Die Probleme

Die Hauptprobleme des Projektes erschlossen sich, als wir versucht haben uns mit Moodle zu verbinden. Im Normalfall würde man für eine solche Aufgabe immer die API, also die Schnittstelle der Plattform nutzen, mit der man interagieren möchte. Moodle bietet eine sehr umfangreiche und mächtige API, da diese von Seiten der DHBW aber deaktiviert ist und wir in naher Zukunft wohl keinen API-Zugang erhalten, müssen wir einen anderen Ansatz verfolgen. Der Steakholder stellte sich das ganze über Browser-Automatisierung vor. Diese Möglichkeit automatisiert Abläufe zu erstellen, welche einen Nutzer simulieren, ist schon etwas älter und gilt als ziemlich fehleranfällig. Auch wenn diese Möglichkeit von “Best Practice“ weiter nicht entfernt sein könnte, evaluierten wir aus Mangel an Alternativen unsere Möglichkeiten.

Ansatz 1: Nightmare

Nightmare ist eine Bibliothek für high-level Browserautomatisierung. Dabei bleibt Nightmare relativ klein und erreicht somit eine vergleichsweise hohe Performance.

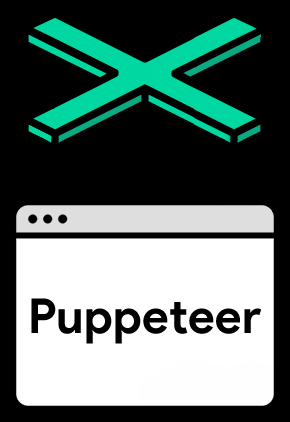
Die Methoden von Nightmare sind relativ simpel gehalten und grundlegend würde Nightmare alles bieten, um das Projekt umzusetzen.

Das Problem bei Nightmare.js ist aber das Nightmare.js selbst auf Electron basiert, deswegen ist es leider nicht möglich NightmareJS aus unserem Programm heraus zu nutzen.

Ansatz 2: PhantomJS

PhantomJS ist ein WebKit, welches Seitenautomatisierung auf den bekanntesten Systemen mittels JavaScript bietet, PhantomJS ist hierfür eine der bekanntesten Lösungen und wurde zumeist im Bereich des Testings benutzt.

Zwar ist PhantomJS etwas langsamer als NightmareJS, würde aber unserer Anforderung theoretisch nachkommen. Aber auch nach massiven Bemühungen ist es mir nicht gelungen PhantomJS in unserem Programm lauffähig zu halten, ein weiteres Problem stellt das Bundeln des Projektes dar, welches mit PhantomJS (nach meinem Wissensstand) nicht möglich ist. Des Weiteren wird PhantomJS seit einigen Jahren nicht mehr weiterentwickelt, dies ist also ein Pferd auf dass, sollte man auf Langlebigkeit/Nachhaltigkeit wert legt, nicht unbedingt setzen sollte.

Ansatz 3: Puppeteer

Puppeteer ist eine weitere Node-libary, welche eine high-level API für Chromium liefert. Mit der Standardeinstellung im Headlessbetrieb würde auch Puppeteer den gewünschten Parametern entsprechen.

Leider verliefen auch Tests mit Puppeteer erfolglos. Die Automatisierung scheiterte mit Abstürzen der Software, genauere Analysen ergaben hier keinen sicheren Grund, es könnte aber mit dem gemeinsamen Laufen verschiedener Chromeinstanzen zusammenhängen. Zudem besteht wie auch schon bei PhantomJS, bei Puppeteer das Problem des Bundelns mit Electron.

Ansatz 4: Pakete

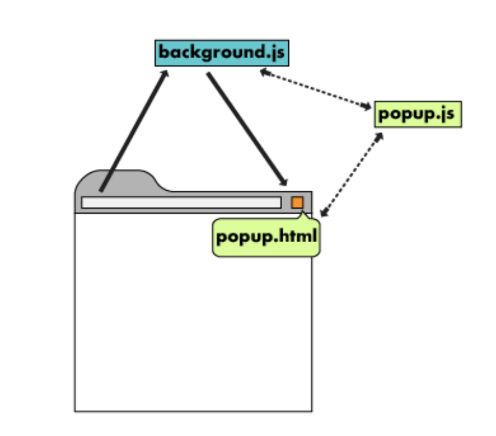
Im Gegensatz zu einer Lösung, welche das direkte Userverhalten durch Klicken von Elementen auf der Betreffenden Moodleseite nacharmen würde, versimpeln wir mit diesem Ansatz den Aufbau.

Statt dem Prinzip der Browserautomatisierung mittels eines Headlessbrowsers zu folgen, versenden und empfangen wir bei diesem Ansatz nur die Pakete. Dies bietet den Vorteil, dass wir keine zusätzliche Browser Instanz brauchen würden und nimmt dem Ganzen so einiges an Komplexität.

