Project no. VI20172020068

Tools and Methods for Video and Image Processing to Improve Effectivity of Rescue and Security Services Operations (VRASSEO)

System architecture and ViAn Server

report vContinuous

Jaroslav Zendulka, Vladimír Bartík, Tomáš Volf, Radim Kocman

Brno University of Technology
Faculty of Information Technology
Božetěchova 1
Brno, 612 66, Czech Republic

December 2020

Contents

Abstract

This technical report describes overview of a system for managing video-data and metadata and one of its component - ViAn server. This component is responsible for storing and accessing videos and for storing, reading and querying data extracted from them by sensing modules. The report describes APIs provided by the ViAn server, video system supporting basic operations with video streams and a simple GUI for ViAn fundamental functionality demonstration.

1 Basic system overview

1.1 Hardware architecture

The hardware architecture of the system consists of sensing devices, field stations and a main central server. The deployment of the software components to this hardware nodes is shown in Fig. 1.

1. *Sensing device* — The task of the sensing device, typically a camera, is to acquire video data for further processing. Input video data will be captured by one or more cameras, will be streamed by the system's field station. At the same time, however, these video data on the input side may be pre-processed and streamed together with extracted metadata to the field station.

2. *Field stations* — Here, images/video and metadata extracted from external or internal modules are stored in the system. Field station system allows moving data to a central server and query data stored on the central server.

3. *Central server* — The server stores the previously captured data, namely images/videos and their metadata.

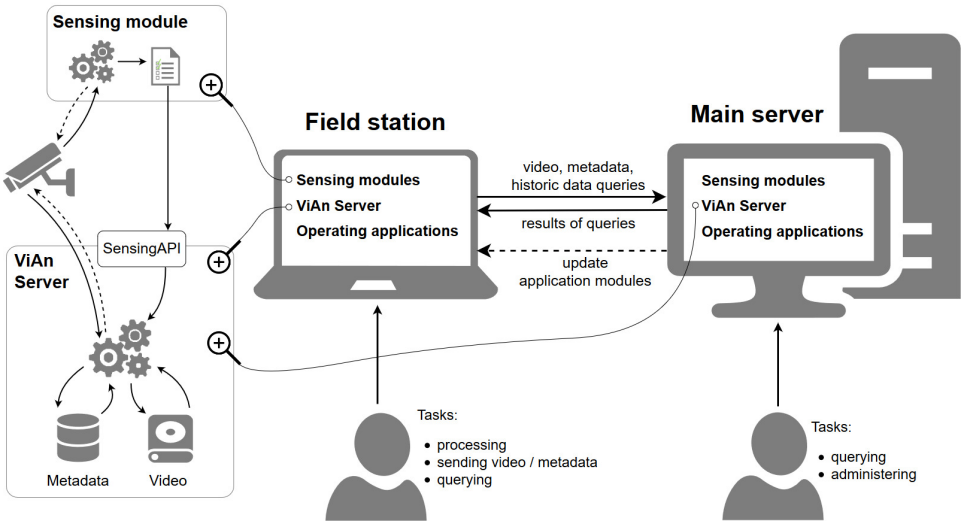

Figure 1: Basic concept of data storage in VRASSEO

1.2 Logical architecture

The logical architecture of the system consists of ViAn Server (VideoAnalysis Server), sensing modules and operational applications.

1. *ViAn Server* provides management of video data and metadata extracted from them. It provides database and basic analytical services. Its task is to support the management of video data and extracted metadata, including the support of some analytic tasks over these metadata ViAn Server provides the following set of APIs:
   (a) *ViAn SensingAPI* — the structure of data extracted from video by sensing modules definition and storing the data. Sensing modules communicate with ViAn Server via this API only.
   (b) *ViAnAPI* —- provides services to end users and to applications they use.
2. *Sensing modules* primarily serve to extract metadata from data captured by sensing devices. An example of such a module may be one extracting information about moving objects as people, cars etc. in the observed area. Several sensing modules can be located at one computing node.
3. *Operational applications* define the required task, manage its execution and visualize the results.

ViAn Server with processing and analytic services and operational applications can be deployed on both the field station and the main server. The field station deployments can provide limited functionality compared to the main server one, for example only some processing and analytic modules. Some modules can be physically deployed to devices other than a field station or a central server, such as a computer dedicated to more computationally intensive metadata extraction.

Field stations receive multimedia data and metadata from the sensing device and allow the user to quickly respond to the processing of this data. In addition, the user should be able to decide which data and metadata from his station should be synchronized with the main server, or query the server for relevant information (e.g., previous occurrences of the object in previous analyses).

The communication of the three basic system components in a case of starting a sensing module and receiving data produced by it is depicted in Fig. 2. First, the user logs on the application, which authenticates him to ViAn Server (1). Then he selects a sensing module he wants to use. The application stores the information about the selected module in ViAn Server, obtains an authentication token for the module (2) and sends the request to run the module to a corresponding node of modules (3). The module first sends a header specifying the structure of data that will be sent (4) and sends data repeatedly.

