# TASK REPORT

## Video Player

Kavindu Dodanduwa

10/10/2018

# Table of Contents

# LIST OF FIGUARES



# LIST OF TABLES

# 1 Introduction

Goal of this task was to create a video player capable of playing videos with different quality levels. Each video will be broken into small chunks. At the client end, these videos chunks should be retrieved, appended to player and played without interrupting playbacl. Selection of the chunk and its quality is determined by a pre-defined policy.

A major requirement of the task was to use HTML5 video player. This enables the video player to utilize live streams as well as support flash video playback.

# 2 Technology Selection and System Design

There were few considerations when selecting the appropriate technology. First consideration was the requirements of the task. Implementation required to support HTML5 video player. So, the design should be done with support for this. Secondly, the technical competency of author had to be considered. Author's previous experience on Python, HTML and Docker was used when designing the system.

As the second step, an overall architecture diagram was created to identify and understand system requirements. This architectural diagram is given below,

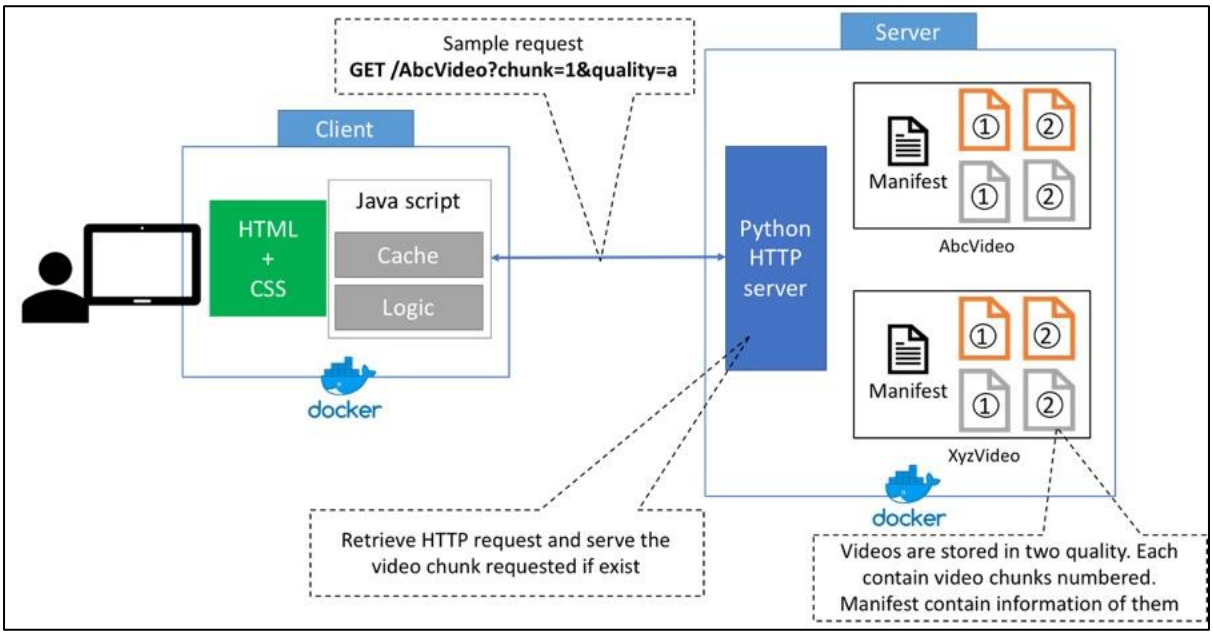


*Figure 1 Overall system architecture*

As highlighted in “Figure 1”, there will be two major components in the implementation. They are the client and the server.

Client component focuses on video player which faces the end user. It is running on a browser and is written and developed using web development technologies. Also, client implementation will be hosted on a http server. This http server will be deployed as a docker image for ease of testing and to support multiple operating systems (ex: - Windows, Mac OS or Linux).

On the other hand, server component focuses on video hosting and retrieval logic. Implementation runs on a Python http server. Same as the client, server will be deployed as a docker image. Hosted videos are chunked for specific lengths. Each segment is hosted in different qualities (ex:- high and low). Server receives chunk retrieval requests which contain identification of the video, requesting chunk id and the expected quality of the chunk. A sample request will look similar to below,

**http://<server>:<port>/videos?id=abc&chunk=abc1.webm&quality=high**

*Figure 2 Sample request to retrieve video chunk*

Server also has the capability to send a metadata document to client. This metadata document contains information on video length, available segments and available quality profiles. This allows client to dynamically detect and request video chunks.

