

Design Document for DeckTechCentral

PennWest California

CMSC 4900 – Senior Project I

Dr. Chen

12/05/2023

Team Member	Major	Project Phase
Christian Messmer	Computer Science	Implementation
Paul Shriner	Computer Science	Analysis
Adir Turgeman	Computer Science	Presentation
Luke Vukovich	Computer Science	Design

Instructor Comments and Evaluation

Table of Contents

Instructor Comments and Evaluation	1
Table of Contents	2
Abstract.....	3
Description of Document.....	4
Purpose and Use	4
Ties to the Specification Document	4
Intended Audience.....	4
Project Block Diagram	5
Design Details	7
System Modules and Responsibilities.....	7
Architectural Diagram	7
Module Cohesion.....	8
Module Coupling.....	8
Design Analysis.....	10
Data Flow Analysis	10
Design Organization.....	11
Description of Classes/Objects.....	11
Functional Description	16
Decision: Programming Language/Reuse/Portability	29
Implementation Timeline	30
Design Testing.....	30
Appendix.....	32
Team Details	32
Workflow Authentication.....	34
Report from the Writing Center	35
References	36

Abstract

The current document is the design of the DeckTechCentral (DTC) software product, a full stack web application for viewing, creating, and managing Magic: The Gathering (MTG) deck lists. Magic: The Gathering is a trading card game with in-person and online playing options (Wizards, n.d.). The document begins with an overview of the system architecture by characterizing the separate modules and their coupling. The next topic outlined is the purpose of each class in each module and lists of their data members and methods. Then provided is a detailed description for each method of the classes, a narrative describing user interaction, and an implementation timeline. The final topic described is the testing process used in the production of this document.

Description of Document

Purpose and Use

The purpose of the Design Document is to fully describe the architecture and technical details of the project. These include but are not limited to the modules for the project, implementation details such as programming languages used, testing details, and an implementation timeline. This document will be used by the DeckTechCentral development team throughout the project to stay on track.

