UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Collaborative real-time video editor with peer-to-peer connection

Author:
Mariia KULYK

Supervisor:
Oleksandr ZINEVYCH

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Collaborative real-time video editor with peer-to-peer connection

by Mariia KULYK

Abstract

This thesis describes the development of a peer-to-peer collaborative video editor using WebAssembly. We state the reasons behind its development and main technologies that were used. We list some of the existing solutions on the market that have similar functionality and some of the projects that were created with the same technologies as ours. This work describes what we have achieved, explains the potential of our application and shows how it could be further improved.

Acknowledgements

First of all I would like to thank my supervisor Oleksandr Zinevych for his suggestion to work on this topic and for his support.

A special thanks to David Wallace, my personal manager from JustAnswer for his help and support in my education path.

I would like to express my gratitude to the Ukrainian Catholic University for the 4 years that I have spent there and for all the knowledge I gained during my study.

I want to thank my parents for encouraging me to study in the Ukrainian Catholic University and pushing me when it was needed.

Last but not least, I would like to thank all the people I met in this university and who helped me a lot during my studies.

Contents

List of Figures

List of Tables

List of Abbreviations

AST	**A**bstract **S**yntax**T**ree
DOM	**D**ocument **O**bject **M**odel
JS	**J**ava**S**cript
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
WASM	**W**eb **AS**se**M**bly
WebRTC	**Web** **R**eal **T**ime **C**ommunication

Chapter 1

Introduction

More than 30 years ago Web was created as a simple network of documents, connected to each other and intended to be used by scientists in universities around the world. Little does anyone could predict the impact and success of this invention back then. Web has been growing exponentially in size since its first introduction and nowadays it is the biggest and most popular application platform ever, accessible almost anywhere and anytime across various devices.

Today, everything is brought to the Internet, even though a lot of tasks were never meant to be done in the browser. Web applications become more and more complex and demanding: games, 3D rendering, video and audio processing. Consequently, technologies used in web development are changing and becoming more advanced, as well. JavaScript — the only natively supported programming language on the Web, was created as a simple scripting language, intended to make web pages more dynamic, by adding animations. Over the time, JS developed to the world's most popular programming language, that can be used both on the client and server side of the applications. Though possibilities and power of JavaScript are vast nowadays, they are often not enough to perform really heavy-load tasks on the Web.

In 2017, to satisfy the needs of the high-performance web applications, a new technology, called WebAssembly, was created, as the result of collaboration between Google, Microsoft, Apple, and Mozilla engineers. In general, WebAssembly is an assembly-like binary code that is a result of compilation from source languages like C, C++, Rust, etc. Low-level WASM code can be run in modern web browsers and moreover it has a near-native performance. This is particularly useful for running computationally intensive tasks.

1.1 Video Editing

On of the high-load operations, WebAssembly can effectively be used for, is video editing. Video processing is an extremely resource hungry, and it is a challenging task to efficiently process gigabytes of media files in the web app. Because of that, video editors were primarily developed as desktop applications. However, as a demand for accessible from any of your devices and easy-to-use apps grows, more and more video editors are brought to the Internet. Today, the most common architecture for online video editors is a cloud-based approach, where the heavy video processing is done on the cloud-server. Table 1.1 shows pros and cons of this approach [4].

Pros	Cons
Does not need user to have a powerful hardware	Limited editing features
Easy to create an ability to edit video collaboratively	Cloud storage and computing power usage can be costly
Application does not have to be downloaded and is accessible from any computer	Uploading large media files from a retail internet connection can cause long waiting time

TABLE 1.1: Cloud-based approach for video editors

Though, traditional cloud-based video editors have undeniable advantages, they still have some drawbacks, mainly because of the network bandwidth constraints. In this work we are going to investigate how WebAssembly may provide a possible solution. We intend to develop a prototype that could be expanded to a business needs spike. Our intention is to build a demo of a fully in-browser video editor with a client-centered architecture using JavaScript and WebAssembly. Besides, our market research showed that currently there are hardly any applications that provide the ability to edit the video collaboratively. That is why another main goal of our work is to make our editor collaborative, using peer-to-peer approach, so as several users can edit the video simultaneously.

Chapter 2

Background Knowledge

2.1 WebAssembly

WebAssembly (WASM) is a low-level binary code format for a stack-based virtual machine. It is designed as a portable compilation target for programming languages, mainly C/C++ and Rust. In a nutshell, it provides a possibility to run code written in multiple languages on the web with near native performance.

2.1.1 WebAssembly goals

WASM was created with the following goals [16]:

- Be efficient — WASM code can has to be ran at near-native speed by taking advantage of hardware capabilities.
- Be portable — one has to be able to execute WebAssembly code across different platforms.
- Be readable and debuggable — the code should have a human-readable text format.
- Be secure — WebAssembly has to be run in a safe execution environment.
- Don't break the web — WASM should be compatible with other web technologies.

2.1.2 How it works

Historically, JavaScript is the only natively supported programming language in the browsers. That's why in order to explain how WebAssembly works in web and how it delivers a faster performance, we will firstly take a look on how JavaScript runs in the browser.

