Team 9

Members: Jesse DeBok, Caden LeCluyse, Jack Ford, Ben Schulte, Harry Ahrenholtz

Sprint 1 Requirements Artifacts

Collection of Features for Research how Modern Password Managers work (ID: 1): (Jesse)

- How do password managers store passwords on the computer? Most password managers store a file on your computer that they encrypt, and then decrypt when the user logs in. They also can store them on the cloud so users can access them from any device, however I think for ours we should store it on the device in the application folder, encrypted.
- Do they require the user to login before using the password manager? Most do, however some give the option for no log in. Those that do require login are usually cloud-based and require 2 factor authentication to use on any system, so for ours I think we can do fine with just 1 factor credentials.
- Are the applications self contained as a single .exe or are they a folder with information inside of the folder? Most of them work by downloading an executable which is an installer. The installer runs and creates a folder within the program files folder on windows, which contains all of the needed information for the application. The installer also creates the desktop icon which links to an executable. Since we are using python this would link to a file which calls the python function. Our installer should make sure python is installed on the system, or include a copy in the program directory (this is a good option). We could also bundle the program and the python runtime into a single file.
- How are the most popular password managers designed and developed? Many are open source, and require a license to be used professionally but are free for personal use and small amounts of saved password.

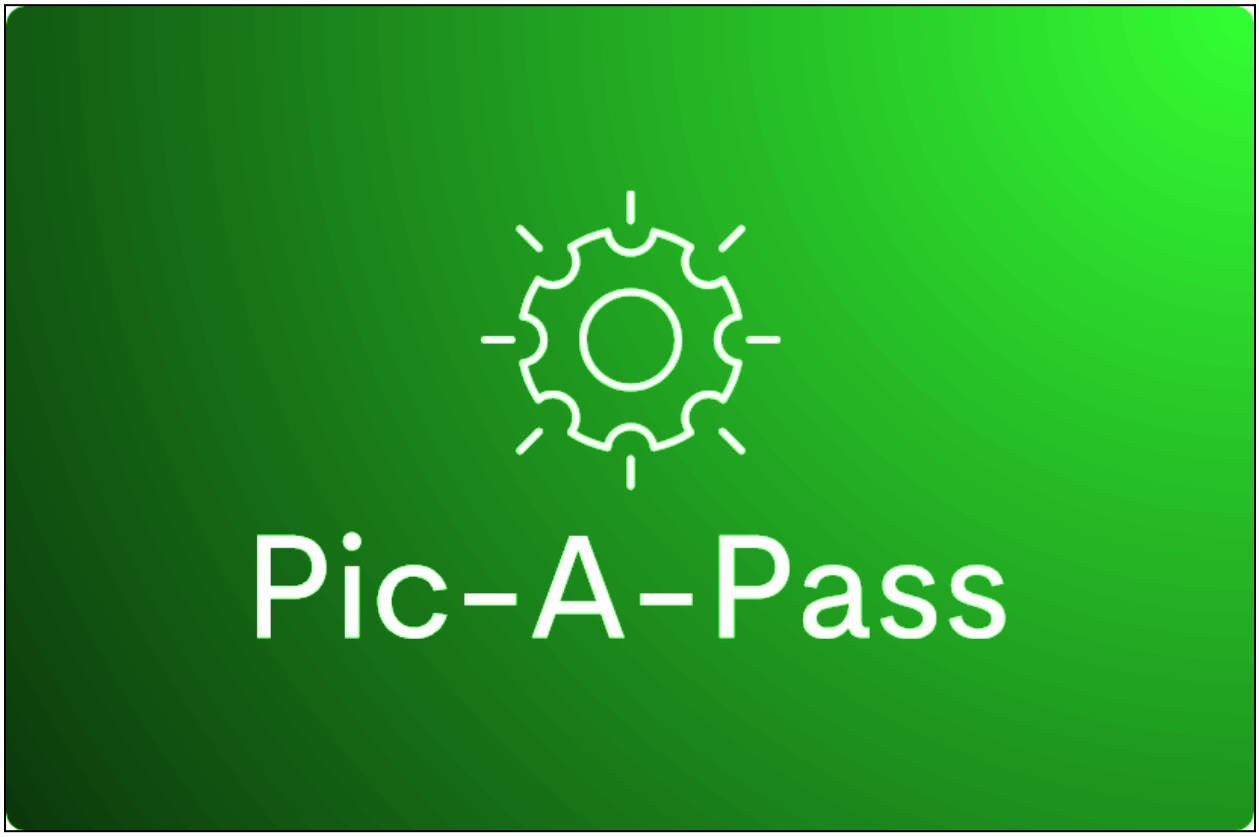
Collection of Features for Initial GUI Design (ID: 7):

VOCAB:

