CS 3354 Software Engineering
Final Project Deliverable 2

Digital Repository and Auction for Cards (DRAC)

Austin Hunt
Ibad Momin
Nikita Ramachandran
Derek Zhou
Nabil Khan
Gaby Salazar
Joel Varghese

Delegation of Tasks

Ibad: Created and maintained the Github Repository, wrote the System Requirements, programmed the basic login and collection functionalities, and conducted Unit Testing on said functionalities.

Nikita: Created the Scope Document and worked on the Final Presentation.

Nabril: Described the Architecture Pattern used for the program and worked on the Final Presentation.

Derek: Wrote the Primitive Task Delegation, Market Comparison, Price Estimation, Software Costs, Market Comparison, and edited Slides.

Joel: Wrote about the chosen Development Process Model, Hardware Costs, and Personnel costs, in addition to creating the Software Process Model.

Austin: Created the Use Case Model, wrote the Response to Instructor Feedback, Detailed Delegation of Tasks, and Conclusion, aided Derek in the calculation of the Price Estimation, and formatted the report.

Gaby: Created the Sequence Diagrams, wrote the Development Schedule and worked on the Final Presentation

Part 1: Project Deliverable 1 Content

I. ADDRESSAL OF FEEDBACK

Original Proposal Document:

Title: DRAC (Digital Repository and Auction for Cards)

Group Members:
Derek Zhou, Nikita Ramachandran, Ibad Momin, Austin Hunt, Nabil Khan, Gaby Salazar, Joel Varghese

What we will be doing:
We will be creating “Drac” a card management software for trading card games such as MTG or PTCG, which has trading functionality and market functionality, as well as database lookup and deckbuilding.

Motivation:
Most of the members in our group play or have played card games, and so we wanted to make a database to sort out systems that we use in our games, like card sorting, deckbuilding, and live price lookup. We expect this tool will be used to help players sort out their cards and plan our future decks for tournaments or competitions.

Workload:

Derek Zhou	**Database Engineer:** will create and maintain the database to store user’s information and cards they have in their portfolio.
Nikita Ramachandran	**UI UX Designer:** will design a user-friendly and visually appealing interface for a mobile app and website, ensuring intuitive navigation and seamless experience for users.
Ibad Momin	**Devops Engineer:** will implement and manage continuous integration and continuous deployment (CI/CD) pipelines to

	streamline software development and ensure efficient and reliable delivery.
Austin Hunt	**Project Manager:** will coordinate project activities, allocate resources, and track progress and use agile methodologies to ensure the timely and successful completion of software within scope, schedule, and budget**.**
Nabil Khan	**Solutions Architect:** will design and create the high-level technical solution blueprint, including system architecture, integration strategies, and technology stack selection, to align with business requirements and facilitate the successful development of software projects**.**
Gaby Salazar	**Backend Engineer:** will develop and maintain the server-side components and logic of the software application, ensuring efficient data processing, robust functionality, and scalability
Joel Varghese	**Frontend Engineer:** will develop and maintain the client-side components, implement UI designs, and integrate with backend systems while prioritizing speed, accessibility, and responsiveness.

Instructor Feedback:

> "A lovely topic!! Once complete, it will be very useful for those who are to meet game card exchanges in the cyber world. Please consider implementing it fully, if you can. No pressure w.r.t. grade on implementation. In the final report, please make sure to include comparison with similar applications -if any-, make sure that you differentiate your design from those, and explicitly specify how. Fair delegation of tasks. Please share this feedback with your group members. You are good to go. Have fun with the project and hope everyone enjoys the collaboration."

Response:

> We will distinguish ourselves by combining several existing services into one. For instance, *TCGPlayer's* marketplace is already similar to what we have in mind, but their collection management system is fairly bare bones, lacking the variety of viewing and playtesting options many players want. Conversely, higher quality collection managers like *Moxfield* lack the capabilities of online marketplaces, and oftentimes are limited to only one card game. We aim to combine the capabilities into one service to streamline the processes of collection management and card exchange.