JavaScript Engine

JavaScript runs in a certain environment, that is most often a browser or Node.js. A software component that executes JS code is called a JavaScript engine. It transforms the human-readable JS code it into the machine code, which can be run by the computer. Each environment has its own engine, the most popular one is V8, written in C++ and used by Chrome and Node.js. Figure 2.1 shows an example of how JavaScript engine works.

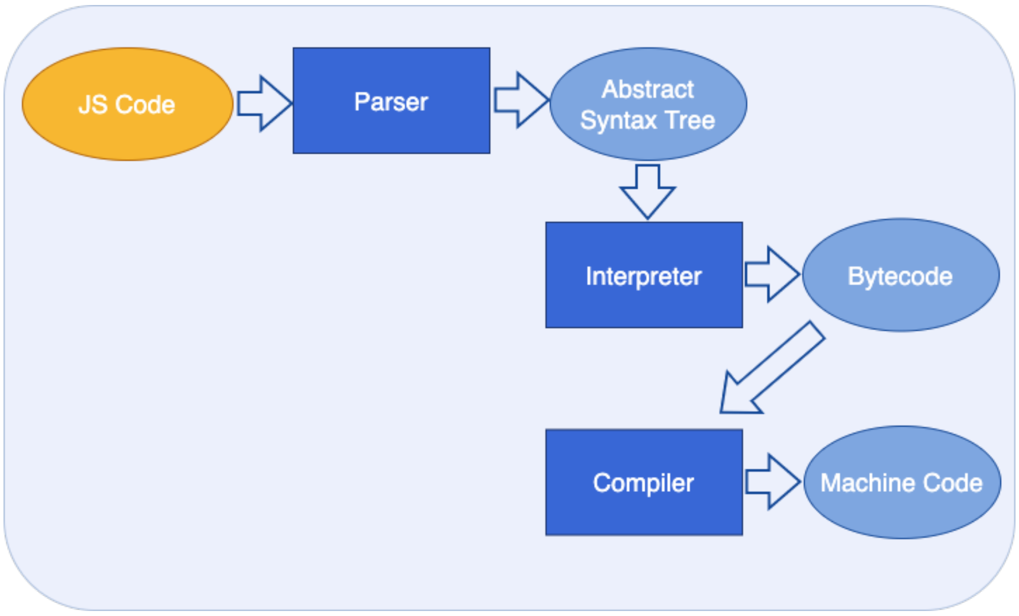

FIGURE 2.1: JavaScript Engine

- **Parser.** JavaScript source code is firstly passed to the parser. This part of engine goes through the code and turns it into an Abstract Syntax Tree (AST), which basically is a tree representation of the code.
- **Interpreter** Based on the AST received, the interpreter generates a Bytecode, which is an intermediate step between the abstract language code(e.g. JavaScript) and the machine code.
- **Compiler** The Bytecode is then sent to the compiler that produces an optimized machine code.

WASM in JavaScript environment

The reason why WebAssembly works faster becomes obvious if we take a look at the Figure 2.2. WASM code goes straight to the compiler, skipping the parsing and transformation to the Bytecode. It is possible, because the format of WebAssembly code was designed to be as fast as possible for the browser to decode. Conversely, JavaScript syntax contains dynamically-typed variables, a lot of redundancy and rules that must be checked before it can be run. Moreover, for JS code, compiler does a lot of optimization. C/C++ code, though, is heavily optimized even before it is transformed to WebAssembly. This means compiler does not spend additional time for doing any optimizations.

