COS 333 Final Project: Parsnip

Shuyang Li, Timothy Seah, James Wang, David Zhao

Introduction

Our project began as a side thought to our original idea of making a social media app intended to connect students with similar interests. Why pursue macros and subject ourselves to web browsers and networks, we probably asked ourselves, when we can make a perfectly good social media app. Somehow, the idea seemed attractive and we embraced it in a reactionary sort of way in response to all the hype over social media apps in this day and age.

As it turns out, writing a standalone JavaScript program that is guaranteed to interact well with the myriad of websites out there is a rather unsavory task. Because many modern websites are fairly sophisticated in their design, creating a tool that could be robust to a seemingly endless variety of special and often dynamically generated webpages felt a little like continually screwing on attachments to an increasingly unwieldy Swiss Army Knife.

If it is any indication of the amount of problems and setbacks we ran into, compare and contrast the features touted by our original and current elevator pitches:

<u>Elevator Pitch I</u>

There are many tasks on the web that are repetitive, tedious, or otherwise lend themselves well to automation. Things like price monitoring or class registration are especially good candidates for this. Suppose that one is looking to purchase a plane ticket for a journey that is months away. Given the capricious nature of airfare rates, the frugal flyer is well-served by

monitoring price changes for a prospective flight for at least a few weeks. Ordinarily, one would have to do this by manually visiting a URL multiple times a day, every day, resulting in no small amount of impending anxiety and sleeplessness. This is a decidedly unpleasant experience. We want to offer a better way. Our project is Parsnip, a Chrome browser extension that allows a user to easily record, store, and execute macros to automate tedious online tasks, no coding required. Upon execution, these macros will automatically open a new browser window and execute in sequence all recorded mouseclicks, entered text, and other actions, no human input required. And there's more—these macros may be executed either on a schedule or conditional on some value element on a webpage, whether that be a price change or the appearance of a new HTML element. Just record, tweak, save, and relax. With our tool, one need not repeat the same unpleasant online task twice ever again.

Elevator Pitch II

Have you ever wished that you could automate clicking items and filling text fields on webpages? Well, now you can! Parsnip is a free Chrome extension that allows one to record, store, and execute macros with no frills, no coding required. It's simple—just enter Parsnip's recording mode, click on webpages or enter text as usual, and Parsnip generates JavaScript code corresponding to those browser actions in real-time for later use. More robust to a variety of webpages than other free existing alternatives, Parsnip has been proven to work on sites like Facebook and YouTube. Parsnip is suitable for work, entertainment, or more. Available on the Chrome Web Store.

Design

Interfaces

Surprisingly little of the initial design logic changed as we transitioned from whiteboarding to coding. Nearly all of the original features we had planned—the pop-out extension window, the pane to view the text of the generated macro, saving and loading via a back-end server—remained the same from start to finish. Compared to the interface of iMacros, a free Chrome extension found on the webstore, we wanted to make ours a cleaner, more stripped-down version with equivalent if not greater functionality.

Frontend

Our first major decision was over whether to build a web app or a Chrome extension. Initially we thought learning Django would be a good idea in order to build a web app to support the server save and load functionality of the extension. Instead, we chose to build a Chrome extension rather than a web app. A Chrome extension has access to every tab on the user's browser, while a web app does not. In the early stages of our project we considered using iFrames to overcome this limitation, but iFrames have unreliable behavior with many websites.

We opted for Chrome in particular because it the browser that each of us happens to use, because Chrome has respectable market share behind IE, and because Chrome users are generally regarded as being more savvy than the average web user about customizing their browsers in the form of skins and extensions. As such, there was no option not to write JavaScript. The Parsnip pop-out window was written in HTML/CSS.

Recording macros on different types of websites was tricky because websites frequently alter their DOMs in unpredictable ways. In order to fix this problem, we wrote a function that identifies elements based on their attributes that weights each attribute relative to other attributes according to its presumed importance in determining the identity

of the element in question. Thanks to this function, our app is able to record macros more reliably than iMacros in some domains, notably Facebook.

