Dungeon Forge Database Design

Prepared for: Joseph Gradecki, Professor
Prepared by: Adam Ure, Student
October 29, 2023

OVERVIEW

This document outlines the database design for the Dungeon Forge application. Topics covered in this document include a description of the databases used within the creation of the system, the structure of each database, and the format of the contents within. To provide context on the system for which these databases have been design, a brief description of the Dungeon Forge project is also included.

Dungeon Forge Project

In the game of Dungeons and Dragons, one group member takes the role of dungeon master. The dungeon master is responsible for planning and preparing the adventures for the rest of the team members, playing the enemies and non-playable characters within the game, and adapting the plot to make the group's experience enjoyable. This is a taxing responsibility, forcing the dungeon master to spend days, weeks, or even months preparing for the adventure to come. To save the dungeon master's time and reduce the complexity of starting a new adventure, the dungeon forge project automatically generates adventures for groups of adventurers. These campaigns come with an environment, a plot, enemies, combat, intrigue, and sometimes even lead to a player character's demise. They are designed to be runnable out of the box, with the dungeon master simply needing to read ahead to know what happens for the adventurers prior to the next session. No mapping characters, defining plot twists, or creating scale replicas of the scene. The dungeon forge project generates the materials needed start and run a fun campaign the entire group can enjoy.

Databases

To support the application described above, I have deemed two databases necessary for the implementation of this project. The first is a reference database, storing links, images, and details about enemies and items. Though the generator should create characters for the individuals to interact with, the application will need some resources to populate less creative portions of these campaigns (i.e. stat blocks, images, etc). Additional details on this database will be provided later in this document. The second database is the archive of generated campaigns. This database is used to store generated campaigns and provide an easy method to lookup the campaign itself. A stretch goal for the project would be to convert the generated campaign into a PDF, and therefore store generated campaigns in an s3 bucket rather than a database. If, in the scope of the capstone project, we reach the point of generating the PDFs, this second database would be rendered obsolete and I would no longer support that data store.

REFERENCE CONTENT DATABASE

As described in the databases section above, the first database we will maintain is a list of available enemies the player characters may encounter. The plan is to create a MySQL database that will contain tables for monsters and items. The main benefit of creating a relational database like MySQL is that the database can handle relationships between table items. Currently, the database doesn't include any such relationships. However, the project can easily be expanded to include cyclical relationships between types of enemies (e.g. a goblin often fears

a bugbear so bugbears often use them as underlings). This type of cyclical relationship would allow our generation algorithm to add weights to encounters. Taking the example provided above, a weight for encounters would mean if the generator determined the adventurers should encounter a bugbear they would also likely encounter a few goblins during the same encounter. Another stretch goal would be to add references to abilities and items, allowing for more customizable characters during encounter generation. Overall, this future potential for relationships between enemies, items, and other future tables justifies my decision to implement a relational database above a document database.

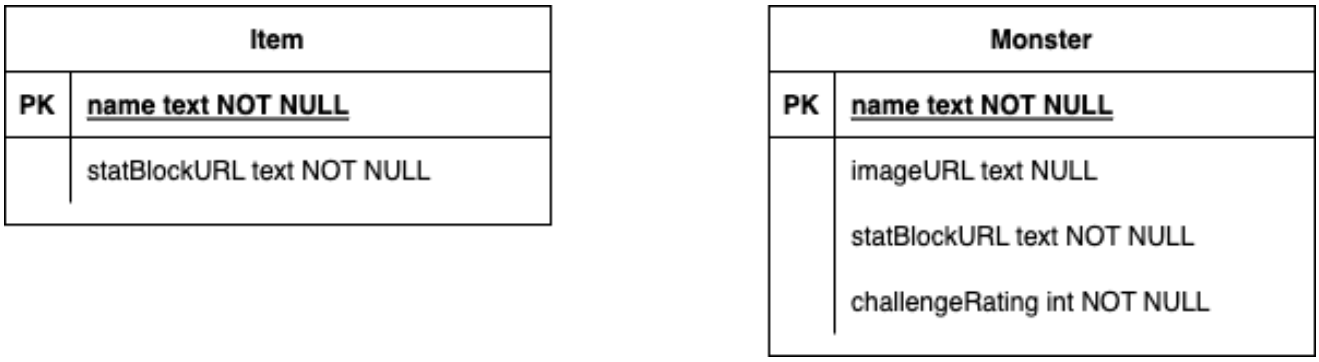

Figure 1. Reference Content Database Tables

The role of objects in this database is to provide references to be added to the final campaign document. Campaign events often include combat encounters, which require references to monster stat blocks for easy tracking. Similarly, players will often pick up items while traversing through a dungeon. These objects need to be clearly defined, and their abilities marked so the dungeon master can describe them to the players.