The communication is designed such a way that each of these components can be located on a separate physical device.

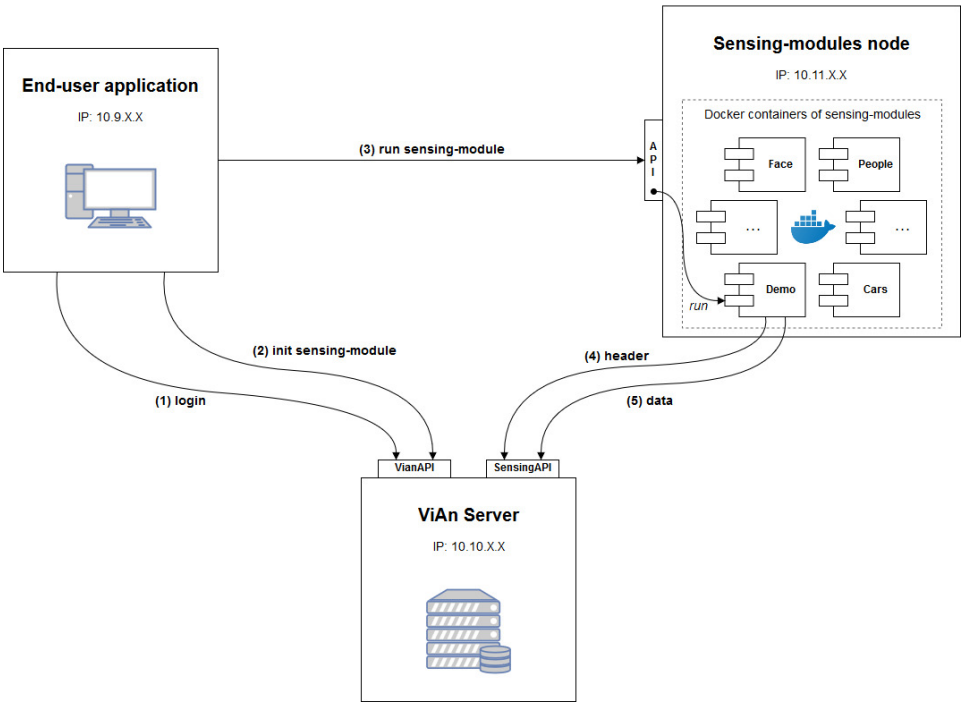

Figure 2: Communication of basic system components

2 Database schema

The schema shown in Figure 3 allows storage of datasets from various sources of video data. It also contains tables designed for storage of information about the sources, e.g. cameras, sensing modules etc. The PostgreSQL database is used for the storage.

The ViAn database consists of the `PUBLIC` schema and the schemas for each dataset. The main schema (`PUBLIC`) contains general information about datasets stored in the database, users which create them, cameras and sensing modules. Information about sensing modules includes types of sensing modules (`Sensetypes`), sensing modules (`Sensemodules`) and instances of sensing modules (`Sensemodules_instances`). The instances can be grouped in the sensing module nodes (`Sensenodes`). Information about detections, which are performed by sensing modules, are stored in the `Dataset_sensemodules_configs` table. Here, for example, information about the dataset for output data, the input camera or the start time and finish time of the detection are stored. The `fetch_stream` attribute indicates if the video data are also stored for the detection. The `Sequences` table represents the video data, if it is stored together with data from the sensing modules.

For each dataset there is a new schema in the PostgreSQL database created. Its name is derived from value of the `code_name` attribute in the table `Datasets`. The main table of the schema for a dataset is `Configs`, which is connected with the `Dataset_sensemodules_configs` table in the `PUBLIC` schema. Information about access token and time information are stored as well. Data from the sensing modules are stored into `Sensedata_cars_XXX` or `Sensedata_face_XXX` tables. Their names are derived from the `data_storage` attribute values in the `Configs` table. If the version of a sensing module is changed, information about the older version's storage, attributes and data types is stored into the `Sensemodule_datastorage` table.