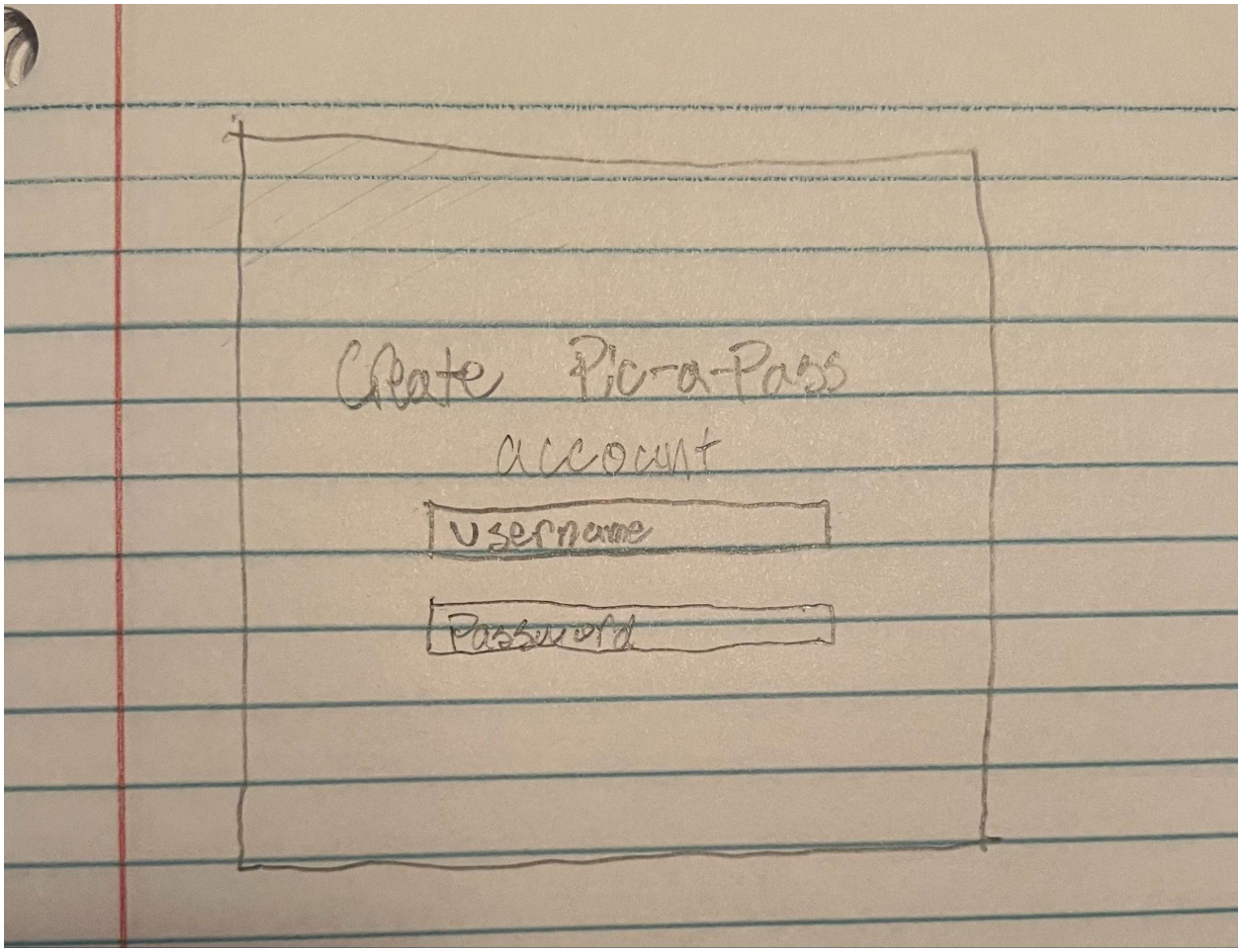
credentials = login info

password profile = set of URL, username, password

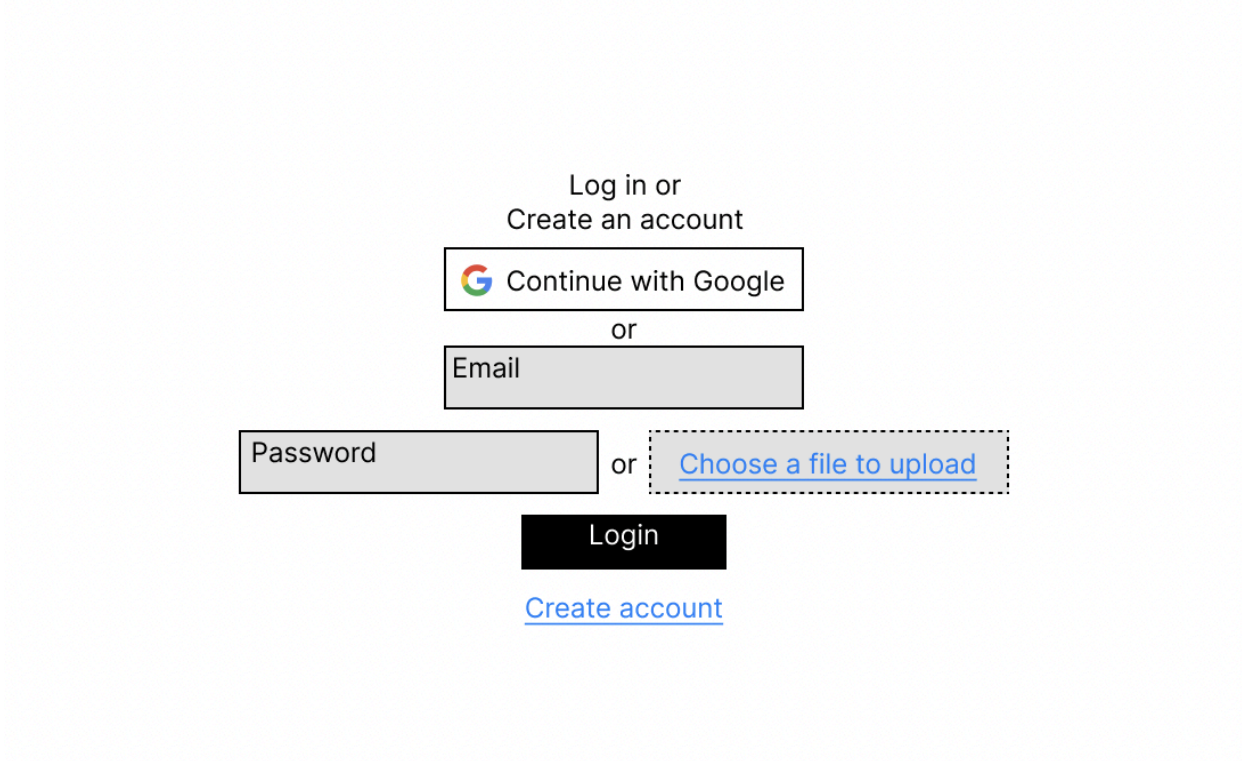
Design the screen when the application opens up (Jesse)