II. GITHUB REPOSITORY

Link to Repository: https://github.com/ibadamomin/CS3354-DARC

III. DELEGATION OF TASKS

Ibad	Github Repository (2), System Requirements (5)
Nikita	Github Repository (2) (Scope Document)
Nabril	Architectural Design (9)
Derek	Delegation of Tasks (3), Class Diagram (8)
Joel	Process Model (4)
Austin	Addressal of Feedback (1), Use Case Model (6), formatting
Gaby	Sequence Diagrams (7)

IV. PROCESS MODEL

The process model that we have chosen to work with is the Incremental model, as card game communities tend to be very specific about the systems they use, and launching into a saturated market means that we need as much feedback as possible in between releases in order to keep up. Additionally, many of the subsystems we have in mind should all be capable of functioning independently, so developing in increments would reduce overhead. A database would likely be the first component created, then a collection and account system, then a seller/buyer system integrated into the accounts, then finally the display systems, deckbuilding, etc. This way, each part of the system can be made as good as possible before a full release. In addition to this, the project should always be being improved, and so the incremental model can allow for continuous improvement throughout the lifetime of the system.

V. SOFTWARE REQUIREMENTS

Functional Requirements:

1. User should be able to login using their credentials and access their account.
2. User should be able to view and organize their collections and decks.
3. System should update each card's current market price based on the current asks and bids.
4. User should be able to buy and sell cards in the cards marketplace.
5. After each trade, the system should update the inventory of the user.

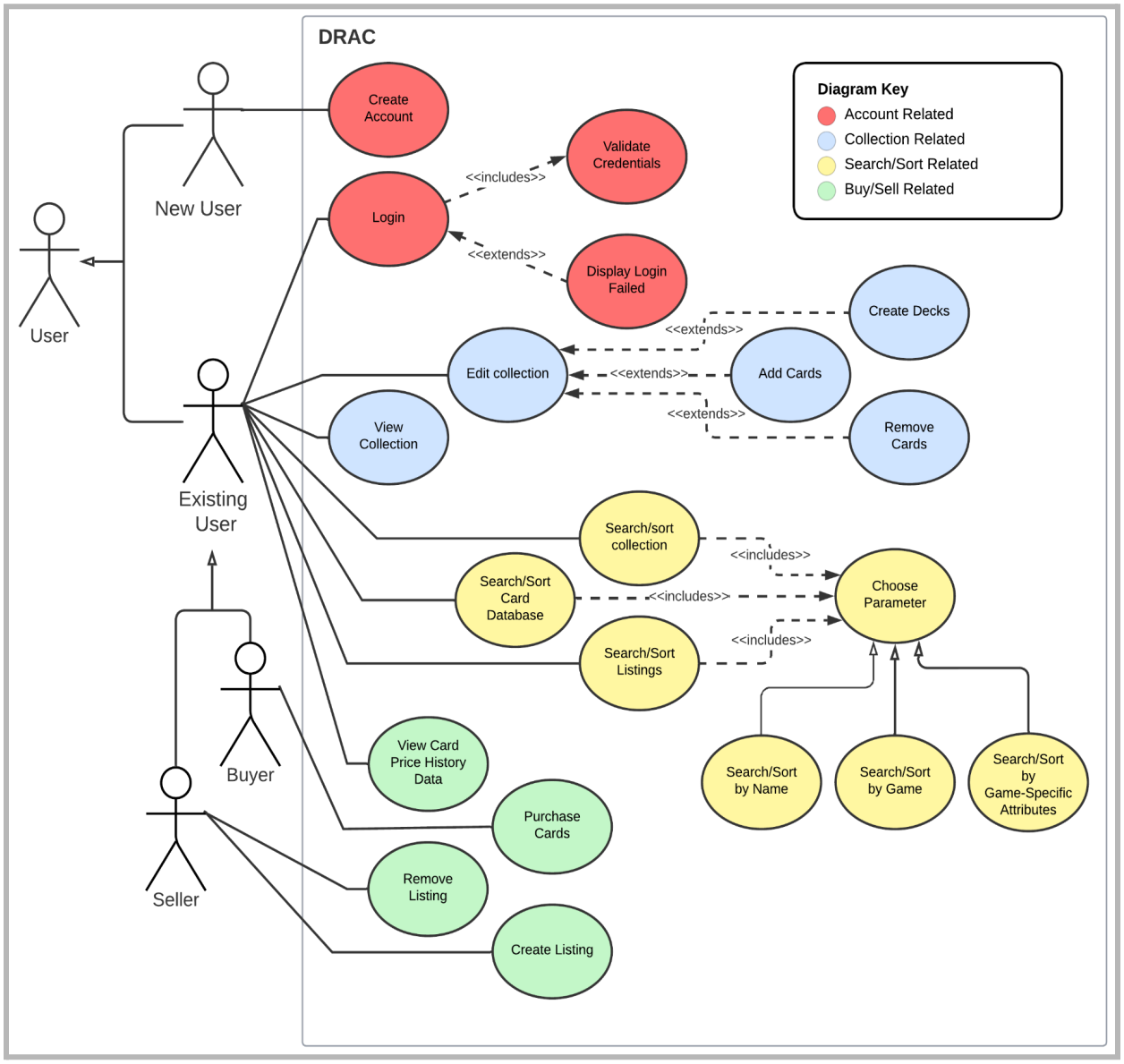
Non-Functional Requirements:

1. Usability requirements:
    a. *User Interface (UI) Efficiency:* The application should have an intuitive and user-friendly UI to ensure that users can navigate and perform tasks with minimal training.
2. Performance requirements:
    a. *Response Time*: The application should respond to user interactions within 2 seconds or less to provide a smooth and responsive user experience.
    b. *Scalability:* The system should be able to handle a growing number of users and card listings without significant performance degradation.
3. Space requirements:
    a. *Data Storage:* The application should efficiently manage and optimize data storage to prevent excessive disk usage and ensure quick access to data.

4. Dependability requirements:
    a. *Availability:* The system should be available 99.9% of the time to ensure users can access the marketplace consistently.
    b. *Fault Tolerance:* The application should be able to recover gracefully from unexpected failures to minimize downtime.
5. Security requirements:
    a. *Data Encryption:* Sensitive user data, such as payment information, should be encrypted both in transit and at rest to protect against unauthorized access.
    b. *Authentication and Authorization:* The application should have strong authentication and authorization mechanisms to ensure that only authorized users can access specific features.
    c. *Data Privacy:* User data must be handled in compliance with relevant data protection regulations, such as GDPR.
6. Environmental requirements:

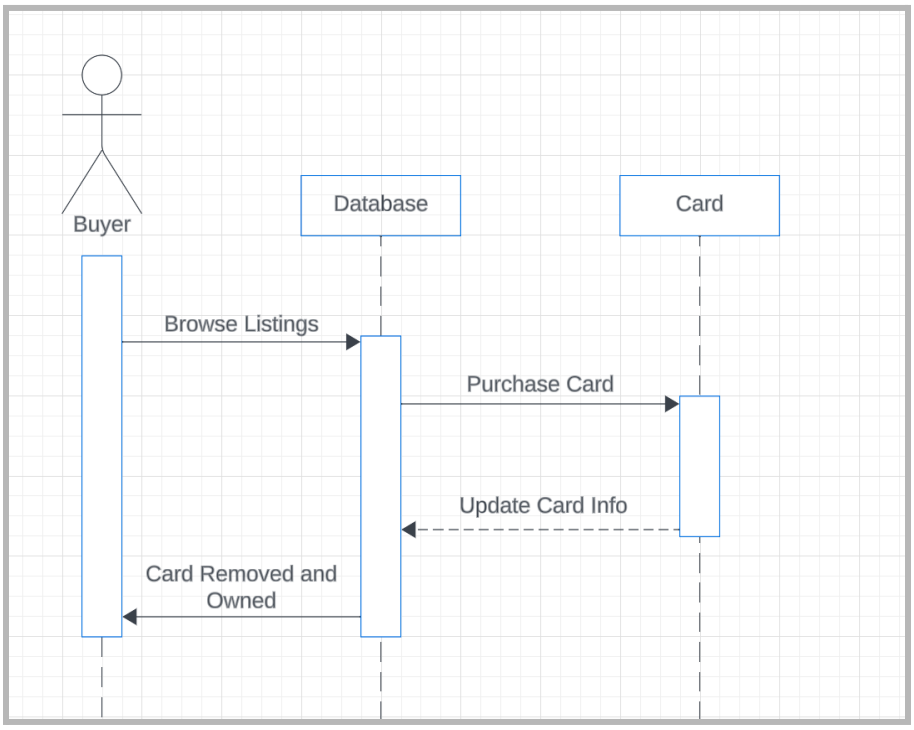
    a. *Energy Efficiency:* The application should be designed to minimize energy consumption and reduce its environmental impact.
7. Operational requirements:
    a. *Maintenance and Support:* A support system should be in place to address user issues promptly and provide necessary updates and maintenance.
    b. *Backup and Recovery:* Regular data backups and a robust recovery plan should be in place to prevent data loss and minimize downtime in case of system failures.
8. Development requirements:
    a. *Code Quality:* The application's code should follow best practices and coding standards to ensure maintainability and extensibility.
    b. *Testing:* Rigorous testing procedures, including unit, integration, and user acceptance testing, should be in place to ensure the application's reliability.
9. Regulatory requirements:
    a. *Compliance:* The application should comply with all relevant local, national, and international regulations, including tax laws and intellectual property rights.
10. Ethical requirements:
    a. *User Data Ethics:* The application should respect user privacy and data ethics, ensuring that user data is not used for unethical purposes.
11. Accounting requirements:
    a. *Financial Reporting:* The application should have mechanisms for accurate financial reporting, including revenue tracking, transaction records, and tax compliance.
12. Safety/security requirements:
    a. *User Safety:* The application should incorporate features to ensure the safety of users data and authenticity of cards user buys on the marketplace.

