Yu-Gi-Oh Card Stock system: Analysis

# Problem Background and Propositions

## What is Yu-Gi-Oh?

Yu-Gi-Oh is a turn-based physical trading card game, with many small shops involving themselves in the business of buying, trading for and selling the cards, sets of cards and booster packs for the game either individually and in bulk.

### Simplified card anatomy

The simplified anatomy of a card down to:

* The Type of card (Monster/Spell/Trap card)
* What the card does to the flow of the game & match (eg: “Draw 2 cards next turn”)
* A unique card and set ID to identify each individual card (all have a set ID but not all may have a cardID, though further research is needed into the exceptions.)
* Other attributes the card possesses (eg: Fire type, machine, quick spell. All depends on the kind of card.)

### Assessing the value of a card

The factors that determine the value of a card are as follow:

* The Rarity of the card, depending on where it came from and how likely you are to obtain it.
* How used the card is, any severe damage will drastically impact the overall price of the card.

In order to simplify the system, the card shop’s system does buy or sell any cards that are on the scale of being a collector’s item, merely cards that are intended to be used in play.

#### Card Wear Quality Tiers

Within the current system, quality is separated into different categories, using the website CardMarket.com (Cardmarket, n.d.) as a guideline:

* Factory-New (FN) (CARD SETS & BOOSTER PACKS ONLY)
  + Brand new pack of cards, only applies to card packs and untampered sets
* Near-Mint (NM)
  + Preserved & generally clean, but may have minute damage in form of white spots
* Excellent (EX)
  + Damage visible upon closer inspection, however in very good condition
* Good (GD)
  + Damage that is visible, however no damage in form of bending or water damage on card
* Light-Played (LP)
  + Card has seen a bit of wear, however isn’t tampered with via marker or other scribbling.

##### Additional damages and wears

Additional information uses a key word to convey any additional damages or properties of a single or multiple cards, they are as follows:

* Bent
  + Card shows signs of bending, placing the card into the Light-Played Category
* Scratched
  + Self-explanatory, this will limit the card’s quality to Good at best
* Water Damaged (Water)
  + Only cards that have sustained minor water damage may be accepted, being limited to the Light-played category.

# Current System Visualised



# Current System Summarised

## Information stock log holds

The stock log, which is a multi-page physical paper document with rows and columns, stores the following information:

* Name of card
* Individual ID of Card
* Card Set ID
* Quantity of Card
* Quality of card \*
* Additional information (reserved for quality of card, often will be blank) \*

\*Refer to The section **“Assessing the value of a card”** in the document

## Types of goods being sold and bought using the system

These are the only acceptable goods that this store will be involved in:

* Individual cards for purpose of playing, which are of acceptable quality
* Unopened Sets of cards
* Unopened card booster packs (random cards inside, quantity may differ)

If any of the following points are met, the shop will **not** make a purchase:

* If the card is signed
* If the damage on the card is severe enough for it to not be playable in a sleeved deck within a tournament (If it lands in the Played (PL) category of cards (Cardmarket, n.d.))
* If the card is a collector’s item, meaning that if
  + The card is scarce
  + The card has been pristinely preserved
  + The card or cards are part of a discontinued and unplayable set, such as the first generations of cards.

## How Current System Operates

### Selling Cards

1. Customer asks reception “hey have you got this card/these cards?”
2. Reception person asks stock manager to check if the specified item(s) are in stock
3. Stock manager goes to the stock log and reads through it, using the name of the card or the ID of the card
4. If not in stock, stock manager tells reception no and if yes, stock manager goes to fetch the card(s) in question
5. Stock manager updates the stock log by removing the items taken out of the stock
6. Stock manager returns to reception desk with the goods
7. Transaction is made

### Restocking Cards

1. Delivery person enters and hands over a document of the transaction (or a document has been produced prior by the receptionist)
2. Delivery guy is told by reception where they should leave the goods
3. Receptionist notifies stock manager
4. Stock manager picks up the fresh batch delivered
5. Stock manager adds and sorts the new cards into the stock’s storage appropriately
   1. The stock manager may request the receptionist to hand him over the document temporarily that lists the contents of the box, in order to verify the contents.
6. Stock manager updates the quantity of any existing cards with the same quality, adding any new cards that are unique on as well.
7. Stock manager notifies receptionist that everything has been sorted
8. (if not done so already) receptionist stores the transaction document in a place to keep as a physical backup document.

### Restocking through customer

1. Deal is made with a customer and a sum is paid out to them
2. Receptionist notifies stock manager
3. Stock manager takes the newly acquired card(s) and sorts them into the stock storage
   1. Stock manager may ask for a document to verify the quantity and quality of the bought cards
4. Stock manager updates the stock log, adding any new card entries and updating the quantity column for any existing cards that are in stock
5. Stock manager notifies Receptionist, who (if they haven’t already) stores the transaction document detailing the pricing, quantity and names of the purchased items: this is stored in a physical location where the rest of the transaction longs are held.

### The role of the supervisor

A supervisor within the current system is a person who will take complaints, but also at any point can see how a process is taking place, as well as check transaction histories to ensure a consistent workflow is achieved by the employees.

## Problems In Rich Picture Diagram

One of the problems with the current system is the lack of proper method of communication between the reception desk and the stock manager; here are a few examples taken from the supervisor:

* “There were many times when the stock manager was unable to contact the stock manager due to them being busy sorting out the stock storage or updating the stock log”
* “There was a time where Sharron, our receptionist, forgot to notify the stock guy about this new package that arrived: it stood there for almost a week!“

A Second problem prevalent is a major one and it is the physical limitations of the stock log: the numbers in the list refer to the numbers as seen on the diagrams within the **Current System Visualised** section of the document

* [2.1 & 4]: The log book is misplaced or missing or damaged or has been stolen.
* [2.2]: The log book has missing records
* [3]: The log book hasn’t been updated when an item was taken out already

# Overall Problem Summary

The primary problem with the current system is inconvenience and unreliability:

Therefore, the clients I wish to target for the new implementation are:

1. The Stock Manager
2. The Receptionist
3. The Supervisor, monitoring the whole process

The reason that I wish to implement a new, digital system is it will increase the ease of access for all of the mentioned personnel, but also allow for shifts in and extra activities that the roles can do in order to save time, primarily: the reception desk can remotely access the stock log from her desk in order to check if the specific item is in stock, also the stock manager can update the stock log more reliably and is able to more easily correct any mistakes that would have been permanent in a physical log book, such as a mistake with the ID of a card wrote in pen in the middle of a set of logs.

Additionally, the digital implementation of the system will make going back through transaction logs faster, as the stock system can hold a transaction history.

Last and most important reason is the ability to create frequent backups to the log, meaning that in the event that the original was destroyed, there is at least 1 additional backup version, (which should be backed up every change and perhaps a copy of multiple backups should be made, up to some time frame where old backups are deleted.)

# Roles of users within the system

## Selling Stock

|  |  |  |  |
| --- | --- | --- | --- |
| User | Their Function in the system? | Primary/Secondary User? | Are they processing/Inputting/Outputting Data? |
| Emily (Receptionist) | Take an order and pass it onto Stock Manager. | Secondary | Processing Data & Outputting Data to Stock Manager |
| Dave (Stock Manager) | Updates & checks the stock log & stock. | Primary | Inputting, Outputting and Processing data to, from and within the system. |
| Customer | They provide the receptionist with their order they wish to buy. | Secondary | Inputting their order |
| Supervisor | Look at transaction history, the whole system and backup copies of both. | Secondary | Processing Data from logs & input any waned changes to be carried out by the employees. |
| Engineer | Debugs the system | Primary | Alters, Reverts, and tests the integrity of the system |

## Refilling/Buying stock

|  |  |  |  |
| --- | --- | --- | --- |
| User | Their Function in the system? | Primary/Secondary User? | Are they processing/Inputting/Outputting Data? |
| Emily (Receptionist) | Takes the transaction log and passes it onto stock manager, also stores a physical backup copy. | Secondary | Processing & Outputting order form to the stock manager |
| Dave (Stock Manager) | Using the transaction log, they check the order & update the stock log accordingly. | Primary | Inputting & Processing that relates to inputting the right information & checking if the order form matches what has been delivered. |
| Delivery Person | Gives transaction log to the supervisor and leaves the package to be collected by stock manager. | Secondary (Outside of company) | Inputs the transaction order and physical data to be picked up down the line by the stock manager |
| Supervisor | Look at transaction history, the whole system and backup copies of both. | Secondary | Processing Data from logs & input any waned changes to be carried out by the employees. |
| Engineer | Debugs the system | Primary | Alters, Reverts, and tests the integrity of the system. |

# User Needs & Wants for New System

## Receptionist

* I want to be able to access the stock log so Stock Manager doesn’t need to go and do it, also it’ll just save time on both their and the customer’s part
* I want to more easily see the changes done to the log so that I don’t need to sift through a load of papers and spend a lot of time looking for things in case that there’s a problem.
* When I leave the order to be filled into the stock log to the Stock manager, they often take way too long! I want to be able to access the stock log myself and be able to make changes myself.
* Making backups is very hard when there’s a lot in our stock or when someone damages the current log, I want to have the option to create a backup with the press of a button!

## Stock Manager

* Every time Receptionist needs to check the stock log she goes and asks me to do it because she’s busy.
* If Receptionist isn’t busy and takes the stock log, I cannot view nor update it while she has it with her!
* It can be tiring walking around with this big stock log, or walking to and from the desk it sits on: I would want to have better accessibility from where the stock actually is and where I can get access to the stock log.

## Supervisor

* I don’t want to go through stacks of paper to monitor the transaction history and order documents, I want a simplified view for every time an order was put into the stock log and who did it!
* I want to be able to access the database from home.
* I want to make sure that no rogue employee can just go and destroy out stock log!
* If there’s a way to make backups more frequent that’d be great, sometimes the stock manager can get a bit clumsy and damage it quite often during break times!
* When you make that system of yours, I want to be sure I can manage, maintain, and debug it properly! (THIS POINT IS CARRIED OVER TO THE ROLE OF AN ENGINEER, CURRENTLY THE SUPERVISOR FILLS IN BOTH OF THE ROLES UNTIL FURTHER NOTICE)

## Customer

* I don’t want to be waiting around as much for feedback for them to tell me that they don’t have the thing I want in stock!
* I don’t want to wait around hoping to get the thing I want to order, only for the receptionist to tell me that there was a mistake with the system or something and that they don’t actually own the thing they said few minutes ago that they did! That’d be outrageous!

# Data Collection and Storage

## Current system data collection

The current system stores the transaction logs for any re-stocks that are purchased by the manager (by an external contact/system), a physical copy detailing the date, time amount and company which delivered the goods to the physical location.

For any time that a customer may want to manually sell at least 1 card to the store, the details, such as Name, Date of Birth, contact details and address must be stored for 5 years (or any other reasonable time period).

## New System’s expected data collection

The new system will have to store all of the information the current system is required to store, however in addition to this, it has to securely store passwords used as proof of authentication to commit certain actions that require an elevated access level: for example, a receptionist may view the stock, but is unable to add nor delete any elements to it, while a stock manager would be able to.

