L2B-18 Team Report

Introduction

The Virtual Interview Session (VIS) is a mock interview tool for computer engineering co-op students who want to improve interview delivery skills and become more comfortable with the interview setting. Using the tool, users can conduct practice interview sessions and receive quantitative feedback for their eye contact, speech sentiment, and speech clarity. As a result, VIS enables users to self-evaluate their performance and additionally makes interview practice more accessible and flexible to the users. Now, interview practice can be done on their own schedule and within the comfort of their own home!

System Description

The VIS project has a companion website that serves as a user interface. The companion site allows users to start and stop interviews and select between knowledge and behavioral questions. Once a user begins an interview, two DE1-SoC boards are responsible for capturing the user's interview performance. One DE1-SoC board streams the camera feed to a VGA monitor and periodically captures 224 x 224 images using a custom Qsys component. The images are also compressed in the Qsys component using a JPEG-inspired encoding algorithm. Another DE1-SoC board uses the HPS's audio drivers to record audio. Both the audio and video are sent to the server using HTTP POST requests, where machine learning models hosted on the server and Google Cloud Platform will produce scores for eye contact, sentiment analysis, and speech clarity. On the companion website, users can navigate to a feedback page to view a session transcript and their eye contact, speech sentiment, and speech clarity scores. The companion site also has functionality to view their transcripts and feedback for the previous interview sessions.

The VIS had a few features that were especially challenging to work on. One such feature was the image encoding algorithm due to the amount of time and effort required to explore implementation options and build up the necessary understanding of the fundamental concepts related to image compression. Additionally, managing the continuous image transfer between the HPS and the backend was difficult because it could overflow the storage or cause the data flow to be delayed. A final challenging aspect of the project was interfacing with a camera in hardware and getting its data to the server. This required us to gain a good understanding of existing projects so that we could build off them without compromising the original functionality.

Breakdown of Member Contributions

Azwad Sadman

- Used Google Speech-to-Text API for implementing speech recognition
- Developed a mechanism to analyze sentiment of the text using Google Natural Language API
- Designed and developed the front-end application using React framework

Christopher Chang

- Researched and deployed an eye contact machine learning (ML) model to the server
- Developed a software-based image decompression algorithm
- Designed an image compression algorithm using hardware and software

Daniel Lee

- Deployed AWS server to communicate with frontend, hardware, Google Cloud Platform and eye contact ML model, sending and receiving information to be analyzed and displayed.
- MongoDB database is built to store interview questions, analysis results and user information.

Patrick Creighton

- Read and processed D8M-GPIO camera data in hardware
- Created a Qsys component that registers pixel values and sends them to the HPS upon request
- Read and processed image and audio data in the HPS and sent it to the server

Conclusions and Solution Assessment

We consider our final design to be a success since we were able to meet many of the goals and implement many core features established at the beginning of the project. We were able to successfully design and program hardware to capture audio and images; implement the necessary hardware, software, and cloud systems to score eye contact, speech sentiment, and speech clarity; and create a user interface and server to conduct interview sessions and manage the flow of data between individual components. These completed features satisfy the original project goals and serve as the hardware, cloud, user interface, and networking components needed to satisfy course requirements.

Overall, the project is not very robust in its current state, as it requires actions to be taken in a specific order to maintain functionality. In hindsight and had we had a better understanding of how long certain features would take, we could have limited the project scope to devote more time to developing a more robust system. In addition, we could have allocated more resources toward the complex tasks earlier in the project's development. This way we minimize the risk of being unable to address things due to a lack of time.

Patrick Creighton Individual Report

Student Number: 70067053

Contribution

Contributions are listed in order from largest to smallest effort.

Read and process D8M-GPIO camera data in hardware

I worked on this task by myself. I started by reading over the code and documentation for the D8M demonstration projects to select one to be the starting point of our hardware component. I adapted the demonstration to:

1. Output the coordinates of the pixel whose RGB values are being outputted
2. Draw a bounding box around the pixels that make up the input of our eye contact machine learning model
3. Control and verify the features above in real time switches and hex displays

Next, I exported this data to a custom Qsys component based off Tutorial 1.8 of CPEN 391 2019W that continuously registers it until the HPS requests an image. On an image request, the registers stop updating and are fed into a module that computes its DCT, which can be read from the HPS via the Avalon Memory-Mapped Interface.

It took a significant amount of time and effort to design and implement this task; I had to gain familiarity with the more advanced features of Quartus, and explore and choose from a variety of possible implementations. I will go into more of the challenges I experienced in the Challenges section of the report.

Read and process image and audio data in the HPS and send it to the server

I worked on the image portion myself, and worked with Azwad on the audio portion. The HPS is running Linux.

For the image portion, I wrote C functions to control and read from our memory-mapped hardware component. Specifically, I saved the DCT output to an array and verified that the control signals of the hardware component that I output matched what I expect at different points in time. Then, I created a Python wrapper script that sends the DCT output to the server as a JSON object. This script also has run length encoding implemented in software.

For the audio portion, I wrote a Python script that uses the provided audio drivers to read data from the Mic In port. This data is calibrated to the microphone we use before it is played back via the Line Out port or sent to the server as a JSON object.

I also set up the DE1-SoC to configure and connect to the internet, as well as load the drivers and executables our scripts use, on boot to make our project easier to run.

Project management

I set up a mirror repository that we worked off so that I could have administrative access. I created issue labels, milestones, discussions, Kanban boards, and wiki pages to organize and coordinate our efforts. I reviewed pull requests to ensure the main branch remained clean and functional as we integrated tasks, and wrote documentation on the repository's directory structure as well as instructions on how to set up, run, and develop on our project.

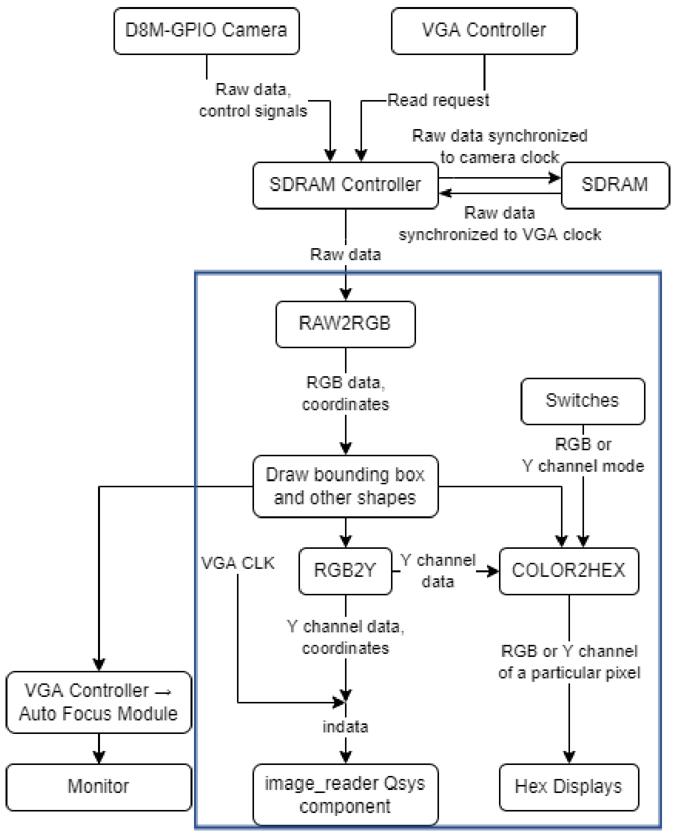
Optimize DCT implemented in hardware

I helped Chris with this task by suggesting ways to minimize the number of bits required for the inputs, outputs, and internal signals while retaining accuracy. For example, changing the fixed-point representation that is used.

Detailed Design

Read and process D8M-GPIO camera data in hardware

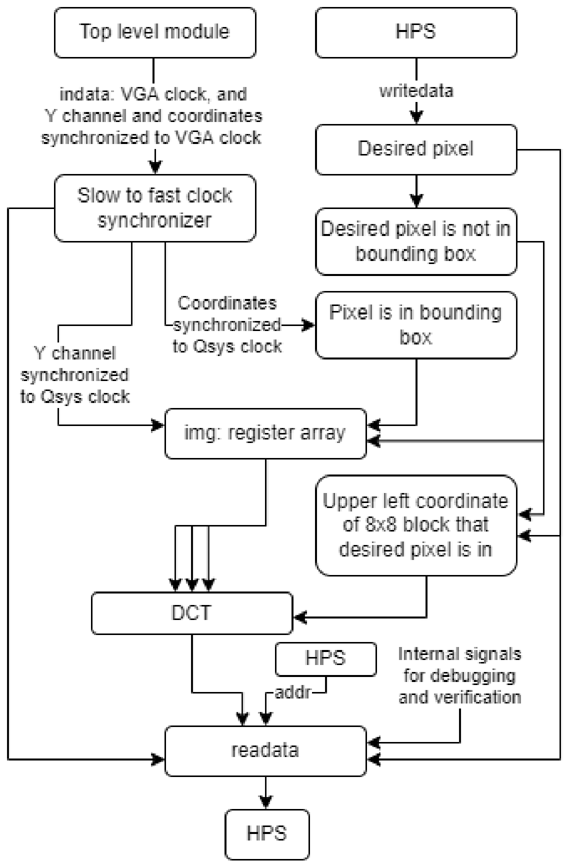
Top-level module


D8M-GPIO Camera
VGA Controller
Raw data, control signals
Read request
Raw data synchronized to camera clock
SDRAM Controller
SDRAM
Raw data synchronized to VGA clock
Raw data
RAW2RGB
RGB data, coordinates
Switches
RGB or Y channel mode
Draw bounding box and other shapes
VGA CLK
RGB2Y
Y channel data
COLOR2HEX
Y channel data, coordinates
RGB or Y channel of a particular pixel
VGA Controller → Auto Focus Module
indata
Monitor
image_reader Qsys component
Hex Displays

The D8M frame buffer demonstration project was used as the starting point for our top-level module, upon which I added the features inside the blue outline. SDRAM is used to store 1 frame of the video feed at a time. The write side FIFO of the SDRAM controller is connected to the D8M, and updates the frame when the camera has a horizontal and vertical sync. The read side FIFO of the SDRAM controller is connected to the VGA controller, which sets its output read request when it wants to read a new frame. This is then fed into the raw to RGB conversion module, which I adapted to output the pixel coordinates of the outputted RGB values. A bounding box around the 224 x 224 pixels in the center of the display is drawn, along with various other shapes that I use to test changes to the code against a known input. The resulting RGB values and coordinates are used for three different things:

1. Displaying the video feed on a monitor. The bounding box serves as a reference for the user, to help them keep their face within the pixels that are sent to the eye contact ML model.
2. The Y channel of the YCbCr format is approximated using shift operations from the RGB values. This 8-bit integer is sent along with the VGA clock and coordinates to the image_reader Qsys component. By only using the Y (grayscale) channel, approximating it as an 8-bit integer, and using shift operations, we minimize the amount of hardware required to compute the result, and the number of bits that need to be sent to image_reader. Furthermore, there was a negligible performance difference in the eye contact ML model when using grayscale instead of full-color images as input.
3. Coordinates of the pixel, along with its RGB and Y channel values, are sent to the COLOR2HEX module. Based on the position of switch 1, the RGB or Y channel of a particular pixel will be outputted to the hex displays approximately once a second. This module is used to check that we can consistently read a pixel and verify the Y channel computation.

Qsys component


Top level module
HPS
indata: VGA clock, and Y channel and coordinates synchronized to VGA clock
writedata
Desired pixel
Slow to fast clock synchronizer
Desired pixel is not in bounding box
Coordinates synchronized to Qsys clock
Pixel is in bounding box
Y channel synchronized to Qsys clock
img: register array
Upper left coordinate of 8x8 block that desired pixel is in
DCT
HPS
addr
Internal signals for debugging and verification
readata
HPS

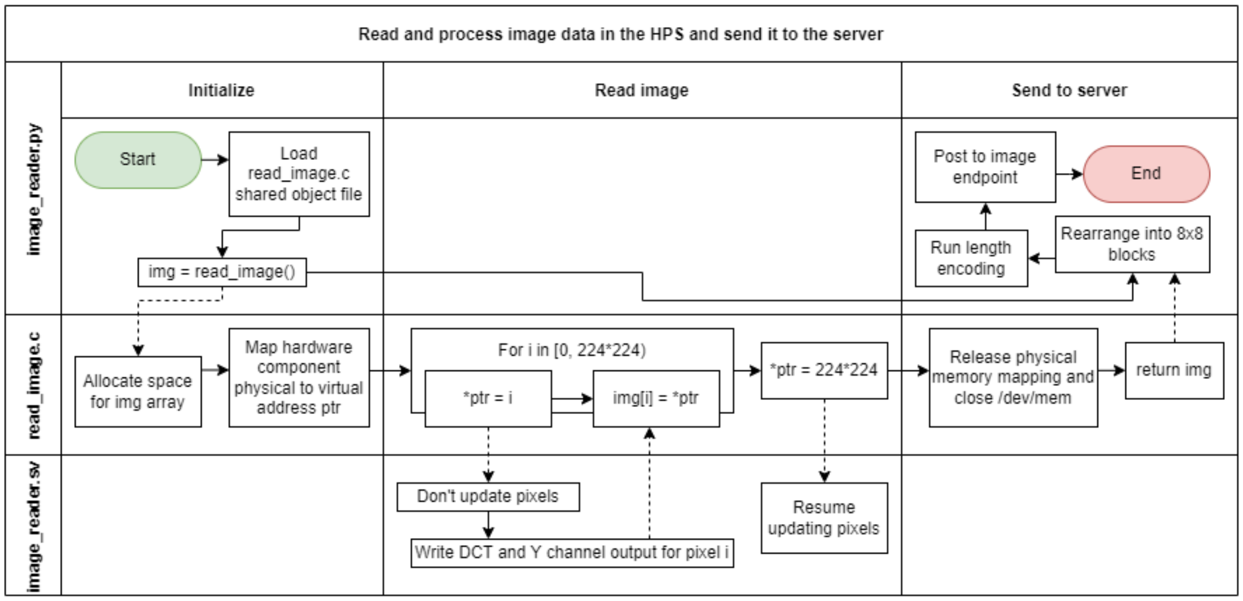
Y channel value and coordinates are received from the top-level module, then synchronized to the faster Qsys clock. If the coordinates of the pixel are in the bounding box, and the coordinates of the pixel desired by the HPS are not, the Y channel will be saved to img, a register array of size 224*224 pixels x 8 bits per pixel. We want to stop updating img when the desired pixel is in the bounding box because we don't want it to change while the HPS is in the middle of reading an image.

DCT works in 8 x 8 blocks, so in our 224 x 224 image we have 28x28 8 x 8 blocks. The upper left coordinate of the 8 x 8 block containing the desired pixel is computed so that the block can be fed into the DCT module. Based on the address given by the HPS, we return the DCT computation, Y channel, or internal signals for the desired pixel (see Appendix A for the exact image_reader output). Currently DCT is only performed on one 8 x 8 block, see the Challenges section below for the compilation issues we encountered and potential solutions, and Chris' report for more details about our DCT implementation.

Read and process image and audio data in the HPS and send it to the server

In both the image and audio Python scripts, constants were extensively used to be able to quickly change parameters, toggle features, and debug code.

Image


Read and process image data in the HPS and send it to the server
Initialize
Read image
Send to server
image_reader.py
read_image.c
image_reader.sv
Start
Load read_image.c shared object file
img = read_image()
Allocate space for img array
Map hardware component physical to virtual address ptr
For i in [0, 224*224)
*ptr = i
img[i] = *ptr
*ptr = 224*224
Release physical memory mapping and close /dev/mem
return img
Don't update pixels
Write DCT and Y channel output for pixel i
Resume updating pixels
Post to image endpoint
End
Run length encoding
Rearrange into 8x8 blocks

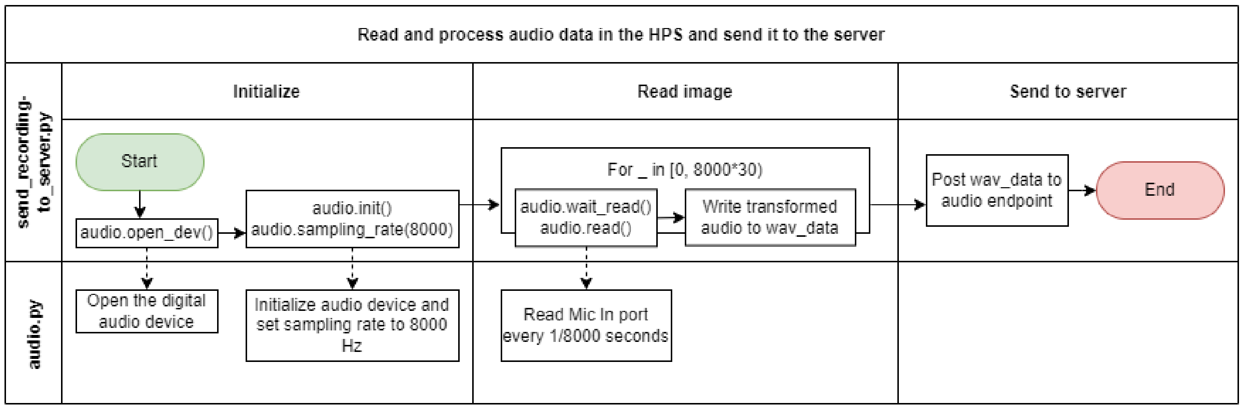
Before running the Python script, the C code needs to be compiled into a shared object file named test.so (cc -fPIC -shared -o test.so read_image.c) and located in a directory in the LD_LIBRARY_PATH environment variable (export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/test.so). Additionally, fpga/demo_batch/video.sof must be loaded onto the DE1-SoC.