In combination, both client and server complete the design of the video player. Having two separate systems allow individual systems to be altered independently from each other. For example, changing the server component can be done as long as it adheres to the request-response contract established between client and server.

Besides client and server implementation, there were few tools used for video encoding and chunking. Author mainly used ffmpeg and MP4BOX during the implementation process.

# 3 Implementation

First step of the implementation was to create a GIT repository. This allows author to maintain the code base in a standard manner as well as provide a platform to share work with evaluators. Following repository was created for this purpose,

**https://github.com/Kavindu-Dodan/video-player.git**

The implementation was done in iterations. Each week author set few goals and at the end of the week progress was reported. At the same time GIT repository was updated with latest codes. Iterative approach helped author to improve the implementation gradually. Following sections contains details of four iterations (from 11/09/2018 to 10/10/2018) showing implementation details went into final product.

## 3.1 Initial investigations

Initial investigations stated with HTML5 Video player. This element is defined as **<video>** in HTML pages. Developers can simply hard code the source of the video. When the web page is opened in a browser video source will be played automatically.

To test out capabilities of **<video>** element, author first tried to dynamically alter video source through JavaScript. This was achieved in very short time, but video playback was not smooth. There were gaps visible when video segments changed.

Along with these initial investigations, author started working on the server side. Implementation of server side was done on top of CherryPy HTTP server. Video chunks were stored inside server. Chunks were exposed through RESTful web URLs. From client side, JavaScript was used to retrieve these chunks. Then they were appended to **<video>** element for playback.

Furthermore, Docker was used to run both client and server. From client end, author used NGINX to host and expose HTML page with built in video player. From server end, author used lightweight alpine Linux image. On top of alpine Linux image, author created a Python environment to run CherryPy server.

With successful video serving and play back, initial investigation phase was concluded. These investigations, making of Docker images created the foundation for improved version of the video player.

## 3.2 Media Source Extensions

As the next step, author started investigations on Media Source Extensions (MSE). MSE is a W3C maintained recommendation originated in late 2016. The main intention of MSE is to allow JavaScript to manipulate media elements, dynamically. It supports both audio and video playback. Below diagram is extracted from the documentation to highlight the standard's intentions