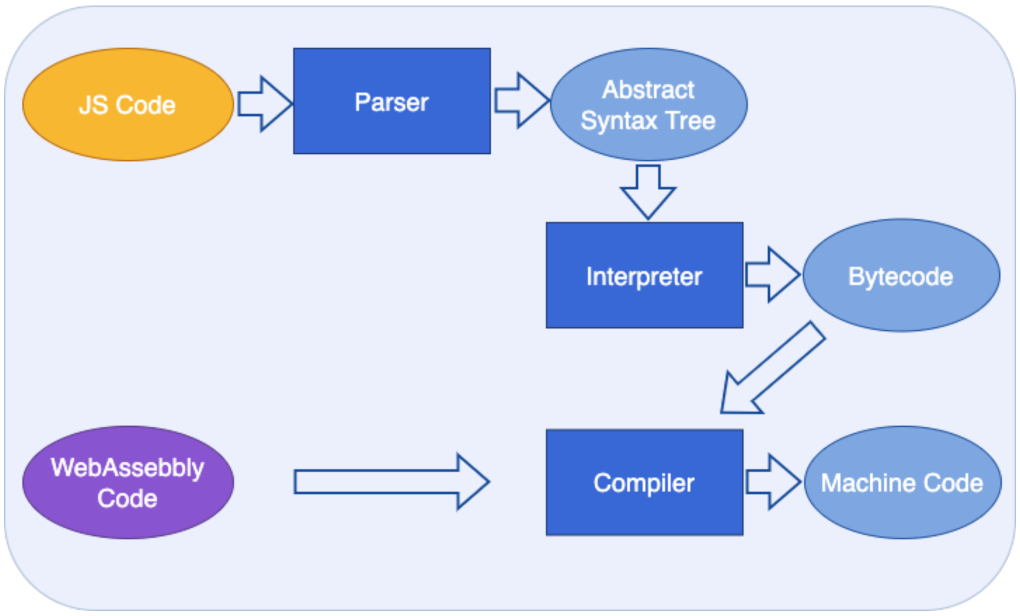

FIGURE 2.2: WebAssembly interaction with JavaScript Engine

Contrary to a popular belief, WebAssembly is not a replacement for JavaScript. It is designed to run alongside JS. WASM is good for computationally intensive tasks, while Javascript is still perfect for a lot of other things on the Web (e.g. DOM Manipulation). With WebAssembly JavaScript APIs, one can use WASM in a JavaScript app. This allows to take advantage of WebAssembly's better performance and JavaScript's expressiveness and flexibility in the same apps, without writing low-level WebAssembly code directly.

2.2 Peer-to-peer

Peer-to-peer is the primary architectural style for distributed systems and usually is referred as opposite to client-server systems. According to this approach, each node has the same capabilities and can initiate connection with other parts of the system. This kind of architecture is commonly used for media sharing and high-performance computing. There are three types of peer-to-peer systems:

- **Centralized** architecture type requires central server to store information about connected peers. This server is used for peer discovery, while data exchange is done between peers directly. The main advantage of such systems is less signaling needed. However, they lack scalability and are vulnerable to attacks.

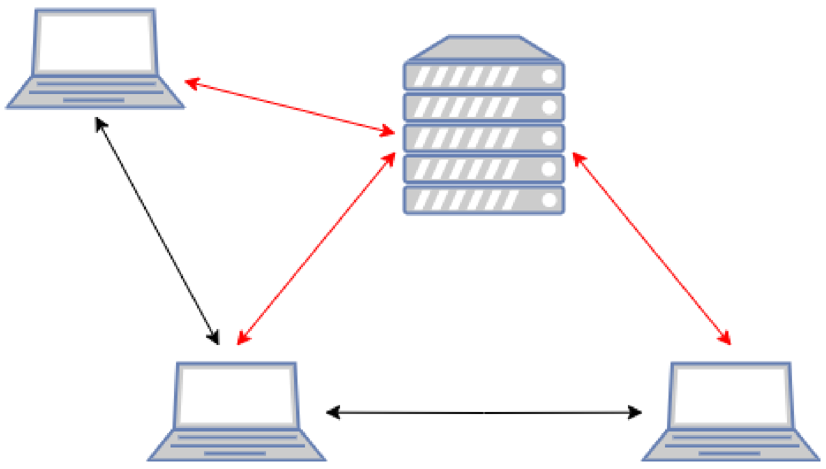

FIGURE 2.3: Centralized P2P system

- **Decentralized** architecture type can be seen as the only “pure” peer-to-peer architecture, since it does not require a central server. It is sustainable to failure and if one node fails, this hardly affects the whole system. Another advantage of decentralized architecture is its scalability. However, it is quite hard to maintain such systems.

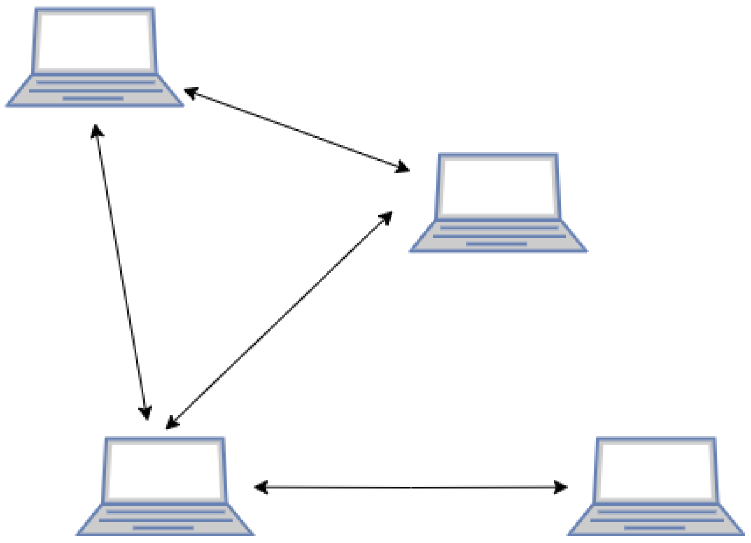

FIGURE 2.4: Decentralized P2P system

- **Hybrid** architecture inherits features of both centralized and decentralized P2P systems. Peers with more computation power become super-peers. Peers connect to super-peers in the same way as nodes are connected to server in the centralized approach. This way super peers become responsible for connection of several centralized peer-to-peer systems.

