Patrick Flood
03/24/23

Database Structures

MongoDB is my choice for GamerLand because it is a document-based NoSQL database. It will be flexible and scalable since it uses JSON-like documents. This will allow for efficient managing of unstructured data, like user-generated content, which is a part of the core of GamerLand (reviews, user profile information, etc.). MongoDB makes sure that the application can handle any high traffic and perform well as the data will hopefully and eventually grow.

Based on the requirements of the GamerLand web app, the database will contain four collections according to the current MVP. These will be Users, Games, Reviews, and Ratings. The User collection will store all the user information, including the user's name, email address, password, and any other relevant details. The Games collection will store all the game information, including the game title, description, console, release date, developer, publisher, genre, and potentially more details as I see what can be deemed relevant. The Reviews collection will store all the reviews posted by users for a particular game. It will include the user's ID, the game's ID, the actual review text, the user's rating, and other details. The Ratings collection will store all the ratings posted by users for a particular game. This collection will have the user's ID, the game's ID, the rating, and again, any other relevant details. I have also included two stretch features that I will not go into detail on quite yet as they are not considered MVP.

As for the subsequent JSON structure, the collections will be as follows:

1. **Users (MVP)**

```
{
  "userId": "string",
  "firstName": "string",
  "lastName": "string",
  "email": "string",
  "password": "string",
  "gender": "string",
  "country": "string",
  "createdAt": "string",
  "updatedAt": "string"
}
```

2. **Games (MVP)**

```
{
  "gameId": "string",
  "title": "string",
  "description": "string",
  "releaseDate": "string",
```

```
  "developer": "string",
  "publisher": "string",
  "genre": "string",
  "console": ["string"],
  "imageUrl": "string",
  "createdAt": "string",
  "updatedAt": "string"
}
```

3. **Reviews (MVP)**

```
{
  "reviewId": "string",
  "userId": "string",
  "gameId": "string",
  "text": "string",
  "rating": "number",
  "createdAt": "string",
  "updatedAt": "string"
}
```

4. **Ratings (MVP)**

```
{
  "ratingId": "string",
  "userId": "string",
  "gameId": "string",
  "rating": "number",
  "createdAt": "string",
  "updatedAt": "string"
}
```

OR it could possibly be this (depending on if I ultimately choose to do 1-10 scale for ratings rather than a 1-5 “star review” system):

```
{
 "userid": "string",
 "gameid": "string",
 "rating": {
   "value": "integer",
   "max": 10,
   "min": 1
 },
 "comment": "string"
}
```

5. **Discussions (Stretch Feature)**

```
{
"discussionId": "string",
"userId": "string",
"gameId": "string",
"title": "string",
"text": "string",
"createdAt": "string",
"updatedAt": "string"
}
```

6. **Gamer Search (Stretch Feature)**

```
{
"searchId": "string",
"userId": "string",
"searchText": "string",
"createdAt": "string",
"updatedAt": "string"
}
```

I will note that each document in the database represents an individual video game. Since a video game does not have any direct relationship with other video games, there is no need for me to have any document be embedded in multiple other documents. Each video game document can exist completely independently from each other in my database. No need for any connection to other video game documents. I will also note that I plan on using the RAWG.io API and OpenVGDB for the game images, descriptions, and other deemed necessary data. Their implementation seems necessary and even vital to making this web app and its proper database happen.