Any password made by an employee can be changed so long they remember it, if not then a verification process may be required, meaning that the system may need to store contact details of the employees as well.

# Indirect Benefits of New System

* Both the stock manager and receptionist will be able to view the stock log and any transaction logs simultaneously.
* Transaction logs will be easier to search, with a friendly interface.
* Adding and removing new stock will be easier and will take less time to do so.
* An ability to quickly create backups of the system and the data on it.
* No more accidental physical damage to the stock log.
* Easier to undo a typo mistake.
* Certain typo detection for fields and required fields will need to be filled in to proceed.

# Feasibility

## Technical

* Minimum of 3 computer devices are required.
* A central server machine OR cloud storage is required.
* Visual Studio monthly subscription license required.

## Economic

* Monthly cost of Visual Studio Business licence. ($45 per month)
* Training or Hire of an employee to maintain the security of the central server device and data stored. (If no online functionality will be present, then only account for local and physical attacks.), needs to also know how the system and it’s database operates in the event of a fault with either system or data stored.
* A back-up drive is required to hold any backups of the new system.
* (IF SERVER AND BUDGET ALLOWS THIS) A Backup power source unit in the event of a power outage.
* Minimum wage for an under-18 worker (£4.35 an hour at the time of writing, 30th October 2021) for the duration of the time I work on the design, implantation and testing of the new system.
* Purchase of any Hardware that is essential for the functionality of the new system.
* (IF ONLINE FUNCTIONALTY IMPLEMENTED) Monthly cost for an ISP

## Legal

* Ensure all the user and employee data is stored safely and securely, using encryption, and preventing any employee from reading other peoples’ sensitive data.
* Any licenses need to be kept valid.
* Copyright on cards and their images must display the Konami copyright mark and text, commonly located in the bottom right section of the card’s border.

## Operational

* Each connected device requires an up-to-date version of Visual Studio.
* Updates to the system need to be performed to maintain security of the data: this needs to be done when the store is closed and when it’s appropriate to restart the server.
* Database files need to be stored on the central device; however, any backups should be stored on a drive or device separate to the one holding the main database.
* Automatic and manual backups of the database should be made.
* (IF SCHEDULE ALLOWS, may not be required due to low number of people involved in changing the information on database) An indexing solution to prevent deadlock
* (Alternative) Database file locking if in use by another user.

## Schedule

* Due to the current Schedule for production, Online functionality may be considered however this is subject to change.

# Data Flow of System

## Level 0 Data Flow Diagram

## Level 1 Data Flow Diagram

## Data Flow Dictionaries

### 1.1 Customer Buys Goods

|  |  |  |
| --- | --- | --- |
| Data Item | Description | Datatype |
| Buying\_order\_details | The details that are required to pinpoint the item the customer wishes to buy. | Items\_in\_purchase + Item + Quantity\_desired |
| Buyer\_payment\_details | Information about how the customer payed and general information about the transaction. | Single\_item\_quantity + Item + Items\_in\_purchase + Price + Total\_Price + Date + Time |
| Bought\_confirmation | A confirmation in the form of a receipt. | Single\_item\_quantity + Item\_Name + Price + Total\_Price + Date + Time |
| Check\_if\_in\_stock | Checks whether an item is within the database of the stock, which is done individually for each item within the order. | Item\_ID + Item\_Quantity |
| Check\_confirmation | A confirmation that returns from a query about checking if an item of given quantity is found in the stock. | Item + Item\_Quantity |
| Item | A group of properties for a given item, being a card or pack of cards | Item\_Name + Item\_ID + Item\_Wear + Item\_Quantity + Item\_Extras |
|  |  |  |
| Price | The cost of an individual item | FLOAT with precision up to Tenths column (X.XX). |
| Total\_Price | The combined cost of all items in a transaction. | FLOAT with precision up to Tenths column (X.XX). |
| Date | Today’s date. | DATE (DD/MM/YYYY). |
| Time | Current time. | TIME (HH:MM:SS). |
| Quantity\_desired | The quantity that customer is seeking to buy of a specified singular item. | INTEGER. |
| Single\_item\_quantity | The quantity of a singular item in the final transaction. | INTEGER. |
| Items\_in\_purchase | The number of different items within the transaction (one set and two different cards would make the number of different items in purchase equal to three).  This number should be changed if any of the desired items are either not found in the stock or are no longer wanted, but also if any extra items are added in a later stage. | INTEGER. |
| Item\_Quantity | The amount of an exact item that is present in the stock, | INTEGER. |
| Item\_Name | Name of a card or pack of cards. | STRING, 55 Characters in length. |
| Item\_ID | The identification code of a card or, if applicable to sets, a set of cards. (SET-LN999 is an example of how most cards are identified, HOWEVER NOT ALL CARDS HAVE THIS EXACT FORMAT OF CHARACTER AND INTEGER).  The longest possible card ID having 4 characters for set prefix, 2 for region abbreviation and 3 integer characters for set position number. | STRING, 9 Characters in length |
| Item\_Wear | A two character prefix assigned to a card after its quality has been graded. | STRING, 2 Characters |
| Item\_Extras | Extra information that will dictate the price of the item, primarily factors such as a special rarity of card, THIS FIELD CAN BE NULL. | Null-able, STRING 10 Characters in length. |

### 1.2 Customer Sells Goods

|  |  |  |
| --- | --- | --- |
| Data Item | Description | Datatype |
| Selling\_order\_details | The details about what the seller wishes to trade in for cash. | Item + Quantity\_of\_items\_offered |
| Seller\_payment\_details | Details of how the seller was paid, when and what items were gained for the stock. | Single\_item\_quantity + Item + Items\_in\_transaction + Price + Total\_Price + Date + Time |
| Customer\_credentials | Personally identifiable information to be held for an appropriate amount of time in case of the customer changing their mind, which may be required. | Customer\_Name + Customer\_phone + Customer\_email |
| Sold\_confirmation | Details in form of a receipt for the seller to hold | Single\_item\_quantity + Item\_Name + Price + Total\_Price + Date + Time |
| Update\_stock\_details | Details of each single item that are required to update the stock log: **this data is sent FOR EVERY ITEM IN THE FINAL TRANSACTION** | Item + Single\_item\_quantity |
| Item | A group of properties for a given item, being a card or pack of cards | Item\_Name + Item\_ID + Item\_Wear + Item\_Extras |
|  |  |  |
| Update\_confirmation | A Boolean value that determines whether there was success In updating the stock log with the information provided: if a feature will be implemented for adding in bulk, then this will flag up any incorrectly entered data and preserve the data prior to the upate. | BOOLEAN |
| Suggested\_price | This value can be generated by the system, however except for that, it is primarily verbal. | FLOAT with precision up to Tenths column (X.XX). |
| Seller\_negotiated\_confirmed | A confirmation that the seller is happy to sell what they suggested at the price suggested by the shop, however this value should be allowed to be changed and is not stored in a database, only In a transaction log. | BOOLEAN, Verbal and is done in-person during suggested pricing and potential haggling |
| Single\_item\_quantity | The quantity of a singular item in the final transaction. | INTEGER. |
| Items\_in\_transaction | The number of different items within the transaction (one set and two different cards would make the number of different items in purchase equal to three).  This number should be changed if any of the desired items are either not found in the stock or are no longer wanted, but also if any extra items are added in a later stage. | INTEGER. |
| Item\_Name | Name of a card or pack of cards. | STRING, 55 Characters in length. |
| Item\_ID | The identification code of a card or, if applicable to sets, a set of cards. (SET-LN999 is an example of how most cards are identified, HOWEVER NOT ALL CARDS HAVE THIS EXACT FORMAT OF CHARACTER AND INTEGER).  The longest possible card ID having 4 characters for set prefix, 2 for region abbreviation and 3 integer characters for set position number. | STRING, 9 Characters in length |
| Item\_Wear | A two character prefix assigned to a card after its quality has been graded. | STRING, 2 Characters |
| Item\_Extras | Extra information that will dictate the price of the item, primarily factors such as a special rarity of card, THIS FIELD CAN BE NULL. | Null-able, STRING 10 Characters in length. |

### 1.3 Re-Stocking through delivery

|  |  |  |
| --- | --- | --- |
| Data Item | Description | Datatype |
| Restock\_payment\_details | Details containing information about what has been delivered + details about the delivery itself. | Single\_item\_quantity + Item + Items\_in\_Transaction + Total\_Price + Date + Time + Distributor |
| Restock\_Update\_Details | Details containing the information that needs to be added into the stock log. | Item + Single\_Item\_Quantity |
| Restock\_confirmation | A confirmation in the form of a receipt, with each single item listed individually and the total cost for the whole transaction. | Single\_item\_quantity (for every item) + Item\_Name + Total\_price + Date + Time |
| Item | A group of properties for a given item, being a card or pack of cards | Item\_Name + Item\_ID + Item\_Wear + Item\_Extras |
|  |  |  |
| Single\_item\_quantity | A number representing the quantity of a single type of item within order | INTEGER |
| Items\_in\_transaciton | A number of unique items within the order. | INTEGER |
| Restock\_Update\_confirmation | A confirmation that items were successfully added to the stock and stock log. | BOOLEAN |
| Item\_Name | Name of a card or pack of cards. | STRING, 55 Characters in length. |
| Item\_ID | The identification code of a card or, if applicable to sets, a set of cards. (SET-LN999 is an example of how most cards are identified, HOWEVER NOT ALL CARDS HAVE THIS EXACT FORMAT OF CHARACTER AND INTEGER).  The longest possible card ID having 4 characters for set prefix, 2 for region abbreviation and 3 integer characters for set position number. | STRING, 9 Characters in length |
| Item\_Wear | A two-character prefix assigned to a card after its quality has been graded. | STRING, 2 Characters |
| Item\_Extras | Extra information that will dictate the price of the item, primarily factors such as a special rarity of card, THIS FIELD CAN BE NULL. | Null-able, STRING 10 Characters in length. |
| Total\_Price | The combined total of the order | FLOAT with precision up to Tenths column (X.XX). |
| Distributor | The name of the company that the restock was purchased from. | STRING, 30 Characters in length |
| Date | The date the transaction was completed. | DATE (DD/MM/YYYY). |
| Time | The time that the transaction was completed. | TIME (HH:MM:SS). |

# My Proposed Solution

My proposed system involves the use of a computer algorithm that is accessible from multiple connected devices, which will digitally store an up-to-date stock log, as well as back-up copies accessible to the system maintenance staff.

The algorithm will also be connected to a localised database on which the relevant information shall be both stored, deleted and manipulated within.