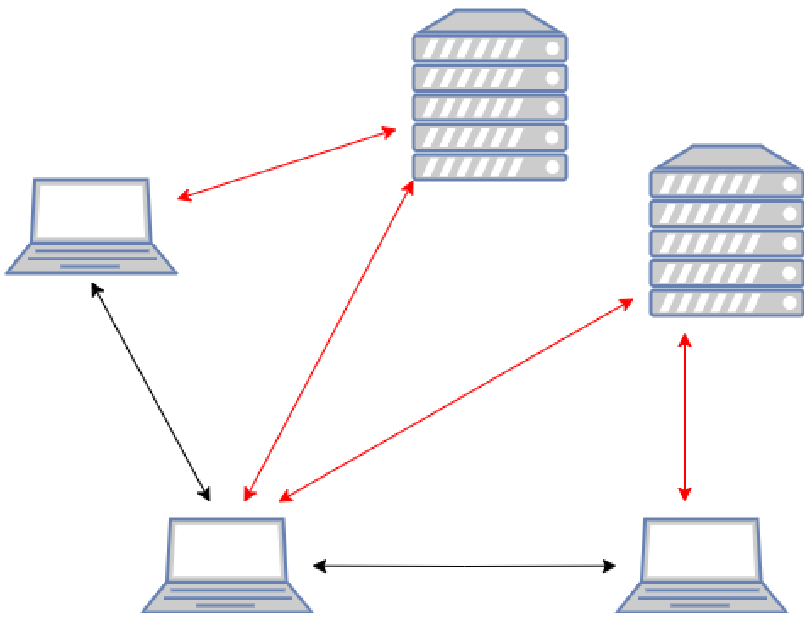

FIGURE 2.5: Hybrid P2P system

Chapter 3

Related Works

3.1 Video editors

Nowadays, there is a great variety of video editors available both online and offline. We will describe several popular apps among them, that implement different approaches to video editing application architecture.

3.1.1 Clipchamp

Clipchamp [6] is a video editor available online that provides some typical video tools, including video crop, trim, adding audio and text. Apart from loading media from local files, it also allows to edit a video, recorded from user's front camera or screen.

This video editor implements client-side approach, meaning that full video production pipeline runs in browser. It does some operations, like storing project data in the database, account creation, login-logout and other management and billing functions in a cloud service (Google Cloud Platform). But all the heavy work related to video processing is done locally, using hardware resources available on the user's end computer [5].

Besides, Clipchamp video processing stack was recently optimized to take advantage of Chrome's WebAssembly SIMD for some operations, including 4K video decoding and encoding. This helped to speed up these particularly demanding workloads and improve performance up to 2.3 times [3].

3.1.2 Veed.io

Veed.io [14] is an online video editor that includes editing tools, like video resizing, crop, trim, adding subtitles, transition, filters. Videos can be loaded from user's computer, recorded from camera or uploaded from YouTube.

Veed.io is an example of editor that implements a cloud-based approach, where all the video processing is done on the external server hosted in cloud. The general idea behind Veed.io architecture is the following. User does all the necessary editing on the front-end. Then a set of instructions for all the edits that user did is created and sent to the cloud C++ renderer. It creates a video based on this information and then user gets the result on the front-end [11].

3.1.3 Flixier

Flixier [10] is an online cloud-based video editor, as well. It provides all the same functionality for video redacting as Veed.io, but it also allows collaborative editing. If multiple people are working on the same video, the changes are automatically synced when user clicks the Save button. Collaborative editing in Flixier is implemented through storing all projects and assets in the cloud and then sharing them to several users. Video processing is also done on the cloud servers.

3.1.4 Shotcut

Unlike previous editors, Shotcut [13] is an open source desktop application, meaning that it has to be downloaded to the user's computer. It is cross-platform and can be ran on Windows, MacOS and Linux operating systems. As other desktop editors, Shotcut has more complex functionality then online video editors, including detaching audio from video clip, deinterlacing, cross-fading, multi-format timeline, a lot of audio editing features, support of 4K resolution and various video and audio formats.

All the video editing operations in Shotcut are done locally on user's device. Main technologies this editor is built with are: Qt — UI framework, MLT — multimedia authoring framework, FFmpeg — multimedia framework, Frei0r — video plugins.

3.2 Other WebAssembly applications

We will now also review several popular applications that are not video editors but they are successfully using WASM in production.

3.2.1 Figma

Figma [9] is a widely-used collaborative interface design tool. This application is written in C++ and previously it was ran in the browser by cross-compiling it to asm.js, which is a subset of JavaScript. After WebAssembly released it became a replacement for asm.js in Figma. The biggest benefit from using WASM there is a faster load time, that includes the time to initialize the application, download the design file, and render the design for the first time. After switching to WebAssembly load time sped up by more than 3 times regardless of document size [15].

3.2.2 AutoCAD

AutoCAD is a 2D and 3D drafting software app, used by engineers and architects. It was developed in 1982 as a desktop application, written in C++. In 2018 an AutoCAD web app [2] was created, that gives users an anytime access to create CAD drawings from any computer web browser. It was achieved by using WebAssembly that brought the whole AutoCAD C++ code base to the web.

3.2.3 Amped Studio

AmpedStudio [1] is an online music sequencer and sound editor. Using it, one can record, create and edit arrangements, process them with effects, connect microphone and music instruments. This type of work is usually done with desktop applications because of the heavy computations that have to be performed for sound generating and editing. AmpedStudio uses WebAssembly to do all this right in the browser.

Chapter 4

Proposed Approach

In this chapter we will describe features we implemented and technologies we were using. While developing our solution, we were focused on two main tasks: implementation of some basic video editing functions and ability for multiple users to work on one video at the same time. The application does not need to establish peer-to-peer connection to work and can be used in a single mode. After connecting with a peer, users are able to edit the same video simultaneously and see each other changes applied.

