



**VisionLabs**  
MACHINES CAN SEE

# **VisionLabs LUNA PLATFORM 5**

**Quick start guide**

**v.5.57.0**

## Contents

|                                             |           |
|---------------------------------------------|-----------|
| <b>Glossary</b>                             | <b>3</b>  |
| <b>Introduction</b>                         | <b>5</b>  |
| <b>1 Distribution package structure</b>     | <b>6</b>  |
| <b>2 Distribution documentation</b>         | <b>7</b>  |
| 2.1 Operation manuals . . . . .             | 7         |
| 2.2 Reference manuals . . . . .             | 7         |
| 2.3 Developer manuals . . . . .             | 9         |
| <b>3 Getting started</b>                    | <b>9</b>  |
| 3.1 Requests creation . . . . .             | 11        |
| 3.1.1 Using OpenAPI specification . . . . . | 12        |
| <b>4 Example</b>                            | <b>14</b> |
| 4.1 Create account . . . . .                | 14        |
| 4.2 Create handler . . . . .                | 15        |
| 4.3 Generate event . . . . .                | 17        |

## Glossary

| Term             | Description                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------------------|
| Attributes       | Basic attributes and descriptor.                                                                                                |
| Avatar           | Visual image of a face used in the user interface.                                                                              |
| Basic attributes | Age, gender, and ethnicity.                                                                                                     |
| Body parameters  | Body characteristics (presence of backpack, headwear, color of clothing, etc.) determined on the source image during detection. |
| Bounding box     | Rectangle that bounds the image space with the detected face or body.                                                           |
| Bucket           | Logical entity used to store objects.                                                                                           |
| Descriptor       | Data set in closed, binary format prepared by recognition system based on the characteristic being analyzed.                    |
| Detector         | Neural network used to detect either faces or bodies, or both faces and bodies in the source image.                             |
| Events           | Immutable objects that include information about a single face and/or human body.                                               |
| Face             | Changeable objects that include information about a human face.                                                                 |
| Face parameters  | Facial characteristics (emotions, mouth parameters, head position, etc.) determined on the source image during detection.       |
| Handlers         | Changeable objects that store rules for image processing.                                                                       |
| Image parameters | Image characteristics (width and height, aspect ratio, size, etc.) determined on the source image during detection.             |
| Landmarks        | Reference points on the face or body used by recognition algorithms to localize the face or body.                               |
| Matching         | The operation of matching descriptors stored in the database.                                                                   |
| Samples, Warps   | Normalized (centered and cropped) image obtained after face or body detection, prior to descriptor extraction.                  |

| Abbreviation      | Decoding |
|-------------------|----------|
| DB                | Database |
| LUNA PLATFORM     | LP, LUNA |
| LUNA PLATFORM API | API      |

| Abbreviation                       | Decoding             |
|------------------------------------|----------------------|
| LUNA PLATFORM Accounts             | API                  |
| LUNA PLATFORM Faces                | Faces                |
| LUNA PLATFORM Image Store          | Image Store          |
| LUNA PLATFORM Matcher              | Matcher              |
| LUNA PLATFORM Events               | Events               |
| LUNA PLATFORM Sender               | Sender               |
| LUNA PLATFORM Handlers             | Handlers             |
| LUNA PLATFORM Python Matcher       | Python Matcher       |
| LUNA PLATFORM Python Matcher Proxy | Python Matcher Proxy |
| LUNA PLATFORM Backport 3           | Backport 3           |
| LUNA PLATFORM Backport 4           | Backport 4           |
| LUNA PLATFORM Admin                | Admin                |
| LUNA PLATFORM Configurator         | Configurator         |
| LUNA PLATFORM Tasks                | Tasks                |
| LUNA PLATFORM Licenses             | Licenses             |

## Introduction

The “[Distribution package structure](#)” section describes the distribution package content.

The “[Distribution documentation](#)” section lists all the documents included in the distribution package.

The “[Getting started](#)” section will help you to get started with LUNA PLATFORM.

The “[Example](#)” section provides an example of sending a face recognition request to the LUNA PLATFORM with a detailed description of the request and response bodies.

