6.08 Spring 2022 Final Project

LINK TO DEMO: https://youtu.be/kL8vj1RAu_Y

Documentation

Modifications

The system contains two main components: the ESP client which deals with taking the pictures and creating basic edits; and the server which handles displaying images and more complex edits of the images. The server requires no changes to setup, however, certain modifications must be made to be used properly. Each ESP client is associated with a unique username that can be modified. Any images captured from the camera are bound to the given name and are scoped to the given username in the server.

Basic Process

The ESP client has two main modes of operation: operations related to the camera, and operations related to the modification of images captured. If the user selects the "Take A Picture" option, then the ESP waits for the shutter button to be pressed, simulating an actual camera. Upon pressing the button, the ESP requests the most recent image buffer from the camera and converts the raw bytes into a base64 encoded representation of the image. This string is then sent over to the server through an HTTP request, before resetting to the original state.

If, instead, the user selects "Edit Recent Picture," then the ESP provides a black canvas on which the user can draw points and shapes. These changes are tracked and added to the image on demand. The changes are of two types: vocal commands and joystick movements.

Vocal commands are forwarded on to Google's speech recognition API to be processed. From here, the text is converted into information about the attributes of a change: its shape, height, width, and color. This is then used to construct a new change and is then displayed.

On the other hand, each pixel that is drawn using the joystick creates a new change whose shape is a point. At the end of the editing process, the cumulative changes are sent to the server and are then used to create a new version of the image.

To view all of the images posted to the server by a certain user contained in merged_image.db, you can make a get request with a url like this: https://608dev-2.net/sandbox/sc/team78/project/merged_request_handler.py?ID=user

The user is the ID of the user whose images you want to view. There is also an optional parameter you can add called name where you can specify the exact name of the image you'd like to view. This way, you only see that specific image posted by the specified user. The image display page shows you every version of each unique image by constantly cycling through each version of the image in a loop. From this main page you can click on any of the images to edit the original version. Clicking on an image reroutes you to a new display with the editing interface, where you can edit the image using sliding bars with different editing tools. From that page, there's a button to send the current edited version of the image to the database with a post request. After sending the post request, you are automatically rerouted to the original display page , with the updated version reflected in the loop for that image.

When an edited image from the server is posted, the server side code queries the database to find the highest version of the image you are trying to repost that currently exists in merged_image.db. It then increments that version by one and assigns that number as the version of the newly posted image. This way, the most recent edited photo will have the highest version number of all forms of the original. Then, when you are taken back to the display page, the newest version will appear in the cycle of image history. When a new "original image" is sent from the ESP or a manual POST request, it is sent without a name parameter. The server then queries the database to find the highest number name in the current database and increments it by one. It then posts the new original image with this incremented name–so the latest original image has the highest integer name.

When an image is posted there is also an optional filter parameter you can pass in. The filter parameter can take the values of bw, emboss, contour, edges, min and max. Each of these parameters applies a different filter to the image before it's posted to the database. Most of the filter names are self explanatory but min expands the minimum pixels in an image and max expands the maximum pixels in an image, making the darker or lighter parts of the image appear thicker.

State diagram

State 0: IDLE STATE - Display the menu selection and wait for the user to select. If "Take A Picture" is selected, go to state 1. If "Edit Recent Picture" is selected, go to state 4. If the user decides to open a browser, transition to state 15.

State 1: SHUTTER INPUT STATE - Wait for the shutter button (button 39) to be pressed, if it is pressed, go to state 2. Else, stay in state 1.

State 2: CAMERA STATE - Take a picture using the ArduCam and wait for the internal image buffer to be filled. When the "done" flag is set to true, go to state 3.

State 3: UPLOAD STATE - Encode the buffer into base64 and send the base64 image and username to the server merged_request_handler.py through a POST request. Transition to state 11.

State 4: ESP EDIT STATE - Wait for either the audio button (button 39) to be pressed or the joystick to move. If button 39 is pressed, go to state 5. If button 26 is pressed, go to state 9. If button 45 is pressed, then go to state 16.

