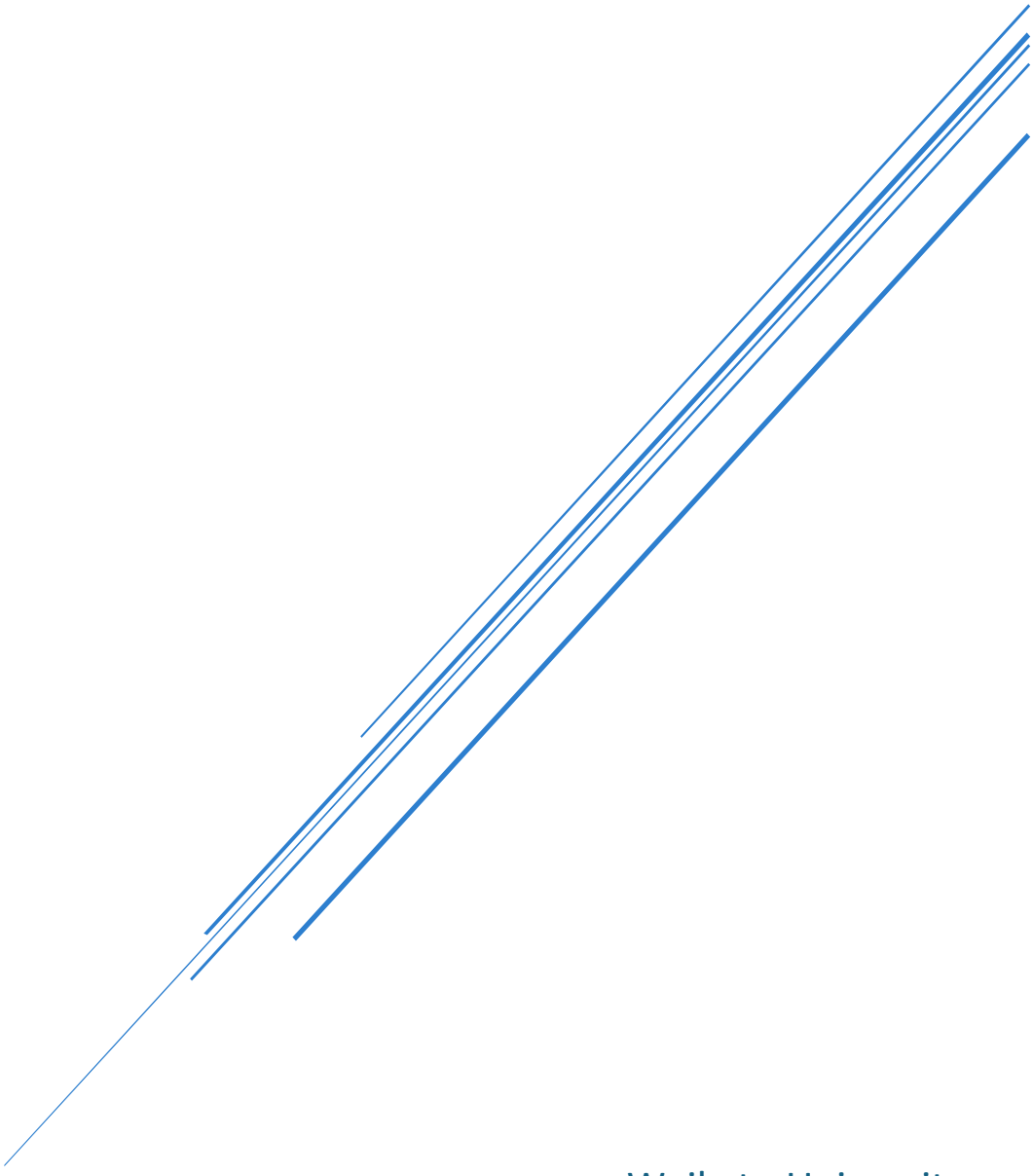
INDIVIDUAL REPORT

Jayden Litolff, Ultimate Video Player

Project Manager: Michael Peddie

Teammates: Luke Fraser-Brown, Shreyaa Senthil Kumar, Jack Unsworth, and Leo Van Der Merwe



Waikato University
COMPX241

Contents

Introduction

The groups project is an Ultimate Video Player that can do everything Generate captions with Speech to Text, Perform Optical Character Recognition, search through the spoken and visual text and provide a usable timeline for longer form videos.