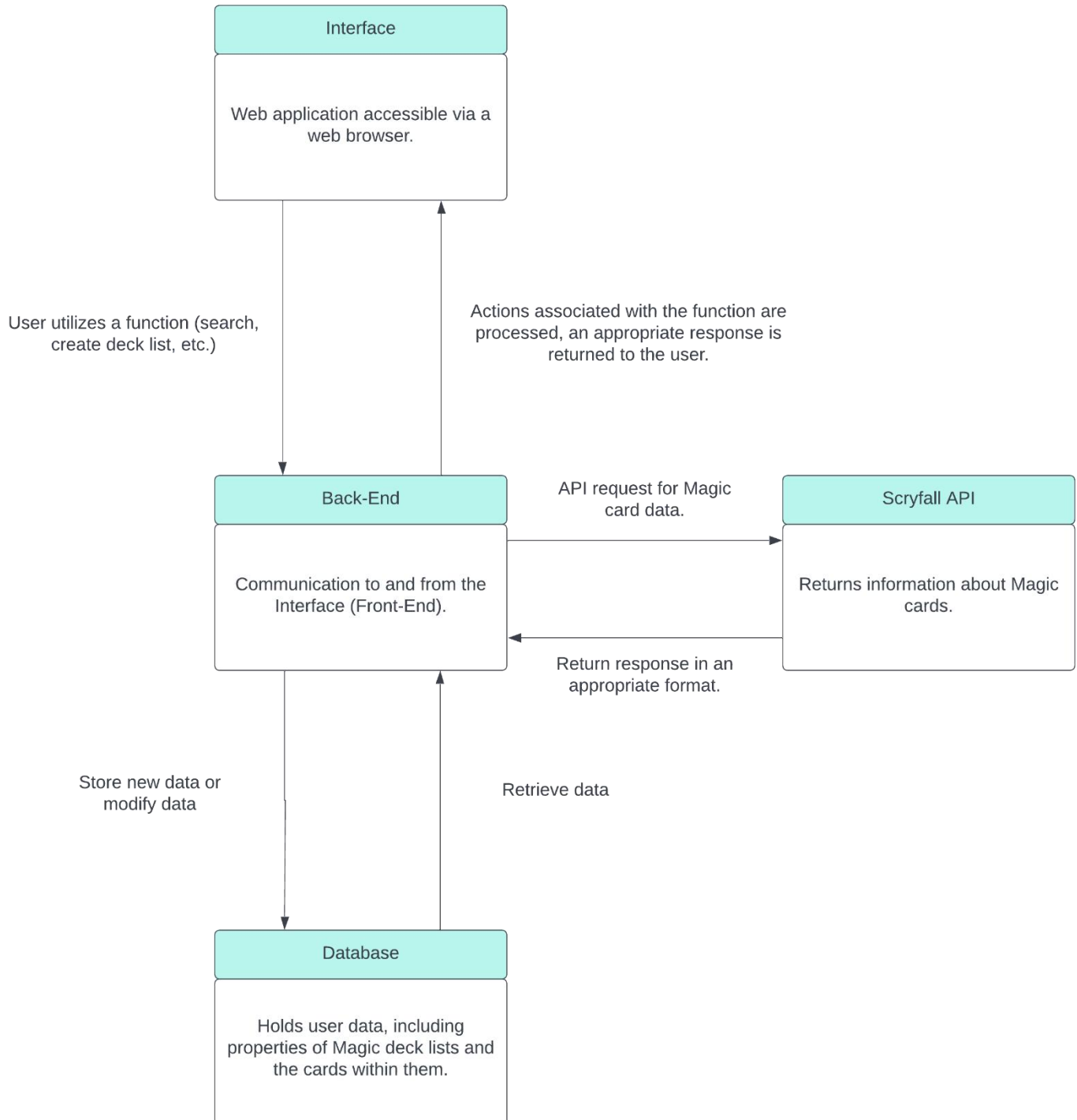
Ties to the Specification Document

The Specification Document aimed to give a broad understanding of what DeckTechCentral does to a potentially nontechnical audience. This document builds and extends on the Specification Document from a development perspective.

Intended Audience

The intended audience of the Design Document is software developers. Particularly, it will be used by the DeckTechCentral team during the development process. After the project is released, other developers may find this document useful in understanding implementation details and decisions made by the DTC team. This document will contain technical details and concepts, and as such, is not intended for someone without technical knowledge.