ER diagram 2019.png

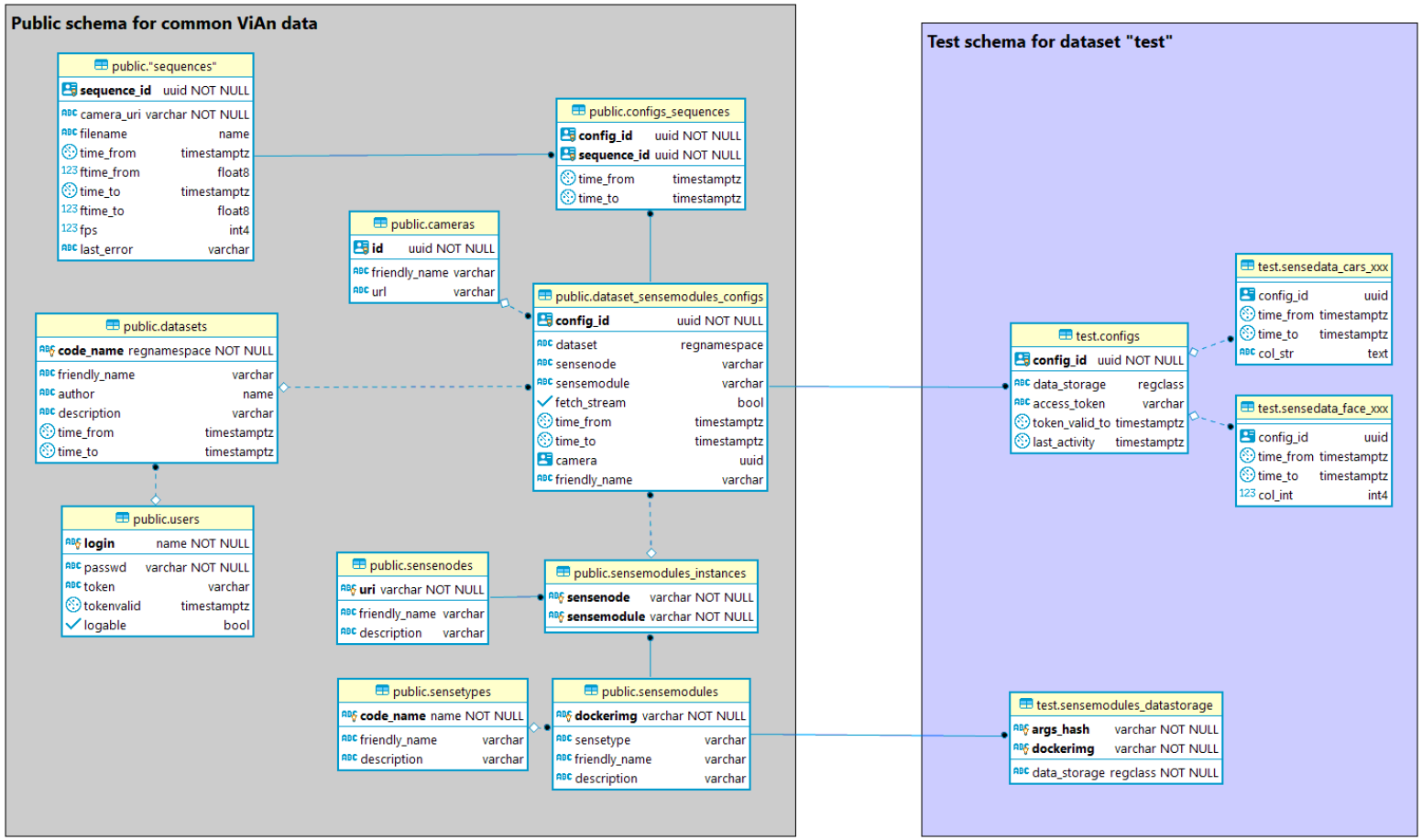

Figure 3: ViAn Database design

3 Video System

This section gives a detailed description on how video data are handled in the system. It provides insight into the used notions, technologies, and principles. Note that the concepts described here are used throughout the whole system: ViAn server, sensing modules, and operational applications.

3.1 Basic Notions

The proposed video system can be roughly divided into the following four essential parts. We always give a brief introduction of the notion followed with further technical details.

Video Streams

When video data needs to be transfered either from an outside source (camera, drone, etc.) into the system or between different parts of the system (ViAn server, sensing modules, and operational applications), it is done with video streams.

Inside the system, we always use the encrypted video stream protocol RTMPS (Real-Time Messaging Protocol over a TLS/SSL connection). All video streams from outside sources are automatically and seamlessly converted into this format.

Video Files

When a video stream needs to be archived for later use, it is stored as a standard video file on the ViAn server.

We use the file format FLV (Flash Video), which is fully compatible with RTMPS and can store all the data without any additional complex format conversions.

Real-Time Timestamps

When a sensing module detects an event in the video stream, this event needs to be linked with a specific real time and a video frame. For this purpose we use real-time timestamps that are transfered inside RTMPS.

We use the standard unix time as a representation for the current real time. More specifically, since 32-bit timestamps in video streams are not able to carry the full unix time with milliseconds, we modify the RTMPS stream in such a way that each video frame carries a timestamp that represents the current unix time starting from the day when the stream was launched. Then, if a component of the system knows the launch date of the stream, the full unix time can be easily reconstructed.

Video Boxes

To standardize the workflow with the proposed video system, we have prepared a variety of preprogrammed units called video boxes that can handle common complex video tasks (such as conversions between different formats of video streams, the preparation of real-time timestamps, etc.). The ViAn server and sensing modules commonly use these video boxes to handle the required tasks. More details can be found in the following section.

3.2 The Structure of Video Boxes

Video boxes are designed as Docker containers that can self-sufficiently perform their required function. Structurally, the content of the boxes can be divided into the following parts:

1. the python script that controls the function of the box,
2. the ffmpeg command that handles the video data,
3. the nginx server that distributes the result outside, and
4. the stunnel that handles the encrypted connection.