For my role in this project, I took on the responsibility of developing a timeline that is suitable for the purpose of navigating longer duration videos. In a shorter duration video both fine and coarse navigation can be made with the same control, i.e. move the mouse a small amount for precise navigation and move the mouse more for rough / rapid scrubbing. But as video durations get longer the smallest amount of time that the user can navigate is limited by the precision with which they can move their mouse. This loss of precision becomes very frustrating when the user is unable to navigate to an event in the video which would have been trivial to seek to in a shorter duration video. The goal for this part of the project is to restore this precision to the user when watching longer content, without limiting the ability to skip forward or backward in large chunks of time when needed, and without needing separate controls that the user would have to switch back and forth from when needing high or low precision by integrating captions generated by other team members.

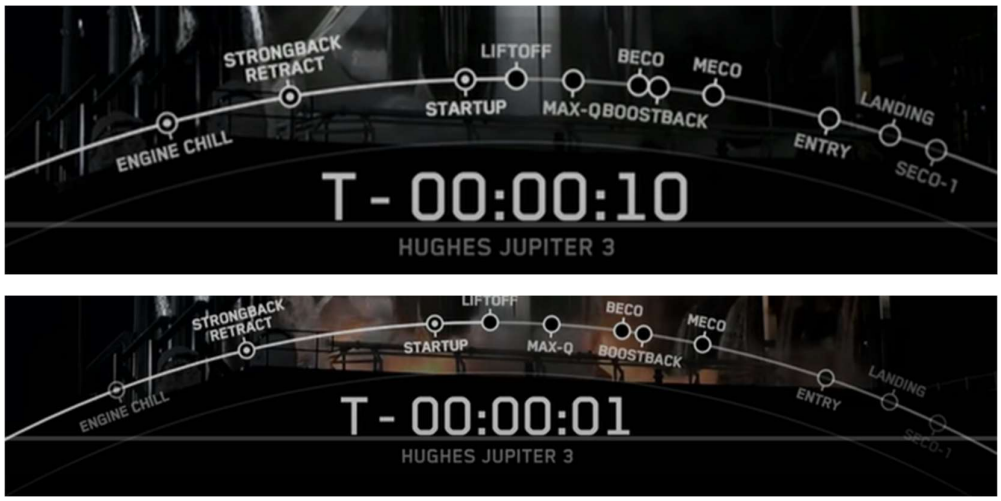
As a part of making the application architecture work, I was assigned to work on the Flask API server to perform basic file handling and calling the other members scripts when necessary.

The timeline is applicable to university students who are frequently in situations where you want to rewind a very slight amount to make the lecturer repeat something, or accurately scrub past chunks of the lecture they don't want to watch. I also frequently run into the timeline issue when watching VOD's (Video on Demand, i.e. livestream replays) on YouTube where I want to skip multi-tens of minutes forward in the video and have the playback land accurately at the start of someone talking or some other event. I usually end up making minutes worth of 10 second scrubs to get to where I want to be.