Project Block Diagram

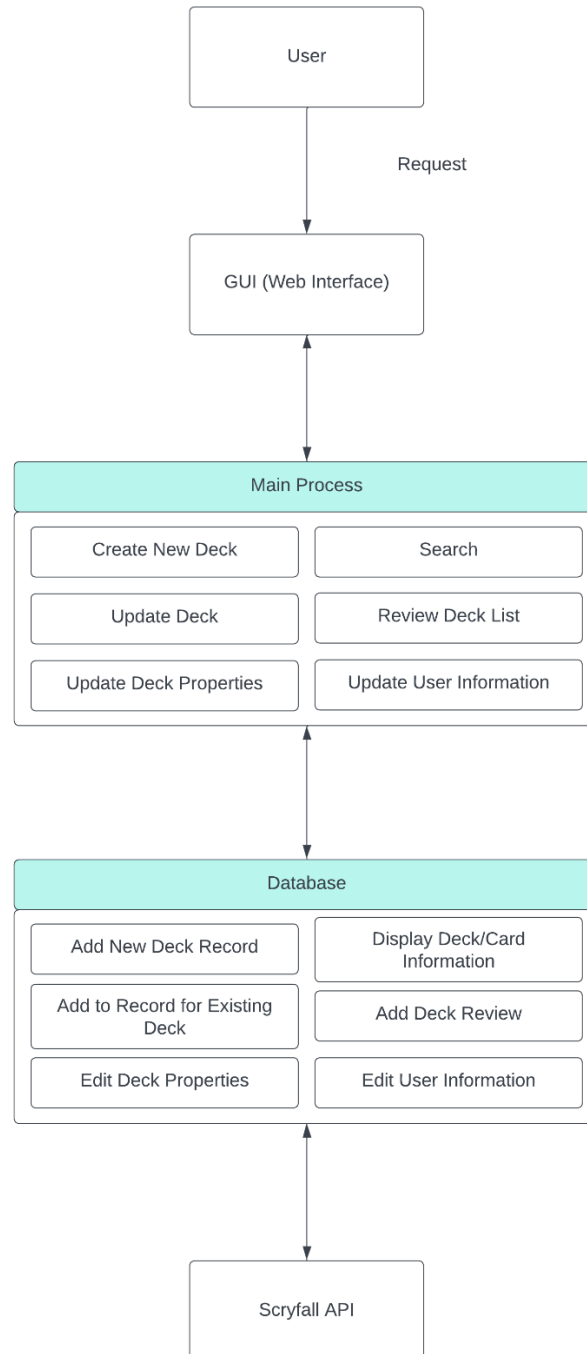


The DeckTechCentral application will consist of three parts: a user interface (Front-End), Back-End, and Database. The user interface will run on a modern, up-to-date web browser. A user can perform actions through the user interface, for example adding a card to a deck list. This request will be intercepted by the Back-End. To obtain information on a Magic card, an API request is made to the Scryfall API, which returns information pertaining to the card. When the data has been prepared, it will be stored in the database. Data can be retrieved from the database, and responses to the user are returned to the user interface from the Back-End.

Design Details

System Modules and Responsibilities

Architectural Diagram



A user will make a request using the GUI (Web Interface). These actions are part of the Main Process, including “Create New Deck”, “Update Deck”, “Search”, and more. When an action is performed, data will need to be accessed or edited, which is part of the Database. Database actions include “Add New Deck Record”, “Edit Deck Properties”, “Edit User Information”, and more. In the case of returning information for a Magic card, the Database may not have the needed information already available. This will require making an API request to the Scryfall API. Data is sent and received between the GUI (Web Interface), Main Process, Database, and Scryfall API. Ultimately, the user’s request will be completed.

Module Cohesion

Cohesion refers to the relationships within a given module (GeeksforGeeks, 2019). In the DeckTechCentral application, related functionality is kept together in their respective classes. For example, the Deck class holds information about a deck, while the User class holds information about a user. This results in high cohesion within modules, allowing for greater clarity and comprehension.

