https://github.com/Hungry-munk/CRNN-for-OCR-

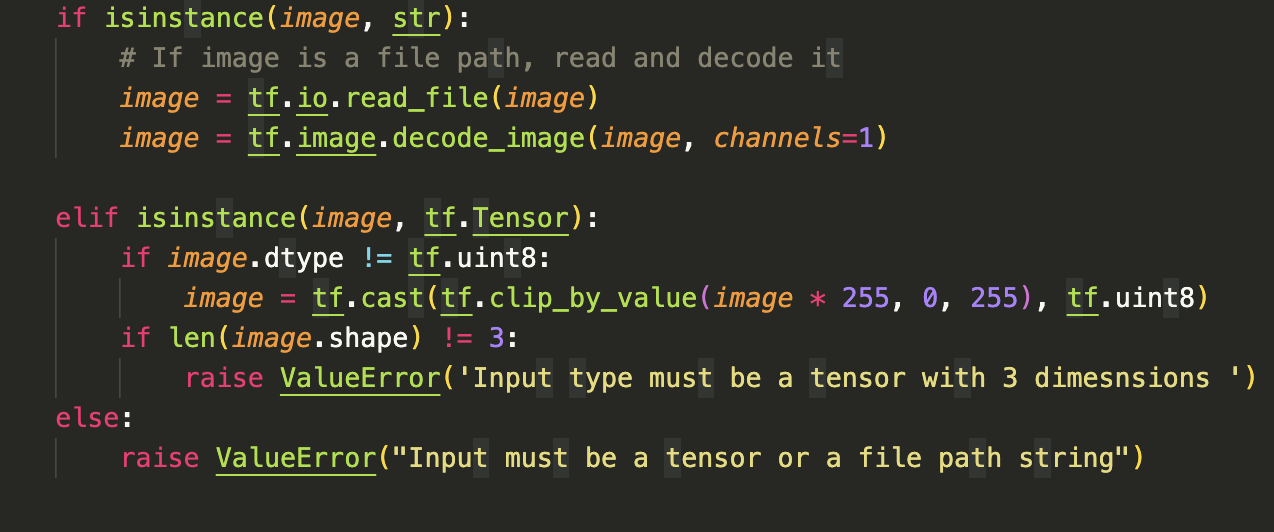
**C06 – Programming skills**

**Validation techniques:**

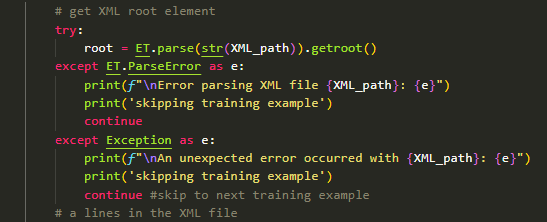
Various validation techniques were used throughout the code base to ensure correct data types and structures are being used passed into functions and systems.

An example is:

The code below was used in the **random\_pad** function. The validation ensures that the image tensor that was passed in is of the correct data type (uint8) and if it isn’t it will convert it. It will also ensure that the tensor is 3 dimensional (for height, width and channels) and if not throw and error. Finally, if the image is if the data type doesn’t match, as it needs to be either a string with the data path to the image or image tensor.



Another example:  
the code below is used the batch\_generator function to ensure that the XML file being loaded to extract the text to create ground truth labels was not corrupt and was indeed an XML file; this was also crucial as it caught bad XML files and stopped them from stopping the code execution during run time meaning that the whole data pipeline was not brought to a halt because of one bad data sample.



Another example:

The code below is simple and just decodes an encoded label back into a string of text. The error catcher ensures that the vocab dictionary contains all the necessary values in the vocab to ensure that we don’t get an unexpected key error.  
A screen shot of a computer code