## Objectives for the creation of the proposed solution

The list of objectives I intend to review by the testing stage of the development of my solution are the following:

* Create an offline database which holds the stock log and its information within.
* Implement a method of authentication to allow users of different access privileges to use the database implemented.
* Allow for both automatic and manual back-up creation.
* Keep track of all changes made to the main log, including date, time, and user.
* Implement a solution that will prevent deadlock: timestamp logging.
* Create an autonomous process in which A document of all changes made by users are recorded
* Create sufficient error handling upon the launch of the application (like if the database file is missing or perhaps is corrupt.)
* Create a feature which automatically attempts to recover the most recent, non-corrupt database file in the event the primary file fails.
* Allow for manual debugging of the system for the manager (or system manager if one is hired) if needed with the use of a special mode which displays verbose information about the database and queries.
  + Allow for test queries to be used in debug mode, where the original database is preserved
  + Allow access to an automatically generated document where the changes made by each user are recorded, along with the time, date and access level
    - Allow for the document to be exported as a back-up or for analysis
  + Create a Feature which will go through the entire database and search for errors
    - A document should be created which provides information about the error, as well as what data-item caused the error. (This will be mainly used if someone manages to insert an invalid record into the structure, which bypassed prior checks.)
* Allow for single and bulk entries to be added, removed, and updated to and from the database.
* Prevent incorrect data from being entered with error handling for easy use.
* Check against a set of valid card IDs to:
  + ~~Allow for autofill of text.~~
  + Allow for a single card to be linked to a whole pack, vice versa.
  + Allow for validation checks to ensure data entered, such as ID of card is correct.
* A database which holds all the sellable cards and a separate one keeping track of the stock: that way when a user searches for a card that is not in stock, they can still get information on the card.
* Perhaps a small transaction history for every kind of card, dependent and/or independent of rarity of a card? (eg: to check how much a specific kind of card sold for: make this a toggle feature perhaps, so the user doesn’t have to wait for the log to appear?)
* If time allows a process in which the value of a card is calculated based on rarity and factors such as wear and manually set parameters, such as an extra value if the card is new (not if it’s very old, considering this is not a collectors shop but a shop dedicated to buying and selling playable condition cards.)
* If time allows, create a transaction log which will allow users to view all the bought and sold card orders, as well as the price of each individual item inside: this will require an additional layer for when cards are added or deleted from the stock, however a price tag should be made as an optional field.

**All of these are subject to change if seen unfit during the development process.**

Yu-Gi-Oh Card Stock system: Planned Solution Design

# System Design Overview

## New System Design Visualised

### Adding Stock

Whenever a user with permission wishes to add an item, regardless of in bulk or not, a form will appear in which they have to enter crucial information about the card, which will get checked before being submitted into the stock database.

A picture containing diagram

Description automatically generated

 If the user fails to add an item, an error message will appear but the adding stock form will stay up, meaning that they can quickly retry.

A picture containing diagram

Description automatically generated

### Removing Stock

Whenever an authorised user wishes to delete an item or multiple items in bulk (multiple copies of the same item), they will be prompted to enter the details about the card, which will be checked and if the card has been pin-pointed, a deletion confirmation window will appear, in which the user will need to agree to delete the item.

Preferably, all the items that are pending removal should be kept in a buffer, where multiple items can be grouped together into a single order before being permanently removed from the stock.

A picture containing graphical user interface

Description automatically generated

 If any one item removal query fails, an error message will appear: this will appear if either the search parameters are incorrect, too broad, or extreme.

Diagram

Description automatically generated with medium confidence

### Searching & Updating Stock

Whenever a user wishes to see whether an item exists in stock, they will enter any uniquely identifiable information which will bring up a profile of that card with the card’s information as well as how many are in stock.

Related items, such as cards in the same bundle may also be shown.

A screenshot of a computer

Description automatically generated with medium confidence

 If the search query fails or is too vague, a relevant error or information box may appear which will communicate to the user the error encountered and a potential fix.

Diagram

Description automatically generated

Updating the stock will works the same way a search does, however it allows the user to tweak the price or quantity of the item: mainly should be used to correct mis-input of quantity or price; if a change fails due to incorrect input, a relevant error message will appear and the change will not occur.

# New System Summarised - IPSO Charts

IPSO Stands for Input, Process, Storage, Output

## Inserting a new stock item(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Process | Storage | Output |
| - Card ID  - Card condition  - Quantity to add  - Price of item (optional) | - Connect to card database (READ ONLY)  - Connect to stock database  - Check if card details are valid  - Perform checks on quantity and price  - Handle Errors | - Store information about new card if not already present in stock  - If present of same ware, increase quantity and override price if one was provided  - Append changes onto activity log | - Confirmation to the user that the item has been added to the stock.  - Error messages relating to the user’s inputs or any flaw in the program. |

## Queuing for removal of item(s)

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Process | Storage | Output |
| - Card ID  - Card Condition (optional, different wares could appear) | - Connect to card database (READ ONLY)  - Connect to stock database  - Check if card details are valid  - Perform checks on quantity  - Handle Errors | - Hold onto changes in memory | - Visual confirmation to user  - Error message for if the process fails |

## Item(s) deletion confirmation

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Process | Storage | Output |
| - Items Pending deletion  - User deletion confirmation/ cancellation  - User queue alter request | - Connect to card database (READ ONLY)  - Connect to stock database  - Check if card details are valid  - Handle Errors | - Removal of records from the stock database  - Append changes onto activity log, with card ID and quantity to allow for manual recovery (or automated, if implemented). | -Visual confirmation to user  - Error message for if the process fails |

## Searching For stock item

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Process | Storage | Output |
| - Card ID (optional)  - Identifiable card details (name, level, etc.) | - Connect to card database (READ ONLY)  - Connect to stock database  - Check if card details are valid  - Handle Errors  - Extract relevant information to output for user | N/A | - Details about item and items relevant to the search query |

## Updating a stock item

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Process | Storage | Output |
| - Card ID  - request to update  - quantity/price to update | - Connect to card database (READ ONLY)  - Connect to stock database  - Check if details to modify are correct  - Handle Errors | - stock database  - append changes onto activity log | - Update confirmation  - Error message if update process fails. |

# Modular Design Hierarchy Chart

# Data Sources and Destinations

## Identifiyng Primary & Secondary Users

### User Level Access

This access level applies to the receptionist and any guest users that have been registered onto the system.

The users under this access level are secondary users, as they seek information from the system but do not alter the database or store on the database.

### Manager Level Access

This access level applies primarily to the stock manager role in the shop: users under this access level are **primary users**, as they alter the data stored in the databases and files

### Supervisor Level Access

This access level is identical to access level of a user: **secondary user.**

### Administrator Level Access

This is the highest possible user access level on the system, able to both alter and search through the system’s database(s), as well as having access to backup options: users of this access level are **primary users**.

## Sources of Data

|  |  |  |
| --- | --- | --- |
| Data | Source | Description |
| Stock Item Information | Manager OR Admin Access Level users | Data that includes information such as Set ID, Name of item, quantity of item and (if applicable), information about the quality of item |
| User Log in Credentials (ENTERED) | All users | Username and Password provided and compared to authorise a user into the system. |
| User Log in Credentials (STORED) | Encrypted File(s) | Username and Password credentials of all users registered onto the system. |
| Reference Item information | Locally Stored Database | This data is taken from a publicly available GitHub source (Bennett, 2017), which extracts information from the Yu-Gi-Oh Fandom Wiki (Fandom Coropration, 2005) |

## Destinations of Data

|  |  |  |
| --- | --- | --- |
| Data | Destination | Description |
| Stock Search Query Result | Search Result Form open for user. | This is an output that is the result of a search of the stock database through the Search Form: contains the name, quantity and quality of items that match the search. |
| Item Details | Item Form | Contains all relevant information about an item in the stock database. |
| Log-in credentials (Single) | Secure File(s). | Existing or New user’s credentials.  This information will be stored in an encrypted file. |
| Log-in credentials (All users) | Secure File(s). | All existing user log-in credentials.  Accessible only by the administrator, in the event of an employee forgetting their password. |
| Changes made to stock items (timestamped) | Latest Changes Log. | Any and all actions that involve the creation or modification of a record in the database are recorded here. |
| Back-up Files created via debug mode | Back-up folder location | Relevant files, such as back-up database files and back-up copies of changes log. |

# Database Structure

The database will consist of multiple tables, with a central table acting as the one which holds all of the stock quantities and branches off to the different tables with foreign keys.

The information on cards will be extracted from the most recently available version of document from source (Bennett, 2017): URL available from [References](#_References) section of document: for the purposes of this project, the data used will be limited but made expandable for future full implementation.

## Card Information neeed

The following is a list of the information about each card and (wherever appropriate) card set, which will be stored within the database:

* Internal Item ID (INTEGER, will be used to refer to a default template version of an item, such as card)
* Name (String with maximum capacity of 55 for name length: the name of a card set or card)
* Price (Float/Real datatype, tweakable)
* Quantity (Integer, maximum value of 9999 or equivalent: duplicate records are grouped and quantity is incremented)
* Notes (String, maximum value of 255: additional quality or general notes about item are held.)
* SetID (String, maximum length of 11: many cards have many different sets they originate from, so they will often contain multiple set IDs, so it can be used to differentiate between different records as well, however another property of the set IDs is that they share a prefix at the beginning, as well as a prefix for the language of set after the dash, eg: XXXX-EN001 **THE MINIMUM LENGTH FOR A SETID IS 2**)
* Rarity (String, maximum length of ~3: determines how rare a card is, with different card sets being able to provide different rarities to the same card.)
* ImageLocation (String of max length 50: stores the location of the image on the local system)

## User Information needed

These are the user credentials which will allow the system to operate:

* Username (string, maximum length of 30)
* UserID (Integer, used to pin point a user)
* Password (String, maximum length of 30 and minimum length of 10)
* AccessLvl (String OR integer of minimum length to signify the level of privileges that the user has on system.)
* First Name (String, maximum length of 20)
* Surname (String, maximum length of 30)
* Email (String, maximum length of 50)

All of the user credentials will be encrypted before being stored.

Username, Password and AccessLvl are the most important pieces of information for the function of the authentication method of the program, meanwhile the rest of the information can be utilised by an administrator level user who will be able to more easily pin point a user who is unable to access their account.

## Customer information needed

Some customers may have a pre-existing account on the system on which their purchases are tracked.

In order to store a customer account instance the following information is required:

* First Name (String, maximum length of 20)
* Surname (String, maximum length of 30)
* Customer ID (automatically assigned)
* Email (String, maximum length of 50)
* Telephone (String, maximum length of 15)

In this current state of the program, the users do not possess access to the system, however this can be expanded on in future changes to system.

## Changes made in database information

The change log will hold the following:

* Timestamp (TIME/STRING)
* StockID of item record (SAME AS STOCKID OF STOCK)
* Description of change (STRING of length max 255)
* UserID: User who changed it (SAME AS USERID FROM FILE OF USERS)
* ChangeID (INTEGER)

## Customer Order made

An order needs to store the following:

* OrderID
* All individual items ordered (using a separate entity OrderLine)
  + OrderLine holds the StockID of the item, which can point to the item in question as well as:
    - The Price of a single of that item type
  + The quantity of the item of a type purchased
* CustomerID associated to order
* Date & Time of order
* Combined cost of the order, gathered from the information of all OrderLine records with the orderID matching this order.