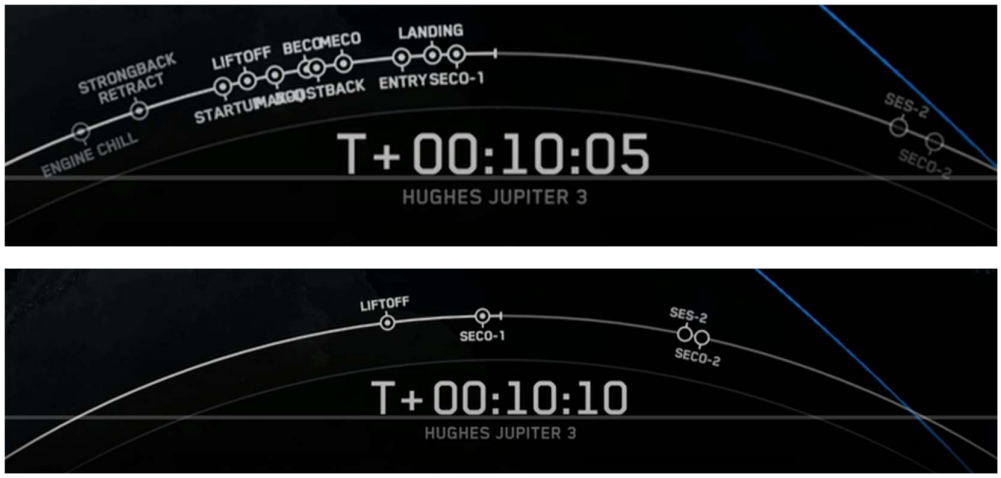
Background

Inspiration

In research for this project, I drew inspiration from the graphical broadcast elements used by SpaceX during their rocket launch livestreams, which represents the scheduled events during the flight with a dynamic timescale that adjusts to the density of events in each phase of flight. A great example of the timeline can be found in the Hughes JUPITER 3 Mission - SpaceX - YouTube, and the adjusting of the timescale can be seen occurring around the T-10 Seconds and T+10 Minutes marks.



Zooming In at liftoff



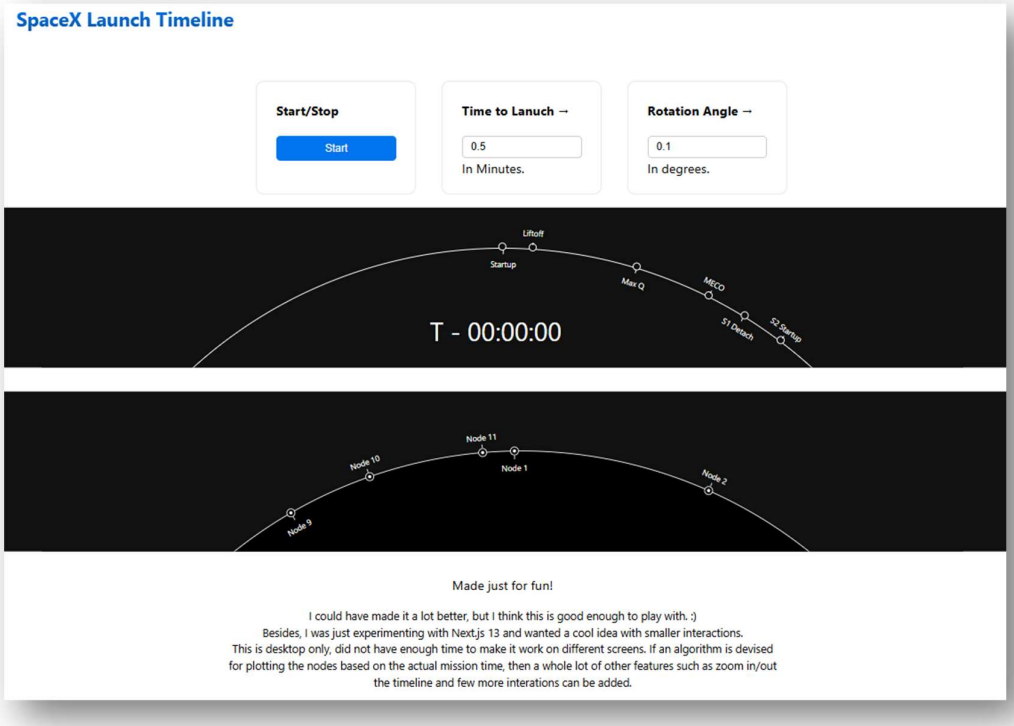
Zooming Out at second stage cut-off

I realised that this timeline solves a very similar problem to the fine / rough scrubbing dichotomy occurring in the video player timeline when watching a longer video. The timeline zooms in when there is a high density of events (i.e. when a user is going to want high navigation precision) and zooms out when the events are sparser (i.e. when a user is going to want to perform a large scrub to skip to the next event).