The first thing image_reader.py does is load the shared object file so that it can call the C function it defines, read_image(). This function returns img, an integer array of size 224 x 224 that it reads from the memory-mapped image_reader Qsys component. To read image_reader on Linux, we have to map the physical address defined in the Qsys file to a virtual address. Then for each pixel, we write the pixel number that we want then read its data. As shown in Appendix A, the DCT output for that pixel will be in the upper 16 bits of the integer, and the Y channel will be in the lower 8 bits. After all 224 x 224 pixels are read, we write a value that is outside the domain of desired pixel numbers [0, 224 x 224) to resume registering the latest camera data.

After releasing the physical memory mapping and closing the file descriptor for /dev/mem, we return img.

Back in the Python script, img is a NumPy array of type c_int. We split img into its 8 x 8 blocks, and encode the first one since that is what our hardware currently supports. The software can encode all the blocks with just a change of a number. The encoding scheme is described in Chris' report. The encoded data is posted as a JSON object to the server via the image endpoint.

Audio


Read and process audio data in the HPS and send it to the server
send_recording_to_server.py
audio.py
Initialize
Read image
Send to server
Start
audio.open_dev()
audio.init() audio.sampling_rate(8000)
For _ in [0, 8000*30)
audio.wait_read() audio.read()
Write transformed audio to wav_data
Post wav_data to audio endpoint
End
Open the digital audio device
Initialize audio device and set sampling rate to 8000 Hz
Read Mic In port every 1/8000 seconds

Before running the Python script, the audio drivers need to be inserted into the Linux Kernel (insmod ~/Linux_Libraries/drivers/audio.ko). Then I use the Python API in ~/Linux_Libraries/PyDe/audio.py to record audio from the Mic In port.

The first thing that send_recording_to_server.py does is open the audio device, initialize it, and set it to a sampling rate of 8000 Hz. Then it records raw data for 30 seconds using audio.wait_read() and audio.read(). To minimize the number of bits that need to be sent to the server, we use the lowest sampling rate, and save only one channel to wav_data. Furthermore, the Google Cloud Platform speech-to-text feature requires mono-channel audio. We divide the raw data from the audio device by a constant that we experimentally determined to produce the best recordings with the microphone we use. This data is then posted to the server as a JSON object to the server via the audio endpoint.

Challenges

Image signal degradation

The main issue that plagued the hardware component of our project was the image signal degradation that resulted from adding a Qsys file to the project. Certain colors could not be displayed, and this would vary from compilation to compilation.

The first Qsys file I tried was from the DE1-SoC Computer System project in the Monitor Program. I was met with an error during fitting: the fitter was trying to place 2 PLL's at the same location (see Appendix B for the error message). I resolved it using the ALTCLKCTRL component as suggest in this forum answer, but noticed that these PLL's were using the same global clock, which lead to me to discover that the error could be resolved simply by using different global clocks (e.g., CLOCK2_50 and CLOCK3_50).

When the project compiled successfully, I was met with the image signal degradation issue. I first thought this was a timing error, and tried adjusting the design constraints for the relevant clocks with no success. I also tried removing components from the Qsys file, and using my own, to isolate which component was causing the problem, but that didn't help either. What was particularly frustrating was that some of my compiles had no signal degradation, but when I recompiled the same configuration signal degradation appeared! Out of the 50+ compiles I did, only 2 of them had no signal degradation.

After multiple days with no progress, I had to make the tough decision of moving on. I realized that this type of signal degradation would not significantly affect our eye contact machine learning model's performance, as we were giving it a grayscale image as input. We later verified this hypothesis. Luckily one of our last compiles had no signal degradation, so we used that SOF file for our demo. This SOF did not perform DCT in hardware.

Compilation time

Another challenge we faced was the time it takes to compile our Quartus project. With both the image and audio features, the project took around 40 minutes for my laptop to compile. Thus, I decided to create 2 SOF files: one with the image features and the other with audio. This cut down compilation time of each project to under 20 minutes, which significantly sped up the development process of each feature. In our demo, the camera is connected to the DE1-SoC running the image SOF, and microphone to the one running the audio SOF. Since we send data to the server using HTTP POST requests, nothing on the server had to be changed. If we had more time, I would have recombined the audio and video features into one project and compiled it one last time to have everything working on one DE1-SoC.

We encountered another compilation issue when trying to perform DCT on the entire image. Quartus got stuck on the fitting stage, and did not finish even when I ran it overnight. If it ever would have completed it would most likely give a fitter error saying that our design cannot fit on the DE1-SoC. I helped Chris significantly reduce the hardware required by the DCT module by switching the decimal representation from 32-bit fixed point to a custom representation. With these changes we were able to compute DCT on a single block, but not for the entire image. I think that storing pixels in 8 x 8 block order rather than in entire image order would have significantly reduced the logic required to compute the DCT of an entire image, and would be the first thing I would try if I revisited the project.

Team Effectiveness

Looking back on this project, I am very proud of what we were able to achieve and overcome given the resources, time, and prior experience that we had. Although we were not able to fulfill all the goals we had outlined in the proposal, I believe we made smart decisions about what to sacrifice that would minimally affect our project's functionality and impact. Throughout the project I felt like we had good team dynamics: we made decisions together so that everyone was on the same page, and were open to giving and receiving help. I also had several key takeaways from this experience.

The importance of the design phase of the hardware design process

My approach to designing software involves trying out a bunch of different implementations to see what works best, then rapidly and iteratively refining that implementation. I initially began the hardware component using this brute force software design approach, and it ended up costing me a lot of time. I should have spent more time in the design phase to compare and refine different implementations before I implemented them, as hardware takes longer to write and compile, so the cost of failure is higher.

Fully understand the code which you use or adapt

Initially, I was compiling the demonstration projects and adapting them before taking the time to read through the documentation and code. This might be okay if the task at hand is well-defined and isolated from the existing codebase, but still assumes that the existing codebase is designed and tested well. Since I was merging code from 2 existing codebases, I came across a lot of bugs that were difficult to find the root cause of. I should have taken a more iterative approach of reading and understanding code segments before using or adapting them, or using them as a reference to write my own code. However, starting from scratch would take more time, which brings me to my next takeaway.

Consider feasibility more when project planning

I think we were all so excited about the project idea that we did not take into account if our goals were possible given the resources, time, and prior experience we had as much as we should have. As a consequence, we stretched ourselves a bit thin and were not able to fulfill all our proposal goals. If we limited our project scope more, we would have been able to deliver a more refined end product.

Know when to give up on an approach

It can be tough for me to give up on an approach, as it means accepting defeat. When I get in the zone, I can always think of more things to try, but can ultimately end up wasting a bunch of time. Moving forward, I aim to take breaks more frequently so that I can re-evaluate problems from a higher perspective, allowing me to better decide what to do next.

Acknowledgements

- Custom Qsys Component: [CPEN 391 2019W T2 Tutorial 1.8]
- D8M-GPIO Demonstration Project: [D8M-GPIO CD-ROM]
- DE1-SoC Computer System: [Monitor Program]
- fpga/SevenSegmentDisplayDecoder.v: CPEN 311 2021W T1 labs
- Audio drivers: [Intel FPGA Academic Program Linux SD Card Images]

Appendix

Appendix A

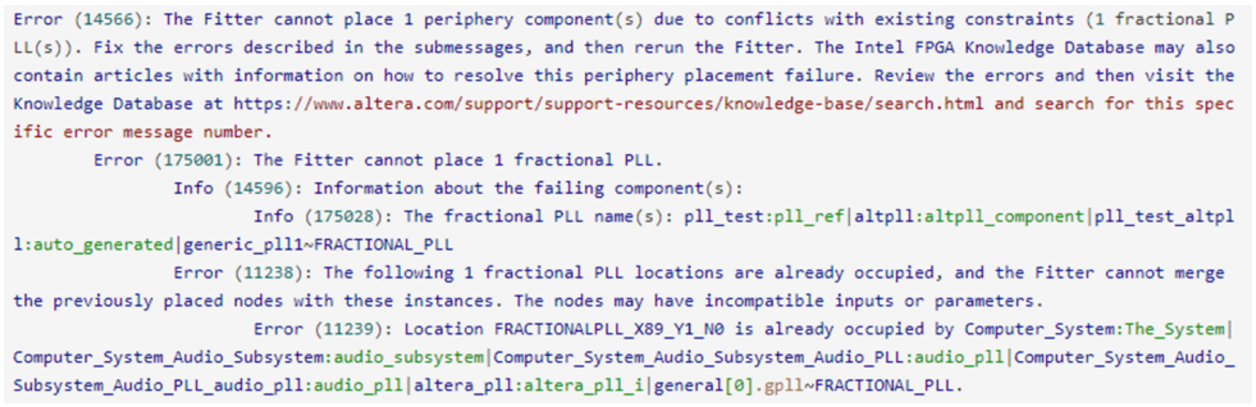
image_reader Qsys Component Output

addr	readdata
0	16 bits dct_out, 8 bits 0, 8 bits img
1	16 bits blockUpperLeft, 8 bits blockUpperLeftX, 8 bits blockUpperLeftY
2	6 bits 0, 1 bit dpibb, 1 bit pibb, 8 bits block_ind, 8 bits desiredX, 8 bits desiredY
3	32 bits desired_pixel

dpibb = desired_pixel_in_bounding box; pibb = pixel_in_bounding_box

Appendix B

Fitter Error



```
Error (14566): The Fitter cannot place 1 periphery component(s) due to conflicts with existing constraints (1 fractional P
LL(s)). Fix the errors described in the submessages, and then rerun the Fitter. The Intel FPGA Knowledge Database may also
contain articles with information on how to resolve this periphery placement failure. Review the errors and then visit the
Knowledge Database at https://www.altera.com/support/support-resources/knowledge-base/search.html and search for this spec
ific error message number.
        Error (175001): The Fitter cannot place 1 fractional PLL.
                Info (14596): Information about the failing component(s):
                        Info (175028): The fractional PLL name(s): pll_test:pll_ref|altpll:altpll_component|pll_test_altpl
l:auto_generated|generic_pll1~FRACTIONAL_PLL
                Error (11238): The following 1 fractional PLL locations are already occupied, and the Fitter cannot merge
the previously placed nodes with these instances. The nodes may have incompatible inputs or parameters.
                        Error (11239): Location FRACTIONALPLL_X89_Y1_N0 is already occupied by Computer_System:The_System|
Computer_System_Audio_Subsystem:audio_subsystem|Computer_System_Audio_Subsystem_Audio_PLL:audio_pll|Computer_System_Audio_
Subsystem_Audio_PLL_audio_pll:audio_pll|altera_pll:altera_pll_i|general[0].gpll~FRACTIONAL_PLL.
```

Christopher Chang Individual Report

Student Number: 27150887

Contribution

For the VIS, I researched and identified machine learning models that could score a user's eye contact. This model was adapted into a server-side script that scored eye contact using an image received from the DE1. Next, I implemented an image encoding algorithm to process image data before the DE1-SoC sends it to the server. I spent a majority of my time and effort exploring different implementation options and creating an efficient and accurate algorithm. See the Detailed Design section for the final implementation and the Challenges section for implementation options I explored. I also worked with Patrick Creighton to explore ways to optimize the algorithm. Furthermore, I also implemented a script to run image decoding on the server with the help of Daniel Lee. This script reconstructs an image received from the DE1-SoC. Finally, I modified the eye contact machine learning model to take in the reconstructed image and streamlined its performance.

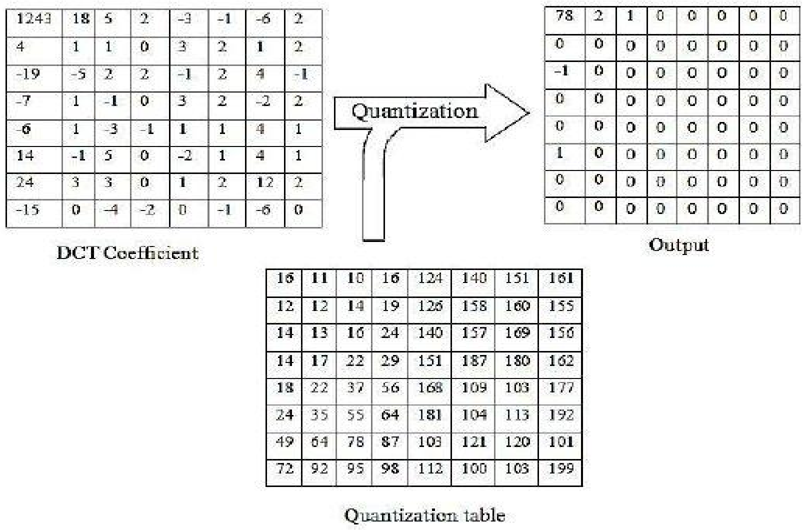
Detailed Design

This section provides an outline of each task, background information, an overview of the task's implementation, and the value it brought to the project.

Image Encoding Algorithm

The image encoding algorithm consists of a hardware component and software component. The algorithm's hardware component is a SystemVerilog module that performs lossy compression on the top left 8 x 8 portion of an image taken by the D8M-GPIO camera. The reason compression is performed on an 8 x 8 region instead of a full 224 x 224 image is covered in the Challenges section. After compression is completed, the compressed output is sent to the software component, which encodes the data using a Python script running on the DE1-SoC HPS's Linux distribution. Lastly, it's sent to the backend server using an HTTP request and serves as an input to the eye contact machine learning model. The main benefit of this algorithm is that it reduces the amount of data sent in an HTTP request body while still maintaining the ability to reconstruct and analyze the image on the server.

Because my own compression algorithm shares some similarities with JPEG, a brief description of JPEG is as follows. JPEG splits an image into 8 x 8 non-overlapping blocks called Minimum Coded Units (MCUs). For each MCU, the discrete cosine transform (DCT) is computed and its result is quantized using predetermined coefficients. These two steps are able to capture the most important features of an MCU. Afterwards, the quantized DCT results undergo a value run-length encoding along a zig-zag pattern and these value run-lengths are further encoded using Huffman codes. An example of a quantized DCT output is given below in Figure 1 and the zig-zag pattern used to encode value run-lengths is shown in Figure 2.



1243	18	5	2	-3	-1	-6	2
4	1	1	0	3	2	1	2
-19	-5	2	2	-1	2	4	-1
-7	1	-1	0	3	2	-2	2
-6	1	-3	-1	1	1	4	1
14	-1	5	0	-2	1	4	1
24	3	3	0	1	2	12	2
-15	0	-4	-2	0	-1	-6	0

DCT Coefficient

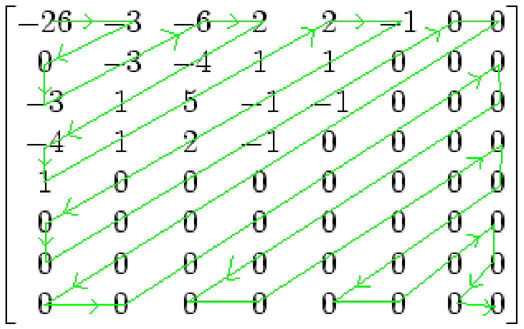
78	2	1	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Output

16	11	10	16	124	140	151	161
12	12	14	19	126	158	160	155
14	13	16	24	140	157	169	156
14	17	22	29	151	187	180	162
18	22	37	56	168	109	103	177
24	35	55	64	181	104	113	192
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	199

Quantization table

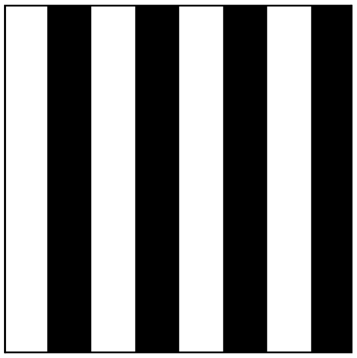
Figure 1: A quantized DCT output computed by dividing each element of the DCT coefficient output matrix with the corresponding element in the quantization table, From Research Gate. Date Accessed: April 13, 2022.



