**Appium – summery (twitch && Facebook)**

In short, it is a mobile test automation framework which supports both IOS and Android mobile web apps. Appium is a great choice for our research as it offers an “Inspector” which gives the ability to locate elements of an application and perform basic actions on them by querying elements by XPath, ID and more. In other words, we gain access to any elements that are “exposed” to the user (Examples: “buttons”, “texts”), something that enables building a complete automation for creating datasets.

We used Appium to extract video **Quality**, **Start time delay** and **Stalling** (Rebuffering events) from various OTT’s (Twitch, Facebook, YouTube, and Netflix). As every OTT has its own way of working, some adjustments had to be made, and thus every OTT got its own “yaml” configuration file to be automated separately.

**Automating Facebook:**

Facebook provides various video screen sizes, something which may be troublesome to automatedly “scroll and land on” (from regular feed) exactly the right frame of the picture that is needed to be seen properly for the automation to be able to work correctly. Thus, we used Facebooks built in option to create playlists to accurately extract the needed videos by choosing those with fitting sizes as well as the number of videos. As our “Samsung Note 8” screen fits 6 Facebook videos in one screen, each playlist had 12 videos, 6 at the start of the playlist and 6 more after a hard swipe to the bottom of the playlist to avoid inaccurate swipes with a bigger number of videos. As Facebook does not let the user know which video quality is experienced, we had to instruct the automation to choose a wanted quality at the start of each video, which implies that the videos were chosen not only by their frame size but also by fitting quality.

To obtain Rebuffering events times we used the “rotating stalling element” which can be seen on the screen and as we have control of all elements that are visible to us, they can be used for measuring appearing start and end times of such elements by logging. Same goes for Start time delays, at the start of each video the automation waits up to a fixed number of seconds (Example: 5) for loading the first picture on screen and waiting for the rebuffering sign that indicates whether a start time event occurred.

The entire workflow with Facebook was divided as follows:

1. video quality along with stalling.

2. Start time delay.

It was divided as such as we do not need to watch the entire videos only to compute the start time delays of each, whereas video quality and rebuffering events are extracted from whole videos where all video lengths are measured and are played up to a predefined length to avoid buffering another video at current videos time.

**Obstacles with automating Facebook:**

1. Screen sizes and hard to control swipes which were already discussed.

2. Upon finishing watching a video, the buffer stays at the same place previously left and thus every time that the same video is needed to be watched again using different qualities, the buffer needs to be reset back to start by the automation.

**Automating Twitch:**

Twitch is the world’s leading live streaming platform for gamers, it handles hundreds of thousands of views per day and provides new content hourly. It is an excellent source for video streaming analysis and is a lot easier to automate than Facebook.

Similarly to Facebook, the workflow of automating twitch videos also divides to gathering video quality along with stalling and start time delays. Twitch has a vast variety of videos and all of them share the same window size, thus all that is needed is to query some content name and go through the non-live videos from the given output. Twitch, like Facebook does not let the user know which quality is being experienced but the wanted quality can be chosen if such exists per video.

For the quality-rebuffering part, the automation receives the wanted quality (Examples: 360p, 480p, 720p, 1080p) and jumps from video to video until some video meets the required quality, once a video is chosen with the correct quality value from the video settings, a start time is designated and the total watch time of the video is saved until it reaches a fixed length of the video, in our case we chose to watch 60% of each video (in order to not have the possibility to buffer the next video along with the current one) and afterwards exit the current video, designate the end time and search for the next match. the whole automation will work and gather such data for some fixed total of good-matched videos time, we chose it to be 1 hour every run to not deal with too big files, which helps managing the dataset.

For the start time part, as there are many videos to check, the automation will run for a fixed time number, click on a video, check if a start time delay occurred and measure it same as in Facebook by the “rotating stalling elements” length, exit the video and go on until the fixed time is up.

**Obstacles with automating twitch:**

Almost all queries take more time to process than the application provides in order to use trivially wanted functions (Example: new screen options that can be used by tapping the screen disappear after 2 seconds where the query takes 4 to process). the solution was to find other ways to access the wanted data by finding and using other approaches as the trivially once take more time. (Example: clicking a needed button takes at least 4 seconds, but clicking the screen at a defined point designated by an (x,y) value takes significantly less time and does the trick, of course such a workaround works after researching the patterns of the application and knowing where such buttons appear)