# Adam Atanas

## Video Manipulation

The video manipulation part of the project uses content scripts (Javascript that is injected into the actual webpage, rather than running inside the Chrome extension) to obtain and manipulate videos on the webpage.

One key aspect was site-to-site variation. YouTube has its own built-in timestamp mechanism (adding an &t=TIME into its URL), so the video manipulator code takes advantage of it. However, most other sites don’t offer such a convenient feature, so we had to implement a different set of code for them.

## Video Pauser

The video pauser (Get Video Time) works by finding a list of all video tags on the current webpage, then chooses the one that's currently playing (unless we're on YouTube, in which case it chooses the only available video). It then proceeds to pause that video and obtain its timestamp via the HTML5 API, and pop up a message with a URL to resume playback of that video. The URL will consist of the page URL, plus some extra variables. For YouTube, it adds an &t=TIME, where TIME is the time it obtained previously.

A major challenge we faced was attempting to somehow encode the timestamp of a non-YouTube video in such a way that playback could be re-enabled. The URL needed to be copy/paste-able into the notepad, as one of the main motivations of the project was to take notes and insert URLs into them that correspond to various positions in the lecture videos, to avoid having to fumble around with the controls to get the video at the right place (which is especially troublesome with longer videos). Thus, attempting to inject JavaScript that would somehow resume playback was impossible, since the user might access the URL from anywhere (not necessarily the notepad), and there might be many different websites with many different constantly-changing timestamps. What we eventually settled on was adding two extraneous variables (playbackTime and videoSource) to the GET request in the browser (which, being a substring of the URL, will be available to the Video Resumer method, described below). We chose variable names that we thought were unlikely to be used on real websites, so we thus minimized the chances of actually breaking the website. These two variables encoded values for the timestamp and the source of the video to be played back (which was necessary in case there were multiple videos on the webpage; we need to know which one to play back).

## Video Resumer

The video resumer (Resume Video) then resumes playback of the video at the timestamp encoded in the URL. YouTube automatically begins replaying the video at the correct time, but other websites won’t and the video resume is required to do so. Even so, video resume still works on YouTube, where it simply looks for the t= flag and resumes playback at that time. Originally, this wasn’t necessary, but if CS50 2X is run it can reset the timestamp, in which case being able to press Resume Video to get it back is very helpful. For other websites, it parses the URL generated by the video pauser (in particular, it gets the values for the two extra variables in the GET that video pauser inserted). It then gets the video to resume by searching through all video elements on the webpage, and getting the one whose source matches the videoSource variable specified in the GET request. It then sets the time of that video to playbackTime, the other variable specified in the GET request (which, remember, is a substring of the URL).

Note that this means the two programs only work if the video is in the form of an HTML5 video element. This is where CS50 2X comes in, as it can transform a variety of video players (in particular, JWPlayer, which is featured on isites) into FlowPlayer, which uses the HTML5 video tag and is hence accessible to the two functions above. These programs should thus work on most websites in which CS50 2X works, and will likely not work if CS50 2X doesn't.

A major challenge I faced were attempting to hook into the JWPlayer API. After long hours of fumbling around the API, I eventually realized it was not possible due to security practices. This is when I realized we needed CS50 2X in our project, since it can change JWPlayer into FlowPlayer, which my code can successfully integrate with. At first I tried cross-extension calling (i.e. calling CS50 2X from within Harvard Lecture Helper), but CS50 2X did not give me the permissions in its manifest.json file in order to do this. This led to the decision to bundle CS50 2X directly into Harvard Lecture Helper, which has since worked perfectly. A huge thanks to David Malan, who let us bundle the source code for CS50 2X into our extension. Without him, this project would be impossible.