Monster Table

As seen in Figure 1, the table for monsters is extremely simple. This data structure would be easy to represent in a document database. Even so, the section above details the justification for using a SQL database instead of NoSQL.

The contents of this monster table include the name, which must be unique. This enables us a clear search path to fetch the statBlock information.

The stat block data is provided by an open source project called D&D 5e API (see https://github.com/5e-bits/5e-srd-api for details). This project is available through an MIT license, allowing me to leverage the API without any restrictions.

In some situations, a monster's image may not be available. In that situation, the server can either generate the image and create a new URL, or simply not display an image for the monster. These images are only used for UI purposes to make the content more visually appealing.

The final data point in the monster table is the challenge rating. This is used to validate a generated encounter. Depending on the player's expected levels at a certain encounter and the amount of enemies, an encounter may overwhelm the players and provide a negative experience. Therefore, the combined challenge rating of the entire encounter will be compared to the adventurer count and levels to determine if an encounter is acceptable for the campaign.

Item Table

Figure 1 also depicts the item table, which is just as simple as the monster table. The name is the primary key, providing easy information lookup when encountered through a generated campaign. The stat block data is also provided by the same D&D 5e API. This generated information will allow the generated campaign to include details about the items seen throughout the campaign.

Review

This reference content database is designed to be simple at the beginning, enabling us to implement the MVP of the Dungeon Forge application. As additional features are implemented into the campaign generator, this database will become more complex to account for game lore. The generator will, in the future, account for those complexities and provide more robust campaigns.

CAMPAIGN DATABASE

The campaign database is extremely straightforward, storing the generated campaigns for future access. The format of each campaign is somewhat flexible, making a relational database unrealistic for this database. Therefore, I will leverage Dynamo DB in AWS, which is a NoSQL solution to hold the data for each campaign. The campaign document will be formatted like the pseudo-JSON in Figure 2.

Campaigns are complex documents which hold various pieces of information, and can be formatted differently depending on the event. Therefore, I abstracted the specifics of each section into easily parsable html scripts. When the web client fetches the campaign, they will receive a JSON document with the campaign broken down into different pieces. A campaign has a few different key components: the environment, the background, and the adventures.

The environment represents the world in which the campaign takes place. The campaign should describe in detail where the players are, what historical event they are living through, and what the environment is like, so they can adequately prepare themselves for what lies ahead. Campaign backgrounds include what caused the players to arrive in the location where the adventure starts, and what their end goal seems to be.

The most crucial part of a campaign is the adventures themselves. Most campaigns include multiple story arcs, where the adventurers can encounter enemies, traverse dungeons, slay dragons, and help the common-folk. The number of story arcs depends on the number of players and how long they want the campaign to last. This type of information is provided to the generator through the campaign generation form. As a stretch goal, the number of story arcs and adventures in the campaign will change according to the inputs provided by the user. The MVP, however, creates one single story arc and events to satisfy that story arc.

When the front-end receives the campaign entry, the client will first display the metadata (title, number of players, levels, overview, etc), and then display the content of the campaign in the following order:

1. Campaign Setting
2. Adventure Hook
3. Adventures

4. Resolution

```
{
    _id: string,
    name: string,
    image: string?,
    setting: {
        _id: string,
        name: string,
        map: string?,
        html: string
    },
    adventureHook: {
        html: string
    },
    adventures: [
        {
            title: string,
            html: string,
            events: [
                {
                    html: string
                }
            ]
        }
    ],
    resolution: {
        html: string
    },
    html: string,
    numPlayers: int,
    startLevel: int,
    endLevel: int
}
```

Figure 2. Campaign Document JSON

Review

It is important to note that a stretch goal of the Dungeon Forge application is to make the above database table irrelevant, by providing a unique identifier for the campaign upon creation, but returning a PDF file that is stored in an S3 bucket, instead of the raw JSON. This simplifies the command, ensures accurate formatting, and enables the direct exporting and printing of final campaigns.