FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Web interface for real-time video analytics system
Student:	Bc. Vladislav Khachaturian
Supervisor:	prof. RNDr. Tomáš Skopal, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2020/21

Instructions

The goal of the diploma thesis is a design and implementation of a web portal for visualization and management of the Videolytics system. The portal should support the following functions:

- Playback of a live stream, the source of which could be either an IP camera or a local file.
- Navigation in the video stream - stop and resume, frame skipping (forward, backward).
- Playback of several video streams simultaneously including identification of the same objects and their trajectories.
- Visualization of detected objects and their trajectories in the video playback.
- Management of detection processes.
- Support for editing of interaction objects for further analytics, like a number of line crossings, realtime spent within an area, etc.
- Statistics generation and reporting

A part of the thesis will be an experimental evaluation of the web portal functionality based on real use- cases.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 6, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Web interface for real-time video analytics system

Bc. Khachaturian Vladislav

Department of Software Engineering
Supervisor: prof. RNDr. Tomáš Skopal, Ph.D.

May 27, 2020

Acknowledgements

I would like to thank prof. RNDr. Tomáš Skopal, Ph.D., my tutor, for all his advice, patience, and time he spent helping me finish this thesis. I would also like to thank Antošík Vojtěch, Dobranský Marek Mgr., Mařík Matouš, Smrž Dominik Bc., Ďurišková Dominika – students of Charles University in Prague, for their contribution into our mutual project Videolytics.

Declaration

In Prague on May 27, 2020 . . . .

Citation of this thesis

Khachaturian, Vladislav. Web interface for real-time video analytics system
Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstract

Tato diplomová práce je věnována návrhu a implementaci webového rozhraní pro systém video analytiky v reálném čase Videolytics. Hlavní cíl daného systému je extrakce, uložení a zpracování vysokoúrovňových rysů z video streamu. Výsledná aplikace je schopna spravovat různé zdroje dat a moduly: video stream, detekce a trajektorie objektů z databáze, procesy pro generování dat.

Vyvinutá aplikace se dotýká různých aspektů: řízení video streamů, efektivní dotazování na databázi, přesná (v rozsahu milisekund) synchronizace zdrojů dat, řízení procesů serveru.

Udržování velkého počtu různých aspektů uvnitř jedné aplikace je velmi složité. Z tohoto důvodu je webový portál Videolytics založen na architektuře mikroservisů. Webový klient komunikuje se službami pomocí protokolu HTTP, zatímco každá služba dosahuje svého určitého odlišného cíle.

Webový portál spolupracuje s dalšími moduly implementovanými v různých programovacích jazycích. Všechny fungují na jednom Unix serveru, aby bylo možné předejít zbytečnému zatížení sítě. Sdílení zdrojů a závislosti balíčků jsou vyřešeny pomocí technologie virtualizace docker.

Webový portál je implementován pomocí webového serveru Apache v kombinaci s jazykem PHP.

Klíčová slova webová applikace, streamování videa, video analytika, vizuální analytika, microslužby, docker, PHP

Abstract

This master's thesis is dedicated to the design and implementation of web interface for real-time video analytics system named Videolytics. Main target of this system is extraction, storage and processing of high-level features from video surveillance data. The resulting application is able to combine various data sources and modules: video stream, object detections and trajectories from database, management of processes for data generation.

Developed application touches on various different aspects: video stream management, effective database querying, precise (milliseconds range) synchronization of data sources, server process management.

Maintaining large number of different aspects inside one application is highly complex. This is why Videolytics web portal is based on microservices architecture. Web client communicates with services using HTTP protocol, while each service accomplishes it's certain distinct goal.

Web portal collaborates with other modules implemented in different programming languages. All of them are functioning on single Unix server to avoid redundant network load. Resource sharing and dependencies are resolved using docker virtualization technology.

Web portal is implemented using Apache web server in combination with PHP language.

Keywords web application, video streaming, video analytics, visual analytics, microservices, docker, PHP

Contents

List of figures

List of tables

1 Introduction

Video surveillance: potential and challenges

Nowadays amounts of produced data are terrifying and continue to grow exponentially [1]. There are different kinds of data: it can be valuable or useless, public or private, take up a little storage space or lots of it, easy to process or extremely hard.

Video surveillance takes a special place in this classification. It is potentially valuable as it contains data about people and transport movement. Collecting such statistics may be useful in many ways: for the municipalities to optimize traffic and public transport or for local businesses to predict suitability of certain location.

Before data is processed it needs to be collected. In big cities there are video cameras almost on every corner [2], but not every camera is publicly available. Many are owned by government or private enterprises, so it can be problematic to get access to them.

Manual video processing is possible but requires a lot of resources. Considering simplest use-case – determine amount of people in sight of a certain camera in certain time interval. There must be large enough storage, because even compressed videos take up a lot of disk space. Storing H264 encoded video in 720p quality from ten cameras for a month will require about 7712 terabytes of space. Even then manual processing takes some time and gathered statistics can become irrelevant. One more problem is to ensure security of all that data.

Automated processing requires a complex system, which must be able to do multiple things:

- detect different kinds of objects;
- in order to count them somehow distinguish same objects for whole interval of their presence in sight;
- build objects' trajectories;
- convert data into structured format and store in database;
- provide interface for data visualization and querying.

Improvements

Described system is already complex enough. But one more complication will bring a great improvement. And that is – system should process all data in real-time. Advantages of such processing are:

- no need in huge storage for video files, because structured data can be stored in database and take up a relatively little space;

- real-time statistics are the most valuable ones;
- if no video is stored and gathered statistics are impersonalized, there is no possibility to violate GDPR **[3].**

Such system also should be modular, because then it will become:

- easier to scale;
- easier to develop – subdivision into modules allows to utilize best fitting programming languages and technologies for each concrete task, while developers may work in their field of specialization independently.

Videolytics

What is described above is currently evolving video analytics system Videolytics. It is developed by group of students from Charles University and CTU under the guidance of prof. RNDr. Tomáš Skopal, Ph.D. Every student is engaged in development of his own part of system.

The system consists of independent modules, which are communicating through the central database. Current modules in development are:

- Detection module,
- Trajectory module – TrajAn,
- Real-time processing module – LiveED,
- Web client module,
- Reidentification module – ReID,
- Querying module.

Web client module

Subject of this work is development of Videolytics web client module. Its responsibilities are:

- Video stream management – connecting to IP camera or creation of stream from local file;
- Visualization of features from the database – real-time (with a small delay) drawing of detections and trajectories on top of video stream, synchronization of data, exclusion of redundancy while querying database;
- Management of server processes and other modules – their startup/shutdown, data flows.

Originally when assignment for this work was created, querying and report generation were also considered part of this module. However during the development collective discussion revealed the fact that it does not depend on other parts of this module and can be a separate application. Thus it was separated as distinct module with independent assignment for other students.

Purpose of work

Purpose of this work is development of web client module for Videolytics system, achieving the goals defined in task assignment and mentioned above module responsibilities, while also fulfilling additional requirements that will come up as a result of related works research and analysis.

Structure of work

- Related works part is dedicated to systems having more or less in common with Videolytics web portal, their strong and weak parts, possible use-cases and applications.
- Analysis part is dedicated to different possible software solutions, protocols, technologies and techniques.
- Design part contains specification of solution components needed to accomplish tasks defined in assignment.
- Implementation is dedicated to programming techniques used to achieve goals defined by design.
- Testing part is dedicated to testing methodology and collecting the data about module performance.

1 Related works

1.1 Video streaming

Video streaming is an actively developing way of delivering content on web [4]. Increase in network bandwidth opened opportunities to transfer video streams in high-definition quality online [5]. Ability to receive video content without storing it on local disk significantly raised effectiveness of information processing.

Some of the most common use-cases of video streaming are reviewed in this section.

1.1.1 Entertainment

This can be observed on the example of fast growing streaming services for entertainment [4]. Main reason of their popularity is that there is no more need of pauses between content deliveries. In the past person needed to go to the cinema or buy a DVD to watch a film, but now streaming services users are able to jump from one video to another without any delay, making their time-spending more enjoyable. And possibility to become a part of live video translation along with other people (on such platforms as Twitch.tv) surely can satisfy one of our social needs.

Stepping aside from live video streams, there is one factor that can be observed in entertainment systems and may be useful while designing Videolytics web portal. There is argument from people that are not willing to use streaming services – less control is provided on the viewed content. Different factors can make service usage less pleasant: unstable internet connection, interrupting and annoying ads, unavailability of content from another country etc. One of the basic human needs is a need for control. Things that are meant to be controlled (like casual TV show watch session), but are uncontrollable, may be repulsive for the end users. That’s why as much control as possible should be provided for the best experience, even despite the unsurpassable limitations of unstoppable video stream.

1.1.2 Criminals tracking

One more benefit of video streaming is opportunity of real-time data processing. Storing video to process it later requires huge amounts of free disk space. But advantages of real-time processing are not limited with decreased system’s costs. When gathering video statistics the delay in processing can be critical, for example for live criminal tracking systems using city cameras in China [6] or Russia [7] (Europe and USA are only starting to introduce such systems, they are doing it more carefully because citizens' rights to privacy are more valued [8]). However those systems are mainly used by governments and ordinary people usually cannot learn about their inner contents. Those systems are based on face recognition and require high video resolution.

Some people may be concerned about the fact, that such systems exist and can be potentially used by government to establish total control of people [9]. However no state owns sufficient processing power to recognize all faces from all cameras [10]. Though there is enough of it to track lawbreakers, even small ones like irresponsible drivers and pedestrians.

People’s security is also a thing that should be considered during development. Design of Videolytics system guarantees that even security breach will not lead to a leak of personal information, just because of the fact that personal information is not stored at all.

1.1.3 Security

Modern neural networks are able to process video streams in real-time, extracting different features. Some kinds of features can be used to produce alarms to notify camera owner about specific events. These kind of video processing is known as object/motion/flame and smoke detection and shape recognition.

Designed web application is not meant for security purposes. But generating alarms is a feature that can potentially be implemented in the future within web module or maybe a completely new one.

1.1.4 Self-driving vehicles

This topic is heatedly debated in mass media and doesn’t need a review. It is worth to mention, that in the process of this technology introduction into our everyday lives video stream processing will be used. However it can be replaced by other technologies, for example with extremely accurate geolocation introduced with 5G, which is both precise and has low latency (8–12 milliseconds). Invention of communication protocol for device position sharing will make self-driving vehicles even better, than video recognition can possibly afford, and with every human being accompanied with 5G device also much safer (for example in case when a car will be able to prevent accident even in complete darkness).

1.2 Video analytics systems: review

Computers are better at processing certain kinds of information than humans. But only humans can interpret results of data processing to accomplish particular goals, without this interpretation and further analysis all of the processed information is useless. The best results are achieved during cooperation between a person and a machine. Taking this into account it is especially important to design computer analytics systems in such way, that results are presented in clear and understandable form.

Importance of this aspect can be observed on modern mistrust of self-driving cars [11]. Statistically they can perform better than humans, but that does not matter, because it is not really clear how a car's "brain" is making decisions. It is a black box which cannot be opened even by developers of those cars, because neural network cannot be understanded after being trained.

This fact is taken into consideration during development of Videolytics system and its web portal. Containing different modules and building features using another features is complex, but this complexity can be managed by visualizing work of each particular module. Thus it is clearly seen how low-level features become high-leveled so the whole system's results become more trustworthy.

Use-cases presented in previous subsection (except for entertainment) are all examples of video analytics. Review of existing systems oriented on those use cases will be a reference point while designing own application.

The main difficulty of doing a review of them is low accessibility: usually such systems are provided as a complex solution for businesses or governments and there is not much information about them in open sources. This is the reason why only a few of them were considered as an examples, furthermore not in many details.

1.2.1 Self-driving vehicles

The most similar to Videolytics detection system are the one from self-driving cars, so they are worth to mention, even though their purpose is different. They detect and classify objects and build their trajectories. But first one collects statistics, while second one tries to predict the future. As such cars don't provide tools for video analytics, in context of this work their UI is not going to be useful for comparison.

1.2.2 Security

There are systems able to collect object trajectories, but mainly for security reasons. One example of such system's developer is PureTech Systems [12]. Their web page provides demonstrations of an application "PureActiv™ Video Analytics" [13]. Those demonstrations are a good source of inspiration to design video analytics system interface.

First example of usage is shadow removal is shown on figure 1.1. The important part is how object's position and trajectory is viewed – rectangle around the object and a line following it. This method of representation is a good example of how detections and trajectories should be shown, because it is intuitive and clear from the first glance. That's why it will be used as an reference point while designing Videolytics interface, until better method of visualization will be found.

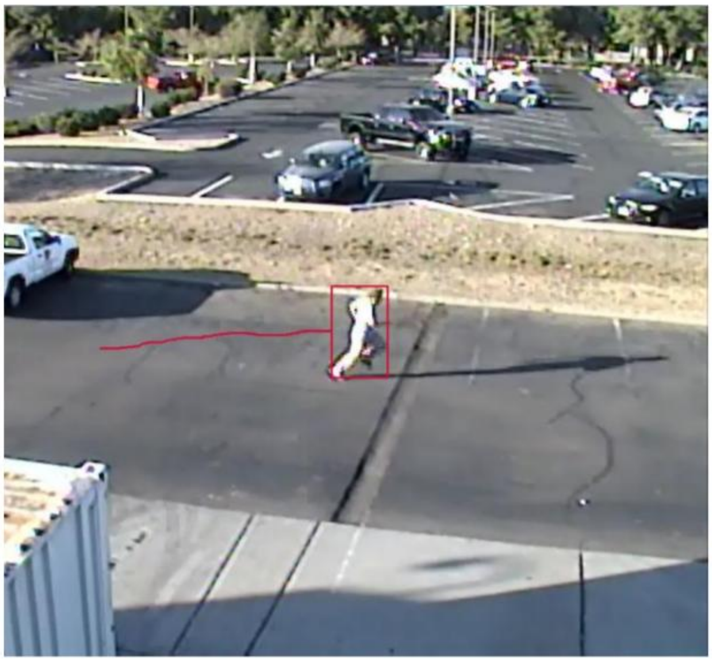
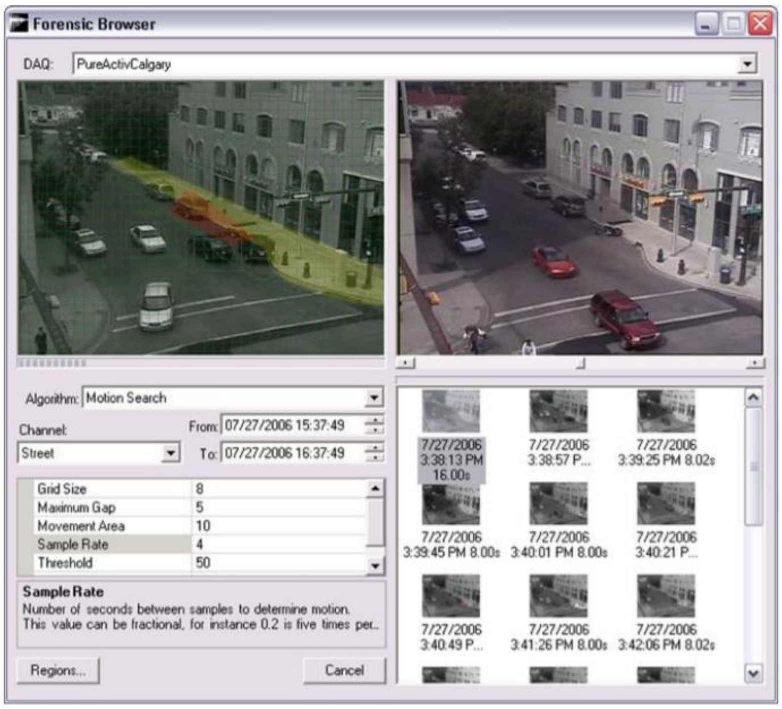


Figure 1.1: PureActiv – Shadow removal demonstration

Second example of “PureActiv™ Video Analytics” usage is GUI for application of specific algorithm shown on figure 1.2.