## 1 Distribution package structure

The package consist of the following directories:

| Directory name         | Description                                                                   |
|------------------------|-------------------------------------------------------------------------------|
| <b>/example-docker</b> | The directory includes all the files required for launching docker containers |
| <b>/extras</b>         | Additional dependencies and helper scripts                                    |
| <b>/docs</b>           | Documentation for LUNA PLATFORM                                               |

The “extras” directory contents:

| Directory name  | Description                                                                     |
|-----------------|---------------------------------------------------------------------------------|
| <b>/conf</b>    | Configuration files for LP services and NGINX                                   |
| <b>/hasp</b>    | HASP utility and files required for license activation                          |
| <b>/utils</b>   | Utilities for working with LP                                                   |
| <b>/VLMatch</b> | Matching libraries and sources required for matching by DB using Python Matcher |

The “example-docker” directory contents:

| Directory name            | Description                                          |
|---------------------------|------------------------------------------------------|
| <b>/luna_configurator</b> | Configurations for the Configurator service          |
| <b>/logging</b>           | Files for LUNA Dashboards, Grafana Loki and Promtail |
| <b>/postgresql</b>        | Scripts for databases creation in PostgreSQL         |

## 2 Distribution documentation

This section covers the documentation package for LUNA PLATFORM. All the documents can be found in “/docs” folder of the distribution package.

### 2.1 Operation manuals

These manuals cover general LP processes, system deployment and architecture.

Most of the documents are provided in PDF and HTML formats. The PDF document is easier for navigation, the HTML document is more convenient to copy the commands.

| File                           | Description                                                                                                                                                 |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LP_Release_Notes.*             | LUNA PLATFORM Release Notes                                                                                                                                 |
| LP_Administrator_Manual.pdf    | Administrator manual for LUNA PLATFORM                                                                                                                      |
| LP_Installation_Manual.*       | General concepts for the full installation of LUNA PLATFORM using Docker containers. The example for installation on a single server is given in the manual |
| LP_License_Activation_Manual.* | The manual includes general steps for license activation                                                                                                    |
| LP_Upgrade_Manual.*            | General steps for upgrading from the previous LP build                                                                                                      |
| LP_Docker_Compose_Example.*    | General concepts for the full installation of LUNA PLATFORM using Docker Compose. The example for installation on a single server is given in the manual    |
| LP_Migration_from_LP3.*        | The manual includes general steps for migration from LUNA PLATFORM 3 to LUNA PLATFORM 5 Backport 3                                                          |
| LP_Migration_from_LP4.*        | The manual includes general steps for migration from LUNA PLATFORM 4 to LUNA PLATFORM 5 Backport 4                                                          |
| LP_Online_Documentation.html   | Link to online documentation of the current LP version                                                                                                      |

\* pdf and html formats

### 2.2 Reference manuals

These manuals describe the OpenAPI specifications for LUNA PLATFORM services. OpenAPI specification is the only valid document providing up-to-date information about the service API. The specification can be used:

- By documentation generation tools to visualize the API.

- By code generation tools.

All the documents and code generated using this specification can include inaccuracies and should be carefully checked.

The manuals can be found in the “./docs/ReferenceManuals” directory. This directory includes documents in the YAML and HTML format. Documents in YAML format contain requests to all LUNA PLATFORM services. You can use them to automatically generate requests in API testing tools, for example, Postman (not described in LUNA PLATFORM documentation). It is not guaranteed that all requests will be imported correctly, manual editing may be required. Documents in HTML format are used to visualize these specification and may not be complete.

The OpenAPI specification for the LUNA PLATFORM services can be obtained from the “get openapi documentation” request to each service. The “Accept” header should take the value “application/x-yaml”.

| File                           | Description                                                                               |
|--------------------------------|-------------------------------------------------------------------------------------------|
| APIReferenceManual.*           | This manual describes all general requests to LUNA PLATFORM services using API service    |
| AdminReferenceManual.*         | Admin service API description. Describes tasks run by the administrator                   |
| AccountsReferenceManual.*      | Accounts service API description                                                          |
| Backport3ReferenceManual.*     | Backport 3 API description                                                                |
| Backport4ReferenceManual.*     | Backport 4 API description                                                                |
| ConfiguratorReferenceManual.*  | Configurator service API description                                                      |
| EventsReferenceManual.*        | Events service API description                                                            |
| FacesReferenceManual.*         | Faces service API description                                                             |
| HandlersReferenceManual.*      | Handlers service API description                                                          |
| ImageStoreReferenceManual.*    | Image Store service API description                                                       |
| LicensesReferenceManual.*      | Licenses service API description                                                          |
| PythonMatcherReferenceManual.* | Python Matcher service API description                                                    |
| SenderReferenceManual.*        | Sender service API description. Describes receiving event notifications using web sockets |
| TasksReferenceManual.*         | Tasks service API description. Describes requests for execution of long tasks             |