VI. USE CASE DIAGRAM

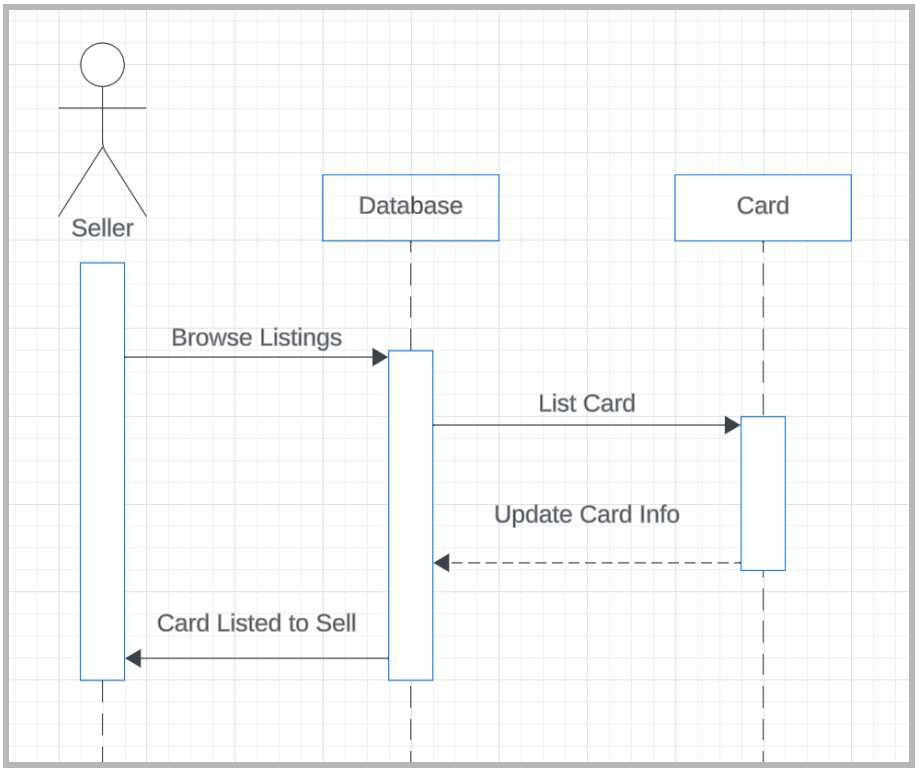


VII. SEQUENCE DIAGRAMS

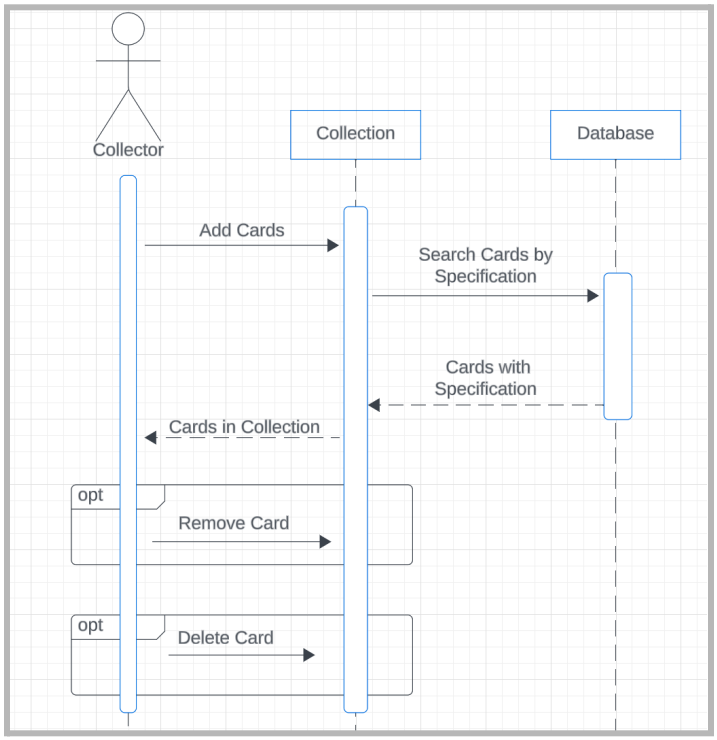
Purchase Card