## Tables for database

My database will consist of the following data tables

### Stock

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| StockID | ItemID (FK) | StockQuantity | StockQuality | StockPrice | StockNotes |
| 0001 | 9999 | 2 | EX | 999.99 | Signed |
| 0002 | 9999 | 9999 | GD | 2.50 |  |
| 0003 | 9998 | 30 | GD | 0.55 |  |

* StockID: 4-digit integer
* ItemID: 4-digit integer
* ItemName: String with maximum capacity of 55 for name length
* StockQuantity: 4-digit integer
* StockQuality: 2-Character String
* StockPrice: 2-decimal place Float, limit of 999.99 or equivalent
* StockNotes: String, maximum value of 255

### Item

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ItemID | ItemName | ItemSetPrefix | Rarity | ImageFilepath |
| 1234 | Test Set | DDDD | N | Images\test.png |
| … | … | … | … | … |
| 9998 | Test Card 1 | SSSS | SR |  |
| 9999 | Test Card 1 | ABCD | UR |  |

* ItemID: 4-digit integer
* ItemName: String with maximum capacity of 55 for name length
* ItemSetPrefix: 4-Character String
* Rarity: 2-Character String
* ImageFilepath: String, max length of 50

### Users

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| UserID | Username | Password | AccessLvl | FirstName | Surname | Email |
| 0001 |  |  |  |  |  |  |

All of this data will be encrypted, except UserID and access level.

Info about all fields found in User Information needed above

### Customers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CustomerID | Firstname | Surname | Telephone | Email |
|  |  |  |  |  |

The details surrounding different customer attributes in above section detailing the needed information.

### Changes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ChangeID | Timestamp | StockID | UserID | ChangeDescription |
| 1 | [HH:MM:SS.SS - DD/MM/YYYY] | 1 | 1 | REMOVED 10 OF ITEM |

Description of changes is determined by action taken by user

### Order

|  |  |  |  |
| --- | --- | --- | --- |
| OrderID | CustomerID | Timestamp | TotalCost |
|  |  |  |  |

### Order Line

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OrderLineID | OrderID | StockID | Quantity | Price |
|  |  |  |  |  |

# Program Data Structures

In order to store information in the system, I will utilise a .mdb database which will be the main store for the information, as it allows for flexible data manipulation and storage.

## Assembling SQL Queries

When it comes the creation of the numerous SQL queries, I shall use Object-Oriented programming, which will allow me to more easily make the queries customisable, but also allows for admin users ease of access to the database using SQL through the program (they don’t need to know the internal workings of the class but simply know the LINQ of the class provided by me), however, also allowing for quicker implementation of code inheriting or utilising the pre-written code.

## Searching and Sorting

In order to sift through the any query results that requires a search algorithm and cannot just be narrowed down by more detail in an SQL query, a binary search or equivalent may be utilised, as it will speed up runtime of a search.

For sorting query results, which cannot be sorted by means of an SQL statement, merge sort will be used: this is because while a query result may on occasion be small, it is expected to potentially hold a large number of items if the search parameters cover a broad range of items: This will also ensure that the impact on runtime of the algorithm will be reduced, in the event of shop’s storage expanding and more stock being added or if more items are made avaliable to be added to the stock.

## Data to be stored

All of the data mentioned in the Database Structure will be stored digitally within .mdb database, however changes made, and order transactions will also be stored in digital text files.

# Data Normalisation

## Un-Normalised

|  |
| --- |
| **StockID**  **StockQuantity**  **StockQuality**  **StockPrice**  **StockNotes**  ItemID  ItemName  ItemSetPrefix  ImageFilePath  CustomerID  UserID  OrderID  OrderLineID  ChangeID  Username  Password  FirstName  Surname  Email  Telephone  ChangeDescription  Timestamp  TotalCost  Quantity  Price |

## 1st Normal Form

|  |
| --- |
| **StockID**  **StockQuantity**  **StockQuality**  **StockPrice**  **StockNotes** |
| **StockID**  ItemID  CustomerID  UserID  OrderID  OrderLineID  ChangeID  ItemName  ItemSetPrefix  ImageFilePath  Username  Password  FirstName  Surname  Email  Telephone  ChangeDescription  ChangeTimestamp  OrderTimestamp  TotalCost  Quantity  Price |

## 2nd Normal Form

|  |
| --- |
| **StockID**  **StockQuantity**  **StockQuality**  **StockPrice**  **StockNotes** |
| **StockID**  CustomerID  UserID  OrderID  ItemName  ItemSetPrefix  ImageFilePath  Username  Password  FirstName  Surname  Email  Telephone  ChangeDescription  ChangeTimestamp  OrderTimestamp  TotalCost  Quantity  Price |
| **StockID**  OrderLineID  ChangeID  ItemID |

## 3rd Normal Form

|  |
| --- |
| **StockID *ItemID\****  **StockQuantity**  **StockQuality**  **StockPrice**  **StockNotes** |
| ItemID  ItemName  ItemSetPrefix  ImageFilePath |
| UserID  Username  Password  FirstName  Surname  Email |
| CustomerID  Username  Password  FirstName  Surname  Email  Telephone |
| ChangeID *StockID\* UserID\**  ChangeDescription  ChangeTimestamp |
| OrderID *CustomerID\**  TotalCost  OrderTimestamp |
| OrderLineID *OrderID\* StockID\**  Quantity  Price |

# Entity Relationship Diagrams

|  |
| --- |
|  |

# Data Validation

## Prices

Any price set within the system should be in the format 9999.99, which corresponds to £9999.99

## Date & Time

Dates must be in the form of DD/MM/YYYY.

Time must be in the form of HH:MM:SS (Hours:Minutes:Seconds)

To combine both date and time into a timestamp, use the form of [DATE TIME], encompassed by square brackets.  
Example: [24/01/2012 18:49:18]

## Primary Keys

Primary keys are to be in form of integers and must allow for at least 9999 unique instances to be stored.

## Set Prefix

The item entity requires a Set Prefix attribute to be in string form of “XXXX”

## Rarity

The Rarity attribute in the Item entity must be in string format of “XX” and must be any of the following:  
“FN”, ”NM”, ”EX”, ”GD” and ”LP” (Factory New, Near-Mint, Excellent, Good, Light-Played: Factory New is unique as it only applies to card sets.)

## Image Location

The image location attribute for the item entity must be in the form of “Images\IMAGEFILENAME.jpg”

This will point to an image file located in a subfolder of the Files folder that is located within the version of the program.

An image location file doesn’t need to be filled in by default.

## Telephone Numbers

Telephone numbers should be stored in string format and must only contain numerical characters from 0 to 9, however they may also contain any regional code if appropriate at the beginning of the number, as well as any brackets used.

Example of a local number format: “07999 999999”  
Example of regional number format: “+999 999 999 9999”

Maximum number of possible character should be approximately 30: numbers are expected to be written sensibly, with a limited number of space characters and any extra symbols ( + , - , (, ) ARE ONLY ALLOWED)

## User Access Level

This information should be In string format of “XXXX”, with only allowed inputs being “ADMN”, “USER” and “MNGR”

# Query Design

For each of the following queries, here is a list of additional fields that will likely be utilised:

* WHERE
* SORT BY

## SELECT

Will primarily be utilised for search functionality of the program.

Example:  
SELECT \* FROM tblStock  
WHERE StockID = 2  
SORT BY StockPrice

## INSERT

Primarily will be used for inserting new stock records.

Example:  
INSERT INTO tblStock (StockID, ItemID, StockQuantity, StockQuality, StockPrice, StockNotes)  
VALUES (‘122’, ‘14’, ‘3’, ‘GD’, ‘3.99’, ‘Bent, Water damaged’)

## UPDATE

Main use in changing user credentials as well as updating price of an item in stock.

Example:  
UPDATE tblUsers  
SET email = ‘bobby14@gmail.com’, AccessLvl = ‘MNGR’  
WHERE UserID = 5

## DELETE

Used in the remove function of the program when stock quantity was equal to a value and now has fully depleted.

Example:  
DELETE FROM tblStock  
WHERE ItemID = 13 AND StockID = 56

# User Interface Design

Some of the forms look different from others, due to a different Visio version being used at the time of their making, primarily when worked on outside of college.

## Sign In Form

### Design

Diagram

Description automatically generated

### Functionality Overview

This is the first form that can be seen by any user running the program.

This form acts as the authentication method for the system, with the users being required to enter their assigned username and password details into the text fields visible on the form.

If at any point of the authentication process an exception in the program is raised, the process is cancelled and a relevant error message box is displayed, detailing the error and any potential advice on avoiding or resolving the error.

The sign-in procedure will send the credentials through an encryption algorithm and compare the output to any stored user credentials (this process should be done securely, preferably by using an external application or module not connected to the main program directly.)

Upon a successful authentication of a user, their access level will determine which type of form they will be allowed to view (user, manager, or administrator forms)

### Pseudocode

|  |
| --- |
| SUBROUTINE Form\_Load:  Make connection with the encrypted user credentials document  Make connection with the encryption module algorithm  END SUBROUTINE  SUBROUTINE Error\_message(message, tip)  Create new instance of error message box with only "OK" button and text matching message parameter & any tip parameter mentioned  END SUBROUTINE  SUBROUTINE BTN\_AUTHENTICATION.Clicked:  Send Username and Password through encryption Function(s) from external module  Store Encrypted username and password results in temporary variables  Query/search external credentials document for match, as well as user access level  IF TXT\_USERNAME\_FIELD AND TXT\_PASSWORD\_FIELD = Empty  CALL Error\_message(text detailing how one or two of text fields are empty & empty tip parameter)  ELSE IF Match found:  Close this form and send user information into a new instance of user form that matches access level  ELSE  CALL Error\_message(text detailing how credentials entered do not match & empty tip parameter)  ENDIF  END SUBROUTINE |

## User / Supervisor’s Form

### Design

### Functionality Overview

Any user that has the lowest access level will be directed to this form upon successful log-in, where they are able to search through the active stock, change their user credentials or log off.

Logging off can be done in two ways: hitting the “log off” button or closing down the form; upon clicking the log off button, the user form will close and the sign-in form will re-appear, meanwhile closing the form will end the execution of the whole program.

The supervisor currently is re-directed to this type of form, however if I do end-up making a separate access level category for them, their influence over system functions can be changed for future use, if needed.

### Pseudocode

|  |
| --- |
| SUBROUTINE Form\_Load(user as referrable object OR first and last name of associated user)  Set name of form to "Welcome, " followed by the full or first name of the user, obtained from parameter field(s)  END SUBROUTINE  SUBROUTINE Button\_Search CLICKED ()  Call new instance of search form  END SUBROUTINE  SUBROUTINE Button\_Account\_Details CLICKED ()  Call new instance of account details form, passing over the user ID or equivalent to be used in a query of database.  END SUBROUTINE  SUBROUTINE Button\_Log\_off CLICKED ()  Create log off confirmation box, storing output of message box in temporary variable  IF message box value returned = TRUE  CLOSE all active form instances still active.  CLOSE form instance  RETURN to sign in form  END IF  END SUBROUTINE  SUBROUTINE Close X Button CLICKED ()  Create program termination confirmation box, storing output of message box in temporary variable  IF message box value returned = TRUE  TERMINATE program  END IF  END SUBROUTINE |

## Manager Form

### Design

### Functionality Overview

This form can only be accessed by users with the “manager” access level, having additional features over the lower “user” access level.