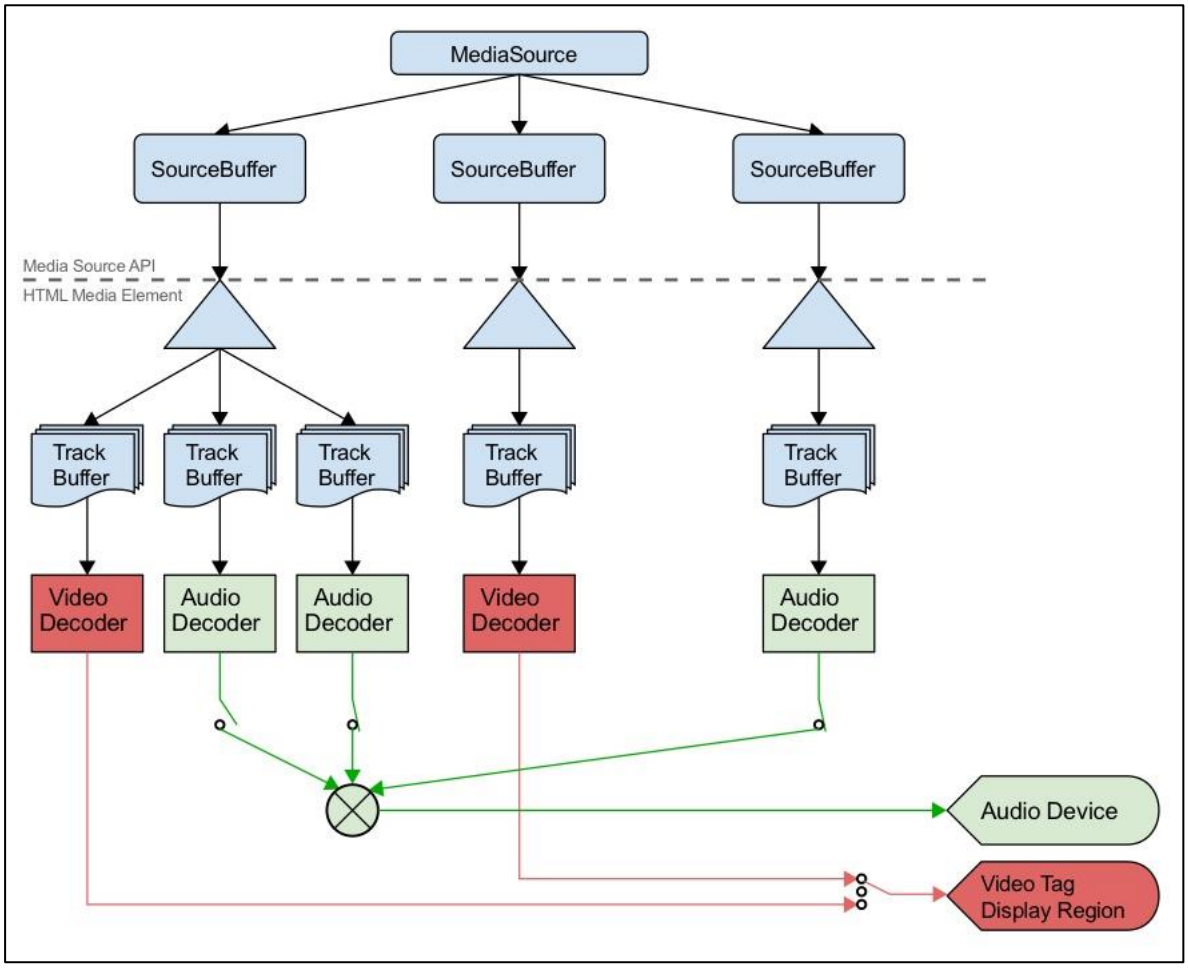


*Figure 3 Media Source Extensions overview*

Though there was a specification and some guides, it was difficult to understand the key concepts of MSE when it comes to implementation. Adding to that author came across video playback issues due to MIME type. One such issue came with MP4 video playback issues due to encoding issues.

Due to these initial issues, author used WEBM video type to test out initial implementations of MSE based video player. Following are the steps involved in playing video segments through MSE based approach.

1. Identify the **<video>** element that needs to be set the video playback
2. Attach a **MediaSource** element to **<video>** by creating an object URL
3. Add a **SourceBuffer** to **MediaSource** created above with the MIME type of video/audio available
4. Wait till **<video>** element open the object URL created in step 2
5. Attach an event listener to handle **SourceBuffer** appended events
6. Append the video chunk to **SourceBuffer**

After the 6th step above, the event created in 5th step will be triggered when video chunk is fully appended. This allows more and more video segments to be added to SourceBuffer. This is the approach used by author to create the video player.

Also, it's a must to detect the final segment of the elements. For this author used metadata document retrieved at the very beginning (before requesting video chunks) from server. This metadata document contained number of chunks, chunk IDs, total video length and MIME type of the video. This information was essential for video player initiation as well as playback. For example, end of video chunks was detected from the number of chunks included in the metadata document. Also, MIME type was detected from this data.

When working with MSE, a key challenge faced was to handle MIME types. Different browsers had different support for MIME types. Below types summaries tested MIME types and supported browsers,

*Table 1 Tested MIME types vs browser support*

| MIME TYPE \ BROWSER | Chrome | Firefox | Safari |
|---|---|---|---|
| WEBM | YES | YES | NO |
| MP4 | YES | YES | YES |

## 3.3 Different quality levels and dynamic quality handling

With the success of MSE based video player implementation, next step was to serve videos with different quality levels.

Due to limited time availability, author selected two quality levels, high and low. Different of quality between these two levels resulted for chunk size difference of four folds. Video player allows users to select video quality at the beginning. By default, video is served in *adaptive* quality. In this setting, video player will alter video chunk quality depending on network bandwidth. Network bandwidth is calculated at the runtime and the next chunk will be fetched based on bandwidth quality. For testing, having a bandwidth greater than or equal to 500 kilo Bytes per second (500 kBp/s) allowed player to use chunks in quality level high. If bandwidth was less, player switched to low quality chunks. These tests were carried out using browser's inbuilt network throttling tool.