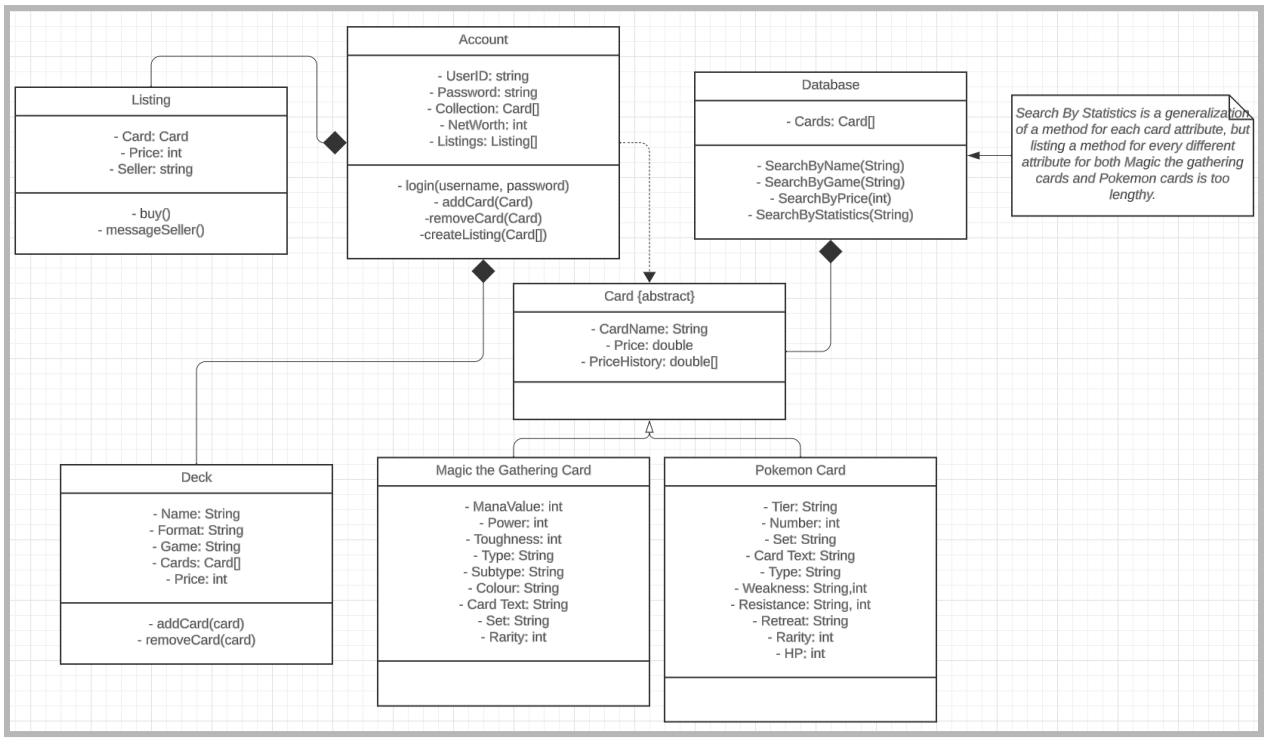
Create Card Listing



Edit Collection



VIII. CLASS DIAGRAM



IX. ARCHITECTURAL DESIGN

We have chosen to use the MVC Architecture Pattern.

