Alex Meier
Software Engineering II
Project Proposal
31 January 2020

Macro Buddy

In software engineering, it is a regular occurrence to have to follow some command line procedure to perform builds, move files, copy logs, and other stray tasks that don't take up much time on their own, but after repeating them dozens if not hundreds of times a week can be a sizeable time sink. The obvious solution to this is to try to automate these small tasks using scripts or other means. The problem with this is twofold. On one hand, writing a script can often end up being much more trouble than it's worth, with the time saved by the automation being dwarfed by the hours you spent writing and debugging a shell script instead of working on other tasks. Secondly, if you write a lot of these automation scripts, they tend to get spread around your file system making you take the extra time to first find the script you want to run before you run it. My solution to this problem is Macro Buddy, a GUI driven macro and script recorder.

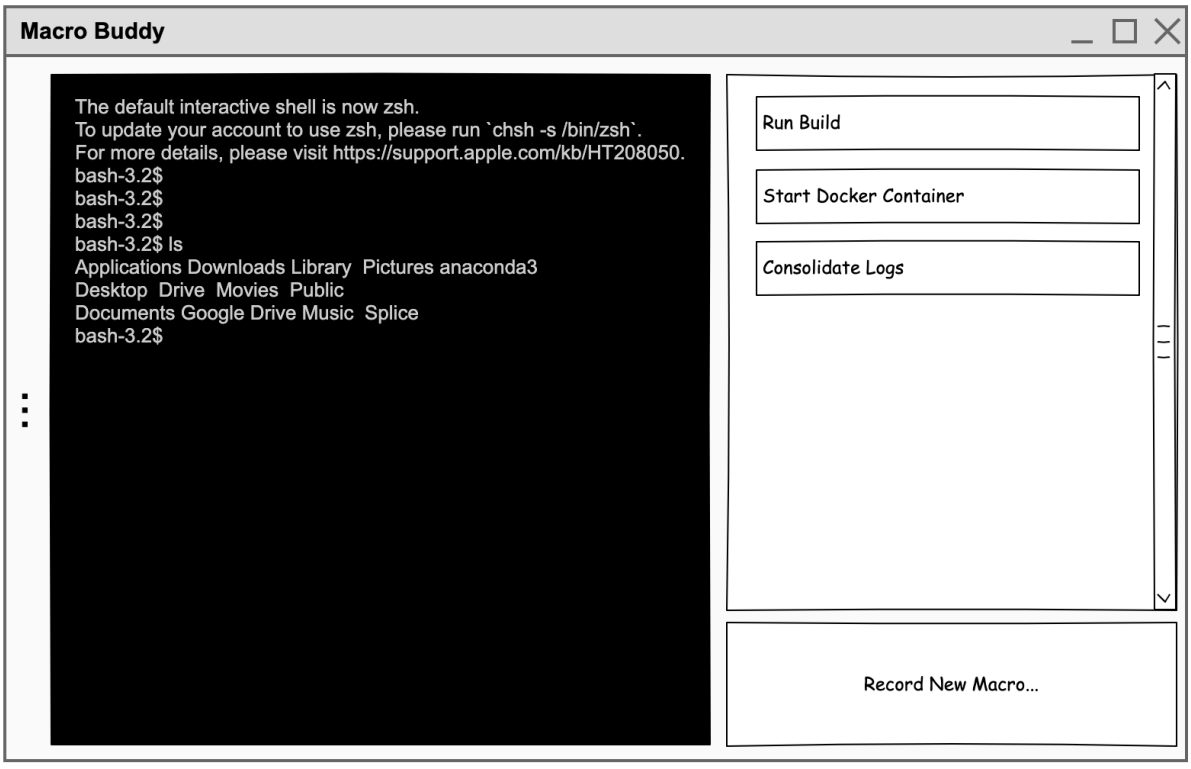

Figure 1:GUI Layout Prototype

Macro buddy allows the user to easily and quickly record a terminal session and generate a script that repeats the process at the click of a button. First, the user is able to press a record button and use the in-app terminal window to type in a sequence of commands. Once the procedure is complete, the user can press "stop", at which point the application scans through the keyboard inputs, and generates a matching script file, and allows you to name the procedure. The app then adds the procedure to a grid of macro buttons, which allow you to

activate previously recorded scripts with a single click. The intended use for this feature is for the user to leave Macro Buddy running on their desktop, so that they have quick access to their automations.

The general approach for building this application is to first create a stripped-down terminal emulator (simply serving as a front end for the user's command interpreter of choice). Recording the macros will be a matter of writing the input stream into a template file that contains some basic setup for whichever scripting language the user requires. The resulting script will be placed in a folder that the user choses. The working directory of the script will be determined based on context when the user initially records the script.

Some pitfalls for this project could be dealing with some common terminal features such as tab completion and pressing "up arrow" to pull a command from recent history. These actions could introduce unintended characters and other unexpected results. The solution to this problem would be either to write an algorithm to clean up the script, to allow the user to edit the script themselves to clean up any issues, or to simply disable these features to prevent the user from using them all together.

The minimal requirements for this project to be considered a success would be a simple interface that allows the user to type in commands, and for the app to be able to generate a runnable script to the user's specification. Some stretch goals for this project could be to allow the user to configure what command interpreter the app uses (BASH, C Shell, PowerShell, etc.). Additionally, an interface could be created to allow the user to make limited use of variables in the scripts, for example if the user wants to copy a set of log files to a zip archive and be able to set the name of the archive as part of the automation.