**CSCI-499 Defense Documentation**

Project Title: Football Card App

Date: 8/25/24

Student Name: Riley Osborne

Degree and Major: Bachelor of Science in Computer Science

Project Advisor Name: Mr. Michael O’Neill

**Statement of Purpose:**

Offer the core tools necessary for a quality card management system such as a pricing tool and optical character recognition software.

Problem Statement:

The current state of the trading card collecting community is in dire need of an inexpensive, popular, and accurate way for individual collectors to manage their card collections. If a collector currently wants to evaluate any one of their card’s values, they either have to manually figure it out with Ebay sold data, use magazines containing cheap and inaccurate prices, or sacrifice an arm and a leg to pay for third party price guide subscription services. The lack of a central location where collectors can access an inexpensive and accurate price guide, a way to safely communicate and exchange cards, and a way to also store all of their collection info is frustrating. This is especially true for smaller or poorer collectors who do not spend over five figures regularly in the hobby like elite card vendors do. My project will focus on these three elements, using a uniform organization process for all cards, so that usability and accuracy will be the top priorities along with security. The benefits will be a sturdy, expandable system, a central place for card prices, collection data management for collectors, and an attractive, secure, and efficient communication network for collectors. God wants us to be good stewards of our money, so it is important to provide a responsible, accurate, and enjoyable fiscal experience for those who would like to partake in the card collecting hobby. The card collecting community needs reliable and efficient collection management, a strong and secure social outlet, and an accurate, automatic pricing system for their collection that does not cost a ridiculous amount. This project strives to bring all of these aspects together in a complete and industry-leading way that has yet to be adequately implemented in the current system.

**Research & Background:**

Background:

I do not have much of a background in programming, but I am most comfortable with the Python language. The most applicable skills I required prior to constructing the project came from my System Analysis and Software Design, Database Management, and User Interface Programming courses, although that is not an exhaustive list. I have experience in a professional environment with code, and troubleshooting and debugging were key skills to helping set up the project environment. When it comes to the project subject: trading cards, I have an extensive background. I opened my first pack of cards before my birthday in October 2016 while we were evacuated from hurricane Matthew. Since then, I have expanded my knowledge and reach in trading cards. My time in the hobby has been essential in evaluating the needs for a card management app and how the hobby works, so that I can create the best requirements for a fully functional app and/or the core tools for a card management app.

Research:

Despite having some background in the trading card hobby and programming, I did not start project construction with anywhere near the amount of knowledge needed to complete the app or any of its components on my own. For the components below is a fleshed-out list of some of the necessary research I had to do to make progress on the project. In general I had to learn a lot of things for the project, and not everything researched was used for the project.

Pricing algorithm:

* + URL manipulation
  + Web scraping
  + Python date time format
  + Exponential Moving Averages
  + Running system processes with Python

OCR program:

* + Python pytesseract library
  + Python EasyOcr
  + Craft algorithm
  + East algorithm
  + Scanning high resolution images
  + Cropping images in Python
  + Python dictionaries
  + Counting time elapsed in Python
  + Multiprocessing in Python

Database:

* PSQL

UI development and production:

* Making PostgreSQL production and development environment

**Project Language(s), Software, and Hardware:**

**Project Languages:**

* Ruby on Rails
* HTML
* Python
* PostgreSQL

**Software:**

I did not enlist the help of software, but I will include below my app deployment platform as well as some of the python libraries (along with source or documentation links) that are essential to the completion of the project along.

* App deployment platform: Fly.io
* EasyOCR – source: <https://github.com/JaidedAI/EasyOCR>
* East algorithm text detector – source: <https://github.com/ZER-0-NE/EAST-Detector-for-text-detection-using-OpenCV/tree/master>
* Pytesseract – documentation: <https://pypi.org/project/pytesseract/>
* Visual Studio Code
* Windows Subsystem for Linux (WSL)

**Hardware:**

* Windows 10 laptop

**Project Requirements:**

<https://docs.google.com/document/d/1nI77Iaz7NYopkGFKNu9-gfjBs9vhYogjyRtMVMep3oU/edit?usp=sharing>

**Project Implementation Description & Explanation:**

The project is a completed web scraping program with an optical character recognition component also still in development. The web scraping program starts by taking user input for a trading card, asking for details such as the sport, player, set name, and if it’s condition has been graded by a professional grading company and/or if it is a short printed variation/parallel **(Figure 1).** When entering the details, the program is designed to list the user’s options and is designed with input validation for the supported choices in each category **(Figure 2).** Then the program sums up the user’s selections and uses the user specifications to construct a specific eBay link which will use eBay’s search with precise keywords to retrieve all sold listings of cards matching the item details from over the last 90 days **(Figure 3).** Another cool add on feature is that the program will automatically open the eBay link with your default browser if run from a windows machine **(Figure 4).** After creating and opening the link in a browser, the program uses the link for web scraping. Web scraping is used to scan through a page’s html structure and extract desired information. The program uses web scraping on our constructed eBay link to identify the number of total sold listings, and the title, sold date, price without shipping, shipping cost, total price, and listing type (best offer, auction, buy it now) for each sold listing. Before proceeding further, the program does a sort on all the eBay items since eBay sometimes fails to sort by sold date (most recent) with 100% accuracy. Once finished ensuring the correct order, the program cleans the scraped data to its most usable format and outputs all the data to a CSV file so the user can scan over the results **(Figure 5).** The program finishes off using a pricing algorithm on the cleaned data to accurately determine what the current market value is of the card the user specified. Along with the current market value, more information is gathered for analysis, such as the average number of listings sold for the past 90 or 30 days, the oldest and newest sold listing, the average value for the oldest 5 and newest 5 sold listings, and much more **(Figure 6).** Even though this project hones in on trading cards, a similar style of web scraping and pricing can conveniently be customed to work for different categories of items sold with a strong eBay presence.