Model Component:

1. Card Model
    a. Card Name
    b. Card ID
    c. Card Type
    d. Card Statistics

2. Collection Model
    a. Cards
        i. List of Cards Owned
    b. Decks
        i. Deck Name

3. User Model
    a. User Information
        i. User ID
        ii. Login Credentials

4. Trade Model
    a. Trade Transaction
        i. Trade ID
        ii. Initiator
        iii. Receiver
        iv. Cards Traded

5. Market Model
    a. Card ID
    b. Price of Card
    c. Listing Date
    d. Seller Information
    e. Quantity

View Component:

1. Card View
    a. View list of cards owned
    b. View details of cards

2. Inventory View
    a. View decks
    b. Manage deck-related actions (ex. add/delete)
    c. Put cards on market
3. User Profile View
    a. View user details
    b. Display owned cards

4. Trade View
    a. View trade requests
    b. View trade history
    c. Interact with trade options

5. Marketplace View
    a. Search for cards
    b. View available cards for purchase
    c. View cards listed by user

Controller Component:

1. Card Controller
    a. Handle interactions with card management

2. Inventory Controller
    a. Manage actions related to decks
        i. Creation
        ii. Deletion
        iii. Modification

3. User Controller
    a. Handle User Data
        i. Registration
        ii. Updates
        iii. Deletion

        iv. Authentication

4. Trade Controller
    a. Manage Trade Actions
        i. Initiate Trade
        ii. Accept Trade
        iii. Reject Trade
        iv. Register Trade

5. Marketplace Controller
    a. Manage Listings
        i. Buying Cards
        ii. Sellings Cards
        iii. Editing Listing

Part 2: Project Deliverable 2 Content

I. Costs & Scheduling

Cost & Effort Estimation

We will be using function point analysis to determine the cost and effort requirements for this project. We will start by classifying each functionality as either a User Input, User Output, User Query, Data File, or External Interface. We then estimate the complexity of each task based on personal experience, and use the following weights to calculated that functionality's function point value:

	Function Category	Count	Complexity		
			Simple	Average	Complex
1	Number of user input		3	4	6
2	Number of user output		4	5	7
3	Number of user queries		3	4	6
4	Number of data files and relational tables		7	10	15
5	Number of external interfaces		5	7	10

These values are then summed to calculate the gross function point value of the software.

<u>User Input:</u>

1. Login (average)
2. Register (average)
3. Collection updates (all simple)
    a. add card
    b. remove card
    c. add deck
    d. remove deck
    e. edit deck details
4. Profile editing (all simple)
    a. Username/Password change
    b. Account detail changes
5. Create Listing (simple
6. Remove Listing
7. Edit listing
    a. Change price
    b. Change dates
8. Bid on Listing (high)
9. Purchase Listing (high)
10. Accept/Deny offer (simple)

12 simple, 2 avg, 2 high = **56 total FP**

User Output:

1. Show Collection (high)
2. Show Deck Content (high)
3. Show all Listings (average)
4. Get profile (seller, buyer etc)
5. Display Card + information (simple)
6. Display Listing + information (simple)

3 simple, 1 avg, 2 high = **31 total FP**

User Queries:

1. Search Card in Database (high)
2. Search Card in Collection (average)
3. Search Profile in Database (simple)
4. Search Card in Listings (high)
5. Search Profile in Listings (simple)
6. Sort Card Display (high)

2 low, 1 avg, 3 high = **28 total FP**

Data Files

1. Pull user Information
    a. Username (simple)
    b. Password (avg)
    c. Card Collection (high)
    d. Payment information (avg)
    e. Listing history (avg)
    f. Purchase history (simple)
    g. Trust Score (simple)
2. Session information (avg)
3. Pull card from Database (high)
4. Pull card from Collection (avg)

3 simple, 5 avg, 2 high = **101 total FP**

External Interfaces:

1. External Card Database (average)
2. Card prices (simple)
3. Card Price History (high)
4. Payment gateways (high)

1 simple, 1 avg, 2 high = **22 total FP**

Gross Function Points:
56 + 31 + 28 + 101 + 22 = 238

Next, the Program Complexity Score, by rating a set of 14 heuristics on a scale of 0 to 5 based on their relevance to the project (0 being no impact, 5 being essential) and summing their scores. These ratings are also estimated based on personal experience.