Description automatically generated

**Critical and creative thinking (design decisions):**

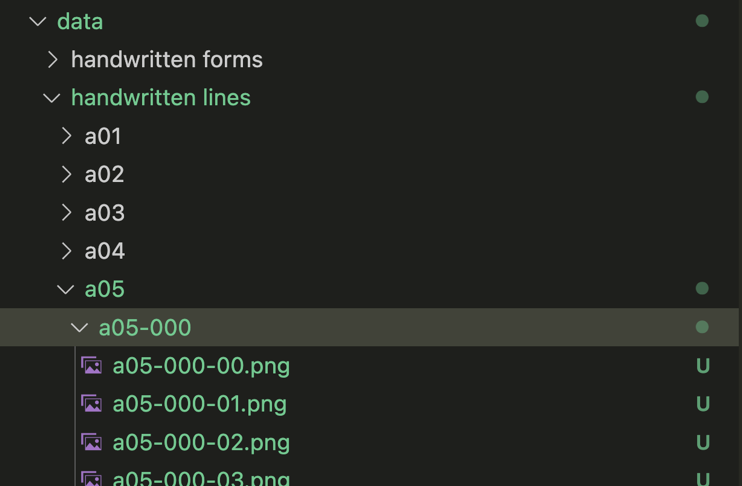
Throughout the entire coding process numerous coding decisions and architectural decisions had to be made after realising that certain things either weren’t possible or didn’t make sense to do.

Some of these solutions to these problems are:

1. **The File iteration problem**

My data set for training the OCR AI engine was the opensource IAM handwriting database. In order to manipulate the image files and get them ready by making TensorFlow tensors.

The challenge was assembling the right string to decode the image file into a tensor, but the names of the files were not consistent alongside the fact that their numerous subfolders that needed accessing also not named consistently.



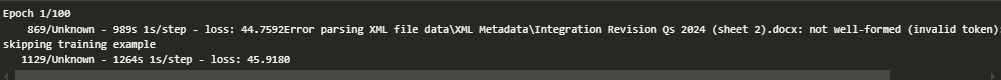
So, I couldn’t dynamically and programmatically create strings to get the image files paths. The way I resolved this problem was learning that using certain libraries I could iterate through retrieve a string representing for the subfolder in the main data folder. The library I chose to do this was pathlib (there were other alternatives like glob and os but pathlib had the syntax I understood the easiest)

The way I resolved the problem was to iterate through the XML file folder that contained the text that was in its corresponding photo but also the file name.  
using the file names I was able to create a string to the exact location of all the images needed for training.

1. **The image shape and network architecture problem**

When it came to preprocess images to make them compatible for training with a TensorFlow Keras CRNN (convolutional recurrent neural network) I initially thought the images could keep their current dimensions and be fed into the model directly. Upon further research I realised that with my limited resources (minimal data, understanding of models and computing resources) I couldn’t use images of any size, they either had to be all be the same size or at least all be of the same height, furthermore they all should be grayscale (to save computing resources). After much deliberation I decided to choose the harder and more effective option of having them be the same height but not width and to ensure that the images contained the same correct information I made to resize them such that they had the same aspect ratio as before to not lose spatial information. To make images all grayscale all images were decoded directly into 1 (grayscale) channel instead of the expected 3 (3 channels for R, G, B) when they first being converted into an image tensor from the string file path.

1. **The dataset pipeline file corruption problem:**

****

When training commenced I found their to be on XML files in my XML folder which would cause error and thus stop training (due to an error). To prevent this from happening I added various try and except statements for data processing throughout the data pipeline to one sample of bad data from stopping the whole training and pipeline process.

**Testing techniques:**

The software functions and processes were thoroughly tested some these tests were:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| What is being tested | How is it being tested | Expected result | Actual result | Comments or solution |
| Bounding box updater | Running image training data into image cropping function and the images inspected as to whether all the text that should be on the image is on the image | 2 images having been cropped from the original containing required text | 2 cropped images strangely cropped not containing the right text | Initially the functions didn’t output correctly cropped images but after iterative bug fixing the function is now effective |
| Image resizing | Training Images run through the preprocessing function and the shape of the images printed out. | All images are the same height and have the same aspect | All images have the same height and appropriate aspect ratio’s | The function was pretty easy so there were no logic errors in the code but got a few errors from misusing the TensorFlow library. |
| Feature map to sequences | A featured map being fed into the f\_map\_to\_seq function and the shape of the sequence printed | Sequence length that is proportional to the width of the image times the chahnnels of the image (an image that has gone through all the convolution layers) | Fixed sequence length | I had to write code to calculate how the CNN layers convolutions effect the length of the width |
| Split complete data into training, cross validation and testing sets | Arrays are split using array splitting techniques and the shape printed to verify correct splitting | 3 arrays that are such that the training set is 75% of the total data and the other are 12.5% | 3 arrays that are such that the training set is 75% of the total data and the other are 12.5% | Other than some basic syntax errors their were no other problems |
| Batch data shape normalisation | Trying to train engine, if the engine has parameters altered then data batches are valid | For training to occur | Type error | Padding a batch was difficult as it required some more thinking into how normalisation was to be carried out alongside advanced tensor manipulation techniques |
| Valid data batch for training | Try ‘yield’ing data from a function to create custom tf.data.Dataset training data for training by feeding valid arguments and data into data\_generator and lmbda | A functional tf.data.Dataset | Custom error, I was thrown telling to entering valid values for the data set to standard size | Fixed my data\_genrator algorithm such that I was getting correct values to create calid data sets of required tf.data.Dataset data structure. |

