Liam McMain
Seth Rayley
Daniel Glass

Library Web app Project

This project was a great exercise in developing an application when working with a team. Our application has 3 pieces. The database, the back-end, and the front-end. The database was created with a Python script for MySQL that takes into account future use and efficient storage of records. The reasoning for using Python was its useful data manipulation libraries (pandas) and my (Daniel) familiarity with it. The reasoning behind using MySQL was simply because it was already used in the course, so it was the easiest to set up. The back-end is an interface, written in Node.js, between the front-end and the database that makes available the queries necessary for all our record retrieval needs. The back-end make it possible to get the records from specific queries at REST. The front-end, written in React, implements an intuitive user interface with a sleek, minimalistic design. The reasoning behind using javascript for the front-end and back-end was two-fold: enabling rapid deployment and the familiarity Liam had with it prior.

The database design was heavily influenced by the diagram provided in the project requirements document. The diagram provided a perfect reference for what tables to make and the keys for each respective table. What the design document didn't include, however, was the creation of cascade tables. The cascading of record drops, I believe, is pretty crucial to the functionality of the database. Due to this, full cascade was implemented in the database. Field values were optimized for space, so dates are stored as ISO-8601 formatted strings. The method of inserting the database was fairly irrelevant as far as the requirements are concerned, but Python was used for the manipulation of data, table creation, and record insertion. The data had to be initially validated by checking character encoding, as well as replacing trouble field values such as apostrophes and NULL values. After validation, the records are iterated through a pandas database inserting each record's fields into their respective tables. Then, the script was given a simple interface for the user to input server details to allow for ease of use. The script, however, was slower than an SQL dump script, so I generated a dump SQL script from MySQL once the database was filled with the Python script for ease of use.

The backend is written using node and express as an api/router running on port 3000. The api is REST API and the frontend makes REST API calls to the backend over HTTP. Each of the different requests such as creating loans, searching for books, getting the current loans for an id, returning a book, calculating fines, etc. are all implemented at a certain endpoint. When the frontend needs to get data from the

backend or update data in the database, it calls one of the endpoints on the backend. All of the data validation is done on the backend to make sure the requests from the frontend are in the right format, that you don't violate any business rules such as checking out more than 3 books, etc.

The frontend is built using React running on port 3000. The front-end was designed with usability in mind, so each page has a unique functionality in the platform. There are different pages for checking out books, searching for books, viewing fines, viewing books checked out by a user, and more. Aesthetic was also a primary focus, but with the time constraints in place, there was a need for the ability to quickly implement a good UI. Because of this, the Material UI library was used to create the majority of front-end pages.