Test Creator System Design

Nathan Gurrin-Smith

May 23, 2023

1 Introduction

This is a program that will be used to generate practice tests from a list of questions. It aims to help improve learning by encouraging frequent, interleaved testing.

2 Use Cases

- Login
- Add, view, edit, and delete presets
- Add, view, edit, and delete questions
- Add, view, edit, and delete categories
- Generate practice tests

3 Technology

The framework being used is PERN (Postgres, Express, React, Node).

3.1 Server (Node + Express)

The main `server.js` file contains the base routes along with the middleware. Additionally, we have the following groups of modules:

- Routes: used to direct requests to the right controller
- Controllers: manages which services are used and client responses
- Services: handles business logic of each task
- Models: handles interaction with the database

Routes

We are using the `/api/` style. The routes file will **only** contain the HTTP method and which controller function it will use. For example:

```
router.METHOD("/", Controller.Function)
```

Controllers

The controllers manage which services are activated when. It also handles client responses. These files are the **only** place where statements of the form `res.RESPONSE` can be put.

Services

These modules handle the underlying business logic of each task the controller asks to be completed. These files are the **only** place where calls to the models can be put.

Models

These modules handle individual interactions with the database. These files are the **only** place where calls to the database can be put.

3.2 Client (React + Next.js)

There are two main pages to the application. The first is the index page, at index.js, where users can create an account and sign in. The second is the user page, at [username].js, which is where all the user specific functions are found (see use cases above).

3.3 Data (PostgreSQL)

Naming Conventions

We will use the following naming conventions for our data model

- Tables and columns will be in PascalCase.
- Tables and columns will be singular.
- Junction tables will be named JunctionTable1Table2

Note: PostgreSQL automatically forces lowercase internally, but we'll always reference by PascalCase.

Note: Because of the above note, we must reference all columns with lowercase in our javascript files.

Data Model

We will have the following data model:

UserAccount		
UserAccountID	Username	Password

ClientAuthentication				
ClientID	LastLogin	RefreshToken	RefreshTokenExpiry	UserAccountID (FK)

Preset					
PresetID	Name	Preamble	Sep	Postamble	UserAccountID(FK)

Collection	
CollectionID	Name

SubCollection		
SubCollectionID	Name	CollectionID(FK)

Question			
QuestionID	Name	Content	Source

JunctionUserAccountCollection	
UserAccountID(FK)	CollectionID(FK)

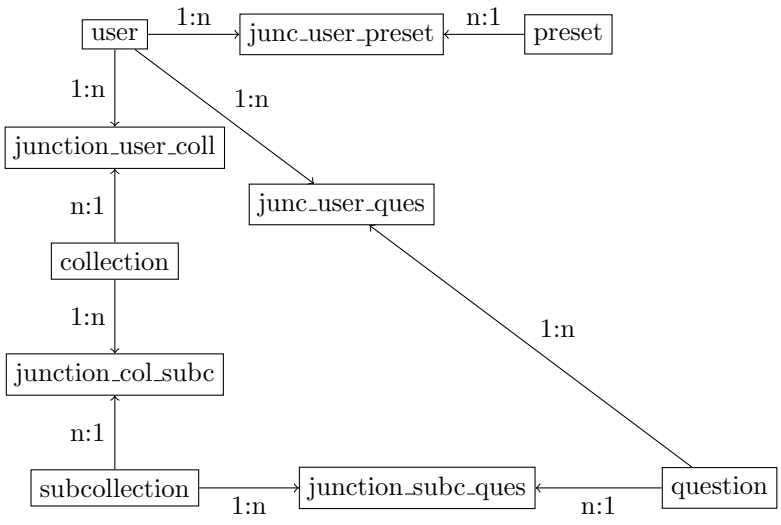
JunctionUserAccountPreset	
UserAccountID(FK)	PresetID(FK)

JunctionSubCollectionQuestion	
SubCollectionID(FK)	QuestionID(FK)

JunctionCollectionSubCollection	
CollectionID(FK)	SubCollectionID(FK)

JunctionUserAccountQuestion		
UserAccountID(FK)	QuestionID(FK)	LastReviewed

Which are related via,



Orphans: Some of the properties above require a parent. These are,

- A Collection requires at least one parent UserAccount,
- A SubCollection requires a parent Collection,
- A Question requires a parent UserAccount and a parent SubCollection
- A Preset requires a parent UserAccount

Thus, when we delete a parent, we not only need to delete entries in the junction table, but also the entries in the child tables since, most frequently, the foreign keys will be stored in the junction table and not the child table. The `cleanX` functions in the `service` modules handle this.

3.4 Jobs

There is a single job that runs once every day which removes all unspecified user accounts from the database. These user accounts are hardcoded into the application and can be added by the developer. The unspecified accounts are removed to keep data costs low.