Design Document

-Niranjan Ravichandran

1.1 Overview

The System Architecture used for the Library system in this project is a popular full-stack web application development stack known as the PERN stack. The PERN stack consists of four main components, PostgreSQL, Express.js, React and Node.js.

There are three levels in the PERN stack architecture. Starting from the client, the first level is the front-end which is served by react js as a Web GUI to make changes to the library database. The Second level is the backend, where express js is used with node js to route various requests from the front-end for database access through the help of restful API endpoints. The third level is the PostgreSQL database server where the queries sent by express js are executed.

1.2 System Architecture and Design Decisions

In this project, the front-end is served at localhost port 3000 and the API calls are made to the express js backend server which is hosted on localhost port 5000, and the express js server stores the database credentials and uses pg package provided by npm to connect to the postgreSQL database. Search values and Borrower IDs are sent to backend through the API call url, whereas the list of books to be checked out/in, borrowers to mark paid and information for creating a new borrower are all sent to the backend as JSON objects and the backend always sends response back as json as well.

The check for book availability when checking out is done in the front-end as the information regarding its availability is already queried during search and would be redundant to query the server again. The check for how many books the borrower already has checked out when checking out books is also done in the front-end to provide the user with an user error message if the number of books being loaned will be over the limit. The decision to create a separate endpoint for book search by ISBN was made as it looks for an exact match whereas the regular search only looks for a substring.

In order to automatically increment keys for new records for the borrower, it was not viable to use SERIAL in database as the borrower ID was a string containing “ID” at the start so instead the system retrieves the current max borrower ID in the table and increments the numeric part of the string and used it as the borrower ID for the new borrower to be inserted.

When making fine payments, the fines for the loans already paid are by default filtered out of the list through the query

1.3 API Endpoints and their functionality

1.3.1 Search and Checkout Books:

GET "/findBook/:searchTerm"

- A request is sent to this endpoint when the user searches for a book in Check-Out section of the GUI
- This method queries the database for ISBN, title, authors of the book and its availability and returns the result

GET "/findBookISBN/:searchTerm"

- A request is sent to this endpoint when the user checks out by ISBN
- This method queries for the book and its availability by looking for an exact match

GET "/checkExistingLoans/:card_id

- This request is sent when checking out and returns the number of books already checked out for the given borrower ID

GET "/checkBorrower/:card_id"

- This method is used to check if the given borrower ID exists in the database i.e is a valid ID and returns a boolean

PUT "/checkOutBooks/:card_id"

- A request is sent to this endpoint when the user checks-out a single book or multiple books after entering borrower ID either through Check-Out or Check-Out by ISBN section of the GUI
- This method sends queries to the database to create new BOOK_LOANS records to be created for the books checked out

1.3.2 Check-in Books:

GET "/findBookLoans/:searchTerm"

- A request is sent to this method when the library user searches for a borrower ID, Name or Book ISBN to check-In in the GUI
- This method queries the database and returns loan ID, ISBN, Borrower ID, Name, date out and due date for loans that have not already been checked in

PUT "/checkInBooks"

- A request is sent to this method when check-in button is pressed on the GUI
- This method Updates the BOOK_LOANS tuple with the current date for date_in for the books checked in

1.3.3 Create Borrower

GET "/checkBorrowerSsn/:ssn"

- A request is sent when the form to create a new borrower is submitted in the GUI
- This method queries the database for any existing records with the ssn provided in the borrowers table and returns a boolean

POST "/borrower"

- This request is sent once the form is submitted in the GUI and checked for duplicate SSN
- This method gets the highest borrower ID and and increments it by 1 and uses it along with other information received through the form to create a new Borrower tuple.

1.3.4 Update and Make Fine Payments:

PUT "/updateFines"

- This request is sent when the Update Fines button is pressed in the GUI
- This method handle all the fine logic from updating existing fines based on date to create new records for fines for loans past due

GET "/getFinesFor"

- This request is sent when searched for a given Borrower ID in the pay fines tab in the GUI

- This method queries the database returns the card id, borrower name and total fine amount owed by the borrower

GET “/getOutFines”

- This request is sent when the pay fines tab is loaded on the GUI
- This method queries and returns the card id, borrower name and total fine for all borrowers with outstanding fines

GET “/allBooksReturned/:card_id”

- This request is sent when ‘Mark paid’ is submitted to check whether the borrower has returned all the books they had checked out by querying and returning a boolean

PUT “/markPaid”

- This method updates all the fine records for all the loans of each borrower and sets the paid attribute to true

1.4 Assumptions

- When searching for books for check-out, the search matches substrings of ISBN, title or Author names and is case insensitive
- Only display books not already checked in by default for check-ins
- The system will have less that 1,000,000 borrowers as the highest possible ID for borrower is ID999999
- When paid total amount is paid and not a partial amount
- Phone Number can be null for borrower

1.5 Additional Information

- The **CS6360_Backend_server/book_parser.py** in the backend server code is used to create insert statements in books.sql to import data from books.csv to Book, Book_authors and Authors table.
- Similarly, **CS6360_Backend_server/borrower.py** is used to create insert statements in borrowers.sql to import data from borrowers.csv to borrower table
- The **CS6360_Backend_server/database.sql** consists the statements to create the Library schema and its tables in postgreSQL