**C07 – File Management**

**Data structure Organisation:**

The primary data structures in the code include an SQLite database, used for storing user data, and external image and XML data, which are processed to prepare training data and labels for the OCR AI engine.

**SQLite Database:**

The SQLite database is a core component, manipulated through the Flask Python backend API. Operations include fetching, adding, and deleting data. For instance, when users attempt to log in or access previous chats, the database is queried to retrieve both images and the corresponding OCR-extracted text from those chats. New data is added to the database when users create new chats, upload images, or obtain OCR-generated text. Similarly, data is deleted whenever users remove their chats. This systematic approach ensures that the backend efficiently handles image storage and OCR data retrieval.

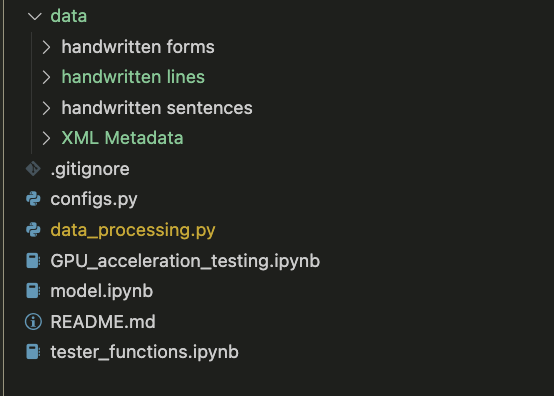
**Image and XML Data for Training:**

The XML and image data are integral to the OCR model's training process. In the data\_preparator function, batches of image data are generated, where the images are converted to floating-point tensors, preparing them for neural network input. Meanwhile, the labels extracted from the XML data are processed into integer sequences. These labels, together with the CTC (Connectionist Temporal Classification) loss function, enable the OCR engine to learn how to recognize characters and words accurately.

**File Management:**

All project files are version-controlled using Git, ensuring robust file management practices. Version control has been crucial, allowing the addition, modification, and deletion of files as needed, while also enabling access to past versions when necessary. Frequently, older file versions were retrieved during development for reference or to recover code that had been altered or removed.

**Folder Structure and File Organization:**

To maintain clarity, a logical folder and file structure was implemented. For example, the OCR engine development code is organized into separate files, ensuring that related functionalities are encapsulated, which reduces complexity. Data used in the OCR training process is stored in a dedicated folder, which contains subfolders that organize the data further for easier access and better readability in the code. Subfolder names followed a consistent naming convention, such as [type of data]\_[content], making it clear what kind of data each folder contains—for instance, a folder named handwritten\_sentences would contain images of handwritten sentences.

**Use of Jupyter Notebooks:**

While the majority of the code is written in Python, much of it was developed in Jupyter Notebooks (.ipynb), which offer the flexibility to run code in blocks. This feature was particularly useful during model development, as certain blocks of code, such as data preprocessing or model training, needed to be run multiple times for testing and debugging. This approach made the development process more efficient and allowed for easier experimentation.

**Other File Types:**

Additional file types, such as .md (Markdown) files and .gitignore, were also used effectively. The .md files provided project descriptions in the GitHub repository, improving documentation. The .gitignore file played a crucial role in ensuring that unnecessary files, such as large datasets in the data folder, were not included in version control. This avoided clutter and made file management easier, particularly when dealing with large datasets.

**File Naming Conventions:**

Consistent naming conventions were followed for Python and notebook files. Python files were named using snake\_case, which not only ensured readability but also simplified cross-platform compatibility. For instance, certain operating systems, like macOS and Windows, handle strings differently, and consistent naming conventions help avoid import errors when working across environments. This attention to detail ensured a smoother development experience and minimized potential issues during deployment.