State 5: RECORDING STATE - If button 39 is pressed and it has been less than five seconds since the beginning of the recording, continue to listen to audio and stay in this state. Otherwise, go to state 6.

State 6: PARSE AUDIO STATE - Send audio to Google's speech recognition API and wait for response. Once the response has been received, go to state 7.

State 7: AUDIO CHANGE STATE - Parse the transcript as a change and add the change to the current list of changes. Set display change flag to true. Go to state 8.

State 8: DISPLAY STATE - If the display change flag is true, display the last change. Set the display flag to false. Go to state 4.

State 9: JOYSTICK STATE - If the joystick was moved, figure out direction and change current pixel location, and go to state 10.

State 10: PIXEL STATE - If button 26 was pushed, then change the current pixel color to be white. Set the display change flag to true and go to state 8. Else, go to state 4.

State 16: EDIT SUBMIT STATE - Send a PUT request to the server with the changes from the edit state. Go to state 0.

State 11: POST STATE - User can submit their image to the database merged_db.py through a POST request to the server merged_request_handler.py given at least 1) a username as a parameter and 2) the image's base64 string as the body.

- If post request is sent from state 3: after receiving server's response, the machine automatically transitions back to state 0
- If post request is sent from the server (from state 14):
  - The server has a built-in time delay of 1 second so stay at state 11
  - Then server refreshes so transition back to state 12/13

State 12: GALLERY STATE - Passing the username as a parameter, the user can view all the images from the ArduinoCam that they posted to the database through a GET request to the server merged_request_handler.py

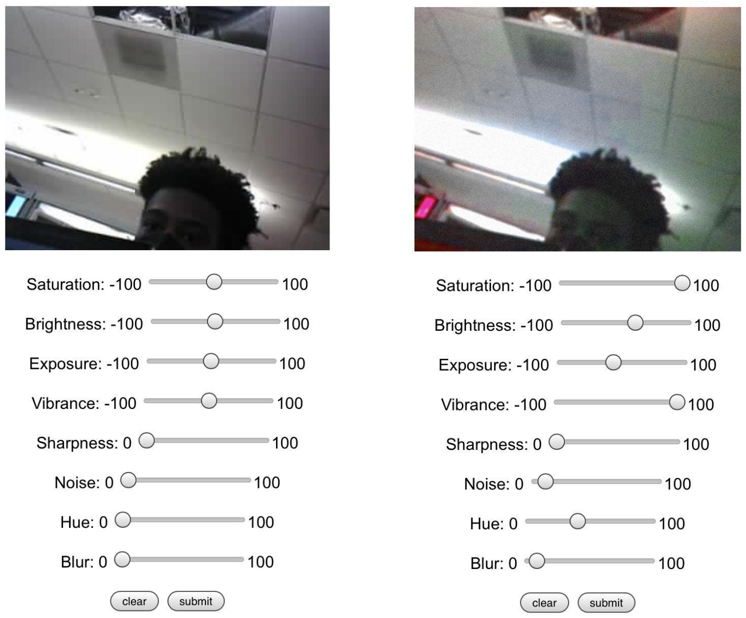
- Transitions to state 14 when the user clicks one of the images on the gallery.

State 13: IMAGE HISTORY STATE - User can view all versions of a specific image given the image name and user as parameters in their GET request to the server merged_request_handler.py

- Transitions to state 14 when the user clicks the displayed image.

State 14: EDITING STATE: Enables the user to manually edit the image they clicked at either state 12 or 13. Note the user is still on the merged_request_handler.py but the HTML view has changed to accommodate the editing interface.