$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -3 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2: The JPEG's zigzag ordering used for value run-length encoding. From Wikimedia Commons. Date Accessed: April 13, 2022.

In my own compression algorithm, I created a combinational logic SystemVerilog module capable of computing the DCT for an MCU and performing quantization. The input to the module is a row-major ordered MCU, where each element is 8 bits long, and the output is a row-major ordered quantized DCT, where each element is 16 bits long. Through simulation and comparison with expected results, I found that my results were able to closely approximate both the DCT and quantization results. In testing, expected DCT and quantization outputs were computed for the MCU shown in Figure 3. Figure 4 compares expected outputs with the outputs produced by the SystemVerilog module.



k_1/ k_2	0	1	2	3	4	5	6	7
0	255	0	255	0	255	0	255	0
1	255	0	255	0	255	0	255	0
2	255	0	255	0	255	0	255	0
3	255	0	255	0	255	0	255	0
4	255	0	255	0	255	0	255	0
5	255	0	255	0	255	0	255	0
6	255	0	255	0	255	0	255	0
7	255	0	255	0	255	0	255	0

Figure 3: The test MCU (Left) and its luminance matrix representation (Right). The matrix is zero-indexed by the variables $k_1$and k_2. Date Created: April 13, 2022.

k_1	k_2	Actual DCT	SV DCT (Base 16)	SV DCT	Difference
0	0	8160	1FE0.00	8160	0
0	1	1039.9829	41B.E0	1051.875	+11.8921
0	2	~0	0	0	~0
0	3	1226.7435	4BB.40	1211.25	-15.4935
0	4	~0	0	0	~0
0	5	1835.9514	728.D0	1832.8125	-3.1389
0	6	~0	0	0	~0
0	7	5228.3475	145B.90	5211.5625	+16.785

Figure 4: Comparison of actual DCT and SystemVerilog (SV) DCT outputs for $k_1 = 0$. Values are in Base 10 unless otherwise specified. For all entries where $k_1 > 0$, actual DCT and SV DCT are zero. See Appendix A for the SV DCT waveforms. Date Created: April 13, 2022.

As seen in Figure 4, the SystemVerilog module can approximate and capture the relative magnitudes of the DCT. Quantization is now applied and its results are compared in Figure 5.

k_1	k_2	Actual DCT	Q. Factor	Actual Q. DCT	SV Q. DCT	Difference
0	0	8160	16	510	510	0
0	1	1039.9829	11	94.5439	95	+0.4561
0	2	~0	10	~0	0	~0
0	3	1226.7435	16	76.6714	75	-1.6714
0	4	~0	24	~0	0	~0
0	5	1835.9514	40	45.8987	45	-0.8987
0	6	~0	51	~0	0	~0
0	7	5228.3475	61	85.7106	85	-0.7106

Figure 5: A table of actual quantized DCT outputs, quantization factors (Q Factors) and SystemVerilog quantized DCT (SV Q. DCT) outputs for $k_1 = 0$. Values are in Base 10 unless otherwise specified. For all entries where $k_1 > 0$, actual quantized DCT and SV quantized DCT output are 0. Date Created: April 14, 2022.

After quantization, it is seen that the SystemVerilog compression module closely approximates the expected quantization and DCT results. This shows the correctness of the compression algorithm.

The lossy compression SystemVerilog module is instantiated within a Qsys component that captures and arranges camera data into a row-major ordered NumPy array. Afterward, a Python algorithm encodes the results using a simple run-length encoding. The encoding takes advantage of the tendency that a quantized DCT output, when converted to a row-major ordered list, has a large amount of zeros at the end. Thus, the encoding I designed is a run-length encoding for those trailing zeros. Furthermore, for decoding purposes, a marker value of -391 surrounds the run length value. An example of this encoding is demonstrated below in Figure 6.

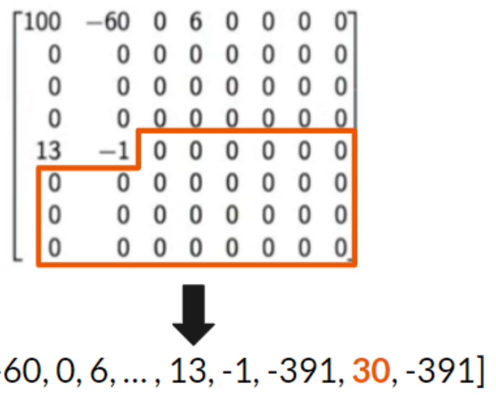


Figure 6: A sample image encoding output used within the VIS. The red highlighted area shows the 30 trailing zeros that are encoded. The intermediate step of forming a row-major ordered list is omitted for clarity. DCT Matrix from ENGEGY (Youtube). Date Accessed: April 13, 2022

Image Decoding Algorithm

As the image is encoded, image decoding is needed to reconstruct the image. The reconstruction is performed on the server using a Python script that I wrote. This script is triggered when a HTTP POST request to the server containing encoded image data is sent to the server. After reconstruction, the image is inputted into a machine learning eye contact model. Thus, the decoding algorithm plays an important role in running the eye contact machine learning model using inputs from our camera and supports the goal of reducing the amount of data needed to represent an image.

Reconstruction consists of undoing the steps of the image compression algorithm. First, the run-length encoding of trailing-zeros is decoded by parsing the data until it encounters the -391 marker value. Second, the row-major ordered list of data is reordered into an 8 x 8 matrix. These first two steps are accomplished using NumPy operations to identify markers and to reshape data into an 8 x 8 matrix. Third, the inverse DCT is computed using the SciPy-FFTPack library and this produces a representation of a reconstructed image.

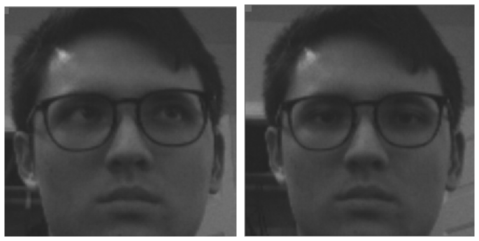


Figure 7: Reconstructed images produced by the image decompression algorithm and viewed using the Python Pillow library. Date Created: April 13, 2022

Eye Contact ML Research, Adaptation, Server Deployment

To implement the eye contact scoring feature, I conducted research and found a convoluted neural network (CNN) Python script that scored eye contact for videos. This script iterated through frames and ran a Dlib face detection model to identify human faces. Next, face images were rescaled to 224 x 224 images and transformed into three 224 x 224 tensors. Each tensor corresponds to one of the red, green, and blue channels. These tensors then served as the input to the eye contact CNN.

When adapting the existing CNN to a server-side script, I first removed the loop that iterates through video frames and then removed the Dlib face detection dependency. Including Dlib added a large computational overhead and caused the CNN to generate scores slowly. Next, I adapted the model to work with grayscale images from the camera. This involved modifying the reconstructed image to adhere to the interface of the CNN. This was done using NumPy and PyTorch to construct a tensor with the required red, green, blue channels.

Developing the server-side script also required installing the necessary dependencies on the AWS EC2 instance. This includes NumPy, PyTorch, TorchVision, and SciPy.

Challenges

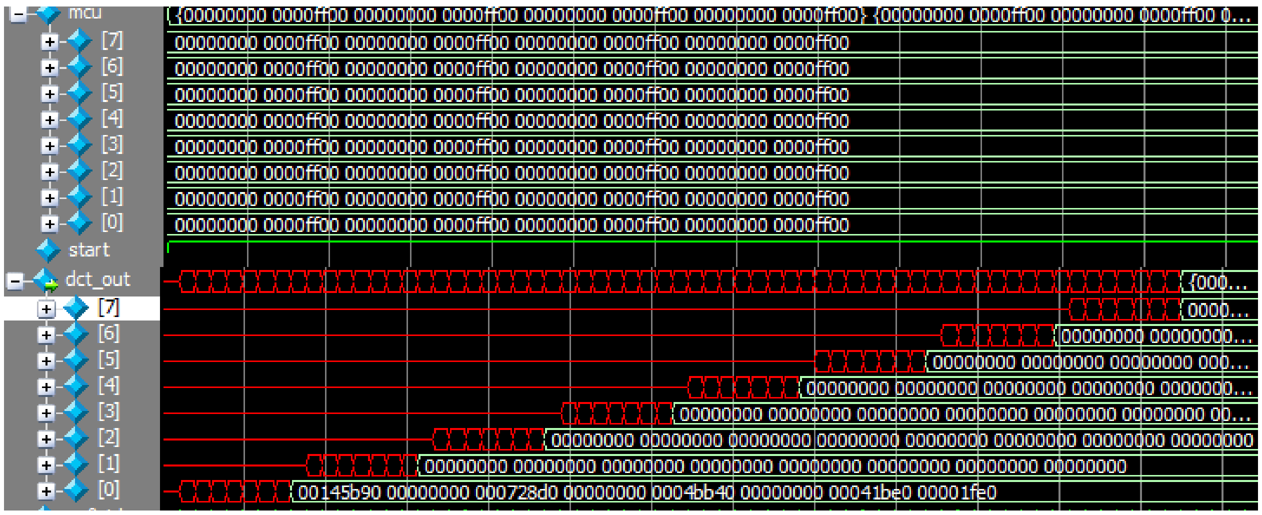
Correctness and Efficiency of DCT

The first challenge I encountered in this project was the development of a correct and efficient DCT algorithm using SystemVerilog. The discrete cosine transform is defined in Figure 8. In this case, $N_1 = N_2 = 8$ since DCT is computed for 8 x 8 blocks. Furthermore, this summation must be evaluated for all integers k_1 and k_2 in the range $[0, 7]$ where k_1 is a row index and k_2 is a column index. The result is an 8 x 8 matrix.

$$\begin{aligned} X_{k_1,k_2} &= \sum_{n_1=0}^{N_1-1} \left(\sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N_2}\left(n_2 + \frac{1}{2}\right) k_2\right] \right) \cos\left[\frac{\pi}{N_1}\left(n_1 + \frac{1}{2}\right) k_1\right] \\ &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N_1}\left(n_1 + \frac{1}{2}\right) k_1\right] \cos\left[\frac{\pi}{N_2}\left(n_2 + \frac{1}{2}\right) k_2\right]. \end{aligned}$$

Figure 8: The Discrete Cosine Transform definition. From *Wikipedia*. Date Accessed: April 13, 2022

My first DCT implementation had two nested FSMs working to compute the DCT on a single MCU. The outer FSM iterated through all k_1 and k_2 while the inner FSM performed the double summation. Despite producing correct results matching that of Figure 4, this FSM implementation was inefficient since it required many clock cycles to produce the DCT output. To put things in perspective, at least 64 clock cycles were needed for each of the 64 values in the output matrix. Thus, a combinational logic implementation was adopted to ensure the DCT output could be computed before the VGA's image data was overwritten.



```
mcu      {00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00} {00000000 0000ff00 00000000 0000ff00 0...
  [7]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [6]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [5]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [4]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [3]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [2]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [1]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [0]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
start
dct_out  {000...
  [7]    0000...
  [6]    00000000 00000000...
  [5]    00000000 00000000 00000000 000...
  [4]    00000000 00000000 00000000 00000000 0000000...
  [3]    00000000 00000000 00000000 00000000 00000000 00000000 00...
  [2]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [1]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [0]    00145b90 00000000 000728d0 00000000 0004bb40 00000000 00041be0 00001fe0
```

Figure 9: Results of the DCT FSM under the packed bus named *dct*, before a row-major order implementation was decided on, which requires many clock cycles to compute. Note all values are in Q24.8 fixed point representation. This was later changed to Q16.8.

A Verilog Implementation of Cosine

One challenge in programming a correct DCT algorithm was implementing a cosine function using SystemVerilog. I explored the following options before deciding to precompute all the cosine terms for the lossy compression module.

Maclaurin Approximation

The first option I explored was a Maclaurin approximation for the cosine function. To determine how many terms were required to produce an accurate approximation, I considered the largest cosine argument used in the DCT. Let $x = [\frac{\pi}{N_2}(n_2 + \frac{1}{2})k_2]$ for this analysis of the cosine term in Figure 10.

$$\cos\left[\frac{\pi}{N_2}\left(n_2 + \frac{1}{2}\right)k_2\right]$$

Figure 10: A cosine term in the Discrete Cosine Transform definition. From *Wikipedia*. Date Accessed: April 13, 2022

The largest cosine argument occurs when $k_2 = 7$ and $n_2 = 7$. It follows that the largest $x \approx 20$. However, after using Desmos to incrementally add terms to the Maclaurin Approximation, I found that it would require far too many terms to get an accurate result for the entire range of inputs to the cosine function. Moreover, the higher order terms such as $\frac{x^{14}}{14!}$, the seventh non-zero term, would result in an overflow for the Q16.8 fixed point representation we used to represent numbers. In Figure 11, a seven term Maclaurin approximation of cosine is shown to only be accurate within the range $[-6, 6]$ whereas the largest input is $x \approx 20$.

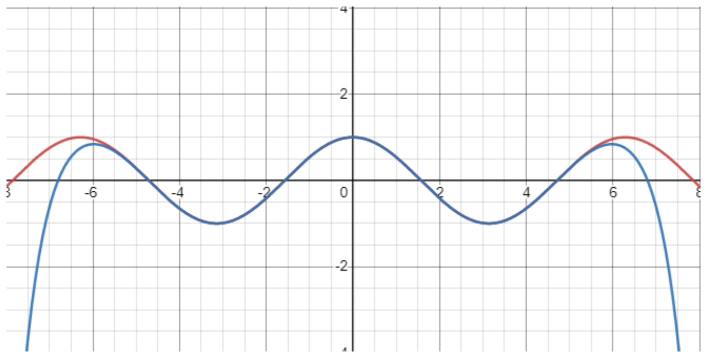
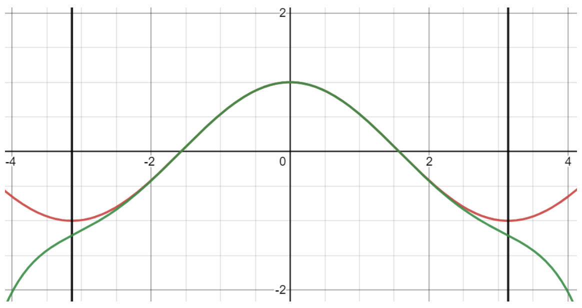


Figure 11: A graph of $y = cos(x)$ (Red) and $y = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!}$ (Blue). Date Created: April 13, 2022

Range Restricted Maclaurin Approximation

The previous investigation of using a Maclaurin approximation showed that it would require many terms to produce an accurate result for larger inputs to the cosine function. This led me to take advantage of the periodic nature of cosine. For every angle, there is an equivalent angle within the range $[-\pi, \pi]$. This range was more appropriate than $[0, 2\pi]$ due to the usage of the Maclaurin approximation, which is centered at zero. For comparison, only four terms are required for a good approximation of cosine within the range $[-\pi, \pi]$ (Figure 12). However, up to eight terms are needed for a good approximation of cosine within the range $[0, 2\pi]$ (Figure 12).

Although this idea seemed promising, finding equivalent angles for inputs to cosine would likely be implemented using an FSM. I decided against this for the same reason a combinational logic DCT implementation was favoured over a DCT FSM. I wanted to avoid any clock cycle overhead and ensure the DCT could be computed before the VGA's image data was overwritten.



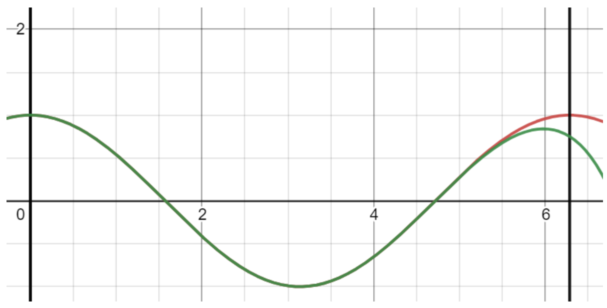


Figure 12: $y = cos(x)$ (Red) and a four-term Maclaurin approximation of cosine (Green) for the range $[-\pi, \pi]$ (Left). $y = cos(x)$ (Red) and an eight-term Maclaurin approximation of cosine (Green) for the range $[0, 2\pi]$ (Right). Date Created: April 13, 2022

CORDIC Algorithm and Final Decision: Precomputing Cosine Terms

Next, I explored using the CORDIC algorithm, which computes cosine and sine for arbitrary angles. The range of acceptable inputs for CORDIC is $[-\frac{\pi}{2}, \frac{\pi}{2}]$. This has the same issue as the range-restriction Maclaurin approximation: You must find an equivalent angle between some particular range, which requires clock cycles to compute. Additionally, it was at this point that I recognized that it was possible to precompute the cosine terms. There was no need to use any algorithms to compute cosine for arbitrary angles because cosine terms only depended on k_1, k_2, n_1, and n_2. Thus, they could be precomputed. This has the benefit of not needing any FSMs or clock cycles to find equivalent angles or compute cosine. Ultimately, this was the approach I took for my encoding algorithm.