I wanted to know if anyone had implemented something like SpaceX's on air timeline element as part of a webpage, and with some research I found spacex-launch-timeline.vercel.app and its

corresponding GitHub repository [HarishChaudhari/spacex-launch-timeline - Github] which had some more features when run locally such as adding more events to the timeline.

The project is just an experiment and does not implement any kind of dynamic zooming in/out and does not have any interactivity past the start / stop button. This is where I began building the ArcSlider from.



Technical Elements

The team decided to use the Svelte framework for the frontend as the framework has a great first party demo that showed how to easily interact with video tags with their features.

To build the ArcSlider for our Svelte frontend, I took advantage of the built in svelte/motion module to drive the animation from one timescale to another. Other svelte features were used such as the reactive functions, which simplifies the process of running the ArcSlider handler when the videos timestamp changes.

For the backend API a python server was decided on, as the machine vision and AI model libraries the team were investigating for the video processors were all available as python libraries. The backend server uses the flask library to provide the web server functionality, Socket IO to allow the web server to notify the clients of completed processing and a SQLite database to keep track of the uploads.

Implementation

The slider can't display all the captions that the speech to text processor generates. So, to solve this issue, certain captions are marked as being of interest during the generation of the WebVTT captions

file by the processor. Jack and Luke who worked on the speech to text processor used a combination of detecting certain characters, certain phrases, or gaps between one caption finishing and the next starting to detect interesting captions.

```
WEBVTT

NOTE Generated by STTFunction 06/06/2024, 16:19:18

00:00:00.000 --> 00:00:05.160
Hello everyone, in this mini lecture we're going to be looking at subqueries. A

00:00:05.160 --> 00:00:13.680
subquerie is a query that is nested within another query. This can be useful for

00:00:13.680 --> 00:00:19.040
breaking our queries up into multiple steps as the inner query will provide us a

NOTE for example

IP
00:00:19.040 --> 00:00:25.040
value or set of values that we can use in the outer query. For example, many of

00:00:25.040 --> 00:00:29.720
the ways we've joined tables together can be rewritten to use subqueries

NOTE contains !?

IP
00:00:29.720 --> 00:00:37.600
instead. So let's look an example. Which lecture is teach comp x223? We have
```

An example WebVTT caption file, captions with IP before their timestamps appear in the ArcSlider

When the ArcSlider component is loaded, it is passed a list of all the captions from our VideoPlayer page which are extracted from the <track> tag on load. This makes the captions much easier to work with as the track tag doesn't store the captions in an array, instead storing them in a difficult to work with array-like object. After being passed the list of all of the video's captions, ArcSlider filters the captions down to only those who's ID's == "IP".

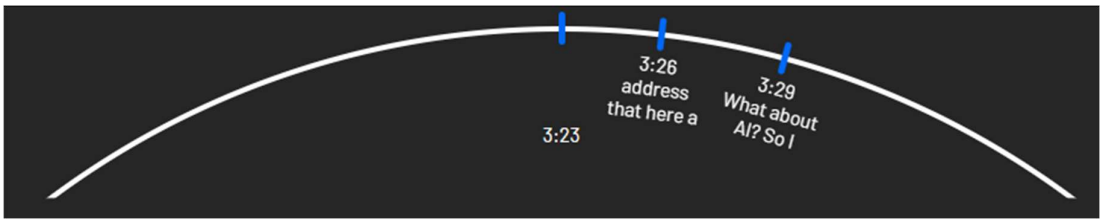
The currentTimeScale is used to calculate a slice of interest point captions that should be currently displayed based on the ArcSlider based off the videos current time stamp plus / minus the current time scale.

For understanding how to layout the html and CSS to actually display the events and slider, the spacex-launch-timeline demo was really helpful. The Arc that the event chips are displayed on is a div with round corners that has been cropped to the top 10%. Each event chip is an invisible copy of the slider div with a rotation applied. The rotation is calculated by determining what percentage of currentTimeScale the events time delta from the videos current time is. The angle of the start / end of the slider is then multiplied by this percentage to determine the angle of the events div. The angle calculations are done in an event handler that is bound to the videos time changing.

