information:**

This is the basic information that a user has in their profile in the application. The mandatory data are those required for the user to be successfully created on the platform. Optional fields may or may not be added depending on the information available in the database.

### **User Profile Information in Humand:**

#### **> Username**

Mandatory, string - `"employeeInternalId": "string"`

This is not the person's name, but the username they will use to log into Humand.

#### **> First Name**

Mandatory, string - `"firstName": "User first name"`

There is no distinction between first and middle names; the full name should be included.

#### **> Last Name**

Mandatory, string - `"lastName": "User last name"`

There is no distinction between first and second last names; the full name should be included.

#### **> Initial Password**

Mandatory, string - `"password": "password"`

Must be at least 8 characters long.

#### **> Direct Manager**

- Mandatory, string -

In the JSON, the `"employeeInternalId"` of the assigned manager must be completed. The direct manager is specified with the following JSON segment (**Important: You must first create the manager's user with their corresponding `"employeeInternalId"`, and then, when creating the subordinate employee's user, set the manager's `"employeeInternalId"` as the subordinate's manager. An error will occur if a non-existent `"employeeInternalId"` is used as the manager).**

```
"relationships": [  
  {  
    "name": "BOSS",  
    "employeeInternalId": "userEmployeeInternalId"  
  }  
]
```

#### **> User Email**

Optional, string

`"email": "user@email.com"`



### > User Nickname

Optional, string - "nickname": "userNickName"

### > User Phone Number

Optional, string - "phoneNumber": "+5491123456789"  
The country code prefix is necessary.

### > User Hiring Date

Optional, string - "hiringDate": "2022-01-01"

### > User Birthdate

Optional, string - "birthdate": "2022-01-01"

## B. Segmentation Items

"item": "Segmentation item name"

Each group has one or more segmentation items. Each user can have none, one, or multiple items for a particular segmentation group.

**Segmentations are not mandatory; they depend on the information available in the database. Some examples can include:**

- Management
- Sector
- Location

**The JSON section corresponding to a user's segmentation at the time of their creation would be:**

```
"segmentation": [  
  {  
    "group": "Management",  
    "item": "Segmentation item name"  
  },  
  {  
    "group": "Sector",  
    "item": "Segmentation item name"  
  },  
  {  
    "group": "Location",  
    "item": "Segmentation item name"  
  },  
]
```

## C. Dynamic or Extra Fields in User Profiles

This is additional information that each community can choose to create for users. The data is optional and may or may not be added depending on the information available in the database.

**Important:** The profile fields written in the API must match 100% (including case sensitivity) the field entered in the Settings → Profiles section within the Humand Admin Panel.

> **Unique Identifier for the Dynamic Field Value (can be obtained via GET)**  
"ID": "7b122126-f8ec-4410-b163-8389ed21e79f"

> **Name of the Profile Field**  
"name": "Dynamic field name"

It is important to note that you can send either **ID** or **Name**; it is not necessary to send both fields. If both fields are sent, each field must contain a value (not be empty).

> **Profile Field Information to Complete**

The JSON structure to complete the profile field information might look like this:

```
"fields": [  
  {  
    "ID": "7b122126-f8ec-4410-b163-8389ed21e79f",  
    "name": "Dynamic field name",  
    "value": "Dynamic field value"  
  }  
]
```

The **value** should be the actual data you want to assign to this dynamic field. If you only have the **ID** or the **name**, you can use either one, but if both are provided, they must both contain values.

Some example dynamic fields might include:

- Position
- Internal Number
- Management
- Building
- Employee ID

Considering this, the complete section of this part of the JSON for user creation associated with dynamic fields would be:

```
"Fields": [  
  {  
    "ID": "7b122126-f8ec-4410-b163-8389ed21e79f",  
    "name": "Position",  
    "value": "Software Engineer"  
  },  
  {  
    "ID": "12345678-9abc-def0-1234-56789abcdef0",  
    "name": "Internal Number",  
    "value": "1234"  
  },  
  {  
    "ID": "abcdef12-3456-7890-abcd-ef1234567890",  
    "name": "Management",  
    "value": "IT Management"  
  },  
  {  
    "ID": "fedcba98-7654-3210-fedc-ba9876543210",  
    "name": "Building",  
    "value": "Headquarters"  
  },  
  {  
    "ID": "01234567-89ab-cdef-0123-456789abcdef",  
    "name": "Employee ID",  
    "value": "EMP001"  
  }  
]
```

In this example, each dynamic field has an **ID**, **name**, and **value**. You can provide either the ID or name for each field, but if both are provided, they must both contain values.:

## Main Difference Between User Segmentation and Dynamic or Extra Profile Fields:

The key difference between these two types of information is that **User Segmentation** acts as a filter in Humand. For example, we could filter a post based on the location of company employees, with "Location" being one of those segmentations. In contrast, **Dynamic Profile Fields** are details that appear in employee profiles; for instance, their identification number. These fields are informative and are not used for filtering.