This form grants access to the following features:

* Searching stock item
* Adding stock item(s)
* Removing stock item(s)
* Accessing History logs
* Altering log-in credentials

Logging off functions the same way as in a user form.

### Pseudocode

|  |
| --- |
| SUBROUTINE Form\_Load(user as referrable object OR first and last name of associated user)  Set name of form to "Welcome, " followed by the full or first name of the user, obtained from parameter field(s)  END SUBROUTINE  SUBROUTINE Button\_Search CLICKED ()  Call new instance of search form  END SUBROUTINE  SUBROUTINE Button\_Account\_Details CLICKED ()  Call new instance of account details form, passing over the user ID or equivalent to be used in a query of database.  END SUBROUTINE  SUBROUTINE Button\_History CLICKED ()  Call new instance of Activity log form  END SUBROUTINE  SUBROUTINE Button\_Add CLICKED ()  Call new instance of Add form  END SUBROUTINE  SUBROUTINE Button\_Remove CLICKED ()  Call new instance of remove queue form  END SUBROUTINE  SUBROUTINE Button\_Log\_off CLICKED ()  Create log off confirmation box, storing output of message box in temporary variable  IF message box value returned = TRUE  CLOSE all active form instances still active.  CLOSE form instance  RETURN to sign in form  END IF  END SUBROUTINE  SUBROUTINE Close X Button CLICKED ()  Create program termination confirmation box, storing output of message box in temporary variable  IF message box value returned = TRUE  TERMINATE program  END IF  END SUBROUTINE |

## Search Form

### Design

Diagram

Description automatically generated

### Functionality Overview

This form is primarily used to search through the current stock of items within the database, mainly being used for when a user is doing a normal search or is looking to remove an item or items from the stock.

The name of the item field can use used to narrow down the search, however the important field is that of Card ID: different IDs associated with an item, such as the Set ID or potentially even the hidden card ID not visible on cards or box sets should be checked.

When a search is initiated, the search form will query the current stock database for any items with matching information, relaying the information to be displayed in the search results form.

Card condition options are as follows (in ascending order of quality):

* All (no filter applied to search)
* Light-Played
* Good
* Excellent
* Near-Mint
* Factory-New (only sets of cards and other miscellaneous items considered, no card in such condition)

If an error occurs prior to or during the search, an error message box appears with information about the error and potential tips on how to avoid or resolve the issue: the user is then shown the search form again, with all of the information in text fields preserved as they were prior to error occurring.

### Pseudocode

|  |
| --- |
| SUBROUTINE Button\_Help CLICKED ()  DISPLAY message information box with the information stated in the diagram or equivalent.  END SUBROUTINE  SUBROUTINE Button\_Run\_Search CLICKED ()  QUERY database requesting the output of all items that match the attributes given in the fields AND are present in the stock  IF Error Occured & caught  DISPLAY message error box detailing additional information about the error if avaliable, otherwise default to standard message.  END IF  STORE the results of the database query, so that every individual item can be accessed  CREATE NEW instance of Search Result Form, passing the object/variable the query results are stored in OR allowing access to that information.  END SUBROUTINE |

## Search Result Form

### Design

### Functionality Overview

This form displays the results of a search query performed by the search form, where extra information about each item matching the search is given, primarily the name of item, condition, quantity, and a small preview image of the item.

Clicking on an item will highlight the rectangular cell of the scrollable menu holding every item, while double-clicking on an item will relay that item’s information into the Item Form and open it up on-top of the search result form.

The “Confirm Selection” button will return the specific item selected back into the search form, which will return it into the remove queue form.

### Pseudocode

|  |
| --- |
| SUBROUTINE Form\_Load (Query\_Results, Removing As Boolean)  IF NOT Removing  Hide Button\_Cancel\_Item\_Selection & Button\_Confirm\_Item\_Selection  END IF  FOR EVERY ELEMENT IN Query\_Results  CREATE a new item in List Box, which displays the image, name of the item, quality of item & quantity of item  END FOR    Change the label\_results\_found text equal to the number of items present in query results  END SUBROUTINE  SUBROUTINE Button\_Cancel\_Item\_Selection CLICKED OR Close\_X\_Button CLICKED  CLOSE this form instance  END SUBROUTINE  FUNCTION Button\_Confirm\_Item\_Selection CLICKED  IF Selected Item has quantity of more than 1  USERINPUT "Enter how many of selected item you want to select: "  Store input into local variable  END IF  RETURN The index of the item selected OR StockID & the quantity to remove  END FUNCTION  #ItemX refers to any item within the list box  SUBROUTINE ItemX\_Field CLICKED  Store the item selected and highlight in light blue: change any text colours automatically to white colour font  END SUBROUTINE  SUBROUTINE ItemX\_Field DOUBLE CLICKED  Create new instance of item form, passing over the item information as a parameter  END SUBROUTINE |

## Item Form

### Design

### Functionality Overview

When a specific record is double clicked in the search result form, this form will open up: it details additional information about the item, as well as information about the item which is present within the most recent change log.

The “Edit Price” button allows users of all access levels to tweak the current item’s price stored in the database.

### Pseudocode

|  |
| --- |
| SUBROUTINE Form\_Load(ItemID)  QUERY database and fetch the information about the item in question  EXTRACT information form the activity log, fetching all of the instances in time order that are related to the exact item in question  Change the text box under the "Item History:" label to contain the information extracted from the activity log, seperated with spacing or appropriate seperation characters  END SUBROUTINE  SUBROUTINE Button\_price\_update CLICKED  Message box USERINPUT "Set price of item: "  IF price is not erronious (has no invalid characters and isn't too large)  UPDATE the item in the main database & log chage into activity log  Display confirmation to the change via either message box or by updating the item's history text box  Display the new price in the price label  END IF  END SUBROUTINE |

## Add Form

### Design

Diagram

Description automatically generated

### Functionality Overview

The add form appears with all of the fields blank and the item quality set to Near-Mint quality

An optional price may be

The note “quality is ignored when the item is a card set” refers to how any and all card sets will automatically be assigned the Factory-New quality.

### Pseudocode

|  |
| --- |
| SUBROUTINE FORM\_LOAD  LOAD any needed information in case it is needed for logging, such as the id of user  ENDSUBROUTINE  SUBROUTINE ADD\_BUTTON\_CLICKED  VALIDATE Data entered into text boxes  IF VALID  ADD record into Database  ELSE  DISPLAY Message box with error info  END IF  END SUBROUTINE  SUBROUTINE FIND\_BUTTON\_CLICKED  SEEK Database for Item ID and FILL textbox with text returned from attribute associated to the item's name in Item table within Database  END SUBROUTINE |

## Remove Queue Form

### Design

### Functionality Overview

This form is entered the first time that a user wishes to remove item(s) from the stock database.

A message box will appear upon the loading of this form, asking whether this removal is tie to a customer’s account: if selected that it is, a message box requests the input of the user’s First Name, Surname, and Email OR Telephone number: if customer with details isn’t found then the user of system should be able to re-enter details, however if a matching user is found, then the window closes & a label is shown with text “Associated to: ” followed by first & last name.

The “Add item to queue” button will prompt the search form to appear, with the result of the search being returned and added to the list box on the right of the form, additionally, the user, during this process, should be prompted as to the quantity of the item that is selected to be added to the removal queue.

Pressing the “Delete” button will make a confirmation box appear, where the user needs to agree that they wish to delete the selected items from the

Pressing the “Remove Selected” button will remove an item which was selected with the cursor out of the list

The process can be cancelled in one of two ways: pressing the Close button for the form in the top right or pressing the “Cancel” button: both buttons carry out the same purpose and their results should be identical: the form is closed and process terminated.

If any uncaught error (by the search and search result form) is discovered, a relevant error message box should appear with the details about the error, and potential tips on how to fix the error.

### Pseudocode

|  |
| --- |
| SUBROUTINE FORM\_LOAD  FETCH any needed information about the user or customers associated with the removal of items  END SUBROUTINE  SUBROUTINE BTN\_ENQUEUE\_CLICKED  OPEN New appropate form (such as search form) which allows the user to select an item to add tothe removal queue  ADD selected item into the Removal queue visible in a list box shown  END SUBROUTINE  SUBROUTINE BTN\_DEQUEUE\_SELECTED\_CLICKED  REMOVE Selected item from the item queue  END SUBROUTINE  SUBROUTINE BTN\_DELETE\_CONFIRM\_CLICKED  DISPLAY Modal text box asking for confirmation  IF User is ok with deletion  DELETE/UPDATE the records, depending on the quantity of items removed  END IF  END SUBROUTINE  SUBROUTINE BTN\_CANCEL\_CLICKED  CLOSE This form  END SUBROUTINE |

## Activity Log Viewer Form

### Design

### Functionality Overview

This form, only accessible the administrator level access, allows the user to Load a Log to view, Create a new, fresh log, back up the currently used log file, change which log is actively being written into, as well as view the filepath of where the log which is being written into is stored.

All of the buttons, except the “Active Log Filepath” will open up a file explorer window, with the file type restriction set to “.txt” file type.

The contents of the activity logs is displayed in in the large, read-only, multi-line textbox.

The name of the current log which is being used by the program to track changes is displayed by a label on the bottom left of the window.

### Pseudocode

|  |
| --- |
| SUBROUTINE BTN\_LOAD\_LOG\_CLICKED  OPEN File explorer allowing user to select a file to open  LOAD txt file information into the list box on the right  END SUBROUTINE  SUBROUTINE BTN\_CREATE\_NEW\_LOG\_CLICKED  SAVE information from selection box as a new file, with the help of a file explorer for save name and location  END SUBROUTINE  SUBROUTINE BTN\_BACKUP\_ACTIVE\_LOG\_CLICKED  SAVE backup file version of the currently active activity log into a present location  END SUBROUTINE  SUBROUTINE BTN\_SET\_ACTIVE\_LOG\_CLICKED  OPEN File Explorer to allow for file selection  SET selected file to be the new active changes log  END SUBROUTINE  SUBROUTINE BTN\_ACTIVE\_LOG\_FILEPATH\_CLICKED  GENERATE new message box displaying the filepath to the file that acts as the currently active activity log  END SUBROUTINE |

## Admin Form

### Design

### Functionality Overview

#### Main Form

This form can only be accessed by users with the “admin” access level, having additional features over the lower “manager” access level.

This form grants access to the following features:

* Searching stock item
* Adding stock item(s)
* Removing stock item(s)
* Accessing History logs
* Altering log-in credentials
* Access user log-in credentials on system & alter them
* Tweak the back-up settings of the system
* Check the integrity of the algorithm and database

Logging off functions the same way as in a user & manager form.

#### Backup Options Sub-Form

The backup options sub form should allow the administrator on the system to set the options for the delay of auto backup, allow manual saving, and loading of database files.