Um hier die Machbarkeit zu überprüfen, haben wir die Pakete bei der Moodleanmeldung und dem Ausfüllprozess abgefangen, diese analysiert und haben uns durch das manuelle Verschicken eigener Pakete, mittels Insomnia (ähnlich Postman), immer weiter vorangetastet. Dieses Vorhaben haben wir allerdings abgebrochen, als der Aufbau durch mehrere Single Sign-on-verfahren und Session-token-verfahren im Ausfüllprozess derart Komplex wurde, dass auch dieser Ansatz an Sinnhaftigkeit verlor. Die Server, auf die wir mittels der Pakete Zugriff erlangen wollten, scheinen sich mit einem Server oder mehreren Servern für die Logins und Rechte Freigabe zu verbinden. Diese Auslagerung der Single Sign-on’s (SSO) setzt so viele verschiedene Session bzw. Authentifikations Cookies, dass der Versuch nach Überwinden des Logins abgebrochen wurde.

Unser Gegenvorschlag

Nachdem wir mittels der benannten Verfahren nicht in der Lage waren, die Vorgaben des Steakholders zu erfüllen, haben wir einen Gegenvorschlag ausgearbeitet. Dieser sah vor, eine Browsererweiterung zu programmieren, welche den gesamten Prozess übernehmen würde.

Es wäre für eine Browsererweiterung durch Nutzung der Browser APIs (z.B. Chrome APIs) möglich, die betreffenden Skripte direkt in Moodle zu injizieren. Dies würde die Notwendigkeit einer weiteren Browserinstanz gänzlich irrelevant machen. Der bis zu diesem Zeitpunkt geschriebene Code hätte im Falle einer Browsererweiterung auch zu großen Teilen weiterverwendet werden können. So hätte der Part für das Einlesen der Tabellen, sowie das Zuordnen von Namen ohne große Anpassung übernommen werden können. Bei der Präsentation haben wir weitergehend auch die anderen Vorteile, die wir in einer Lösung mit Browsererweiterung gesehen haben, hervorgehoben.

Die Vorteile einer Browsererweiterung wären nicht nur eine extreme Kompaktheit und dass sie mit jedem System mit einem unterstützenden Browser lauffähig wären. Sondern auch noch eine wesentlich erhöhte Nutzerfreundlichkeit. Während der ursprüngliche Aufbau noch ein zusätzliches Einloggen in Moodle, um an den Moodle-Link zu kommen vorgesehen hat, der Nutzer sich also 2-Mal hätte Einloggen müssen. So wäre eine Browsererweiterung nur aktiv geworden, wenn der Nutzer sich an einer Stelle in Moodle befunden hätte, an der die Applikation eingesetzt werden könnte, anschließend hätte der Nutzer seine Tabelle auswählen können und die Erweiterung wäre im Aktiven Tab vor den Augen des Nutzers aktiv geworden und hätte die Daten eingetragen.

Diese Lösung hätte unsere aktiven Probleme mit der Einbindung von Moodle auf die eleganteste Möglichkeit bei den gegebenen Umständen umgangen und hätte uns nach eigener Einschätzung ca. drei Tage gekostet. Der Schwachpunkt bei diesem Ansatz lag allerdings bei der Anforderung des Steakholders welche eine Desktopanwendung forderte, da der Steakholder nicht bereit war, diese Anforderung fallen zu lassen, konnten wir diesen Lösungsansatz nicht weiterverfolgen.

Quellen:

<https://de.wikipedia.org/wiki/Single_Sign-on>

<https://www.security-insider.de/was-ist-single-sign-on-sso-a-631479/>

<https://stackoverflow.com/questions/17769011/how-does-cookie-based-authentication-work>

<https://auth0.com/docs/login/spa/authenticate-with-cookies>

<https://phantomjs.org/>

<https://en.wikipedia.org/wiki/PhantomJS>

<https://github.com/ariya/phantomjs>

<https://www.linode.com/docs/development/nodejs/use-nightmarejs-to-automate-headless-browsing/>

<https://github.com/segmentio/nightmare>

<http://www.nightmarejs.org/>

<https://github.com/puppeteer/puppeteer>

<https://developers.google.com/web/tools/puppeteer>

<https://www.npmjs.com/package/puppeteer>

<https://medium.com/devnetwork/how-i-wrote-a-single-page-chrome-extension-2590fef14044>

<https://developer.chrome.com/apps/api_index>

<https://www.chromium.org/developers/how-tos/api-keys>

<https://www.chromium.org/developers/design-documents/desktop-notifications/api-specification>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>

Bildernachweis:

<https://user-images.githubusercontent.com/10379601/29446482-04f7036a-841f-11e7-9872-91d1fc2ea683.png>

<https://www.google.com/url?sa=i&url=https%3A%2F%2Fmedium.com%2Fdevnetwork%2Fhow-i-wrote-a-single-page-chrome-extension-2590fef14044&psig=AOvVaw2qhlD8ErbOJThr_M3gQbP-&ust=1589468004669000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCJDd9vGLsekCFQAAAAAdAAAAABAJ>