## Important Considerations:

Before sending segmentations or dynamic profile fields through the API, we recommend logging into admin.humand.co with the assigned testing community credentials.

In the admin panel, you will find a section called "**Segmentation**." Here, you can pre-load various segmentations and the associated items for each. These are customizable and can be adjusted according to your database. You can perform this loading in bulk.

## Tutorials:

[How do I perform a bulk upload of segmentations?](#)

[How do I perform a bulk upload of segmentation items?](#)

**Note:** In the videos, the section referred to as "**Groups**" has now been renamed to "**Segmentation**."

Another section you will find is **Settings**. Once there, please navigate to the third tab called **Profiles**. At the bottom of the page, you will find a section to upload dynamic profile fields where you can add as many as needed. These are also customizable and can be adjusted according to your database.

## Tutorials:

[What are custom profile fields?](#)

[How do I perform a bulk upload of profile fields?](#)

It is essential that the naming conventions of the segmentations and dynamic fields within the Humand admin panel match exactly with what is sent through the API.

# API Usage

## Users

**POST - Create Users:** This endpoint will be used for creating new users. Below is an example of the JSON for a complete user case, including all provided examples (with all segmentations and dynamic fields included):

```

{
  "password": "password",
  "segmentation": [
    {
      "group": "Management",
      "item": "Customer Experience"
    },
    {
      "group": "Sector",
      "item": "Onboarding"
    },
    {
      "group": "Location",
      "item": "California"
    }
  ],
  "relationships": [
    {
      "name": "BOSS",
      "employeeInternalId": "userEmployeeInternalId"
    }
  ],
  "Fields": [
    {
      "ID": "7b122126-f8ec-4410-b163-8389ed21e79f",
      "name": "Position",
      "value": "Software Engineer"
    },
    {
      "ID": "12345678-9abc-def0-1234-56789abcdef0",
      "name": "Internal Number",
      "value": "1234"
    },
    {
      "ID": "abcdef12-3456-7890-abcd-ef1234567890",
      "name": "Management",
      "value": "IT Management"
    },
    {
      "ID": "fedcba98-7654-3210-fedc-ba9876543210",
      "name": "Building",
      "value": "Headquarters"
    },
    {
      "ID": "01234567-89ab-cdef-0123-456789abcdef",
      "name": "Employee ID",
      "value": "EMP001"
    }
  ],
  "employeeInternalId": "string",
  "email": "user@email.com",
  "firstName": "User first name",
  "lastName": "User last name",
  "nickname": "userNickName",
  "phoneNumber": "5491123456789",
  "hiringDate": "2022-01-01",
  "birthdate": "2022-01-01"
}

```

**PUT - Upsert Users:** This endpoint will be used to update existing users. The JSON that must be sent is the complete user creation JSON, including any modified fields. The body of the JSON in this case is identical to that of the POST request. **Important: To ensure that no information is lost, it is essential that the body of the PUT request contains all user information (even if it has not been modified).**



**PATCH - Partially Update User:** This endpoint will be used for partially updating existing users. Note that dynamic profile fields cannot be modified using this endpoint. The JSON should be sent with only the information you wish to modify.

```
{
  "segmentation": [
    {
      "group": "Gerencia",
      "item": "Segmentation item name"
    },
    {
      "group": "Sector",
      "item": "Segmentation item name"
    },
    {
      "group": "Ubicación",
      "item": "Segmentation item name"
    }
  ],
  "employeeInternalId": "string",
  "email": "user@email.com",
  "firstName": "User first name",
  "lastName": "User last name",
  "nickname": "userNickName",
  "phoneNumber": "5491123456789",
  "hiringDate": "2022-01-01",
  "birthdate": "2022-01-01"
}
```