Not all four parts are always active, their usage varies depending on the specific task (e.g., when a video box archives a video stream, it does not require the nginx server and stunnel).

The following types of video boxes are currently finished and partially integrated into the system:

- *Replicator* — standardizes an input video stream (RTMPS + real-time timestamps) and replicates it for multiple clients. The input is usually an outside video source and the clients are usually sensing modules or other video boxes.
- *Recorder* — records the standardized input video stream into a video file. This video box is used primarily in the ViAn server for the archiving of video streams.
- *Replay* — replays the recorded video file for the use in operational applications. This video box is usually run by the ViAn server, and the operational application can traverse the stream in an arbitrary speed.
- *Restreamer* — replays the recorded video file as a new live stream. This box is useful in cases where we need to use a recoded video file as a new live outside video source.

4 ViAn SensingAPI

ViAn SensingAPI is a REST API interface provided by ViAn Server, which allows a sensing module to sent data to store in ViAn Server. This API uses general mechanism to store data at runtime, which means that it is not needed to predefine data on the server-side in advance - storage is created ad-hoc. Another benefit is simplification for developers in cases, when the data type ouput is changed; then ViAn server creates new storage for this data.

For each request it is needed at least following parts of request:

- `dataset` - passed in URI query string
- `access-token` - passed in request header

ViAn SensingAPI has a very limited number of endpoints, which are divided into two groups depending on what they are processing:

- **data manipulation** endpoints *(for sensing modules)*:
  - `/data/header` - for definition data variables and their types
  - `/data/store`- for sending (storing) data itself
- **video capturing** / **recording** endpoints *(for controlling video stream capture on the ViAn server side)*:
  - `/video/capture/start` - to start recording
  - `/video/capture/stop` - to stop recording

4.1 Data manipulation endpoints

As it was mentioned, these endpoints are designed for sensing modules purposes to transfer data of events detected by sensing modules to the ViAn server data storage. There is clearly set mandatory sequence of endpoints usage:

1. Each module uses `/data/header` endpoint first to define form of data storage.
2. Only afterwards the module can use `/data/store` endpoint as much as it needed to transfer detection data itself.

4.1.1 Responses of data manipulation endpoints request:

HTTP-code of succeeded response: `200`

```
{
  "status": "success",
  "data": {
```

```
    "access_token": "***",
    "token_validity": "2021-01-10T22:35:27+01:00"
  }
}
```

Each endpoint returns HTTP-code `200` if request has succeeded. At the same time following parts are or may be returned:

- `access_token` - new access token required for next request
- `token_validity` - timestamp of token validity otherwise token expires *At the moment returned token is always the same token for easier development and testing purposes of API endpoints.*

4.1.2 Header endpoint: `/data/header`

Header endpoint is designed for data storage creation / preparation. It must be used by sensing module before `/data/store` endpoint usage. A body of request contains JSON of key-value pairs, where key is name of event property and value is a data type of event property itself.

cURL example request:

```
curl -X POST "https://localhost/vian_sensingapi/data/header?da
↪  taset=test" -H  "accept: application/json" -H  "access-tok
↪  en: ***" -H  "Content-Type: application/json" -d "{\"fo
↪  o\":\"int\", \"bar\":\"string\"}"
```

4.1.3 Data store endpoints: `/data/store`

Data store endpoint is designed for data sending (storing) itself. It may be used repeatedly, but always mandatory after the `/data/header` endpoint use only. This endpoint offers 2 modes of data processing in every request:

- single-entry mode
- multiple-entries / batch mode These two modes are distingushed based on the data format only and so there is no need to use some distinguished parametr in the endpoint request.

cURL example request (single-entry):

In this single-entry mode of request a data entry is represented as standard JSON object / collection.

```
curl -X POST "https://localhost/vian_sensingapi/data/store?dat
↪  aset=test" -H  "accept: application/json" -H  "access-toke
↪  n: ***" -H  "Content-Type: application/json" -d "{\"foo\":
↪  1234, \"bar\":\"ViAn module test\"}"
```

cURL request (multiple-entries / batch):

In this batch (multiple-entries) mode of request the data entries are represented as standard JSON array of individual entries, where each individual entry is represented as standard JSON object / collection (similar to single-entry mode)

```
curl -X POST "https://localhost/vian_sensingapi/data/store?dat
↪  aset=test" -H  "accept: application/json" -H  "access-toke
↪  n: ***" -H  "Content-Type: application/json" -d "[{\"fo
↪  o\":1234, \"bar\":\"ViAn module test\"}, {\"foo\":5678,
↪  \"bar\":\"ViAn module test 2\"}]"
```

4.2 Video capturing / recording endpoints

ViAn SensingAPI contains also capturing / recording endpoints, which allows to manage recording stream on the ViAn server side.