The optical character recognition (OCR) component is designed to identify all the text on the front and back images of a trading card, accurately translate the text, and then associate the relevant pieces of translated text to their appropriate attributes. Such attributes can be simpler to identify such as player name, set name, year, and team, or more complex attributes to identify such as if the card is numbered, autographed, if the card is a parallel, if it is then which parallel it is, and if it is serial numbered. Additionally, simpler attributes can be made complex simply from small tweaks such as the card company designing their cards using complex, funky, or script style fonts, overlaying text with other images to partially obstruct the text, or orienting the text differently. Consequently, this component is still under development due to the behemoth task of coercing any OCR engine to accurately translate the text in a respectable amount of time. Since accurate enough translation has not yet been achieved, the other massive task of associating the text to the appropriate attributes has yet to be tackled. The end goal is to achieve automation of entering trading cards into a database. This will be achieved by giving the OCR program high quality scanned images **(Figure 7)** of cards and having it retrieve the information from the card to identify all the card attributes the database would ask for to store it in its schema. The OCR program currently runs two OCR engines (Pytesseract and EasyOCR) multiple times, once with the whole image and then both OCR engines on cropped images of text areas from the image. Currently, the program will show the user the cropped images of the text it found **(Figure 8).** The OCR engines will provide an interpretation of the text found and a confidence level for the interpretation **(Figure 9).** With future development, the program will focus on speed, accuracy, and associating the translated text with the appropriate attributes.