Scaling the DCT Implementation to a Full Image

The last major problem I encountered was to integrate a full DCT implementation within the image reader Qsys component that read the camera data. The goal was to have an entire 224 x 224 image be encoded, but ultimately we had to scale the encoding and compression to only operate on a single 8 x 8 block. The reason for this decision is that we found compilation of the image reader Qsys component with a DCT quantization module instantiated was unable to finish compiling, even if we compiled the project for many hours or even overnight. We theorize that it was due to the amount of logic required for the adding and multiplication of numbers within the module.

Some methods we tried to reduce the logic needed was trying out different number representations and optimizing the computations done in the DCT operations. However, we were unable to explore other options due to project deadlines. In hindsight, we could have reduced the size of the image to a smaller dimension such as 208 x 208 or 200 x 200 that was taken by the camera and then used software to upscale the image to be the right dimensions.

Team Effectiveness

The overall team dynamic was positive and we worked well with each other. Everyone was willing to spend an exceptional amount of their time and effort toward this project. I noticed that everyone showed the ability to take initiative and was able to listen attentively to others ideas as well. One example of the team's effectiveness is during the integration of individual components. It was a consistent trend that the members would collaborate and communicate their ideas clearly. Because of this, the integration process went smoothly.

A two suggestions I would offer in hindsight and given the knowledge I have now is to have started earlier or to find a more feasible way to do image encoding. Our project, as highlighted in the Challenges section, did not live up to its full potential due to time constraints. Namely, a full image encoding algorithm was not integrated within the final project. Given that I now know that the final DCT implementation is too large to fit on the DE1-SoC and required long compilation times, we could have planned around that and explored different image processing options. Starting earlier could have also prevented us from running out of time by allowing us more flexibility and buffer time to change our image encoding approach once we discovered that this was not feasible.

References and Acknowledgments

Chong, E., Clark-Whitney, E., Southerland, A., Stubbs, E., Miller, C., Ajodan, E. L., Silverman, M. R., Lord, C., Rozga, A., Jones, R. M., & Rehg, J. M. (2020). Detection of eye contact with deep neural networks is as accurate as human experts. *Nature Communications*, *11*(1). https://doi.org/10.1038/s41467-020-19712-x

Chong, E. (2020, October 16). *REHG-Lab/eye-contact-CNN: Deep Neural Network trained to detect eye contact from facial image*. GitHub. Retrieved April 13, 2022, from https://github.com/rehg-lab/eye-contact-cnn

ENGEGY. (2016). *54- The JPEG compression algorithm*. Retrieved April 13, 2022, from https://www.youtube.com/watch?v=aFbGqXFT0Nw&t=894s.

Jclin. (2005, September 8). *JPEG ZigZag Pattern* . Wikimedia Commons. Retrieved April 13, 2022, from https://commons.wikimedia.org/wiki/File:JPEG_ZigZag.svg

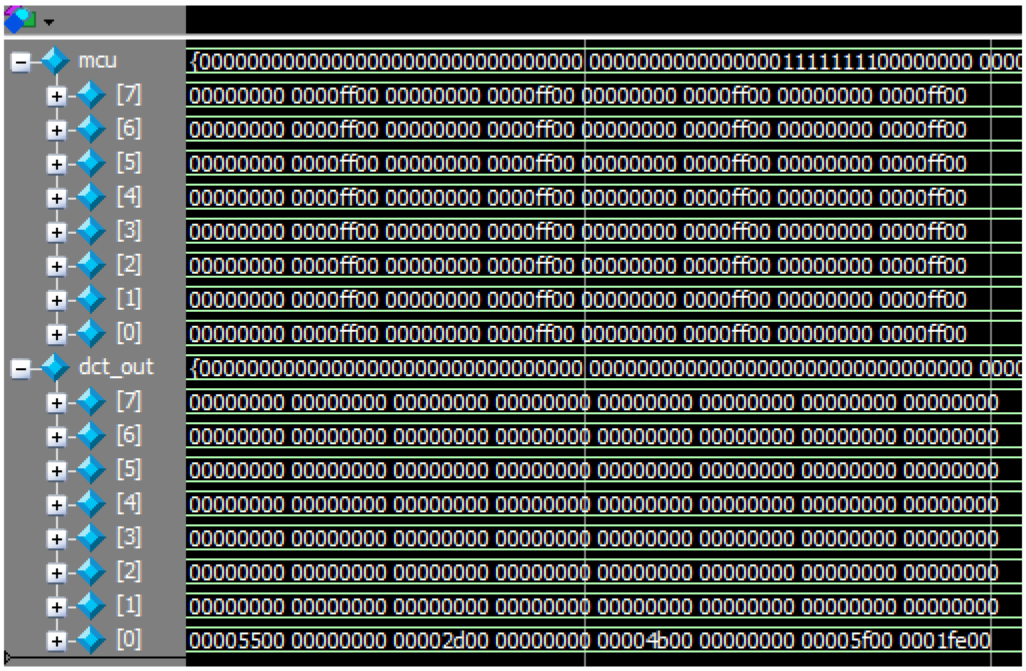
Makki Sagheer, A., Suliman Farhan, A., & E. George, L. (2013). Fast intra-frame compression for video conferencing using adaptive shift coding. *International Journal of Computer Applications*, *81*(8), 29–33. https://doi.org/10.5120/14034-1899

Wikimedia Foundation. (2022, April 11). *Discrete Cosine Transform*. Wikipedia. Retrieved April 13, 2022, from https://en.wikipedia.org/wiki/Discrete_cosine_transform

Appendix

Appendix A

DCT Quantization SystemVerilog module simulation waveforms



```
mcu      {00000000000000000000000000000000 00000000000000001111111100000000 0000
  [7]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [6]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [5]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [4]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [3]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [2]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [1]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
  [0]    00000000 0000ff00 00000000 0000ff00 00000000 0000ff00 00000000 0000ff00
dct_out  {00000000000000000000000000000000 00000000000000000000000000000000 0000
  [7]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [6]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [5]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [4]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [3]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [2]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [1]    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
  [0]    00005500 00000000 00002d00 00000000 00004b00 00000000 00005f00 0001fe00
```

Azwad Sadman Individual Report

Student Number: 39442868

Contribution

There were five components for completing a project according to the course guideline. Among them, I was assigned to work on two components, and they were – Cloud and User Interface. Though cloud and user interface were my prime areas of contribution, I also had a bit of contribution in the hardware component. I was responsible for a minor part of the Hardware component. My contributions on the multiple components are described below:

Hardware Component

The tasks that I have worked on for the Hardware component are as follows:

- Modified the audio driver code to get a good quality audio from the mic-in port of De1-SoC

The python script that used the audio drivers to read data from the mic-in port would generate an audio with a very low volume. So, my task was to increase the volume, and write the audio as a WAV format file.

Cloud Component

The tasks that I have worked on for the Cloud component are as follows:

- Speech-to-Text analysis using Google Cloud Speech-to-Text API
- Sentiment analysis using Google Cloud Natural Language API

The audio data from the De1-SoC that was written as a WAV format was fed into the python script of Speech-to-Text and Sentiment Analysis. The python script was entirely developed by me. This script performs the operation of speech recognition at first. After speech recognition, the transcript of the speech along with the confidence score are found as outputs. The transcript is then passed for sentiment analysis. Sentiment analysis produces the sentiment score by analyzing the sentiment of the transcript.

The speech-to-text and sentiment analysis can be considered as high-quality parts of the project. Among the three important features of our project, two of them are speech-to-text and sentiment analysis. The analysis produces two scores based on the confidence of the speech and the sentiment of the text. The confidence score is the measure of accuracy of the response found from speech-to-text analysis. The score is calculated by aggregating the values which were assigned to each word in the audio by the Google Cloud Speech-to-Text API. A higher score indicates that the individual words were recognized correctly. In a word, the score gives an idea about the clarity of the speech. On the other hand, the sentiment score indicates the emotion of sentences in the transcript. The transcript is the response that is received from speech-to-text analysis. Sentiment analysis uses Natural Language API to calculate the sentiment score. There are four types of emotion, and a specific score identifies a specific type of emotion. So, it can be said that the score gives an idea about the type of emotion expressed by the speaker.

User Interface Component

The tasks that I have worked on for the User Interface component are as follows:

- Developed the front-end application using the React framework of JavaScript
- React router to configure the routes and navigate to the links
- React hooks to update the different states of the app
- React axios to perform the HTTP requests
- React material ui to customize the buttons

The React libraries that were used for developing the front-end are Router, Hooks, Axios, and Material UI. The application has four pages – "Login" page, "Home" page, "Question Display" page and "Feedback" page. A stopwatch was implemented in the "Question Display" page to keep track of time for timing the interview. It offered the functions of start, stop and reset. The questions in the "Question Display" page were fetched from the MongoDB database. The feedback in the "Feedback" page was also collected from the MongoDB database.

Detailed Design

The workflow of my design is shown below using a flow-chart:

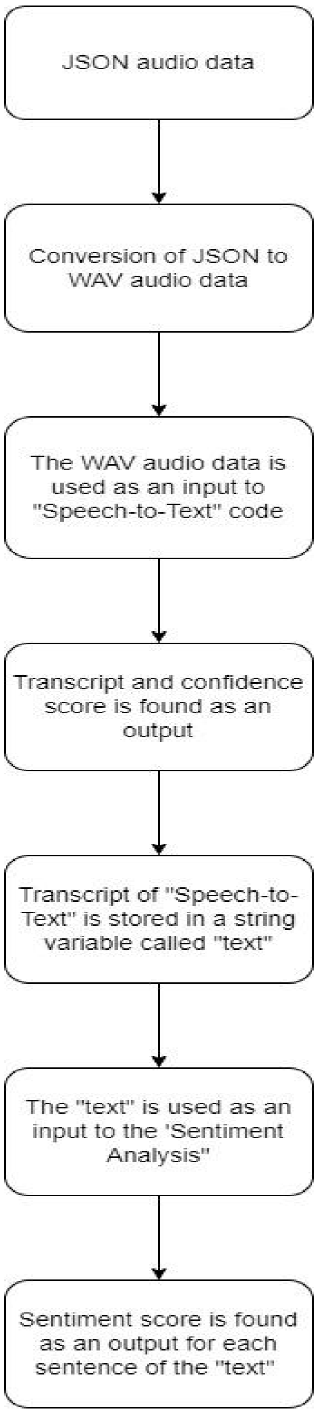

JSON audio data
Conversion of JSON to WAV audio data
The WAV audio data is used as an input to "Speech-to-Text" code
Transcript and confidence score is found as an output
Transcript of "Speech-to-Text" is stored in a string variable called "text"
The "text" is used as an input to the 'Sentiment Analysis"
Sentiment score is found as an output for each sentence of the "text"

Figure 1: Flow-chart of my design

A JSON audio data is sent from the HPS running on De1-Soc board. The audio data was recorded in 16-bit short integer format. It helped to boost the volume as well. The JSON data made the python script easily executable on the server. It was easy to send JSON data to the server rather than a WAV file because JSON data are commonly used for exchanging data to/from a server. My script was provided with the code to convert the JSON data to a WAV data. The JSON data was converted to WAV data using the "BytesIO" package of Python. The advantage of using the "BytesIO" package was that it is an in-memory buffer, so nothing is written onto the disk. In one way, it helped to speed up the conversion from JSON to WAV and in another way, it helped to save some space on the disk. While writing to WAV data, the number of channels was set to one because Google Speech-to-Text API requires a mono-channeled audio. Moreover, the frame rate was set to 8000 because the python audio driver was designed to record audio with this frame rate. The newly produced WAV file was fed into the speech-to-text and sentiment analysis code afterwards. In a nutshell, one python script was used for three different operations:

1. Conversion from a JSON to a WAV data
2. The WAV data was used as an input to the "Speech-To-Text" code and outputted transcript of the WAV file along with the confidence score
3. The transcript was then finally passed to the "Sentiment Analysis" code which extracted sentences from the transcript of "Speech-To-Text" and calculated the sentiment score for each sentence

The detailed design of the three components speech-to-text, sentiment analysis and user interface are described below:

Speech-to-Text

Speech-to-Text feature uses a Speech-to-Text API that is powered by Google's AI technologies. It applies the most advanced deep learning neural network algorithms for automatic speech recognition. There are three main methods for performing speech recognition and they are –

- Synchronous Recognition – speech recognition is limited to audio data of 1 minute or less in duration

- Asynchronous Recognition – speech recognition is possible for audio data of any duration up to 480 minutes
- Streaming Recognition – speech recognition is done for real time audio data from microphone

The synchronous recognition was implemented in our project. This means that the user will get only one minute for answering each question while using our product to practice interviews. A synchronous speech-to-text API request consists of a speech recognition configuration and audio data. The speech and audio recognition configurations are known as “RecognitionConfig” and “RecognitionAudio” respectively.

The “RecognitionConfig” contains the following sub-fields:

- encoding – specifies the encoding scheme of the supplied audio of type “AudioEncoding.” The supported audio encodings are MP3, FLAC, LINEAR16 and so on. For this project, I have used the encoding scheme of MP3 which is shown below. It supports all standard MP3 and WAV bitrates.
- sample_rate_hertz – specifies the sample rate of the supplied audio. The sample rates range from 8000 Hz to 48000 Hz. The supplied audio is written at 8000 Hz so the sample_rate_hertz of 8000 Hz is used for this project.
- language_code – specifies the language and the region to use for speech recognition of the supplied audio. The language_code of “en-US” is used for this project. This language code is an identifier of type BCP-47 where ‘en’ is for ‘English’ language and ‘US’ stands for ‘United States’ as the specified region.
- enable_automatic_punctuation – specifies the punctuations in the text from the speech. It is an optional field of “RecognitionConfig.” When it is set to ‘true’ then the API automatically detects the punctuation marks in the converted text from the speech.

Audio data was supplied to Speech-to-Text API through the “audio” parameter of type “RecognitionAudio.” The “audio” field contains the following sub-field:

- content – specifies the audio to evaluate for speech recognition. It was used in this project to specify the audio needed for speech recognition.

- uri – contains a Google Cloud Storage URI which points to the audio content. It is used when the audio is stored in the cloud storage.

After setting the configurations for speech and audio, the recognition method for speech-to-text has to be selected. The method that was selected was "long_running_recognize." It accepts requests for both synchronous and asynchronous recognition. It has two fields, and they are "config" and "audio". RecognitionConfig is passed to the "config" field and "RecognitionAudio" is passed to the "audio" field. After that, the "result" field of "long_running_recognize" was invoked for obtaining the output of the speech recognition by iterating over the audio data.

Lastly, transcript and confidence score were the outputs received from this API after speech recognition was performed. The output contains the following fields for displaying the results obtained from speech recognition:

- transcript – contains the transcribed text of the speech
- confidence – specifies a value between 0.0 and 1.0 which indicates the confidence of the speech-to-text transcription. Higher score indicates that the words of the speech were recognized correctly.

One of the results from the "Speech-to-Text" was the transcript of the speech of the supplied audio. This transcript was used to perform the "Sentiment Analysis" which is the other feature of our project.

Sentiment Analysis

The Natural Language API of Google has several methods for performing analysis on the text. Sentiment Analysis is one of the methods for extracting valuable information for language understanding. It uses the Natural Language API from Google Cloud Platform to inspect the given text and identify the prevailing emotional opinion within the text. The emotional opinion is identified on the basis of the following emotions – positive, negative, neutral and mixed. Finally, a sentiment score is calculated based on the prevailing emotion.

Sentiment Analysis is performed by using the "analyze_sentiment" method of the Natural Language API. It accepts the following two fields:

- document – contains the data for performing the sentiment analysis request. It consists of the following sub-fields:

  - type – specifies the type of the document. The “type” of the document can either be HTML or PLAIN_TEXT. I have used the PLAIN_TEXT format for the project because the text data was a plain text that was received from Speech-to-Text.

  - language – specifies the language of the document. This sub-field is optional. When it is not specified then the language is automatically detected by the API. Unsupported languages will return an error. I have specified the English language in this sub-field.

  - content – specifies the text to be evaluated. The transcript from Speech-to-Text was stored in a string variable and passed as the content for sentiment analysis.A string variable called “text” was created to store the transcript from speech recognition. First of all, the variable “text” was initialized with empty string. Then a for loop was run over the results from the response of the speech recognition. Each loop generated a sentence for the transcript and was added into the variable “text.” Lastly, this “text” variable was passed to the “content” sub-field of document to perform sentiment analysis.

