Cora Ners – Week 3 – Re-Design Database

Let's Multiply!

Proposed Database: MongoDB

Proposed Containers: See included screenshots for field names/types

UserData

GameData

Why document based?

I went back and forth on the type of database to use here, because I could have used a relational database structure. Had I gone in that direction, I would have a table with all the possible "problems" (i.e. 3x5, 3x6, 3x7 etc.) and each associated with an ID. Since there would be a many-to-many relationship between the problems, answers, dates, etc., then the joining table would simply just be huge – and grow as the student did more and more questions.

I decided on using a document-based structure. Though I'm most familiar with Azure Cosmos DB, I'll be using MongoDB for my final project. I've only used it once before.

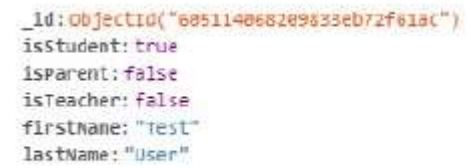
I think this is the best choice for the data for a few reasons. There will be a soft reference from the GameData table to the UserData table based on ObjectID. I think this will work at least. If the user gets deleted from the database for whatever reason, it won't break anything from either container. Since the user would be deleted, we wouldn't query the GameData container with that userID anyways. The JSON data that will be stored will be one entire session for that user. The session would be considered 'done' (i.e. new document created) when the Logout button is clicked. Because the problems that are shown on the screen are random, we'll only be storing the ones that come up and are answered, in the database. When we fetch the history, we can select by user ID and order by date (in theory they should be added to the database in order, but this will be a double-check just in case something gets swapped around). Because document-based databases can contain a ton of records and not

lag typically (because we're not being tightly coupled to other tables) this should be a non-issue for performance reasons.

In my head, this all works. However, it's already keeping me up at night thinking about different ways to implement my application and if the data or schema will change. Another reason I am liking the document-based design is because if I do decide to make some changes during the implementation by adding some new fields, and there's data stored in the database already, any additional fields added to new documents won't break the old documents.

Here are the proposed containers:

UserData



GameData

If I'm being 100% honest, I am fully expecting at least the GameData schema to change within the coming weeks. I was divided about whether to split out the generalized History data list (i.e. date, number of questions attempted, number of questions correct, and a link to the expanded data with each of the questions and answers) or to put it all together as in the GameData container above.

Updates for MVP/Re-Design

Though the majority of this will be the same, there will be a few minor updates. First, there will no longer be a 'challenge' mode sent from the client-side to the server and to the database, unless that Stretch Goal is achieved (for MVP it will simply be practice or evaluation modes). Therefore, the GameData JSON screenshot above has been updated to reflect this (originally it was set to challenge mode). If the stretch feature is implemented, the 'challenge' mode will be placed back in the document as a valid mode.

In addition, I previously stated "The session would be considered 'done' (i.e. new document created) when the Logout button is clicked.". Between the logout button and the 'Quit and Go Home' link (removed for MVP), there needs to be another way for the game data to persist. My thoughts are to persist as soon as a user completes a category, which is a maximum of 12 questions. As soon as the last question in that category is answered, the data would save to the database so it could be retrieved for historical purposes later. I really don't think the 'Quit and Go Home' link stretch feature will be implemented because it's a lot of UI work for little return. There would not be any db changes if that does get implemented.

I am again sure that changes will need to be made when development progresses. At that point, the developer (me) will ask the Product Manager (me) questions and get clarification from the Product Owner (me), one of us will document any changes in the stories in Github. I hope this last part made you laugh a little bit – professionally speaking of course.