**RESTful API Project**

Jake Stavrakas

Department of Informatics, USC Upstate

INFO U 798-01W

Dr. Walters

April 21st, 2024

**Abstract:**

In this interoperability project, I am creating a RESTful API. REST API stands for representational state transfer application protocol interface (redhat). This document presents a step-by-step process of the ideation of the API for the game DigiYuMon. This API will utilize 3 public APIs and format their information into one API in a JSON response format. Since there are three public APIs that I pulled from each had its documentation on their sites to help a user pul the information. I will pull the data from the three, and organize that data into a uniform format consisting of the monster name, monster ID, monster’s hp, and monster’s attack. Then create a way to pull the monsters at random for the game.

1. **Introduction**

My idea is to make my own card game by combining three cards into one called DigiYuMon. This is an amalgamation of three names Digimon, Yugioh, and Pokémon which are Japanese trading card games. Essentially, I want to pull from the three card APIs at random to create this new combination of a video game. This would be like a rock paper scissors game but with an additional twist to entertain fans of all three card games. If you pull a Yugioh card it will beat a Digimon card but a Yugioh card will lose to a Pokémon card. A Pokémon card will beat a Yugioh card, but a Pokémon card will lose to a Digimon card. Finally, a Digimon card will beat a Pokémon card but it will lose to a Yugioh card. Now if you pull two cards of the same type you play the card game the way the creators intended. Minus of course the additional cards such as energy in Pokémon or spell cards in Yugioh. You will just assume you can do anything you can with the card you pulled as if it had all the energy, it needed but still adhere to rules such as HP and attack power.

I want to begin with the game I know the most which is Pokémon and this game will differ from the original slightly. In the original game, you need a 60-card deck with three different types of cards a Pokémon card, an energy card, and a trainer card (pdf). The Pokémon cards evolve but you need the base Pokémon to place down first, a Charmander would evolve into a Charmeleon but if you pull a Charmeleon you cannot place it down without the Charmander first. It is like building a house you need the foundation before you build the second floor. The energy card is attached to a Pokémon and allows it to use different attacks. Some attacks require three energy cards and it could be a match of different types of energy cards(). In DigiYuMon we will only be using the Pokémon card and assuming that the energy of that Pokémon is at max. We will also forego the need to have a primary stage Pokémon, meaning you can play a Charmeleon without having a Charmander first. This will make it very interesting as you could pull a weak starter Pokémon or a coveted legendary Pokémon to battle, which will make it a game of chance.

Now onto the Yugioh rules which I am less familiar with but it starts like most card games with each player having a deck of 40 cards. A combination of monster, spell, and trap cards each with a variation to it and the goal is to get the other player's life points to zero(Yugioh). Each player can place up to 5 monsters on the field each with an attack score and a defense score (Yugioh). Your goal is to attack the monsters on the enemy side so they can't defend against their player's health points while defending your own player's health points with your monsters. In DigiYuMon I will forgo this and just have the higher monster score win if you draw two Yugioh cards. As there are over ten thousand Yugioh cards in total, I decided to narrow down the number pulled from the Yugioh API (Sharma & One37pm, 2022). To do that I narrowed down the information I pulled from the get API documentation to pull specifically the monster cards with the archetype blue eyes dragon. The code for that specific API call is found below.

api\_url = "https://db.ygoprodeck.com/api/v7/cardinfo.php?archetype=Blue-Eyes"

data = fetch\_data\_from\_api(api\_url)

1. **API Creation**

Creating my API I had taken inspiration from the movie API we had made in class but with some tweaks. I created a Python file titled main.py and began to import the dependencies we used in class specifically the flask, and then flask\_restful including Api, Resource, reqparse, abort, fields, marshal\_with, and SQLAlchemy. I then defined the application variables to start with version 1 of this API. Now let me explain what these things mean, Flask is a web framework written in Python that lets you develop web applications easily(Python Tutorial). The items in flask\_restful allow me to create the API, which allows that API to return HTTP error codes such as 404 and marshaling which converts an object in memory into a form that can be sent as a message (Maftei, n.d.).