### Pseudocode

#### Main Form

|  |
| --- |
| SUBROUTINE Form\_Load(user as referrable object OR first and last name of associated user)  Set name of form to "Welcome, " followed by the full or first name of the user, obtained from parameter field(s)  END SUBROUTINE  SUBROUTINE Button\_Search CLICKED ()  Call new instance of search form  END SUBROUTINE  SUBROUTINE Button\_Account\_Details CLICKED ()  Call new instance of account details form, passing over the user ID or equivalent to be used in a query of database.  END SUBROUTINE  SUBROUTINE Button\_History CLICKED ()  Call new instance of Activity log form  END SUBROUTINE  SUBROUTINE Button\_Add CLICKED ()  Call new instance of Add form  END SUBROUTINE  SUBROUTINE Button\_Remove CLICKED ()  Call new instance of remove queue form  END SUBROUTINE  SUBROUTINE Button\_Backup\_options CLICKED  Call new instance of Backup options form  END SUBROUTINE  SUBROUTINE Button\_User\_Accounts CLICKED  Call new instance of User Acounts form  END SUBROUTINE  SUBROUTINE Button\_Run\_Test CLICKED  Perform a series of queries to determine the stability of the system and potential errors  Log errors encountered into a file titled by today's date, time and string " - Integrity Test Log" or something similar  Decide on a good location for the logs: if a folder doesn't exist for it titled relevant then automatically make one.  Create new instance of message box telling the user that the check has finished.  END SUBROUTINE  SUBROUTINE Button\_Log\_off CLICKED ()  Create log off confirmation box, storing output of message box in temporary variable  IF message box value returned = TRUE  CLOSE all active form instances still active.  CLOSE form instance  RETURN to sign in form  END IF  END SUBROUTINE  SUBROUTINE Close X Button CLICKED ()  Create program termination confirmation box, storing output of message box in temporary variable  IF message box value returned = TRUE  TERMINATE program  END IF  END SUBROUTINE |

#### Backup Options Sub-Form

|  |
| --- |
| SUBROUTINE BTN\_AUTOBACKUP\_OPTIONS\_CLICKED  OPEN Modal form allowing for tweaking of the auto backup settings, such as delay between backups and if it is enabled  END SUBROUTINE  SUBROUTINE BTN\_SAVE\_BACKUP\_CLICKED  CREATE New copy of the database file, seperate from the original file  END SUBROUTINE  SUBROUTINE BTN\_LOAD\_BACKUP\_CLICKED  LOAD a backup of the database file and use it instead of the current file  END SUBROUTINE |

## User Credential Management Form

### Form Design

|  |
| --- |
|  |

### Functionality Overview

This Form will allow Administrators to change the credentials of any individuals on the system of their choosing.

The form shows each user stored locally in a list of boxes, where their full name, username and access level are shown.

Upon clicking a field with a user in it, pressing the “confirm” button will open their user profile, where all of their credentials can be edited.

### Pseudocode

|  |
| --- |
| SUBROUTINE FORM\_LOAD  FETCH all of the user information from database  FILL all of the fields with appropriate user information  END SUBROUTINE  SUBROUTINE BUTTON\_CONFIRM\_CLICKED  OPEN New instance of Individual user credentials form  END SUBROUTINE  SUBROUTINE BUTTON\_CANCEL\_CLICKED  CLOSE this form  END SUBROUTINE |

## Individual User Credentials Form

### Form Design

|  |
| --- |
|  |

### Functionality Overview

This function needs to:

* Allow the current user to alter their user details (except UserID and AccessLevel)
* Commit the changes to the database once user confirms their selections

### Pseudocode

|  |
| --- |
| SUBROUTINE Form\_Load  GET information from Form\_User, which called it  Fill in the pre-existing information into the textboxes  END SUBROUTINE  SUBROUTINE BTN\_UPDATE\_DETAILS CLICKED  IF Password field matches the Confirm Password Field AND details are of suitable length:  UPDATE details locally  UPDATE database with new user details  END IF  END SUBROUTINE |

# Test Plan

## Sign In

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Description | Type of Data | Expected Result |
| 1 | Correct User details are entered to log into the database. | Normal | User should be let past the log in form. |
| 2 | Correct username is entered but not password | Erroneous | User isn’t allowed to log in |
| 3 | Correct password is entered but not username | Erroneous | User isn’t allowed to log in |
| 4 | Form closes appropriately when dictated to. | Normal | Form closes and program stops execution |

## User / Supervisor / Manager / Admin Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | | Description | Types of Data | Expected Result |
| 5 | Correct User details are entered | | Normal | User is taken to the correct form type, depending on their access level. |

## Search & Search Results Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | | Description | Types of Data | Expected Result |
| 6 | Item Name and card prefix are entered, which exist in the database | | Normal | Form Search Result will display any instances of records matching entered information |
| 7 | Correct item name but incorrect item prefix | | Erroneous | Display Error message stating no records were found |
| 8 | Correct item prefix entered but incorrect item name. | | Erroneous | Display Error message stating no records were found |
| 9 | Form is closed before any search is made | | Normal | Form closes without performing any search. |
| 10 | Search Results form is closed | | Normal | Form closes without performing any extra operations |
| 11 | Search Result form removal is cancelled | | Normal | Form closes without performing any extra operations |
| 12 | Remove button clicked for a highlighted record | | Normal | Display sub form requesting for deletion quantity. |
| 13 | User selects appropriate quantity to remove of a record | | Normal | Information is returned to remove queue form. |
| 14 | User cancels remove quantity selection | | Normal | Sub Form closes without performing any extra operations and search results form stays open for the user to do something else on it. |

## Item Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | | Description | Types of Data | Expected Result |
| 15 | Search result item is loaded into this form | | Normal | All information is correctly loaded into the form. |
| 16 | No changes log info found about item | | Normal | No information or an indicator of no information is displayed in the list box. |

## Add Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | | Description | Types of Data | Expected Result |
| 17 | User enters correct information about a new item they wish to add. | | Typical | A visual confirmation is shown to the user that the item has been added to the stock. |
| 18 | User enters an invalid item ID (anything that isn’t letters or numbers) | | Erroneous | appropriate error message is shown. |
| 19 | User attempts to leave necessary fields blank | | Erroneous | User is prevented from making a new stock record. |
| 20 | User attempts to enter very long text into fields that is in valid format | | Extreme | Appropriate error message is shown and/or user is prevented from making stock record |
| 21 | User attempts to find item name for an item that doesn’t exist | | Erroneous | Error handling prevents any exceptions from appearing, an appropriate error message box may be shown. |
| 22 | User attempts to add a stock record that exists | | Typical | As long as the notes field is the same, the price is updated and quantity is added to the existing quantity amount, otherwise a new record is created. |
| 23 | User exits form without adding any items | | Typical | Form closes without executing any add functions. |

## Remove Queue Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | | Description | Types of Data | Expected Result |
| 24 | User successfully selects an item to add to remove queue | | Typical | Information about the item added is displayed |
| 25 | User cancels the addition of a new item, after they have pressed the “Add item” button | | Erroneous | Adding process is cancelled. |
| 26 | User attempts to delete an item that exists in the queue. | | Typical | Queue is updated and information about the item is removed. |
| 27 | User attempts to delete an item from queue when it is empty | | Erroneous | Error is handles and a message box prompt may be shown. |
| 28 | User confirms deletion with items In queue | | Typical | Records are updated and/or deleted where appropriate, change is logged and the user receives a confirmation message telling them the process was successful. |
| 29 | User attempts to confirm deletion when the queue of items is empty | | Erroneous | No deletion or updating is carried out: an appropriate message box prompt may be shown. |
| 30 | User closes the form using the cancel button or close button | | Typical | Form closes and any process related to the form and deletion/updating of records is cancelled. |

## Activity Log Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | | Description | Types of Data | Expected Result |
| 31 | User attempts to open a correct log file | | Typical | Information from the log file is dumped into the list box |
| 32 | User attempts to save a log file correctly | | Typical | File is successfully filled with the information stored in the list box |
| 33 | User attempts to open an incorrect file type | | Erroneous | User is prevented from opening file by the file explorer’s filter settings. |
| 34 | User uses the backup active log file function | | Typical | File is backed up in a pre-set directory. |
| 35 | User uses the Show active log filepath function | | Typical | The directory leading to the currently active log file is shown. |

## User Credential Management Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | | Description | Types of Data | Expected Result |
| 36 | User selects and presses confirm button | | Typical | New form instance where the user can edit that account information is opened. |
| 37 | User closes the form or presses the cancel button | | Typical | Form is closed and any related functions are cancelled: no changes are made to any of the user accounts. |

## Individual User Credentials Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | | Description | Types of Data | Expected Result |
| 38 | User enters valid information that is the same or different to the previous information of their account. | | Typical | User details are updated. |
| 39 | User attempts to leave any or all fields blank | | Erroneous | User is prevented from saving the changes and a message box prompt is shown. |
| 40 | User attempts to enter a password exactly 10 or 30 characters in length, with the rest information being correct | | Extreme | User details are updated. |
| 41 | User enters information that is too short (username of two or less characters or longer than 30, as well as passwords with length of less than 10 or more than 30 & any other relevant fields.) | | Erroneous | User is prevented from saving their erroneous details and a message box prompt is shown. |
| 42 | User enters correct details, but the password and confirm password fields don’t match. | | Erroneous | User is prevented from saving their erroneous details and a message box prompt is shown. |

# References

Bennett, K., 2017. *GitHub - KianBennett/Yugioh-Database-Downloader: Downloads ....* [Online]   
Available at: https://github.com/KianBennett/Yugioh-Database-Downloader  
[Accessed 1 Feburary 2022].

Cardmarket, n.d. *Card Condition.* [Online]   
Available at: https://www.cardmarket.com/en/YuGiOh/Help/CardCondition  
[Accessed 2 October 2021].

Fandom Coropration, 2005. *yugioh.fandom.com.* [Online]   
Available at: https://yugioh.fandom.com/wiki/Yu-Gi-Oh!\_Wiki  
[Accessed 1 Feburary 2022].

# DEVELOPMENT

## Form Designs & Code Snippets

### Form\_Login

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Essentially checks if username and passwords are valid, if they are a new form instance of Form\_User is opened

#### Mod\_authenticate

Text

Description automatically generated

Checks if credentials provided match credentials in DB, starting with username and then password.

### Form\_User

Graphical user interface, application, PowerPoint

Description automatically generated

#### Form\_Load

Text

Description automatically generated

#### ReloadUserInformation

Graphical user interface, text, application

Description automatically generated

In the event that the user or an administrator has altered the credentials of the currently logged in account, new information is gathered from the database.

#### Buttons on the form

The buttons on the form all pass relevant information to each of the different forms made to carry out the different functions in the database: they all create a temporary variable associated with a new instance of relevant formA screenshot of a computer

Description automatically generated with medium confidence

### Form\_Edit\_Profile

Graphical user interface

Description automatically generated

#### Form Constructor subroutine

Text

Description automatically generated

Upon the form’s creation, the details of the current account the user is editing, alongside the mode of editing are set and form properties are filled with information depending on whether the form is being used to alter details of a user or customer on system or used to make either one.

#### Functions used to update and add a User

Text

Description automatically generated

The Update\_UserDetails\_User method is used in order to update the array called “UserDetails” with the information received from the different textboxes on the form, meanwhile the Update\_User\_Details method is used to change the information of the selected user I the dataset and update the local database file with the new information: this adding technique assumes that no user or customer record is able to be removed form the database, which would cause it to override information in an incorrect customer or user account.