**PATCH - Update user profile fields:** This endpoint will be used to partially update dynamic profile fields of existing users. It is important that the JSON only contains the information that you wish to modify. In this example, only the dynamic fields that need to be updated are included in the JSON body.

```
"fields": [
  {
    "name": "Puesto",
    "value": "Dynamic field value"
  },
  {
    "name": "Número de Interno",
    "value": "Dynamic field value"
  },
  {
    "name": "Gerencia",
    "value": "Dynamic field value"
  },
  {
    "name": "Edificio",
    "value": "Dynamic field value"
  },
  {
    "name": "Fecha de Inicio Puesto Actual",
    "value": "Dynamic field value"
  }
],
```

**PATCH & POST - Partially Update User Segmentation:** These endpoints are used to partially update a user's segmentation. In the previous PATCH endpoint for user updates, the segmentation block must be sent complete to update the user's information, meaning all segmentations and items the user belongs to must be sent. If you want to partially update a user's segmentation, there are 2 new methods for a new endpoint.

**POST - /users/:employeeInternalId/segmentations:** This endpoint is used to assign the user the items of the segment that are sent, without needing to send the complete details of the other items or segments. Important: This applies to new assignments, meaning if the user was part of another item in the same segment, it will not "replace" it (in that case, see the next method).

**Example:** Suppose we have two segments: 'Location' and 'Tasks.' The 'Location' segment contains the items 'Location A,' 'Location B,' and 'Location C.' The 'Tasks' segment includes the items 'Administrative,' 'Commercial,' and 'Operational.'

Now, consider a user X who is part of the item 'Location A' in the 'Location' segment and the item 'Administrative' in the 'Tasks' segment. If we send a request to this endpoint to add the item 'Commercial' from the 'Tasks' segment to user X, it will be added to that item. User X will also remain in 'Administrative' from the 'Tasks' segment and in 'Location A' from the 'Location' segment.

```
{
"segmentations": [
{
"group": "Segmentation group name",
"item": "Segmentation item name"
}
]
}
```

**PATCH - /users/:employeeInternalId/segmentations:** This endpoint is used to replace the items sent to the user within the same segmentation, without modifying the segmentations that are not sent.

**Example:** Suppose we have two segments: 'Location' and 'Tasks.' The 'Location' segment contains the items 'Location A,' 'Location B,' and 'Location C.' The 'Tasks' segment includes the items 'Administrative,' 'Commercial,' and 'Operational.'

Now, consider a user X who is part of the item 'Location A' in the 'Location' segment and the item 'Administrative' in the 'Tasks' segment. If we send a request to this endpoint to add the item 'Commercial' from the 'Tasks' segment to user X, it will be added to that item and will no longer be part of the 'Administrative' item in the same segment, while still belonging to 'Location A' in the 'Location' segment.

```
{
"segmentations": [
{
"group": "Segmentation group name",
"item": "Segmentation item name"
}
]
}
```

**DELETE - Delete Users:** This endpoint will be used to delete users. For this, you only need to share the username (employeeInternalId).

## Segmentations

**POST - Segmentations:** This endpoint will be used to add a new segmentation. Below is the schema to follow.

```
{
  "name": "Segmentation group name",
  "itemNames": [
    "Segmentation item name"
  ],
  "visibility": "ALL"
}
```

In "itemNames" we will include all items associated with that new segmentation.

The visibility ("visibility") can be ALL (for all users), ADMINS\_ONLY (for administrators only), or USER\_AND\_ADMINS (for the user and administrators).

**PUT - Segmentations:** If we want to modify the current segmentations or items, we should use the PUT Segmentations endpoint with the complete corresponding JSON (similar to the previous one).

## Personal Documents

**POST - Document:** This endpoint allows uploading personal documents (payroll receipts, contracts, etc.) for users.

Some of the fields that can be filled out are as follows:

### > file

Type: string (binary)

