Introduction: Mohamed

Welcome to the second workshop of Advanced Web Technologies Project. Our Topic handles the Video Streaming Mixer Library.

We will present to you the following contents of our Agenda : Mohamed

Wrap up: Problem Statement
Proposed Solution
Tools
Data Structures
Functions
Details on the implementation
Algorithm A and B
Challenges
Demo
and finally, Schedule and Next Steps

35 sec

****

in this slide I would like to highlight the Problem Statement : Mohamed

Adaptive bitrate streaming standards, like HLS, provide a solution to avoid buffering while consuming video streams over the internet.
HLS produces different versions of a video with lower, medium and high quality of resolution and bitrate. In order to enjoy multiple streams in a continuous manner without pausing, these streams should be compatible with each other in terms of resolution. Not every stream shares the same resolutions as the other.
We need to implement a strategy that finds and selects which streams are a match in order to combine them into a single one. We will use the hls-parser library to obtain, manipulate and create a new video stream text manifest with the final master playlist.

1:00 Min

****

the Proposed Solution includes 2 strategies : Mohamed

Strategy A: Match representations against first element of input array
Strategy B: Find intersection of representations

****

to achieve our goal we used the following tools : Mohamed

IDE: **Visual Studio Code**
Version Control: **Git**
Programming language: **Javascript**
free hls-parser library
Free HLS m3u8 URLs: https://ottverse.com/free-hls-m3u8-test-urls/

31 sec

****

Data structures: Yuni

I'm going to talk about the data structures that we're going to use in this project. First we have a String array made of the different URLs that we're going to use as an input. Then we have a Variants dictionary. The keys are the corresponding indexes of the URLs array that I mentioned before and have as a value an array of the different variants of the stream as an object representation. Then we have an Objects array filled with the objects representation of type Playlist of the multiple streams.

Functions: Yuni

Here we have a list of functions that we're going to use in this project. First we have a method called run() that will serve as our main function. In this method we're going to first fetch the text data from the URLs in the input array. Then having this text data we're going to use it to parse the object representation from them using the hls-parser library in order to initialize the data structures mentioned before: the Objects array and the Variants dictionary. Then, we need to use a helping function called remove duplicates because for each master playlist of the input array there can be several variants copies of it that have the same resolution but different bitrate but for now we only care about the resolution so we're going to remove the duplicates present in the Variants array. Also, we will call the method getResolutions() passing an array made of Variants and we need to only get the resolution attribute. Then we have Algorithm A and Algorithm B that I will explain later.

Details on the implementation: Yuni

First we will define an array of strings URLs hardcoded in the program. Then, for each stream URL we will fetch its manifest text data. Then we will parse the manifest text into object representation using the help of the hls-parser library. Then, we will create the needed data structures mentioned before: an Object array and a Variants dictionary. Finally, we will execute algorithm A and B.

Algorithm A: Yuni

We need as a final output a master playlist that contains the exact same representations available from the first video stream of the input array. We will extract the resolutions of the first element of the array from the variants dictionary using the getResolutions method mentioned before. These are our needed resolutions, then we will iterate over the other streams and for each we will obtain its variants. We will compare each needed resolution against all other representations available for every other stream, checking if it is present. We will save the object representation of the matching streams into an array for later use

Algorithm B: Yuni

We need as a final output a master playlist that contains the representations resulting from an intersection, if one exists, of all the streams variants. From the variants dictionary we get the representations for each stream, making an array of arrays. Then, for every stream representation we check if it exists in every other streams variant. The number of times a representation should appear in the other streams in order to be in the intersection should equal the amount of video streams minus one. If no representation is present in every stream the intersection is empty.

Challenges: Poonam
I am going to explain the challenges to execute the Algorithm and give you a demo: We are in the early stages of our implementation of the project, we first tried to find tools (resources) to achieve our possible solution , after parsing we had vast data with lots of duplicate items,which we removed to get better results.

We have lots of variant data with different sets of attributes. mainly we are focusing on bandwidth and resolution to achieve our playlist for the video streaming mixer library.

Demo: Poonam

I'll give you the demo of our result which we are trying to get.

When we run our code with some set of urls of manifest stream we get output for Algorithm A and B.

As my teammate explained both the algorithms,
In first Algorithm A we are taking a set of resolutions with different values of width and height which are present in the first given manifest url and with that we are comparing or trying to find out the exact match, and we have found these sets of resolution with the help of Algorithm A.

And for Algorithm B, we have sets of these resolutions with width and height but here we are not finding any common variants with the same resolution.

It means all have a unique set of resolutions.

For the other set of URLs : we can see we are able to find the match of the variant with the same resolution.

Which we can further manipulate to make a master playlist for the video streaming mixer library.

****

In this slide I'd like to highlight our project's **Schedule**: **: Mohamed**
So today the 14th of June is the 2st workshop taking place: we achieved the following work elements: Implement the first Strategy, Study of HLS Apple and HLS Parser Documentation ,Setup the development environment - the repository, deployment and Parse the HLS manifest into Object representation using the HLS JS Library.
Moreover, the 3d workshop which will take place on 19th of July include: the Output final master HLS manifest with the matching streams, setup as a dependency Library and Improve strategies if needed. Nevertheless the project will be finished on the 31th of July by Submitting the final code and the Scientific paper.

****

the Next Steps will be : Mohamed

Refine and improve the algorithms if needed
Create and output final master playlist manifest using hls-parser with the matching streams from each strategy.
Refactor code to deploy as a dependency. Algorithms A and B should receive URLs as parameters.
1:20 sec