\* yml and html formats



## 2.3 Developer manuals

These interactive reference guides are intended for developers and DevOps. The manuals can be found in the “./docs/ServiceManuals” directory. The manuals contains a description of the work of LUNA PLATFORM services with a detailed disclosure of technical nuances.

| File                                      | Description                                                                                      |
|-------------------------------------------|--------------------------------------------------------------------------------------------------|
| APIDevelopmentManual/index.html           | Server installation, documentation of tornado-handlers, PostgreSQL usage, admin statistics, etc. |
| AdminDevelopmentManual/index.html         | Common administrative routines                                                                   |
| AccountsDevelopmentManual/index.html      | Accounts service description                                                                     |
| Backport3DevelopmentManual/index.html     | Backport 3 service description                                                                   |
| Backport4DevelopmentManual/index.html     | Backport 4 service description                                                                   |
| ConfiguratorDevelopmentManual/index.html  | Configurator service description                                                                 |
| EventsDevelopmentManual/ Index.html       | Events service description                                                                       |
| FacesDevelopmentManual/Index.html         | Faces service description                                                                        |
| HandlersDevelopmentManual/index.html      | Handlers service description                                                                     |
| ImageStoreDevelopmentManual/index.html    | Image Store service description                                                                  |
| LicensesDevelopmentManual/index.html      | Licenses service description                                                                     |
| PythonMatcherDevelopmentManual/index.html | Python Matcher service description                                                               |
| SenderDevelopmentManual/index.html        | Sender service description                                                                       |
| TasksReferenceManual/index.html           | Tasks service description                                                                        |

## 3 Getting started

There are several useful guides to get started with the LUNA PLATFORM.

The “[LP\\_Administrator\\_Manual](#)” includes all general information about LUNA PLATFORM:

- terminology,
- image processing workflow,
- process of working with the received data,
- objects created and tasks performed,
- architecture and interaction of services,
- structures of databases,

- description of the service settings.

Before working with LUNA PLATFORM, it is recommended to read the section “General concepts” in order to understand the general principles of working with LUNA PLATFORM.

The distribution package does not include the docker containers. You need to download them from the Internet. See the “[LP\\_Installation\\_Manual](#)” for more information.

Once LUNA PLATFORM is up and running, open the document “[./docs/ReferenceManuals/APIReferenceManual.html](#)” containing a description of requests to the LUNA PLATFORM.

### 3.1 Requests creation

LUNA PLATFORM does not have a default user interface. To work with the system, you need to send requests via the API.

If necessary, you can use the LUNA CLEMENTINE 2.0 user interface (not included in the distribution package).

LUNA PLATFORM consists of several services that interact with each other. The main interface for working with the LUNA PLATFORM is the API service. The service is designed to receive user requests and redirect them to other LP services. For example, to detect a face in an image, you need to send a request to the API service, which will redirect the request to the Handlers service, where the face detection will be performed, and then the response from the Handlers service will be redirected to the API service, where the user will receive the detection result.

If necessary, you can send requests directly to other services, but this method is not recommended and is intended only for certain purposes and experienced users.

General requests to LP are sent via API service, using its URL:

```
http://<API server IP-address>:<API port>/<API Version>/
```

Here:

- <API server IP-address> - IP address where the API service is deployed
- <API port> - port where the API service is deployed. The port is set during container startup (default is 5000)
- <API Version> - API version (always 6)

Example:

```
http://10.16.8.152:5000/6/
```

Requests can be sent via CURL or using API tools (for example, Postman).

Almost all requests sent to LP 5 require authorization. There are three types of authorization available in LUNA PLATFORM:

- **BasicAuth.** Authorization by login and password (set during account creation);
- **BearerAuth.** Authorization by JWT token (issued after the token is created);
- **LunaAccountIdAuth.** Authorization by “Luna-Account-Id” header, which specifies the “account\_id” generated after creating the account.

**LunaAccountIdAuth** authorization has the lowest priority compared to other methods and can be disabled using the “ALLOW\_LUNA\_ACCOUNT\_AUTH\_HEADER” setting in the “OTHER” section of

the API service settings in the Configurator (enabled by default). In the [OpenAPI specification](#) the “Luna-Account-Id” header is marked with the word **Deprecated**.