**2.1 Monster Model**

Next, I created a model titled MonsterModel using the (db. Model) function from SQLAlchemy. SQLAlchemy is an open-source tool that allows the user to work with databases and map information in a database easily (SQLAlchemy, 2018). This allowed me to create a data frame with Monster ID, the monster name, the attack, and the hp or health points of the monster. I then created the database structure with db.create\_all from the flask tool which creates the tables I defined above and the database( Quickstart — Flask-SQLAlchemy Documentation, 2010).

Then I had to add the arguments to allow for a put request, a put request is where I define what is being parsed into my API. To do this, I had the following code

monster\_put\_args = reqparse.RequestParser()

monster\_put\_args.add\_argument(

    "name", type=str, help="Name of the monster is required", required=True

The monster\_put\_args = reqparse.ReqestParser() enables adding and parsing multiple arguments in the context of a single request. So I can add multiple arguments like the name one underneath which I made a string type meaning it must be a string, not an integer. I also mandated that the name must be included with the code required = True, a bit of Boolean logic at the end meaning if the name is unavailable you cannot parse the monster. I adjusted these for each field as this was the name one that required a string of letters, for the hp argument I would require the type to be int or integer depending on the game I chose.

To map the Monster. model to the columns I used the resource\_fields = {“id”: fields.String, etc}. after mapping these I had to set up the resource functions for CRUD an acronym for Create, Read, Upload, and Delete. To do this we created four functions one for each letter in CRUD. The Get is Read, Post is Create, Put or patch is Update, and Delete is Delete. We also included error codes with messaging such as 409 “Monster ID Taken” or 404 “Monster doesn’t exist, cannot update”.

Finally, I had to register the resource called Monster to the API with the code below Api.add\_resource(Monster, “/” + app\_version + “monster/<int:monster\_id>”). This registration allows me to Get and Post data meaning I can now retrieve and post information into the API. The final step is to to run the API body with the following code

# Run the API body

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

After running the API body in the main.py file it ran with no complications which can be seen in the code below.

C:\Users\jakes\anaconda3\Lib\site-packages\flask\_sqlalchemy\\_\_init\_\_.py:833:

warnings.warn(FSADeprecationWarning(

\* Debugger is active!

\* Debugger PIN: 999-698-078

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Running on <http://127.0.0.1:5000/> is the server I made via this main.py file.

* 1. **Integrating The 3 APIs**

The next step is to integrate the data from the three public APIs into mine. The Digimon information is formatted in a data frame; I had to change some parameters to get it to work with my Monster Model. To do this I mimicked the code from our movie ranking API assignment and tweaked it to import data from the public API into my own. This code can be seen in detail on my git.hub page under the file Digi.put. The put method is a request used in REST APIs that allows the user to create, and update a resource (Faber, 2023). I got the code to post to my API from the Digimon API, resulting in all the Digimon on the website being posted into my API. This totaled around 2000 Digimon and in my first version of this API, I kept it as is for testing purposes

Due to the large number of Pokemon and Yugioh compared to Digimon, I would need to limit the number of monsters pulled from each. Otherwise, this imbalance would lead to people playing the game with lackluster odds so I had to settle the score. I decided on 33 monsters from each game and I made this decision after the Pokemon Put phase. The easiest way to implement that is to make variables for starting\_id = 2776 and the num\_monsters = 33. In this case, 2776 was the number of Digimon in the system at the moment and the number of monsters I wanted to place into my API from the Pokemon source was 33. I made these variables for any user to be able to adjust as they please, they can increase or decrease the number of monsters and specify where they want them in the architecture. I then made another variable titled monsters\_id that combined these two so that the new monsters added in would start at 2776 and only put in 33 monsters to keep the number of monsters 33. I used the following code monster\_ids = [starting\_id + i for i in range(num\_monsters)] (Thompson, 2021). If data: is the code we used in class and it checks to see if data exists and isn’t null or none. If the entry has data it will return the Boolean value TRUE, and the code will allow the data to be pulled into my API. This is what I had in my testing stage to ensure the Pokemon could be added into the API which is why I did not worry about the number added from each of the three APIs until later on. Currently, I have 33 monsters from each group and I used this code to segment the Pokémon immediately after the 33 Digimon.