1. Reliable Backup/Recovery:	**3**	8. Online Master File Updates:	**4**
2. Data Communications:	**4**	9. Complex Inputs/Outputs/Queries:	**1**
3. Distributed Processing:	**1**	10. Complex Internal Processing:	**2**
4. Performance Criticality:	**2**	11. Reusable Code:	**0**
5. Use in Existing Environments:	**5**	12. Design Includes Installation:	**0**
6. Online Data Entry:	**5**	13. Use by Multiple Organizations:	**0**
7. Multi-Operation Data Entry:	**3**	14. Ease of Use:	**5**

Total Program Complexity Score: 35

The final functions point value is calculated using the following equations:

FP = GFP * (0.65 + 0.01 * PCS) = 238 * (0.65 + 0.01 * 35) = 238 * 1 = 238

Assuming our team is capable of a functionality of 60 function points per person week (working 40 hours a week, no weekends), the expected duration of effort would be 238 / 60 = 3.967 person weeks. With a team of 7 developers, the total project time would be around 4 days.

Personnel Costs:

The estimated cost of personnel for DRAC involves various different considerations for salaries and training costs. Based on the project roles and industry standard salary ranges, we can provide a rough estimation for the average monthly cost per person. Based on averages from indeed.com we can break down the average cost per month of each position on our team to eventually determine the overall average cost of personnel per month for a seven person team. All averages are based on indeed's estimations for that job title in Dallas, TX. Our team consists of a Database Engineer, UI/UX Designer, DevOps Engineer, Project Manager, Solutions Architect, Backend Engineer, and Frontend Engineer. On average, in Dallas, TX, the monthly salaries for these positions are as follows: $7706, $6783, $8775, $6174, $9972, $9709, $7590. This is $56,709 per month in total which means that the average cost per month per person is $8,101.28 for our seven person team.

Software Product Costs:

Surprisingly, there should be almost no external costs incurred by this project. Licenses for softwares should mostly be from open-source server management software, and should as such bring no additional costs. For the legality of the card game, each company has a different policy on fan content and websites using their cards, but each of them boil down to nothing[1,2]. The two games we are focusing on are Magic the Gathering and Pokemon, and each of these card games just require that the external fan site or website be free to access - the usage of their assets cannot be commercialized.

Hardware Product Costs:

The estimated cost for hardware products can be estimated by looking at the pricing models of popular cloud service providers like AWS, Azure, and GCP. Cloud services could become a key component for hosting our application and databases while ensuring scalability. At the moment, considering a lower usage of cloud services, the estimated monthly cost for this type of infrastructure could be anywhere from $0 - $800 per month [3]. At the maximum scope, this comes out to about $9,600 per year.

Project Schedule:

The start date of the project would be November 20th 2023 and the end date would be November 23. Based on the previous calculations, the duration of this project would roughly be around 4 days. As previously stated, weekends will not be counted in our schedule and our group members will be working at least 8 hours a day.

Total Cost and Pricing:

Based on the estimated monthly labor price of $56,709 per month, this 4 day project would have a total labor cost of $7,561.20. However, due to the high amount of uncertainty in our effort estimations, we will double the expected duration for the purpose of pricing. Thus the estimated labor cost would be $15,122.40. As there are no additional hardware or software costs to consider during development, as the cost of cloud computing would be the responsibility of the customer, this would be the minimum break-even price of the product. Further pricing discrepancies would depend on the customer and the financial situation of our development team.

II. Test Plan

Prototype:

In the process of designing and coding a Minimum Viable Product (MVP) for a software named DRAC, we developed login and sign-up pages for users. After a user successfully logs in, the system prompts them to enter the card information they wish to add to their portfolio. To verify our code, we conducted tests on the userAuthorization() method, responsible for authorizing users to log in if the entered username and password exist in our database. For the implementation of this program, we opted for Java, leveraging the experience of the team members with the language, allowing us to create a minimum viable product with minimal technical complexity.