The Configurator service contains settings for all services.

In order to use one of the types of authorization, you must have an account. The easiest way to create an account is to send a POST request “create account” to the API service. When creating an account, you must specify the following data: login (email), password and account type (account type). The account is created at the installation stage after the launch of the API service.

If the authorization type is not specified in the request, an error with status code 403 will be returned.

### 3.1.1 Using OpenAPI specification

Specification includes:

- Required resources and methods for requests sending.
- Request parameters description.
- Response description.
- Examples of the requests and responses.

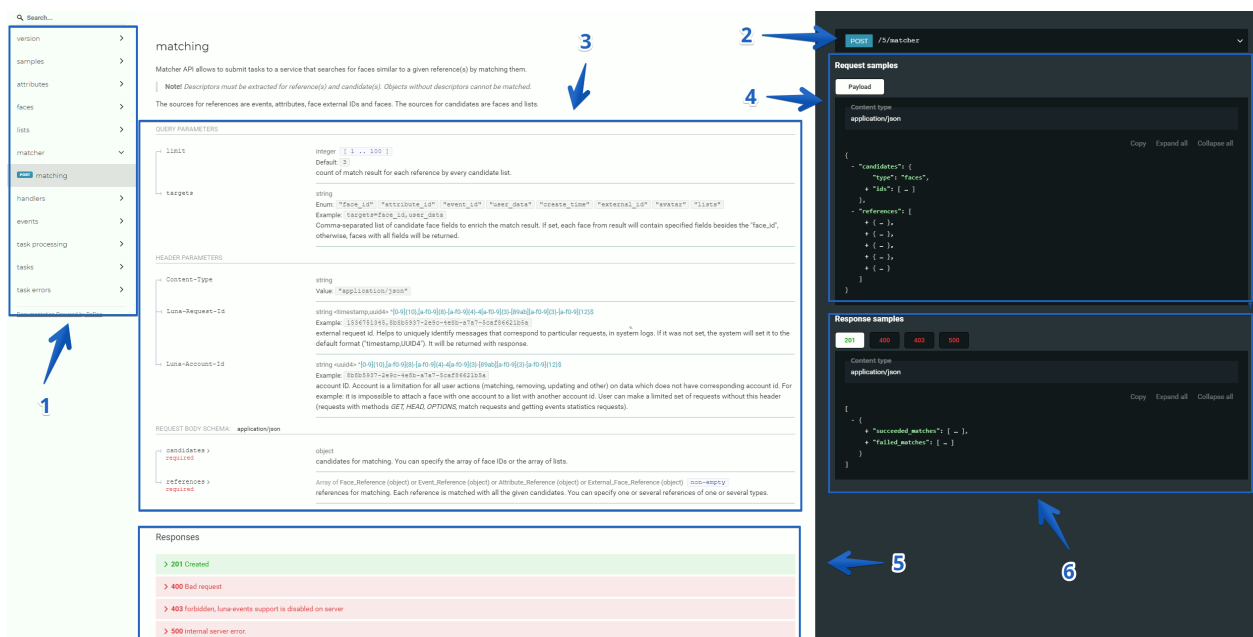


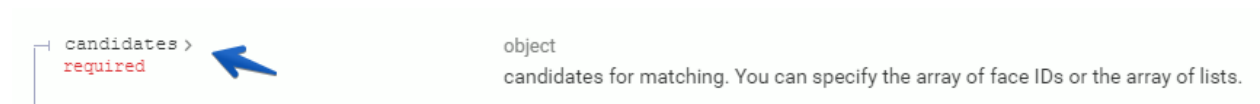
Figure 1: OpenAPI documentation

The HTML document includes the following elements:

1. Requests, divided into groups.
2. Request method and URL example. You should use it with your protocol, IP-address, and port to create a request. Example: POST `http://<IP>:<PORT>/<Version>/matcher`.

