Design Document

Perry Wu

CS 6360.501

Database Design

Design of the System:

The overall design of the system is based on PostgreSQL, an Express.js middleware, and an Angular.js front-end. There are many assumptions and design decisions that have led to the choice of frameworks.

Angular.js is a popular JavaScript front-end framework. This would serve as the portal in which users can interact with the library application. This front end would execute queries into the Express.js middleware, which serves an Application programming interface to hit the Postgres database.

The Postgres database for the most part retains all of the original schema constraints, with some additions. First, **Borrower** only has the Primary key ID, name, and SSN. There are other tables, namely **bname** with the first, last name, and email, that references the primary key borrower. This enables flexibility if eventually **borrower** can have multiple names. Further, there is an **address** table that has the first line, city, state and zip that references the **borrower** primary key. This enables further reuse along with the possibility to reuse the address table for another table in the future.

In order to adhere to the requirements, some constraints were added. Most notably the **SSN** is required to be unique. This allows us to throw an error when creating a borrower with the same SSN. This fulfills the requirement of not allowing a single borrower have two different **card\_id**’s. Fines have the loan paid amount as a postgres “money” data type, in order to better facility the translation of data types when submitting and calculating fines. Further, a particularly special constraint within book\_loan is the custom constraint using ts\_range. This ensures that a book loaned out cannot be loaned again by checking the differences in the check in and check out date. This allows us to have much more robust validations on the database side, so we would have to do less coding on the front end.

Another notable assumption and design choice are that currently, the front-end houses a lot pre-made queries in order to do front-end validation. This includes the check for 3 or more books being checked out by the same borrower. This check is done through a query, with the angular front-end having the logic on whether or not to proceed or deny the request. In preliminary research it seems that it is very difficult to implement some constraints that can otherwise be done in simpler queries in Postgres. The same is true for other validations, such as marking a fine as paid, where the front-end does not allow the user to mark a fine as paid if it hasn’t been checked in.

In an attempt to make things scalable, an Express.js middleware was used in the project. This potentially allows the Angular front-end to have abstracted queries and gets, where we put all the query logic in the middleware and none of it is exposed to the front-end. In reality and in the interest of time, however, building an Application Programming Interface to do all this is unnecessary and overkill for what the project needs, and thus many of the SQL queries and functions reside within the front-end instead.

An interesting design choice was made to use Docker to run the postgres database. This enables us to solve the problem of having local machines be different from one another given the same codebase. Docker enables us the ability to build once, and run everywhere with the same exact configuration every time, as opposed to manually building the database and suffering through different version changes and discrepancies when it comes to querying and modify data through the database. In Docker, the sql is loaded with all the pre-made SQL files to perform initial insertion and deletion. It is through this that we can potentially have many people working on the same project without the fear of the “works on my machine” syndrome.

In the dockerized database, the image is first pulled from a central repository based on the version we specify, and then the script automatically loads the necessary SQL to be ran inside of the initialization phase of the database. This guarantees the same database all the time, every time.

In sum, all these design choices allow us to potentially scale up the application in the future with multiple people working on the same project. Many database design practices were also implemented from the range of foreign keys to reference other tables, to customized constraints to check before inserting a row or checking out a book. These design choices allow us to achieve the requirements of the project.