Unit Testing:

To conduct the method testing, we employed JUnit testing. We provided the test with two valid usernames that we added to the database for testing purposes, and two invalid usernames that do not exist. The test results confirmed our expectations, as the two valid usernames returned a true output, while the other two returned false, indicating successful test completion.

III. Market Comparison

Introduction:

There are many other comparisons for this particular type of product, but the market remains spread out - most players use multiple different websites at the same time, as each website is distinctly lacking in certain areas.

Markets - (TCGPlayer):

TCGPlayer is the best market system for any card game, including Pokémon and Magic: the Gathering. However, it is also very cluttered and visually unappealing, making it harder to use. It also lacks a fleshed out collection/card system, which also leads to a shallow search system - only searching names and basic details.

Deckbuilders - (Moxfield/Tapped out/Archidekt):

These are all deckbuilders, which are online systems used to keep track of one's cards and also their decks. However, these websites all do not have any sort of primary market - they do not have their own market, and end up quoting prices from other websites and redirecting the user, making the process unnecessarily difficult. Quoting from external websites also means there is conflicting information - often times, prices are much lower (and sometimes, albeit rarely, higher) than the price listed on these websites. Having a first-party market would alleviate these issues, and would also allow for more complex pricing data, like price history. Additionally, as a deckbuilder, it is often important to see card statistics - what cards are popular in professional play, what cards are often used in combination, etc. These websites also do not have this functionality.

Databases - (Gatherer/Pokémon Trading Card Database):

Gather is Magic: the Gathering's card database, and the Pokémon Trading card Database is Pokémon's. These systems are barebones, only implementing a database and nothing else - no market, price or deck system. Magic: the Gathering has a similar database, Scryfall, which is much more powerful, and also shows some card/price statistics, but still does not implement a market or collection system.

Meta Analysis - (EDHREC):

EDHREC is a meta analysis website - it analyzes the meta by scraping data from other deck builders and shows users how cards are used together, and what combos exist, or tend to exist, with certain cards. It has no collection system or deckbuilding support, and no first-party market, but it does show prices pulled from other websites.

Pricing Statistics - (MTGStocks):

MTGStocks is a website for tracking the prices of cards, and also produces articles about cards, formats, and upcoming sets. It also has no other systems in place.

DRAC:

Our app aims to combine the most important factors of these together - using a first-party market, we can create a price history in a much simpler manner, and using a first-party deck builder, we can create a meta system in a similar fashion. It is also possible to scrape from MTGStocks and EDHREC to bolster data. This can allow for a smoother deck building experience - rather than trying to trawl through 30 years of cards to find a good

match for a deck, users can be recommended cards based on their archetype, or how they are trying to build a deck. Then, when they create their deck, they can place their order right then and there, rather than going to a third party website and being hit with other costs they were unaware of.

It should also be noted there is also a gap in the market for these kinds of websites for Pokémon. While they do exist, none are as concrete as Magic: the Gathering's.

IV. Conclusion

While the overall planning of this project went well, we did have to make some slight compromises to our original plans. While we are content with our execution of the marketplace feature planning, we were not able to flesh out the collection management system as much as we wanted to. This was the key feature that would have differentiated our product from existing programs (namely TCGPlayer). However, due to a lack of experience with collectable trading cards on the development team and the amount of planning required for even the basic collection management, we were not able to reach our initial expectations. Future development of this project should focus on bolstering the collection manager, such as by adding features such as a playtesting environment or game-specific card attribute searching.

Works Cited

[1] "Fan Content Policy," *Wizards of the Coast*, Jun. 01, 2022. https://company.wizards.com/en/legal/fancontentpolicy

[2] "Assets Use Terms - The Pokémon Company North America Official Press Site," *press.pokemon.com*. https://press.pokemon.com/en/Assets-Use-Terms

[3] A. Rai, "What is the Cost of Cloud Computing for Small Businesses?," *blog.economize.cloud*, Oct. 20, 2023. https://blog.economize.cloud/small-businesses-cloud-computing-cost/ (accessed Nov. 18, 2023).