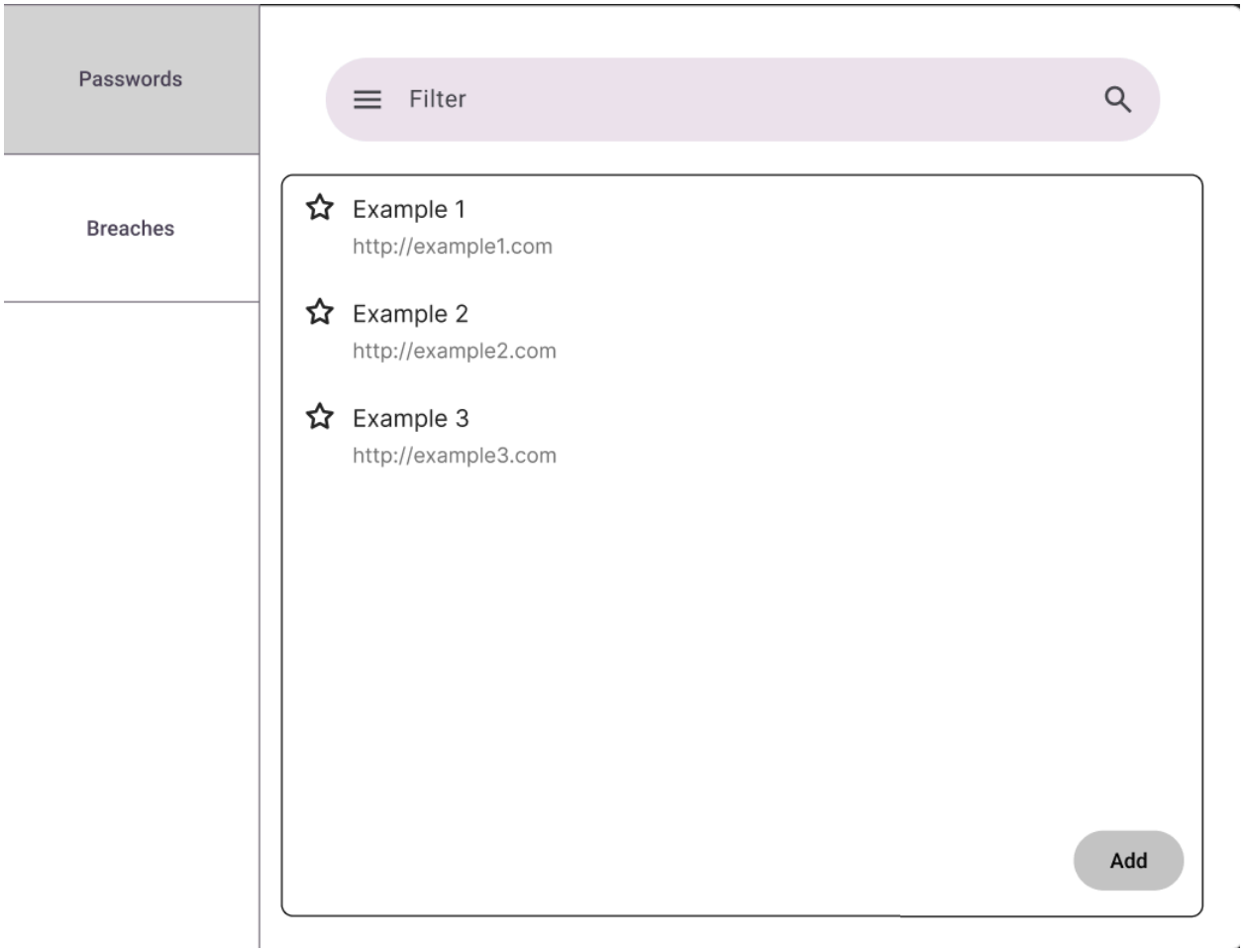
Design the screen for when the user is creating their Pic-a-Pass credential (Create account)



Design the screen for when the user is entering their password for Pic-a-Pass (Login)