Figure 1. Web Scraper user input options

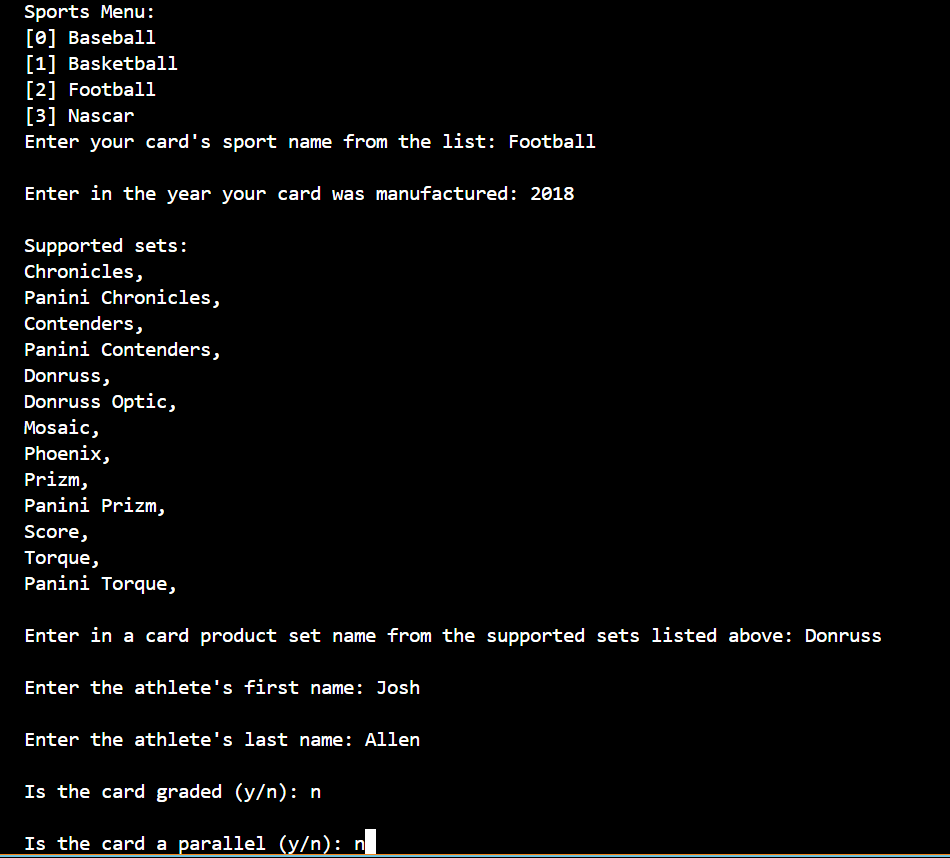
****

Figure 2. Web Scraper options list and input validation

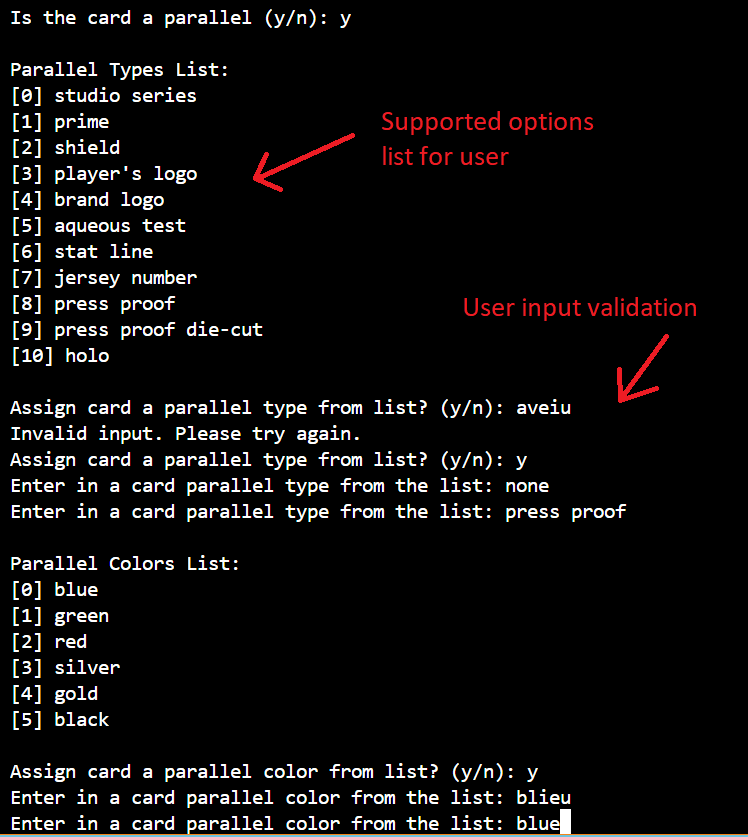
****

Figure 3. Web Scraper input confirmation & URL

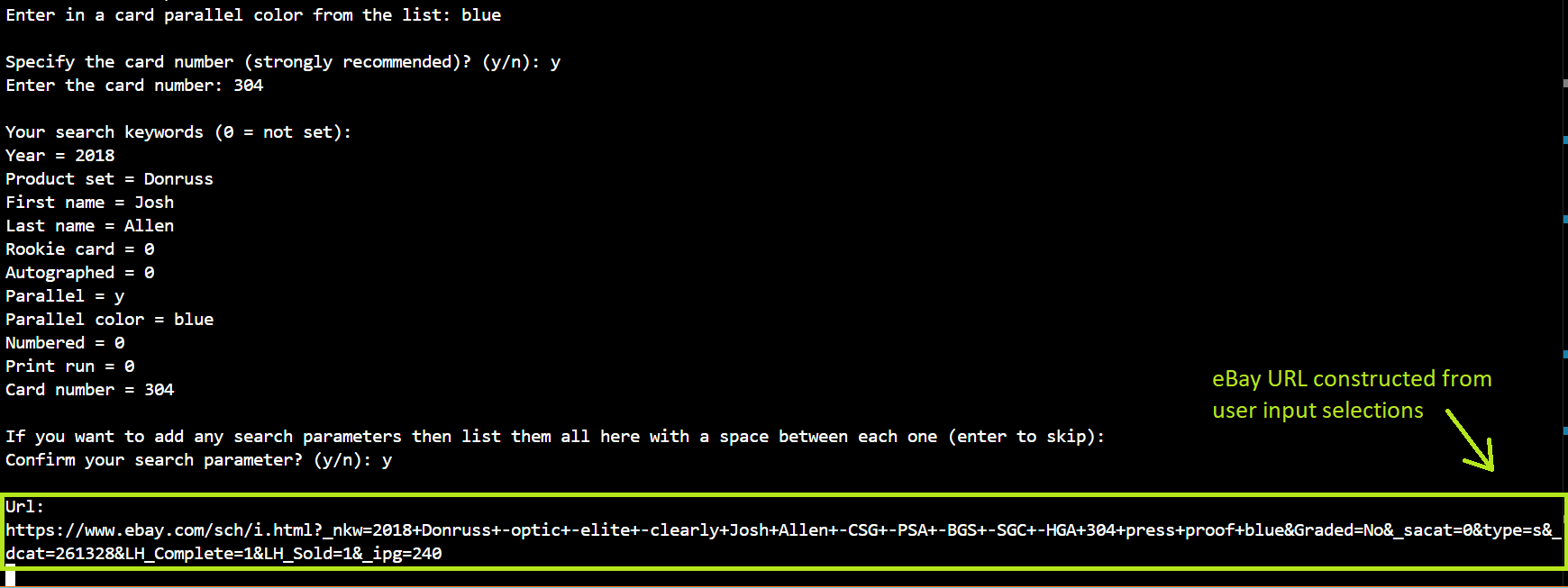
****

Figure 4. Web Scraper opens created URL in browser (Windows)