## 3.4 Encoding tools – ffmpeg and MP4BOX

For video encoding and splitting, author used ffmpeg and MP4BOX. These tools were new to author, so there was a steep curve.

These tools were used for following purposes,

1. Splitting videos to chunks
2. Encoding chunked videos
3. Identify MIME type in MSE supported format

Once mastered different commands, author created few scripts to support the automation of chunking and encoding processes.

## 3.5 Limitations of the current implementation

Following are the limitations of the implementation,

1. No investigation was carried out on WebSocket to transmit video data
2. Only two quality levels were used for videos
3. No buffering and delaying for chunk loading (All chunks are loaded when SoureBuffer is ready to accept a new chunk)

# 4 Installation and Usage Guide

## 4.1 Prerequisites

The only requirement to run the video player is to have Docker installed in the system. Docker can be installed in Windows, Mac OS as well Linux operating systems. Also, some knowledge about command line operations are required to start and stop Docker containers.

Also, you need to have an internet connection to build the required images. Client and Server components require base images to be downloaded from central Docker registry.

## 4.2 Setting up the video player

Following are the steps required to setup video player

1. Checkout the GIT repository from https://github.com/Kavindu-Dodan/video-player
2. Open a command prompt and navigate to **Client** folder

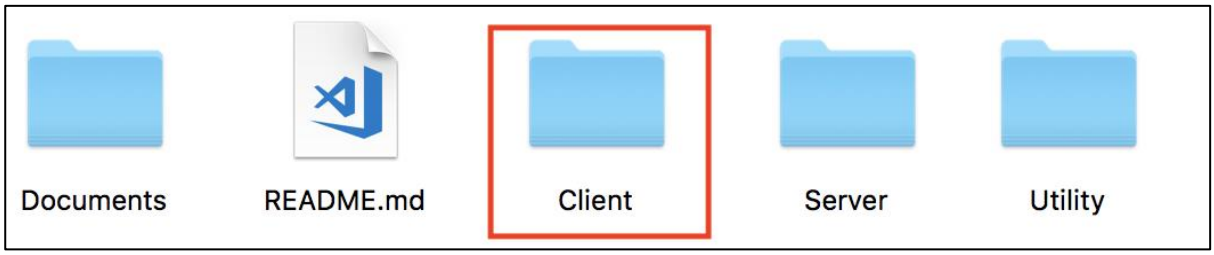


*Figure 4 Client folder of source*

3. Type the command d**ocker-compose up**
   This will spin up a Client container in your machine. Note that, it can take some time to finish the starting at the first time. That's because when you run this command for the first time Docker creates the Client image.

4. Open another command prompt and navigate to Server folder

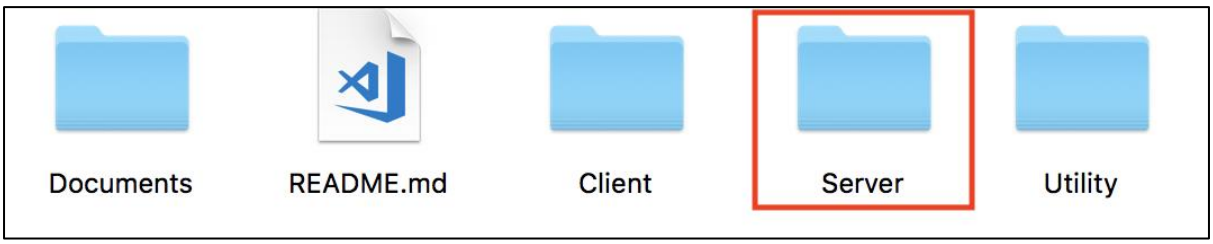


*Figure 5 Server folder of source*

5. Type the command d**ocker-compose up**

   Same as in step 3 this command will take some time to finish executing for the first time.