DAQ: PureActivCalgary

Algorithm: Motion Search

Channel: Street

From: 07/27/2006 15:37:49

To: 07/27/2006 16:37:49

Grid Size	8
Maximum Gap	5
Movement Area	10
Sample Rate	4
Threshold	50

Sample Rate
Number of seconds between samples to determine motion. This value can be fractional, for instance 0.2 is five times per...

7/27/2006 3:38:13 PM 16.00s
7/27/2006 3:38:57 P...
7/27/2006 3:39:25 PM 8.02s
7/27/2006 3:39:45 PM 8.00s
7/27/2006 3:40:01 PM 8.00s
7/27/2006 3:40:21 P...
7/27/2006 3:40:49 P...
7/27/2006 3:41:26 PM 8.00s
7/27/2006 3:42:06 PM 8.02s

Figure 1.2: PureActiv – Forensic Browser

This graphical user interface does contain a set of control elements that can be used in Videolytics web application: video player, source selector, video

time interval selector, and attributes’ settings. However, this particular design does not meet modern standards (probably because this application was presented in 2012).
Downsides of presented interface are:

- Video player shows the stream while applying specific algorithm and results are saved as screenshots from video, while it would be much better to visualize results on top of video.
- Video time interval selector’s behavior is not intuitive, because it is not clear how much time shift will occur on button press a how big is the interval of possible values. This control will also be used in developed application, but in another form.
- Attributes’ settings don’t represent as much information as they could. Using sliders instead of numeric values will significantly improve usability by defining strict value interval. And again there should be some kind of visualization of how parameter change will affect processing results. For the best experience fully dynamic visualization on top of live video must be used, this will be one of the main goals to achieve.

1.2.3 Crowd analytics

Somewhat similar are queue control systems like AllGoVision [14], that implements one of the developed use cases of Videolytics system (specific object class counting in a defined region). Unfortunately, screenshots provided by official web present only highly complex interfaces with 10 to 30 controls to setup the analytics process. No visualization of processing or its results is shown.

Another analytics system providing statistics similar to what Videolytics does is CrowdANALYTIX Inc [15]. However those systems are based on statistical models and don’t use live surveillance data in any way. Overview of their software is provided only to companies.

1.3 Motivation

- Highly complex video analytics system must provide highly controllable and customizable interface for visualization of results and guarantee security of processed data.
- There are no systems that are completely the same as Videolytics. So no application can be used as a reference point while designing web portal, it is rather combination of services described above.
- Use-cases from similar systems (like the ones for queue control) can be implemented in the future.
- Video analytics systems are highly complex and are used by big companies or governments. There is little to no information about such systems publicly available. So research and comparison in relation to them is very limited.

2 Analysis

2.1 Software requirements

Conducted related works research has revealed further software requirements for the developed web application:

- **Compatibility**: Videolytics system consists of separate modules which provide better scalability. As the entire video stream is processed in real-time, only structured data is stored on disk. With usage of distributed database it is possible to deploy several servers each running certain module/modules communicating with each other through the shared remote database. When using many servers it may be possible that different machines will be running different operating systems. While specific tools can be executed inside virtual environment (docker container), key software like web server or database will be installed in a standard way to simplify their administration. Thus this software should be possible to install and maintain on any operating system.
- **Easy deployment**: As mentioned above, key software must be compatible with any operating system. It is also preferable that this software will be easy to deploy in case of cluster expansion or server migration. Because of the fact that each module of the system is developed separately using different programming languages, deploying them should be as simple as possible. For example necessity to compile software from sources during each deployment is unwanted.
- **Universality**: It isn't known in advance what kinds of data inputs and outputs must be supported: IP cameras use various communication protocols and video formats, while Videolytics modules would prefer to stick to one specific protocol and format as input. Thus some kind of universal interface for video stream must be provided.
- **Independence**: as many different modules should be able to be installed within a single operating system, they should be as virtualized as possible to avoid potential conflicts (for example dependency hell).
- **Control**: As it was discovered during the related works research, control over video stream should be provided, so used software must provide comprehensive possibilities as well as exhaustive documentation. Not only live video streaming must be supported, also offline stream simulation should be provided for debugging purposes. The reason why a static video source is needed is to observe module behavior and various processing modules under the same conditions. So it is convenient to be able to simulate video stream from static source but at the same time treat it as a live stream.

- **Loose coupling**: As developed application is going to manage other modules, it will be tightly coupled with them. This can lead to various problems during development, when a small change in one element will require a lot of changes in interfaces between modules. One way to resolve this communication problem is to couple the modules with smaller interfaces that can be considered as submodules or services. This brings to an idea of microservices architecture [23] or it can be said to divide and conquer strategy. Presented application design should use separate services to realize communication between modules, while the other tasks can also be achieved taking advantage of microservice architecture. Separating application functionality into independent services not only provides loose coupling, but also makes it easy to write, reuse, test and integrate the same code for different purposes. Division of an application into parts is also a good way to make it more understandable not only for programmers, but for the end users as well by presenting schemes and diagrams showing services interaction and data flows between them, and this is one of the core requirements of this work that was defined during the related works research. So chosen software should support implementation using microservices.

2.2 Web server

Selection of the web server will be made among three of the most popular nowadays candidates: NGINX [16], Apache [17] and Node.js [18]. They provide similar functionality. The choice will be based on minor advantages/disadvantages considering defined requirements and tasks. Desirable capabilities include simplification of video streaming, accessing server's filesystem and command line.

2.2.1 NGINX

When searching for video streaming opportunities among web servers highly popular solution is based on NGINX web server. That's because there are lots of different modules extending server functionality, one of which is `ngx_http_hls_module` [19]. It is widely used to implement video streaming web applications, providing HTTP Live Streaming (HLS) support for MP4 and MOV video files.

Features:

- Modules extending server functionality can be installed.
- Video stream input format: MP4 or MOV.
- Video stream output format: RTMP (Real-Time Messaging Protocol) stream or HLS stream.

Advantages:

- Low memory usage.
- Better performance than other candidates can provide – more requests per second, better thread handling. Though the developed web application is not required to utilize that, only several users must be served simultaneously, as each of them will significantly load the server.
- Various server modules (for example for PHP support) which are mostly free.
- While streaming a local video file, stream position can be fully controlled by user.
- Server's command line is accessible with help of one of the modules (Lua).

Disadvantages:

- The input for the module is limited with MP4 or MOV formats. In case when another video format is provided from camera or local storage, additional conversion with external tools is required, and that is unwanted load for the server.
- HLS stream output provides one of two formats: `.m3u8` or `.ts`. These formats can be played in web browsers, but special video player must be used, as HLS is not natively supported in some of the most popular web browsers like Chrome or Firefox [20]**.** As for Internet Explorer its version 11 or higher must be used as well as operating system Windows 8 or newer, the only other way is to use video player based on Adobe Flash technology [21], support of which is currently disabled in modern browsers by default for security reasons and its further development is stopped this year. Even if compatibility problems with web browsers were solved, there are further Videolytics modules that will use the stream. These modules are written in different programming languages, so their support of HLS is questionable.
- RTMP stream output uses RTMP protocol. Similar to previous HLS stream case it can be played only with Adobe Flash Player and possibility to process it with Videolytics modules is unknown.
- NGINX HLS module is not free, it comes within a NGINX Plus package. It provides some useful features like load balancing but at a price of $2500 a year per server instance. There is no possibility to pay just for one particular module or it is not listed on official web, whereas a minimum of two modules is needed (HLS and Lua).
- During the testing it turned out that server configuration is not that easy as it is described. Dynamic modules are installed with relative ease, but they were not used in testing configuration. Modules for RTMP streaming and PHP support are static and their installation required some advanced configuration. Even then functioning of these modules was unstable. Possible cause of this probably was the version of the used operating

system (CentOS 8), where irresolvable old package conflicts occurred. And as it was mentioned above, the system should be deployable on any operating system without such problems.
- Documentation is present, but for the RTMP module some of the described features were not functioning at all (for example `exec` directive). Possible reason of that is unknown, as server has not provided error logs for each configuration directive.

Conclusion:

NGINX was a first candidate to use as a web server for Videolytics. Described features and advantages looked promising. But as a consequence of many problems mentioned above, this variant was rejected.

2.2.2 Apache

Apache HTTP Server was launched in 1995 and since then stays one the most popular web servers [17]. The fact that it is old doesn't make it a bad option, because it was going through the path of development all these years. As it is open-source project, for many years of voluntary community work many packages were implemented, which can be installed using standard package manager on any Linux operating system. There even exists a H264 video streaming module [22], but it has two big disadvantages: it is not free for commercial use and it doesn't provide complete documentation (only a few tutorials). But this doesn't really matter as there are plenty of other streaming tools that can be used alongside Apache.

Advantages:

- Reliable, fast and flexible web server with a long history.
- Well-documentated.
- Easy to administrate and to get technical support as it is highly popular and thus frequently used.
- Deployment and configuration is really simple task – it is only needed to install server and needed modules, place web site at default folder and run the server.
- Simple module installation allows using PHP language to implement server-side operations.
- Server changes can be done instantly without restarting it.
- Apache can be run even on Windows server without problems.

Disadvantages:

- Despite easy deployment, advanced server configuration can be complex.
- There is no built-in support for video streaming, existing modules are hard to use. So this task must be achieved with some external tool. Fortunately with usage of PHP and docker almost any soft can be utilized.
- PHP language is often not considered as a good option for several reasons. However if utilized according to its original purpose (programming simple

server-side services) and not overused way too much (for example trying to implement a new framework), PHP stays a decent option.

Conclusion:

Apache is a good web server option in context of current task. Sufficient tools are provided to complete it, while server popularity brings many other advantages. Mentioned weaknesses are not significant and can be managed.

For the purposes of this work Apache HTTP Server will be used.

2.3 Server-side scripting

One of the tasks to accomplish is management of server-side processes, particularly various Videolytics modules. This includes their startup/shutoff and checking execution status. Using heavy server-side technologies (like Java) is sure a way to do that, but at the same time is redundant. Another option is manage those processes using simple command line commands that can be called with light-weighted scripting language.

When it comes to choosing such language, the two most popular variants are PHP and NodeJS. They provide similar functionality and both are capable of accomplishing assigned task, both not having significant drawbacks. However there are a few things worth to mention in context of Videolytics web portal module‘s responsibilities:

- Both provide sufficient tools to implement video streaming, but not with features that are needed. It is possible to publish a simple video stream from local file, but when it comes to live streaming, an external tool is needed. And rather than combining two different tools for the same purpose, it is more convenient to leave video streaming task for the specialized software and use scripting language only for management of that software.
- Both are able to communicate with database with installation of corresponding modules.
- Both are able to execute a command line on server, thus managing server processes.
- Advantages of PHP include:
    - simpler deployment on server, just because it is an Apache module that starts working ones installed and does not require anything else;
    - being faster [24], which is not so profitable considering that the biggest load on server will come from the other modules.
    - integration with HTML that allows to easily combine both when needed;
    - less code amount required – a simple `.php` file with a script in it is ready to be executed without server restart or any other code changes;

    - NodeJS is a newer technology and author of this work had more experience with PHP.
- Advantages of NodeJS include:
    - same server and client programming language syntax makes a code easier to write and understand;
    - support for asynchronous operations (which can reduce server load during the database querying).

As a conclusion, for the purposes of current task scripting language PHP is a slightly better option for the developed web application.

2.4 Database

Main requirements for database are:

- **no NoSQL** – there will be a lot of connected data and a lot of complicated requests for them, so standard relational database is the best choice;
- **speed** – database speed is one the most critical parts of the whole system, sufficient amount of requests per second must be achieved to accomplish live video processing;
- **compatibility and maintenance** – same as for all of the used software, it should be easy deployable and administratable on various operating systems, as well as accessible with PHP.

Selection is made from the most popular relational database management systems: Oracle [25], MySQL [26], SQL Server [27], PostgreSQL [28].

From these Oracle and SQL Server does not meet the compatibility requirement, as the first one has no support for many operating systems and the second one is run exclusively on Windows.

As for performance, it is hard to tell which one of MySQL and PostgreSQL is better. Different studies show different results in different conditions.

Final decision was made during collective discussion of Videolytics project members, and according to their personal preferences and experience the choice was made in favor of PostgreSQL. For the whole exploitation period its speed was enough to achieve desired goals, even before data scheme optimization.

2.5 Internet protocols

There are many other protocols and technologies that can be used and it is too difficult for now to decide which to stick to. These subsection is more dedicated to determination of what cannot be used. This will be helpful during the selection of video streaming software in the following subsection.

2.5.1 Transport layer

There are two main types of transport protocols – TCP and UDP. TCP protocol is more reliable:

- it requires connection between server and client verified with three-way handshake;
- error and packet loss handling mechanisms;
- integrity of data is guaranteed – client knows in which order received packets should be viewed

Mentioned above features of TCP are not provided by UDP, making it more lightweight and generally faster for sending a lot of requests.