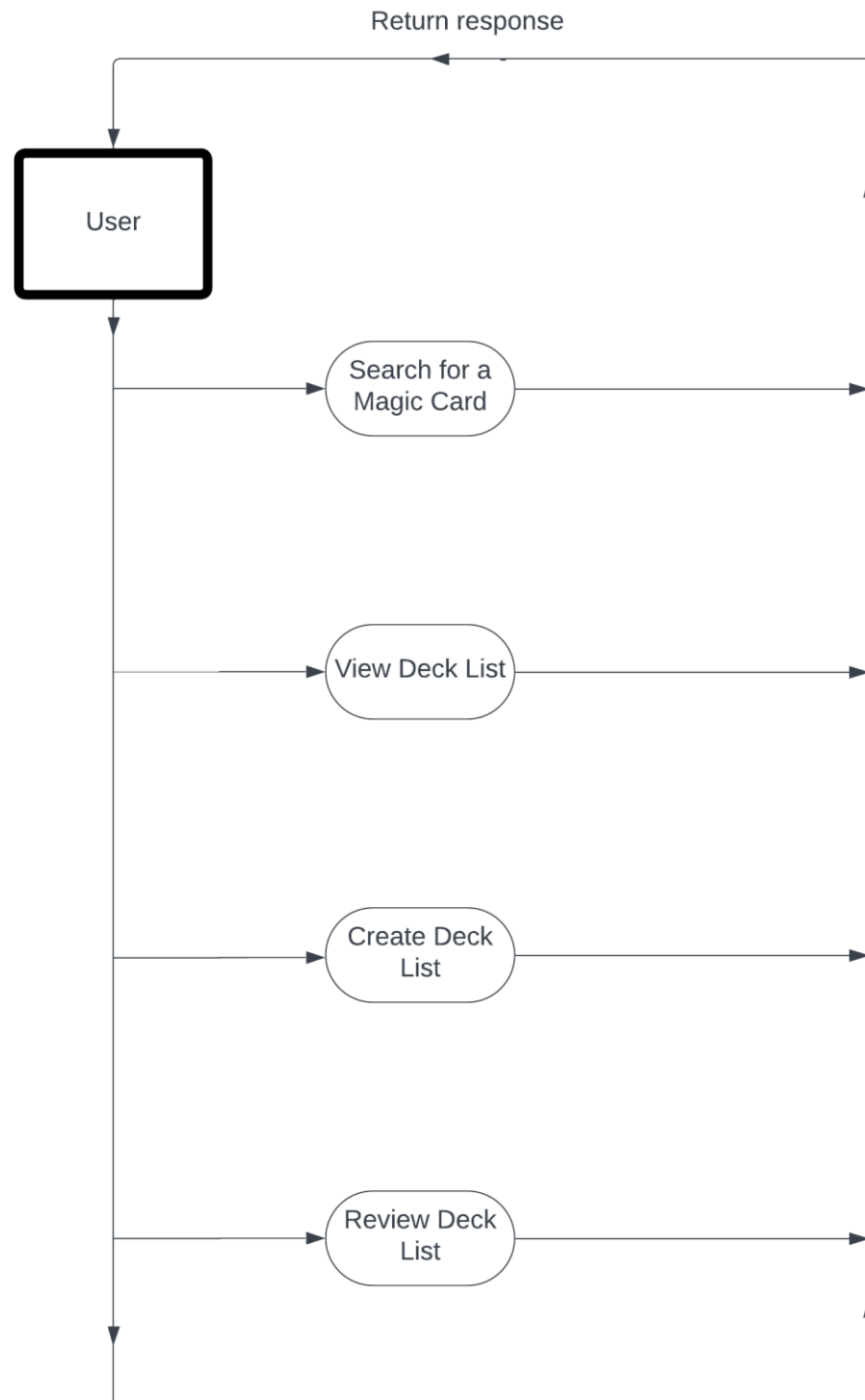
Module Coupling

Coupling refers to the relationships between modules (GeeksforGeeks, 2019). In the DeckTechCentral application, while modules do have some relationships between each other, they are still separate modules and can stand on their own. For example, a User class object can be connected to multiple Deck class objects, but User and Deck themselves are separate. The data modification and accessor functions for the Deck class can’t be used on the User class and vice versa. A similar description can be made between the Deck and Card classes. This results in

reduced coupling, where the separate modules work together to result in the DeckTechCentral application.

Design Analysis

Data Flow Analysis



The provided data flow diagram gives a broad overview of the requests and responses in the DeckTechCentral application. The four main actions a user will be able to take are as follows: Search for a Magic Card, View Deck List, Create Deck List, and Review Deck List.

Search will take the given parameters and pass them to the Scryfall API. The response from the API will be parsed and evaluated. This can succeed or fail based on if the parameters given by the user are for a valid Magic card. View Deck List will take the user to the landing page for the corresponding deck. This can fail if a user tries to view an invalid deck or a private deck (which they don't have permissions to view). Create Deck List takes the user to the landing page with the functionality required to create a deck list. Finally, Review Deck List takes the user to the landing page for reviewing a deck list with its corresponding functionality.

At the end, responses generated will be returned to the user. These can be in the form of success or error messages. Functionality will need to be present to handle the responses appropriately for the user. For example, failure messages will need to be evident to the user, while success messages do not need to be (they could be shown in the browser console).

Design Organization

Description of Classes/Objects

Tabular Description of User Class

Module Name	User Class
Module Type	Class
Data Members	username : string userID : GUID verified: boolean email : string banned : boolean
Member Functions	None.

Description	The User class the information that all users on DeckTechCentral have. A User object will hold information about a user which can be viewed (and possibly changed) by the user. It will also enable functionality of the application or disable it entirely, based on the verified and banned data members.
-------------	---

Tabular Description of Moderator Class

Module Name	Moderator Class
Module Type	Class
Data Members	None
Member Functions	suspendUser() banUser() restoreUser()
Description	The Moderator class is a derived User class. This will have the same properties of a User class, but in addition have Moderator specific properties.