**A screenshot of a computer

Description automatically generated**

Figure 5. Web Scraper output results to CSV file

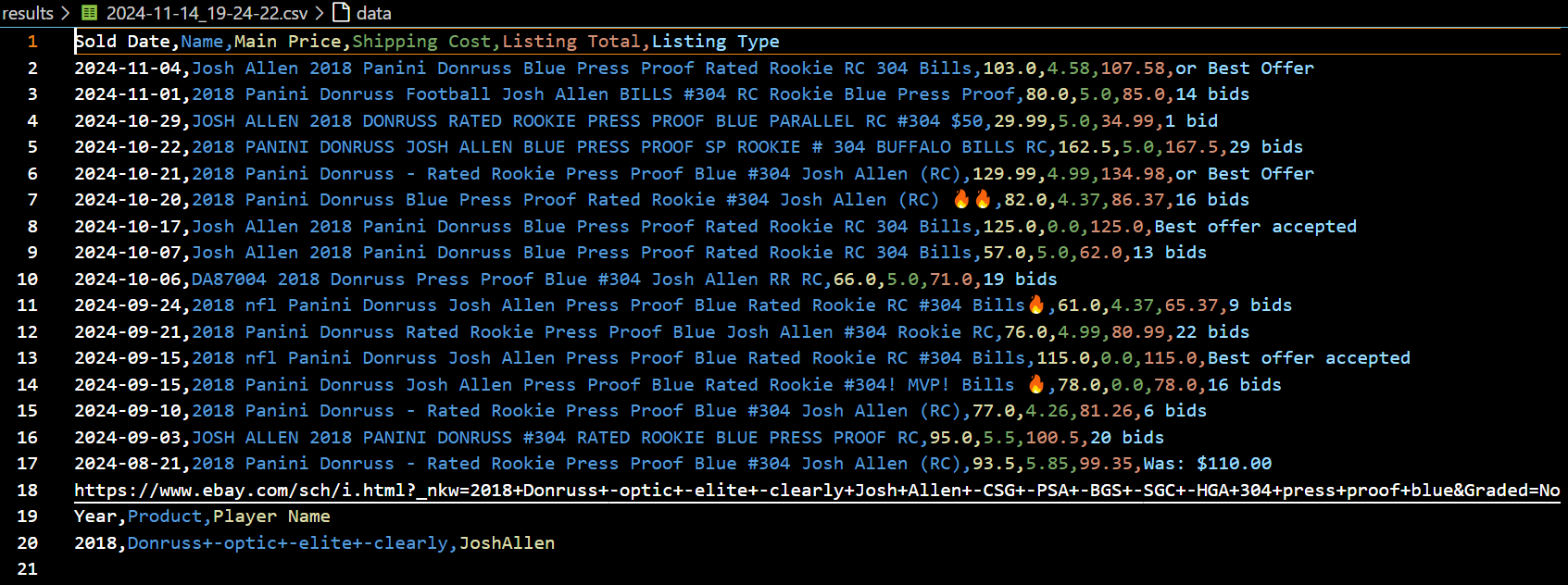
****

Figure 6. Web Scraper sold listings and price analysis

**A screenshot of a computer screen

Description automatically generated**

Figure 7. OCR Program high quality input image

**A football player holding a football

Description automatically generated**

Figure 8. OCR Program detected text examples

**A collage of a football player

Description automatically generated**

Figure 9. OCR Program results analysis

**A screenshot of a computer

Description automatically generated**

Project Repository: <https://github.com/RileyOsborne26/CSU-Senior-Project>

**Test Plan and Results:**

Plan and Results table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component or Function | Requirements  Reference | Test Description | Test  Number | Test Date | Results |
| Web Scraper | #9Fa004 | Clickable link created from URL manipulation | #1w | 9/9/24 | Pass: The program not only has a clickable link, but python opens it in browser for you |
| Web Scraper | #9Fa004 | Web scraped listings returned are 98%-100% match to parameters used | #2w | 7/9/24 | Pass: Using the card number along with existing logic, accuracy is 98%+ easily |
| Card Pricing | #9Fa004 | Each card’s listed total price equals the item price plus the shipping price | #1p | 7/8/24 | Pass: The total price is accurately calculated |
| Card Pricing | #9Fa010 | Algorithm returns values for edge case card value types (low value, rare, etc.) | #2p | 11/4/24 | Pass: Super excited about this one, it works well for returning accurate values for edge case cards!!! |
| Card Pricing | #9Fa010 | Algorithm uses different pricing methods to capture recent sale numbers | #3p | 9/30/24 | Pass: the algorithm uses both fixed listings number ranges and a range of sales over x many days for pricing |
| Card Pricing | #12Na01 | The total time to price a card will take no more than 2 secs. with target of 750 millis | #4p | 11/8/24 | Pass: 1-3 seconds for web scrapes with little to many results |
| Card Pricing | #12Ncd1 | Pricing Precision and Accuracy tests | #5p-a | 10/28/24 | Precision – Pass: checked all monetary values  Accuracy – Pass: works for high value, low value, fixed, and sold range of dates |
| OCR Program  and UI | #9Fa005 | 1200 dpi images are acceptable input for OCR | #1ocr-a | 8/27/24 | Pass: 1200 dpi front/back images accepted |

Note - The section below is for reference so the tests can be repeated and to read for more detail. To use just match the test number column with its respective test listed below.

**Card Pricing algorithm component tests:**

**Web scraping capability:**

**Test #1w** - Through URL manipulation, the web scraping capability will create a clickable link upon each run that will open to a webpage of Ebay sold data for the card specified (#9Fa004).

**Test #2w** – The results returned with the attributes used with the URL manipulation should return listing results that match the parameters with 100% accuracy. In edge cases with good explanation, accuracy down to 98% is permissible. Certain cases that this test case are looking to exclude are graded cards and parallel cards, but if they are not titled well by sellers then they may fall through the web scraping search (#9Fa004).