While UDP is often used for video streaming due to its advantages, it cannot be used for Videolytics web application. The main reason if that not only the video stream must be shown to end user, but also high level features (detections and trajectories) associated with individual frames of video. Synchronization between these two data sources must be extremely precise, as even a small delay of several frames can be perceived by user and make usage of service less comfortable. Example of such delay is shown on figure 2.1.

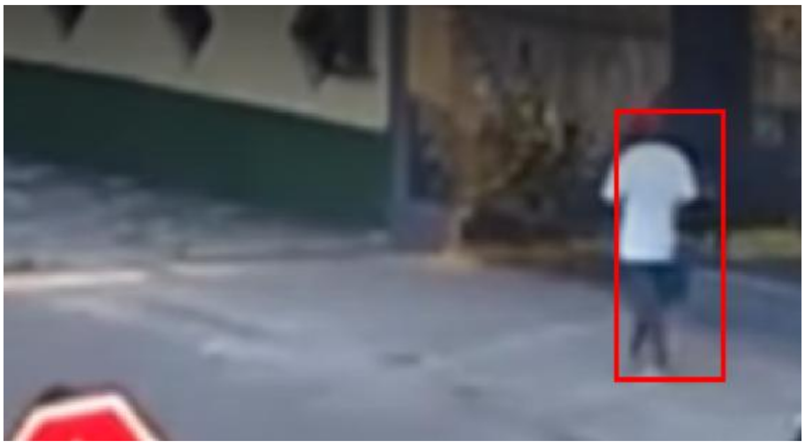


Figure 2.1: Example of 7 frames delay in object detection

Thus the video stream must be highly stable, which can be guaranteed only by TCP protocol. The disadvantage that it is slower is compensated with two factors:

- The data portion of packets is going to have much bigger size then the header because it is a video stream. And the speed of UDP protocol comes partly from smaller packet header size.
- UDP shines when serving large amounts of clients, which is not the case of Videolytics service. There are not going to be thousands of simultaneous users served by one single server, as each connection requires a lot of resources. Number of users able to connect to single server is limited by server's hardware, but in general won't be able to

exceed dozens of clients. To serve more people more servers will be used as the system is highly scalable.

Also UDP packets do not contain information about their order, which can lead to huge desynchronizations.

And there is one more fact that speaks in favor of TCP. High level features associated with video frames are stored in database and will be fetched using PHP, leading to utilization of HTTP, which is commonly used along with TCP. It can be used with UDP, but in this particular case it won’t be for the same already mentioned reasons. And sending those different data with the same protocol will lead to smaller latency dispersion, which will allow for smaller delay of live video.

As a conclusion, there are many factors indicating advantages of using TCP protocol for the developed application. Even if UDP may be a faster transport protocol, it also comes with inacceptable disadvantages. So the final choice is TCP.

2.5.2 Application layer

Traditional video streaming protocols are RTMP (Real-Time Messaging Protocol) and RTSP (Real-Time Streaming Protocol). Both of them are using TCP transport protocol (UDP is also an option). However they maintain client-side connection with Adobe Flash Player. Usage of this technology is undesirable for the same reasons as described in subsection 2.2.1 at the part describing HLS streaming: end of support, need for external player (which makes stream control more difficult), Videolytics modules communication problem.

It is worth to mention HTTP-based protocols, as they provide high compatibility with both web client and Videolytics modules due to the fact that HTTP is commonly used for communication and thus almost anything can utilize it.

As a result of protocols analyze several guidelines have arisen:

- Stick to TCP transport protocol instead of UDP;
- Usage of RTMP, RTSP and HLS technologies as well as usage of external video player is unwanted;
- HTTP communication is preferred.

2.6 Video streaming software

There are two types of video streaming software that is needed:

- First to carry out video conversion to the required format and/or codec.
- Second to establish a video stream source, that can be used by web client and the other Videolytics modules.

The most popular software options will be listed, discussed and compared . Then a final choice will be made.

2.6.1 First task – video conversion

FFmpeg [29] is a complete, cross-platform solution to record, convert and stream audio and video. It provides tools to carry out all of the imaginable video conversion supporting all of the discussed formats, codecs and protocols. Conversion process is highly customizable and all available options are documented in detail with examples. This tools is extremely popular and widely used in many applications including open-source projects, which provides access to many more possible usage examples and scenarios for inspiration. License of FFmpeg allows free commercial use, however distinct codec libraries such as `libx264` (for h264 support) have independent licenses which limit commercial use based on number of service users, which will remain very small in case of Videolytics. One more advantage is that FFmpeg also provides its own streaming tool FFserver.

HandBrake [30] is a free and open-source transcoder for digital video files. It is cross-platform tool with huge variety of supported video formats. While it is definitely possible to output a video stream with it, acceptance of such stream as an input from IP camera is limited to specific formats. Commercial use is fully free. It also provides needed command line tools that are however documented with fewer details than FFmpeg.

Format Factory [31] is a set of Free and multifunctional, multimedia file processing tools. While supporting a lot of video formats, its use via command line is limited and not properly documented, thus cannot be customized. Commercial usage is completely free, however the only available platform is Windows, which is a huge disadvantage.

VLC media player is able to convert between video formats, even video streams. However it is only able to save output to local disk, which is not the best option considering intense local storage usage. Command line interface exists and is documented but is not customizable enough. During the tests with video stream conversion this tool unpredictably changed playback speed, but still provided a surprising variety of functionality for a product positioning itself as a video player.

Comparison of video streaming tools is shown in table 2.1, where significant disadvantages are in bold.

Every tool has at least one significant disadvantage, most of which greatly affects functionality. While certain workarounds can be utilized to compensate some disadvantages, it is not the best option. From the set of reviewed tools only one satisfies all defined software requirements and it is FFmpeg. Its disadvantage is that it is not completely free, because some of its libraries, including necessary libx264 for H264 support, have distinct licenses. This required library however is free for commercial use under certain conditions, and more specifically number of users per month, which is going to remain very low for Videolytics system due to its nature.

Table 2.1: Comparison of video conversion tools

	FFmpeg	HandBrake	Format Factory	VLC
cross-platform	yes	yes	**Windows only**	yes
H264	yes	yes	yes	yes
command line interface	yes	yes	**limited**	yes
customization	excellent	good	**unknown**	**poor**
documentation	excellent	good	**poor**	yes
commercial use	**limited**	allowed	allowed	allowed
input stream	yes	**limited**	**unknown**	yes
output stream	yes	yes	**unknown**	**no**

So considering all the above the best option among the reviewed candidates is FFmpeg. As a bonus it comes with its own video streaming software FFserver.

2.6.2 Second task – video streaming server

There are plenty of implementations of video streaming server. While making decision open source projects are preferred as they are generally better documented, customizable and clear, while also being free.

Paid servers may have better UI (some even provide web GUI), but they also have a many disadvantages:

- being more heavy, so it is much harder to deploy them including for initial testing purposes;
- payment system is one more responsibility to worry about when managing server, which can stop the entire system when forgotten and resumption of work may last at least for a duration of a bank transaction;
- it costs money, which may not be a lot but still is acts as a demotivation for project members;
- all the other software used for now is free, so Videolytics maybe can someday also become an open source project if all of its parts are also open source.

When choosing from free options, the possible choices are FFserver [32], SRS [33], PHP FFmpeg Video Streaming [34], mkvserver_mk2 [35] and many more. There is a huge list of such software containing 47 tools [36]. Only a few of them will be considered.

FFserver comes within FFmpeg package. Communication between them is implemented with `.ffm` (FFserver live feed) stream format. When the server is configured, no further customization is required. Once conversion is launched, stream source is generated according to the configuration file. These tools work perfectly in pair which is a huge advantage. As FFserver comes with FFmpeg, it brings all its benefits described in previous subsection: support of various formats, codecs and protocols. Documentation is not so good and comes as a set of configuration examples, a full list of supported directives is not provided, so configuration of FFserver can be done only with trial and error method. There is also a controversial disadvantage described on the main page of the server and that is the end of its support. This also means that new official versions of software will not appear. However as this project is also open source, distinct libraries for it can still be developed by enthusiasts and libraries for the most widely used standards (specifically H264) are already implemented long ago and won't ever need a change. It also may be problematic to install FFserver on the newest operating systems versions as the newest FFmpeg packages come without FFserver and the older ones are not present in repositories. As docker will be used to run this software, so this is not a problem at all.

SRS is similar to FFserver in all relations, the only differences are that it is an individual tool not connected with FFmpeg and that binaries are not provided. This makes usage of SRS a bit more complicated, as further FFmpeg configuration will be needed and building software from source code may be tricky to do inside a docker container. An attempt to build SRS on test server has failed and this tool has been left as a reserved option.

PHP FFmpeg Video Streaming is a wrapper, but for another tool PHP-FFmpeg, rather than FFmpeg. This tool makes it easier to work with FFmpeg in PHP, both of which are already a part of the developed application. The disadvantage is that the output stream is whether HSL stream or DASH (Dynamic Adaptive Streaming over HTTP), both of which will require a separate video player and bring compatibility issues during integration with Videolytics modules.

Mkvserver_mk2 (or Matroska Server Mk2) is a software to produce stream in matroska format (more known by its extension names `.mkv`, `.mk3d`, `.mka`, `.mks`). Though this software is recommended to use on FFserver main page, it has a lot of limitations. Firstly this output format is not natively supported by modern browsers, which brings the same problems as for PHP FFmpeg Video Streaming. Secondly this project is poorly documented, all its description is located at the main page, where there are a few lines about software, usage examples that consist of two command lines and architecture description. Thirdly, no binaries are provided and again same as for SRS build on test server has failed.

Summary of reviewed video streaming servers is shown in the following table 2.2. Significant disadvantages are written in bold.

Table 2.2: Comparison of video conversion tools

	FFserver	SRS	PHP FFmpeg Video Streaming	Mkvserver_mk2
binaries	available	**only source**	available	**only source**
output	anything	**RTMP, HLS or HTTP FLV**	**HLS or DASH**	**matroska**
documentation	average	good	good	**poor**
configuration	easy	easy	easy	**missing**
customization	rich	rich	rich	**missing**
friendly with	FFmpeg	-	FFmpeg and PHP	-

FFserver's end of support did not fit onto the table and moreover is not that significant. In fact when FFmpeg was chosen, it became first candidate for streaming purpose. Testing has proven its capabilities of accomplishing assigned task (server can be deployed from zero in minutes), while fulfilling all the requirements. Comparison with other software was made just in case something better or at least interesting will come up. Moreover it allows using HTTP protocol for stream which is a preferred option.

As a conclusion FFserver is the best fitting option for given task.

2.6.3 Summary

- Web server – Apache
- Server-side scripting – PHP
- Protocols:
  - Transport layer - TCP
  - Application layer – HTTP
- Video streaming software:
  - Video conversion – FFmpeg
  - Video streaming server – FFserver (part of FFmpeg)

3 Design

3.1 Virtualization

Docker virtualization technology allows creation of an independent environment inside of an operating system. This environment much like virtual machine is based on some operating system's image (docker also allows to build own modification of images). This helps to solve compatibility problems, when some software for whatever reasons should not or cannot be run on parent operating system. One more benefit is that multiple containers can easily be run or stopped using the same image. With regard to Videolytics system, there is hierarchy of responsibilities - to ensure nothing goes wrong on production server, it has only one administrator. To use resources of the machine other team members use docker container to install whatever software and run whatever processes they want without causing any conflicts such as dependency hell. At the same time containers can communicate with parent's filesystem or network or other containers without interfering with parent processes (except that they share computing resources).

From chosen software Apache web server with PHP and PostgreSQL database can be installed almost on any operating system and once deployed and run won't need almost any management: updates on server will be made by changing files containing web pages' code in local storage, updates of database will be made with administration tool PgAdmin or its analogue running on different machine. These two processes are critical for the whole system and putting them inside a docker container will bring one more layer of management and complexity, in case of accidental container shutdown or corruption there is need to run other process that will restart it and to make backups. One more reason not to use docker for the web server and database is that they are used by every developer, so there is no need to make a separate environment. Virtualization is a good technology, but it should not be overused, and this is the case.

As for other remaining software – FFmpeg and FFserver – there are several reasons to put them inside a container. Firstly FFserver cannot be easily installed on any operating system, to do so it sometimes must be built from source code. The reason for this is that newest version of operating systems don't always provide sufficient backward compatibility for software by not including older versions of that software into repositories. Secondly there can be several instances of FFmpeg conversions running simultaneously, each serving certain user's or module's needs.

For this two purposes separate docker images are defined based on latest release of Ubuntu operating system, as it provides access to FFserver package without need to build it from the source. Benefits of predefined docker images is that they are built with set of instructions (Dockerfile), which allows to manage software on every step: installation, configuration (in this case by copying configuration file from parent system) and even running it automatically when container starts. This allows going through all the difficult stages at the beginning and then run needed software by simply starting a container.

To avoid disambiguation in names of containers and FFserver stream feeds, several rules are defined:

- Name consists of its general name and a numeral suffix when needed;
- General name represents the nature of an object:
    - `feed` – live stream feed,
    - `ffmpeg, ffserver, liveed` – containers,
    - containers will also have a word instance on their general name to explicitly distinguish them from the corresponding docker images;
- Numeral suffix following general name is a unique user id.

3.2 Unique user id

As it was mentioned above, multiple users may request for different video streams simultaneously. When someone is watching such stream in a specific moment of time the other must not interfere with it in any way. So there is need of some kind of service providing information about server status. There is a possibility to run such service with FFserver, it is implemented as a kind of video stream with format `status`, being actually a HTML page. Unfortunately, contents of that page cannot be configured and provide almost the whole variety of information about the server, furthermore status of stream feeds is not provided as a clear true/false variable, but still can be derived from the few lines at the bottom of the page. Another downside of this output format is that it is not supposed to be read by a computer. Parsing HTML on client side will mean that the user will have access to server's details, which are not really needed for client application functionality. So the solution is to develop a server-side HTML parser, which will consume that status page and produce only the required set of values. FFserver status page can then be configured to be available only from the server itself, limiting access from the outside.

Output of this service will be in JSON format, as it is JS-native (easy to process in browser), but also human-readable (convenient for development and debugging). Output variables will have boolean type and show, whether FFserver or a specific feed is running or not.

