After our team had finalized the design for our project, we began to lay out how the full stack would be connected from front-end to back-end. Our user interface (UI) is made with React.js, which provides the layout of our library system. The web framework 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. 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 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 easily to remove duplicates.

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 other aspects user interface. 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 for the user.

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 back-end. From the client, requests are made, whether that is to update the book loans, fines, or borrowers' tables. Then, Node.js receives the request from the client and processes it before it serves the request to the database on the back-end. Once the database information is fetched or modified, it is then sent back to the server. The final step now is the processed data being updated on the front-end. A major benefit to using Node.js is its ability to multitask- it is able to use a single thread while processing multiple concurrent requests.

One of the first big group decisions made was whether or not to work with an ORM- in which Prisma was elected. ORMs create a layer between higher-level languages and databases. More specifically, Prisma bridged JavaScript to our PostgreSQL database, allowing us to streamline communication. Beyond the major benefit of readability, the abstraction between layers, greatly reduced complexity in database management.

For state management, our team chose to use Redux which keeps track of the changes made throughout the runtime of the client side. When running, Redux stores the application state in a container- one single object- and each component of the application can access any part of this store. By centralizing data, Redux allowed for the ability to keep track of the changes of

state that the application went through. This in turn would make tracing problems simpler. Redux allowed us to maintain the data across our application with multiple components, while remaining independent of those components. In contrast, sharing data across components on the same level without Redux increases complexity in that it may be difficult to determine where a state should live in order for data to flow properly.

In order to clean the given data file for the project, our team used the R programming language and JavaScript to clean the data. R allowed us to view the .csv file in its entirety in order to clear empty cells and unnecessary data. Once our data was cleaned, we were able to export a clean, new .csv file which had all the necessary data for our project. This allowed us to then work with the ORM and properly generate the database.

Overall, for the entirety of the project, each of the above elements listed helped our team to implement this project. React.js enabled us to have the front-end of our application look and feel like a real library-management-system. To communicate with our desired server, we chose Node.js, which simplified the connection between our front-end to the database. Regarding the database aspect, Prisma made handling our database simple by allowing us to work with the higher-level language of our choice and reducing any difficulties that may come from reading/writing directly in query language. Lastly, Redux provided state management which kept hold of our data. Redux also handled any updates made to the data and changed states whenever necessary. In deciding whether or not to use each of these elements, efficiency and scalability were taken into consideration.