Tabular Description of Deck Class

Module Name	Deck Class
Module Type	Class
Data Members	name : string deckID : GUID cardList : list<Card> user : User type : string public: boolean reviews : list<Review> dateCreated : date
Member Functions	None.
Description	The Deck class holds the information for a Magic deck. A deck will be made up of several Card objects, in addition to property data members such as the name of the deck. These will be viewable and editable by the user.

Tabular Description of Card Class

Module Name	Card Class
Module Type	Class
Data Members	name : string manaValue : int manaCost : string color : string colorID : string type : string subType : string description: string textBox : string power : int toughness : int image : string
Member Functions	None.
Description	The Card class the information for a Magic card. This allows for readily accessing the information of a single Card object and connecting several Card objects to one or more Deck objects.

Tabular Description of Search Class

Module Name	Search Class
Module Type	Class
Data Members	input : string sortType : int result : string
Member Functions	sendInputToAPI() parseResult()
Description	The Search class handles searches done by the user. The input will be obtained and sent to the Scryfall API. Then, the result will be parsed and returned. This class is not meant to show an error message (such as for an invalid search parameter), that will be handled by the search interface class.

Tabular Description of Review Class

Module Name	Review Class
Module Type	Class
Data Members	content : string title: string date: string userID: string visible: boolean
Member Functions	None.
Description	The Review class holds information for a deck review. Reviews are made by users and will have a title, date, and content. There is also be a “visible” variable, as it’s possible the review won’t be visible due to the user being banned.

Tabular Description of Deck Interface Class

Module Name	Deck Interface Class
Module Type	Class
Data Members	None.
Member Functions	displayDeckInfo() displayCardNames() displayCardImages() sortCards() createDeck() deleteDeck() changeDeckInfo()
Description	The Deck Interface class holds the functionality for realizing the user interface of a deck list. It will allow the user to view basic information about the deck (title, date created, rating), view the cards of the deck, and sort the cards into different orders.

Tabular Description of Review Interface Class

Module Name	Review Interface Class
Module Type	Class
Data Members	None.
Member Functions	displayReview() sortReviews()

	createReview() deleteReview()
Description	The Review Interface class holds the functionality for realizing the review interface for a given deck list. It will allow the user to view the reviews, sort the reviews, and create/delete their own review.

Tabular Description of Search Interface Class

Module Name	Search Interface Class
Module Type	Class
Data Members	None.
Member Functions	displaySearchBox() displayQuery() showSearchOptions()
Description	The Search Interface class holds the functionality for realizing the search box. It will display a search box, display the query as the user is typing in, and display search options such as date the card is made (this will be appended to the search query when it gets passed to the API).

Tabular Description of Profile Interface Class

Module Name	Profile Interface Class
Module Type	Class
Data Members	None.
Member Functions	displayUserInfo() changeUserInfo()
Description	The Profile Interface class holds the functionality for realizing a profile page for the user. It will display various information such as username and email, as well as allow the user to change information (if possible).

Tabular Description of Main Interface Class

Module Name	Main Interface Class
Module Type	Class
Data Members	None.
Member Functions	displaySearch() displayStatusBar() displayDecks() enableModeratorView()
Description	The Main Interface class holds the functionality for realizing the main interface; it will be the primary element the user interacts with. It will display the search box, status bar, and a list of public decks. When possible, functionality from other classes will be used (for example the displaySearchBox() function from the Search class can be used, reducing redundant code).

Functional Description

Moderator Class

suspendUser()

Input:

userID for the user to suspend, time duration for the suspension, and reason for suspension.

Output:

The user will be suspended for time given. When they attempt to login, the suspension and time will show.

Return Parameters:

A success/failure value will be returned.

Types:

Strings for userID, time duration, and reason. Boolean for the success/failure value.

`banUser()`*Input:*

userID for the user to ban, reason for ban.

Output:

The user will be permanently banned. When logging in, they will be notified of this.

Return Parameters:

A success/failure value will be returned.

Types:

Strings for userID and reason. Boolean for the success/failure value.

`restoreUser()`*Input:*

userID for the user to unban or unsuspend.

Output:

The user will have any kind of ban lifted. They can log in as normal.

Return Parameters:

A success/failure value will be returned.

Types:

Strings for userID. Boolean for the success/failure value.

Search Class

sendInputToAPI()

Input:

The string to use as the search to send to the Scryfall API.