**Data and file security:**

All files and data are securely hosted on GitHub’s servers, which function both as a hosting platform and a backup solution. GitHub ensures security with robust protocols, and private repositories are only accessible to authorized users, enhancing protection. Files and data are safeguarded by my password and multi-factor authentication (MFA), so even if my password is compromised, unauthorized access is prevented without the MFA credentials. Additionally, data encryption prevents man-in-the-middle attacks during file transfers to GitHub’s servers. GitHub's strong security measures make it highly resistant to hacking attempts.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The files and data are also stored on my local PC and laptop meaning as development occurs on the PC thus GitHub acts as a backup, as when code or data is meaningfully changed that iteration of the software is pushed up for backing up. GitHub backups also allow access to the files from any local machine if needed. This also means that an events-based threat must occur across all devices to lose all code which is incredibly unlikely.

All files on local machines like the PC and mac are also encrypted and password protected on top of access to the devices already being passwords and pin protected.

A screenshot of a computer error

Description automatically generated

**C08 - Usability testing**

**Usability test preparation:**

The core functions of the software solutions are too able to create and account, login and upload and image and manipulate chats.

In order to test for all of the core functionalities effectiveness I will measure the amount of time it takes to be able to complete all of these takes from different states of the software alongside observational notes about user behaviour on the site. Individuals will also be surveyed about the UX and UI experience they had; for instance, the test will include the time it takes to for someone to create an account alongside the ease and accessibility of doing so being queried from them.

**Results documentation:**

**User 1:**

| **Task** | **Time (rounded)** | **UI/UX Experience** | **Observations** |
| --- | --- | --- | --- |
| **Create account** | 22 secs | - Liked overall design. - Felt intuitive. | - Surprised by how the confirm password box appeared only after entering the initial password. - No major concerns noted. |
| **Upload** | 11 secs | - Appreciated the upload menu design aesthetically. | - Disliked the plain white and black color theme, said it reminded them of ChatGPT. - Liked the overall upload experience. |
| **Delete chat** | 4 secs | - Found it very intuitive. | - Immediately went for the correct icon without hesitation. |

**User 2:**

| **Task** | **Time (rounded)** | **UI/UX Experience** | **Observations** |
| --- | --- | --- | --- |
| **Create account** | 30 secs | - Liked the overall design. - Felt intuitive. | - Annoyed by the confirm password box appearing after entering the initial password. - Minor improvement suggestion: Show both password fields at once. |
| **Upload** | 32 secs | - Liked the upload menu aesthetically but not the color theme.  - compared it to ChatGPT. | - Disliked the white and black color theme, - Thought the upload button was too big, reducing clarity in other elements  - took longer than expected due to searching for good image |
| **Delete chat** | 4 secs | - Found the process very intuitive.  - Appreciated the immediate feedback on deletion | - Easily identified the delete button, suggesting it was placed well in the design.  - |

**User 3:**

| **Task** | **Time (rounded)** | **UI/UX Experience** | **Observations** |
| --- | --- | --- | --- |
| **Create account** | 22 secs | - Liked the design, felt smooth overall. - Intuitive experience. | - Noticed a delay in how the confirm password box appeared, caused brief confusion. - Suggested more consistency in form field appearances. |
| **Upload** | 11 secs | - Enjoyed the upload menu’s look, disliked the color scheme.  - Said the design felt too similar to ChatGPT. | - Found the upload button somewhat oversized, which took focus away from other UI elements. - |
| **Delete chat** | 10 secs | - Process was intuitive and efficient.  - Commented that the ChatGPT-like design doesn’t feel original or  - relevant to the app's purpose. | - Immediately clicked the correct icon. - Found the confirmation prompt useful |

**Additional Observations and summary:**

* **Colour Theme Consistency**: Users across the board weren’t a massive fan of the black and white colour theme. Some felt it was too plain, with multiple references to ChatGPT's design. Future designs should consider offering customizable themes or a more vibrant colour palette to differentiate the interface.
* **Button Sizing and Positioning**: Users suggested that the upload button was too large, which could distract from the clarit0079 of other UI elements. Smaller, well-placed buttons may improve usability.
* **Form Field Behaviour**: The appearance of the confirm password box after entering the initial password was a common pain point. Users expect consistency in form field display and flow.
* **Chat Deletion**: Most users found the delete chat process intuitive, indicating that the current placement and design of the delete icon are effective.
* **Overall UI/UX**: all users made a reference to the ChatGPT like interface and how it didn’t really make sense as it was not a ‘chat’ bot and the text being presented like ChatGPT was unnecessary, users also wanted more character to bine able to be recognised by the OCR engine alongside just a more accurate OCR engine.