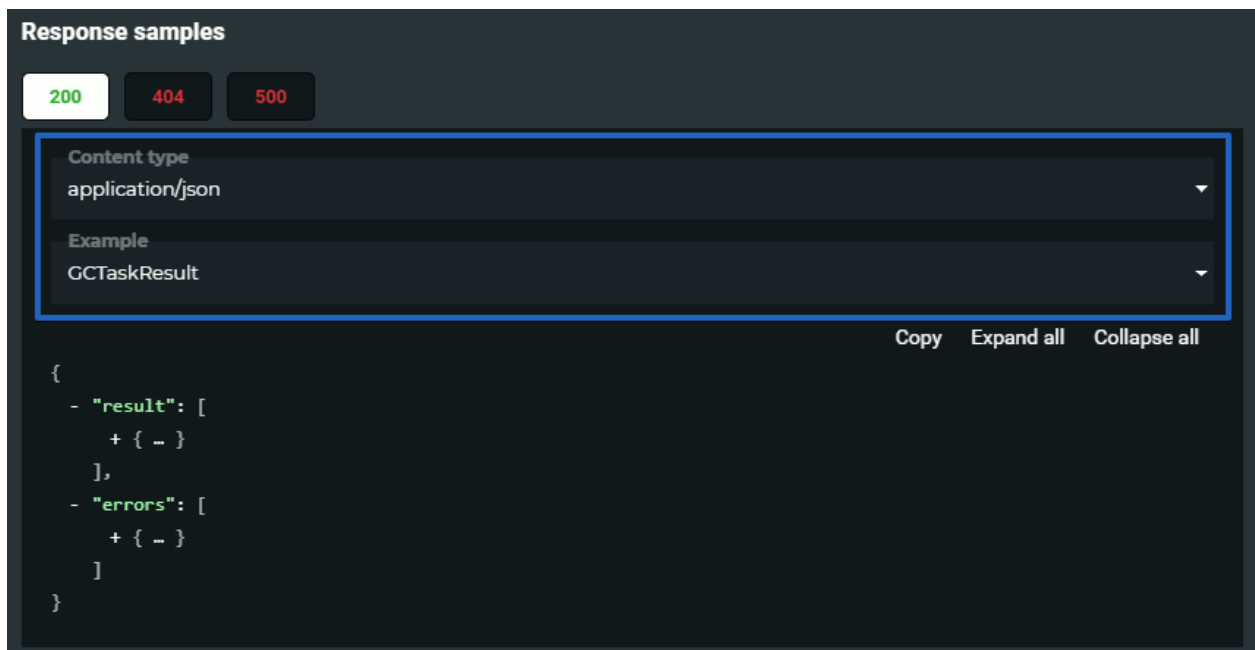
3. Description of request path parameters, query parameters, header parameters, body schema.
4. Example of the request body.
5. Description of responses.
6. Examples of responses.

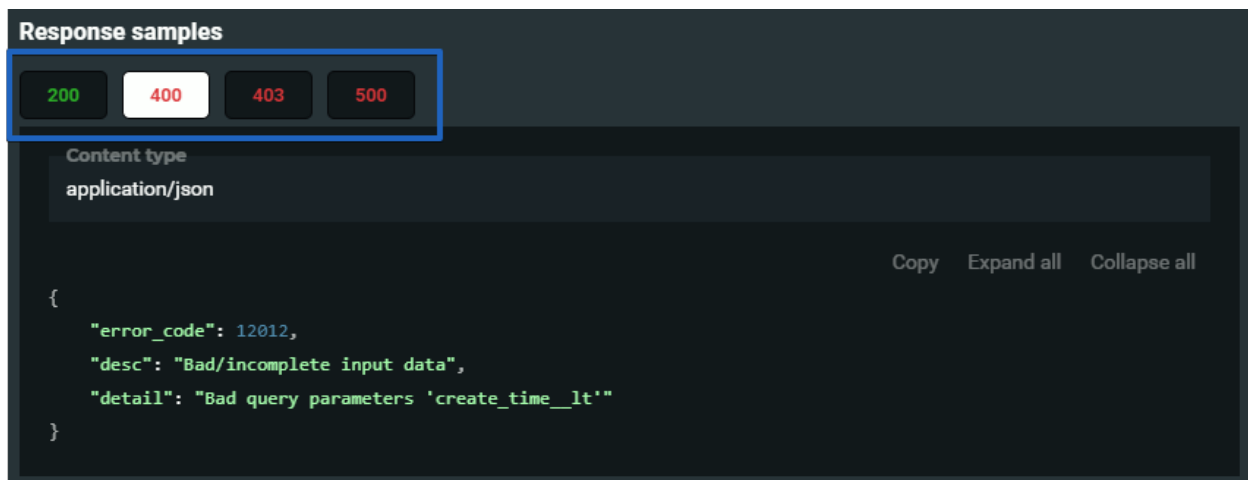
You can expand descriptions for request body parameters or response parameters using the corresponding icon.