4.2.1 Start capturing endpoint: `/video/capture/start`

This endpoint arranges, that the necessary records in the database is filled and run on the background the auxiliary script for recording stream to the sequence file on ViAn server datastorage side. This auxiliary script reads an important information from the stream (for example synchronization timestamp) and add it to the particular records about sequence in the database, it internally uses `recorder` VideoBox for the recording stream itself.

cURL example request:

```
curl -X POST "https://localhost/vian_sensingapi/video/capture/
↪  start?dataset=test" -H  "accept: application/json" -H  "ac
↪  cess-token: ***" -H  "Content-Type: application/json" -d "
↪  {\"camera\":\"rtmps://localhost/camera/live\"}"
```

Response of request:

HTTP-code of succeeded response: `200`

```
{
  "status": "success",
  "data": {
    "capture_id": "2e42a800-e81f-475c-a9ea-6b6c86536cab"
  }
}
```

4.2.2 Stop capturing endpoint: `/video/capture/stop`

This endpoint finishes particular records about recorded sequence in the database and stop the background script running in endpoint `video/capture/start`.

cURL example request:

```
curl -X POST "https://localhost/vian_sensingapi/video/capture/
↪  stop?dataset=test" -H  "accept: application/json" -H  "acc
↪  ess-token: ***" -H  "Content-Type: application/json" -d "
↪  {\"capture\":\"2e42a800-e81f-475c-a9ea-6b6c86536cab\"}"
```

Response of request:

HTTP-code of succeeded response: `200`

```
{
  "status": "success",
  "data": []
}
```

5 SensingMQ2ViAn sender

SensingMQ2ViAn sender was created to facilitate implementation of a specific ViAn sensing module. It shades a programmer of the specific module from the knowledge about the communication between module itself and the ViAn server. It also fully replaced previous `Base for C++ sensing module` (which is now deprecated), nevertheless it is build on Base for c++ sensing module foundations and some of its parts.

SensingMQ2ViAn sender is now conceived per module, so for each module is required itself instance of sender. It is necessary to run SensingMQ2ViAn sender first, MQ address is passed as argument on module startup. SensingMQ2ViAn sender is based on message queues, specifically `RabitMQ`. Workflow is following: when sensing module detects new event, sensing module inserts detected data into message queue. On the opposite site, where SensingMQ2ViAn listens and waits for new data in message queue, the data was readed, transformed and sent to ViAn server autonomously without any further intervention of sensing module developer. For this exchange it is determined partly mandatory format of message:

Mandatory format of message for queue

```
{
  "vian_token": "...",        # Access token of sensing module
    (passed as argument on module startup)
  "vian_server_url": "...",  # Base URL of ViAn server (passed
    as argument on module startup)
  "vian_project_id": "...",  # Dataset / work space of data (
    passed as argument on module startup)
  "header": {
    # Collection of ALL features and their types of sensing
    module (in format "feature": "data type")
    ...
  },
  "data": {
    "frame_ts": <some tst>   # MANDATORY timestamp from
    Replicator VideoBox
    # Collection of features and their values (in format "
    feature": "value"
    ...
  }
}
```

Feature `frame_ts` is not necessary to define in header, because it is automatically added during request processing on ViAn server side, on data section the value of `frame_ts` is expected (it is mandatory item).

6 ViAnAPI

ViAn VianAPI is a REST API interface provided by ViAn Server, which allows end users and applications to make queries and other database operations on the data stored in the ViAn database. It allows to get metadata information about sensing modules, cameras and datasets. It also provides functionality to start or stop the detection and to read data obtained as a result of various detections. It is also planned to add functionality that enables to replay a video, which is stored on the server.

For each request it is needed:

- `access-token` - passed in the request header At the moment access tokens are not used, they are designed for future use. Each endpoint returns HTTP-code `200` if request has succeeded.

6.1 ViAn VianAPI endpoints overview

ViAn VianAPI consists of several endpoints:

- **metadata querying** endpoints:
  - `/dataset/*` - for querying information about datasets in the database
  - `/camera/*` - for querying information about cameras used for detections
  - `/sensing-module/`, `/sensing-module/all`, `/sensing-module/id` - for querying information about sensing modules used for detections
- **detection data and metadata** endpoints
  - `/detection/running`, `/detection/past` - for getting information about detections, which are running or which were finished in the past
  - `/detection/data` - returns the whole data from a detection - the data storage table contents
  - `/detection/records` - returns the output data from the detection, which satisfy a given condition
- **starting and stopping the detection** endpoints
  - `/sensing-module/init` - starts the detection
  - `/sensing-module/stop` - stop the detection
- **responses metadata** endpoints
  - `metadata/*` - returns the information about attributes returned by endpoints mentioned above

6.2 Metadata querying endpoints

These endpoints include querying metadata about datasets, cameras and sensing modules.

6.2.1 The Dataset endpoints: `/dataset/*`

Three possibilies to query metadata about datasets are available:

Getting information about a dataset by its identifier: `/dataset/`

For a given identifier (name) of a dataset, which is the only input parameter of this endpoint, all attribute values of a dataset are returned.