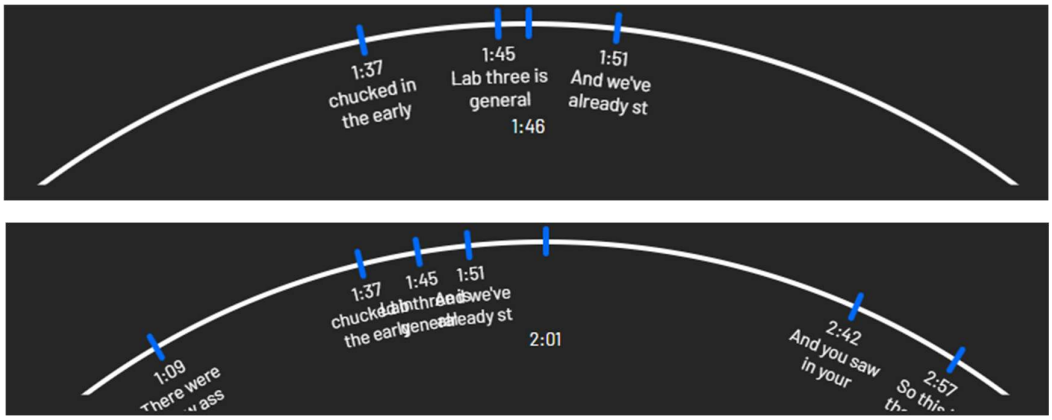
Navigation through the video can be performed by dragging the ArcSlider, when the mouse move event occurs the time moved through the video scales based on the currentTimeScale so that a given amount of mouse movement should always map to the same amount of rotation on the ArcSlider.

The dynamic zooming uses the svelte/motion tweened function to animate between timescales. The timescale zooms out to the next event if during a given section of video there are no upcoming events visible on the right-hand side of the ArcSlider. The timescale will also zoom in if any of the upcoming events end up overlapping / displaying on top of one another when they approach the play head.

Our ArcSlider element displaying relevant event timestamps within a section of video



The ArcSlider after the previous screenshot, zoomed in to display the denser upcoming events



The ArcSlider zooming out after the events around 1:50 to allow easier scrubbing to the 2:42 event

Conclusion

From my experience in this project, I learnt that designing dynamic animated user interaction that must respond to any data that could be thrown at it is really hard to do. Developing the ArcSliders' dynamic zooming was the most challenging part of this project, frequently resulting in situations where the slider would get caught in a feedback loop of zooming out and then zooming back in. I'm confident that I have stopped the feedback loop from occurring, but it is a known weakness to me that the user experience of the ArcSlider heavily depends on the arrangement of interest point captions within a video not ending up in a distribution that makes the ArcSlider very twitchy / undecisive about whether it wants to zoom in or out as the video plays past certain points.

Originally, I was unsure of the decision to develop the frontend in Svelte as I didn't have experience with any JavaScript frontend frameworks. I was convinced / reassured by Leos first early prototypes that were able to interact nicely with the flask backend that it would be alright. Over the course of the project working on the ArcSlider component in Svelte, assisting Leo with minor elements in the rest of the user interface design (Surprisingly ran into lots of contrast issues when we tried to display video controls and other elements on top of the test videos) and being forced to interact with react in my web development paper, I have surprisingly come to love svelte or at least understand why developers rank the development experience highly.

The development of the backend API in flask really isn't anything new or special. The backend API mainly exists to glue together the tasks of taking uploads, managing their state and processing them with the OCR and STT scripts before the everything is served back to the user. Leo helped with the backend endpoint design in the places where he needed endpoints to handle other parts of the frontend's interaction such as thumbnails, etc.

The ArcSlider is a great prototype, and if future work was to be done in this area I would recommend working on making the events that have occurred appear less cluttered when a zoom out occurs and developing a more robust method of determining the currentTimeScale.