After successful run of above commands, you will see an output similar to below,

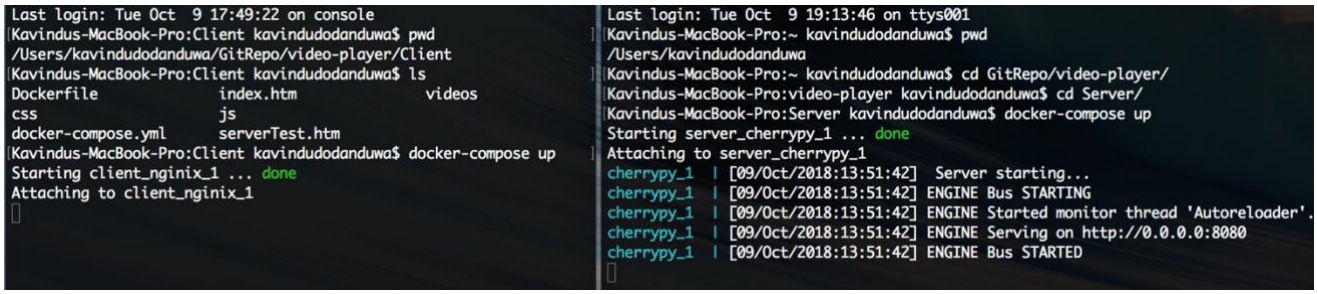


*Figure 6 Successful starting of Client and Server containers*

Now you can access the video plater by visiting URL [http://localhost:8080/](http://localhost:8080/)

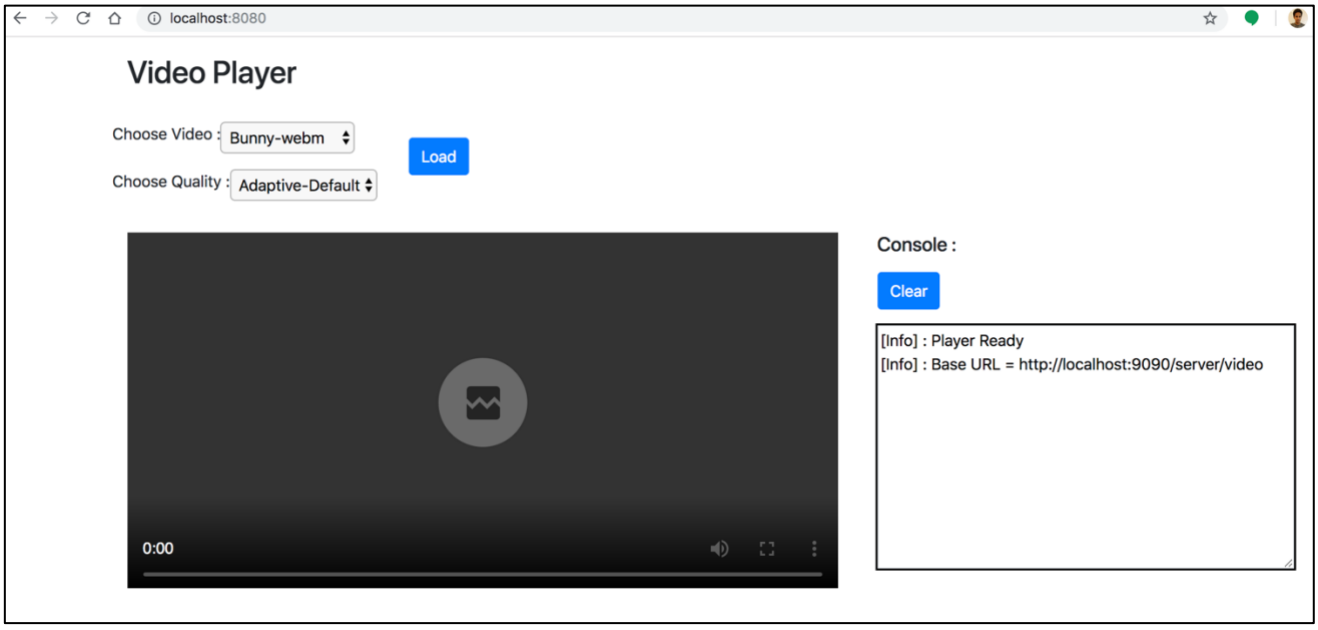


*Figure 7 Welcome page of the video player*

## 4.3 Using the video player

Before trying to use the player, make sure you are running the Server container in your machine. This is essential to retrieve video metadata and video chunks.

There are several control capabilities provided by the video player. You can choose the video to be played, alter the quality level of the playback and observe logging information in the provided console. Following diagram highlight these controls.