The algorithm of getting the unique id is:

0) If user is not watching the stream, he does not need the unique id. Process of getting the id starts after the user initializes stream start;
1) Get the server status in JSON format from the defined service;
2) If FFserver is not running, send a request to start it and wait for it to start;

3) Choose any of the available stream feeds, get a unique id from the feed name numeral suffix (defined by naming convention);
4) Start the video stream, so that the id becomes reserved;
5) If the stream has started successfully, remember the id till the end of the stream. Otherwise return to step 2.

3.3 Database

The process of designing the database was cooperative between all the Videolytics project members. Adjustments and agreements in relation to every individual module ware made, so the whole design process will not be described here, as it involves too much context of the other modules. Instead only the final result will be shown to provide necessary context needed for the further design and implementation.

Database schema is shown on figure 3.1.

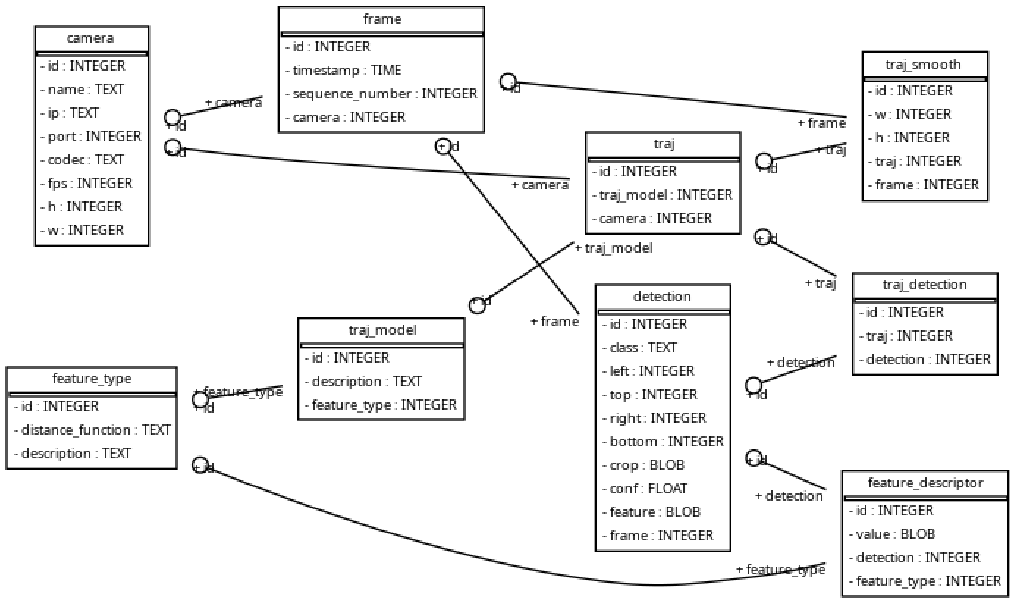


Figure 3.1: Database schema

Table descriptions:

- Camera - available cameras or videos in the local storage
  - id: PK;
  - name: For videos this is the name of the file;
  - ip: Only for live camera;
  - port: Only for live camera;
  - codec: Only for live camera;
  - fps;
  - h: height, w: width.

The last three fields were added to reduce web module complexity. First database versions didn't have them, to determine these parameters the module was using separate service based on `mediainfo` tool. Whereas that was enough for local videos, for the live stream sources it is more complicated. So the fields for fps, height and width were introduced, defining video parameters.

- Frame - a record for each frame of each camera/video:
  - id: PK;
  - camera: ID of the camera this frame corresponds to (FK to Camera);
  - timestamp: Time the frame was captured;
  - sequence_number: Sequence number of the frame within given camera. This is a replacement for timestamp field in cases where it is not good enough (during frame drops etc.).
- Detection - a record for each part of frame containing detected object:
  - id: PK;
  - frame: FK;
  - class: the class of the detected object (person, car etc.);
  - left, top, right, bottom: left/top/right/bottom-most coordinate of the bounding box of the detection in the frame;
  - feature: Legacy column; not used for now;
  - crop: Whole image (crop) of the detection in numpy-byte format;
  - conf: Confidence of the algorithm for detection (0 – 1).

As for coordinates, originally [x, y, w, h] were used, but such names (x and y) collided with trajectory centroids definition. To avoid that fields were renamed.

Some field names (for example class or left) are colliding with PostgreSQL in-built operator names, which must be considered while constructing SQL queries – to avoid disambiguation field names must be quoted.

- Traj_model - Models of trajectories:
  - id: PK;
  - description;
  - feature_type: Which feature type was used when creating this model (FK to feature_type).
- Traj - Auxiliary table to connect detections and trajectories. One row represents one trajectory:
  - id: PK;
  - traj_model: FK to traj_model;
  - camera: In which camera/video is this trajectory (FK to camera).
- Traj_detection - Actual assignment of detection into trajectories:
  - id: PK;
  - traj: ID of trajectory (FK to traj);
  - detection: Which detection we assign (FK to detection).

- Traj_smooth – the result of trajectory smoothing:
  - id: PK;
  - traj: FK to traj;
  - frame: FK to frame;
  - w: horizontal coordinate;
  - h: vertical coordinate;
  - The other tables are used by the other modules only are not touched on in this work in any way.

3.4 Modes

Developed application works with various data from different sources. Those data will be visualized and synchronized between each other. Correct client-side visualization is highly important not only for the end user, but also for the other developers, as they will tune behavior of their modules. In this context application must be designed in such way, that it can provide interface to process the same data as an input over and over again. Using this method web portal will also become a testing tool for the other modules including itself.

However live video stream is not repeating itself, so for testing purposes there must be available some kind of live stream simulation.

One way of doing so is to create constant repeating stream source of defined interval. This solution leads to an unsolvable paradox – proper testing cannot be done if the interval is too short, as there will not be enough video material to work with, but at the same time effective testing cannot be done, because the developer must wait for the defined interval each time he wants to test specific moment.

To overcome this an another approach is used, which is more difficult to implement, but at the same will greatly pay off later during development, testing and even normal usage. The approach is to introduce three different modes of application, each with its own purpose and appropriate functionality.

These modes are:

- Offline and real-time simulation – a repeating video stream is created from a file located in local storage with a defined by user playback starting position.
  - In case of offline mode detections and trajectories are preliminarily computed and stored in the database.
  - In case of real-time simulation on each start of the stream all the detections and trajectories are created in real-time and stored in the database. Then when the user leaves they are removed from the database, as this is a simulation only mode.
- Real-time – video stream source is an IP camera, detections and trajectories are generated in real-time the same way as for simulation

mode, with the only difference that the features are then not removed from the database, but instead stored for further analytics.

Though the features are extracted in real-time, there is a delay of several seconds before they are ready to be visualized.

A better representation of what these modes have in common is a following table 3.1.

Table 3.1: Comparison of web client modes

	offline	real-time simulation	real-time
stream source	local file	local file	IP camera
feature generation	preliminarily	real-time with a delay	real-time with a delay
features after the user leaves	left as they are	removed afterwards	saved

3.5 Use-cases

Use-cases that should be supported by this application come up from the task assignment combined with requirements developed during the related works research (summed up in subsection 1.3).

- Playback of a live stream:
    - Choose the stream source (local file or IP camera);
    - Choose the stream start time (when the source is local file);
    - Start the stream;
    - Stop the stream.
- Visualization of detected objects and their trajectories in the video playback:
    - Enabling and disabling detections in general and for individual detection classes;
    - Changing opacity and line width of detections layer;
    - Customizing information about detections: font size, enabling and disabling displaying of class name, confidence, id, coordinates and size;
    - Enabling and disabling trajectories;
    - Choosing trajectories model;
    - Changing trajectories width, fade time, curve radius, opacity;
    - Enabling and disabling smooth trajectory fading;
    - Changing framerate of detections or trajectories layer.

Management of detection processes is automated to reduce complexity for users, so not being a use-case.

3.6 User interface

User interface design is a result of the related applications' research and collective discussions, and the result is a decision to follow material design [37] conventions, the main of which is responsiveness in all senses. By that it is meant that not only the design will adapt to screen resolution, but all the visualization customization changes will take effect as fast as possible (preferably immediately).

The color theme choice is mainly affected by the fact, that video is almost always present in the user's view. One characteristic of a video is that when it does not perfectly fit inside its container (which considering responsiveness will sometimes be the case), black strips appear at the top/bottom or left/right borders. Theoretically they can be recoloured, but it may be hard when they are a part of video it is better to stick to standard black. Thus black will be one of the main colors used in an application, background and header should be distinct from black to emphasize the video, which already makes a total of three. Though almost any good combination can be used, the main part of the UI is still a video, so any chosen dominating color may interfere with the rest of UI. The decision is to choose the most neutral color palette – shades of gray. As pure white makes too much contrast with pure black, it is preferred not to use it. Instead final choice of three main colors is a combination of black, dark and light gray.

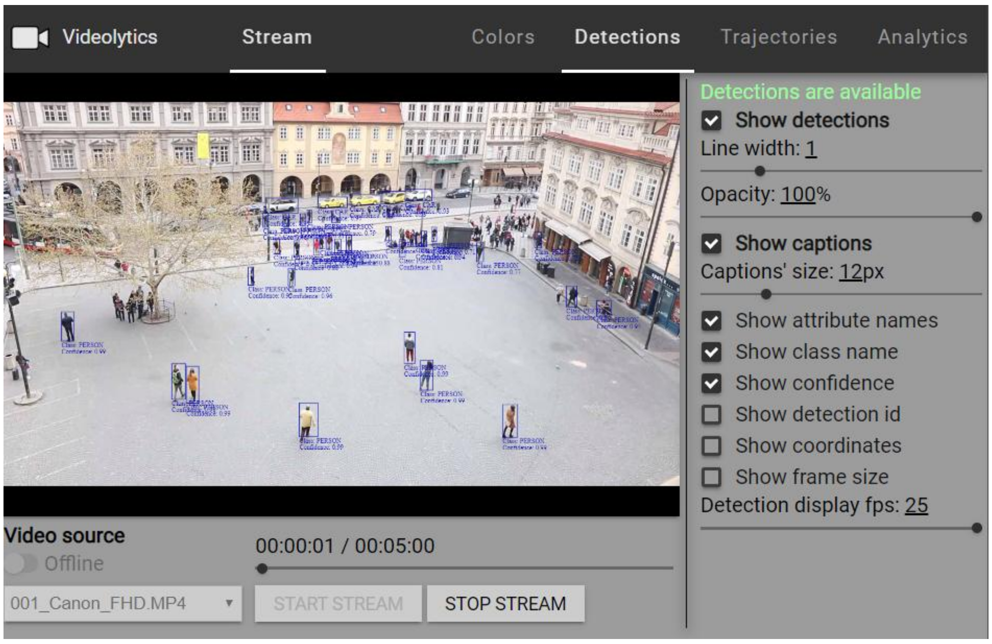
Material design suggests making UI flat. While remaining flat it can be divided into background and top layer, where the top contains important elements such as the video, buttons or header. General logic is to make those top elements more contrasting and closer to pure black and white while also casting a shadow to the background, which strives for the middle gray.

The rule for input forms is to make user avoid mistakes by elimination of text inputs. Combo boxes, progress bars or other elements are used instead. Disabled elements will have lighter gray color.

During the layout design standards of video streaming and web in general are considered: header is located at the top of the web page, video – in top left corner, video controls – at the bottom of the video, and all the visualization tools – in the remaining space at the. Considering use-case list of previous subsection, there are going to be a lot of customization elements. To avoid the user being overwhelmed by this variety they are divided into categories.

Responsiveness manifests itself not only in adjustment to window size (which by the way is also the case). Customization tools panel can have different width depending on which category is chosen, while video width must adjust to its size. The user may also want to change panel's width in the way like any desktop application would allow. All that should be accompanied by appropriate smooth animations. One more way to "respond" to user is to save all his settings and restore them when he comes back.

Following screenshots are showing the resulting design and how it adapts to different window or panel size.



Detections are available
- [x] Show detections

Line width: 1

Opacity: 100%

- [x] Show captions

Captions' size: 12px

- [x] Show attribute names
- [x] Show class name
- [x] Show confidence
- [ ] Show detection id
- [ ] Show coordinates
- [ ] Show frame size

Detection display fps: 25

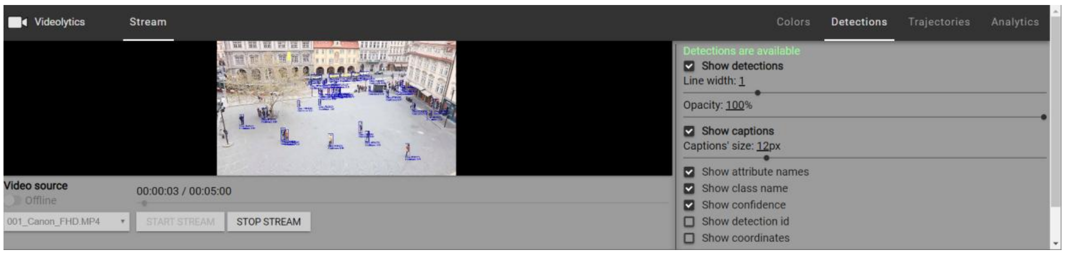
Video source

Offline

00:00:01 / 00:05:00

001_Canon_FHD.MP4

Figure 3.2: Wep app – Compact window layout



Detections are available
- [x] Show detections

Line width: 1

Opacity: 100%

- [x] Show captions

Captions' size: 12px

- [x] Show attribute names
- [x] Show class name
- [x] Show confidence
- [ ] Show detection id
- [ ] Show coordinates

