TTRPG Manager

Summary

The idea of this project is to create a user-friendly interface where players and DMs of Dungeons and Dragons alike can manage and overview their character sheets, their worlds' contents and their campaigns.

Context

The usual game of D&D(Dungeons and Dragons) is filled with many sheets of paper containing important information for the game. In an immersive and long campaign, it can easily become difficult to find any information amidst all the lore papers, character sheets, and other miscellaneous information that could be needed. To solve this problem and organise information in a cloud environment, we came up with the idea of creating a digital solution for preparing, storing and organising information for an easier and better user experience. We also plan on adding some tools to use during the campaign if possible.

Goals

Goals

Non-Goals

- Maps: We do not want to make a map manager. There are many other, more suited applications for this that work both online and offline, as well as other non-digital solutions.
- Combat: along with maps, it also isn't our job to simulate combat, player/creature health, or anything else of the sort. We want to focus solely on keeping information that is not updated too often, such as stats, health and notes.

Design

Design

The backend will be written as a REST api, featuring various endpoints. Overall, the endpoints should allow for the following:

- Players and DMs can create and delete accounts
- Players can perform CRUD operations on a limited amount of character sheets
- DMs can perform CRUD operations on campaign sheets
- DMs can perform CRUD operations on private campaign notes in a directory structure
- Optionally, there should also be a way to upload images

Endpoints

Endpoints

Database / Backend

Design

For the database, we will be using postgresql, as it is the system we are most familiar with. It is relatively simple to use with plenty of resources on how to containerize it. Performance isn't really a concern for us, as we do not expect a large user base or many servers.

Docker Containers

The database and the backend will each run as two separate, monolithically structured containers. The backend will act as the connection between the frontend and the database.

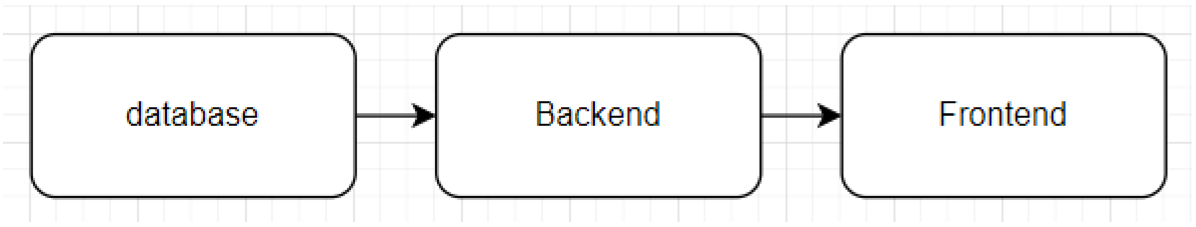
Frontend

Design

Consistency and user-friendliness is our top priority. Navigating through our application should be easy and straightforward. We strive for functionality over aesthetics, but that doesn't mean it is completely forgotten. We will use react with typescript to write our frontend, as it is simple and fast to set up. Additionally, there are many resources on how to use and dockerize react applications.