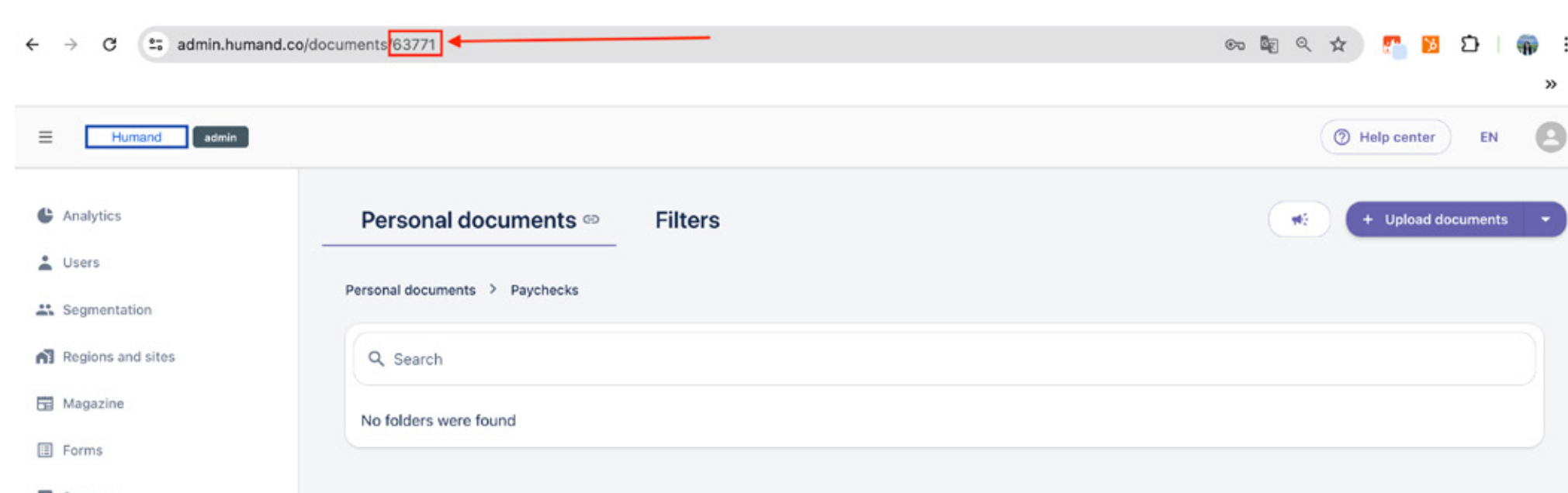
Description: This field should contain the document in binary format, meaning the file is attached in its raw format without processing.

### > folderId

Type: integer

Description: The 'folderId' is a numerical identifier corresponding to the folder where the information will be stored. To find this value, access the platform from the administrator account and navigate to the 'Personal Documents' section. The 'folderId' is located in the URL of the folder where the information will be stored. This identifier remains constant over time; whenever a specific file is sent to a particular folder, the ID of that folder stays the same. Collaborators will not be able to see files belonging to other users.

In the following example, the folder ID would be: 63771.



### > name

Type: string

Description: This field is used to specify the name of the file being sent, which must have a unique naming convention. It is recommended that the file name includes the collaborator's 'internalId'.

### > sendNotification

Type: boolean

Description: This field indicates whether a notification should be sent when the file is submitted. If the value is 'true', a notification will be sent. If 'false', no notification will be sent.

### > signatureStatus

Type: string

Description: This field indicates whether a signature is required on the file. The status of the signature should be specified, such as whether it is needed or not. A possible response option is "PENDING."

### > signatureCoordinates

Type: object

Description: This field requests the coordinates where the signatures should be placed. If a signature is required, you can send a template of the receipt to your Humand representative, and we will provide the coordinates soon.

### > allowDisagreement

Type: boolean

Description: This field determines whether signing with disagreement is allowed.

### Example of a Java to send:

```
{
'folderId': '1',
'name': 'File name',
'sendNotification': 'false',
'signatureStatus': 'PENDING',
'signatureCoordinates': '[{"page":0,"x":0.09078528515064561,"y":0.7916669970581939,"height":0.05259172329577442,
"width":0.21792615674318508}]',
'allowDisagreement': 'false'
}
```

In the file upload process, it is essential to understand that data should not be sent in JSON format, but rather in a format known as 'form data'. To achieve this, we need to follow these steps in addition to the previously mentioned details:

**1. Set Headers:** Ensure that the request headers include the key **Content-Type** with the value **multipart/form-data**. This indicates that we are sending data in 'form data' format.

**2. Attach File:** Ensure that the request headers include the key **Content-Type** with the value **multipart/form-data**. This indicates that we are sending data in 'form data' format.

By following these steps, we will be ready to send files appropriately through HTTP requests using 'form data', ensuring effective communication with the server.