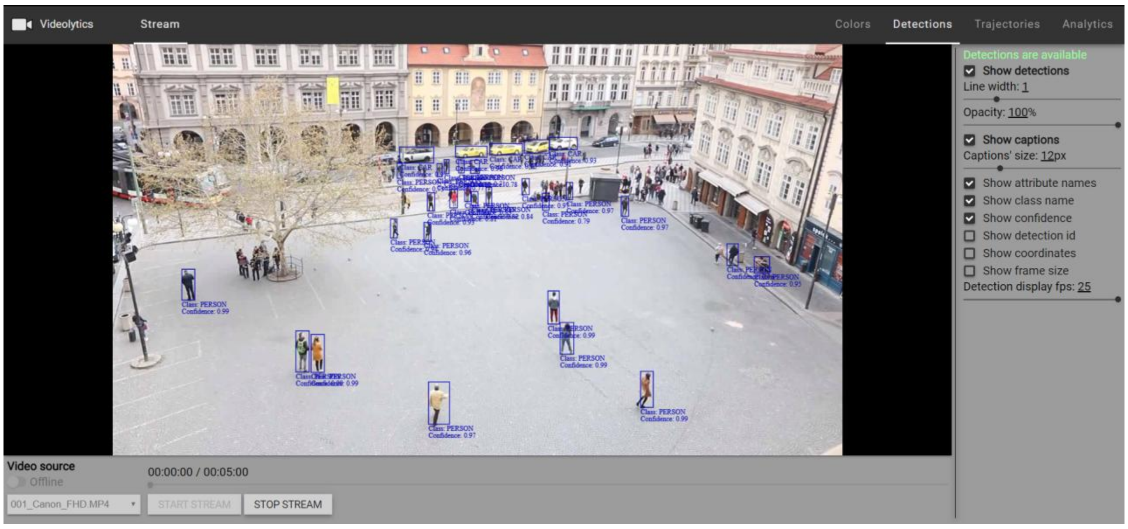
Video source

Offline

00:00:03 / 00:05:00

001_Canon_FHD.MP4

Figure 3.3: Wep app – Wide layout



Detections are available
- [x] Show detections

Line width: 1

Opacity: 100%

- [x] Show captions

Captions' size: 12px

- [x] Show attribute names
- [x] Show class name
- [x] Show confidence
- [ ] Show detection id
- [ ] Show coordinates
- [ ] Show frame size

Detection display fps: 25

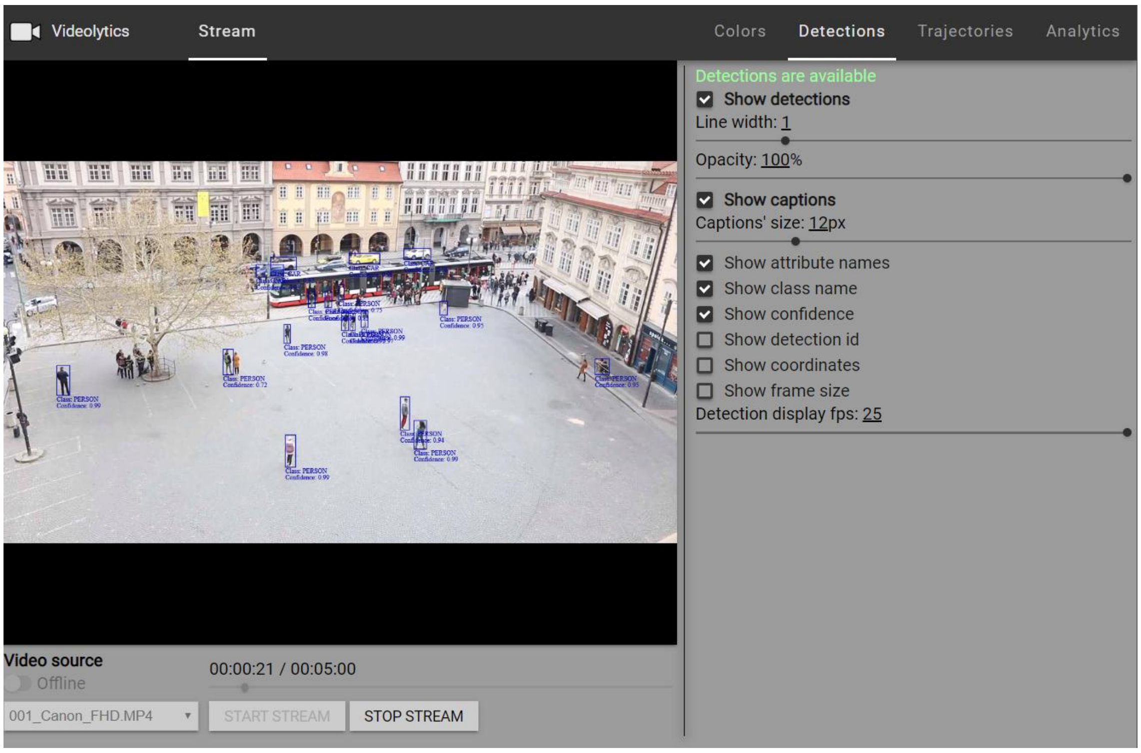
Video source

Offline

00:00:00 / 00:05:00

001_Canon_FHD.MP4

Figure 3.4: Wep app – Fullscreen window layout



Detections are available
☑ Show detections
Line width: 1
Opacity: 100%
☑ Show captions
Captions' size: 12px
☑ Show attribute names
☑ Show class name
☑ Show confidence
☐ Show detection id
☐ Show coordinates
☐ Show frame size
Detection display fps: 25

Video source
Offline
00:00:21 / 00:05:00
001_Canon_FHD.MP4
START STREAM
STOP STREAM

Figure 3.5: Wep app – Extended panel

3.7 Architecture

The architecture of the application is divided into two main parts: client-side and server-side. The latter can further be split into container related and database related, which can be server-side only as the user must not have access to the details about containers or the database's credentials.

Organization of server-side processes is based on the concept of microservices architecture (its advantages are described in analysis section). Instead of programming a huge monolithic application, it is divided into individual services as much as possible. Those services can then be initialized by the client with a simple HTTP request. This allows utilizing asynchronous communication: instead of taking care of various application states with flags or anything else, the client simply follows a chain of asynchronous requests and callbacks.

In this subsection a general description is provided, full detailed service specification is located at subsection 4.4.

3.7.1 RESTfulness

Common architecture for creation of web services is REST (Representational state transfer) [38]. The designed architecture follows some constraints of REST:

- Client-server;
- **Statelessness** – the server is somewhat stateless except that it only knows whether the client is connected or not, which is another convenient simplification;
- **Cacheability** is a bit controversial, as the client maintains a cache of data, but also metadata about that cache, so unwanted requests are not made at all.

However **uniform interface** is not reached. Moreover explicit goal was to avoid it, as it could reduce understandability of the architecture. Almost all of the services are limited with GET operation (only one carries out a DELETE), while not simply getting the resource state, but rather a specific part of info about it, so uniform naming makes unclear, what the service is actually providing.

For example, a frame can be considered as a resource, for the implementation of synchronization two services must be defined: to get first available frame or to get a frame nearest to provided timestamp. While there can be a single resource `frame`, it is not clear that sending a timestamp as part of HTTP request will completely change the meaning of the response. Function-style naming provides better understanding of what the service does, so these two will have names `getStartFrame` and `getNearestFrame`.

Yet the application is not needed to be RESTful at all, as it does not provide API and accessing its services outside the GUI makes no sense. It does not claim to be, but still may be called semi-RESTful.

3.7.2 Container related services

Container related services provide interface to manage containers and can be categorized by their general purpose:

- **Container startup**: `ffmpeg`, `ffserver`, `liveed`. Response of this service has two phases. First the client receives empty response meaning the server is preparing environment for container startup, second signalizes successful of failed state. More detailed description of this process and why it is designed is provided in subsection 4.2.1.
- **Container stop**: `docker_stop`, `docker_killer`. First is actually stopping the containers, while second watches for client's connection state and shuts down the containers when user stops the stream or leaves the page. More detailed description of this service is provided in subsection 4.2.3.
- **Informational**: `ffserver_status`, `list_local_videos`, `video_duration`. There services behave in a simple request/response manner and provide some information to the client. The first one is a wrapper around FFserver status page, the reasons why this is needed are discussed in subsection 3.2. The last service can fit into all the categories, as it starts container, provides some information and then cleans up after itself. More information about it is provided in subsection 4.1.4.

- **Helper**: `stay_connected.php`. This service is used only by the other ones and is actually a code that changes default service behavior when included. It makes service to not stop its work if the client closes connection, so that containers will continue their work despite possible network problems. More details are provided in subsection 4.2.3.

3.7.3 Database related services

Database related services are introduces in the order of their call by client:

- Initial state, where user has just came to the web page. At this point he needs to discover available cameras to be able to start the stream. As own live IP camera is not yet available, only files in local storage are listed with help of service `list_local_videos`.
- When the camera is chosen, further information about it is obtained with help of `getCameraByName` service.
- If local video with already extracted and stored features is chosen, some information about them can be found in the database and displayed even before the stream start. First the presence of features should be controlled with `haveDetections` and `haveTrajectories`. Then if they are present, further information can be discovered and more specifically a list of object classes (person, car etc.) and a list of trajectory models, obtained with `getClassList` and `getModelList` accordingly.
- Then video stream can be started. Its playback is possible without any metadata, however to know exactly what data are needed to be fetched a reference to database is needed. That is a frame id corresponding to first video frame, which is received from `getStartFrame`. With this frame further orientation inside the database is possible, as frames are connected with camera id and sequence number.
- For better synchronization between video stream and features (described in subsection 4.3) there should be a service that can be used periodically to control exact frame inside database corresponding to current video frame. That cannot be calculated with start frame id only, as unexpected frame drops may lead to synchronization errors accumulating proportionally to time passed. Instead an expensive operation of control by timestamp is periodically called with `getNearestFrame`.
- If an option to generate features in real-time is chosen, corresponding processes (LiveED for detections) should be initialized. It is done with help of container related services that are among the rest creating a temporary camera in the database. To get information about that new camera `getLiveEdCam` service is used, then after the user ends the session by stopping the stream or leaving the page all the temporary data must be cleaned, which is responsibility of `cleanDbFromTestLiveED`.
- Finally to provide all these services access to the database a unified connection service `connection.php` is used. When included by the other services a connection to the database is established.

3.7.4 Client side

The client web application is represented with single HTML page. It is built around the main class *App* and its structure is as follows:

- `stream.html` – main page of the application, includes every other file below, contains all the GUI elements and a minimum amount of code (event handler function names and *App* instantiation and initialization);
  - **`style.css`** – not only actual styles, some additional element classes like `disable-css-transitions` that are not present in the HTML itself and are needed to implement some complex behavior with JS;
  - **`common.js`** – contains scripts that are independent on the rest of the application;
  - **`httpRequests.js`** – the same as common, but specifically distinguishes wrappers for HTTP requests (implementing "send and forget" and "polling" behaviors);
  - **`App.js`** – the main application class, provides access to general application settings, all the managers and other functions;
  - **`init.js`** – implements *App* initialization function, which is separate from the main class constructor. *App* constructor can be called anywhere, while initialization function connects it mutually with the caller's context, so must be called from the HTML page;
  - **`helpers.js`** – a bit similar to common functions, but more specific for the *App*.

Then a list of managers follows. Each of them is a separate class with a reference to the parent class *App*, making possible a communication between distinct managers. Their names speak for themselves:

- **`StreamManager.js`**
- **`DetectionManager.js`**
- **`DetectionColorManager.js`**
- **`TrajectoryManager.js`**
- **`AnalyticsManager.js`** – this part of an application is developed by another student independently as part of another work.

Such structure is convenient way to separate contexts, where each class works with its own set of variables, communicating with parent or other manager classes when needed. In this way additional functions or even managers can be added without interference with already existing ones.

3.7.5 Server-side service diagram

To make server-side services interaction with client clearer than with just a textual description, a diagram was created. It is shown on figure 3.6.

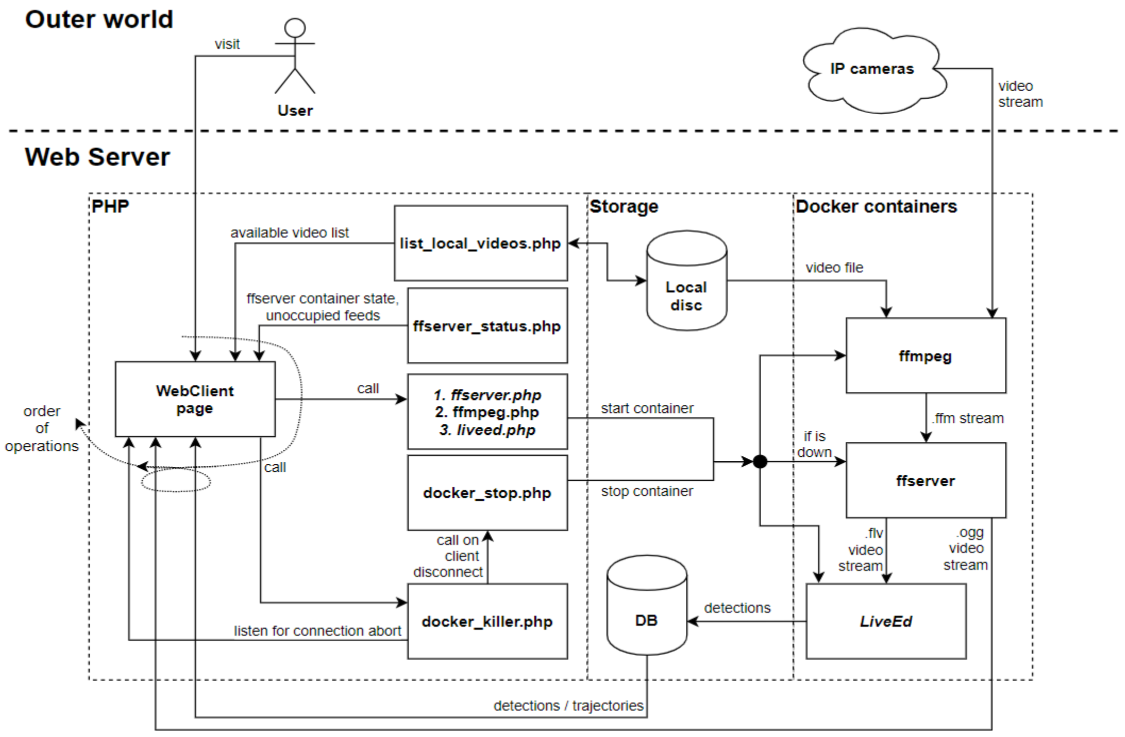


Figure 3.6: Server-side service process diagram

Central part of the diagram is Web Client page (though it is located in the most-left side and not in the center). The order of Web Client's interactions with the services should be considered from the top in clockwise direction:

- Initial phase:
  - The user comes to the web page;
  - Available video list is received.
- Playback phase:
  - Check server status to generate unique id;
  - Start docker containers in the listed order. FFserver is started if it is not running, LiveED is started if real-time detection generation is chosen;
  - Start process killer service;
  - Wait for a delay until features are extracted if real-time detection generation is chosen;
  - Fetch features from the database until the end of work.
- Cleanup phase: `docker_killer` checks if client is connected and stops docker containers if:
  - Client has stopped the stream and has disconnected from the killer;
  - Client has left the page.