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/dataset?dataset=tes
↪  t" -H  "accept: */*" -H  "user-token: 123"
```

Response of a successful request: `200`

```
{
  "status": "success",
  "data": {
    "friendly_name": "test",
    "author": "vb",
    "description": "testovaci",
    "time_from": "2019-05-05T00:00:00+02:00",
    "time_to": "2019-05-06T00:00:00+02:00"
  }
}
```

Getting information about all datasets: `/dataset/all`

This endpoint does not have any input parameter and returns information about all datasets in the database. It is returned as a list of items with the same attributes as the previous endpoint.

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/dataset/all" -H  "a
↪  ccept: */*" -H  "user-token: 123"
```

Advanced querying of datasets: `/dataset/id`

This endpoint returns a list of dataset identifiers. The dataset, identifier of which is in the response, must satisfy a condition given in the request body. Here, it is possible to specify arbitrary of the following attributes and their values:

- `sensing-module` - sensing module, data of which are stored in the dataset
- `sensing-node` - sensing node where a sensing module is placed
- `sensetype` - required type of sensing module
- `camera` - camera, data of which are stored in the dataset
- `time` - time, for which the dataset contains data

cURL example request for condition: sensing-module = 'modul'

```
curl -X POST "http://localhost/vian_vianapi/dataset/id" -H  "a
↪  ccept: */*" -H  "user-token: 123" -H  "Content-Type: appli
↪  cation/json" -d "{\"sensing-module\":\"modul\"}"
```

Response of a succesful request: `200`

```
{
  "status": "success",
  "data": [
    "test"
  ]
}
```

6.2.2 The Camera endpoints: `/camera/*`

Similarly to the Dataset endpoints, three possibilies to query metadata about cameras are available:

Getting information about a camera by its identifier: `/camera/`

For a given identifier (UUID) of a camera (the only input parameter of this endpoint), all attribute values of a camera are returned.

cURL example request

```
curl -X GET "abcdefghlocalhost/vian_vianapi/camera?camera=cd0a
↪  bc34-9bdd-42fe-8fef-0d5f8d814d77" -H  "accept: */*" -H  "u
↪  ser-token: 123"
```

Response of a successful request: `200`

```
{
  "status": "success",
  "data": {
    "friendly_name": "kamera",
    "url": "1.11.1.1:2222"
  }
}
```

Getting information about all cameras: `/camera/all`

This endpoint does not have any input parameter and returns information about all cameras in the database. It is returned as a list of items with the same attributes as the previous endpoint.

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/camera/all" -H  "ac
↪  cept: */*" -H  "user-token: 13"
```

Advanced querying of cameras: `/camera/id`

This endpoint returns a list of camera identifiers. The camera, identifier of which is in the response, must satisfy a condition given in the request body. Here, it is possible to specify arbitrary of the following attributes and their values:

- `sensing-module` - sensing module, which is connected with a camera
- `sensing-node` - ssensing node where a sensing module connected with a camera is placed
- `sense-type` - required type of sensing module
- `dataset` - identifier of a dataset where data from a camera is stored
- `time` - time, in which the data from the camera was stored

cURL example request for condition: dataset = 'test'

```
curl -X POST "http://localhost/vian_vianapi/camera/id" -H  "ac
↪  cept: */*" -H  "user-token: 123" -H  "Content-Type: applic
↪  ation/json" -d "{\"dataset\":\"test\"}"
```

Response of a succesful request: `200`

```
{
  "status": "success",
  "data": [
    "cd0abc34-9bdd-42fe-8fef-0d5f8d814d77",
    "78c9d135-2803-4322-9f3e-f4a6330fd741"
  ]
}
```

6.2.3 The Sensing Module metadata endpoints: `/sensing-module/*`

Similarly to previous endpoints, three possibilies to query metadata about sensing modules are available:

Getting information about a sensing module by its identifier: `/sensing-module/`

For a given identifier (Docker image) of a sensing module, the only input parameter of this endpoint, all attribute values of a sensing module and the list of sensing nodes, where this sensing node is placed, are returned.

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/sensing-module?sens
↪  emodule=modul" -H  "accept: */*" -H  "user-token: 123"
```

Response of a successful request: `200`


```
{
  "status": "success",
  "data": {
    "dockerimg": "modul",
    "sensetype": "face",
    "friendly_name": "face detection",
    "description": "face detection",
    "sensenodes": [
      "https://localhost/sensingnode2_api",
      "https://localhost/sensingnode_api"
    ]
  }
}
```


Getting information about all sensing modules: `/sensing-module/all`

This endpoint does not have any input parameter and returns information about all sensing modules in the database together with sensing nodes, where the sensing modules are placed. It is returned as a list of items with the same attributes as the previous endpoint.