4.1 Architecture

To implement our app we have used a client-side approach. It is used more widely nowadays and is the opposite to a server-side approach. Server-side means that all the data interpreting and calculations are performed by the server. Then, after client sends the HTTP request, server returns HTML, that is displayed by client. Contrary, client-side approach only accesses server for the data. After it is received and processed, HTML is changed correspondingly to it. Our application needs access to external data only twice: to establish peer-to-peer connection and to upload the video. We quickly determined that the client side approach has limitations because of the system hardware and browser memory. For a production application, we would consider a cloud-based editor engine. The cons of file transmission to and from the engine should then be weighed with the benefits of being able to process larger files faster.

One of our main aims was to use WebAssembly for all the heavy video loading and processing. We ended up using FFMPEG.WASM for all the video editing tools. It enables video processing, converting and streaming right inside the browser.

Another important goal of this work was to implement a peer-to-peer connection within a video editor for the collaborative editing. It was implemented with the WebRTC standart. Figure 4.1 shows the architecture that we designed. Signaling is a process of setting up, controlling and terminating connection. Each peer needs to get its own id and then exchange it with other peer. This communication includes peer server. After establishing connection all data is sent between peers directly.

We are planning to add multiple users support. This will require further investigations to figure out maximal number of peers that can be handled at the same time. Currently, we are considering sticking to four users at one time. Our existing architecture could easily be scaled to this number of users.

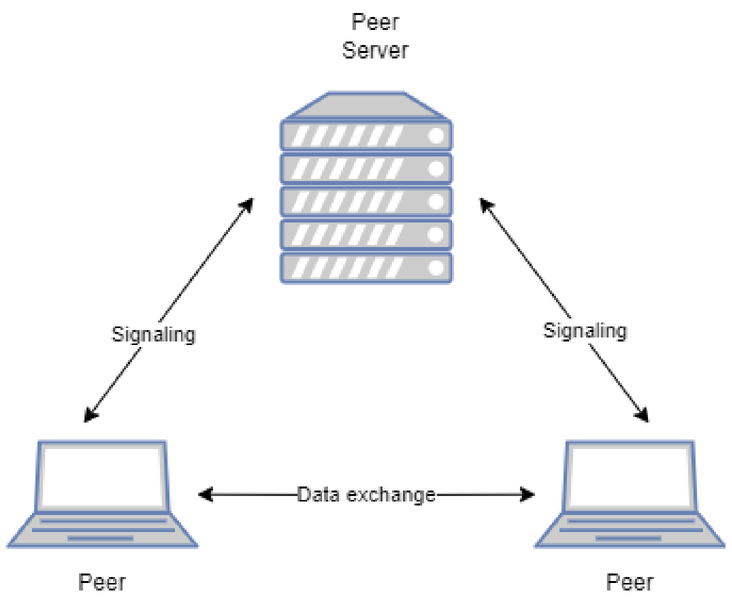

FIGURE 4.1: Peer-to-peer connection

4.1.1 React

React is a library, which allows to build easy-maintainable UI's. It is especially suited for building client-centered applications. Component based approach helps implement an easy maintainable logic. Also React is highly beneficial in working with the Document Object Model (DOM), which is simply the UI of the application, represented as a tree data structure. DOM updates can be expensive, if we re-render the whole UI after each change. To avoid that, React is using a virtual DOM, which is a virtual representation of the real one. If some changes are made in the UI, new virtual DOM tree is created and then is compared to the previous one. After that, the virtual DOM calculates the best possible way to update the real DOM, making as less changes as possible.

4.1.2 WebRTC

WebRTC[17] is an open-source project that allows to implement real-time communication. It supports generic data exchange, alongside with data streams. WebRTC technologies are available for all modern browsers as well as for native clients for all major platforms.

PeerJS

PeerJS[12] is a wrapper for WebRTC browser's implementation. It provides developer the way to easily establish peer-to-peer connection. Peer-to-peer connection can be created with ID solely. For brokering connections Peer Server is used, however peer-to-peer data does not pass through the server, it is used only as broker. PeerJS provides free cloud-hosted server, so that users don't need to run it on their own.

4.1.3 FFmpeg.wasm

FFMPEG.WASM[8] is a Javascript/Webassembly port of FFMPEG. FFMPEG is widely used in recording, converting and streaming video or audio files. It is cross-platform and has wrappers for plenty of programming languages. FFMPEG has a plain JS wrapper but as our intention was to use WebAssembly for video processing on the client side, WASM build of FFMPEG was chosen.

Chapter 5

Application

This chapter describes our current solution, its user interface and data flow. We are also going to compare our application to the existing solutions on the market to find out which features benefit our users most, which features our application lacks and which could be improved.

5.1 App usage

After launching the application user sees the landing page.