**Card pricing results:**

**Test #1p** – For each web scraped Ebay listing, the card’s total price will match the listing’s shipping price plus the listing’s item price (#9Fa004).

**Test #2p** – Pricing algorithm remains functional by returning price values for cards that are of low and high value, rare cards, and prices for cards of players currently trending upwards or downwards (#9Fa010).

**Test #3p** – Pricing algorithm is built to change its pricing method depending on the amount of recent sales (#9Fa010).

**Test #4p** – Pricing Algorithm speed benchmark from start to finish for a singular card, which includes the web scraping done for the calculations, should take no more than 4 seconds with a target time of 750 milliseconds (#12Na01).

**Test #5p-a** – Precision, Accuracy, and Reliability tests: (Precision) Pricing algorithm will only deal with monetary values to the nearest penny and (Accuracy) the pricing for each card will be within 10% of the manually calculated current average market price. Since the algorithm behaves differently with less data, the manually calculated current average price will be defined as the median sold price for the same range of sold data used by the program when the average number of cards sold per day for the last 30 days is 1.143 or less (#12Ncd1).

**Optical Character Recognition component tests:**

**Test #1ocr-a** – This test is for the OCR program and the UI, which will be addressed in Test #1ocr-b. Make sure the 1200 dpi scanned images of the front and the back of a card are accepted into the OCR program as direct input (#9Fa005).

**Challenges Overcome:**

Most challenges overcome for me can be defined as brand new areas of learning and exploration which were integral in determining how successful my project would end up being. There are a few implementation hurdles, even for the parts of the project which I have started but are part of future improvements, such as learning how to set up and use PSQL so that I can have a Postgresql production and development environment with Ruby on Rails.

There are a few hurdles I spent a lot of time attempting to overcome but could not, such as attempting multiprocessing to speed up my optical character recognition and attempting to set up and run the OCR program on a system which had a graphical processing unit, since my computer only has a central processing unit. I was able to get my hands on a computer which had a GPU and got my set up on the computer eventually. My motivation for this was attempting to speed up my OCR program since it was running so slow. After getting the set up on the computer with a GPU, I was unable to get it to run since the GPU model was not supported by the OCR python libraries. In addition, when I set aside some time to attempt multiprocessing as a next best option to speed up my OCR, I tested it and found that it actually ran slower than without multiprocessing for my CPU.

Learning how to do URL manipulation by examining the structure of the links eBay use. Doing so helped me to manually build a URL with the keywords and site filters that I needed for web scraping sold data of my cards. Looking over the structure of eBay’s URLs and figuring out how to best search for accurate listings was necessary along with understanding the correct keywords specific to the test case card that I was using. This was a huge hurdle because of the lack of rigid structure from seller to seller on eBay for their listings. People describe cards differently because the sellers are in charge of creating the listings, so I had to figure out how to code the logic for search keywords. The logic is based off of three things: the sport, year, and brand. With that figured out, I was able to figure out that coding the logic based off each unique combination of sport, year, and brand was the way to go. It is tedious to have to add functionality per each unique combination of these three attributes, but since the brands have unpredictable changes depending on the sport and year, this was the best solution I found to almost guarantee 100% relevant search results on eBay so that I would be scraping price data for only the listings that were relevant matches to each of my test cases.

I had to slow down and learn a lot of new tactics to be able to web scrape in python. I had to learn how to use python’s BeautifulSoup library so that I could scrape the data I needed from ebay. In order to user BeautifulSoup correctly I had to figure out how the HTML was structured on eBay’s pages that return search results, and if eBay changes their structure or HTML code too much as they did at one point in the development process, then I need to alter my code to adapt to the changes. The last thing I needed to learn was cleaning my data. As a data lover, I had heard of data cleaning but never used it in depth. I studied the format of the results my web scraping was returning me and coded the logic necessary to parse out the irrelevant characters in the results and reformat the dates into the format I found most useful. I ran into two road blocks with eBay. First, eBay does not correctly sort by date with 100% accuracy, so I coded a sort to ensure the data being returned would always be correctly sorted by date sold. Second, eBay was returning a different total number of search results from my web scraping on different test runs. Upon extensive testing, examination of the code and the URLs built for web scraping, I have determined with high confidence that the issue is not with my code and is not something that I can control unfortunately. Coming to that understanding took a whole lot of testing and examining code.