Output:

A JSON object containing the data of the result from the search query.

Return Parameters:

None.

Types:

String for search, and JSON object for the result.

parseResult()

Input:

JSON object to parse.

Output:

An array of card objects that have been initialized by the data from the JSON given. If no cards were found, then no card objects will be in the array.

Return Parameters:

None.

Types:

JSON object for the input for parsing, and an array of card objects to return.

Deck Interface Class

displayDeckInfo()

Input:

None.

Output:

Information for the deck is displayed.

Return Parameters:

None.

Types:

Types are the same as in their related classes.

displayCardNames()

Input:

None.

Output:

The names of the cards in a deck will be displayed.

Return Parameters:

None.

Types:

Types are the same as in their related classes.

displayCardImages()

Input:

None.

Output:

The images of cards in the deck will be displayed.

Return Parameters:

None.

Types:

Types are the same as in their related classes.

sortCards()

Input:

ENUM of how to sort the cards.

Output:

The cards will be displayed to the user in the desired sort.

Return Parameters:

None.

Types:

Types are the same as in their related classes.

`createDeck()`

Input:

None.

Output:

A new deck object is created.

Return Parameters:

None.

Types:

Types are the same as in their related classes.

`deleteDeck()`

Input:

GUID of the deck to delete.

Output:

The deck with the matching GUID is deleted.

Return Parameters:

None.

Types:

Types are the same as in their related classes.

changeDeckInfo()

Input:

GUID of deck to edit, and a JSON object of the new data for the deck info.

Output:

The deck's information will be changed.

Return Parameters:

None.

Types:

GUID for deck and JSON object for data. Other types are the same as in their related classes.

Review Interface Class

createReview()

Input:

GUID for the deck to review, text of the review, and user making the review.

Output:

A new review object is created.

Return Parameters:

None.

Types:

GUID for the input data. Other types are the same as in their related classes.

`deleteReview()`

Input:

GUID for the deck to delete.

Output:

None.

Return Parameters:

None.

Types:

GUID for the deck. Other types are the same as in their related classes.

`displayReview()`

Input:

GUID of the deck to show reviews from.

Output:

JSON object of all reviews for the given deck.

Return Parameters:

None.

Types:

GUID for the deck and a JSON object for reviews. Other types are the same as in their related classes.

sortReviews()

Input:

ENUM of what to sort the reviews by.

Output:

Reviews are not sorted according to the sorting method displayed.

Return Parameters:

None.

Types:

ENUM for sorting method. Other types are the same as in their related classes.

Search Interface Class

displaySearchBox()

Input:

None.

Output:

The search box is displayed.

Return Parameters:

None.

Types:

None.

displayQuery()

Input:

The string of the user input obtained from the input stream medium.

Output:

The input string is displayed in the search box.

Return Parameters:

None.

Types:

String for the user input.

showSearchOptions()

Input:

None.

Output:

The search options, including sorting and dates, will be shown.

Return Parameters:

None.

Types:

Related types from other classes will be used, such as names of deck categories.

Profile Interface Class

displayUserInfo()

Input:

None.

Output:

The user information will be changed.

Return Parameters:

None.

Types:

Related types from other classes will be used.

changeUserInfo()

Input:

The information to be changed.

Output:

The information will be changed.

Return Parameters:

None.

Types:

Related types from other classes will be used. For example, changing the username will use the same type of string.

Main Interface Class

displaySearch()

Input:

None.

Output:

When the search bar is pressed, the related functions from the Search Interface class will be called.

Return Parameters:

None.

Types:

Related types from other classes will be used.

displayStatusBar()

Input:

None.

Output:

Status bar elements will be displayed to the user. This is intended to be constantly running, as the status bar has to always be visible or accessible.

Return Parameters:

None.

Types:

Related types from other classes will be used.

displayDecks()

Input:

ENUM for the sorting method.

Output:

A list of public decks will be displayed in accordance with any options the user selected, such as a sorting method.

Return Parameters:

None.

Types:

An ENUM will be used for the sorting method. Related types from other classes will be used.

enableModeratorView()

Input:

The userID for the user.