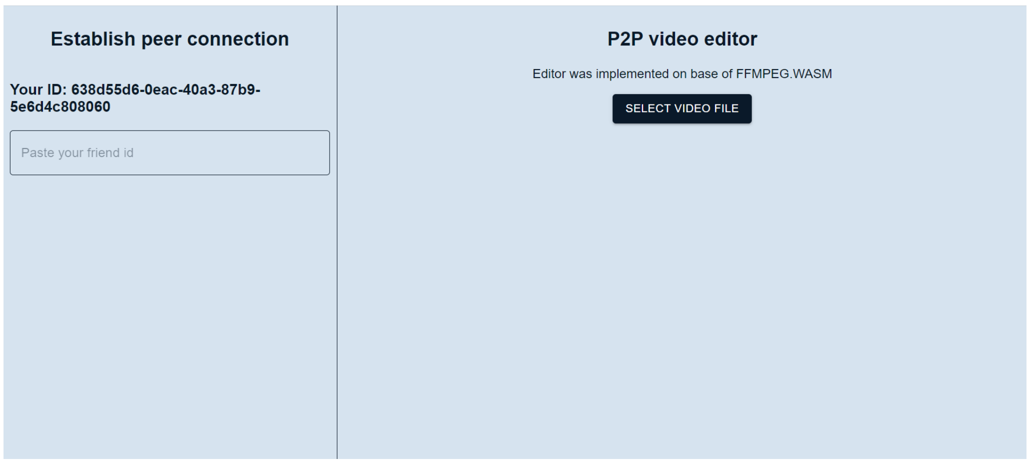
Establish peer connection
Your ID: 638d55d6-0eac-40a3-87b9-5e6d4c808060
Paste your friend id
P2P video editor
Editor was implemented on base of FFMPEG.WASM
SELECT VIDEO FILE

FIGURE 5.1: Landing page

To work in a collaborative mode, each of two users has to paste other's id to the field. After id exchange one of the users chooses video to work on. Editor can also be used in a single mode. In this case user just has to select the file.

After selecting the file, editor page is displayed.

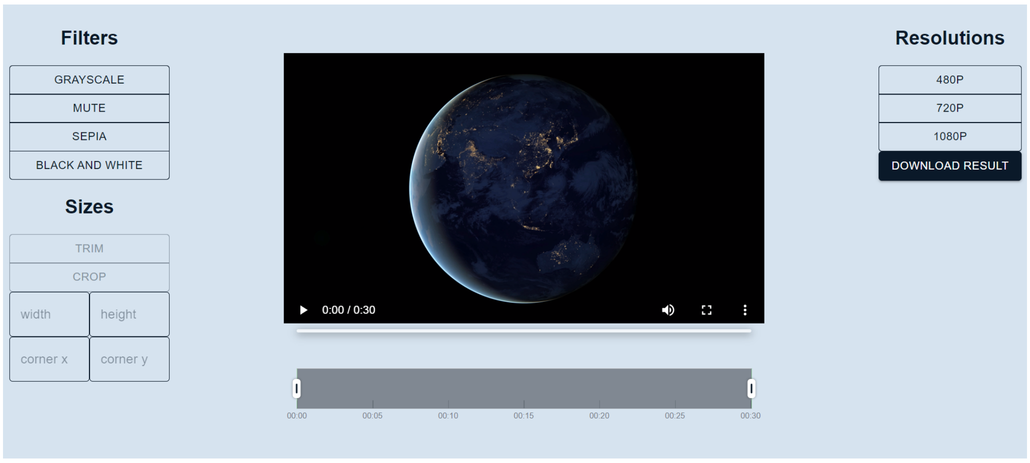

FIGURE 5.2: Video editor page

For now we have the following video editing tools:

- **Filter appliance** changes color scheme of each video frame
- **Trimming** takes off either part from the beginning or end of the video
- **Cropping** allows to cut off video frames and adjust cadres size and proportions
- **Resolutions** changes the number of pixels in the frame

In the collaborative mode, when one of the users applies any function, both users see circular progress until function appliance is finished. After that next function could be used.

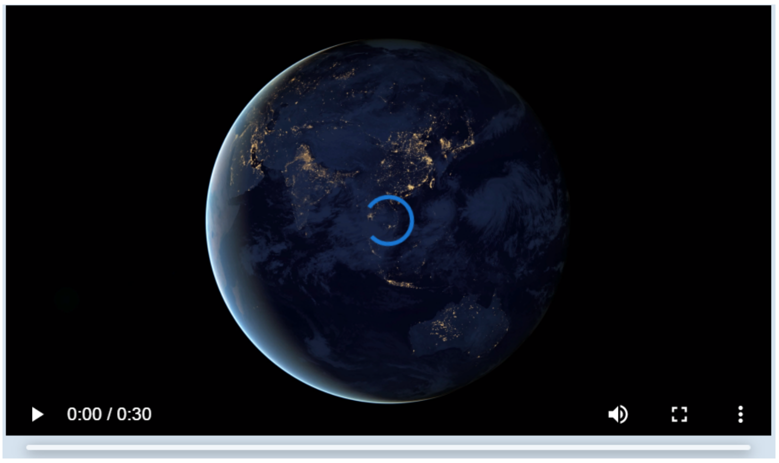

FIGURE 5.3: Waiting view

5.2 Data flow

While user interacts with the landing page, FFMPEG.WASM loads. It will be used to execute all video editing functions. Selecting a file triggers peer ids exchange and sending video file to another peer. When one peer applies any function, the info about the feature used is sent to another user. They receive the name of applied function and other additional info. Then both peers start to execute function each on its copy of the video.