References/Bibliography

The finished project code is on GitHub at michael-wazowski/ultimate-video-player - GitHub

ArcSlider

- spacex-launch-timeline.vercel.app
- HarishChaudhari/spacex-launch-timeline - Github
- <track>: The Embed Text Track element - HTML: HyperText Markup Language | MDN (mozilla.org)
- svelte/motion • Docs • Svelte

Backend API

- Flask-SocketIO — Flask-SocketIO documentation
- SQLite
- Quickstart — Flask Documentation (3.0.x) (palletsprojects.com)

Logbook

Research before group assignment

06/03

Have idea for the video player project during the tutorial, use the SpaceX launch timeline with events as the playback slider.

Screenshot of a SpaceX livestream

Could Hash and detect when screen significant change, show as events on the timeline. Name the events from the auto generated captions David thinks the player should have. Scale the timeline to a certain density of (Sentences? Events? Chapter Markers? Average volume over time change?). Would mouse snapping to the events be useful? Will the rotation speed need to ramp or would it be fine to just start and stop.

- How to get timestamp for each sentence? · openai/whisper · GitHub
- Cut a video without re-encoding with ffmpeg - Github Gist
- Cut video without re-encoding - r/ffmpeg

13/03

Discussing other project options in the tutorial

20/03

Found the example launch timeline demo that I remembered seeing, the demo is very limited as it is just trying to recreate SpaceX's on-air asset.

- spacex-launch-timeline.vercel.app
- HarishChaudhari/spacex-launch-timeline - Github

27/03

Project Voting Day, given 16pts to spend on desired project. Selected most of the points to go towards the Ultimate Video Player project.

Group Assignment

02/04

Assigned to the Ultimate Video Player group. Teammates apart from myself are Luke Fraser-Brown, Shreyaa Senthil Kumar, Jack Unsworth and Leo Van Der Merwe. The group project manager is Michael Peddie

06/04

Investigating methods that could potentially be used to make the timeline if we decide to build the project as a web application.

- Draggable HTML5 video timeline (codepen.io)
- Draggable | GSAP | Docs & Learning
- updating to GSAP 3 | GSAP | Docs & Learning
- Video timeline dragging and click - GSAP - GreenSock
- Drag and Rotate JS (codepen.io)

Timeline behaviour

- Would scrolling to zoom the scale of the timeline be useful?
- Probably not practical to have a view where the timeline is zoomed all the way out. The timeline is probably going to be optimized for smaller jumps between events within a section of the video, rather than a blind drop in. Will still have the normal video progress bar for coarse navigation.

10/04

Tutorial with David, discussed how video player is probably going to be a web app.

David suggested investigating some existing JavaScript video player libraries to see if they have anything to offer that would be useful for the project.

- https://github.com/videojs/video.js/wiki/Skins
- https://www.npmjs.com/search?q=keywords%3Avideo.js&page=1&perPage=20

David also suggested an idea for a more advanced form of the project where it could have an associated web extension that could replace the YouTube player, probably outside the scope of our project.

11/04

First group meeting

Luke, Jack, Micheal, and I were present.

Got consensus on the project being a web application. We are going to use a flask backend based off Shreyaa's findings that the more common recognition libraries are mainly in python.

Got assigned to work on the frontend and the API that will serve up the results of the processing done by other team members.

Discussed the video libraries mentioned by David in yesterday's tutorial, ultimately going to just use the native html5 <video> tag unless we run into limitations as it looks good enough and the libraries don't really offer anything of value for our project.

Going to focus on functionality before worrying about style.

Started work on making the basic flask webserver

- Got a static test index page hosted.
- Managed to get a file upload to work, has some issues with the browser page state once the file is uploaded.
- Python Flask Api Background Task - tiagohorta1995 could be useful once the other team members have their processing scripts working.

13/04

Leo is taking over most of the work for the general stuff in the frontend, based on his recommendation that we use some kind of JavaScript framework. He recommended the Svelte framework which has a demo of how to interact with the video element which I was impressed by, and he also has some experience with it.