### **Example in Python:**

In this example, the file is uploaded using the requests library in **Python**, ensuring that the data is sent as 'form data'.

```
import requests

url = "https://api-prod.humand.co/public/api/v1/users/Employee_internal_id/documents/files"

payload = {'folderId': '1',
'name': 'File name',
'sendNotification': 'false',
'signatureStatus': 'PENDING',
'signatureCoordinates': '[{"page":0,"x":0.09078528515064561,"y":0.7916669970581939,"height":0.05259172329577442,
"width":0.21792615674318508}]',
'allowDisagreement': 'false'}
files=[
('file','document.pdf',open('/Users/username/Downloads/document.pdf','rb'),'application/pdf'))
]
headers = {
'Content-Type': 'multipart/form-data',
'Accept': 'application/json',
'Authorization': 'API_KEY'
}

response = requests.request("POST", url, headers=headers, data=payload, files=files)

print(response.text)
```

### **Example in Java:**

```
var myHeaders = new Headers();
myHeaders.append("Content-Type", "multipart/form-data");
myHeaders.append("Accept", "application/json");
myHeaders.append("Authorization", API_KEY);

var formdata = new FormData();
formdata.append("file", "voluptate adipisicing Duis veniam enim");
formdata.append("folderId", "1");
formdata.append("name", "File name");
formdata.append("sendNotification", "false");
formdata.append("signatureStatus", "PENDING");
formdata.append("signatureCoordinates", "[object Object]");
```

## Time Off Requests

**GET - Get time off requests:** This endpoint allows you to retrieve vacation and leave requests made in Humand.

**Some of the fields to complete are as follows:**

### Page

Type: integer

Description: The 'page' field specifies the current page number being requested. This information is mandatory. For example, "page=2" indicates that the second page of results is being requested.

Below is an image for illustration:

Registration 1	Page 1
Registration 2	Limit 10
Registration 3	
Registration 4	
Registration 5	
Registration 6	
Registration 7	
Registration 8	
Registration 9	
Registration 10	
Registration 11	Page 2
Registration 12	Limit 10
Registration 13	
Registration 14	
Registration 15	
Registration 16	
Registration 17	
Registration 18	
Registration 19	
Registration 20	
Registration 21	Page 3
Registration 22	Limit 10
Registration 23	
Registration 24	
Registration 25	
Registration 26	
Registration 27	
Registration 28	
Registration 29	
Registration 30	

### > Limit

Type: integer

Description: This field defines the maximum number of items or records to be retrieved per page. The maximum allowed value is 500, thus controlling the size of each segment of data delivered. This information is mandatory. For example, "limit=10" indicates that up to 10 records are desired per page.