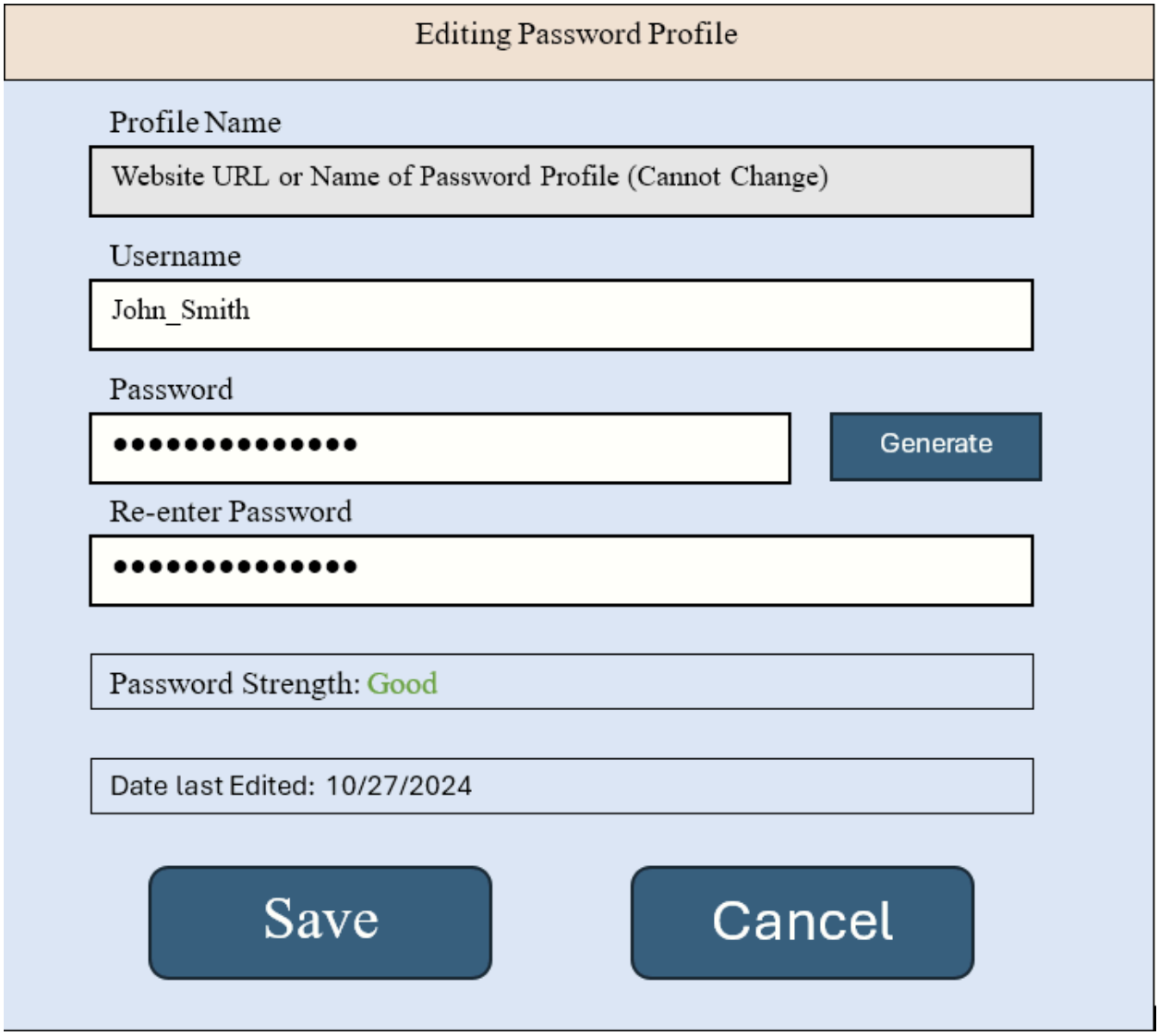


Design the screen to display the list of passwords the user has

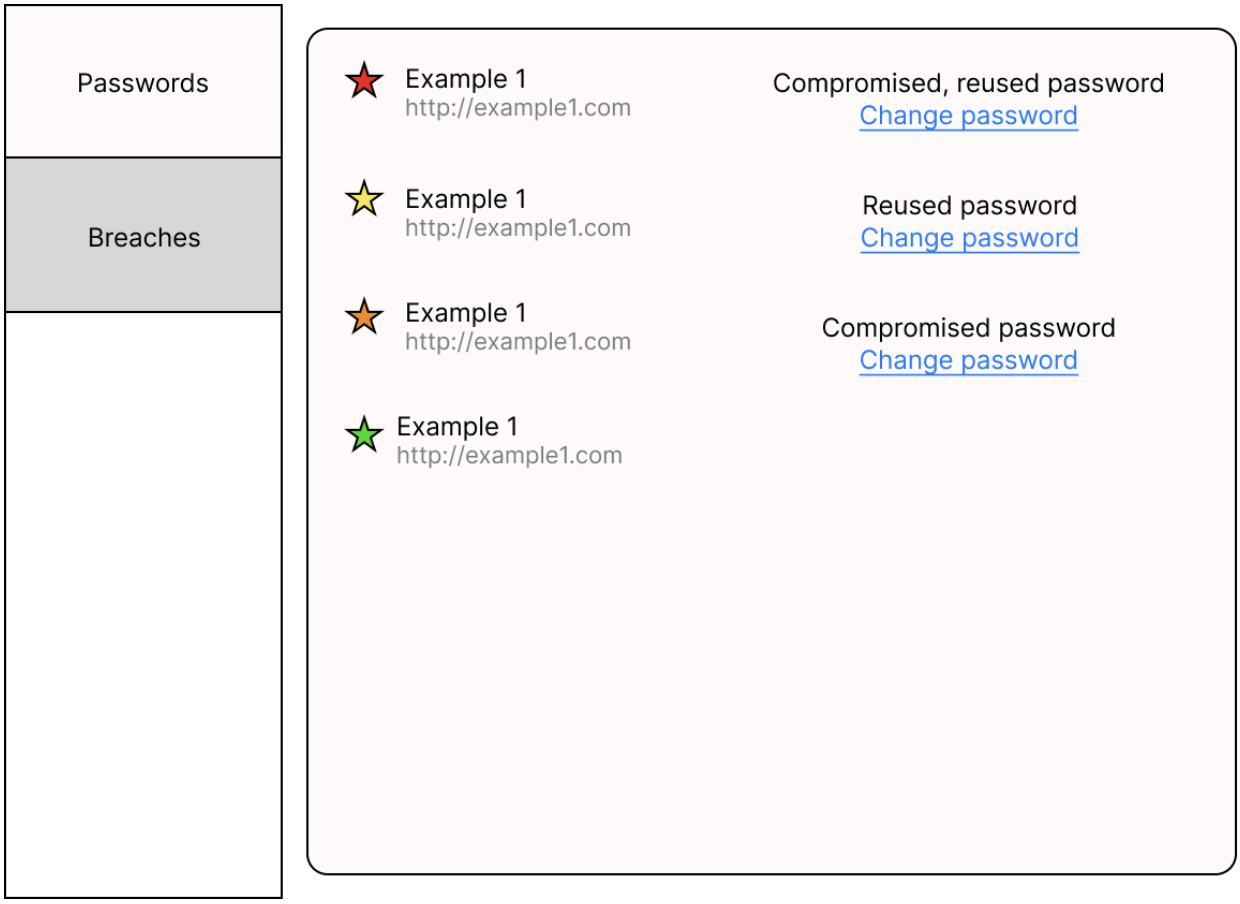


Design the screen when a user clicks on the password profile

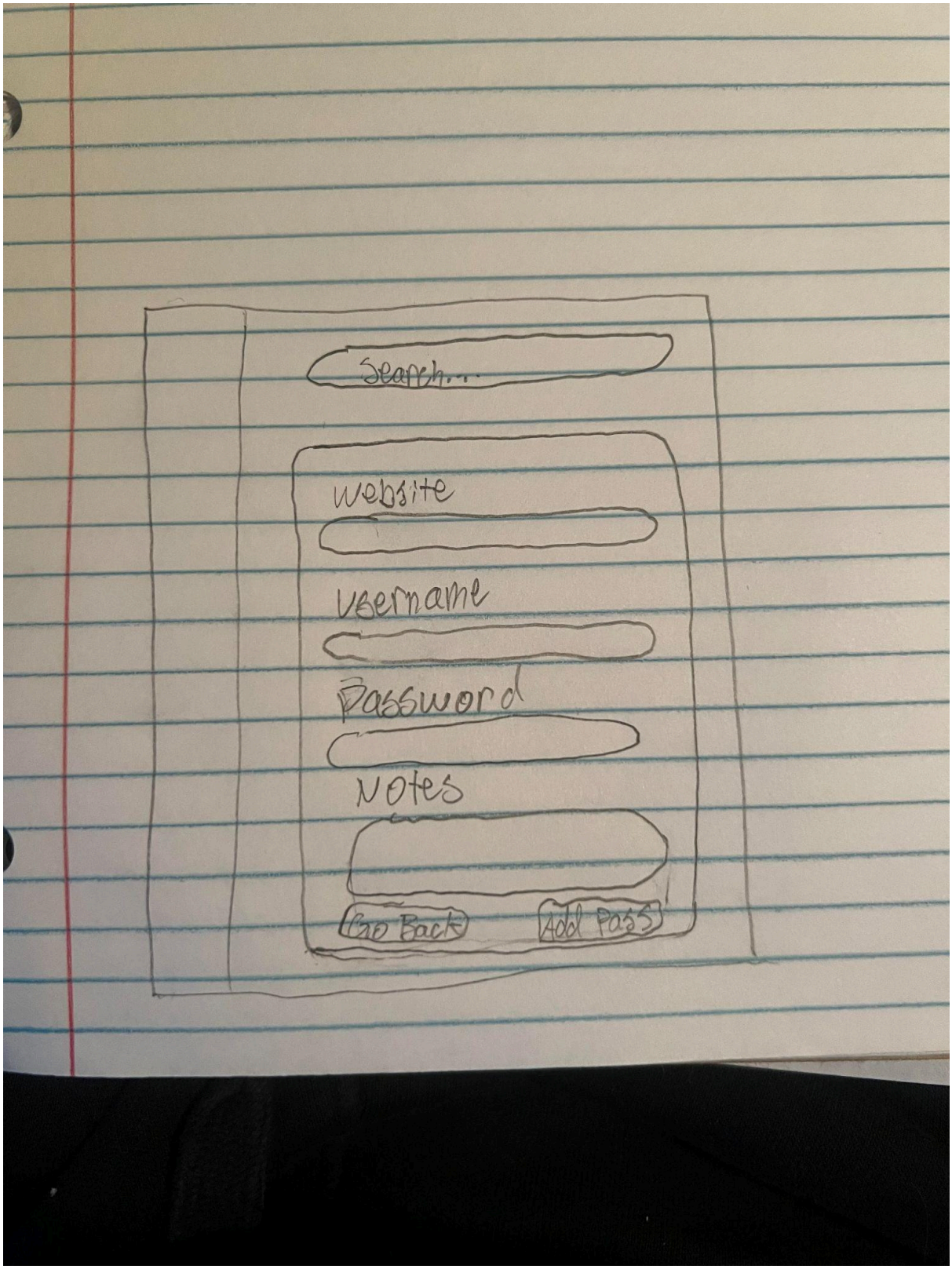
Design the screen for when a user is editing previously created password profiles