That was the most complex part of a diagram, the other ones don't need and explanation.

3.7.6 Additional frame provider service

For the needs of the analytics module one more service `camera` is added. It is independent from the rest of the application and provides a frame image by camera name. It is also is a part of web portal so will be listed in the full specification among the other services.

4 Implementation

4.1 Virtualization

When building docker images with Dockerfile, rather than just installing software it also needs to be configured and run with proper parameters. These parameters can be sent to container when running it with `-e name=value`. Those parameters are then accepted by container and according to Dockerfile instruction put instead of `${parameter}` strings.

4.1.1 FFmpeg container

Dockerfile for FFmpeg image just defines software to install and then runs it with a command line command. All of the implementation features reside in this command line – the way to call it and parameters used:

- FFmpeg should provide constant stream source, for that a specific parameter `-stream_loop` can be used. However testing revealed that this parameter is bugged and not working properly. To achieve the task a workaround was invented. As command line is used to start conversion, all its functionality can be utilized, including infinite loops construct `while :; do …; done`.
- FFmpeg can be converting input faster or slower that the output requires. To read input at its native framerate parameter `-re` is used.
- To start stream simulation from specific position parameter `-ss ${position}` is defined.
- Input video file is also sent using parameter `-i ${videoFile}`.
- H264 codec is enabled with `-c:v libx264`.
- Sometimes when converting video frame drops may occur. By default FFmpeg skips such frames and continues to write output as if nothing happened while also shifting its timestamp. For the developed application this moment is critical, as such shifts along with timestamp rewriting will lead to desynchronization between video and detections/trajectories. Solution to this problem is not a single parameter, but a combination of them:
  - `-use_wallclock_as_timestamps 1` – force FFmpeg to write timestamps according to current system time and not shift them during frame drops (this parameter is not even documented and was found in code examples).
  - `-vsync 0` – passthrough, each frame is passed with its timestamp from the demuxer to the muxer. By default frames can be dropped to ensure constant framerate, leading to timestamp corruption.
  - `-enc_time_base -1` – use the input stream timebase when possible.

- Finally set the output pointing to FFserver feed element with unique user index (as described in **subsection 3.2**): `http://localhost:8081/feed${feedIndex}.ffm.`

4.1.2 FFserver container

FFserver works in pair with FFmpeg. Connection between them is established with special `.ffm` live feed format. FFmpeg just sends output in this format to the address on which the server is listening. Video formats to convert to are defined in server's configuration file instead of specifying them with each FFmpeg call. Actually there is no conversion to .ffm and then to another format, FFmpeg finds out the needed output format from server's configuration and converts directly into it, sending the output directly to the server's output endpoint through the pipeline. While this may seem to be complex at first, it is really easy and convenient way of distributing single video or video stream to multiple independent outputs in different formats.

Dockerfile configuration for FFserver image instructs to install software, copy already written configuration file to the right place (default FFserver configuration folder) and the run the server without any parameters (all configurations resides in the previously copied file).

The configuration file fully defines how server will work. It is quite large (300+ lines), so only the important parts will be described.

First few lines describe general server parameters like port or connections limit, nothing particularly interesting.

The second part is describing live feeds in `.ffm` format. Once run the server will open those feeds for incoming data, and now FFmpeg can establish the pipeline and send data to the defined feed. The order is important, FFserver must be running for the pipeline to arise, and if it's not the convertor will simply return an error saying the output address does not exist. Each feed configuration describes name, maximum size of feed contents and address which is allowed to send the stream (in this case it must be localhost – 127.0.0.1).

The third part is describing the actual stream. Each stream connects to some feed, so there can be multiple streams in different formats generated from one video source. Those formats include video codec, resolution, framerate, bitrate and other. It is not so convenient that there is no way to create some kind of template to configure all stream outputs at once. For the other applications where there is only one constant live stream that might not be a problem. But for the developed one there will be not only live streams, but also stream simulations, which multiple users may want to play and control individually. For this reason many stream outputs must be created, with each user watching its own. Optimistic guess is that the current production server can provide a maximum of 6-7 stable video streams, more will lead to a decrease in performance. So current number of available streams is set to 10. Thus there must be 10 almost identical copies of stream configurations with the only difference in feed name.

There is one more thing that increases configuration file length almost twice. Originally it was thought that web browser and Videolytics modules will use the same stream for their purposes. But during the tests (which unfortunately were not documented and cannot be provided in this work) it appeared that it is done easier with two different formats - `.ogg` and `.flv`.

While most of the browsers natively support playback of both of them, for some unknown reason (probably something connected with different chunk encodings) `.flv` video stream provided by FFserver could not be played with HTML5 video player and inside the player the stream was automatically downloaded by browser, while `.ogg` didn't have such problem. The fact that this player can be used is a big advantage, as it is already implemented in most of the browsers while providing all necessary functionality and no other third-party player is needed.

As for Videolytics modules, it was much easier to work with `.flv` format. Using two different formats may lead to twice as much server load, but cooperation of FFmpeg tools is cleverly organized – because the same codec is used, the stream can be packaged into two different formats without significant load. Thus the only downside of this stream split is increase in configuration file size, which is not a problem at all, as it is written only once.

The last part of configuration file defines server status page name, which will be used to determine unique user id (described in subsection 3.2).

4.1.3 LiveED container

LiveED container runs processes to provide real-time detections generation. It is designed within another work, for this application an example provided in documentation is used. However a few parameters sent to LiveED container are related to this work and are worth to mention:

- --name *'container_name'* – docker parameter
- `-c` *`'camera_name'`* – this actually means camera name inside the database, which will be created at the start of LiveED work. All the detections for the inputted stream are associated with this camera.
- `'stream_url'` – the last parameter for LiveED sent without name, pointing at `.flv` live stream.

Each of these parameter names follow the naming convention, having unique user id at the end.

4.1.4 Duration container

To provide better control of the stream that is generated from local file, application provides interface to set stream start position, which can be anywhere from zero to the end of file.

Maximum possible video time position could be defined statically in database, but that would lead to problems if the file is changed. So a dynamic approach is used, utilizing FFmpeg capabilities. A command line can be run to get video duration. But because it is calling FFmpeg tool, it is also needed to be run inside a container. Video file name must be passed as a parameter, Dockerfile should instruct to simply install FFmpeg and then run a command line with this parameter: `ffmpeg -i /mnt/videos/${videoFile} 2>&1 | grep Duration | cut -d ' ' -f 4 | sed s/,//`

This command will transform information about video duration to a format `H:mm:ss` (H may have more than 2 digits, while `mm` and `ss` exactly two).

4.2 Container management

4.2.1 Startup

First step of working with each mode is choosing the video source where start position may be set for local video. Processes are started in particular order using principle of asynchronous communication – next process starts to load after the previous one has responded with acknowledgement of startup.

- **`ffserver`** if is not running
- **`ffserver_status`** (Live feed status service) to determine unique user id;
- **`ffmpeg`** that connects to FFserver live feed corresponding to chosen id;
- **`LiveED`** (if in real-time mode) that connects to video stream started by cooperative work of FFmpeg tools.

To get process startup acknowledgement two different approaches may be used:

- The simple one. As processes are started with bash commands inside PHP, the same page may also return its status as a response, while keeping connection with client alive. Downside of this approach is that during execution of bash command PHP is not able to do anything else. Running docker container as a daemon could solve this, but then one more layer of control is needed to check if daemon process has not failed. A message signalizing startup of a container may be sent instead, that can cause process chaining problems if they are starting too slowly. This is not the case for defined order as FFserver and FFmpeg are starting immediately, while LiveED is being the last in the chain.
- The complex one – run independent service that will control other services work. While this is theoretically ideal solution, it increases architecture complexity and number of client-server connections to handle.
- The first approach is used until changes are needed.

Containers are started with command line execution within PHP. Client needs to send HTTP request to appropriate PHP page and then the server starts the container. Such process with then run independently from the client, though some response may be sent, for example a discussed above acknowledgement. After receiving it a connection between the client and the server can be closed, so container will still be running in a separate server process while the client doesn't need to maintain active connection for the application work. This is convenient in case of network connection interruption on both client and server sides, as there will be no need to restart all the services. However such independence may cause a situation, where unwanted process continues to run after the client has left. Solution to avoid that is discussed in following subsections.

4.2.2 Communication

Containers are running on the server, but the result of their work is used by both the server and the client. Communication in both directions is maintained with HTTP requests from the client followed with responses from the server. Such connection can be established on the client side with help of XMLHttpRequest or its more convenient jQuery library AJAX wrapper `jQuery.get()`.

The first one in the context of this work is used only to implement custom behavior to get container startup acknowledgement. When initializing such process with PHP a HTTP connection is made. First response from server is empty, meaning the client should wait. Then the server pushes new data to the page, and if that data signalizes successful container startup, client can continue to create his process chain.

Simple request and response is not enough to implement such behavior, as a minimum of two responses is being sent. There are several technologies that can be used here known as real-time web protocols: polling, long-polling, streaming and server-side events. Any of them could be used, but as only two chunks of data are needed to be received and delay between them is very short (even instant most of the time), there is almost no time advantage of using more advanced technologies. Downside is however higher complexity, so the simplest polling is used – connection if kept alive by both sides, client periodically checks for wanted message and disconnects after getting it and then executes callback function. Only a few services use this behavior: FFmpeg, FFserver and LiveED container starters. It is implemented with single JS function `runPhpScript(url, callback)`, which is for simplicity taking as arguments request URL and callback function, same as JQuery asynchronous request function.

4.2.3 Shutdown

As the client leaves when he gets acknowledgement of container startup, there must be some kind of control that will prevent unwanted processes from continuing to run. Obviously this control process must somehow know, whether the client is still using the application or not. As the server cannot directly send requests to the client (as he doesn't listen for them), this responsibility stays on the client. But an extreme case of connection interruption must be considered too, so the client must somehow initialize process shutdown even when he leaves unpredictably.

PHP provides tools that can help to achieve this task. One of them is possibility to continue script execution even when the client leaves (that can be also used to minimize necessary amount of active client connections). This is realized with `ignore_user_abort(true)` method. The other tool is connected with previous one, it provides a flag showing if user has left - `connection_aborted()`.

A service watching for active connection will use those tools, checking in infinite loop whether the client has left. And when so, a cleanup service will be called. This new service is added to the design section.

4.3 Visualization

When the whole process chain is initialized, client can start fetching the data for visualization. There are three factors that are needed to be handled:

- **Fetching** – application must always have data from database to work with. To achieve that a technique is used that can be described as a near future window. The client defines a window that must always be available to him, and when he has less data, he fetches a new portion from the database. The last fetched position is remembered and used when requesting for the new data to minimize amount of database operations. To further reduce server load the amount of fetched data is a few seconds bigger than the window size, so bigger chunks are transferred through less connections when possible.
- **Synchronization** – data from the database are drawn of top of video stream and must correspond to it within approximately 200 ms. To achieve that a combination of three approaches is used:
    - **By timestamp** – both local video and live stream frames contain a timestamp, which is stored inside the database, so the features can reference to it. While at first sight timestamps are completely enough to synchronize data, it is not that simple – there are two major problems. First one comes up from incorrect or missing timestamps, they can be generated on the fly for live stream when missing or set explicitly for local videos, so they are always present, but still are not always perfect. The second problem depends on connection limitations – for a 30 fps video there will be 30 timestamps per second, querying for that makes a huge load on database. But thanks to the other approach there is no need to send so many requests.
    - **By frame sequence id** – local videos that are already processed offline have their features stored inside the database without any frame drops. This allows to synchronize the data only once at the start of the

video, and then calculate frame sequence id offset using the knowledge of video fps. While making less requests to the database, the downside is a high potential for errors. First or a group of first timestamps may be incorrect which will lead to the desynchronization of the whole video. Also frame drops may occur, which can shift the balance unpredictably.

  - **Combination of approaches** – the two already discussed approaches can be combined to eliminate or minimize downsides while keeping advantages of each of them. The idea is to carry out the synchronization by timestamp periodically to reduce load on the database, while the rest of the time go on according to frame sequence id shift minimizing frame drops consequences. If some timestamp is incorrect, it will only affect a small period of time, but then application will continue its normal work.

- **Delay** – feature extraction requires some time, which is constant due to pipeline organization of detection processes. This time between current stream timestamp and features appearance in database is about 2 seconds, this is necessary delay for a live stream to make visualization possible. Offline preprocessed videos don't require a delay.

4.4 Full service specification

All of the services described in design section are further specified here in the following table 4.4. Page width does not allow inserting a description column, which is instead provided in subsection 3.7. Database related services' outputs can be in a form of SQL statement, where the star symbol is a shortcut for all relation fields. For reference a detailed database description is located subsection 3.3.

Table 4.1: Full service specification

Name	Directory	Inputs	Output	Output format
`camera.php`	.	name (string) - name of videofile in /mnt/data/videos	image of camera view	image/ png
`cleanDbFromTestLiveED.php`	./db	uuid	-	-
`getCameraByName.php`	./db	name (text) - camera name	`SELECT * FROM camera`	JSON
`getClassList.php`	./db	cameraId (int)	`SELECT class FROM detection`	JSON

Name	Directory	Inputs	Output	Output format
`getDetections.php`	`./db`	frameLeft (int), frameRight (int) - frame interval	`SELECT id, frame, class, left, top, right, bottom, conf FROM detection`	JSON
`getLiveEdCam.php`	`./db`	uuid	`SELECT * FROM camera`	JSON
`getModelList.php`	`./db`	cameraId (int)	`SELECT model_id, description FROM trajs_models`	JSON
`getNearestFrame.php`	`./db`	cameraId (int), timestamp (int)	`SELECT * FROM frame LIMIT 1`	JSON
`getStartFrame.php`	`./db`	cameraId (int)	`SELECT id, cast(extract(epoch from timestamp) as integer) as timestamp from frame`	JSON
`getTrajectories.php`	`./db`	model_id (int); frameLeft (int), frameRight (int) - frame interval	`SELECT traj_id, model_id, frame_id, x, y FROM trajs_centroids`	JSON
`haveDetections.php`	`./db`	cameraName (text)	true/false	text/plain
`haveTrajectories.php`	`./db`	cameraId (int)	true/false	text/plain
various scripts	`./php_scripts`	(uuid for `docker_killer.php`, `docker_stop.php`, `ffmpeg.php`, `liveed.php`)	Text indicating script state	text/plain

5 Testing

5.1 Approach

The development process is not yet finished because of the mutual dependence of the Videolytics modules. The whole system is constantly changing, introducing new functions (and sometimes bugs too). A change in one module sometimes leads to a change in the other and sometimes opens new possibilities. Considering the nature of the system not only testing but the whole development process sticks to agile methodology, rather than the traditional one. These is a common big goal, but to reach it distinct subgoals are introduced, reached and discussed every week.