I struggled with getting my web scraping eBay sold listings results and my OCR results even close to where they needed to be. I figured it out for the web scraping as aforementioned, but the OCR results have taken and are still taking a lot of time, research, testing, and experimentation. I started by learning two separate OCR engines, EasyOCR and Pytesseract, since one was not performing accurately enough over the other for me to be able to select one OCR engine as the sole source of OCR. I was not getting accurate enough results with the quality of images I was passing in, so I learned what DPI (dots per inch) was and set up scanning software that worked with my model printer to generate 1200 dpi generated scans of my cards. This did not suffice, so I made improvements by separating my OCR from text detection. After learning how to do text detection, I learned how to crop, create, and show images in python. I kept using the EasyOCR engine along with the Pytesseract because in addition to its outperforming Pytesseract in accuracy sometimes, EasyOCR had helpful built-in tools. One tool allowed me to do my text detection when passing EasyOCR the whole image for OCR. The second tool allowed me to take the results from that initial OCR run, which included bounding box coordinates for the areas where text was found, and convert the bounding boxes into cropped images. Learning how to crop and create a new image from the original along with using the cv2 library to show the image allowed me to pass cropped images to Pytesseract for translation and to view the discrepancies between what the text actually says versus what the initial OCR run interpreted the text as. Since Pytesseract and EasyOCR work differently, I researched the Pytesseract configurations (PSMs) and expanded to running the three PSMs that work best with cropped images of detected text. I also had to study the confidence levels being returned by the engines on how confident they are in the accuracy of their interpretations. All these technique took much research and testing. Additionally, even if I correctly interpret every character on a card correctly, I still need to figure out how to assign the relevant text to the correct attribute in a database. The OCR engines are not going to be able to tell me whether the text it interpreted is a player’s name, the team name, or brand name.

Another thing I struggled with was creating a pricing algorithm which could appropriately capture the current market price of a card. I decided to blend a mix of using an average sale price for a certain number of most recently sold listings and using an average sale price for all sold listings over a certain number of days. I set up the program to decide on which approach to use based on the number of total listings sold over a month.

Handling edge cases has been and always will be my biggest challenge. There is so much variety in card style, layout, card material type, font type, and colors that getting OCR to perform accurately for as many cards as possible is a big challenge. For my web scraping and pricing, there is a lot of variety even year to year for the same sport and brand, so identifying the correct keywords to use and omit from eBay searches for the perfect web scraping is also just as challenging.

**Future Enhancements:**

Test Plan:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component or Function | Requirements  Reference | Test Description | Test  Number | Test Date | Results |
| Web Scraper | #12Ncd1 | Scraping set up to run during lowest eBay site and the project system’s usage times | #3w | n/a | Inconclusive: script behavior not set up and UI not implemented |
| Card Pricing | n/a | Best offer listings returning their actual prices for build version 1.1 fix | #1.1p | n/a | Inconclusive: additional web scraping for best offer listings is an additional feature in build version 1.1 |
| Card Pricing | #12Ncd1 | Pricing Reliability test | #5p-b | n/a | Reliability – Inconclusive: enhancement item |
| OCR Program  and UI | #9Fa005 | 1200 dpi images are acceptable input for OCR from the UI specifically | #1ocr-b | n/a | Inconclusive: UI to OCR is a future enhancement |
| OCR Program | #9Fa006 | 95% of important text is detected | #2ocr | n/a | Inconclusive: good for image front but image back needs more work to test then pass |
| OCR Program | #9Fa006 | Highest confidence level OCR run is most accurate 90% of the time | #3ocr | n/a | Inconclusive: not confident in image front results and the image back needs work before it is testable |
| OCR Program | #9Fa006 | 1 of 5 OCR runs translates the text accurately 90% of the time | #4ocr | n/a | Fail: I have not gotten the OCR accurate enough to get an accurate translation 90% of the time yet. |
| OCR Program | #9Fa006  #12Ncd1 | Program will correspond the correct attribute with the interpreted text (90%) | #5ocr | n/a | Inconclusive: OCR unable to correlate results with the correct attributes |
| OCR Program | #9Fa007 | Machine learning uses best match images in identification to improve results | #6ocr | n/a | Inconclusive: machine learning not implemented and would be the last thing added |
| OCR Program | #9Fa008 | OCR successfully submits partial results to UI | #7ocr | n/a | Inconclusive: attribute assigning and UI not implemented |
| OCR Program | #12Na01 | The total OCR process will take 5 seconds per card with target time of 1.5 seconds | #8ocr | n/a | Inconclusive: The total OCR process is not fully implemented |
| UI | #9Fa001 | Search your cards using keywords | #1u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa002 | View value of all cards in profile for a selected filter | #2u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa003 | Anonymous card owners can not be seen by others | #3u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa007 | Users have a section available during OCR with best match identification options | #4u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa008 | Section exists for storing partial card identifications | #5u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa011 | An individual page per user’s card exists with graphed historical pricing data | #6u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa012 | Marking of user’s card quantities available for eligible cards in their collection. | #7u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa013 | Card attribute sorts, user created wish lists and folders function properly. | #8u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa013 | Application sorts take less than 4 seconds | #9u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa003 | Communication between users controlled by the card owner for privacy and security | #10u | n/a | Inconclusive: only have a partial UI template |
| UI | #12Na01 | Simple searches are under 2 seconds and complex searches under 4 seconds | #11u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa013 | Search filters do not return any out of scope results | #12u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa003 | Card owners can set a “card status” that other users can view successfully | #13u | n/a | Inconclusive: only have a partial UI template |
| UI | #9Fa014 | Card percent value changes are functional | #14u | n/a | Inconclusive: only have a partial UI template |
| UI | #10Nab1 | The ascetic of the UI will be professional and follow the requirements document | #15u | n/a | Inconclusive: only have a partial UI template |
| UI | #11Na01 | The UI help page is functional and contains the required materials per user feedback | #16u | n/a | Inconclusive: only have a partial UI template |
| UI | #11Nb01 | Personalization and Internationalization standards test | #17u | n/a | Inconclusive: only have a partial UI template |
| UI | #11Nde1 | Politeness and Accessibility standards test | #18u | n/a | Inconclusive: only have a partial UI template |
| UI | #12Na01 | Webpage total load time is 5-7 seconds | #19u | n/a | Inconclusive: only have a partial UI template |
| Database | #9Fa008 | Database successfully stores manually completed partial identifications | #1d | n/a | Inconclusive: do not have a normalized database |
| Database | #9Fa011 | Database stores pricing over time for all cards | #2d | n/a | Inconclusive: do not have a normalized database |
| Database | #9Fa012 | Database constraints work for storing card quantities | #3d | n/a | Inconclusive: do not have a normalized database |
| Database | #11Nb01 | User purchasing preferences will not be stored | #4d | n/a | Inconclusive: do not have a normalized database |
| Database | #12Ncd1 | Reliability test for storing a daily market price per card | #5d | n/a | Inconclusive: do not have a normalized database |
| Database | #12Nfg1 | Database capacity and scalability test for the dynamic storage implementation | #6d | n/a | Inconclusive: do not have a normalized database |
| System | #9Fa009 | System test OCR, pricing, storage, and viewing of various cards | #1s | n/a | Inconclusive: need all components to test |
| System | #14N001 | The student can unofficially privately maintain, improve, and commercialize the system | #2s | n/a | Inconclusive: need all components to test |
| System | #15Nb01 | The system will meet the Integrity and Privacy security standards | #3s | n/a | Inconclusive: need all components to test |
| System | #16Na01 | The system will meet the cultural standards | #4s | n/a | Inconclusive: need all components to test |