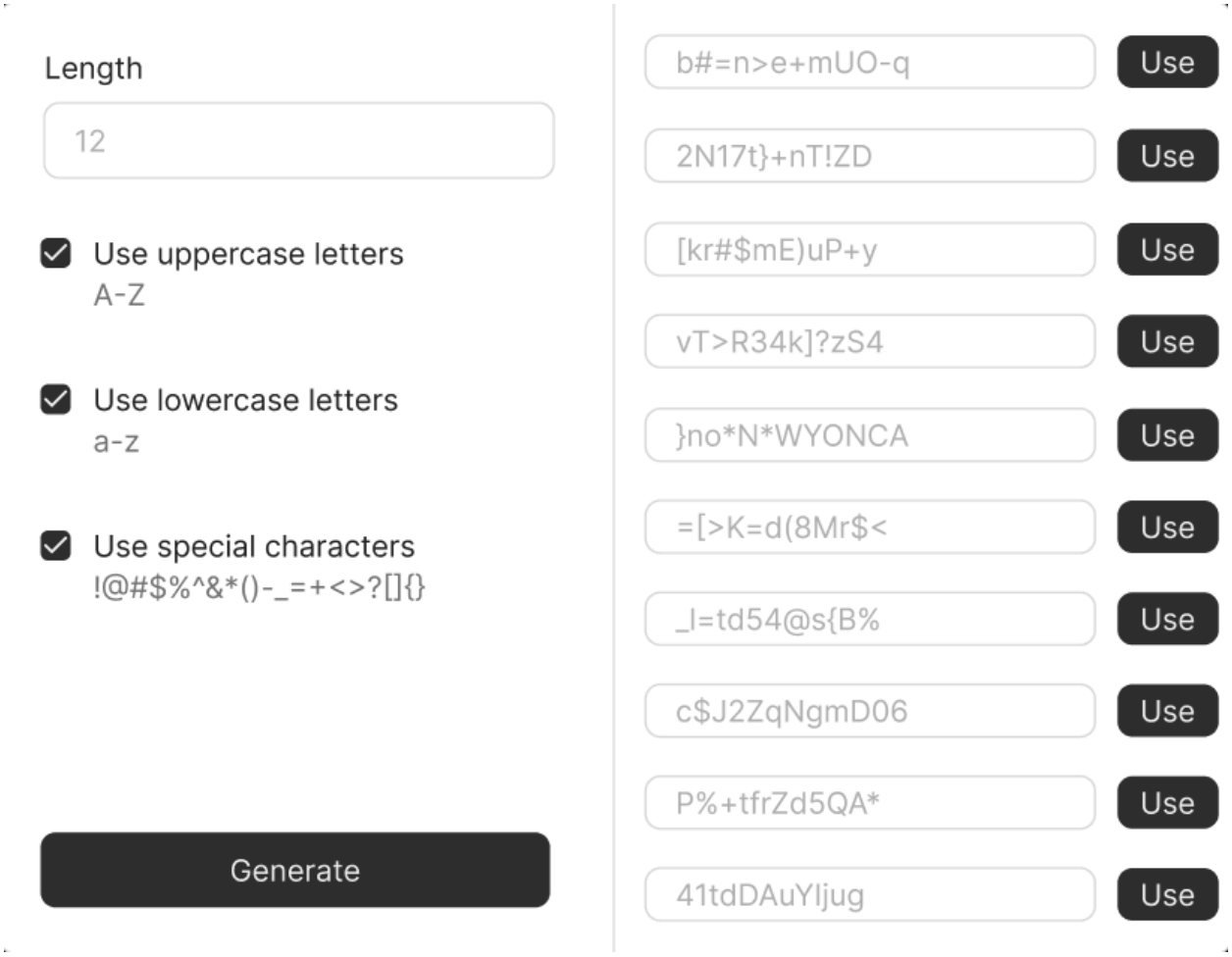
Design the screen for displaying the list of breached passwords