- encoding_type – specifies the encoding scheme to process the output. There are four types of encoding schemes available for performing the sentiment analysis request - NONE, UTF8, UTF16, and UTF32. It was recommended to avoid using “NONE” because the encoding dependent information was set at -1 if “NONE” was used. I have used the encoding_type of UTF8 because it accepts python data.

Finally, the analyze_sentiment was called after passing all the required fields for performing the sentiment analysis. The output contains the following fields for displaying the results obtained from sentiment analysis:

- text – each sentence from the text is extracted and printed on the user interface

- sentiment score – specifies a value for the type of emotion analyzed from the sentences in the transcript. The type of sentiments with the sample score values are mentioned in the table below –

Sentiment	**Sample Score**
Positive	0.8
Negative	-0.6
Neutral	0.1
Mixed	0.0

It is to be noted that the API suggests the differences between positive and negative emotion rather than specifically identifying the positive and negative emotions. For example, the emotions "sad" and "angry" are considered as negative emotions. A transcript with a neutral score will indicate a response with low emotion. When the positive and negative values cancel each other, then the type of sentiment is known as mixed sentiment.

While calculating the sentiment, the first step was to separate each sentence from the entire transcript. Then the Natural Language API was used to calculate the sentiment score of each sentence separately. It helped to analyze the sentiment of each sentence separately. This will enable any user practicing with our product to be able to practice sentences with more positive emotion.

User Interface

A front-end application was developed as part of the user interface component. It was implemented using the React framework. The user interface of the "Login" page, "Home" page, "Question Display" page, and "Feedback" page are displayed below:

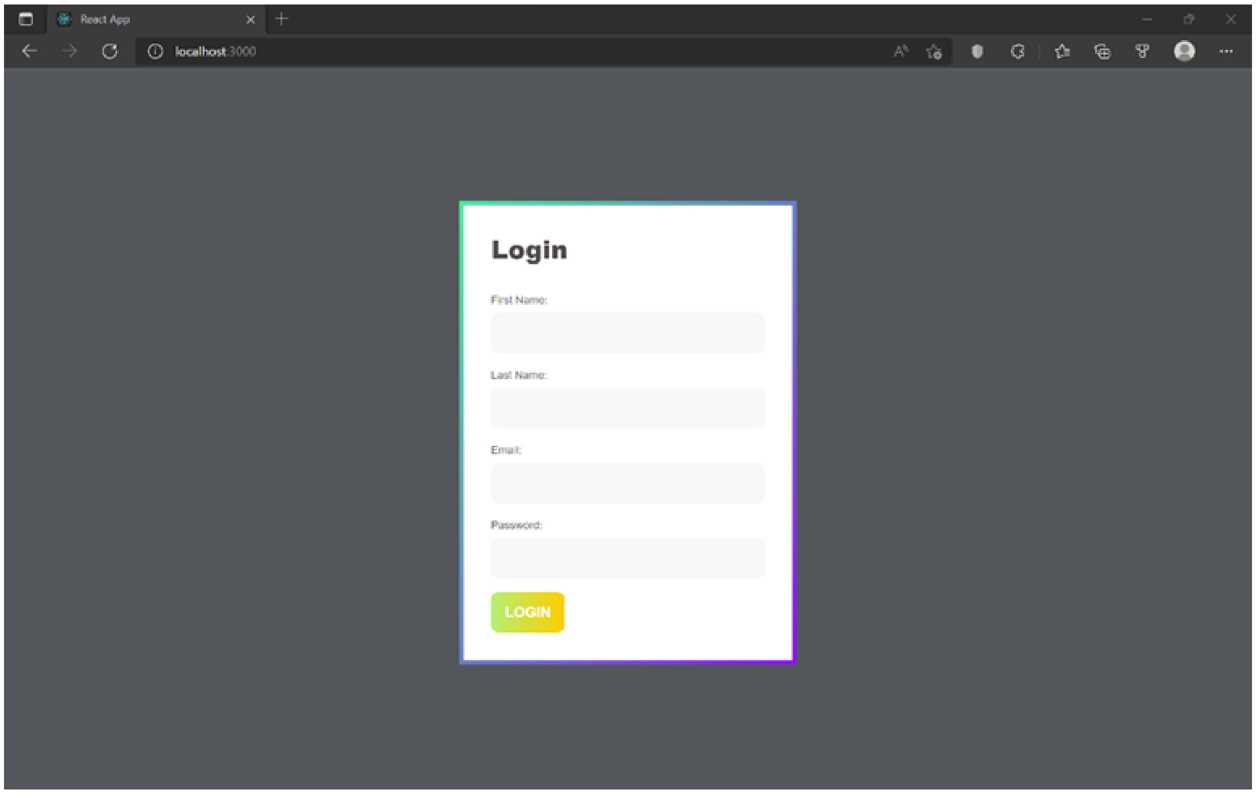


Figure 2: Login page

A user can login by using the login form. The authentication of the users was not implemented so it supports only one user.

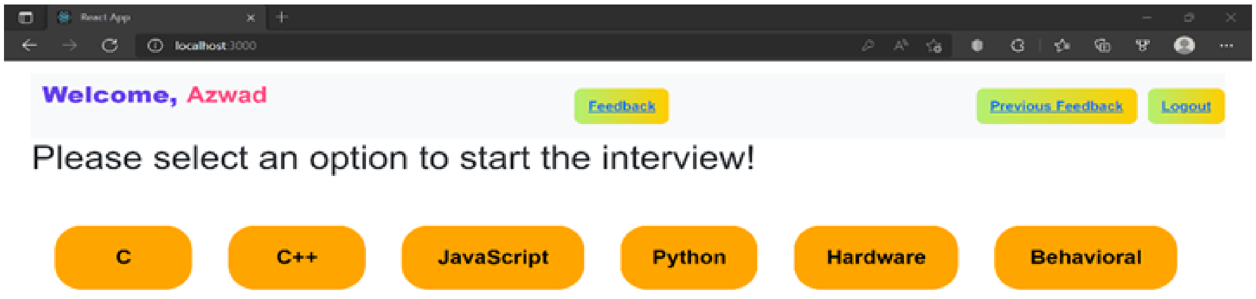
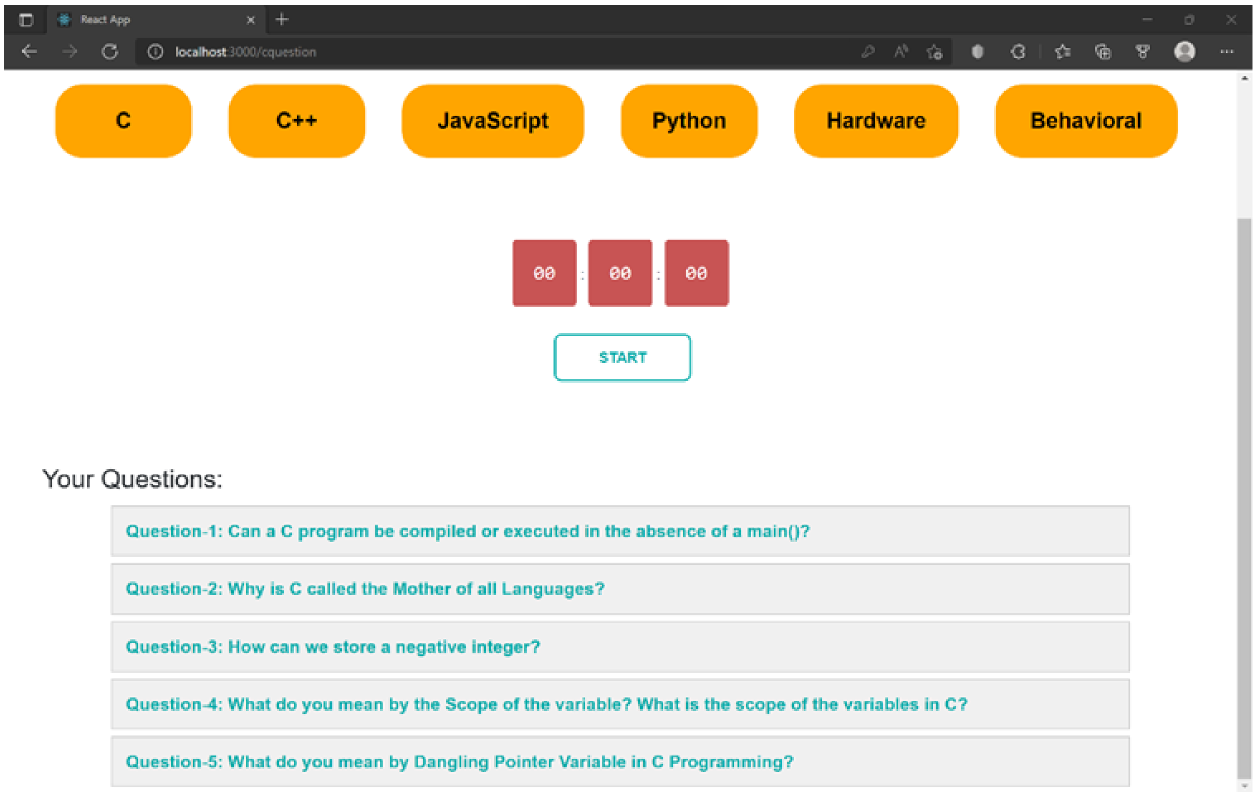


Figure 3: Home page

After logging in, the user is welcomed with this home page. It has buttons for selecting a type of programming language or behavioral questions to practice. After completing the practice session, the user will be able to see the feedback by clicking the "Feedback" button. Whenever the user starts a new session then the previous feedback gets updated with the lastly stored feedback results.



Your Questions:

Question-1: Can a C program be compiled or executed in the absence of a main()?

Question-2: Why is C called the Mother of all Languages?

Question-3: How can we store a negative integer?

Question-4: What do you mean by the Scope of the variable? What is the scope of the variables in C?

Question-5: What do you mean by Dangling Pointer Variable in C Programming?

Figure 4: Question Display page

This is an example of the questions that can be seen by the user when clicking the "C" button. A timer was implemented in all the "Question Display" pages. There is a button to "Start" the timer which marks the start of the interview. Once the "Start" button is clicked, an interviewee can either "Stop" or "Reset" the timer. The user interface of the timer is displayed below:

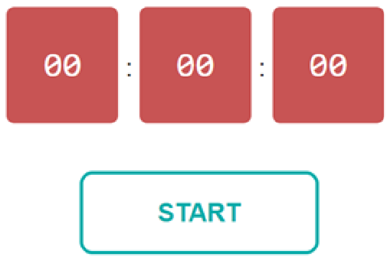


Figure 5: Timer Start button

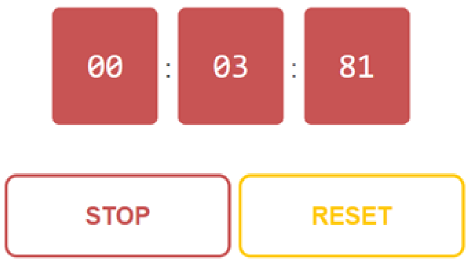
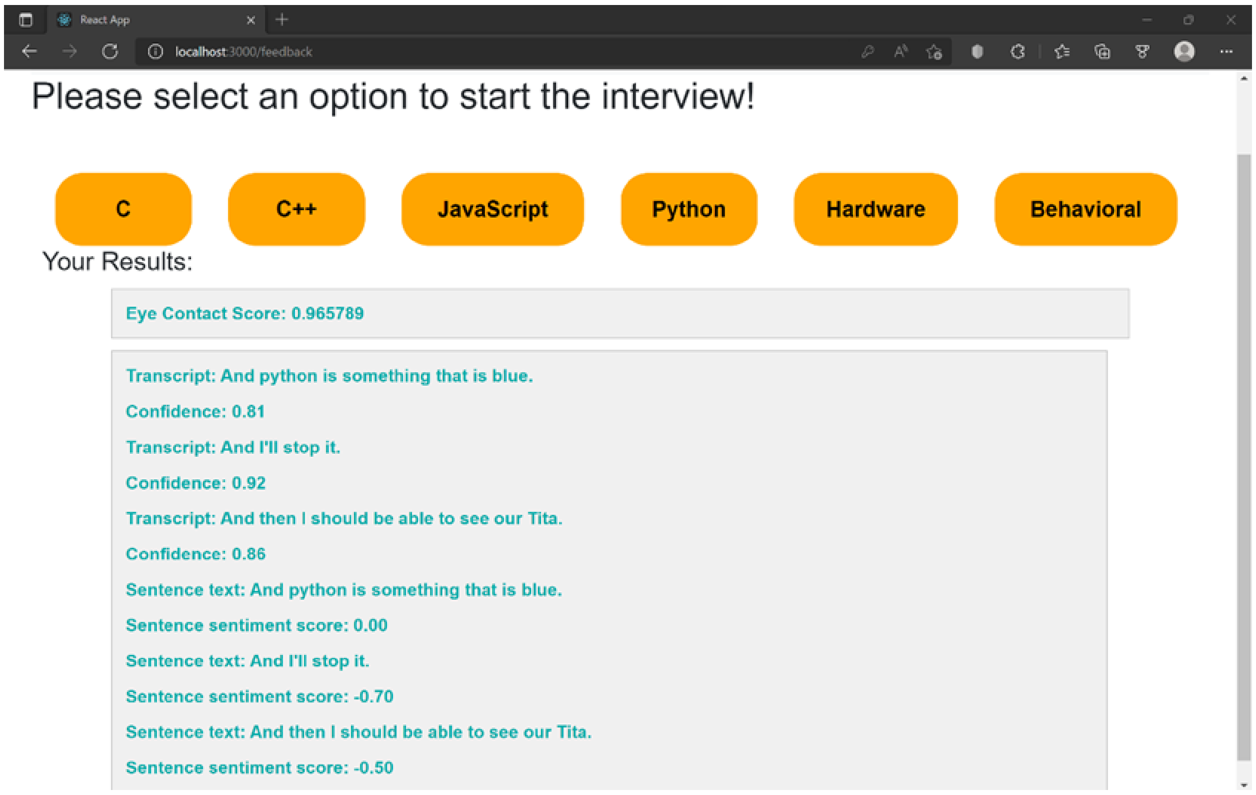


Figure 6: Timer Stop and Reset button



Eye Contact Score: 0.965789

Transcript: And python is something that is blue.

Confidence: 0.81

Transcript: And I'll stop it.

Confidence: 0.92

Transcript: And then I should be able to see our Tita.

Confidence: 0.86

Sentence text: And python is something that is blue.

Sentence sentiment score: 0.00

Sentence text: And I'll stop it.

Sentence sentiment score: -0.70

Sentence text: And then I should be able to see our Tita.

Sentence sentiment score: -0.50

Figure 7: Feedback page

This is an example of displaying the feedback to the user based on their practice session. The "Average Eye Contact Score" is calculated by the machine learning model of eye contact. After that, the feedback for Speech-to-Text and Sentiment Analysis are shown with the help of confidence and sentiment score.

However, the following libraries of React were used for developing the front-end application: Router, Hooks, Axios, and Material UI.