This fact and the mutual dependence of the modules define specificity of testing: there is no final big testing, instead it is continuous. Developed visualization tools are used by the other developers which are also taking the role of testers.

Advantage of this approach is constant control of module functionality. When during the system change something stops working, that fact is immediately reported within the group of developers so the solution (sometimes collective) can be found as fast as possible.
Disadvantages of this approach are:

- The whole module testing almost makes no sense. It may provide information about application's state a specific moment in time, but that ceases to be relevant after the first system change. A specific versioning system was not yet introduced.
- As testing is carried out constantly by everyone, it is not properly documented.

For the moment when this work is written all the application parts function correctly. There is an unsolved problem that influences web portal, but it comes from another module, so nothing can be done about it. The problem is that LiveED container won't stop for some reason after sending termination signal for 6 minutes. For that period it blocks the other live feature generation processes, making live visualization not possible, but after 6 minutes it stops and the system returns to normal state. Offline mode works without any restrictions.

Application is located on a web server with address:
`http://videolytics.ms.mff.cuni.cz/`.
This link also acts as a demonstration of Videolytics system.

5.2 Demonstration

Demonstration of application output is shown on figure 5.1, where a video is played and object detections and trajectories are drawn on top of it.

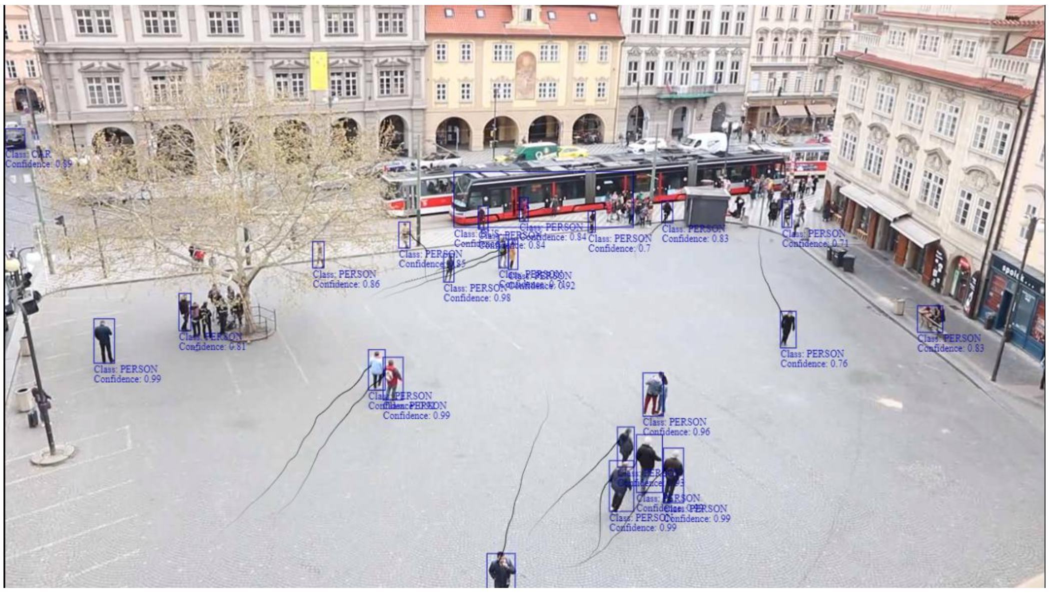


Figure 5.1: App work demonstration

Unfortunately all the possible UI variants cannot be show, at it would take dozens of screenshots. Even so a static image cannot fully represent application's working process, how video width and various visualization parameters (see use-cases at subsection 3.5) can be changed with immediate response from application.

The time for the real-time detection generation module to initialize is 20 seconds, the minimal delay between the actual stream time and the appearance of features is 2 seconds.

5.3 Load

While functionality is tested periodically and is correct for the most of time, the other module parameters are constant and can be measured.

For example the load on server, that can be viewed in context of the database of CPU – the two narrow parts of the system.

5.3.1 Database

Administration tool PgAdmin for PostgreSQL provides a dashboard showing current server load. Interesting parts of it are transactions per second and number of block I/O operations.

Screenshots of the dashboard will be provided with a sum up in form of a table.

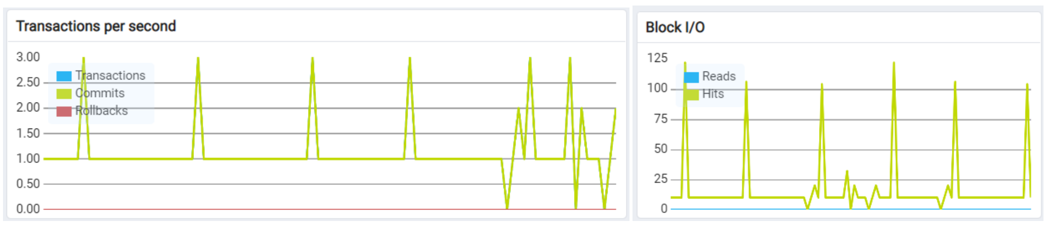


Figure 5.2: Database dashboard - idle

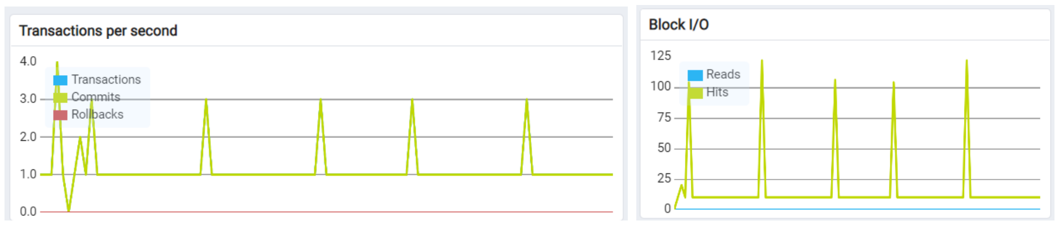


Figure 5.3: Database dashboard – stream only

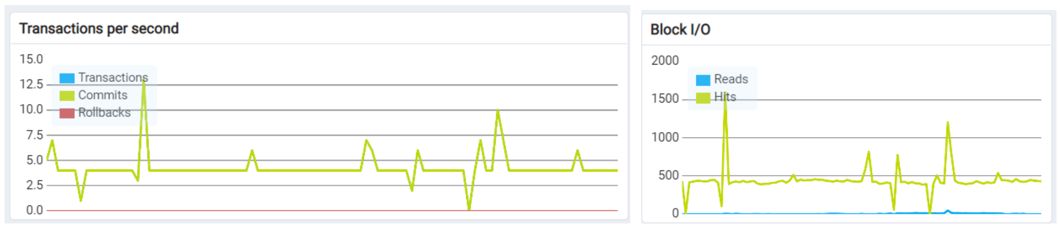


Figure 5.4: Database dashboard – detections only

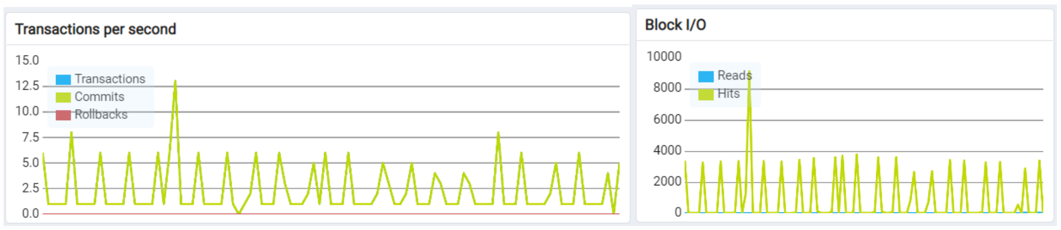


Figure 5.5: Database dashboard – trajectories only

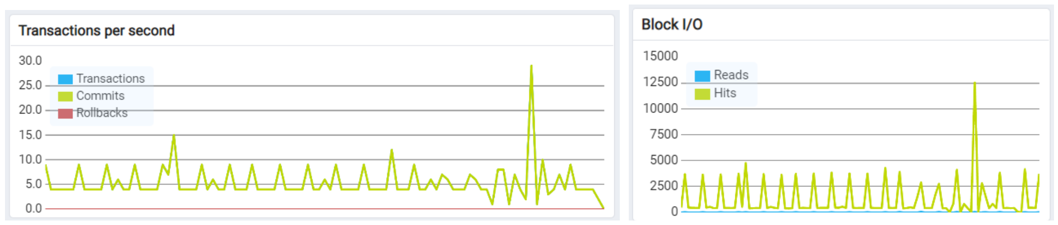


Figure 5.6: Database dashboard – detections and trajectories

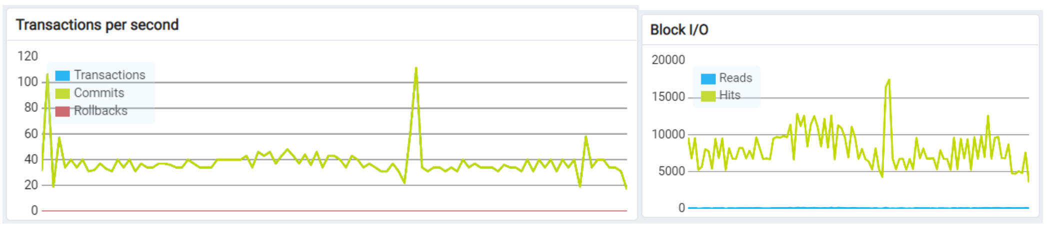


Figure 5.7: Database dashboard – real-time detection generation

Table 5.1: Measurements of database load

Approximate average values	Transactions per second	Block I/O
Idle	1	10
Stream only	1	10
Detections only	4	450
Trajectories only	3	200
Detections and trajectories	5	650
Real-time detections	35	7000

From the results it can be seen, that database load during visualization is low. The load during trajectory fetching is far from constant, but it can be smoothed out only for the cost of more frequent transactions, which will probably only increase server load, as the total amount of blocking I/O operations will remain the same.

For the comparison while LiveED container is running the numbers rocket up, but the database still handles it. The reason of increased load is that web module does only SELECT and DELETE operations, while LiveED also does primarily more expensive INSERTS.

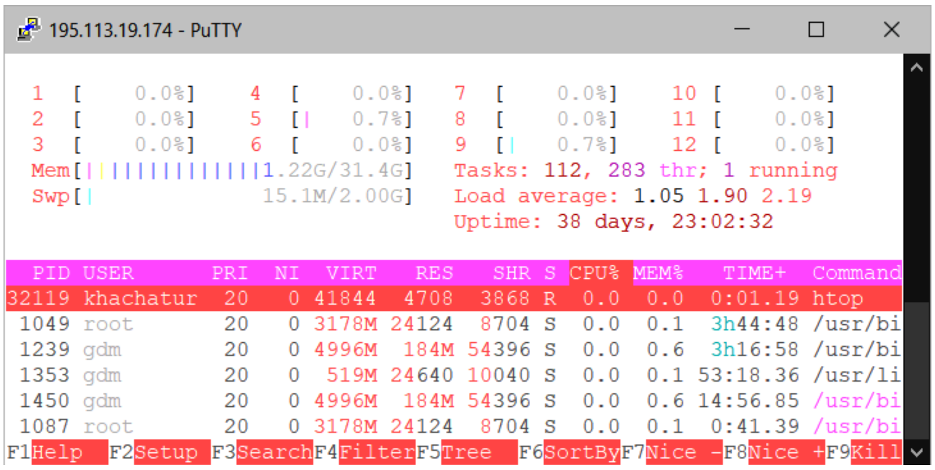
5.3.2 CPU

Database operations are not CPU heavy because of its proper indexation and small complexity of queries. There are inner joins, but between two tables by their index with selection, and the one between three joins relatively small amount of data. Overwhelming CPU resources are taken by video conversion into proper stream formats. This load gradually changes with different video resolution, fps and bitrate. These parameters cannot be too low, as that will not only reduce video quality for the end user, but also make impossible to detect objects.

Current FFserver video settings for all feeds are:

- VideoSize 1280x720
- VideoFrameRate 30
- VideoBitRate 3000

The results of measuring the server load are shown on the following screenshots and in sum-up table. Screenshots have inverted colors to settle better on a paper.