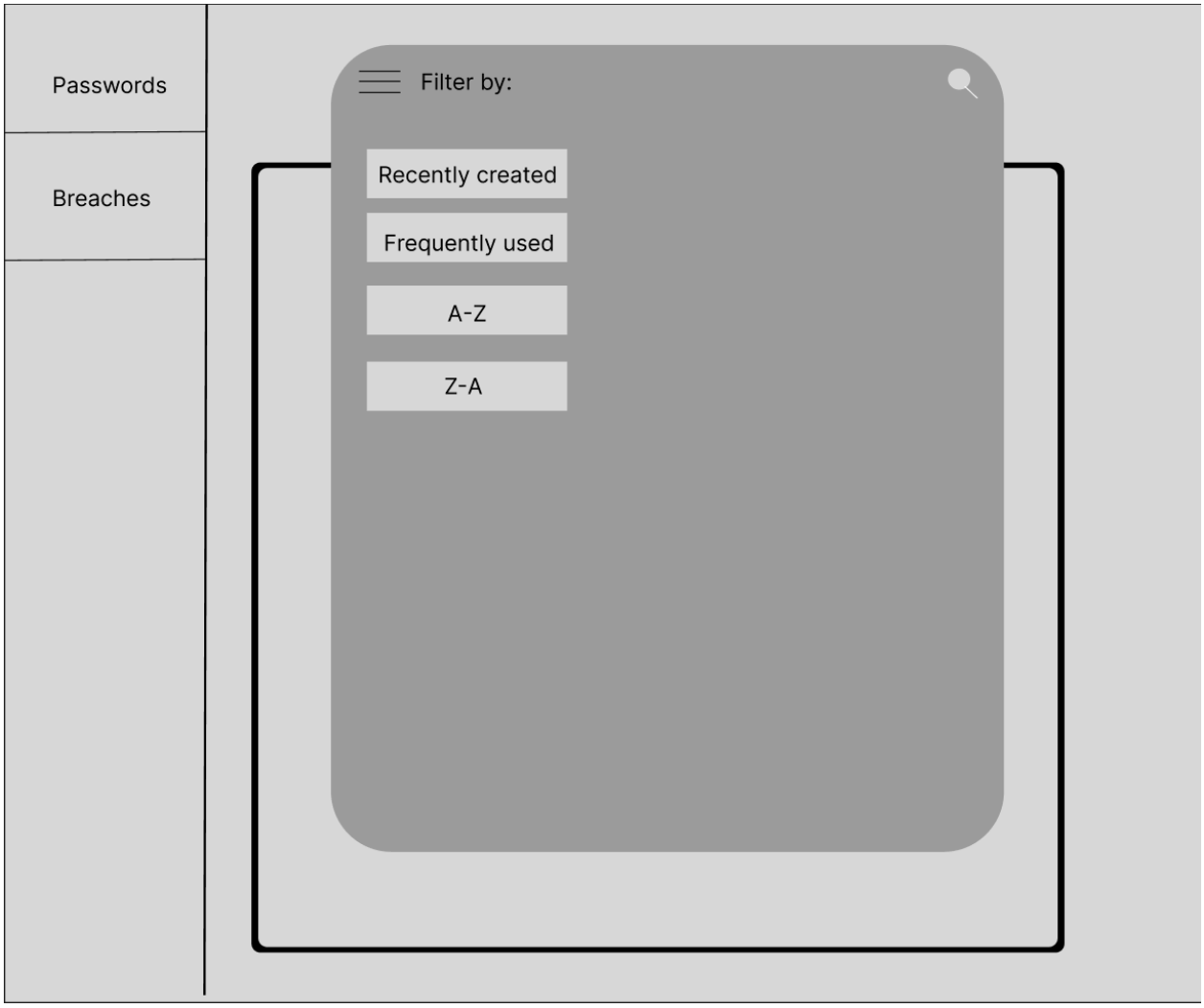
Design the screen for when a user is adding a new set of credentials



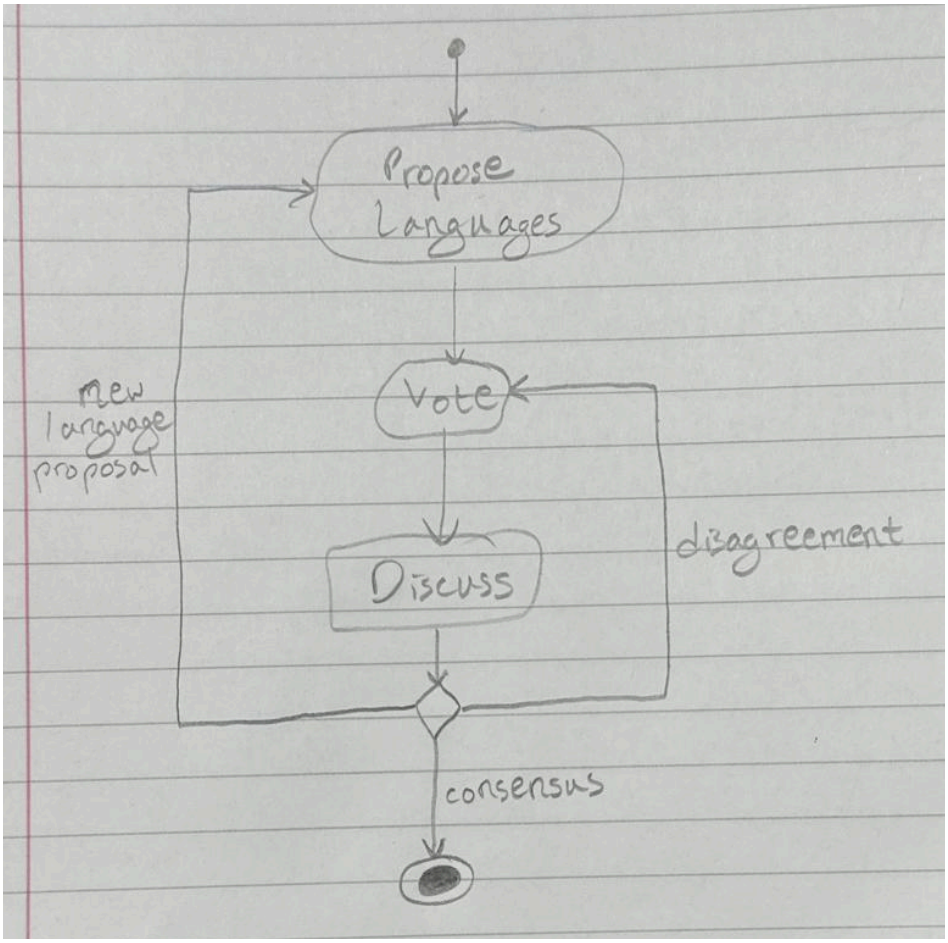
Design the screen for using the password generator