- Router: It was used to configure the routes among the pages of the application. It enabled the navigation functionality by changing the browser URL. For example – clicking the "C" button from the "Homepage" helps to navigate to the "/cquestion" page by changing the URL. It is shown below:

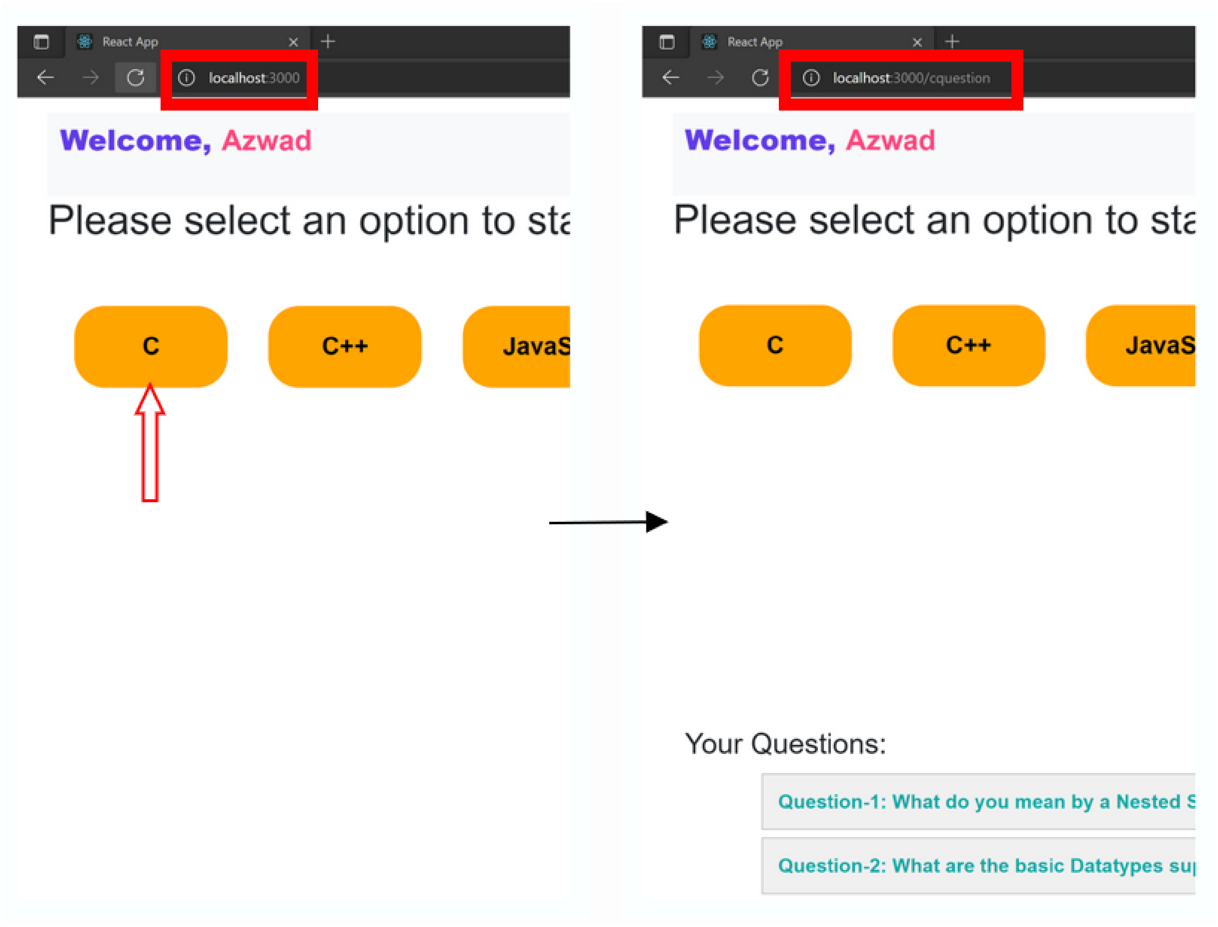

React App
localhost:3000
Welcome, Azwad
Please select an option to sta
C
C++
JavaS
React App
localhost:3000/cquestion
Welcome, Azwad
Please select an option to sta
C
C++
JavaS
Your Questions:
Question-1: What do you mean by a Nested S
Question-2: What are the basic Datatypes su

Figure 8: React Router example

- Hooks: It allowed me to use state features without writing a new class. When the buttons for the questions were clicked then the state of the application was changed to display the questions. A state was also used to keep track of the timer. Moreover, “useState” was used to update the state of the present session feedback and the previous session feedback.

- Axios: It is an HTTP client library that allows to make requests to a given endpoint. It is a promised-based HTTP client which gives the ability to take advantage of JavaScript’s async and await functions. It was used to send “get” requests to the server from the front-end application. A “get” request was implemented on the feedback buttons for fetching the results from the database. It was also applied on the home page buttons for selecting a type of language to get the questions from the database.
- Material UI: It is an open-source, front-end framework for React components. This library was used to design the buttons on the home screen. The properties of color, borderRadius, height, width, bgcolor, fontSize, padding, justifyContent, alignItems and onClick were used to style the buttons. onClick was used to implement the button handler function of changing routes to display questions. The buttons are displayed below:

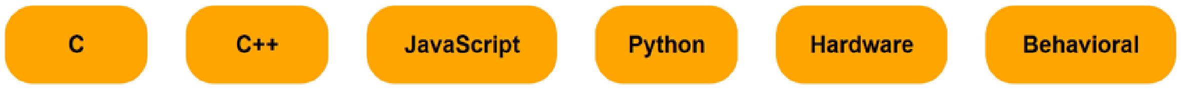


Figure 9: Material UI buttons

Extension of My Design for Future

My design for the “Speech-To-Text” can be extended in the future for performing “Asynchronous Recognition.” Asynchronous recognition offers up to 480 minutes of audio data whereas Synchronous recognition offers only one minute. So, it will certainly allow students to spend more time on each question while taking the interview with our product. The audio data needs to be uploaded to the bucket storage of the Google Cloud Platform to perform asynchronous recognition. I have developed a python script to automatically upload the audio from the filesystem to the cloud storage of Google. This script uses the “storage” package from Google Cloud. After that, I have to use the “uri” sub-field of “RecognitionAudio” to specify the “uri” of the Google Cloud Storage where the audio content is stored. The “uri” can be specified by the following way: “uri = ‘gs://’ + bucket_name + ‘/’ + audio_file_name.” Here, bucket_name is the bucket where the audio file is stored on Google Cloud Storage and audio_file_name is the name of the audio file which needs to be analyzed. In the above mentioned way, my design for “Speech-to-Text” can be extended to Asynchronous Recognition from Synchronous Recognition.

Challenges

One of the big challenges in front-end development was to update the state of the feedback. There are two feedback buttons, one for displaying the results of the present interview session and the other for displaying the results of the previous interview session. Each feedback button displays the results of average eye contact score, confidence score, and sentiment score. One axios “get” request was made for both confidence and sentiment score and another “get” request for average eye contact score. Since two separate get requests were sent to the server while clicking each feedback button so the first request used to get overwritten by the second request and would produce a blank white screen. This was solved by nesting the two axios “get” requests. So, it means that, upon the first successful response from the first “get” request, I have triggered the second “get” request inside the first axios request. After that the function “updateMetricsProps” was invoked which updates the state of the feedback at the end of the interview session.

Installing Google Cloud SDK in the Amazon web server was also a challenge. It required Linux with a browser to authenticate the user credentials. Since I do not have a Linux machine, so it required the use of a Virtual Box to use the browser for authentication. It took a significant amount of time to install, authenticate and run the Google Cloud SDK commands on the server.

Team Effectiveness

The overall team effectiveness was wonderful and positive. All of the members in our team had a well-defined task to work on from the first day of group meeting. We had set internal deadlines to finish assigned tasks before the sprint demonstrations. Our group used the "Projects" tab of GitHub to keep track of the tasks. So, we had a clear idea about the tasks which were to be done, in progress and completed. As part of team communication, we used the social media "Discord" to discuss the various components of the project. In times of need, we have helped each other out to accomplish our goals of the project. All in all, our team had all the elements for a successful teamwork.

My Learnings from the Project

The last week of the project was very hectic. Our group had met almost all of the last seven days to ensure all the parts were done in time. We had spent about 40 hours in the last week to improve the likelihood of success. It enabled me to push myself hard to finish my assigned components in order to contribute to a working product. Hard work, perseverance, time management, communication, and effective teamwork are some of the practical skills that I have learnt from this project. I have also learnt the following technical knowledge from this project:

- Hardware – testing audio driver code in HPS, and installing python packages in HPS by connecting it to internet
- Cloud technology – using APIs from Google Cloud Platform to write code by following documentations
- Front-end development – implementing basic React framework libraries to develop a working user interface
- Automation – writing python scripts to automate functions

References and Acknowledgements

Axios in React: A Guide for Beginners. GeeksforGeeks. Retrieved April 15, 2022, from https://www.geeksforgeeks.org/axios-in-react-a-guide-for-beginners/

Coding with Elias. (2020, February 14). *Full React Project – Stopwatch* [Video]. YouTube. https://youtu.be/qwKh4pH7KAk

Natural Language API Basics. Google Cloud Natural Language. Retrieved April 15, 2022, from https://cloud.google.com/natural-language/docs/basics#sentence-extraction

Read and write WAV files using Python (wave). tutorialspoint. Retrieved April 15, 2022, from https://www.tutorialspoint.com/read-and-write-wav-files-using-python-wave

React Router. W3Schools. Retrieved April 15, 2022, from https://www.w3schools.com/react/react_router.asp

React useState Hook. W3Schools. Retrieved April 15, 2022, from https://www.w3schools.com/react/react_usestate.asp

Sentiment Analysis Tutorial. Google Cloud Natural Language. Retrieved April 15, 2022, from https://cloud.google.com/natural-language/docs/sentiment-tutorial

Speech-to-Text basics. Google Cloud Speech-to-Text. Retrieved April 15, 2022, from https://cloud.google.com/speech-to-text/docs/basics#embedded-audio

Daniel Lee Individual Report

Student Number: 18576249

Contribution

For this project, I used the Node.js Express generator to construct the backend base code, then updated server.js to create all of the backend routes. The backend of this project takes images and audio input from the HPS, then runs the Google Api and Machine Learning module to perform sentiment and eye contact analysis(Figure 1). The findings of the analysis would be kept in a MongoDB database. I set up MongoDB and collections to store user information, analysis results, and interview questions for each interview category (C, C++, Javascript...). AWS EC2-instances were also set up to give external access to the server (IP: 34.222.245.107). In addition, I collaborated with Patrick and Chris to determine an efficient method of sending images and audio from the HPS to the server. The final implementation can be seen in the Detailed Design part, while the obstacles can be found in the Challenges section. In addition, I collaborated with Azwad and updated frontend code to make it work with the backend. This allows users to select an interview category, begin a virtual interview, and review the results. I also set up a database collection so past metrics of earlier interview methods can also be found in a separate previous feedback page on the companion website. I mostly focused on the server-side integration of several design components (HPS, Google Api, ML module, and frontend) communicated via AWS server. I was able to automate the data flow process between components as a result of this.

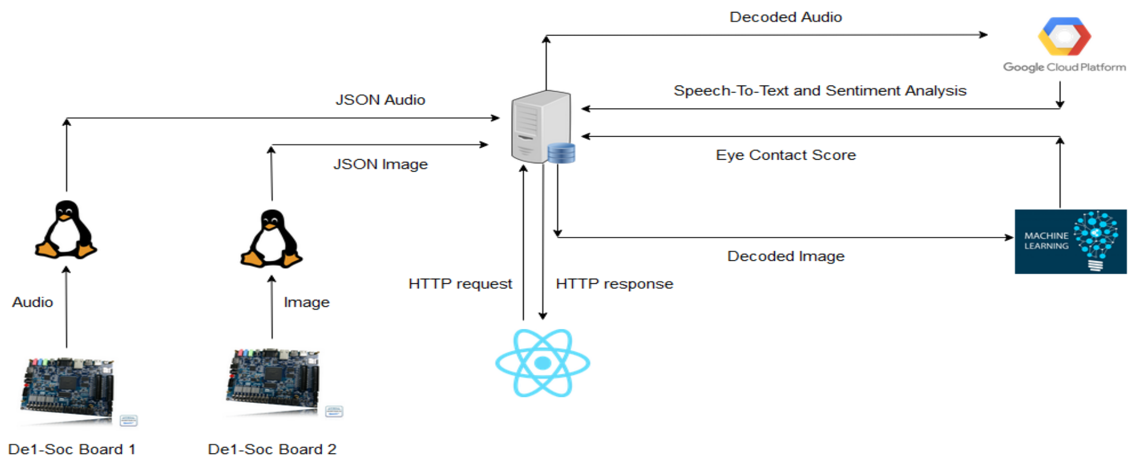


Figure 1: Project Architecture for Virtual Interview session

Detailed design

An outline is provided here for each task, as well as background information, a description of how the task was implemented, and its value to the project.

The main files where code was written for the backend are:

a) **server.js**
   i) virtual-interview-session/backend/server.js
   ii) This file contains all the routes and main setup for the express server, including all the dependencies

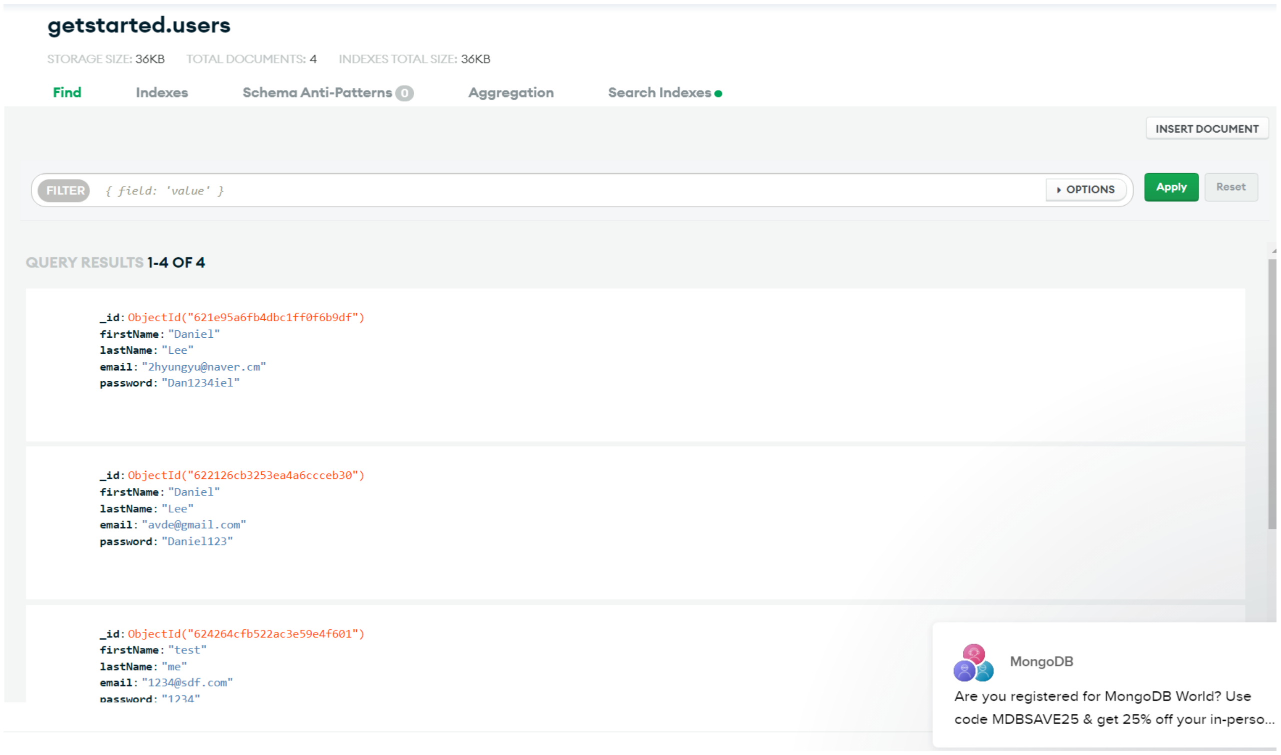
b) **database.js**
   i) virtual-interview-session/backend/database.js
   ii) This file wraps a mongoDB connection to provide a higher-level abstraction layer for manipulating the database objects in this project

1) Backend - Web Page

Backend interacts with frontend to manage user information, start virtual interview sessions, generate random interview questions for each interview category, and display matrices for current and previous virtual interview sessions. To implement the backend, I followed Buecheler's "The Dead-Simple Step-By-Step Guide for Front-End Developers to Getting Up and Running With Node.JS, Express, and MongoDB". Express generator was used to quickstart the development of the application, and then the default routing file server.js was modified to handle all the required routes for the backend. These routes respond to various http requests by either updating the JSON data stored in the database, or fetching data from the database and forwarding it to the requesting ip address. To allow external access to these servers, AWS ec2-instances(IP: 34.222.245.107) have been set up. The backend express server was hosted on port 6000 of the AWS server. In server.js, I set up the following endpoints for our web page.

a) /api/users

- This route is used by the frontend to post and retrieve user information. As a result, the backend express server updates/forwards JSON data (containing user information) from the MongoDB user collection (Figure 2). This endpoint creates new JSON data with user information and updates the database in response to a post request. It also returns the user's information from the database in a get request. If the database does not contain any corresponding JSON data, the frontend will receive http status 400 error messages. <u>This feature is implemented for our future improvement.</u>



```
_id: ObjectId("621e95a6fb4dbc1ff0f6b9df")
firstName: "Daniel"
lastName: "Lee"
email: "2hyungyu@naver.cm"
password: "Dan1234iel"

_id: ObjectId("622126cb3253ea4a6ccceb30")
firstName: "Daniel"
lastName: "Lee"
email: "avde@gmail.com"
password: "Daniel123"

_id: ObjectId("624264cfb522ac3e59e4f601")
firstName: "test"
lastName: "me"
email: "1234@sdf.com"
password: "1234"
```

MongoDB
Are you registered for MongoDB World? Use code MDBSAVE25 & get 25% off your in-perso...