**Modification Documentation:**

**Colour Theme Customization:**

* **Issue:** Users found the black and white theme too plain, with multiple references to ChatGPT's design.
* **Solution:** Introduce customizable themes, allowing users to choose between different colour palettes. This could include a more vibrant and engaging theme, while still offering a minimalist option for users who prefer it. Adding a "dark mode" and "light mode" toggle could enhance user satisfaction.

**Button Sizing and Positioning:**

* **Issue:** The upload button was considered too large and distracting.
* **Solution:** Reduce the size of the upload button and ensure it’s aligned with the overall design. The button could be placed in a more intuitive position, such as next to the file name input field, to improve clarity. You could also explore designing buttons with distinct, visually appealing colours that fit the new customizable theme.

**Form Field Consistency:**

* **Issue:** Users were frustrated with the confirm password box appearing only after the initial password was entered.
* **Solution:** Display all form fields at once to maintain consistency and improve the flow of the registration process. Consider providing real-time validation (e.g., password strength meter) and inline error messages to guide users through the form in a smooth, predictable manner.

**Chat Deletion Process:**

* **Issue:** The chat deletion process was found to be intuitive and quick, indicating no major problems.
* **Solution:** Maintain the current delete chat UI but consider adding a confirmation prompt to prevent accidental deletions. Providing users with the ability to undo a deletion for a brief period (e.g., 5 seconds) can also enhance user experience.

**Overall UI/UX Consistency:**

* **Issue:** Multiple users referenced the ChatGPT-like interface and mentioned that it felt out of place.
* **Solution:** Redesign the interface to be more aligned with the software’s actual functionality. Since the software isn't focused on chat interactions, eliminate design elements that mimic a chat interface. Ensure the UI is optimized for the specific tasks the software is meant to perform, offering a cleaner, more focused design. This can also reduce cognitive overload.

**OCR Engine Enhancements:**

* **Issue:** Users want more characters to be recognised and overall better OCR accuracy.
* **Solution:** Update the OCR engine to improve its recognition capabilities, possibly by retraining it with a more diverse dataset to handle a wider variety of fonts, symbols, and handwritten text. Consider adding a user interface option for manual correction of OCR results, where users can adjust recognised text in case of errors, as a part of RLHF (reinforcement learning with human feedback) to improve the engine and UX.

**C09 – Software Evaluation**

**Evaluation strategies:**

**Efficiency:**

Overall, the development of the software was not particularly efficient. The project became expensive largely due to the complexity of developing the OCR engine. Limited computing resources, minimal data, and technical debt made the process time-consuming. A significant amount of time was spent figuring out how to effectively use the available data and developing preprocessing algorithms to make it suitable for training the OCR engine. This process took much longer than anticipated because I had to constantly learn new algorithms and data manipulation techniques to create usable training data. Additionally, I needed to adapt the data to fit the engine’s network architecture, which involved a steep learning curve.

Building the backend APIs for the website also proved time-consuming. Since I had no prior experience in this area, I had to learn and implement concepts as I went along. This led to inefficiencies, particularly with file handling and the overall time required to complete various software tasks which resulted in having to reimplement certain things again as the first implementation caused issues down the line.

**Software Evaluation Documentation:**

**Effectiveness:**

The effectiveness of the solution can be measured upon the following categories:

|  |  |  |
| --- | --- | --- |
| Category | | Evaluation |
| Readability | | All fonts’ buttons were easily understood due to difference in the |
| Attractiveness | | The UI overall was aesthetic, the black and white colour theme made everything easy to distinguish and read all the text. |
| Clarity | | The language in the software was all age appropriate and headings and subheadings and the common conventions of the webpage were consistent. |
| Accuracy | | The accuracy of the OCR engine wasn’t perfect it makes frequent mistakes and would always output the wrong thing when it came to characters the engine was never trained on. |
| Accessibility | | Make the software accessible to all types of users was not in the scope of the project, there is no support for other languages (however the google translate browser extension is always available), and the website has essentially no support for the visually impaired to work with screen reader. The hearing-impaired won’t have any problems because there is no audio related functionality in the solution. |
| Timeliness | The OCR engine can take some time due to limit server resources alongside the database fetching when it comes to retrieving user data (eg, past chats and login details) | |

**Development Model Evaluation Documentation:**

Now that the development is complete, it’s clear that the waterfall model was effective in maintaining focus and ensuring the project met its core objectives, but it also had limitations, particularly when unforeseen challenges arose. Despite this, other development models, such as Agile or iterative models, likely wouldn’t have been more effective given the nature of this project.