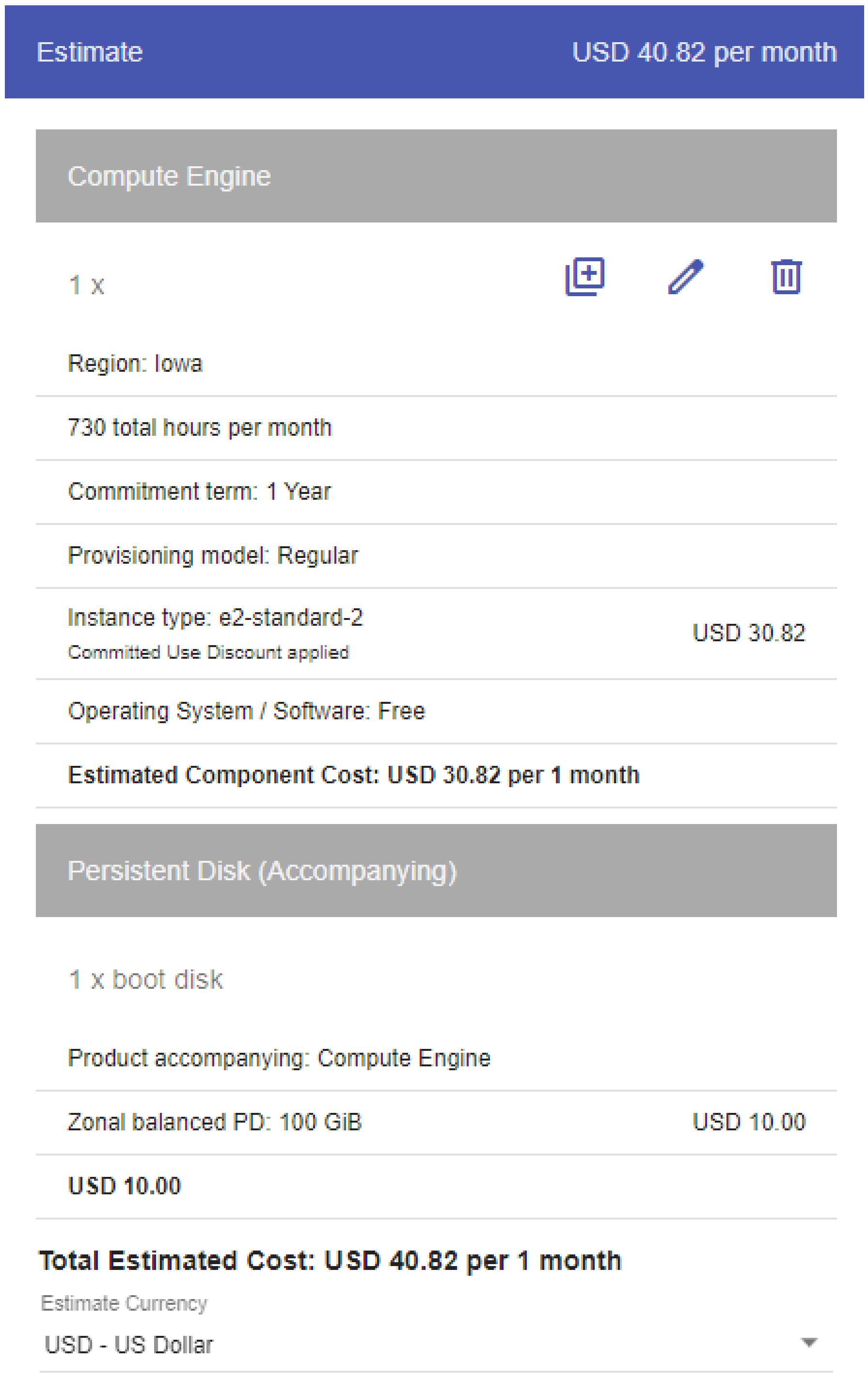
Docker Container Structure

We will be using a monolithic structure for the frontend, where an instance of the program is run on a single docker instance. We will do this because we do not need any of the benefits of other systems and value the simplicity of a single container structure.
We will also use a monolithic container structure for the backend and the database respectively. We are doing this because we do not need this to be particularly scalable, as well as wanting it to be simple. Additionally, we do not need to make it expandable in any real way.



Costs

Using the price calculator services of Google cloud, we can estimate a monthly cost of around 40 USD per month. This estimate is low, as we do not expect to have many users. In fact, we expect at most 100 concurrent users, with each one only taking at most one or two kilobytes of persistent memory. Due to this lack of need for any real performance, we simply chose very low values, which makes it cost only around 40 USD, or 35 CHF, to run on a Google Cloud server.



Estimate — USD 40.82 per month

Compute Engine

1 x

Region: Iowa	
730 total hours per month	
Commitment term: 1 Year	
Provisioning model: Regular	
Instance type: e2-standard-2 Committed Use Discount applied	USD 30.82
Operating System / Software: Free	
Estimated Component Cost: USD 30.82 per 1 month	

Persistent Disk (Accompanying)

1 x boot disk

Product accompanying: Compute Engine	
Zonal balanced PD: 100 GiB	USD 10.00
USD 10.00	

Total Estimated Cost: USD 40.82 per 1 month

Estimate Currency

USD - US Dollar

Implementation

Starting out, we divided the project as follows: someone would be responsible for the frontend and most of the documentation, and the other person would be responsible for the backend and the database. After discussing and writing down some of the basic planning docs, we got to work.
We started by just making the frontend and backend, without any containers or anything else, since we didn't know a lot about how to build something from the ground up in a container. In hindsight, this wasn't a very good idea.
By week eight, we were done with the frontend and backend, so we started working on docker and the database. This was more difficult than we had anticipated, and it cost us a lot of time. We would have been prepared for this situation and spent some time working on the project in our freetime, but this was difficult because we had to write the BMS finals for maths and French, as well as the fact that we had 9 weeks instead of 10 to finish the module. Additionally, we had an üK during this time, which made it hard to work during some weeks of the module.
We still did some work at home, mostly to try and fix the frontend and finish the backend completely. It worked with the frontend , but not the backend. By the 9th day, we didn't have the things done that we needed; all we had was a frontend that couldn't connect with the backend yet, and a backend/database that mostly worked, but still had some problems that needed to be worked out. We tried to use docker for both the frontend and backend, but it required some restructuring that we couldn't do in time. Due to this, the project is now somewhat weirdly structured, and we have some dysfunctional docker-related files in out folders.
On the last day, we did what we could to finish everything and make the containers work and communicate properly, but we didn't manage. We were only able to get the parts to work separately, and even then we're unsure if they work properly as docker containers. The reason why we failed wasn't a technical issue, but instead a lack of time to finish the project and get it to work the way it needed to. It would have been possible if we'd had even one more week, or if we hadn't had any final exams. In terms of what exactly we didn't finish: we couldn't get the frontend to work as a container. The backend and database worked in containers and were able to communicate

Testing

For testing, we used postman. This means that we tested the backend, and by extension, the functionality of the database. We didn't do any testing on the backend, since we were unable to get it to work properly in a container.
The postman tests are visible in the github repository.

The current state

The current state of the application is as follows:
The backend is done. The database is done. The backend and database can communicate. The backend is tested partly. The frontend is mostly done, but does not work fully with the backend and is not dockerized. The backend is also not dockerized.