Final Design Stack

After our team had finalized the design for our project, we began to lay out how the full stack would be connected from the front-end to the backend. Our user interface (UI) is made with React.js, which provides the layout and output of our library system. The web server that we paired with React.js was Node.js. In order to smooth out communication between the server and the database, we used the object-relational mapper (ORM) Prisma to maintain the constraints of the database. We employed PostgreSQL as the query language for the database. Redux was implemented to perform state management on the client side. The programming language R was then the final element of our project. We then cleaned the given .csv file that contained existing information to be used in the library management system with R and JavaScript; specifically, R allowed us to remove duplicates easily.

Through the use of React.js, JavaScript, and CSS, our front-end website is a simple and easy to use tool for the librarian to navigate through the books database. React reduced the design complexity of the application, so more resources could then be spent on user interface design. The data is split into tables, such as a fines table, so that the librarian can readily perform tasks requiring specific information without needing to sift through an overwhelming amount of data. Navigation among the various tables is made seamless through the use of buttons for each page. When needing to check on an overdue book, check for a specific title, or any fines owed, there is no hassle when it comes to a librarian using our application.

The server environment that our group chose was Node.js. The purpose of using Node.js was to act as an in-between from the front-end to the backend. From the client, or the front-end, requests are made from our website whether that is to update the book loans, fines, or borrowers’ tables. Then, Node.js receives the request from the client-side and processes it before it serves the request to the database on the backend. Once the database information is fetched or modified, it is then sent back to the server. The last stop in this process is that the front-end, the website, is now updated with the updated data. Node.js works on the concepts of threads by allowing multiple threads to access the front-end while a single process is running. This allows concurrent requests to take place while the server is up and running.

When our group first met, we discussed if we should use an ORM in order to preserve the database constrains. Prisma was the tool that our team had elected to go with. Prisma provides a connecting framework between the server and the database. With the use of Prisma, understanding the structure and constrains of the database was made straightforward. This is accomplished through the schema language of Prisma, a simple solution to ensuring database relations. This reduced complexity in database management made it an attractive choice for our project.

For state management, our team chose to use Redux which will keep track of the changes made throughout the runtime of the client side. When running, Redux stores the application in a container and each component of the application can access any part that is stored in the container. Redux allowed our team to keep track of the changes of state that the application goes through so that tracing problems is made simple. Finally, we chose to implement Redux because it allowed us to maintain the data across our application with multiple components while each component was independent on its own.

In order to clean the data file given for the project, our team used R programming language and JavaScript to clean the data. R allowed us to view the .csv file in its entirety. R allowed us to clear empty cells and data that was unnecessary. Once our data was cleaned, we were able to export a new .csv file which was cleaned and had all the necessary data for our project. We also used a JavaScript application in order to go to the different tables that was needed.

Overall, for the entirety of the project, each of the above elements listed helped our team to implement this project. React.js made making the front-end of our application look and feel like a real library-management-system. What our application looks like was made simple by the use of React.js. To communicate with our desired server, we chose Node.js which simplified the connection between our front-end to the database. Regarding handling the database-side of things, Prisma made handling the constraints of our database simple by the use of its schema language. The schema language, provided by Prisma, reduced the difficulty of understanding the relations that were in our database. Lastly, Redux handled our state management which kept hold of our data for a certain state. Redux also handled any updates made to the data and changed states whenever necessary. All these elements helped our team to produce a basic but functional library-management-system.