### > States

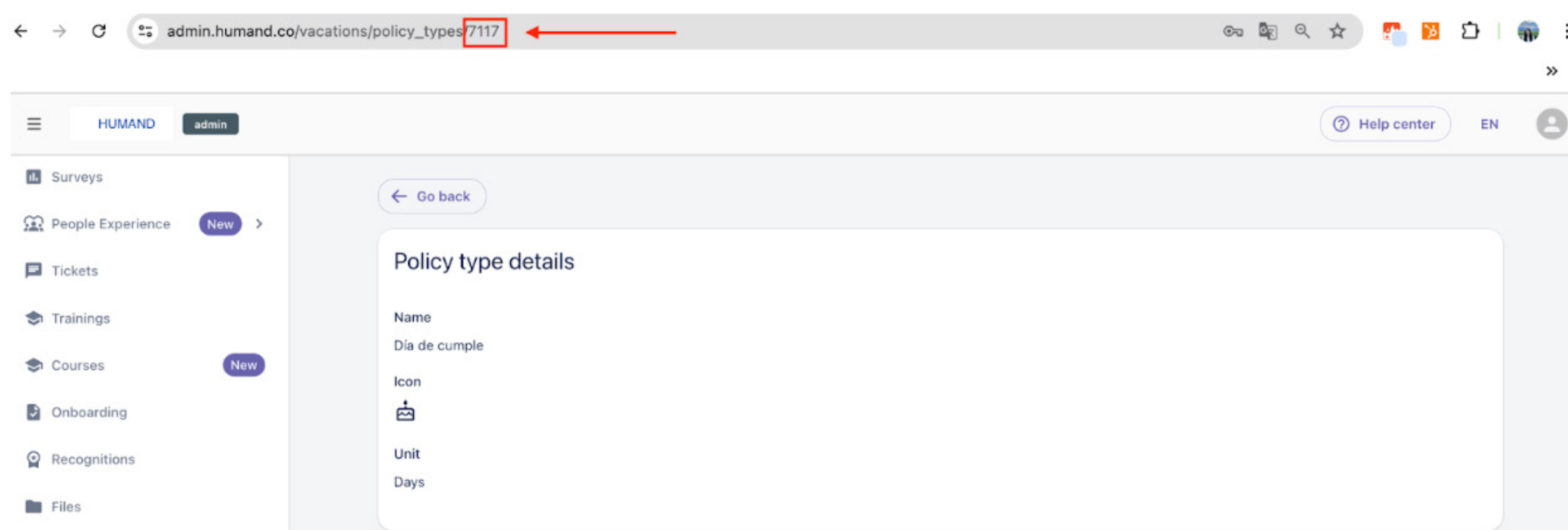
Type: string

Description: This field allows you to specify the status of the request when retrieving information. The options are approved, in progress, rejected, and cancelled. This is optional. Possible values: APPROVED, REJECTED, IN\_PROGRESS, CANCELLED.

### > PolicyTypeIds

Type: string

Description: In this field, you can choose the type of policy related to the requests made, such as Vacation, Study Leave, Maternity Leave. To find the policy ID, you can check the last numbers that appear in the URL when entering its details:



> **from.date**

Type: string

Description: In this field, you can obtain requests whose days are contained from the selected date.

> **to.date**

Type: string

Description: In this field, you can obtain requests whose days are contained up to the selected date.

> **resolutionDateFrom**

Type: string

Description: "From" date for when the request was approved, rejected, or cancelled.

> **resolutionDateTo**

Type: string

Description: "To" date for when the request was approved, rejected, or cancelled.

**An example of a JSON could be the following:**

```
{
  "page": 1,
  "limit": 100,
  "states": "APPROVED",
  "PolicyTypeIds": "1,2,3",
  "fromDate": "2023-01-01",
  "toDate": "2023-12-31"
}
```

**POST - Manual Correction of Time Off Balances:** From this endpoint, you can adjust the vacation balances of employees by adding or subtracting available days.

> **userEmployeeInternalId**

Type: string

Description: This is the username for whom days will be edited in the Vacation policies. This information is mandatory.

> **operation**

Type: string

Description: This field determines whether days are being added or subtracted. The options are "ADDITION" or "SUBTRACTION." This information is mandatory.

> **amount**

Type: double

Description: This field represents the number of days to be added or removed in a Vacation policy. This information is mandatory.

> **observations**

Type: string

Description: This field is visible only in the database. It serves to provide a description of the addition or subtraction of balances. This information is optional.

A possible JSON would be the following:

```
{  
  "employeeInternalId": "userEmployeeInternalId",  
  "operation": "ADDITION",  
  "amount": 1,  
  "observations": "Observations"  
}
```