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/sensing-module/all"
↪  -H  "accept: */*" -H  "user-token: 123"
```

Advanced querying of sensing modules: `/sensing-module/id`

This endpoint returns a list of sensing module identifiers. The sensing modules, identifiers of which is are the response, must satisfy a condition given in the request body. Here, it is possible to specify arbitrary of the following attributes and their values:

- `camera` - camera, data of which are processed by a sensing module

- `sensing-node` - sensing node where a sensing module is placed
- `sense-type` - required type of sensing module
- `dataset` - identifier of a dataset where data from the sensing module is stored
- `time` - time, in which sensing module is active

cURL example request for condition: dataset = 'test'

```
curl -X POST "http://localhost/vian_vianapi/sensing-module/id"
↪  -H  "accept: */*" -H  "user-token: 123" -H  "Content-Type:
↪  application/json" -d "{\"dataset\":\"test\"}"
```

Response of a succesful request: `200`

```
{
  "status": "success",
  "data": [
    "demo",
    "modul"
  ]
}
```

Getting the list of all sensing nodes: `/sensing-module/all-sensenodes`
This endpoint does not have any input parameter and returns information about all sensing nodes in the database. It is returned as a list of items with the following attributes:

- `uri`
- `friendly-name`
- `desciption`

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/sensing-module/all-
↪  sensenodes" -H  "accept: */*" -H  "user-token: 123"
```

6.3 Starting and stopping detections endpoints

6.3.1 Starting a new detection: `/sensing-module/init`

This endpoint executes a database function, which stores information about a new detection into database and if this process is successful, returns access token and its validity. To start a detection, these parameters must be given in the request body:

- `dataset` - identifier of a dataset where data from a detection should be stored
- `sensing-node` - sensing node where a sensing module is placed
- `sensing-module` - sensing module used for the detection
- `camera` - camera used for the detection
- `friendly-name` - friendly name of a detection, which is initialized
- `fetch-stream` - boolean value, which indicates if the video from this detection has to be stored

cURL example request

```
curl -X POST "http://localhost/vian_vianapi/sensing-module/ini
↪  t" -H  "accept: */*" -H  "user-token: 123" -H  "Content-Ty
↪  pe: application/json" -d "{\"dataset\":\"test\",\"sensing-
↪  node\":\"https://localhost/sensingnode_api\",\"sensing-mod
↪  ule\":\"modul\",\"camera\":\"1.11.1.1:2222\",\"friendly-na
↪  me\":\"new detection\",\"fetch-stream\":true}"
```

Response of a succesful request: `200`

```
{
  "status": "success",
  "data": {
    "access_token": "eb1a38382a15bf58bf4c82b86fa80a50",
    "token_validity": "2021-01-13 00:00:00+01"
  }
}
```

6.3.2 Stopping a running detection: `/sensing-module/stop`

Stopping the detection requires sensing node, sensing module and camera. Meaning of these parameters are the same as in the previous section. If all parameters are valid, a database function, which stores information about detection stop time, is executed. No values are returned.

cURL example request

```
curl -X POST "http://localhost/vian_vianapi/sensing-module/sto
↪  p" -H  "accept: */*" -H  "user-token: 123" -H  "Content-Ty
↪  pe: application/json" -d "{\"sensing-node\":\"https://loca
↪  lhost/sensingnode_api\",\"sensing-module\":\"modul\",\"cam
↪  era\":\"1.11.1.1:2222\"}"
```

6.4 Detection data and metadata endpoints

These endpoints provide the possibility to get information about running detections and detections finished in the past and to read the results of detections from their data storage tables.

6.4.1 Getting metadata about running detections: `/detections/running`

This endpoint returns all attributes of all running detections (i.e. detection, which has an empty value of the attribute `time_to`) for a dataset given as a parameter of the request. The response contains a set of items, each representing one detection. For each detection, this set of attributes and its values is returned:

- `config_id`: identifier of the detection
- `friendly_name`: a friendly name of the detection
- `sensenode`: identifier of a sensing node where the detection is running
- `sensemodule`: a sensing module, which makes the detection
- `camera`: identifier of a camera, for which the detection is made
- `fetch_stream`: boolean value, which indicates if the video from this detection is stored
- `time_from`: start time of the detection
- `data_storage`: a table, which contains the results of the detection
- `access_token`
- `token_valid_to`
- `last_activity`

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/detection/running?d
↪  ataset=test" -H  "accept: */*" -H  "user-token: 123"
```

6.4.2 Getting metadata about past detections: `/detections/past`

This endpoint returns all attributes of all detections finished in the past for a dataset given as a parameter of the request. The response contains the same attributes as the previous endpoint, moreover, the attribute `time_to`, which indicates the finish time of a detection is included.

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/detection/past?data
↪  set=test" -H  "accept: */*" -H  "user-token: 123"
```

6.4.3 Reading the whole output data of a detection: `/detection/data`

For a given dataset and detection, this endpoint returns the whole content of a data storage table, where the results of detection are stored. The attributes returned in the response depend on the storage table schema.