As for adding a user, an insertion into the database is performed, where the new user record is appended into the database.

A screenshot of a computer

Description automatically generated

#### Updating or adding a new customer

Text

Description automatically generated

Functions nearly identical to the user-focused functions, however these account for the differences in the structure of the customer and user tables, as well as the entered information into the text boxes (the textbox initially used for the username input field for a user is re-purposed to serve as a field for a customer’s telephone number instead.)

#### When Confirm button is clicked

Text

Description automatically generated

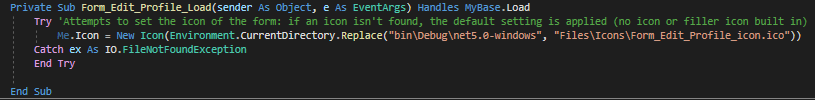
A handful of checks are performed in order to ensue that the data entered is valid depending on if a user or customer is being added, while outputting message box prompts for any invalid instances of data.

#### Miscellanious changes

Text

Description automatically generated

Ensures that the password field colour is correctly changed when either field doesn’t match.



Sets icon of form.

### Form\_Search

Graphical user interface

Description automatically generated

#### Methods, Constructor, Form\_Load and Set dictionary

Text

Description automatically generated

Sets the icon of the form, the mode of operation and fills in the dictionary with the values that correspond to the index of the quality selected (this feature, however, while present, is redundant in this version of the program, allowing any developer seeking to incorporate the filter for the quality of stock into the search form to do so.)

#### Validate Field Data, Help button clicked & GetResultFormSelection

Text

Description automatically generated

Validate\_field\_data is Commented extensively: checks if the data in the fields is in a valid format, meanwhile when help prompt button is clicked, a verbose message box is displayed detailing the form that the data should be in, as seen in the method below.

GetResultFormSelection returns the information about selected item when it was being selected for removal, for the remove form to have a look at.

#### Search button clicked

A screenshot of a computer

Description automatically generated with medium confidence

In the even that the card set prefix was entered as the entire code (XXXX-XX01), the code is trimmed.

Once all the data is valid, the data is passed into QueryItemTable method.

#### Query Item & Query Stock Table

Text

Description automatically generated

A check is carried out to see if item in question exists in the database: whether the name, set prefix or both have been provided, the condition of the query will be different.

Item dataset is passed into QueryStockTable by reference, where it will use it to match the relevant stock records to the selected item records, before tossing both of the datasets into search result form to display the information.

The quality filter

### Form\_Search\_Result

Graphical user interface

Description automatically generated

#### Constructor, attributes, load, and Get\_Info\_For\_Removal

Text

Description automatically generated

On load, set some local methods to parameters passed on from Form\_Search.

When the form loads: if the mode of the form is set to “Remove”, show the removal controls, then call the methods CreateRecordOrder and UpdateRecordDetails.

#### Create\_Record\_ORder & Update\_Record\_Details

A screenshot of a computer

Description automatically generated with medium confidence

Sets a private method RecordOrder as a list of objects, where each item holds the information about every item, which was passed on from Form\_Search.

Text

Description automatically generated

Sets the information of the textboxes to the information matching RecordOrder item, at the current NavigationIndex, which serves as a variable that tracks how much the user has scrolled through the records.

#### Confirm\_Delete & Record navigation methods

Text

Description automatically generated

When confirm delete button is clicked, information is prepared into the local method to be read by the search form, which will pass it onto the remove form.

Every time a new record is scrolled on with the navigation buttons, the information in the text boxes has to be changed and the buttons need to be enabled and disabled appropriately.

### Form\_Remove

Graphical user interface, application

Description automatically generated

#### Constructor, Removing, and adding info onto removal stack

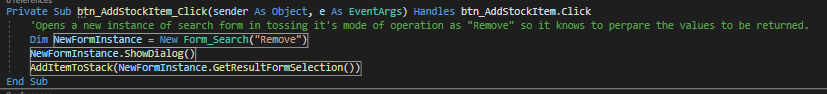
Text

Description automatically generated

UserID is provided so they can be associated with any changes to stock they commit.

If statement checks in AddItemToStack ensure that an item which is nothing is added onto the stack.

#### Adding item to remove stack



#### Confirming deletion of items

Text

Description automatically generated

Stock dataset is established, and removal mode is pending selection, where the user can choose to either make the removal a customer order or not: if they decide to make it an order, the user is able to access all avaliable customers within the database and either make a new customer or select an existing customer, based on their First name, surname and email.

Then, either the module handling order logging is called, or one which handles logging changes to system.

#### Remove items form stack button & cancel button

A screenshot of a computer

Description automatically generated

### Form\_Log\_Viewer

Graphical user interface, application

Description automatically generated

Graphical user interface, application, table

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, application, Word

Description automatically generated

#### Constructor & Form load

Text

Description automatically generated

Disables ability to back up database for any user that isn’t an administrator.

#### Menustrip functionality & module Backup

Text

Description automatically generated

Depending on what menu item is selected, a different file explorer or function is selected.

Text

Description automatically generated

#### Opening and saving a file

Text

Description automatically generated

Either loads the contents into the list box or saves contents from list box into a desired file, overriding it in the process.

### Form\_Add\_Item\_Or\_Stock

Graphical user interface

Description automatically generated

#### Constructor, Form Load and setting up quality dictionary

A screenshot of a computer

Description automatically generated with medium confidence

Quality dictionary automatically converts the index of the combo box into a valid quality, meaning that the text within isn’t restricted and can be customised.

#### Add to stock checkbox & adding image functionality

Text

Description automatically generated

Check disables previously enabled fields and shows the options for adding stock.

Browse button will open a dialog which allows the user to select an image form the dedicated image folder for the item they wish to add.

#### Button Add

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Establishes parameters for adding into the item table: if the checkbox that determines whether the item is added to stock table is not checked, the item is added into the item table only.

Text

Description automatically generated

If the item is being added as a stock record, the process will be interrupted if the price is not in the format of X.XX, then, the WhereCondition will determine if a stock record will be updated or added, depending on if an existing stock record is found or not (screenshot below)

Text

Description automatically generated

Finally, the stock values are established once the WhereCondition is established and either an insert or update is ran & different information is logged in the changes log.

### Form\_User\_Managment

Graphical user interface, application

Description automatically generated

#### Attributes, Constructor, form Closing and Setting up combo box

A screenshot of a computer

Description automatically generated with medium confidence

When the form is closed, the combo box information is cleared

Combo box information is different for if users or customers are being displayed.

Depending on the table being used and mode, certain buttons are hidden in the constructor.

#### Form Load

A screenshot of a computer

Description automatically generated with medium confidence

When the form is loaded, a message box will prompt the user asking if they wish to view users or customers: this option is skipped if the user comes from the removal form as only customers are desired then.

#### Editing or making new accounts

Text

Description automatically generated

Depending on whether a new customer or user is being made or existing one is being edited, the first temporary value in the array UserValues is utilised as a value which tells the edit profile form which type is being passed onto it.

#### Fetching customer info, selecting customer and closing form

A screenshot of a computer

Description automatically generated

Both of the methods are used to set up and toss the information about the selected customer onto the removal form.

### DB Communication Module

#### Initialisation & functions

Text

Description automatically generated

The main connection to the database file, established as a new Database object.

By far the most referenced function in the program is the GetDatabaseObject function

Graphical user interface, text, application, chat or text message

Description automatically generated

Used to close the connection to the card database.

#### DataBase Interface

Text

Description automatically generated

#### SQL\_Query\_Maker Class

Used by the DataSet class to create text for the SQL queries.

Text

Description automatically generatedText

Description automatically generated

#### Data Set Class

A screenshot of a computer

Description automatically generated with medium confidence

Establishes a new able from the database, depending on provided parameters

A screenshot of a computer

Description automatically generated with medium confidence

InsertNewRecordIntoDB automatically updates the database file with a new record appended.  
DeleteRecordInDB does the same but for a deletion.  
UpdateDB simply re-fills the information from the dataset back into the database, terminating the dataset in the process.

A screenshot of a computer

Description automatically generated

#### Data Base Class

Text

Description automatically generated

Establishes connection to the database in the constructor, meanwhile connection toggle opens and closes the connection.

Text

Description automatically generated

Accessibility features which are used to set the connection of the database, as well as get the connection status of it.

### Log Changes Module

Text

Description automatically generated

Formats the information provided to it, updates the database table & automatically writes to the currently active changes log file.

### Log Orders Module

Text

Description automatically generated

Similar to changes module, however this time it deals with a second data set which it needs to insert new records for every item within the order.

Text

Description automatically generated

Writes the information into a file that is marked by the order ID In its filename.

### DB Config Module

Most of the functionality of this module was made redundant due to technical difficulties implementing the automatic database backup system.

Text

Description automatically generated

Text

Description automatically generated

## Testing

### Signing In

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 1 | Correct User details are entered to log into the database. | Normal | User should be let past the log in form. | PASS | 1:15 - Development Walkthrough video |
| 2 | Correct username is entered but not password | Erroneous | User isn’t allowed to log in | PASS | 1:15 - Development Walkthrough video |
| 3 | Correct password is entered but not username | Erroneous | User isn’t allowed to log in | PASS | 1:15 - Development Walkthrough video |
| 4 | Form closes appropriately when dictated to. | Normal | Form closes and program stops execution. | FAIL | 27:46 - Development Walkthrough video |

### User Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 5 | Correct User details are entered | Normal | User is taken to the correct form type, depending on their access level. | PASS | 1:15 – Development Walkthrough video |

### Search & Search Result Forms

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 6 | Item Name and card prefix are entered, which exist in the database | Normal | Form Search Result will display any instances of records matching entered information | PASS | 6:55 – Development Walkthrough video |
| 7 | Correct item name but incorrect item prefix | Erroneous | Display Error message stating no records were found | PASS | Refer to Testing evidence section below |
| 8 | Correct item prefix entered but incorrect item name. | Erroneous | Display Error message stating no records were found | PASS | Refer to Testing evidence section below |
| 9 | Form is closed before any search is made | Normal | Form closes without performing any search. | PASS | 10:55 – Development Walkthrough video |
| 10 | Search Results form is closed | Normal | Form closes without performing any extra operations | PASS | 10:50 – Development Walkthrough video |
| 11 | Search Result form removal is cancelled | Normal | Form closes without performing any extra operations | PASS | Refer to Testing evidence section below |
| 12 | Remove button clicked for a highlighted record | Normal | Display sub form requesting for deletion quantity. | PASS | 18:10 – Development Walkthrough video |
| 13 | User selects appropriate quantity to remove of a record | Normal | Information is returned to remove queue form. | PASS | 18:10 – Development Walkthrough video |
| 14 | User cancels remove quantity selection | Normal | Sub Form closes without performing any extra operations and search results form stays open for the user to do something else on it. | PASS | (IDENTICAL TO TEST 11)  Refer to Testing evidence section below |

### Search Result Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 15 | Search result item is loaded into this form | Normal | All information is correctly loaded into the form. | PASS | 14:26, 15:42 , 17:00, 18:00 |
| 16 | No changes log info found about item | Normal | No information or an indicator of no information is displayed in the list box. | FAIL | FEATURE REMOVED DURING DEVELOPMENT |