While one function is executing no other could be applied. Triggering appliance of other functions may lead to unexpected results: there are chances that initial function will be applied as well as executing of initial function will be interrupted and new feature will be applied. In future versions we plan to add functions queue so that they can be applied one after another.

In the collaborative mode no other feature could be applied until both peers complete function execution. Otherwise, this can led to the same results as in the single mode. Solution, that is easiest in implementation, is to share messages between peers about the current state. If any of peers haven't completed execution, we put the selected function in queue and run it after the previous execution is ended. When user completes editing they can download the resulting video by pressing the corresponding button. Then edited video will be downloaded in the MP4 extension.

5.3 Comparison to existing solutions

To compare our video editor with any other solution on the market, we need to decide from which point of view our comparison will be carried out. We can compare video editing functionality, speed of commands execution and collaboration mode convenience. As our application is still a prototype, we will leave speed out of scope of our comparison. Besides, our editor has much less editing functions than most of the video editors have. That is why we decided to focus our comparison on the collaboration mode.

Clipchamp and Veed.io, we reviewed in the related works chapter, do not offer collaboration tools. Clipchamp documentation suggests using shared account for collaboration needs. Flixier free version allows to share the video link. This provides the ability to review the work that has been done and to suggest some changes. Flixier paid plans like "Creator" allow genuine collaboration. Logged in user can create a team and send and invitation via e-mail and after approval new users are able to join the team. All team members can work on the same video, however they are not able to do it simultaneously. If user will try to edit the video while another user is working on it, he will see the pop-up message that states, that project is currently edited by another user.

For today, commercial video editor that allows simultaneous collaboration does not exist. Project that is closest to our solution in its collaboration capabilities is not video editor, but a real-time collaboration platform Evercast [7]. Evercast is a low-latency WebRTC solution, which combines HD life streaming, collaboration with on-screen drawing and time-stamped notes. It is ideally suitable for collaboration, but it is used in a different approach than ours. Evercast does not have an embedded video editor, however it allows to stream usage of any video editing software. While work session is streamed, all users of the team have a video chat. All chat participants have the ability to make some on-screen notes. However, all users, except from the one who is streaming, are not able to interact with the video editing software. Our solution is quite the opposite, as it allows multiple users to interact with the editing tools, but does not provide the ability for team members to communicate. This means, that both Evercast and our solution require additional software to make process of the video editing convenient.

In case our solution will enter the market, we consider targeting it as a solution for non-professional users. Using Evercast is a great option, however it requires user to purchase both Evercast and video editing software, which will cost a lot. Non-professional users can even use out solution with free conferencing tools like Zoom or Skype.

Chapter 6

User feedback

This chapter is devoted to investigation whether our application has value not only as a theoretical project, but as working solution for video editing. We gathered feedback from several users, and despite the overall impression was positive, there is still space for improvement.

> *" This video editor has a simple, intuitive interface and therefore is easy to use. I have firstly tried to use it in a single mode and work on the video by myself. I have used the filters on the videos of different sizes and extensions and they all work as expected. I would like more effects to be added to the existing ones, as well as the ability to apply them only to a certain selected part of the video.*
>
> *"Trimming of the video works well and I like how it is implemented with the range slider. I also find the crop feature very useful, however I think it would be great to have the ability to use it not only with coordinates, but also with a selection tool directly on the video area. It is nice that the result can be downloaded in different resolutions, having different extensions there would be useful, as well."*
>
> *"What I like most about this editor, is the ability to edit the video with another person. It's simple and is done with personal ids exchanging. I have tried this with my friend and it works well, we were able to work on the video simultaneously and see each other changes. I wish it would be possible for more than two people to use that at one time. Collaborative editing is a really useful feature, I have not seen it much in other editors, so, to my mind, this app has a great potential."*
>
> *" From the cons perspective, I would like to have ability to return to the main menu and start editing another video. Also I think it would be great to merge video timeline and trim timeline functionality into single entity. Or at least make length of both timelines the same."*
>
> *" The core feature I was using the most was the ability to trim or crop videos without losing the initial quality. Personally, I was struggling a lot while searching for a website that could do it for free before discovering this app (all of the popular websites were demanding money for this; otherwise, they would only allow me to download a new video in a much lower quality)."*
>
> *" The app is fully functional and pretty straightforward to use. Of course, there is a performance issue - the video upload is almost instant, but each filter takes 5-10 seconds, depending on the filter. Video editing with a friend was a feature that I had never seen before. At this stage, it's a little confusing (maybe a pop-up with tips and explanations on how to use it along with a call to action button would help)."*

> *"Interface is intuitive, however I have some suggestions for improvement user experience. First of all, current color scheme is not the best choice if we talk about accessibility, especially for elders and people with dyslexia. Also it may not be obvious how to use crop and trim functions. So i suggest to disable them until user puts in required data into corresponding fields."*

We have already analyzed these reviews and added minor fixes to out implementation. Figure 6.1 displays the previous iteration of our interface. We are looking forward to further work with feedback and implement new features.