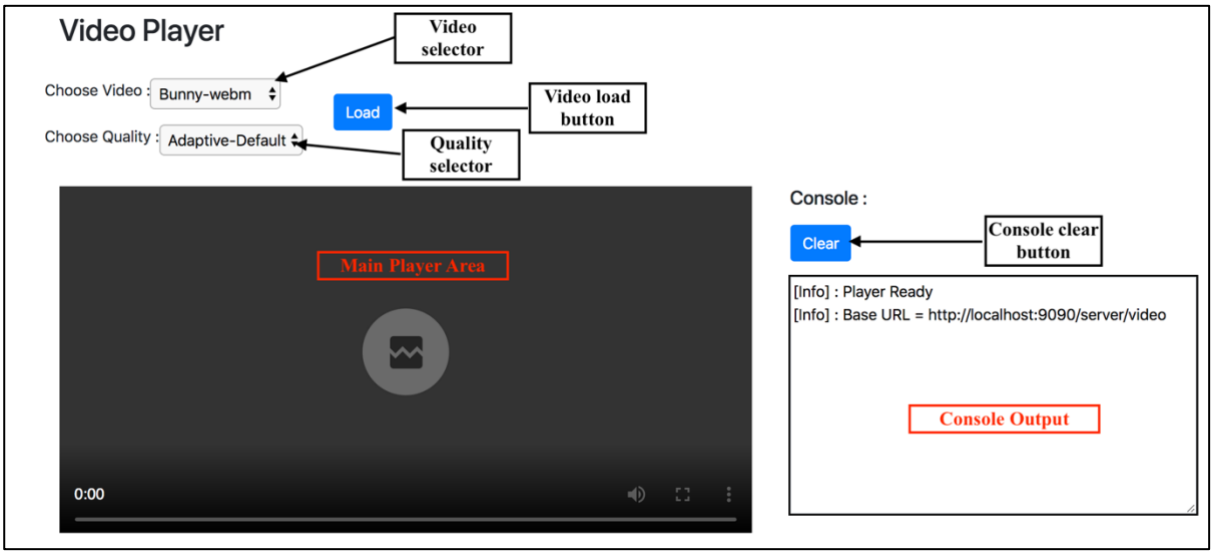


*Figure 8 Video player functions*

You can choose desired video from video selector dropdown. At the moment there are four videos hosted in the server. Note that naming of the video explains the MIME type of the video. Also, if you try to play a video with MIME type the browser you are using doesn't support, you will get an error explaining the issue.

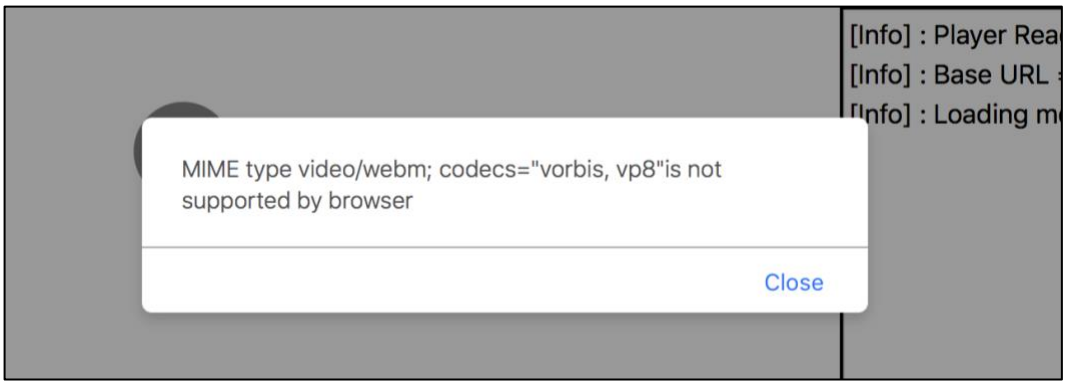


*Figure 9 Playback error due to unsupported MIME type*

A successful video loading will update the console with chunk loading data and video player area will contain a thumbnail of the loaded video. You can start the playback using play button.

