HFetch: A User-Friendly API Client for the Schoology LMS

Nathaniel Waterman

March 3, 2018

Abstract

Schoology is popular and powerful, but it takes time to check for homework posted there. This was the reason for designing HFetch, which uses a RESTful API to fetch, display, and provide details about assignments from Schoology in 2 clicks, or 1 if it is launched automatically.

The introduction to Schoology's API documentation mentions that there were no Schoology API clients at the time it was written, but an implementation in Java was found on GitHub (rvanasa/schoology-api). This paper intends to inform the reader about HFetch, seemingly the second such API client.

Contents

Chapter 1

Introduction

Schoology is a learning management system (LMS), a website on which teachers can post assignments and resources for their students to access anywhere.[4] Navigating to Schoology is inconvenient, and once on the site, all that is usually needed is the sidebar, which lists upcoming assignments. It can be hard to remember to check Schoology, so an automated utility is extremely useful. Because of this, a Unix shell script[1] was written which fetches assignments from the website, and an AppleScript App which displays the assignment information in a user-friendly GUI. Bash and AppleScript, a MacOS scripting language, were chosen for this task because they integrate well and both run on MacOS, as the school that HFetch is intended for has an abundance of Macs.

Schoology has a RESTful API in order to allow for the creation of tools to enrich the Schoology experience.[5] REST is a standard for web services which allows for the access and manipulation of web resources by way of a set of stateless, (each message is independent of the others) predefined operations.[3]

Chapter 2

Problem

HFetch was designed to solve the problem of students forgetting to check Schoology for their assignments. This ruins one of the main benefits of a learning management system: Teachers are able to post homework without explicitly telling the class, and can expect each student to complete the assignment. Failure to complete homework, in this case due to forgetfulness, can sabotage learning and ruin grades.

Chapter 3

Solution

HFetch is the solution to the aforementioned problem of forgotten homework. It runs a shell script which downloads course information, then uses this to determine the IDs of the courses and goes on to query the server as to the assignments in each of these courses. Once the assignments have been obtained, the script passes them on to HFetch to be displayed and exits. If, however jq[2], a JSON parser, isn't installed or the files storing the user's API credentials are missing, the shell script returns an error which HFetch identifies and leads the user through the installation of jq and possibly the package manager Homebrew, or the obtaining and entry of their credentials, passing these to the shell script, which saves them on the disk.

3.1 System Model

3.1.1 AppleScript App

The AppleScript frontend requires MacOS to run.

3.1.2 Shell Script

The Bash Script must have a Unix-like environment with bash and jq installed, or an internet connection and package manager with which to download these dependencies.

Chapter 4

Technical Challenges

A few lessons were learned while designing HFetch:

When testing the shell script, it was discovered that requests to the server must be sent with increments of at least 1 second, as the PHP time() function returns the time in seconds, not milliseconds. Also, OAuth was found to be uncompromising in security - when the API credentials on disk became corrupted (somehow), jq[2] failed to parse the server's responses due to them being error messages instead of JSON. It took much troubleshooting to find that the root of the problem was not jq at all, but a credential issue. The shell script uses jq to parse the JSON from the API, but since jq is not installed by default on MacOS, it had to be installed during setup. However, as MacOS does not have a package manager, the package manager Homebrew had to be installed in order to install jq!

Chapter 5

Conclusions

These programs make homework easier to remember, and allow for automation, such as with the cron utility[7] or Automator[6] on MacOS. With cron, for instance, the app's executable could be scheduled to run at certain times and remind the student of their upcoming assignments. Furthermore, with Automator, HFetch could be run with a keystroke or as a dictation command (by speaking).

In the future, a user interface should be created for Linux as the current one is written in AppleScript, new versions of this program should be created for Windows and iOS, and a similar client for the Powerschool API should be created or added as a part of HFetch.

Bibliography

[1] *c47s/getassgns*. URL: https://github.com/c47s/getassgns/.

[2] *jq*. URL: https://stedolan.github.io/jq/.

[3] *Relationship to the World Wide Web and REST Architectures*. URL: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest.

[4] *Schoology*. URL: https://www.schoology.com/.

[5] *Schoology API Documentation*. URL: https://developers.schoology.com/api-documentation/design-architecture.

[6] *Welcome to Automator*. URL: https://support.apple.com/guide/automator/welcome/mac.

[7] *What are cron and crontab, and how do I use them?* URL: https://kb.iu.edu/d/afiz.