**Figure 2:** Expand descriptions

You can select the required example for request body or response in corresponding windows.





**Figure 4:** Response example

When specifying filters for requests you must use a full value, unless otherwise noted. The possibility of using part of the value is indicated in the description.

## 4 Example

There are two main approaches when performing the requests.

The first and main approach is to set the rules of detection, estimation, extraction, matching, etc. in a single **handler** object. After that, you need to create an object **event**, which will give a result based on all the rules specified in the handler. Using this approach is the most optimal from the point of view of business logic.

The second approach is to sequential performing of requests, i.e. in one request you need to perform a face detection and get its result, then use this result in an extraction request and so on.

This section provides an example of using handlers and events using CURL requests with a detailed description of the request bodies and responses.

See the detailed information about the approaches in the section “Approaches at work” in the administrator’s guide.

### 4.1 Create account

Without an account, it is impossible to send most requests to the LUNA PLATFORM. The account is created at the installation stage after launching the API service. Create the account using the following command, if it has not been created yet:

```
curl --location --request POST 'http://127.0.0.1:5000/6/accounts' \
--header 'Content-Type: application/json' \
--data '{
  "login": "user@mail.com",
  "password": "password",
  "account_type": "user",
  "description": "description"
}'
```

To use BasicAuth authorization in a CURL request, you need to convert your username and password to Base64 format. To convert, you need to specify a username and password in the format `login:password`. In this case, `user:password`. The Base64-encoded login and password look like this:

```
dXNlcjpwYXNzd29yZA==
```

## 4.2 Create handler

This section provides an example of creating a simple handler using CURL requests. Using this handler, it will be possible to:

- detect the face
- define basic attributes (gender, age)
- extract its descriptor
- save a sample to a disk (in a bucket)
- save a face to the Faces database

**Description of request:** The request enables you to create a handler.

**Type of request:** POST

**Request:** `http://127.0.0.1:5000/6/handlers`

The request to create a handler is executed with the parameters specified below:

| Parameter name         | Description                                                 | Value                |
|------------------------|-------------------------------------------------------------|----------------------|
| <b>Request headers</b> |                                                             |                      |
| Content-Type           | Type of content in the request                              | application/json     |
| Authorization          | Basic-authorization. Username and password in Base64 format | dXNlcjpwYXNzd29yZA== |
| <b>Request body</b>    |                                                             | See below            |

| Parameter name | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Value |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
|                | <p>Data entry string.</p> <p>It is necessary to specify:</p> <ul style="list-style-type: none"> <li>• description - description of the handler</li> <li>• policies - handler policies</li> <li>• policies &gt; detect_policy &gt; detect_face - face detection parameter</li> <li>• policies &gt; extract_policy &gt; extract_basic_attributes - parameter for extracting basic attributes</li> <li>• policies &gt; extract_policy &gt; extract_face_descriptor - parameter for extracting descriptor</li> <li>• policies &gt; storage_policy &gt; face_policy &gt; store_face - parameter for save face in the Faces database</li> <li>• policies &gt; handler_type - "0" - static, "1" - dynamic, "2" - lambda</li> </ul> |       |

Example of a CURL request, to execute a request from the command line:

```
curl --location --request POST 'http://127.0.0.1:5000/6/handlers' \
--header 'Authorization: Basic dXNlcjpwYXNzd29yZA==' \
--header 'Content-Type: application/json' \
--data '{
  "description": "Simple handler",
  "policies": {
    "detect_policy": {
      "detect_face": 1
    },
    "extract_policy": {
      "extract_basic_attributes": 1,
      "extract_face_descriptor": 1
    },
    "storage_policy": {
      "face_sample_policy": {
        "store_sample": 1
      }
      "face_policy": {
        "store_face": 1
      }
    }
  }
}
```



```

    },
    "is_dynamic": false
  }'

```

If the request is successful, the system returns the handler ID and its address.

Example of a response to a request:

```

{"handler_id":"f2831884-65b9-4b94-9639-f10f4d5f042d","url":"\\6\\handlers\\f2831884-65b9-4b94-9639-f10f4d5f042d", "external_url": "http://127.0.0.1:5000\\6\\handlers\\28e6358a-3753-4442-8310-617a8bba14bf"}

```

After the handler is created, its ID (parameter `handler_id`), account ID (parameter `account_id`), creation time (parameter `create_time`), last update time (parameter `last_update_time`), description (parameter `description`) and policies (section `policies`) are recorded in the table `handler` of the `Handlers` database.

