1. Name: Thomas Quinn Langsfeld
Title: Magic: The Gathering Search App

2.Project Description:

A website service for searching and displaying Magic: The Gathering trading cards. Users are able to search for specific cards as well as manage a collection of favorite cards and a single deck consisting of cards. Set up for one client and one MTG set only.

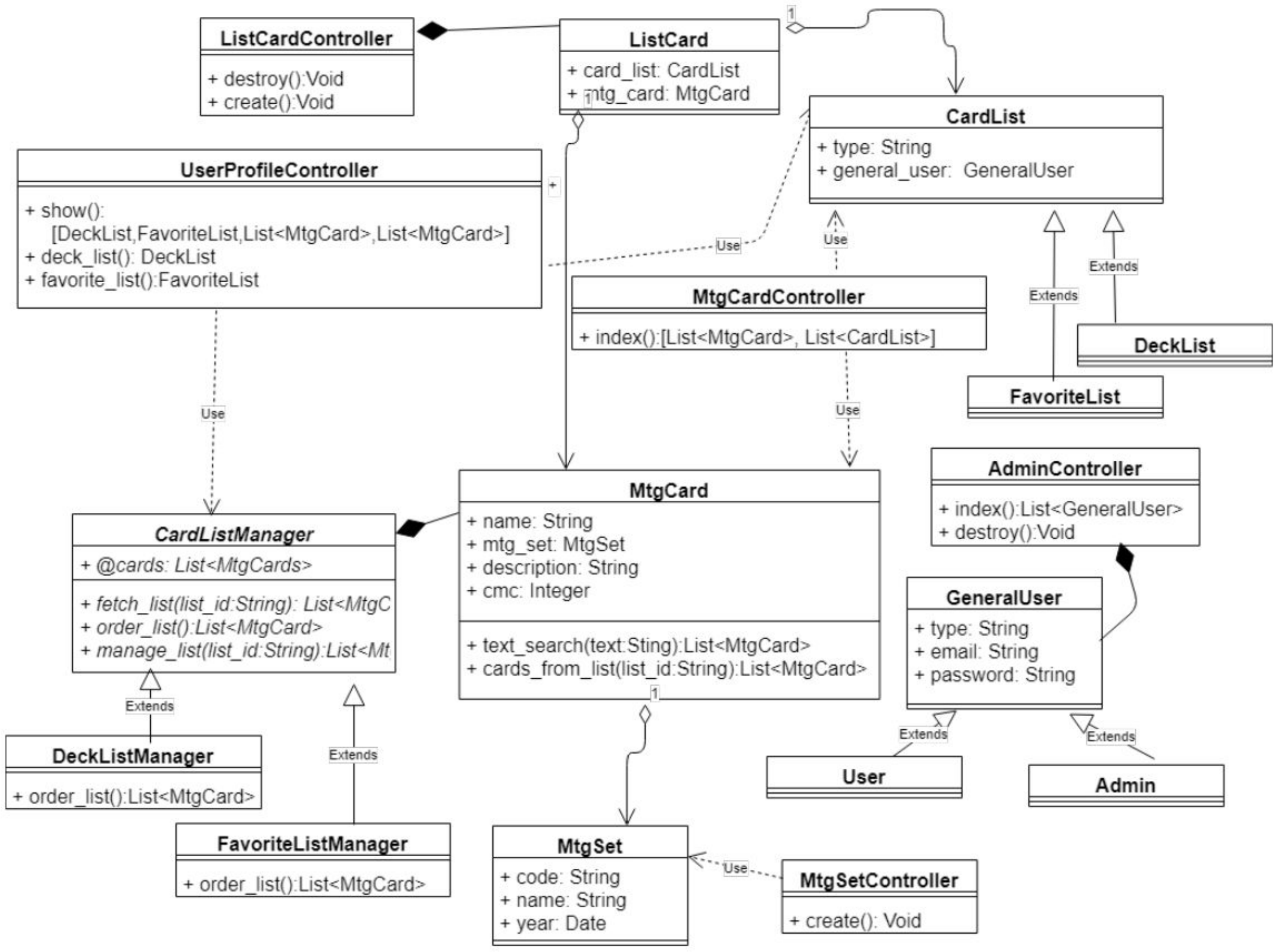
3.Features Implemented:

Feature ID:	Feature Title:	Feature Description:
Y1	Search	User, Admins can search for cards
Y2	Sign Up	Users,Admins can sign up
Y3	Update Site	Admin can update site list of MTG cards
Y4	Add to FavoriteList	Users, Admins can add a card to their favorite list
Y5	Add to DeckList	User, Admins can add a card to their deck list
Y6	Remove from FavoriteList	User, Admins can remove a card from their favorite list.
Y7	Remove from DeckList	User, Admins can remove a card from their deck list.
Y8	Deactivate users	Admin can delete users,admins

4.Features Not Implemented:

Feature ID:	Feature Title:	Feature Description:
N1	Add Admin	Admin can create account for new admin
N2	Create Deck	User,Admins can create a new deck list
N3	Create Fake	User,Admin can create a fake card
N4	Delete Fake	User,Admin can delete a fake card

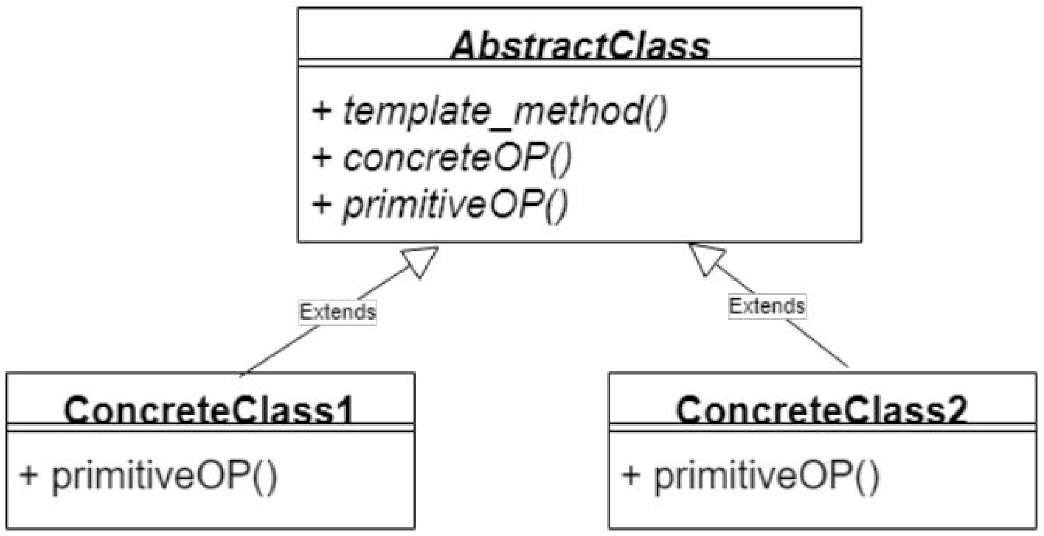
5.Final Class Diagram:



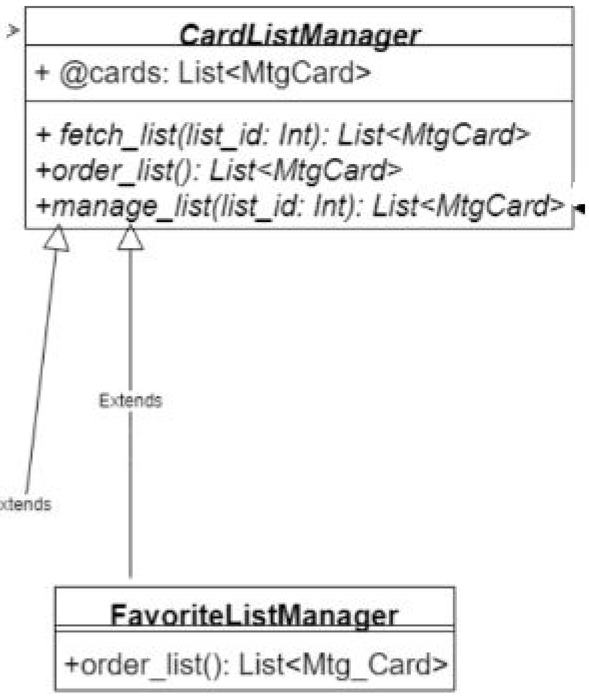
Note: All the models in the above class diagram (any class that is not a Manager or Controller) inherit from Ruby on Rails *ApplicationRecord* from the ActiveRecord ORM.

6.Template Design Pattern:

Template Class Diagram (adapted from *CSCI4448/5448 Object-Oriented Analysis and Design*, Elle Boese):



My Template Implementation Class Diagram:


CardListManager
+ @cards: List<MtgCard>
+ fetch_list(list_id: Int): List<MtgCard>
+order_list(): List<MtgCard>
+manage_list(list_id: Int): List<MtgCard>
Extends
Extends
DeckListManager
+order_list(): List<MtgCard>
FavoriteListManager
+order_list(): List<Mtg_Card>

I chose the template design pattern because of the similarity between deck lists and favorite lists. Both are collections of MtgCards but they need to be displayed differently. The fetching from the database and compiling of the lists are the same, only the order of the display of the lists is different. Cards from deck lists need to be displayed in ascending converted mana cost order and cards from favorite lists need to be displayed in alphabetical order, making the template design pattern a perfect choice. I implemented a super class *CardListManager* and two subclasses DeckListManager and FavoriteListManager. The super class has the implementation for getting the lists from the database and the subclasses have the implementation for sorting and ordering the cards in the list based on their needs. Using the template design pattern allows for easy future extension of functionality of both types of card lists.

7.What I Have Learned:

I have learned the complexity of designing even simple systems and the need for deliberate planning and design of these complex systems. My project was no feat of software engineering being a solo project, however, it still quickly became a complex system as I developed it. I tried to be deliberate and design and plan the system in a good way but I am still a novice at the design aspect and did not consider all the areas where problems could arise in my system. In a system more robust and complex than mine, i.e. business software, the need for good, deliberate design is even more important because there are so many people working on the code base. It was important in my solo semester project and I was the only one writing code, so, it is paramount when there are dozens or hundreds of people developing the software. In other words, good design is difficult and takes much practice to get good at but greatly increases the extensibility and maintainability of a software system. My final product was a fairly simple model of a Magic: The Gathering card collection but in reality there is much more going on. Starting with solid object oriented principles and design patterns will allow me to grow this project to encapsulate the more nuanced aspects of the game in my software. I have seen this most directly in my implementation of the template design pattern. I chose only to model the differences between sorting the two different types of Magic card lists. However, there are more differences that can be modeled and already having the template pattern in place allows for easy extension and addition of those nuances. This has led me to a greater understanding and respect for the ideas of encapsulation of what varies, polymorphism, and all of the object oriented paradigm.

Another aspect I have learned while taking this class and completing this project has been a greater understanding and respect for the Ruby on Rails object oriented framework. I used it during my internship last summer and that was the driving force in my choosing to use it for this project. Despite the potential for a golden hammer situation I am happy I chose to use this difficult to learn, robust object oriented framework. I was able to better grasp how Rails achieves the “magic” it does under the hood by learning about object oriented design patterns. As a result I feel I have a better understanding of Rails and of OO principles in general. I will note that I found some difficulty in translating some of the Java specifics from class to Ruby specifics in my project as the two languages implement some details differently. Ultimately, though, the ability to study OO principles in two different languages simultaneously was a benefit as I feel it has led to a deeper understanding of OO on my part.