- After editing, the user can choose to clear the image so the state remains at state 14
- The user can reload the page so to go back to state 12/13 (the one transitioned from)
- The user can submit the edited image as a new version edit so transitions to state 1



Side-by-side comparison of the type of image edits that may be made at state 14

State 15: GET STATE - Instead of interacting with Arduino, user sends a GET request to the server merged_request_handler.py

- While waiting for response, stay at state 15
- If name is not given as parameter transition to state 12
- If name is given as parameter transition to state 13

Design Challenges

- The original time lapse feature was meant to be an interactive feature where on the ESP32, certain original images are marked as part of a time lapse and replayed accordingly on the server after post submission. However, this was an idea proposed for week 4, so we took heed of our time constraint and decided on a simpler time lapse feature, where an image looped through all its corresponding version edits.
- Arducam and LCD display (solution was to change the frequency in the source code of TFT User Setup library)
- Displaying base64 encoded strings from the IMU and the server. The issue was with how we were trying to display the image in HTML, not the bas64 string itself. The source attribute of the images was not being set correctly.
- For better cohesion, we integrated the entire server side, including two different GET requests, and PUT/POST requests into one script merged_request_handler.py.
- The initial idea was to use an IMU to draw with a cursor and separately use voice commands to draw shapes. However, we opted to ease the user experience by replacing the IMU with a joystick and draw shapes based on the position of the cursor. This not only made drawing shapes less like reading a very old car manual, but also greatly simplified the voice commands.

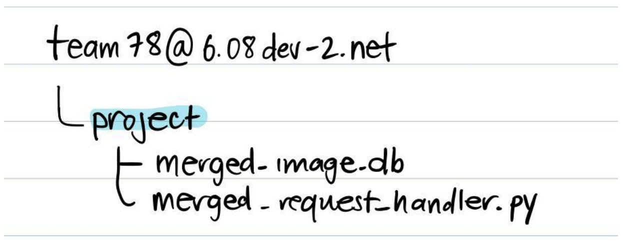
Parts List

Item	QTY	Description	Cost/UNIT	Supplier	Part Number	Link
Arducam Mini Module Camera Shield	1	Camera Shield	$25.99	Amazon	UCT_B0067	https://www.amazon.com/Arducam-Module-Megapixels-Arduino-Mega2560/dp/B012UXNDOY/ref=sr_1_4?dchild=1&keywords=arducam&qid=1617653366&sr=8-4
Joystick Module	1	Joystick Component	$8.59	Amazon	B103348	https://www.amazon.com/01-Performance-Transducer-Interfaces-Two-Dimension/dp/B09GY2NCTC/ref=sr_1_2?keywords=joystick+module&qid=1652044998&s=videogames&sr=1-2

A majority of the parts used have been given in the base kit. However, this project requires a new component called the Arducam Mini Module Camera Shield to take photos. Additionally, a 2-axis joystick will allow users to draw over and edit images freely.

Code Layout

Serverside



PUT request - merged_request_handler.py

The PUT request of merged_request_handler.py accesses the latest image version of the image that the client wants to edit. This is determined through the two query parameters, the id of the user and the name of the image to edit. Using these two, a database query is generated to find the highest version of the image with the given name under the user with the given id. From here, the changes that are sent in the body of the request. The body is a JSON object that describes the list of changes, where each change object describes a specific change, such as the width, height, x, y, and color of an ellipse or rectangle. These changes are then applied to the image using the Pillow library and is then re-inserted into the database as a new version. The Pillow library deals with image processing and modification, which perfectly suited our needs for this part of the project.

GET request - merged_request_handler.py

For these HTML displays rendered by merged_request_handler.py, two major JavaScript modules via CDN script sources are utilized to achieve our intended functionality. JQuery is a very ubiquitous standard module for upgraded JS functions, while CamanJS is a flexible image editing library that fulfills all the editing attributes we wanted to support on our server.

The GET request of merged_request_handler.py contains two separate page displays: the first page is the image gallery, which may be of just one image if name is specified in the parameter. This first page displays images with an HTML <img> tag blocked by a <button> tag. The button and img tags both contain metadata in its attributes to carry information about each image's name, version, and base64 representation. These buttons are all in a class called "image_button," and the images each have a class that corresponds to their name. The grouping of these classes is so that similar objects can share mutual functions and are easy to extract in JavaScript. Animations are achieved through the function animate() that is called by the window every 500 milliseconds.

On the editing interface, we use range <input> tags with unique HTML IDs to extract user input and after every drag, the JavaScript function applyFilters() is called and updates the image based on the user input's edit inputs. The function clear_image() is called when pressing the "Clear" button and reverts the image back to before any server-side edits have been made. The function submit_image() is called when pressing the "Submit" button and sends a POST request to the same server before reloading the site back to its original gallery display.