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/detection/data?data
↪  set=test&detection=a941c5fd-7d55-4721-a8d8-47499adc46fc"
↪  -H  "accept: */*" -H  "user-token: 123"
```

6.4.4 Querying the output data of a detection: `/detection/records`

For a given dataset, detection and a condition, this endpoint returns the records from a data storage table, which satisfy the given condition. The condition `attname = value` is expressed by two parameters of the request: `attname` and `value`. The attributes returned in the response depend on the storage table schema.

cURL example request

```
curl -X GET "http://localhost/vian_vianapi/detection/records?d
↪  ataset=test&detection=a941c5fd-7d55-4721-a8d8-47499adc46fc
↪  &attname=foo&value=1237" -H  "accept: */*" -H  "user-token
↪  : 123"
```

6.5 Responses metadata endpoints

This endpoint returns information about all attributes and their types of each entity returned by other endpoints, for example cameras, datasets, sensing modules etc. These endpoints have no parameters. Here is the list of endpoints, which can be used to get metadata:

- `/metadata/dataset-atts` - returns set of dataset attributes returned by the `/dataset/*` endpoints
- `/metadata/sensemodule-atts` - returns set of sensing module attributes returned by the `/sensing-module/*` endpoints
- `/metadata/sensenode-atts` - returns set of sensing node attributes returned by the `/sensing-module/all-sensenodes` endpoint
- `/metadata/camera-atts` - returns set of camera attributes returned by the `/camera/*` endpoints
- `/metadata/detection-atts` - returns set of detection attributes returned by the `/detection/current-detections` endpoint

Moreover, there is the `/metadata/table-atts` endpoint, which allows getting the list of attributes and their data types. For this endpoint, two parameters are necessary:

- `dataset`: name of the dataset, which the table is part of
- `tablename`: name of the table

cURL example request for the `metadata/camera-atts` endpoint

```
curl -X GET "http://localhost/vian_vianapi/metadata/camera-att
↪  s" -H  "accept: */*" -H  "user-token: 123"
```

Result of this endpoint: Response of a succesful request: `200`


```
{
  "status": "success",
  "data": [
    {
      "attname": "friendly_name",
      "typname": "varchar"
    },
    {
      "attname": "url",
      "typname": "varchar"
    },
    {
      "attname": "id",
      "typname": "uuid"
    }
  ]
}
```

7 ViAn GUI

This section describes a demonstration operational application with a graphical user interface — ViAn GUI. ViAn GUI is developed as a multi-platform application and is based on technologies python, fbs, and PyQt that ensures compatibility between various operation systems. The application interacts with the rest of the system primarily over ViAn API and SensingNode API. Note that the application is currently in early development. In its current state, it enables to manage the following parts of the system (parts written in italics are under construction):

- cameras — view, *add*, *edit*, *remove*
- detectors — view
- datasets — view, *add*, *remove*
- detections — view, *start new detection*, *stop running detection*
- records — *view*, *remove*

The application will be also further extended with the ability to browse video data and to perform search queries over detected events.

A screenshot of the early version of ViAn GUI is in Figure 4.

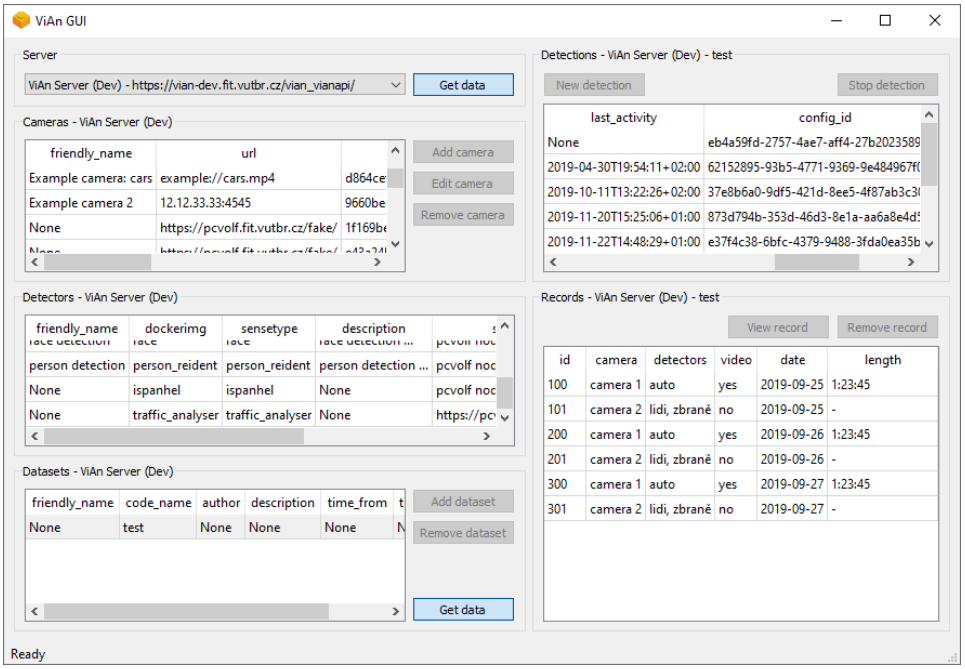

Figure 4: ViAn GUI (early development)