I don't have any experience with JavaScript frameworks, but this is going to be a bigger project than anything I have done with just plain JS so I agreed that Svelte should be used.

The Flask Backend and the ArcSlider components are the main things that I will be working on

30/04

Reviewed the progress made by Leo over the mid-semester break.

01/05

Added the ArcSlider component that Leo had begun to the video player page.

First Tutorial after mid-semester break, showed Leo's frontend uploading a file to the Flask API and playing it back.

02/05

Merged Leos progress into the main branch as the primary frontend.

Figuring out how to get the text data from the backend, was considering having the generated text stored in the Flask SQLite database and passing it to the frontend in some kind of JSON data structure. Did some research into how the web handles captions natively and determined that the WebVTT (https://developer.mozilla.org/en-US/docs/Web/API/WebVTT_API) is going to be the better option to use to get the data to the frontend with the features that the <track> tag.

Made the link to get back to the video list so that a user isn't trapped once they load a video.

08/05

Serve the thumbnails Leo generated and the captions (Test files, not generated ones yet) from the API.

- /uploads/<id>/vtt
- /uploads/<id>/thumb

Started Figuring out how the timescale is going to work for the ArcSlider.

Have a currentTimeScale and a targetTimeScale, when an event reaches the end
- remove the event from the list
- add the next event from the subs not in the list
- set timescale
    end of timescale = new event from subs - current timestamp
    start of timescale = current - (end-current)

Animate currentTimeScale towards targetTimeScale by x amount of the difference

09/05

Switched the ArcSlider over to using angle instead of X,Y transform for the positioning of the chips on the slider.

- Made the rotators that the chips are in are invisible copies of the elipse-path
- Had an issue with clip-path preventing overflow and cutting off the top half of the chip
- Remove events when they go off the end of slider.
- Read some svelte documentation and figured out the tweened() for animating time scale changes, have a test button at the moment to trigger the animation

Issues with getting / reading the captions track as JavaScript stores the cues in an “Array-Like” object, not in an actual array.

10/05

Got it working to make chips from the marked captions (once I figured out it was cue.id not cue.ID), only when scrubbing forward though right now.

Ran into weird array slice issue when trying to pull the chips within the current timescale from the array of all chips.

Still need to calculate the timescale changes.

12/05

Implement some timescale change logic, had issues with the timescale fighting when a zoom out adds more events, which are removed due to being to dense, then after removing events timescale tries to zoom back out again. Hopefully current implementation shouldn’t jump around too much.

15/05

Pull /list when going back to menu so that the user sees the latest state of the API.

Fix video poster after upload bug where videos were showing the thumbnail of the previous video.

Comment and tidy up some of the arc slider logic.

22/05

Assist Jack and Luke with python filesystem interaction for the STTFunction opening video files, and writing text files to the right location

25/05

Get the threading doing work upon file uploaded.

Considering potentially using WebSocket or something else to notify the frontend when processing done.

- https://medium.com/@chandan-sharma/powering-flask-with-websockets-ca9f5a097ad9

29/05

Added controls to the video player to change the playback rate via a toggle button, allows for 0.25x to 2x in increments of 0.25.

Restructured the flexbox CSS being used for the overlaid video controls so that there is a div for the left and a div for the right that are space-between (also idk why all of those elements were set to display as flexbox inside of the flex? the browser said that it was an invalid property)

30/05

Added Web Sockets so that the frontend can be informed when the processing of a video in the backend finishes.

Helped Shreyaa with the OCRFunction so that it runs faster than real time by adding multithreading

03/06

Added the endpoint for the VTT files created by the OCR process and added a basic display of the OCR text below the caption text when in expanded mode.

Fix the ArcSlider to pull from VideoPlayers captions, so that the 100ms delay isn't needed and can instead process on load.

Fix contrast issues with the video controls, delete icon, and processing spinner not being visible against the video / thumbnails.

04/06

Added a filter search to the caption and OCR side boxes, modify the keyboard shortcut listener so it doesn't pick up search box input. Tweaking CSS Size and alignment