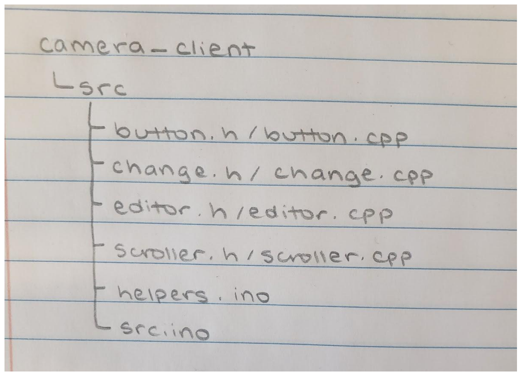
POST request - merged_request_handler.py

If the POST request doesn't specify a name, it is assumed the image is an original sent from the ESP. The request handler gets the highest integer name currently in the database and assigned that number incremented by one to assign to the new image with a version of one. Otherwise, if a name is passed in with the image, it is assumed to be an edited version of another image. The request handler queries the database to identify the highest version of the image and assigns the most recent edited version this integer incremented by one. If the post request specified a filter parameter, it applies the specified filter to the image and then posts the image with the given edits, user, name and version.

Database - merged_image.db

The database, merged_image.db, located in the project folder of our team's server has four attributes: id, img, name, and version. The id is text identifying the user that sent this image to the database and img contains the base64 encoding of the image as text. Name and version are integers that identify if an image is an original picture, or an edited version of a previously existing photo in the database. When an image is posted from the ESP, it is not sent with a name parameter. If an image is sent without a name parameter, this indicates that it is an original image and the merged_request_handler.py queries the database to find the highest current name in the database. It then increments that number by one and assigns that integer as the name of the new image along with a version of one to indicate that it is the first version of this image. These four fields are then inserted into merged_image.db. If an image is reposted from the image editing HTML page, it is sent with the same name parameter as the original, unedited image. The server side code in merged_request_handler.py then queries the database to find the highest current version of that image. It then increments that number by one to get the version that corresponds to this new image. With the same name as the original image and the incremented version value this new edited photo is inserted into the database.

ESP-side



```
camera-client
└src
  ├button.h / button.cpp
  ├change.h / change.cpp
  ├editor.h / editor.cpp
  ├scroller.h / scroller.cpp
  ├helpers.ino
  └src.ino
```

button.h / button.cpp:

This file contains the declaration and implementation of the Button class from the classy button exercise from week 4. No modifications were required to this class. This class is used to help detect button presses, specifically knowing when to take a picture and send the changes. This class was not used in determining when to draw pixels onto the screen because of the way that the class relays changes about the state of the button press.

change.h / change.cpp:

This file contains the definition and implementation of the changes that are done on the ESP side. The structure contains the shape drawn, x, y, height, width, and the color, although the meaning of these values changes based on the shape drawn. These files also contain helper functions for dealing with conversions to and from other types, such as string.

editor.h / editor.cpp:

This file contains the Editor class which abstracts the implementation details behind the editing functionality. Specifically, it manages the different changes that will be submitted to the server and also deals with parsing changes from audio transcripts. This class also defines the maximum number of changes that can occur, which has been hard-coded to be 400.

scroller.h /scroller.cpp:

This file contains a modified version of the WikipediaUI exercise. Specifically, it deals with selecting either to take a picture or to edit the most recent picture. The class also makes selections accessible through a function that returns the selected index. Additionally, it allows for an arbitrary number of options to be made.

helpers.ino:

This file contains various helper functions that deal with miscellaneous tasks, such as sending HTTP/HTTPS requests, setting up the ArduCam and the IMU, and connecting to WIFI. This file is similar to the support_functions.ino file that is provided in the labs.

src.ino:

This file contains the main source code for the ESP client. It orchestrates the multiple classes together by initializing the required buttons, scroller, editor, presenting a user selection menu in the loop, and delegating selections to the responsible class.