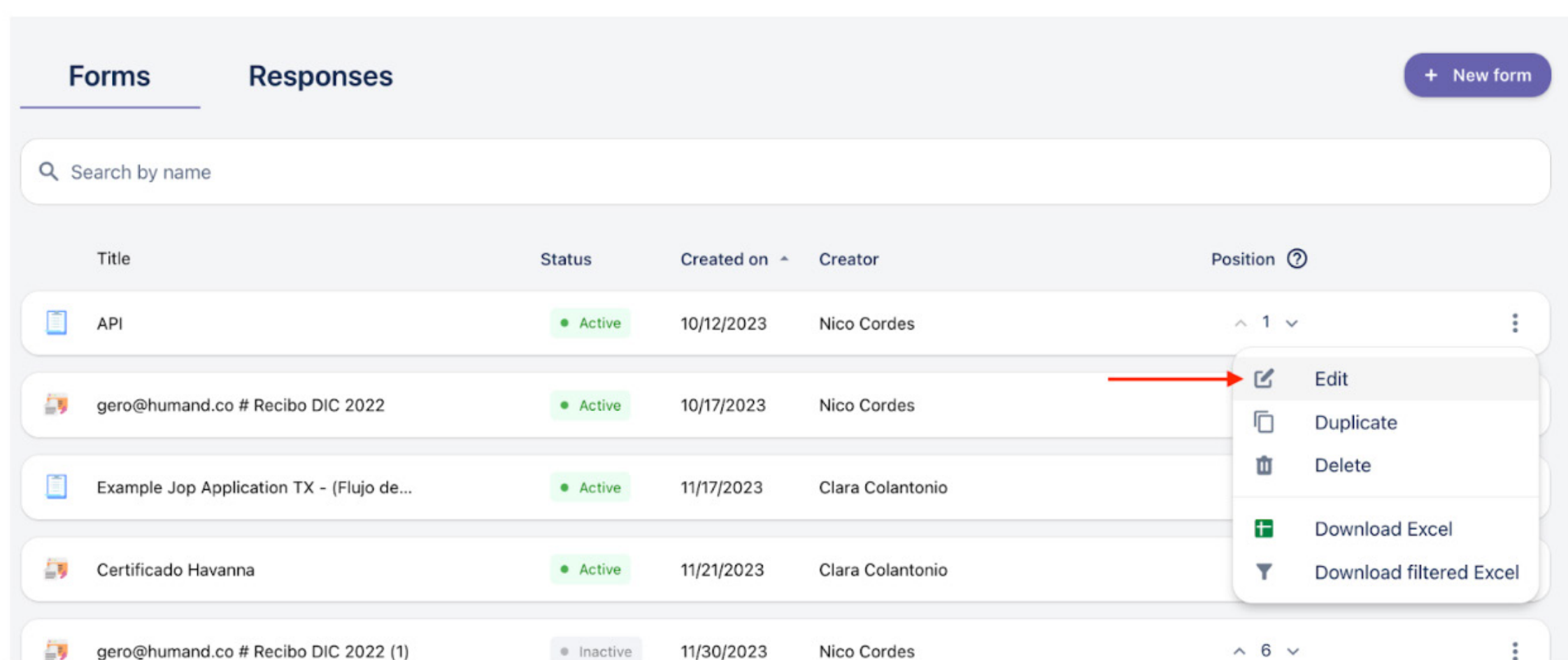
## Forms

**GET - Forms:** From this endpoint, you can obtain the results of a specific form.

### > ID

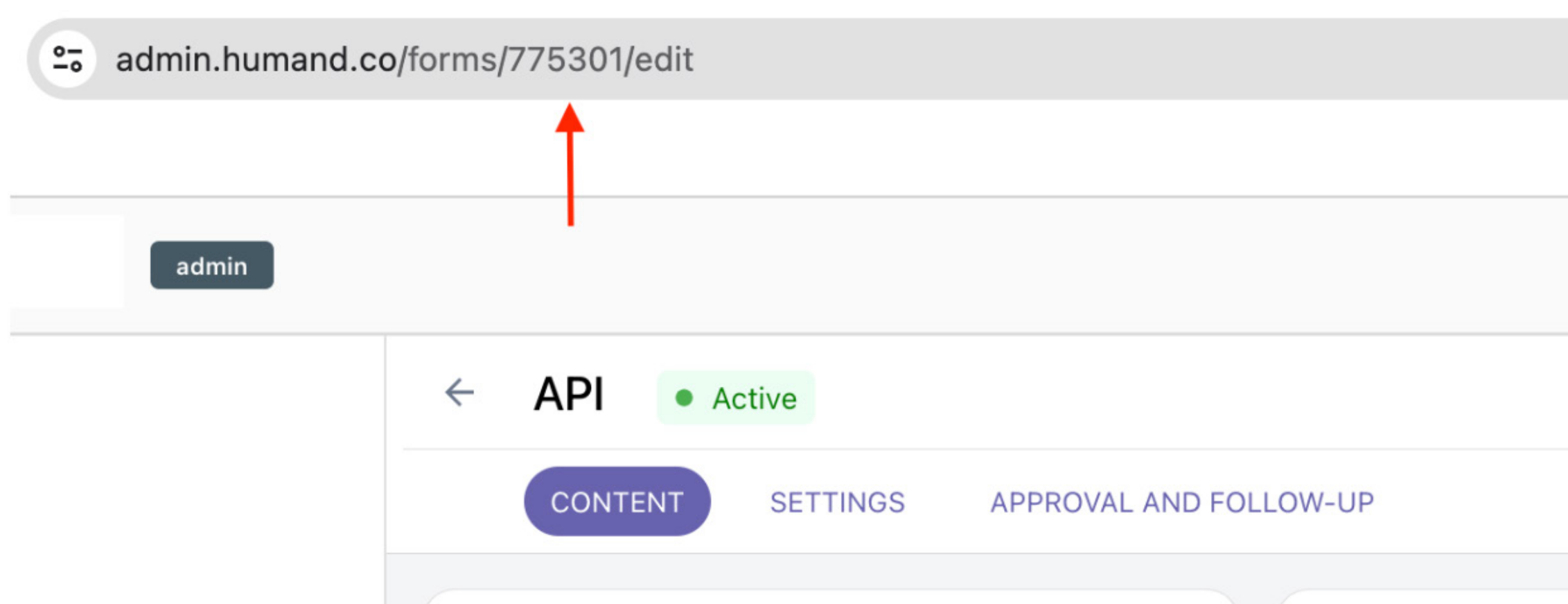
Type: Integer

Description: The 'ID' is a numeric identifier corresponding to each form. To find this value, you need to access the platform from the administrator account and navigate to the Forms section. The 'ID' can be found in the URL of the form, which can be accessed by clicking on Edit:



Title	Status	Created on	Creator	Position
API	Active	10/12/2023	Nico Cordes	1
gero@humand.co # Recibo DIC 2022	Active	10/17/2023	Nico Cordes	
Example Jop Application TX - (Flujo de...	Active	11/17/2023	Clara Colantonio	
Certificado Havana	Active	11/21/2023	Clara Colantonio	
gero@humand.co # Recibo DIC 2022 (1)	Inactive	11/30/2023	Nico Cordes	6

It's important to note that this identifier remains constant over time.





In the following example, the form ID would be: 775301

#### > page

Type: Integer

Description: The 'page' field specifies the current page number being requested. This information is mandatory. For example, "page=4" indicates that the fourth page of results is being requested.

#### > limit

Type: Integer

Description: This field defines the maximum number of items or records to be retrieved per page. The maximum allowed value is 500, controlling the size of each data segment delivered. This information is mandatory. For example, "limit=10" indicates that up to 10 records are desired per page.

#### > order

Type: String

Description: This field serves to sort the form responses to see which were Rejected and which were Approved. The possible values are: ASC, DESC, ASC\_NULLS\_FIRST, DESC\_NULLS\_LAST.

#### > orderBy

Type: String

Description: This field is used to sort the responses based on who answered the form or by the completion date. The possible values are: CREATOR, CREATED\_AT.

#### > creationDate

Type: String

Description: This field is used to search for responses on a specific date. The date format should be YYYY-MM-DD.

A possible JSON would be the following:

```
{
  "id": 1465536,
  "page": 1,
  "limit": 100,
  "order": "ASC",
  "orderBy": "CREATED_AT",
  "creationDate": "2024-05-02",
}
```

## Time Tracking (WIP)

Humand Time Tracking has two dedicated APIs to receive time records for integration into the Time Control system, which can be consumed within the platform and its reports.

### TIME TRACKING - TIME ENTRIES INTEGRATION

The following methods allow for the synchronization of time entries within Humand generated from other systems or terminals.

### General Functionalities:

- Receives time records with UTC-0 timezone: All received records must be in this timezone, as the system will reflect this information based on the timezone configured in the community.
- The body of the following endpoints allows for receiving one time record per request.
- If a record is received followed by another with the same "Type" for the same day, the last received record will be rejected.
- Both past and future records can be synchronized, ensuring that the START record is synchronized first, followed by its END record.

### POST - /time-tracking/entries/clockIn - Clock-in entries:

Receives START type time records for any collaborator in the community.

#### > employeeld

Type: Integer

Description: This is the "User" field configured within the community.

#### > Now

Type: Timestamp

Description: This is the time record to be synchronized in UTC-0.

#### > Comment

Type: String

Description: This is a free field used to add any identifier for the terminal used, which will later be included as a comment in the time record within the system.

### A possible JSON would be the following:

```
{
  "employeeld": 1,
  "now": "2021-07-10T22:50:00.010Z",
  "comment": "string"
}
```

### POST -/time-tracking/entries/clockOut - Clock-out entries:

Receives END type time records for any collaborator in the community.

#### > employeeld

Type: Integer

Description: This is the "User" field configured within the community.

#### > Now

Type: Timestamp

Description: This is the time record to be synchronized in UTC-0.

#### > Comment

Type: String

Description: This is a free field used to add any identifier for the terminal used, which will later be included as a comment in the time record within the system.

**A possible JSON would be the following:**

```
{  
  "employeeid": 1,  
  "now": "2021-07-10T22:50:00.010Z",  
  "comment": "string"  
}
```

**Support and Contact**

If you have any questions or need further assistance, please feel free to contact our support team via email or through the WhatsApp group.

 [help@humand.co](mailto:help@humand.co)

**Conclusion**

This manual provides a comprehensive introduction to the API and its usage. We hope this guide helps you effectively integrate and utilize our API.

humand

Thank you for  
choosing Humand!

Date of Edition: July 2024