At this stage, nothing else is saved to any databases. A handler is a simple set of rules that cannot create any objects other than itself. In order to use these rules on a certain image, you need to generate an event.

### 4.3 Generate event

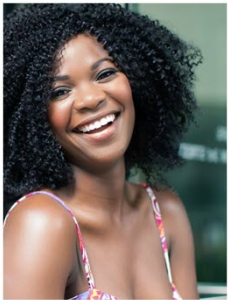
**Description of request:** The request enables you to generate events and process them with the appropriate handler.

**Type of request:** POST

**Request:** `http://127.0.0.1:5000/6/handlers/{handler_id}/events`

The request to generate an event is executed with the parameters specified below:

| Parameter name           | Description                                                 | Value                                |
|--------------------------|-------------------------------------------------------------|--------------------------------------|
| <b>Request parameter</b> |                                                             |                                      |
| <code>handler_id</code>  | ID of the created handler                                   | f2831884-65b9-4b94-9639-f10f4d5f042d |
| <b>Request headers</b>   |                                                             |                                      |
| Content-Type             | Type of content in the request                              | image/jpeg                           |
| Authorization            | Basic-authorization. Username and password in Base64 format | dXNlcjpwYXNzd29yZA==                 |
| <b>Request body</b>      |                                                             |                                      |

| Parameter name | Description                                                                       | Value              |
|----------------|-----------------------------------------------------------------------------------|--------------------|
|                | Jpeg image transferred to the server where LP is deployed:                        | @/root/example.jpg |
|                |  |                    |

Example of a CURL request, to execute a request from the command line:

```
curl --location --request POST 'http://127.0.0.1:5000/6/handlers/f2831884-65b9-4b94-9639-f10f4d5f042d/events' \
--header 'Authorization: Basic dXNlcjpwYXNzd29yZA==' \
--header 'Content-Type: image/jpeg' \
--data-binary '@/root/example.jpg'
```

If necessary, you can specify an image in Base64 format without transferring it to the server from LP.

If the request is successful, the system creates an event.

Example of a response to a request sorted by blocks:

1 The “images” block defines the general data about the image. If the image was processed unsuccessfully, the status “0” and an error code with a description and a link will be returned. The “exif” parameter contains information about the metadata of the processed image.

```
"images": [
  {
    "filename": "raw image",
    "status": 1,
    "error": {
      "error_code": 0,
      "desc": "Success",
      "detail": "Success",
      "link": "https:\\\\docs.visionlabs.ai\\info\\luna\\troubleshooting\\errors-description\\code-0"
    }
  },

```

```

        "exif": {}
    }
],

```

2. The “events” block, which defines all the information about the generated event.

```

"events": [
    ...
]

```

2.1. The “face\_attributes” sub-block defines the attributes extracted using the “extract\_policy” handler policy, namely, the basic attributes (the “extract\_basic\_attributes” handler parameter) and the quality of the descriptor (the “extract\_face\_descriptor” handler parameter).

The descriptor itself is not issued in the response to the request. See the section “Descriptor formats” in the administrator manual.

Note that the “attribute\_id” parameter and its “url” are “null”. This indicates that the attributes were not saved to the Redis database because the “store\_attribute” parameter is not enabled for the “storage\_policy” > “attribute\_policy” policy. However, at the moment of creating the “face” object, attributes are attached to it. Since we are creating a face and saving it to the Faces database (the “store\_face” handler parameter), a face with associated attributes will be visible in the Faces database. That is, attributes are issued in the response and stored in the Faces database.

Since the “store\_sample” handler parameter of the “face\_sample\_policy” policy is enabled, the “sample” object will be created and stored in a bucket (the Image Store service contains a link to this bucket).

The “store\_sample” handler parameter of the “face\_sample\_policy” policy is always enabled by default. We explicitly specified it when creating the handler. If you delete it from the request body, the sample will still be created. You need to set the parameter value to 0.