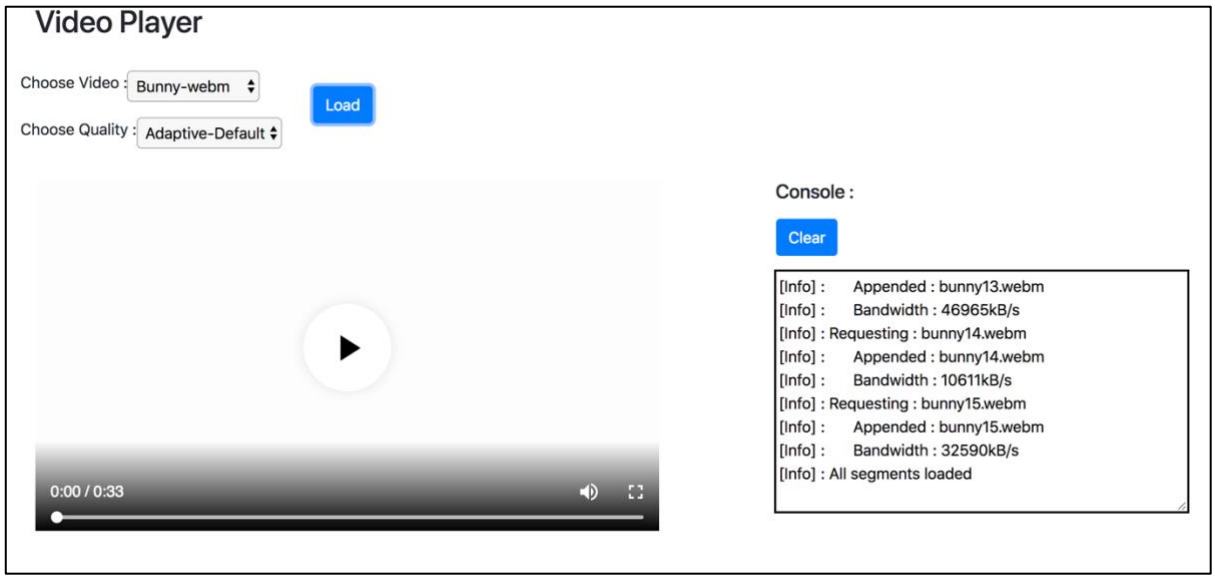


*Figure 10 Video is ready to play*

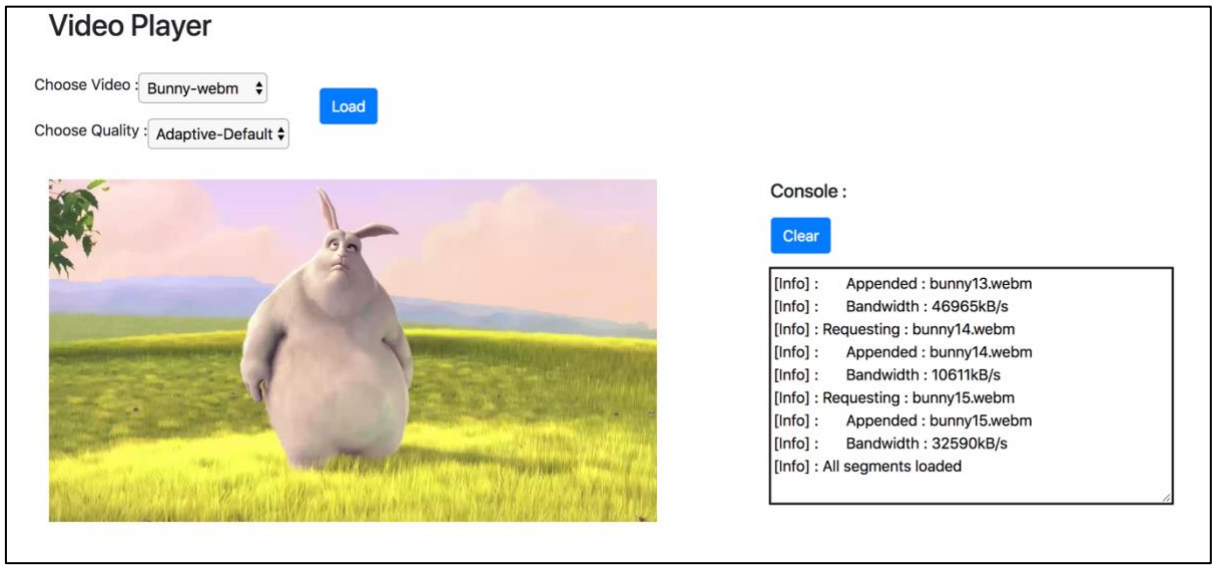


*Figure 11 Successful video playback*

You can try out loading the video in different quality levels. Also, if you use browser throttling, you can see video player load chunks in lower quality.