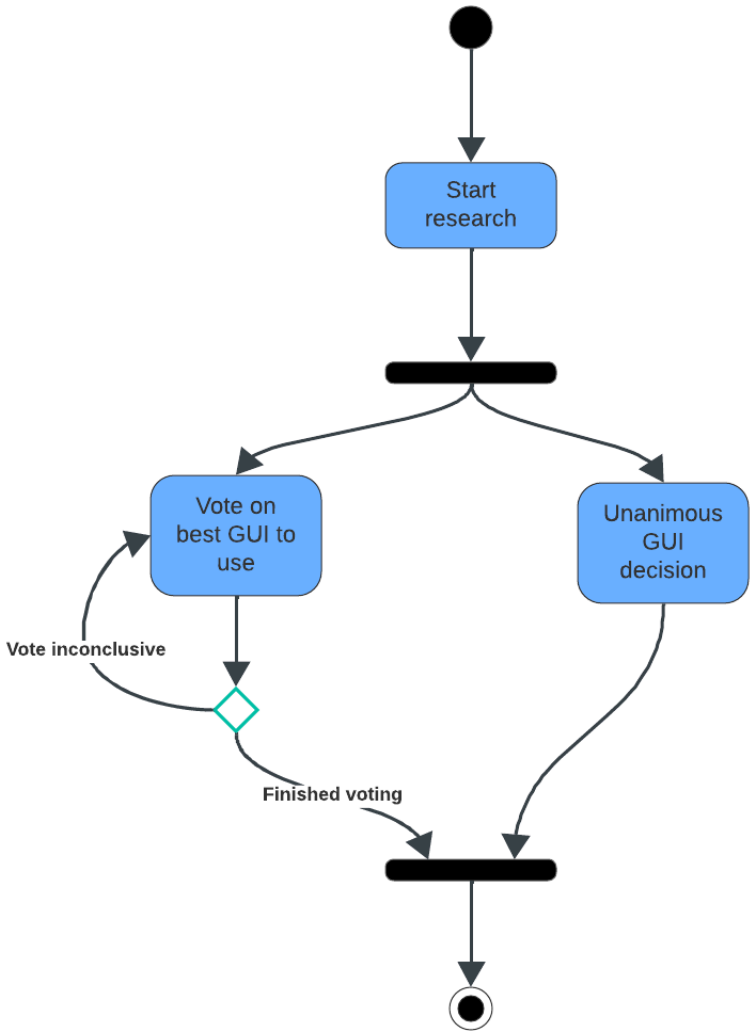
Design the screen for editing the search parameters



Requirements Artifact for Pick Language (ID: 4): No loop, consensus after discussion for Python. Answer: python. We decided Python was the best for GUI creation after doing research for GUI options within Python, especially for our short development time.



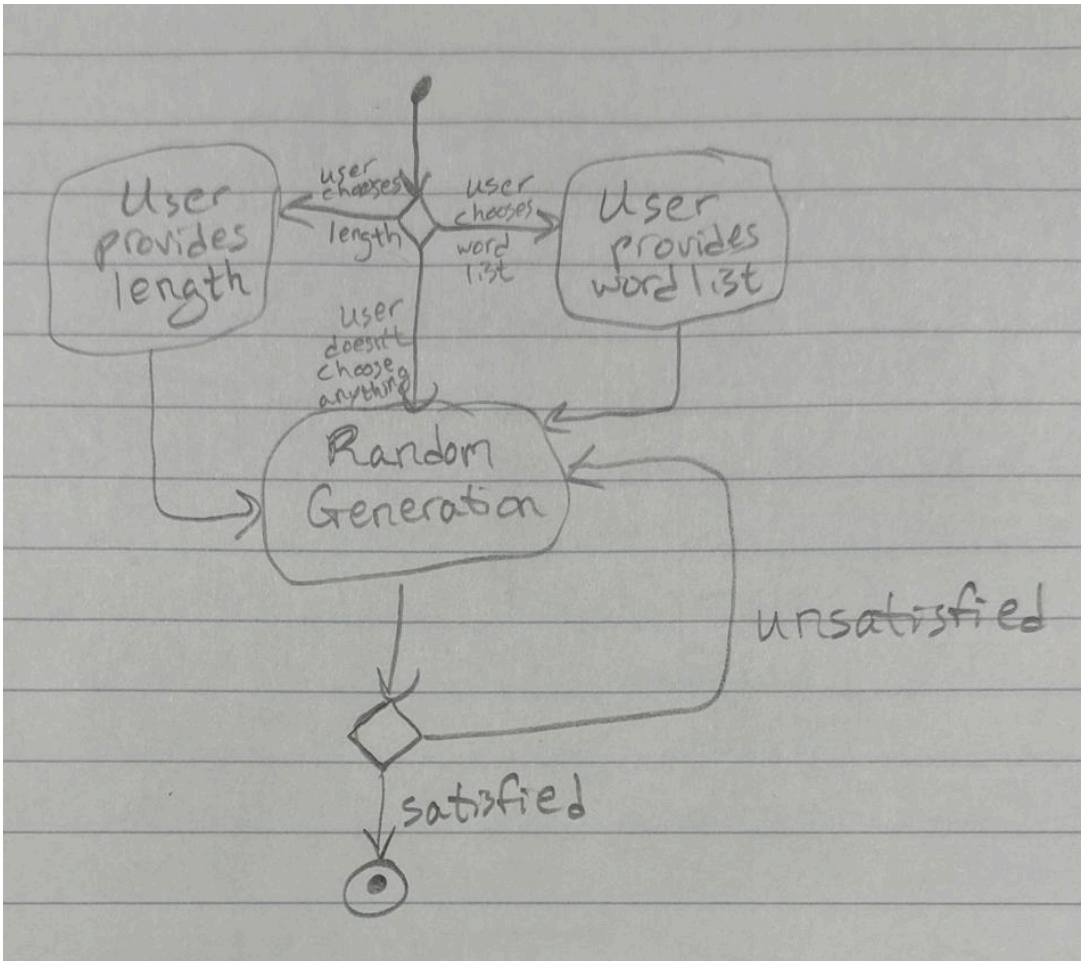
Requirements Artifact for Research GUI options in our Picked Language (Python) (ID: 3):
We decided to use pyQT for our GUI builder tool, as it works cross platform and has plenty of documentation for building GUIs.