 starting\_id = 33

    num\_monsters = 33

    monster\_ids = [starting\_id + i for i in range(num\_monsters)]

Now the Yugioh put phase where I placed the data from the Yugioh API into my own. I started after the Pokémon put which led to adding in 33 monsters after the 3026 monsters from Pokémon and Digimon. I followed the same steps as the Pokemon put stage except for the health point stat I had to change. In Yugioh, the game doesn’t give monster cards a health point value so I set a standard 1000 across the board for each of these Yugioh. I did this by using the item.get the function and place the number 1000 after the comma. In the Pokemon put stage the code looks exactly for hp\_value except there is no comma or 1000 so it pulls the hp value associated with each monster.

hp\_value = item.get("hp", 1000)

After the testing phase where I ensured all the APIs could be pulled from and implemented into my API, I had to clean up my product.

**2.3 Driver Application Creation**

I combined the three get.py documents into one driver application titled driver.py at the recommendation of my professor. This consolidation helped keep the data accumulation segment of the API process more organized. To do this I combined the three get.py files with the main.py file to have all the code in one place. Before this driver file, the user had to go sequentially from file to file, starting with the main.py then the Digimon, followed by Pokémon, and finally the YuGiOh file. This new driver file makes it more organized and replaces the need for the user to know the exact order to run the Python code in. I also made each one have the variables for starting\_id, num\_monsters, and monster\_ids to ensure my API had only 99 monsters.

**2.4 Update.py**

From our movie example assignment in class, I created an update.py file to update the data in the monster model manually. This allows any user to change the monsters to their liking, for example, if the person wants to boost the attack score of their favorite monster they can. Similar to how I adjusted the hp codes to equal 1000 for the Yugioh monsters in the previous put phase. The code below illustrates the update.py file.

import requests

BASE = " http://127.0.0.1:5000/"

APP\_VERSION = "v1/"

response = requests.patch(BASE + APP\_VERSION + "monster/5", {"hp": 10077})

print(response.json())

Let me explain the code above, requests.patch I chose the base and current app version and when choosing the monster, I input the monster id to be 5 indicating the 5th monster will be the one whose stats are adjusted. The monster\_id can be chosen by selecting a number after the backslash on the monster which is 5 in this example. To change any of the variables like hp, attack, or defense you choose the one you want to edit in parenthesis with a quotation like this{“hp”:x}) where you input the number in the place of x. In this case, I made Achillesmon which is the fifth monster have 10077 hp to test this feature and the result can be seen below.

{'id': '5', 'name': 'Achillesmon', 'attack': 0, 'hp': 10077}

1. **Creating the get.py**

After successfully creating the Driver.py I had to create a get.py file where I could pull the data that I had stored, specifically the 99 monsters. Since I want this to be a game, I decided to create a random number generator to pull from one to ninety-nine monsters at random which is found here (Manjeet\_04, 2019).

# printing digits

letters = string.digits

a = "".join(random.choice(letters) for i in range(2))

print(a)

In this code segment, I introduced the variable 'a' to randomly select a two-digit string of digits, ranging from 01 to 99. This enables the random selection of monsters from my collection. While the original example used characters our implementation involves letters(Shepard, 2017). The 'for i in range' snippet functions similarly to a variable like 'x' in a traditional mathematical problem.

import requests

BASE = "http://127.0.0.1:5000/"

APP\_VERSION = "v1/"

response = requests.get(BASE + APP\_VERSION + "monster/" + a)

print(response.json())

This is pulling from my API whose BASE is <http://127.0.0.1:5000/>, the version is v1, and, the monster model which would pull a number from ‘a’ the random number generator. I defined each of these above the response requests.get() and then printed the response in a JSON format which would reveal a monster. JSON format stands for Java Script Object Notation and is a commonly used way to store and transport data over web servers (W3Schools, 2019). An example of what JSON looks like will be seen in the monsters chosen from the get.py file. This is an example of the results of my executing the get.py.