Appendices

Appendix A

Shell Script (getassgns)

```
#!/bin/bash

interactive=1
justDoCredentials=0
tty=$(tty) # tty to direct interactive output to

while getopts "gret:" opt; do
        case "$opt" in
                g) # GUI mode - non-interactive
                interactive=0
                tty="/dev/null"
                ;;
                r) # Delete the API credential files
                rm "$HOME/.scgyKey"
                rm "$HOME/.scgySecret"
                ;;
                e) # Exit after modifying credentials
                justDoCredentials=1
                ;;
                t) # Specify tty for interactive output
                tty="$OPTARG"
                ;;
                *) # Invalid option
                echo "usage: $0 [-gre] [-t tty]"
                ;;
        esac
done
```

```
# Make sure we can find jq if it is installed
PATH="$PATH":/usr/local/bin/
jq --help > /dev/null 2> /dev/null || { # Test if jq is installed
        echo "Coulndn't find jq." > /dev/stderr
        if [ $interactive -eq 1 ]; then
                echo "Install it?"
                echo -n "(yes or no?) "
                read -n 1 answer
                read

                if [ "$answer" != "y" ]; then
                        exit 5 # 5 - User decided not to install jq
                fi
                # Try every possible way to install jq
                brew install jq || \
                pkg install jq || \
                sudo bash -c '
                        apt-get install jq ||
                        dnf install jq ||
                        zypper install jq ||
                        pacman -Sy jq'
        else
                exit 4 # 4 - The GUI needs to install jq
        fi
}



# Try to read the credentials. If they are missing, prompt the user.
# If the user doesn't respond (or we're in GUI mode),
# exit with an error - 2 if failed while trying to read Consumer Key,
# 3 if failed while trying to read Consumer Secret.

read consumerKey < "$HOME/.scgyKey" || {
        echo -n "Consumer Key: " > "$tty"
        read -t 5 && echo "$REPLY" > "$HOME/.scgyKey" || exit 2
        # 2 - Missing Consumer Key
}
```

```
read consumerSecret < "$HOME/.scgySecret" || {
        echo -n "Consumer Secret: " > "$tty"
        read -t 5 && echo "$REPLY" > "$HOME/.scgySecret" || exit 2
        # 2 - Missing Consumer Secret
}

[ $justDoCredentials -eq 1 ] && exit
# Stop if option -e (exit after modifying credentials) was specified



IDs=( $(
curl https://api.schoology.com/v1/users/25493325/sections -sH "$(php -
 | jq -r .section[].id
) ) # Obtain the IDs of the user's classes

length=${#IDs[@]} # Find how many classes the user has

[ $interactive -eq 1 ] && tput sc > "$tty"
# If we're in interactive mode, save the current cursor position so
# we can return here in order to overwrite the previous progress perce

result=$(

for i in "${!IDs[@]}"; do

        [ $interactive -eq 1 ] && tput rc > "$tty"
        echo -n $((i * 100 / length))"%" > "$tty"
        # Display progress percentage

        # Get each class' assignments, try again if this fails
        until curl https://api.schoology.com/v1/sections/"${IDs[$i]}"/
        do
                sleep 0.1
        done

done

)
```

```
if [ $interactive -eq 1 ]; then

        tput rc > "$tty"

        {
        echo "Name\`Due Date"

        # Parse the json returned by the server for the due date and
        # name, keeping only ones that are due in the future
        echo "$result" | jq -r '.assignment[] |
        select(
        (.due | strptime("%Y-%m-%d %H:%M:%S") | mktime)?
         >
        ($date | strptime("%Y-%m-%d %H:%M:%S") | mktime)
        )
        |
        "\(.title):`\(.due)"' \
        --arg date "$(date +"%C%y-%m-$(($(date +%d))) %H:%M:%S")"
        } | column -ts "\`" > "$tty"

else

        # Parse the json returned by the server for the due date and
        # name, keeping only ones that are due in the future
        result=$(
        echo "$result" | jq -r '.assignment[] |
        select(
        (.due | strptime("%Y-%m-%d %H:%M:%S") | mktime)?
         >
        ($date | strptime("%Y-%m-%d %H:%M:%S") | mktime)
        )
        |
        "Due: \(.due):: \"\(.title)\": due: \(.due), URL: \(.web_url)"' \
        --arg date "$(date +"%C%y-%m-$(($(date +%d))) %H:%M:%S")" | \
        sort
        )
```

```
        # Sort the assignments by due date
        tempIFS="$IFS"
        IFS='
'

        for line in $result; do
                echo "${line#*:: *}"
        done

        IFS="$tempIFS"

fi
```