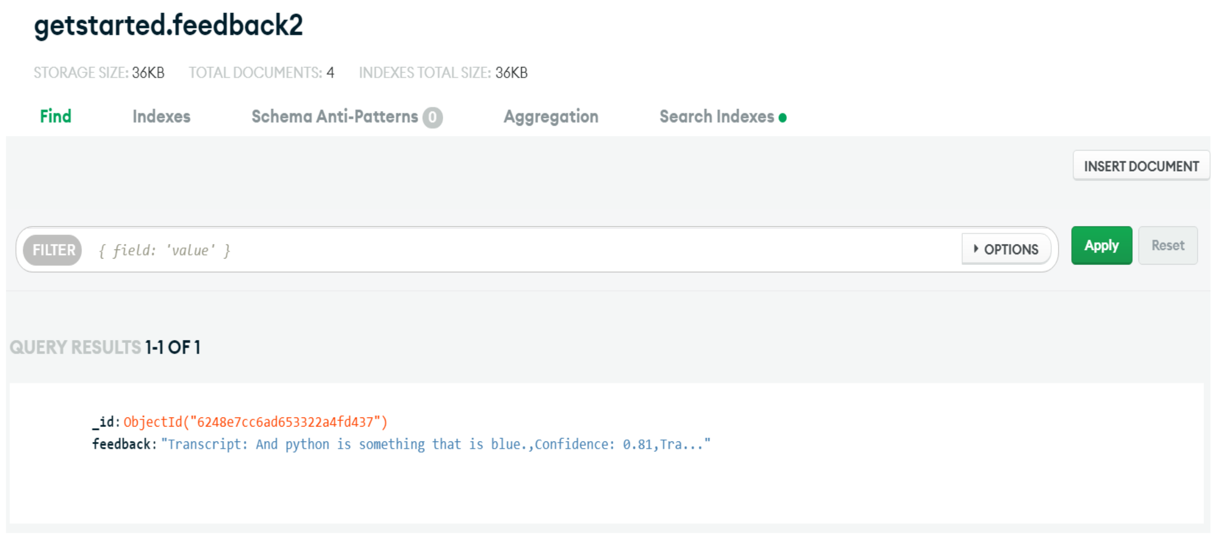
Figure 2: MongoDB user collection - stores user's information(Name, email and password)

b) /api/authentication

- This route is used by the frontend for log-in authentication. The front end will receive http status 401 error messages if the given email address is not registered or the given password is invalid. Otherwise, the frontend will receive http status 200 OK messages. <u>This feature is implemented for our future improvement.</u>

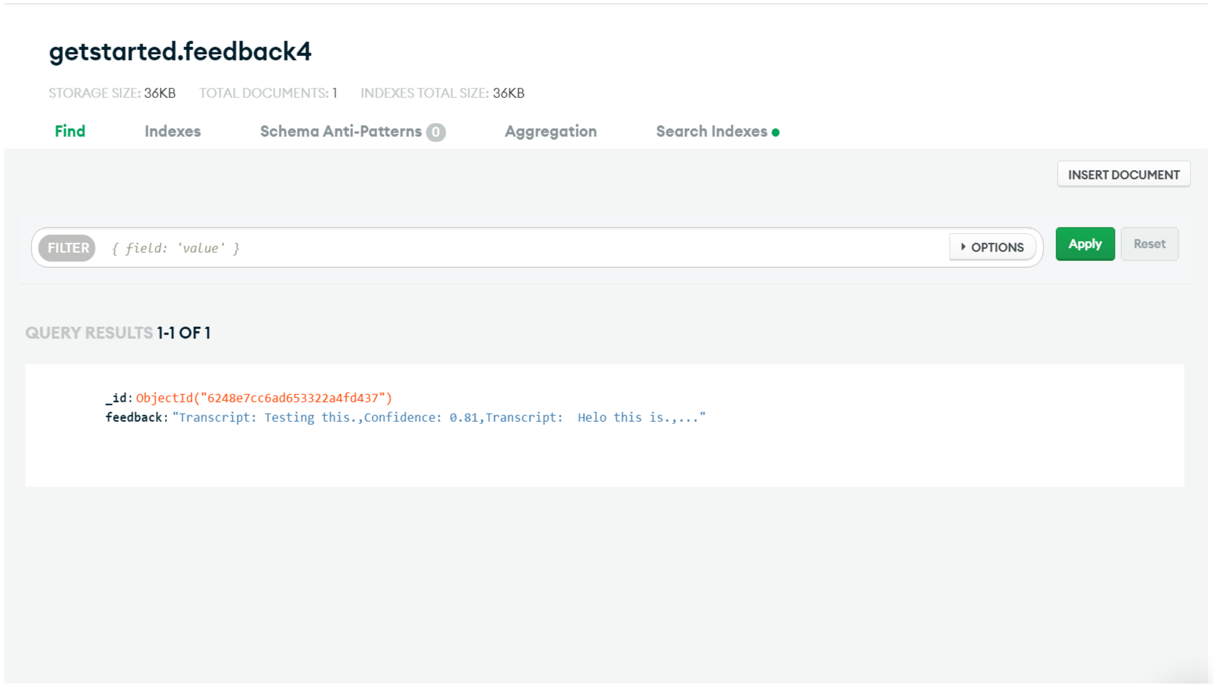
c) /api/start

- The frontend uses this route to start interview sessions. When a user presses the start button, the frontend sends a get request to this endpoint. This means that the server will begin receiving audio and image data from the HPS and will run ML modules and Google APIs on the server side for sentiment analysis and eye contact analysis. As each module returns analysis results, the server will update MongoDB feedback collections (Figure 3) to provide new feedback to the user. Therefore, this collection would only store current interview feedback. Additionally, whenever a get request from this endpoint is received, past matrices from previous interview sessions will be stored in a new MongoDB collection (Figure 4), which only stores past matrices, allowing users to access them in a separate previous feedback sheet.



```
_id: ObjectId("6248e7cc6ad653322a4fd437")
feedback: "Transcript: And python is something that is blue.,Confidence: 0.81,Tra..."
```

Figure 3: MongoDB feedback2 collection - stores sentiment analysis data for current interview session



```
_id: ObjectId("6248e7cc6ad653322a4fd437")
feedback: "Transcript: Testing this.,Confidence: 0.81,Transcript:  Helo this is.,..."
```

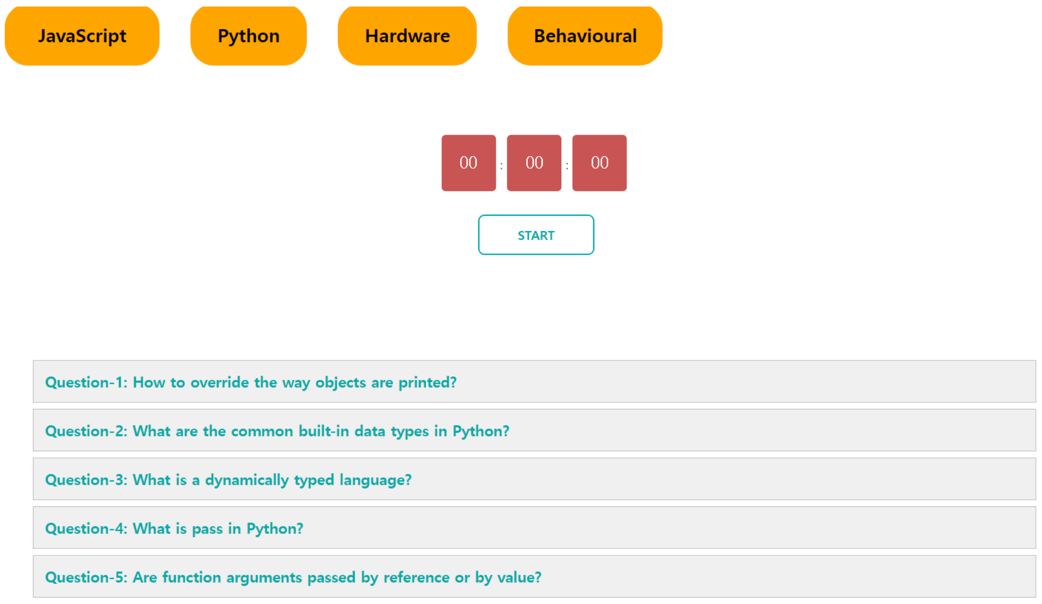
Figure 4: MongoDB feedback4 collection-stores sentiment analysis data for previous session

d) /api/stop

- The frontend uses this route to stop interview sessions. When the user presses the stop button, the frontend sends a get request to this endpoint. This means the server will stop receiving audio and image data from the HPS.

e) /api/python

- If the user presses the python interview category, the frontend will send a get request to this endpoint. Then the server returns 5 random questions from MongoDB python collection(Figure 5 and Figure 6). Following 5 endpoints (c,cplus,javascript,hardware and behavioral) also returns 5 random questions for each interview category. The aggregate module is used to return random questions from the database.



Question-1: How to override the way objects are printed?

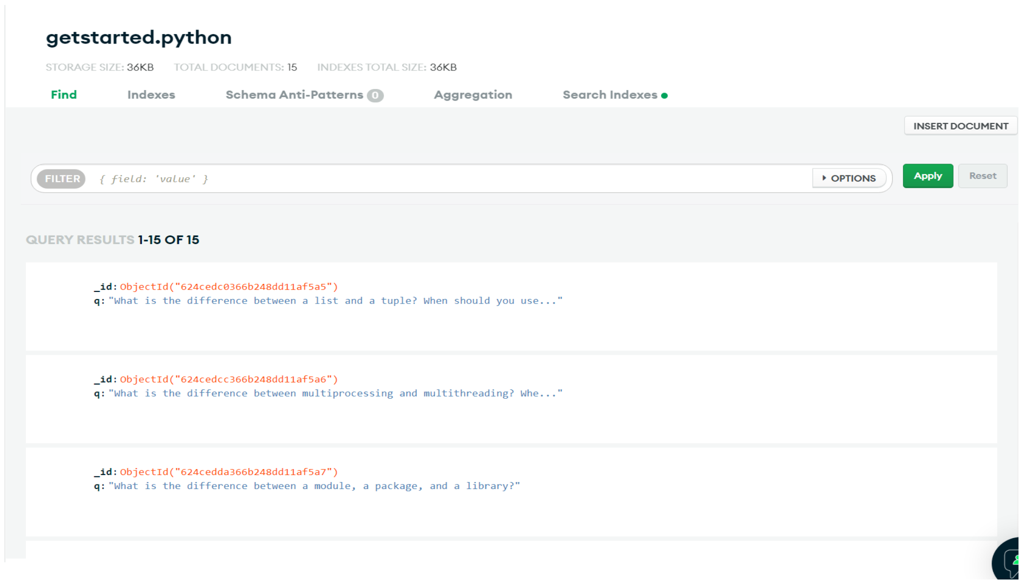
Question-2: What are the common built-in data types in Python?

Question-3: What is a dynamically typed language?

Question-4: What is pass in Python?

Question-5: Are function arguments passed by reference or by value?

Figure 5) Frontend is showing 5 random python interview questions



STORAGE SIZE: 36KB TOTAL DOCUMENTS: 15 INDEXES TOTAL SIZE: 36KB

{ field: 'value' }

QUERY RESULTS 1-15 OF 15

```
_id: ObjectId("624cedc0366b248dd11af5a5")
q: "What is the difference between a list and a tuple? When should you use..."

_id: ObjectId("624cedcc366b248dd11af5a6")
q: "What is the difference between multiprocessing and multithreading? Whe..."

_id: ObjectId("624cedda366b248dd11af5a7")
q: "What is the difference between a module, a package, and a library?"
```

Figure 6: MongoDB python collection - stores 15 python interview questions

f) /api/c

- If the user presses the C interview category, the frontend will send a get request to this endpoint. Then the server returns 5 random questions from the MongoDB C collection.

g) /api/cplus
    - If the user presses the C++ interview category, the frontend will send a get request to this endpoint. Then the server returns 5 random questions from MongoDB C++ collection.

h) /api/javascript
    - If the user presses the Python interview category, the frontend will send a get request to this endpoint. Then the server returns 5 random questions from the MongoDB javascript collection.

i) /api/hardware
    - If the user presses the Hardware interview category, the frontend will send a get request to this endpoint. Then the server returns 5 random questions from the MongoDB hardware collection.

j) /api/behavioural
    - If the user presses the Behavioral interview category, the frontend will send a get request to this endpoint. Then the server returns 5 random questions from MongoDB behavioral collection.

k) /api/feedback1 & /api/feedback2
    - When the user presses the feedback button, the frontend would send get requests to these two endpoints. When the server receives requests from these endpoints, it fetches JSON data(eye contact & sentiment analysis results) from Mongodb database(Figure 3 and 8) and forwards the data to frontend. The frontend receives new metrics of the current interview session.

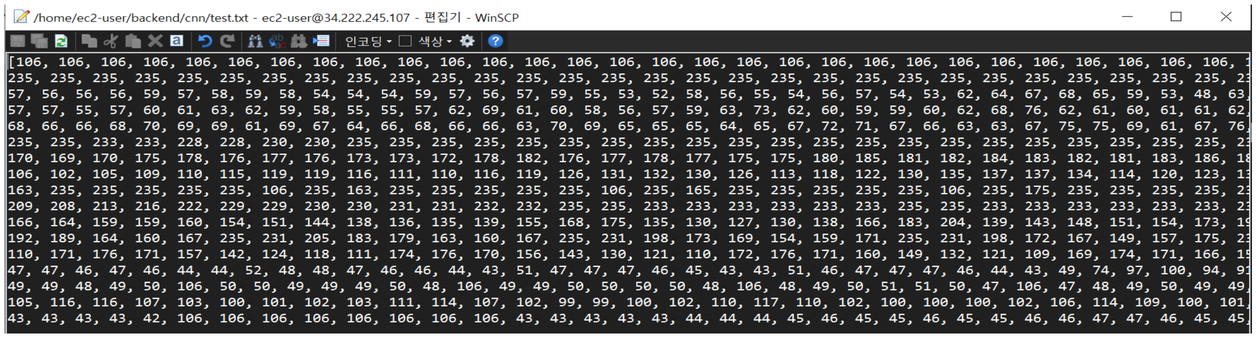
l) /api/feedback3 & /api/feedback4
    - When the user presses the previous feedback button, the frontend would send get requests to these two endpoints. When the server receives requests from these endpoints, it fetches JSON data(eye contact & sentiment analysis results) from Mongodb database(Figure 4) and forwards the data to frontend. The frontend receives past metrics of earlier interview sessions.

2) Backend- HPS, Google API and ML module integration

Backend receives audio and image data from hps via http post request (JSON format), and analyzes sentiment & eye contact using a machine learning module and Google API. When the server receives the data from HPS, it saves them temporarily in a text file (Figure 7 and 9). Next, it runs the module and API, which takes the text file as input. In the MongoDB database, the results from the ML module and API,(sentiment analysis result and eye contact scores) would be stored (Figure 3 and 4). I also updated the frontend so the backend could also decide when data should be received and when it should be stopped with the start and stop buttons. For example, once the interview begins, the server begins receiving data and ends once the interview is completed. The cors module was used to facilitate cross-origin-resource requests to the server so that both HPS in our design could submit data to the backend server.

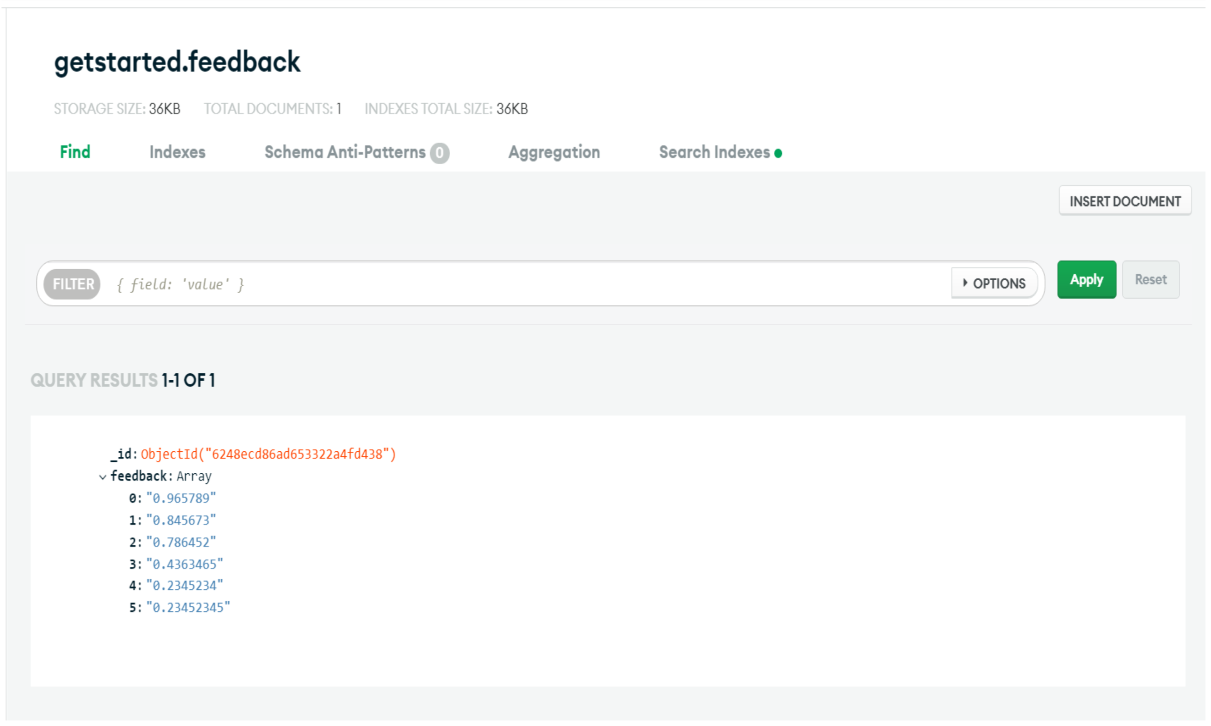
a) /api/image

- For this project we decided to use http post request for image transfer. HPS uses this endpoint to continuously send image data to the server. The raw data format was chosen for image data transfer because it does not require any additional format changes on the hardware side. If the server receives raw images from HPS, it saves them temporarily in a text file (Figure 7). Next, it runs the ML module, which takes the text file as input. In the MongoDB database, the output of the machine learning module (eye contact score) would be stored continuously in array format(Figure 8). The server will convert this array of integers into an average and return it when the frontend requests the eye contact result.