(base) C:\Users\jakes\Documents\DigiYuMon>C:/Users/jakes/anaconda3/python.exe c:/Users/jakes/Documents/DigiYuMon/get.py

27

{'id': '27', 'name': 'Agumon', 'attack': 0, 'hp': 0}

This is the poster child for Digimon named Agumon, similar to Pikachu for Pokemon which is a great first pull for me! Now to see what the second monster is and battle, after executing the get.py again I got the following response. (base) C:\Users\jakes\Documents\DigiYuMon>C:/Users/jakes/anaconda3/python.exe c:/Users/jakes/Documents/DigiYuMon/get.py

61

{‘id’: ‘61’, ‘name’:’Paladin of White Dragon’, ‘attack’:0,’hp’:1000}

So, in this case, I pulled the 61st monster the Paladin of White Dragon would beat Augumon as the Paladin is a Yugioh monster.

1. **Reality vs Expectations**

I did notice that for all the Digimon monsters the attack score and health point scores were either null or a zero which is what I expected. The Digimon card game does not have

I knew when gathering the data from the Yugioh set that their HP had to be set to a number as they have no native health points in their card game. I had however forgotten to add in the attack values with the item.get() function in my driver application. The code below is the updated code that would pull the attack value from the API. Attack = item.get(“atk”)

For the Pokemon the hp values were pulled and uploaded, as each Pokemon has a hp stat point. The attack stat was something that I had trouble implementing, each Pokemon has an attack with a name that might or might not do damage. For instance, Charmander which we referenced at the beginning of the project has one attack named Fire Spin. But the Flareon Pokemon has two moves quick attack with 10 damage with a chance to deal 20 more damage based on the flip of a coin. The other move is Fire Spin with 70 damage since there are multiple attacks with varying damage scores I could not successfully add them into the game. After showing friends and family my code I had the revelation that not everyone will know each monster’s franchise. So I should have put it into my Monster. Model a key for each franchise so that anyone would be able to discern a Digimon from a Pokemon.

Sources:

*Yu-Gi-Oh! TRADING CARD GAME | Official Website*. (n.d.). Yu-Gi-Oh! TRADING CARD GAME | Official Website; Konami. Retrieved April 21, 2024, from <https://www.yugioh-card.com/en/rulebook/>

*What is Flask Python - Python Tutorial*. (2021). Pythonbasics.org; Python. https://pythonbasics.org/what-is-flask-python/

Sharma, A., & One37pm. (2022, January 5). How Many Yu-Gi-Oh! Cards Are There? // ONE37pm. Www.one37pm.com. <https://www.one37pm.com/popular-culture/how-many-> yu-gi-oh-cards-are-there#:~:text=Konami%2C%20the%20creator%20of%20the

Maftei, A. (n.d.). *Marshalling data | RESTful Flask | Flask | Backend | Computer science*. Hyperskill. Retrieved April 21, 2024, from <https://hyperskill.org/learn/step/18204>

SQLAlchemy. (2018). SQLAlchemy - The Database Toolkit for Python. Sqlalchemy.org. <https://www.sqlalchemy.org/>

Faber , J. (2023, February 10). Understanding HTTP methods in REST API development. Www.torocloud.com. <https://www.torocloud.com/blog/understanding-http-methods-in-rest-api-development#:~:text=The%20PUT%20method%20is%20an>

Thompson, J. (2021, March 30). Python For Loop - For i in Range Example. FreeCodeCamp.org. <https://www.freecodecamp.org/news/python-for-loop-for-i-in-range-example/>

Shepard, E. (2017, November 6). ''.join(random.choice(chars) for \_ in range(size)). Stack Overflow. https://stackoverflow.com/questions/47142467/joinrandom-choicechars-for-in-rangesize.

W3Schools. (2019). What is JSON. W3schools.com. https://www.w3schools.com/whatis/whatis\_json.asp

Manjeet\_04. (2019, May 21). Python | Generate random string of given length. GeeksforGeeks. <https://www.geeksforgeeks.org/python-generate-random-string-of-given-length/>