Agile development, while highly flexible and suited for larger teams and projects with evolving requirements, wouldn’t have been ideal for this scenario. The key challenge with Agile is the potential for scope creep—since Agile focuses on continuous feedback and iterative improvements, new features and changes can easily get introduced mid-development. In my case, the project had clearly defined functional and non-functional requirements established early on through client communication. The clarity of these requirements meant there wasn’t much room, nor necessity, for frequent re-evaluation or course changes, which Agile emphasizes. Introducing new features or constantly revisiting the scope would have led to inefficiencies, delays, and possible confusion, especially given that the project was already resource-constrained and technically complex.

Similarly, iterative models that allow for repeated cycles of development and testing wouldn’t have aligned well with this project’s goals. The structured nature of the requirements, combined with the need to focus on building the OCR engine and backend APIs, made a linear approach more suitable. Iterative models might have introduced too much overhead, requiring constant testing and redevelopment, without offering substantial benefits. Since the client and I had a clear understanding of the solution from the beginning, there was no pressing need to revise or expand the project’s scope along the way.

The waterfall model’s linear progression was advantageous in this context because it helped keep the project on track and under control. The defined phases—requirements gathering, design, implementation, testing, and deployment—allowed me to work with clear expectations and ensured that the functional and non-functional requirements were locked down early on. This approach minimized unnecessary communication overhead and enabled me to focus on the technical execution rather than constantly revisiting or reimagining the scope. Agile, with its emphasis on continuous feedback and evolution, would have caused unnecessary disruptions, and iterative models would have led to inefficiencies given the clear direction we had from the outset.

In conclusion, while the waterfall model had some challenges in terms of flexibility, it was the best fit for this project. Agile and iterative approaches would have opened the door to scope creep and excessive revisions, making the development process longer and more complex. By sticking to a structured plan, I was able to deliver a product that met the initial requirements without unnecessary detours or distractions.

**C10 – Project Plan Assessment**

The Gantt chart and project plan proved to be a useful time management tool, particularly during the development and evaluation phases. However, due to my lack of experience with similar projects before the SAT, the time allocated to various tasks was based more on intuition than on a proper assessment of what each task would realistically require. This led to an imbalance, with some tasks receiving more time than necessary, while others were underestimated.

The most significant miscalculation occurred during the software development phase, which took far longer than expected. As I began developing the solution, it became clear that certain aspects were more complex than initially anticipated. This extended the development timeline significantly, from an originally planned completion date of August 14 to September 18, the delay was also aided due conflicting responsibilities such as preparing for other SAC’s and completing work for other subjects. This delay severely impacted the rest of the project, leaving little time for other tasks like usability testing. As a result, the usability testing phase was rushed, and only three users were able to test the final solution, limiting the thoroughness of the evaluation. In addition the proper order of the tasks couldn’t be followed and some tasks that had to be completed before others were completed after wards due to completion length of the solution.

What happened was that the project plan was followed until the development of the solution, that took until the September 18th and everything after was simultaneously completed in a matter of 2 days all simultaneously making the project from the development of the software solution and onwards look something more like this:

**more accurate project plan:**

A screenshot of a calendar

Description automatically generated

In hindsight, a more accurate time assessment in the early planning stages would have improved the overall balance of the project schedule.

**Effectiveness assessment:**

The project plan, as mentioned earlier, was not effective for managing time and tracking progress once the development of the software solution began. However, before that, it served as a useful tool because the tasks were relatively simple and there were no delays in completion. This allowed me to easily check off tasks as they were completed, ensuring everything stayed on track. Its effectiveness, however, was due to the simplicity of the tasks. Since I was new to the process, I was fortunate that none of the tasks were too challenging or caused delays to those dependent on them.