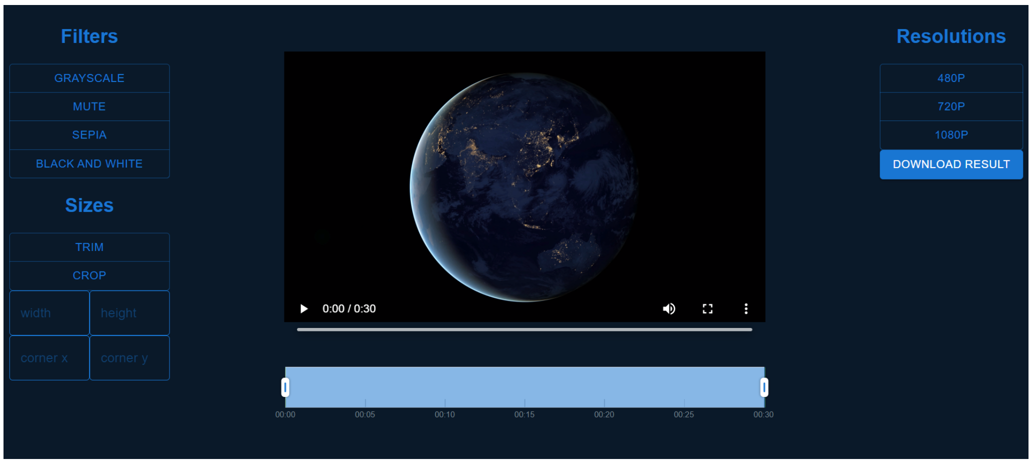

FIGURE 6.1: Previous version of design

Chapter 7

Conclusion

This thesis investigated possibilities of WebAssembly in creating a peer-to-peer video editor application. We showed in theory why WASM can be useful for such heavy-load tasks and have suggested a future spike based on our learnings. Our initial goal was achieved — we introduced a demo-version of an in-browser video editor. With the help of WebAssembly, we managed to do all the video processing fully on the client-side. Our application also provides a possibility to edit the video collaboratively. This was achieved through implementing a peer-to-peer architecture with WebRTC.

To achieve our goals we limited our video editing functionality: we implemented only several basic functions to investigate possibility of the client-based editing. Currently we have limited the number of peers to two. Also we were not focusing on editing speed.

7.1 Future work

We suggest several spikes and ideas to move this project to a production quality peer-to-peer editor. We consider basing the editing engine in the cloud to determine the benefits of beefier and consistently configured cloud servers to perform the editing process. This has to be be weighed against the cost of transmission of the video in and out the server.

A future design could allow SSO login to share video editing. An entire additional system of file management would be handled though such an arrangement. This would be designed with the need to obsolete the linking between the two editors when sharing editing responsibility. Services such as SharePoint could facilitate this idea.

Our peer-to-peer communication is planned to be expanded onto more than two users. We still have to decide on type of architecture we will be using.

As it was our goal to show how shared client side editing could be achieved we offered limited editing capabilities. A viable production quality peer-to-peer editor would incorporate more functionality that would be determined by the needs of the expected audience. For now this product shows a minimal viable solution for peer-to-peer editing and ignores all of the editing functions that would be available in other existing commercial editors.

Also we may integrate some communication functionality such as text or voice chat so that users could coordinate their changes.

Long-term plans include investigating market of mobile-based solutions and creating mobile version of application.

Appendix A

Implementation

Code for the implementation of this project can be found here:
`https://github.com/marusiakulyk/p2p-videoeditor`
App demo:
`https://cheery-melomakarona-012e04.netlify.app/`

Bibliography

[1] *Amped Studio*. 2022. URL: https://ampedstudio.com/.

[2] *AutoCAD*. 2022. URL: https://www.autodesk.com/products/autocad-web-app/overview/.

[3] Sören Balko. *Clipchamp's video editor PWA installs see a 97% monthly growth*. 2020. URL: https://web.dev/clipchamp/.

[4] Marta Chernova. *Should you be editing your video in the cloud?* 2018. URL: https://www.epiphan.com/blog/cloud-video-editing-platforms/.

[5] *Clipchamp: Setting the benchmark for simple, seamless video creation experience*. 2020. URL: https://cloud.google.com/customers/clipchamp/.

[6] *Clipchampt*. 2022. URL: https://clipchamp.com/.

[7] *Evercast*. 2022. URL: https://www.evercast.us/.

[8] *FFMPEG.WASM*. 2022. URL: https://ffmpegwasm.netlify.app/.

[9] *Figma*. 2022. URL: https://www.figma.com/.

[10] *Flixier*. 2022. URL: https://flixier.com/.

[11] Sabba Keynejad. *How to build a video editor*. 2020. URL: https://www.veed.io/blog/how-to-build-a-video-editor/.

[12] *PeerJS*. 2022. URL: https://peerjs.com/.

[13] *Shotcut*. 2022. URL: https://shotcut.org/.

[14] *Veed.io*. 2022. URL: https://www.veed.io/.

[15] Evan Wallace. *WebAssembly cut Figma's load time by 3x*. 2017. URL: https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/.

[16] *WebAssembly Concepts*. 2022. URL: https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts/.

[17] *WebRTC*. 2022. URL: https://webrtc.org/.