Your README.md file should be minimally multiple paragraphs in length, and should explain what your project is, what each of the files you wrote for the project contains and does, and if you debated certain design choices, explaining why you made them. Ensure you allocate sufficient time and energy to writing a README.md that documents your project thoroughly. Be proud of it! A README.md in the neighborhood of 750 words is likely to be sufficient for describing your project and all aspects of its functionality. If unable to reach that threshold, that probably means your project is insufficiently complex.

# Overview

The goal of this project was to create a web-app for character creation of Dungeons and Dragons, based off of content from the SRD 5.1. Full legal disclaimer below:

“This work includes material taken from the System Reference Document 5.1 (“SRD 5.1”) by Wizards of

the Coast LLC and available at https://dnd.wizards.com/resources/systems-reference-document. The

SRD 5.1 is licensed under the Creative Commons Attribution 4.0 International License available at

<https://creativecommons.org/licenses/by/4.0/legalcode>.”

The original goal was (and still is) to make a character creator for my own RPG that is a heavily-modified version of D&D 5E. However, attempting to do this taught me (very quickly) about the importance of defining the scope of your project, and what the minimum-viable product is.

In the end, the version I am submitting lets you create a level-1 character, choosing the Name, Race, Class, and Background of the character. Currently the class-list is limited to either a Fighter or Wizard; however, you also choose any relevant class sub-features or spells. (The Fighter has a ‘Fighting Style’ to choose from, while the Wizard has spells to choose.)

IMPORTANT NOTE: While you can also create an account, in order to log in or out of the website, the account creation, login and logout functionality from the Week 9 Finance problem. I get zero credit for implementing this.

Being logged on allows you to save and/or load a character you have created. Saving it loads it from the cookie (or Flask “Session”) to the database, while loading it grabs a character from the database and passes the relevant features to the cookie (or Flask “Session”).

A major goal of this project was to include client-side AND server-side validation for all user-input.

# app.py

This is what runs the main website itself. The bulk of the work was done in the character-creation section. The character creation section (starting around line 300) more than anything else, is heavily intwined with the “character\_creator.html file” (found in the templates folder), as the goal was to stay on one single page during all of character creation. It also makes heavy use of the “rpg\_char\_create” class and to a lesser extent the “rpg\_char\_load” classes, two heavy-weight classes designed for character creation or loading existing characters (respectively.) The use of these classes allowed me to pass around a single variable (stored in the session) containing all the character data, as well as methods inside these classes for setting character attributes (including input-validation.)

Outside of the character creation section, app.py mostly exists as a handler for calling the many different custom classes and/or functions created for this project.

*Other features of particular note are the “/save\_button” vs “/save\_character” and the “delete\_button” vs “delete -> mention this somewhere in the html section*

# CreateDatabase.py

I’m sure somewhere out there, there is perfectly functional relational-database software I could use that gives me a nice GUI interface for managing a database. However, I am not using such software. All of my work on the database is done on commands in the console. Eventually I got bored of copying and pasting SQL commands whenever I would make a database change (especially as the database grew larger and larger) so I just made a python script to create the database for me.

The database is definitely over-engineered in terms of what is needed for the minimum-viable-product of the final project, but the database structure was created with the intention of potentially storing a large number of users, with a large number of characters built from a large number of options.

# gunicorn\_config.py

IMPORTANT NOTE: this file was taken as-is from elsewhere online, I get zero-credit for this.

This is config file, created according to instructions from: <https://developers.redhat.com/articles/2023/08/17/how-deploy-flask-application-python-gunicorn#the_application>

I get zero credit for creating this file, I simply followed the instructions for creating a Gunicorn config file so that I could host the site on https://www.digitalocean.com/

# helper\_customClasses.py

Mentioned in app.py, this contains two heavy-weight classes. The first, “rpg\_char\_create” is designed to hold choices for character creation – include custom methods of inputting information with input validation. Notably however, it does not do input validation for choosing class features or spells; that was complex enough to get their own functions.

The second class, “rpg\_char\_load” is designed to hold an existing character to view on the website. This class includes a custom method to copy features from the “rpg\_char\_create” class to itself. It also includes a method for saving or loading a character to the database.

# helper\_getFeatures.py

This class contains all of my functions for interacting with the list\_feature\_titles and list\_feature\_descriptions tables in the database. Some of the functions aren’t called in the main app.py, but existing to break down the work of a function into multiple steps. (For instance, I have one function to grab items from the SQL database, which then calls another function to format that text. The only reason for them to be separate functions is it was easier to create them as individual pieces and put them together – functionally they could be combined into one.)

The functions here could probably be broken down further – some of these take feature\_IDs as input as part of displaying items, while others have hard-coded feature\_ids for character creation. (While magic numbers aren’t ideal, this was deemed the best option for the minimum-viable-product for the CS50 Final Project. Reworking features is the next major task planned, but that task it outside of scope for the CS50 Final Project submission.)

Some of the functions here put together HTML code, where one of the app-routes can call this function and return it in JSON format to be put in the page. While that feels somewhat hacky, this HTML code includes results of SQL queries, resulting in at least some of the code needing to be back-end. So it just spits out large blocks of HTML code.

# helper\_getSpells.py

The basic idea of this file is the same as the “helper\_getFeatures.py” file above, except that this one interacts with the list\_spells table instead.

# helper\_loginRequired.py

IMPORTANT NOTE:

This contains the login\_required functions as given to us in the Week 9 Finance problem. I get zero credit for this.

# helper\_magicNumbers.py

Given the amount of instances where I used a specific class\_id as a magic number, or a few instances of the ability\_id of an ability, I made a special-class to store those “magic numbers” in; places where that “magic\_number” would be used, are instead replaced with items from this class.

NOTE: I did not do this for a few places where I used the “magic number” of a feature\_id, because those distinct feature\_id’s are not repeated much, if at all, throughout the project. It would probably be a good idea to do that anyways, but this project is large-enough as is, and I do have a deadline I am attempting to meet.

# helper\_newClassSpells.py

A bunch of code involving the spells belonging to different classes that I didn’t end up using at all, but I also didn’t quite feel like deleting.

# helper\_validateCharacter.py

Functions used by the “rpg\_char\_load” class to validate user-input when a character is loaded.

# “static” folder

Containing the barely-used style\_sheet.css, because just creating the backend and implementing this site took enough time as-is. We also took the “I\_heart\_validator” PNG (and html text) from the Week 9 Finance problem.

There are also a lot of CSV files where I created and modified data prior to importing it to the database.

# import\_text

Some python scripts I created to take my specifically-tailored text files, with strange idiosyncrasies from how I imported them from PDF (the stuff imported from PDF being from the SRD 5.1 on the Creative Commons), and converted them to useable CSVs for either modifying, or importing.

# NonCode

This folder contains a bunch of miscellaneous, non-code files I referenced throughout the creation of this project.