Output:

If the user is a moderator, then moderator specific actions will be shown. For example, moderators can delete reviews using the same functionality the creator would have.

Return Parameters:

None.

Types:

Related types from other classes will be used.

Decision: Programming Language/Reuse/Portability

The Front-End will be written using React, which will allow for high portability as the resulting application will be accessible via a web page. The Back-End will be written using C#, which was chosen due to Christian Messmer's experience using it and its applications towards

Back-End components. Throughout the development process, components will be reused when possible. For example, a Moderator is able to delete an Authorized User's deck, but it would not make sense to design a deck removal function just for the Moderator. Instead, the Moderator can use the same functionality that the Authorized User has access to.

Implementation Timeline

Activity	January	February	March	April	May	May+
Setup development environment and version control						
Minimum Back-End and Front-End Functionality						
Complete Back-End						
Complete Front-End						
Testing						
Documentation						
Write Users Manual						
Release initial version						
Identify possible changes for future versions						

Design Testing

The Design Document was written by the team utilizing a shared Microsoft Word document, with each team member working on different components of the document. Initial testing was done by brainstorming ideas and creating a rough draft outside the document, then creating a final draft inside the document. Throughout this process, team members would share ideas and review each other's work as it was being done. At the end, a final test was done by having each team member read through the whole document.

Testing will also occur during all points of the development process. In the beginning, this will consist of manual code reviews or testing using small driver programs, as large portions

of the application will not be completed until late in the development process. Once the application becomes more fleshed out, testing can include additional functionality to ensure correctness and performance. Finally, with permission from the professor of the senior project course, testing can be extended to include those outside of the development team, which may reveal bugs missed in earlier testing.

Appendix

Team Details

The current revision of the design document was prepared by the team. The team takes on the responsibility of revising the document and preparing all necessary material for the project at hand. The team utilized Discord and public meeting areas to successfully meet online and in person. Contributions to the document at hand are outlined below.

Christian Messmer finished up and improved upon the functional description of the product design. Christian utilized his knowledge and experience with Magic: The Gathering to make several additions and improvements to other sections of the document, including the Description of the Document and Appendix.

Paul Shriner set up the layout and formatting of the document, including elements such as the Table of Contents and Works Cited. Paul worked on the design details such as the tabular description of the classes and made initial versions of the functional descriptions. Finally, Paul set up the Glossary and References.

Luke Vukovich acted as the workflow leader of the document. Luke worked on describing the system modules and their responsibilities. Once the team agreed on the system module design, Luke thoroughly described the modules accordingly. Luke also worked on the design analysis of the product at hand.


Adir Turgeman added details throughout the document to ensure consistency and completeness. He paid special attention to areas where details could potentially mismatched, such as the tabular and functional descriptions of the classes. Finally, Adir reviewed the

document several times and acted as another approval layer to catch potential mistakes and oversights.

The entire team worked collaboratively on specifying the implementation timeline of the product. This also led to the realization of proper design and architecture testing.

Workflow Authentication

I, Christian Messmer, attest that I executed the functions listed within the team details section of the document. Also, I agree with all the information stated within the design document.

Christian Messmer		December 5, 2023
Printed Name	Signature	Date

I, Paul Shriner, attest that I executed the functions listed within the team details section of the document. Also, I agree with all the information stated within the design document.

Paul Shriner		December 5, 2023
Printed Name	Signature	Date

I, Adir Turgeman, attest that I executed the functions listed within the team details section of the document. Also, I agree with all the information stated within the design document.

Adir Turgeman		December 5, 2023
Printed Name	Signature	Date

I, Luke Vukovich, attest that I executed the functions listed within the team details section of the document. Also, I agree with all the information stated within the design document.

Luke Vukovich		December 5, 2023
Printed Name	Signature	Date

Report from the Writing Center

See 12/5/2023 email from Christian Messmer.

References

MKS075. (2019, April 15). *Software Engineering / Differences between Coupling and Cohesion*.

GeeksforGeeks. <https://www.geeksforgeeks.org/software-engineering-differences-between-coupling-and-cohesion/>

What is MTG. (n.d.). Magic: The Gathering; Wizards of the Coast.

<https://magic.wizards.com/en/intro>