### Add Item or Stock Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 17 | User enters correct information about a new item they wish to add. | Typical | A visual confirmation is shown to the user that the item has been added to the stock. | PASS | 14:26, 15:42 , 17:00, 18:00 |
| 18 | User enters an invalid item ID (anything that isn’t letters or numbers) | Erroneous | appropriate error message is shown. | PASS | Refer to Testing evidence section below |
| 19 | User attempts to leave necessary fields blank | Erroneous | User is prevented from making a new stock record. | PASS | Refer to Testing evidence section below |
| 20 | User attempts to enter very long text into fields that is in valid format | Extreme | Appropriate error message is shown and/or user is prevented from making stock record | PASS | Refer to Testing evidence section below |
| 21 | User attempts to find item name for an item that doesn’t exist | Erroneous | Error handling prevents any exceptions from appearing, an appropriate error message box may be shown. | FAIL | FEATURE REMOVED DURING DEVELOPMENT |
| 22 | User attempts to add a stock record that exists | Typical | As long as the notes field is the same, the price is updated and quantity is added to the existing quantity amount, otherwise a new record is created. | PASS | Refer to Testing evidence section below |
| 23 | User exits form without adding any items | Typical | Form closes without executing any add functions. | PASS | 16:18 – Development Walkthrough video |

### Remove Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 24 | User successfully selects an item to add to remove queue | Typical | Information about the item added is displayed | PASS | 17:31, 18:15 – Development Walkthrough video |
| 25 | User cancels the addition of a new item, after they have pressed the “Add item” button | Erroneous | Adding process is cancelled. | PASS | Refer to Testing evidence section below |
| 26 | User attempts to delete an item that exists in the queue. | Typical | Queue is updated and information about the item is removed. | PASS | Refer to Testing evidence section below |
| 27 | User attempts to delete an item from queue when it is empty | Erroneous | Error is handles and a message box prompt may be shown. | PASS | IDENTICAL TO 17:40 -Development Walkthrough video |
| 28 | User confirms deletion with items In queue | Typical | Records are updated and/or deleted where appropriate, change is logged and the user receives a confirmation message telling them the process was successful. | Updating: PASS  Deleting: FAIL | 18:50 , 19:20, 19:50 – Development Walkthrough video |
| 29 | User attempts to confirm deletion when the queue of items is empty | Erroneous | No deletion or updating is carried out: an appropriate message box prompt may be shown. | PASS | 17:40 – Development Walkthrough video |
| 30 | User closes the form using the cancel button or close button | Typical | Form closes and any process related to the form and deletion/updating of records is cancelled. | PASS | IDENTICAL TO 25 (Refer to Testing evidence section below) |

### Log Viewer Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 31 | User attempts to open a correct log file | Typical | Information from the log file is dumped into the list box | PASS | Refer to Testing evidence section below |
| 32 | User attempts to save a log file correctly | Typical | File is successfully filled with the information stored in the list box | PASS | Refer to Testing evidence section below |
| 33 | User attempts to open an incorrect file type | Erroneous | User is prevented from opening file by the file explorer’s filter settings. | PASS | IDENTICAL TO 31, Refer to Testing evidence section below |
| 34 | User uses the backup active log file function | Typical | File is backed up in a pre-set directory. | FAIL | FEATURE REMOVED DURING DEVELOPMENT, REPLACED WITH BACKUP OF DB  11:10 – Development Walkthrough video |
| 35 | User uses the Show active log filepath function | Typical | The directory leading to the currently active log file is shown. | FAIL | SAME AS 34,  11:10 – Development Walkthrough video |

### User management Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 36 | User selects and presses confirm button | Typical | New form instance where the user can edit that account information is opened. | PASS | 25:15 , 25:50, 26:15, 26:30 – Development Walkthrough video |
| 37 | User closes the form or presses the cancel button | Typical | Form is closed and any related functions are cancelled: no changes are made to any of the user accounts. | PASS | Refer to Testing evidence section below, Resolves error seen on 25:30 of Development Walkthrough video resolved |

### Edit Profile Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Description | Data Type | Expected Result | Pass/Fail | Evidence Ref |
| 38 | User enters valid information that is the same or different to the previous information of their account. | Typical | User details are updated. | PASS | 2:45, 4:45 – Development Walkthrough video |
| 39 | User attempts to leave any or all fields blank | Erroneous | User is prevented from saving the changes and a message box prompt is shown. | PASS | Refer to Testing evidence section below |
| 40 | User attempts to enter a password exactly 10 or 30 characters in length, with the rest information being correct | Extreme | User details are updated. | PASS | Refer to Testing evidence section below |
| 41 | User enters information that is too short (username of two or less characters or longer than 30, as well as passwords with length of less than 10 or more than 30 & any other relevant fields.) | Erroneous | User is prevented from saving their erroneous details and a message box prompt is shown. | PASS | Refer to Testing evidence section below |
| 42 | User enters correct details, but the password and confirm password fields don’t match. | Erroneous | User is prevented from saving their erroneous details and a message box prompt is shown. | PASS | Refer to Testing evidence section below |

## Testing Evidence (That’s not Already in video walkthrough)

|  |  |
| --- | --- |
| Test No | Evidence |
| 7 |  |
| 8 |  |
| 11 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 22 |  |
| 25 & 30 | Cancelling here leads to:    If items added:    Cancelling here shows user form |
| 26 |  |
| 31 & 33 |  |
| 32 |  |
| 37 | Added Exit Sub |
| 39 |  |
| 40 |  |
| 41 |  |
| 42 |  |

# Evaluation

## Reflection on Analysis

|  |  |  |
| --- | --- | --- |
| Objective | Met? | Comment |
| Create an offline database which holds the stock log and its information within. | Yes | My system utilises a 2002/2003 version of a .mdb Microsoft Access database. |
| Implement a method of authentication to allow users of different access privileges to use the database implemented. | Yes | The system authentication allows for 3 different access levels: User, manager, and administrator, with system functionalities increasing as access level increases. |
| Allow for both automatic and manual back-up creation. | Partial | Manual backup creation is implemented for the database file and any log files via log viewer, but automatic database backup was unable to be implemented, possible reason is due to potential incomplete process shutdown. |
| Keep track of all changes made to the main log, including date, time, and user. | Yes | Changes log keeps track of user changes to stock items on the system. |
| Implement a solution that will prevent deadlock: timestamp logging. | Yes | The system will keep timestamps for record entries outside of the database. |
| Create sufficient error handling upon the launch of the application (like if the database file is missing or perhaps is corrupt.) | Yes | The system will display an error message if the database file was not found. |
| Create a feature which automatically attempts to recover the most recent, non-corrupt database file in the event the primary file fails. | No | This implementation proved too difficult to implement for me as an individual. |
| Allow for manual debugging of the system for the manager (or system manager if one is hired) if needed with the use of a special mode which displays verbose information about the database and queries. | Partial | Log viewer allows for an investigation into changes made by accounts of Stock manager and administrator privileges. |
| Allow for test queries to be used in debug mode, where the original database is preserved | Partial | Search mode serves as a partial solution to this key area: the administrator can additionally create test items and test stock records for purpose of debugging if needed. |
| Allow for the changes document to be exported as a back-up or for analysis | Yes | Log viewer form allows for manual exporting of document copies which are loaded into it. |
| Create a Feature which will go through the entire database and search for errors  A document should be created which provides information about the error, as well as what data-item caused the error. (This will be mainly used if someone manages to insert an invalid record into the structure, which bypassed prior checks.) | No | This task was made redundant due to checks on what information can be entered into the fields, however a working system debugger like this idea suggests would also be beyond my technical knowledge of implementing. |
| Allow for single and bulk entries to be added, removed, and updated to and from the database. | Yes | Multiple quantities of the same item can be added to the stock at a time, meanwhile multiple different stock records can be removed as well. |
| Prevent incorrect data from being entered with error handling for easy use. | Yes | My program provides sufficient error handling |
| Check against a set of valid card IDs to:  - Allow for a single card to be linked to a whole pack, vice versa.  - Allow for validation checks to ensure data entered, such as ID of card is correct. | Yes | The database uses a field for all items called the ItemSetPrefix, which is a 2,3, or 4 character log code: this code is the same for all cards that are part of a card set, meaning that the card set also possesses this code.  As for validation, a stock record entry cannot be added if a valid item isn’t found in the database first. |
| A database which holds all the sellable cards and a separate one keeping track of the stock: that way when a user searches for a card that is not in stock, they can still get information on the card. | No | Due to difficulties in development, this feature was changed to instead allow for a search item that are already present in the stock record only, however, this implementation is appropriate because the system does not delete records with quantity of 0, so in theory any item that was used in the database for transactions can be searched for. |
| Perhaps a small transaction history for every kind of card, dependent and/or independent of rarity of a card? (eg: to check how much a specific kind of card sold for: make this a toggle feature perhaps, so the user doesn’t have to wait for the log to appear?) | Yes | Order log for each transaction is created, which displays the total cost of order, as well as a breakdown for each item in the order. |

## External User Feedback

I have gotten feedback from two volunteers: an individual who had experience using database systems before to handle small business things & a regular individual with no prior knowledge about database systems.  
While they were chosen by coincidence, they allowed me to get a broader sense of what was good and what was bad about the system.

They have given me the following feedback after I have given them a set of tasks to complete on the system (they were given stock manager privileges):

* “The order log is confusing to read from, I would prefer to get the name of the customer and the information surrounding the order, rather than have to go looking into the database for the information.”
* “Compared to the order log, the change log is a lot more understandable to read information from.”
* “Any change to my user profile I would like followed by a password confirmation check as an extra layer of protection”
* “I like the simplistic design of the forms”
* Ease of use ranking = 8/10
* “I would wish to be able to press enter upon entering my username and password for the system.”
* “Make the order log into a visual table to allow it to be better understood”
* “The log activity form is confusing to look at”
* “The font should be bigger to allow for better readability”
* “Adding an item and then adding it into stock should be able to be done in a single step to avoid confusion & speed up the process”
* “While visually simplistic, it could look more pretty.”
* Ease of use Raking = 8.5/10

## Reflection on Feedback & Developments

From the user feedback, I could gather that the overall design of the system was easy to use, however some features were confusing to look at (mainly the Order Log form and Log viewer form) and the overall design of the system was rather bland & could be more accessible to visual impairment.

A good improvement would be a visual overhaul, with subtle colouring and more room on forms: this improvement would lead to a more appealing design as well as one which is more accessible.  
This could be integrated by using additional toolbox items such as group boxes: a functional design overhaul, where adding items would provide more visual indication to the process would require more in-depth use of toolbox items, item class instances and functionality changes.

A major implementation improvement would involve smarter usage, maintenance, and manipulation of database objects, such as datasets and methods of allowing the removal or records from the database properly.

A search quality filter could have been implemented, as it would allow for a narrowing down of a search.

Another major improvement would be a working implementation of an automatic backup system for both stock files and any order & change logs, in the event of random failure or corruption, such as during a power outage.