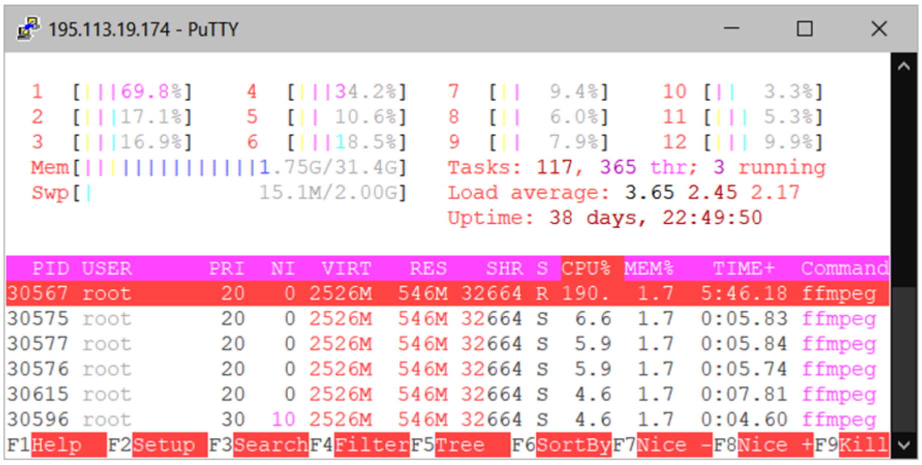


```
195.113.19.174 - PuTTY

  1  [      0.0%]    4  [      0.0%]    7  [      0.0%]    10 [      0.0%]
  2  [      0.0%]    5  [|     0.7%]    8  [      0.0%]    11 [      0.0%]
  3  [      0.0%]    6  [      0.0%]    9  [|     0.7%]    12 [      0.0%]
  Mem[||||||||||||||1.22G/31.4G]    Tasks: 112, 283 thr; 1 running
  Swp[|             15.1M/2.00G]    Load average: 1.05 1.90 2.19
                                    Uptime: 38 days, 23:02:32

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
32119 khachatur  20   0 41844  4708  3868 R  0.0  0.0  0:01.19 htop
 1049 root       20   0 3178M 24124  8704 S  0.0  0.1  3h44:48 /usr/bi
 1239 gdm        20   0 4996M  184M 54396 S  0.0  0.6  3h16:58 /usr/bi
 1353 gdm        20   0  519M 24640 10040 S  0.0  0.1 53:18.36 /usr/li
 1450 gdm        20   0 4996M  184M 54396 S  0.0  0.6 14:56.85 /usr/bi
 1087 root       20   0 3178M 24124  8704 S  0.0  0.1  0:41.39 /usr/bi
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill
```

Figure 5.8: Server CPU load – idle

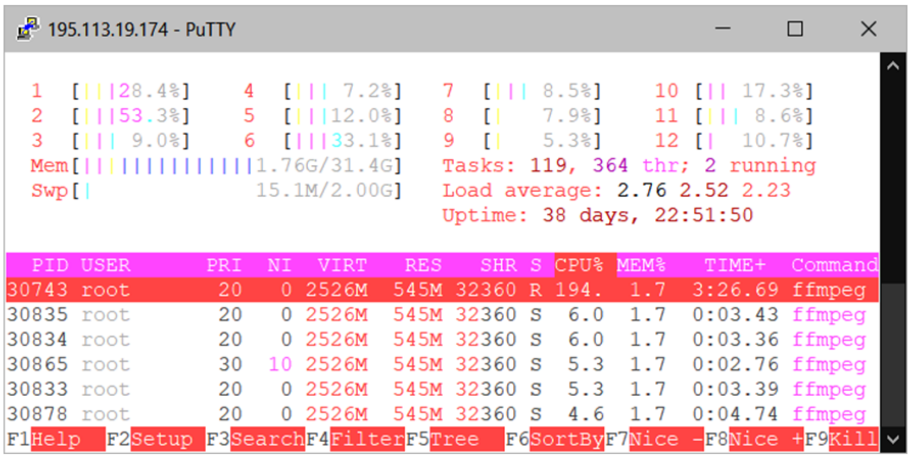


```
195.113.19.174 - PuTTY

  1  [|||69.8%]    4  [|||34.2%]    7  [||   9.4%]    10 [||   3.3%]
  2  [|||17.1%]    5  [||  10.6%]    8  [||   6.0%]    11 [|||  5.3%]
  3  [|||16.9%]    6  [|||18.5%]    9  [||   7.9%]    12 [|||  9.9%]
  Mem[||||||||||||||1.75G/31.4G]    Tasks: 117, 365 thr; 3 running
  Swp[|             15.1M/2.00G]    Load average: 3.65 2.45 2.17
                                    Uptime: 38 days, 22:49:50

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
30567 root       20   0 2526M  546M 32664 R 190.  1.7  5:46.18 ffmpeg
30575 root       20   0 2526M  546M 32664 S  6.6  1.7  0:05.83 ffmpeg
30577 root       20   0 2526M  546M 32664 S  5.9  1.7  0:05.84 ffmpeg
30576 root       20   0 2526M  546M 32664 S  5.9  1.7  0:05.74 ffmpeg
30615 root       20   0 2526M  546M 32664 S  4.6  1.7  0:07.81 ffmpeg
30596 root       30  10 2526M  546M 32664 S  4.6  1.7  0:04.60 ffmpeg
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill
```

Figure 5.9: Server CPU load – stream only



```
195.113.19.174 - PuTTY

  1  [|||28.4%]    4  [|||  7.2%]    7  [|||  8.5%]    10 [||  17.3%]
  2  [|||53.3%]    5  [|||12.0%]    8  [|     7.9%]    11 [|||  8.6%]
  3  [|||  9.0%]    6  [|||33.1%]    9  [|     5.3%]    12 [|   10.7%]
  Mem[||||||||||||||1.76G/31.4G]    Tasks: 119, 364 thr; 2 running
  Swp[|             15.1M/2.00G]    Load average: 2.76 2.52 2.23
                                    Uptime: 38 days, 22:51:50

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
30743 root       20   0 2526M  545M 32360 R 194.  1.7  3:26.69 ffmpeg
30835 root       20   0 2526M  545M 32360 S  6.0  1.7  0:03.43 ffmpeg
30834 root       20   0 2526M  545M 32360 S  6.0  1.7  0:03.36 ffmpeg
30865 root       30  10 2526M  545M 32360 S  5.3  1.7  0:02.76 ffmpeg
30833 root       20   0 2526M  545M 32360 S  5.3  1.7  0:03.39 ffmpeg
30878 root       20   0 2526M  545M 32360 S  4.6  1.7  0:04.74 ffmpeg
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill
```

Figure 5.10: Server CPU load – stream with visualization

The table values are calculated considering the fact that server' CPU has 12 cores. The load of idle state is 1 with 0% CPU load on all cores. Further increases past 1 represent actual server load. For example load of 2 means that one core is fully engaged (though the load can be split between multiple cores).

Table 5.2: Measurements of CPU load

Approximate average values	Load average	Total load, %
Idle	1.05	0.416
Stream only	3.65	22
Stream with visualization	2.76	14.6

There are two things to notice:

- Conversion is split between the cores, that will allow to utilize full CPU capacity when needed;
- Somehow stream with visualization takes up fewer resources, than without it. That is not a measurement error, as the results were double-checked and individual cores also show less load. The reason for this strange behavior is unknown.

Conclusion

As a part of this master’s thesis it was held a related works research about the current state of video streaming technology. Its modern applications, their specificity, advantages and downsides were reviewed in details to highlight features to inherit and problems to avoid during the development.

Then the analysis of research results and existing software and technologies was carried out. At first software requirements were defined and according to them a list of software and technologies for the further development was defined.

After all the tasks and requirements were defined, a design of project was developed. All the different parts of the application were reviewed, algorithms to achieve the tasks and solutions to the problems were found and described. A variant of user interface was proposed. Finally a summary of the project architecture was made, listing all the services to implement and showing their interaction within a scheme.

Then the implementation section follows, explaining the design in further details, providing more concrete view on the technical aspect of the web module.

Finally results of testing were provided with description of used methodology and average numbers characterizing the application in its various working states.

To sum up, apart from original task assignment, this work has also realized the orchestration of various technologies altogether, while at the same time providing simple interface for handling complex server processes.

Future work

Further development of web module within the Videolytics system is being carried out, aiming at more complex tasks and video analytics software market.

Current snapshot of the system will be sent to the Conference on Information and Knowledge Management in June 2020.

Bibliography

[1] IDC: *The Digitization of the World From Edge to Core*. [pdf], ©2018 [cit. 2020-05-07]. Available at: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[2] The Wall Street Journal: *A World With a Billion Cameras Watching You Is Just Around the Corne*r. [online], ©2019 [cit. 2020-05-07]. Available at: https://www.wsj.com/articles/a-billion-surveillance-cameras-forecast-to-be-watching-within-two-years-11575565402

[3] EDPS: *EDPS Guidelines on video-surveillance*. [pdf], ©2010 [cit. 2020-05-07]. Available at: https://edps.europa.eu/sites/edp/files/publication/10-03-17_video-surveillance_guidelines_en.pdf

[4] Livestream Blog – Research*: 47 Must-Know Live Video Streaming Statistics* [online], ©2020 [cit. 2020-05-07]. Available at: https://livestream.com/blog/62-must-know-stats-live-video-streaming

[5] Nielsen Norman Group: Nielsen's Law of Internet Bandwidth [online], ©2019 [cit. 2020-05-07]. Available at: https://www.nngroup.com/articles/law-of-bandwidth/

[6] The New York Times: *Inside China’s Dystopian Dreams: A.I., Shame and Lots of Cameras*. [online], ©2018 [cit. 2020-05-07]. Available at: https://www.nytimes.com/2018/07/08/business/china-surveillance-technology.html

[7] Forbes: *Remember FindFace? The Russian Facial Recognition Company Just Turned On A Massive, Multimillion-Dollar Moscow Surveillance System*. [online], ©2020 [cit. 2020-05-07]. Available at: https://www.forbes.com/sites/thomasbrewster/2020/01/29/findface-rolls-out-huge-facial-recognition-surveillance-in-moscow-russia/

[8] The Washington Post: *Those airport cameras tracking your face may not be legal, study finds*. [online], ©2017 [cit. 2020-05-07]. Available at: https://www.washingtonpost.com/news/the-switch/wp/2017/12/21/scanning-the-face-of-every-american-traveling-overseas-would-be-invasive-costly-and-potentially-illegal-a-new-report-finds/

[9] The Guardian: *Reasons to be fearful about surveillance.* [online], ©2015 [cit. 2020-05-07]. Available at: https://www.washingtonpost.com/news/the-switch/wp/2017/12/21/scanning-the-face-of-every-american-traveling-overseas-would-be-invasive-costly-and-potentially-illegal-a-new-report-finds/

[10] Public Broadcasting Service – TECH + ENGINEERING: *The Limits of Facial Recognition*. [online], ©2013 [cit. 2020-05-07]. Available at: https://www.pbs.org/wgbh/nova/article/the-limits-of-facial-recognition/

[11] CNN Business: Distrust in self-driving cars on the rise after crashes. [online], ©2018 [cit. 2020-05-07]. Available at: https://money.cnn.com/2018/05/22/ technology/self-driving-cars-aaa/index.html

[12] PureTech Systems: PRODUCTS: VIDEO ANALYTICS. [online], ©2012 [cit. 2020-05-07]. Available at: https://www.puretechsystems.com/video-analytics.html

[13] PureTech Systems: PureActivTM Video Analytics. [pdf], ©2012 [cit. 2020-05-07]. Available at: https://www.puretechsystems.com/docs/Video%20Analytics%20Overview.pdf

[14] *AllGoVision Video Analytics*. [online], ©2017 [cit. 2020-05-07]. Available at: https://www.allgovision.com/allgovision-analytics.php

[15] *CrowdANALYTIX*. [online], ©2020 [cit. 2020-05-07]. Available at: https://www.crowdanalytix.com/

[16] *NGINX*. [online], ©2020 [cit. 2020-05-08]. Available at: https://www.nginx.com/

[17] *Apache HTTP server project*. [online], ©2020 [cit. 2020-05-08]. Available at: https://httpd.apache.org/

[18] *Node.js*. [online], ©2020 [cit. 2020-05-08]. Available at: https://nodejs.org/

[19] NGINX: *module ngx_http_hls_module*. [online], ©2020 [cit. 2020-05-08]. Available at: https://nginx.org/en/docs/http/ngx_http_hls_module.html

[20] MUX: *HLS Playback Support*. [online], ©2020 [cit. 2020-05-08]. Available at: https://docs.mux.com/docs/hls-playback-support

[21] *Adobe Flash Technologies* [online], ©2020 [cit. 2020-05-08]. Available at: https://labs.adobe.com/technologies/flash/

[22] *Mod-H264-Streaming-Apache-Version2.* [online], ©2020 [cit. 2020-05-10]. Available at: http://h264.code-shop.com/trac/wiki/Mod-H264-Streaming-Apache-Version2

[23] *Microservice Architecture.* [online], ©2020 [cit. 2020-05-10]. Available at: https://microservices.io/

[24] *PHP vs. NodeJS comparison and benchmarks*. [online], ©2020 [cit. 2020-05-10]. Available at: https://thinkmobiles.com/blog/php-vs-nodejs/

[25] *Database - Oracle*. [online], ©2020 [cit. 2020-05-10]. Available at:

https://www.oracle.com/database/

[26] *MySQL*. [online], ©2020 [cit. 2020-05-10]. Available at: https://www.mysql.com/

[27] *Microsoft: SQL Server*. [online], ©2020 [cit. 2020-05-10]. Available at: https://www.microsoft.com/en-us/sql-server/

[28] *PostgreSQL: The World's Most Advanced Open Source Relational*. [online], ©2020 [cit. 2020-05-10]. Available at: https://www.postgresql.org/

[29] *FFmpeg*. [online], ©2020 [cit. 2020-05-18]. Available at: https://ffmpeg.org/

[30] *HandBrake - Open source video transcoder*. [online], ©2020 [cit. 2020-05-18]. Available at: https://handbrake.fr/

[31] *Format factory – Free media file format processing tool*. [online], ©2020 [cit. 2020-05-18]. Available at:

http://pcfreetime.com/formatfactory/index.php?language=en

[32] *ffserver – FFmpeg*. [online], ©2018 [cit. 2020-05-18]. Available at: https://trac.ffmpeg.org/wiki/ffserver

[33] *GitHub – SRS*. [online], ©2020 [cit. 2020-05-18]. Available at: https://github.com/ossrs/srs

[34] *GitHub - PHP-FFmpeg-video-streaming*. [online], ©2020 [cit. 2020-05-18]. Available at:

https://github.com/aminyazdanpanah/PHP-FFmpeg-video-streaming

[35] *GitHub - mkvserver_mk2*. [online], ©2020 [cit. 2020-05-18]. Available at: https://github.com/klaxa/mkvserver_mk2

[36] *The Top 47 Video Streaming Open Source Projects*. [online], ©2020 [cit. 2020-05-18]. Available at: https://awesomeopensource.com/projects/video-streaming

[37] *Design – Material design*. [online], ©2020 [cit. 2020-05-23]. Available at: https://material.io/design

[38] *jQuery*. [online], ©2020 [cit. 2020-05-25]. Available at: https://jquery.com/

Appendix A
Acronyms

AJAX – Asynchronous JavaScript and XML
CentOS – Community Enterprise Operating System
DASH – Dynamic Adaptive Streaming over HTTP
FFM - FFmpeg Stream File, audio/video format, probably stands for “FFMedia”
fps – frames per second
GUI – graphical user interface
HLS – HTTP Live Streaming
HTML – Hypertext Markup Language
H264 – another name for Advanced Video Coding or MPEG-4 Part 10
JS – JavaScript
JSON – JavaScript Object Notation
MP4 – MPEG-4 Part 14
MPEG – Moving Picture Experts Group
NoSQL – "non SQL" or "non relational" database
RTMP – Real-Time Messaging Protocol
PHP – PHP: Hypertext Preprocessor (recursive acronym)
REST – Representational state transfer
RESTful – services that conform to the REST architecture style
RTMP – Real-Time Messaging Protocol
RTSP – Real-Time Streaming Protocol
TCP – Transmission Control Protocol
UDP – User Datagram Protocol
UI – user interface