The “samples” field of the “face\_attributes” block specifies the identifier of the sample from which the attributes were extracted. Full information on the sample is provided below in the “detections” sub-block.

```

{
    "face_attributes": {
        "attribute_id": null,
        "url": null,
        "samples": [
            "b92b05cd-c871-4533-89ce-0a538f8227d3"
        ],
        "basic_attributes": {
            "ethnicities": {

```

```

        "predominant_ethnicity": "african_american",
        "estimations": {
            "asian": 5.721812167271785e-15,
            "indian": 1.3687103486030773e-16,
            "caucasian": 3.082826702249797e-11,
            "african_american": 1.0
        },
        "age": 25,
        "gender": 0
    },
    "score": 0.8616658210754395
},
}

```

2.2 The “detections” sub-block defines a list of detections detected in the image using the “detect\_policy”. The coordinates of the bounding box of the face (“rect”), five landmarks of the face (“landmarks\_5”), the address (“url”) to the Image Store service (which contains the address to the “visionlabs-samples” bucket with a sample) and the sample identifier (“sample\_id”) are specified here. The field “image\_origin” has the value “null” because the policy of saving the source image was not enabled - “storage\_policy” > “image\_origin\_policy”. If it had been enabled, the source image would have been saved in the Image Store service as a link to the “visionlabs-image-origin” bucket. The “body” field has the value “null” because the “detect\_body” handler parameter was not enabled.

```

"detections": [
    {
        "filename": "raw image",
        "samples": {
            "face": {
                "detection": {
                    "rect": {
                        "x": 103,
                        "y": 74,
                        "width": 195,
                        "height": 275
                    },
                    "landmarks5": [
                        [
                            29,
                            129
                        ],
                        [
                            123,

```

```

        90
      ],
      [
        91,
        148
      ],
      [
        51,
        211
      ],
      [
        160,
        171
      ]
    ],
    },
    "url": "\\6\samples\faces\b92b05cd-c871-4533-89ce-0a538f8227d3",
    "sample_id": "b92b05cd-c871-4533-89ce-0a538f8227d3"
  },
  "body": null
},
"detect_time": "2022-11-28T17:01:09.572910+03:00",
"image_origin": null,
"detect_ts": null
}
],

```

2.3 The “aggregate\_estimations” sub-block is responsible for the general parameters of the face or body obtained from different images. Combining several parameters (for example, age and gender) of one person into one parameter is called **aggregation**. This sub-block is empty because the request for event generation did not specify the “aggregate\_attributes” request parameter.

```

"aggregate_estimations": {
  "face": {
    "attributes": {}
  },
  "body": {
    "attributes": {}
  }
},

```

2.4 The “face” sub-block defines the “face” object obtained as a result of event generation. If the “store\_face” handler parameter of the “storage\_policy” > “face\_policy” policy had been disabled, then

the “face” field would have been equal to the “null” value, i.e. it would not have been saved to the Faces database. The “avatar” parameter specifies the address of the stored sample that will be used as an avatar for the face. Note that the handler parameter “set\_sample\_as\_avatar” from the policy “storage\_policy” > “face\_policy” is responsible for enabling saving the address to the avatar. We didn’t specify this parameter when creating the handler, but it is enabled by default. The “lists” field is empty because we did not specify the policy of linking a face to the list (“link\_to\_lists\_policy”). The “event\_id” field specifies the event ID, which is also stored in the Faces database. The “external\_id” and “user\_data” fields are empty because they were not specified in the event generation request.

```
"face": {
  "external_id": "",
  "face_id": "854f4b56-9a77-4be3-bc6e-797b8f3319ad",
  "user_data": "",
  "url": "\\6\\faces\\854f4b56-9a77-4be3-bc6e-797b8f3319ad",
  "lists": [],
  "avatar": "\\6\\samples\\faces\\b92b05cd-c871-4533-89ce-0a538f8227d3",
  "event_id": "5aaa902f-7085-4dc0-810a-09d170e35c0b"
}
```

2.5 The “filtered\_detections” sub-block will have content if any filter is specified in the filters of the “match\_policy” and the image will not pass according to the specified conditions.

```
"filtered_detections": {
  "face_detections": []
}
```

2.6 Other parameters. The parameters “location”, “user\_data”, “external\_id”, “track\_id”, “tags”, “source” are set in the parameters of the event generation request (see the section “Event object” of the administrator manual). The “matches” field is filled in when using the “match\_policy” policy. The “body\_attributes” parameter will be filled in if parameters from the “body\_attributes” parameter group of the “detect\_policy” policy are enabled.