Note - The section below is for reference so the tests can be repeated and to read for more detail. To use just match the test number column with its respective test listed below.

**Card Pricing algorithm component tests:**

**Web scraping capability:**

**Test #3w** – Reliability for web scraping test is to ensure that the web scraping done in the card pricing is consistently done daily between 4-6 a.m. Eastern Time when the sales activity and site usage will be lowest for the main user demographics’ locations (#12Ncd1).

**Card pricing results:**

**Test #1.1p** – Test the build version 1.1 added feature that does additional web scraping for cards listed as best offer accepted. Make sure all the best offer accepted sold cards are found by the program and then that their actual prices are returned with the additional web scraping (requirement reference is n/a, refer to Test #1p).

**Test #5p-b** – (Reliability) and the pricing algorithm will run daily during 4-6 a.m Eastern time, which has been estimated as lowest site usage times, so that there will be accurate historical pricing data and minimal performance impacts for the main user demographic (#12Ncd1).

**Optical Character Recognition component tests:**

**Test #1ocr-b** – This test part is for the input from the UI to the OCR program. Make sure the 1200 dpi scanned images of the front and the back of a card are accepted as input to the OCR program from the UI (#9Fa005).

**Test #2ocr** – The OCR program can detect the presence of 95% of the text on the card’s front and back that will be used to identify all the card attributes listed in the database (#9Fa006).

**Test #3ocr** – Between all 5 OCR engine runs, most accurate result will be the result with the highest confidence level, with this happening at least 90% of the time (#9Fa006).

**Test #4ocr** – For each piece of relevant detected text in the image, the OCR program will have at least ONE running OCR engine correctly translate it with 90% accuracy (#9Fa006).

**Test #5ocr** – The OCR program will correctly identify the attribute that accurately translated text corresponds to with 90% accuracy and more complex attributes to correlate like if a card is autographed will have a 70% accuracy benchmark. Those are the minimums, with 95% accuracy for simpler identifications and 85% for complex identifications as optical performance benchmarks. An example of this is recognizing that “Josh Allen” is the player name attribute and not the product name (#9Fa006 & #12Ncd1).

**Test #6ocr** – The OCR will use machine learning to provide images to the user as potential matches with the card being scanned in, and the results will get more accurate with continuous user feedback when using the best match images. Attempts to confuse the machine learning with inaccurate results will be unsuccessful (#9Fa007).

**Test #7ocr** – OCR is able to submit partial results to a portion of the UI for manual completion later by the user (#9Fa008).

**Test #8ocr** – Card identification will take no more than 5 seconds per card and the target time will be 1.5 seconds (#12Na01).

**UI component tests:**

**Test #1u** - User shall be able to search for cards using keywords (brand, year, player, team, set, serial numbering, if it is autographed, and if it is graded (#9Fa001).

**Test #2u** - User shall be able to see the total value of a subset of their card collection in their profile through the use of filters such as by sport, player, or set (#9Fa002).

**Test #3u** - Users shall not be able to see the owner of a card in the database if the card owner has set themselves as anonymous for the card (#9Fa003).

**Test #4u** – A section of the UI will contain a confirmation screen where machine learning provided images are offered as possible best matches for the card currently attempting to be scanned in with the OCR (#9Fa007).

**Test #5u** – A section of the UI will exist to successfully store partially complete card identifications from the OCR program and the user will be able to manually complete those partial identifications (#9Fa008).

**Test #6u** – Users can view an individual page for each of their cards containing a graphical display of their card’s pricing history over time (#9Fa011).