As mentioned before, bandwidth throttling to limit network below 500 kBp/s will result in video player to request chunks in quality low. This will again reset back to high if throttling is switched off.

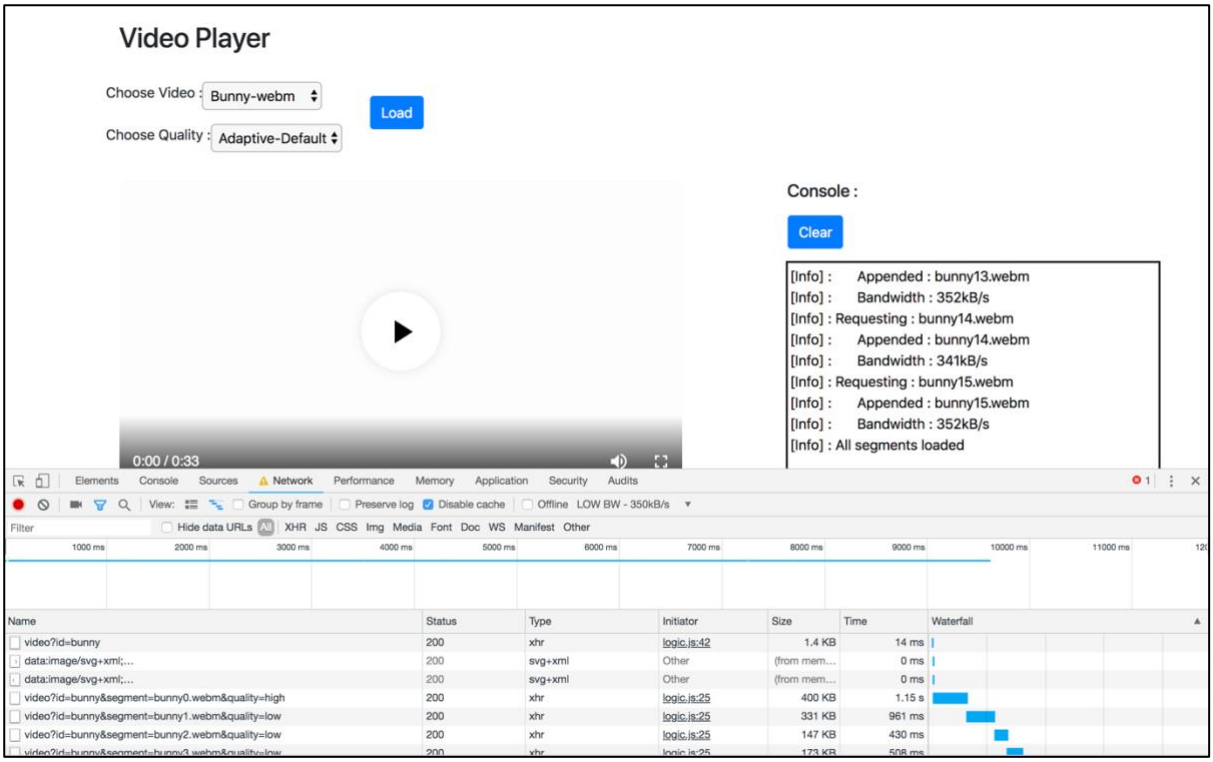


*Figure 12 Adaptive quality selection in action*

# 5 Future work

Current implementation justifies the selection of MSE to implement a HTML based video player. As a maintained standard, it is foreseeable to see more improved, easy to use versions of this standard.

Also, it is expectable to see browsers improving their support for MSE. This is justifiable with increased requirements on video streaming over HTTP and playback through HTML based web pages.

There are several future enhancements that can be done on top of existing implementation. Following list highlights few such enhancements,

1. Testing out WebSocket based communication
2. Enable buffering and reduce memory consumption
3. On-demand loading of chunks based on seeking, time selection by end user

4. Test for different quality levels
5. Implement proper algorithms to decide playback quality

# References


1. Media Source Extension

   https://www.w3.org/TR/media-source/

2. HTML5 Video

   https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video

3. ffmpeg tool

   https://www.ffmpeg.org/

4. MP4BOX tool

   https://gpac.wp.imt.fr/mp4box/

5. Docker tool and installation guides

   https://www.docker.com/

6. CherryPy HTTP server

   https://cherrypy.org/

7. Alpine Linux image

   https://hub.docker.com/_/alpine/