/home/ec2-user/backend/cnn/test.txt - ec2-user@34.222.245.107 - 편집기 - WinSCP
인코딩
색상

```
[106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 1
235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 2
57, 56, 56, 56, 59, 57, 58, 59, 58, 54, 54, 54, 59, 57, 56, 57, 59, 55, 53, 52, 58, 56, 55, 54, 56, 57, 54, 53, 62, 64, 67, 68, 65, 59, 53, 48, 63,
57, 57, 55, 57, 60, 61, 63, 62, 59, 58, 55, 55, 57, 62, 69, 61, 60, 58, 56, 57, 59, 63, 73, 62, 60, 59, 59, 60, 62, 68, 76, 62, 61, 60, 61, 61, 62,
68, 66, 66, 68, 70, 69, 69, 61, 69, 67, 64, 66, 68, 66, 66, 63, 70, 69, 65, 65, 65, 64, 65, 67, 72, 71, 67, 66, 63, 63, 67, 75, 75, 69, 61, 67, 76,
235, 235, 233, 233, 228, 228, 230, 230, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 235, 2
170, 169, 170, 175, 178, 176, 177, 176, 173, 173, 172, 178, 182, 176, 177, 178, 177, 175, 175, 180, 185, 181, 182, 184, 183, 182, 181, 183, 186, 1
106, 102, 105, 109, 110, 115, 119, 119, 116, 111, 110, 116, 119, 126, 131, 132, 130, 126, 113, 118, 122, 130, 135, 137, 137, 134, 114, 120, 123, 1
163, 235, 235, 235, 235, 235, 106, 235, 163, 235, 235, 235, 235, 235, 106, 235, 165, 235, 235, 235, 235, 235, 106, 235, 175, 235, 235, 235, 235, 2
209, 208, 213, 216, 222, 229, 229, 230, 230, 231, 231, 232, 232, 235, 235, 233, 233, 233, 233, 233, 233, 235, 235, 233, 233, 233, 233, 233, 233, 2
166, 164, 159, 159, 160, 154, 151, 144, 138, 136, 135, 139, 155, 168, 175, 135, 130, 127, 130, 138, 166, 183, 204, 139, 143, 148, 151, 154, 173, 1
192, 189, 164, 160, 167, 235, 231, 205, 183, 179, 163, 160, 167, 235, 231, 198, 173, 169, 154, 159, 171, 235, 231, 198, 172, 167, 149, 157, 175, 2
110, 171, 176, 171, 157, 142, 124, 118, 111, 174, 176, 170, 156, 143, 130, 121, 110, 172, 176, 171, 160, 149, 132, 121, 109, 169, 174, 171, 166, 1
47, 47, 46, 47, 46, 44, 44, 52, 48, 48, 47, 46, 46, 44, 43, 51, 47, 47, 47, 46, 45, 43, 43, 51, 46, 47, 47, 47, 46, 44, 43, 49, 74, 97, 100, 94, 9
49, 49, 48, 49, 50, 106, 50, 50, 49, 49, 49, 50, 48, 106, 49, 49, 50, 50, 50, 50, 48, 106, 48, 49, 50, 51, 51, 50, 47, 106, 47, 48, 49, 50, 49, 49,
105, 116, 116, 107, 103, 100, 101, 102, 103, 111, 114, 107, 102, 99, 99, 100, 102, 110, 117, 110, 102, 100, 100, 100, 102, 106, 114, 109, 100, 101,
43, 43, 43, 43, 42, 106, 106, 106, 106, 106, 106, 106, 106, 43, 43, 43, 43, 43, 44, 44, 44, 45, 46, 45, 45, 46, 45, 45, 46, 46, 47, 47, 46, 45, 45,
```

Figure 7: Server temporarily saves raw images in /home/ec2-user/backend/cnn/test.txt

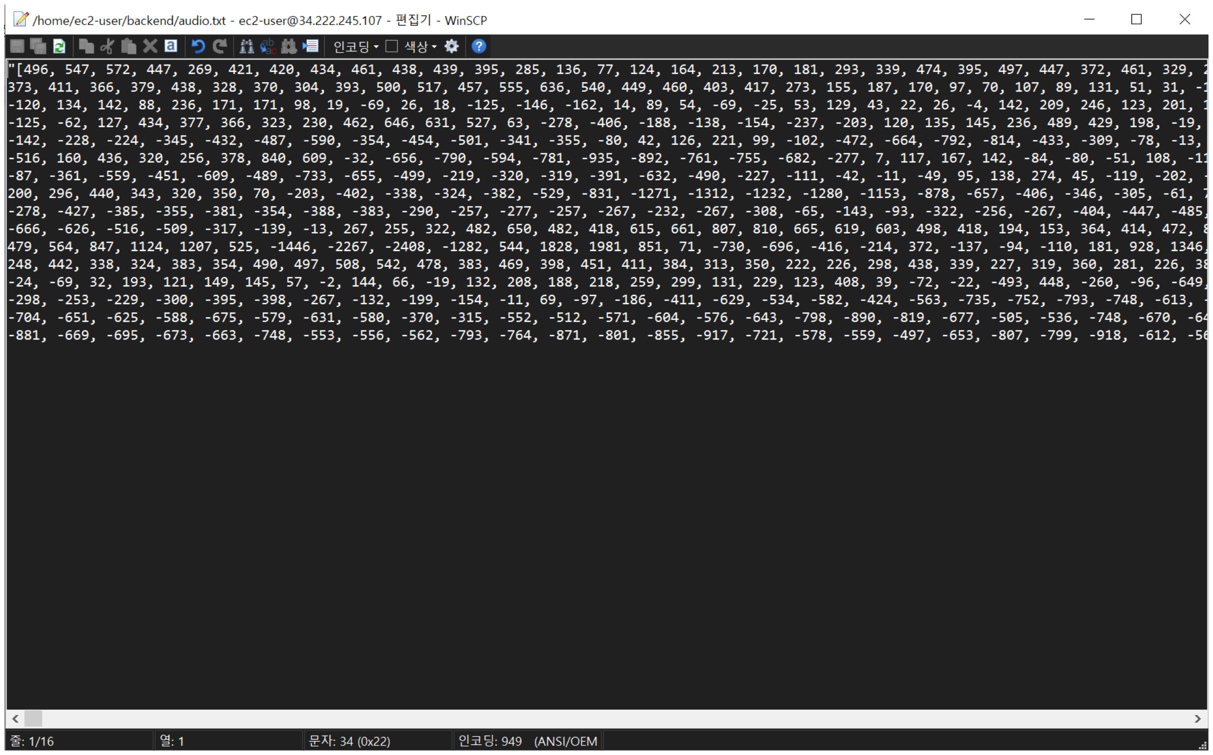


```
_id: ObjectId("6248ecd86ad653322a4fd438")
feedback: Array
    0: "0.965789"
    1: "0.845673"
    2: "0.786452"
    3: "0.4363465"
    4: "0.2345234"
    5: "0.23452345"
```

Figure 8: MongoDB feedback collection- stores eyecontact scores in array format for current interview session

b) /api/audio

- HPS uses this endpoint to send audio data to the server. The raw data format was also chosen for audio data transfer because it does not require any additional format changes on the hardware side. If the server receives audio data from HPS, it also saves them temporarily in a text file (Figure 9). Next, it runs the Google API, which takes the text file as input. In the MongoDB database, the output of this API (sentiment analysis result) would be stored.



/home/ec2-user/backend/audio.txt - ec2-user@34.222.245.107 - 편집기 - WinSCP
인코딩
색상

```
"[496, 547, 572, 447, 269, 421, 420, 434, 461, 438, 439, 395, 285, 136, 77, 124, 164, 213, 170, 181, 293, 339, 474, 395, 497, 447, 372, 461, 329, 2
373, 411, 366, 379, 438, 328, 370, 304, 393, 500, 517, 457, 555, 636, 540, 449, 460, 403, 417, 273, 155, 187, 170, 97, 70, 107, 89, 131, 51, 31, -
-120, 134, 142, 88, 236, 171, 171, 98, 19, -69, 26, 18, -125, -146, -162, 14, 89, 54, -69, -25, 53, 129, 43, 22, 26, -4, 142, 209, 246, 123, 201, 
-125, -62, 127, 434, 377, 366, 323, 230, 462, 646, 631, 527, 63, -278, -406, -188, -138, -154, -237, -203, 120, 135, 145, 236, 489, 429, 198, -19,
-142, -228, -224, -345, -432, -487, -590, -354, -454, -501, -341, -355, -80, 42, 126, 221, 99, -102, -472, -664, -792, -814, -433, -309, -78, -13,
-516, 160, 436, 320, 256, 378, 840, 609, -32, -656, -790, -594, -781, -935, -892, -761, -755, -682, -277, 7, 117, 167, 142, -84, -80, -51, 108, -1
-87, -361, -559, -451, -609, -489, -733, -655, -499, -219, -320, -319, -391, -632, -490, -227, -111, -42, -11, -49, 95, 138, 274, 45, -119, -202, -
200, 296, 440, 343, 320, 350, 70, -203, -402, -338, -324, -382, -529, -831, -1271, -1312, -1232, -1280, -1153, -878, -657, -406, -346, -305, -61, 
-278, -427, -385, -355, -381, -354, -388, -383, -290, -257, -277, -257, -267, -232, -267, -308, -65, -143, -93, -322, -256, -267, -404, -447, -485,
-666, -626, -516, -509, -317, -139, -13, 267, 255, 322, 482, 650, 482, 418, 615, 661, 807, 810, 665, 619, 603, 498, 418, 194, 153, 364, 414, 472, 
479, 564, 847, 1124, 1207, 525, -1446, -2267, -2408, -1282, 544, 1828, 1981, 851, 71, -730, -696, -416, -214, 372, -137, -94, -110, 181, 928, 1346,
248, 442, 338, 324, 383, 354, 490, 497, 508, 542, 478, 383, 469, 398, 451, 411, 384, 313, 350, 222, 226, 298, 438, 339, 227, 319, 360, 281, 226, 3
-24, -69, 32, 193, 121, 149, 145, 57, -2, 144, 66, -19, 132, 208, 188, 218, 259, 299, 131, 229, 123, 408, 39, -72, -22, -493, 448, -260, -96, -649,
-298, -253, -229, -300, -395, -398, -267, -132, -199, -154, -11, 69, -97, -186, -411, -629, -534, -582, -424, -563, -735, -752, -793, -748, -613, -
-704, -651, -625, -588, -675, -579, -631, -580, -370, -315, -552, -512, -571, -604, -576, -643, -798, -890, -819, -677, -505, -536, -748, -670, -6
-881, -669, -695, -673, -663, -748, -553, -556, -562, -793, -764, -871, -801, -855, -917, -721, -578, -559, -497, -653, -807, -799, -918, -612, -5
```

```
줄: 1/16    열: 1    문자: 34 (0x22)    인코딩: 949  (ANSI/OEM)
```

Figure 9: Server temporarily saves raw audio data in /home/ec2-user/backend/audio.txt

Challenges

Efficient mechanism for Audio & Image data transfer

The first problem I faced with this project was figuring out an efficient mechanism to transfer audio and image data between HPS and the backend. We first choose base64 file format for audio and image data transfer since base64 file can be transferred with http post request in JSON format (Appendix A) and also transferring jpg or wav files takes longer timeframes. I was able to test the data transfer from hps to backend for base 64 files (Appendix A). However, it requires a hardware-side data format switch, which could create additional workload for our project. Instead, we choose a raw data format for both types because it eliminates the need for additional data format conversion on the hardware side. The next issue was figuring out how to handle image data transfer efficiently, because unlike voice, there are multiple image data transfers in a single interview session, causing storage to overflow and images to queue (waiting to be used for ML module). To resolve this issue, the server saves raw picture data as a text file on the server side(Figure 7) and uses it to execute the ML module. When a new image transfer occurs, the server will update this text file with the new image and use it to run the ML module. As a result, with each new image transmission, the server changes the text file and uses it to run the ML module. Because the ML module will be executed for each image transfer without waiting for prior results, this could avoid abuse of storage and queue delays.

Set MongoDB Database for previous & current interview feedback

Another issue I ran into was how to handle analysis result data so that the frontend could show the user prior and current interview feedback. The primary goal of this function was to update the database at the appropriate time so that the user could easily distinguish between current and prior feedback. To accomplish this, I modified the frontend code to notify the backend when the interview begins. When the virtual interview session begins, the server will move previous data to a separate database collection (Figure 5) and update the database collection that only records current feedback (Figure 6).

Testing

I believe the major challenge I faced in this project was testing the backend while other components were still being developed. Because the backend is primarily concerned with integrating several components, evaluating data flow was difficult while other components were not yet ready to deliver data in the proper manner. To work around this, I investigated a few linux commands (Appendix A) that perform the same function as hps and provide audio and visual data in a continuous stream. In addition, I wrote simple python scripts to write and read data when the server receives raw image data. This enabled me to anticipate potential problems when integrating the ML module and the Google API. For one example, by testing early, I was able to catch the "Python request entity too large". So, before integrating, I figured out how to alter the input json data and use it to run ML module and Google API. Because we only had a short amount of time to integrate, I was extremely fortunate to be able to test pre-integration.

Team Effectiveness

Everyone involved in this project, in my opinion, gave it their all, with a positive attitude and a willingness to help one another. Everyone seemed eager to spend extra time researching and brainstorming solutions. We were able to connect critical components in a short period of time thanks to the excellent teamwork, particularly for integration. This project taught me the value of communication and cooperation.

For future projects, it would be preferable if we could set a strict time schedule for integration because combining different components may result in a variety of unexpected errors.

References /Acknowledgments:

Buecheler, C. (n.d.). The Dead-Simple Step-By-Step Guide for Front-End Developers to Getting Up and Running With Node.JS, Express, and MongoDB. Retrieved from https://closebrace.com/tutorials/2017-03-02/the-dead-simple-step-by-step-guide-for-front-end-developers-to-getting-up-and-running-with-nodejs-express-and-mongodb

Linuxize, “How to make a post request with curl,” Linuxize, 23-Jul-2020. [Online]. Available: https://linuxize.com/post/curl-post-request/. [Accessed: 15-Apr-2022].

Appendix

Appendix A - Linux commands which sends images in base64/raw data format to the server.

1. (echo -n '{"feedback": "'; base64 WIN_20220303_08_11_19_Pro.jpg; echo '"}') | curl -H "Content-Type: application/json" -d @- http://localhost:3000/api/feedback
2. curl -X POST -H "Content-Type: application/json" -d '{"image" : "'"$(base64 WIN_20220303_08_11_19_Pro.jpg)"'"}' http://localhost:3000/api/feedback
3. scp client@http://localhost:3000/api/feedback:WIN_20220303_08_11_19_Pro.jpg
4. (echo -n '{"img": "'; cat encoded_img.txt; echo '"}') | curl -H "Content-Type: application/json" -d @- http://34.222.245.107:6000/api/image
5. (echo -n '{"wav": "'; cat audio.txt; echo '"}') | curl -H "Content-Type: application/json" -d @- http://34.222.245.107:6000/api/audio
6. curl -k -X POST -F 'image=WIN_20220303_08_11_19_Pro.jpg' -v http://localhost:3000/api/image