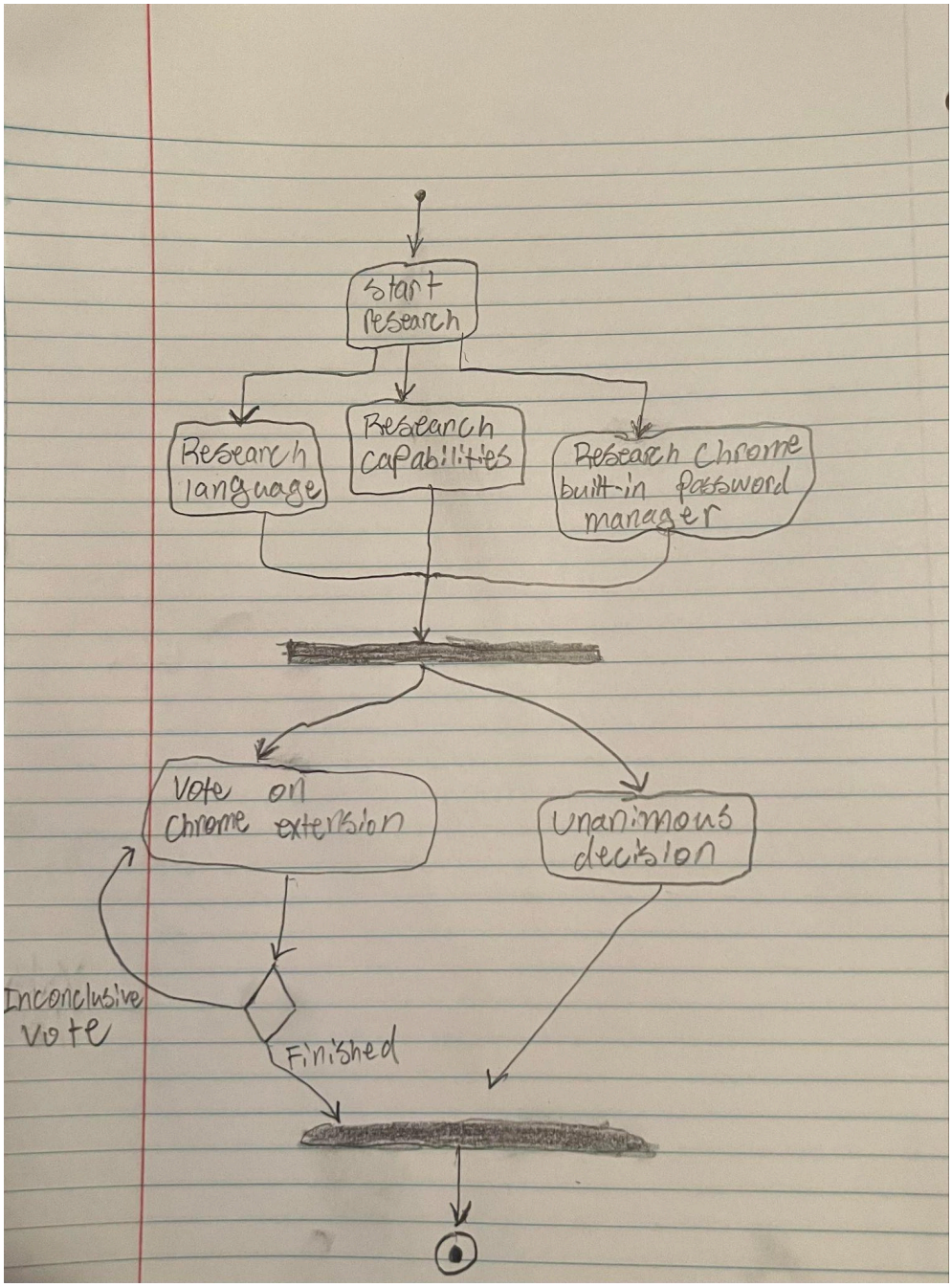
Requirements Artifact for Generate Password (ID: 21):

On GitHub

/



Requirements Artifact for researching chrome extensions (ID: 2):



Answers

- Chrome extensions are purpose built programs that enhance the experience using chrome, our goal is to incorporate our password manager to autofill passwords for the user. These extensions are primarily coded using html, css, and javascript to achieve functionality on

the chrome webpage. The programs use a manifest file as the code to give the extension its specific function.

- While we plan on using PyQT as the framework to build the Pic-a-Pass GUI it is not possible to create an extension that would be compatible with the Google webstore, but there are creators on GitHub that have made it possible to run offline web app that can create an extension running PyScript. This does not fully fit the requirements we are trying to reach on this project. That being said it is possible to connect the python run app with an extension by incorporating native messaging API. This would mean using html, css, and javascript to set up the front end of the extension. This would include communication channels to the python program set up in the manifest file, designing the user interface using html and css.

Collection of features for researching steganography (ID: 5):

- Is using steganography even a viable option for our program?
- If steganography is a viable option, how hard is it to implement?
- How widely used is steganography in the security world?
- How effective is steganography vs. regular password encryption?
- How could steganography be implemented within the program?
- What extra use for the user are we adding?
- Are there other ways of using steganography besides just images?

Answers

- Common vulnerabilities of current password managers revolve around the master password and password vault/database being leaked. Steganography could help mitigate this by storing our keys for the password hashes steganographically and requiring the user to create (or giving the user) a steganographically obfuscated master password. This would mean the user would be prompted to upload a file at login that would go through our own "decryption" algorithm to retrieve the master password from the uploaded steganographic file. This prevents the user from leaking their password by phishing attacks because it is even hidden to them through steganography. For example, if a phishing attack targets a user of another password manager, they could be coerced to give up their master password since they have it memorized or stored somewhere in plain text. However, a user of our password manager may not even know their master password (because it is stored inside a media file), so a phishing attack against a user of our password manager will be unsuccessful because the user may potentially not even know their password.
- One approach to implement steganography is
    - **Convert the Image to Binary**: Each pixel's color value is represented in binary (e.g., a pixel with RGB values of (255, 255, 255) becomes (11111111, 11111111, 11111111)).
    - **Modify LSBs**: Replace the least significant bit of the pixel's binary values with the bits of the data to be hidden.
    - **Reconstruct the Image**: After embedding, save the modified image, which will look almost identical to the original.

- Steghide is one open source tool that can be used to help implement steganography, but since the algorithm is so simple, it might be better to create our own algorithm so that we can modify it to our desired use case.
- Steganography is moderately used in the security world, primarily as a complementary technique to enhance privacy, evade detection, or bolster covert communication. One use case I have seen described for it is to send intelligence over an insecure network. This is useful in communication because anyone that examines the communication over the network will just see a normal media file, but the actual vulnerable content is hidden within the media file steganographically.
- Steganography goes hand in hand with regular password encryption to bolster the privacy of the encrypted message. In our case, the master password may not be encrypted before embedding it into the image file because then that would require the user to keep a key for that encryption algorithm and defeat the whole purpose of using steganography in the first place.
- The extra use would be storing and using media files on their machine in the form of hashed passwords and the master password that shouldn't add any additional use because the media files for the hashed passwords do not need to be touched by the user and the user would already have to memorize or store a master password, so in this case, it almost makes it easier for the user because they don't have to memorize anything or keep it in a secure location because it can be hidden in plain sight.
- Steganography can be used on any media file including, but not limited to:
    - **Images**: This is the most common medium for steganography. It often involves altering the least significant bits (LSB) of the image's pixels to store data without making noticeable visual changes.
    - **Audio**: Data can be hidden in audio files by modifying the LSBs of sound samples or by altering frequencies slightly.
    - **Video**: Video steganography involves concealing data in individual frames, offering more capacity for hidden data.
    - **Text**: Information can be hidden using whitespace manipulation or by adding invisible characters, though it has lower capacity.