In terms of the 3-tier system requirement for all 333 projects, we eventually decided to fulfill the 3 levels of technology all on the local side, in addition to some back-end server fulfillment as well. In terms of the front-end, The **user interface** is done in HTML/CSS, the **process** is implemented by Javascript program code, and finally **data management** is done locally with the chrome.storage API.

Back-end

We faced more options making back-end design decisions. Initially our first proposed use case for the back-end was to implement functionality equal to the front-end's ability to execute macros, so that the user could store macros for our server to execute some arbitrary time later. Secondly, we wanted the back-end to store and serve macros back to the Chrome extension for cloud-based storage. We assigned half of our team to front-end and half of our team to back-end at the beginning to work towards these goals.

What unfolded for our project was that the back-end developers spent their initial man-hours setting up, installing, and configuring the tools required to run parallel logic to the chrome extension, including technologies like Amazon Web Services EC2, Django, SQLite3, and assorted Python libraries. However, by the time we had a minimum viable product on the front-end, we realized that parallel execution on the back-end wouldn't be a feasible end-goal. In its current state, the back-end utilizes Scapy, a python library for packet sniffing, and SQLite3 on an EC2 instance to store whole macros associated with IP addresses and macro names sent from the Chrome extension.

Building

In retrospect, too much time was spent installing and configuring an assortment of back-end technology that didn't end up panning out into tangible functionality. This time could've been better spent in the Chrome extension web element selection logic, a module of our program that ended up being the most complex and time consuming. In addition, we should have spent more time exploring possible front-end libraries that could have better handled web element selection, such as JQuery.

In terms of the server, too much time was also spent on lower level installation and configuration of technologies we didn't end up needing. In the future, AWS EC2 should be the less preferred option to higher level web services technologies such as Heroku or Google App Engine. Finally, significant siloing of technical expertise happened early on in the development process that should've been minimized. "Back-end guys" remained "back-end guys", even when it turned out that the front-end Chrome extension process logic turned out to be much more involved than what was initially expected.

Testing

We tested the app by brainstorming a list of progressively more challenging browser actions for the extension to perform, distributing this list to each member of our group, and having each member independently test each item on the list. Each of us performed a sequence of actions on the app (e.g. open app, hit record, open web page, hit stop, hit save, hit delete, hit load), observing the app's behavior, comparing it against what we expected, and then recording our findings on a shared Google Doc. We did not automate testing because our concerns with the functionality of the app centered around simple exception cases rather than volume of actions.

Certain unanticipated and unsolved problems have cropped up: for example, although Parsnip is capable of clicking through links on Facebook pages and even posting on people's Walls, Facebook's bot detection software often detects the automated behavior and summarily logs the user out. This occurs even with a short delay coded in between actions.

Reflection

Things we would have done differently

Given the chance to start COS 333 over, we would probably not choose this project idea again. We agree that too much of our time was spent trying to solve cross-platform compatibility issues and getting stuck on trivialities of the language. There is no satisfaction in fighting technology. We'd rather work with it than against it. As a result, debugging was a far less gratifying process than it usually is when the fix is an edit to the program logic. We imagine that spending the same number of hours working on a project with a more straightforward technology stack would yield a much better progress-to-working hours ratio.

Additionally, we would assign "homework" for each of us to do independently. In our experience, it's better to avoid situations where two people are working on the same code at the same time. Splitting work up is a better system than trying to make progress on the same thing, which tends to be redundant and creates inconsistencies in the working product.

In terms of the logistics of getting four guys to code together on a regular schedule, we lucked out in that 3 out of the 4 of us are roommates, and the fourth lives fairly close by. Hack sessions consisted of the four of us working in someone's bedroom, on chairs and beds and pillows. Though we often stayed up long enough to hear birds chirping, we never pulled all-nighters for the sake of it. What we would gain in progress from not sleeping we

would lose in productivity from sleep deprivation the next day. We are probably all getting old.