**Test #7u** – When users own more than one of a card that’s in their collection, then users can mark the quantity of that card that they own if the card isn’t graded or serial numbered, which are unique attributes. In the UI, the price total for duplicated cards will also be calculated and viewable in addition to the price of one copy (#9Fa012).

**Test #8u** – Card sorts and User created wish lists and folders will successfully organize cards by their chosen sorting attributes and will gather all the specific cards chosen by the user in the created wish lists and folders. The UI will then also provide the total price value for a folder and wish list based off the cards it contains which have a price listed in the database (#9Fa013).

**Test #9u** – Results sorting like from searches in the database or sorts of a user’s portfolio of owned cards will take less than 4 seconds. An example sort would be listing the cards in a user’s collection from last entered to oldest card entry (#9Fa013).

**Test #10u** – User requests a social interaction with a card owner and the owner is able to control the interaction by accepting or declining to chat with the user. If the card owner is anonymous, then they will remain so with this test to the requesting user (#9Fa003).

**Test #11u** – The absolute max time allotted for any type of search (with and without filters) will be 4 seconds. Simpler searches with one filter or less should take no longer than 2 seconds max (#12Na01).

**Test #12u** – The cards status which indicated how likely an owner is to part with a card can be set successfully and viewed by other users. The statuses are interested/not interested in trading/selling (#9Fa003).

**Test #13u** – User searches with filters will not return results that do not fall within the filter’s scope. An example is that a player filter for Tom Brady cards will not return a result with Randy Moss as the player on the card (#9Fa013).

\*this requirement is also listed in #12Ncd1 with a benchmark of 99% accuracy.

**Test #14u** – Each individual card and folder of cards created by the user will have a percent value change number associated with them for whatever the default time period selected by the user is. The time options are a day, week, 6 months, a year, or their accounts time since created. Each of these options are expected to be accurate with this test (#9Fa014).

**Test #15u** – The UI will be professional, modern, and organized in appearance. Using agile framework and test user input, the design will be improved upon until a 4 star quality minimum is achieved (#10Nab1).

**Test #16u** – There will be a functional section of the UI designed to provide needed supplemental help materials for guidance in using the various components as determined by user feedback (#11Na01).

**Test #17u** – The website’s personalization and internationalization standards to test here are using USD pricing for all monetary values and English as the only available language (#11Nb01).

**Test #18u** – Test the website’s politeness and accessibility by getting user feedback on if the wording is professional and polite throughout the system (#11Nde1).

**Test #19u** – Webpage total load time is 5-7 seconds (#12Na01).

**Database component tests:**

**Test #1d** – The database will successfully store manually completed partial card identifications from the UI (#9Fa008).

**Test #2d** – The database is able to store historical pricing data for each unique card in the database over time, starting with when it is first entered into the database by some user and tracking it daily from then on (#9Fa011).

**Test #3d** – Card quantity is stored and monitored by constraints which ensure that quantity can not be marked for cards with a grade ID number or serial number present (#9Fa012).

**Test #4d** – User purchasing preferences will not be found stored inside the database (#11Nb01).

**Test #5d** – The database will collect from the pricing algorithm or through some other means store an accurate daily price value for each card for reliability purposes, which will ensure the store of historical data for each card (#12Ncd1).

\*If this test fails badly due to overwhelming performance and timing costs for daily pricing calculations, then potential future improvements ideas would be a code modification to calculate a card’s value for a past date using the same method for calculating the card’s value currently.

**Test #6d** – Capacity and Scalability tests: If using the dynamic database implementation, the storage usage requirements will be a 50% usage minimum to 80% capacity in use maximum optimally. The database should be running just as effectively at 50% capacity used as at 80% capacity used. This test also includes insuring that the database dynamically accesses additional storage resources when the capacity used breaches 80% (#12Nfg1).

**System tests:**

**Test #1s** – The tester shall be able to successfully complete the OCR, pricing, database storage, and UI viewing of a card for each of the following sports (football, baseball, basketball, racing) and each of the following attributes present on the card (autograph, memorabilia, graded, serial numbered, non-serial numbered parallel, rookie card, base card, insert). That is a total of 12-32 cards to test depending on the attributes present per card (#9Fa009).

**Test #2s** – The project manager, lead developer, and lead tester roles are all assigned to the student Riley Osborne, who will have the capability to independently take on all of those roles after board approval of the project as a side project in order to deliver on all of the future improvements and the required system maintenance. These roles main roles will be able to be delegated along with additional personnel into teams with defined scopes of work upon any potential commercial deployment. Proper licensing and proprietary ownership will be ensured solely to the student Riley Osborne before going commercial (#14N001).

**Test #3s** – The Integrity and Privacy of the system will be tested for each of the components to ensure that the user’s information remains private.

UI: the tester will not be able to use cookies, HTML/java script injections, or any other means to gather information on a user and their collection which is private. This included a secure and functional anonymous card owner feature.

Web Scraping: test whether the user’s cards that are marked as anonymous owner can be discovered by attempting to access the link used in web scraping.

Database: Ensure the user’s private information can not be accessed through attempted database exploitation (#15Nb01).

**Test #4s** – The system will culturally be tailored to US citizens and non disabled people (#16Na01).

**Defense Presentation Slides:**

<https://1drv.ms/p/s!AnMG4cNIPXDCgq5LUSBYYBbMfrSjoA?e=YQHh2c>