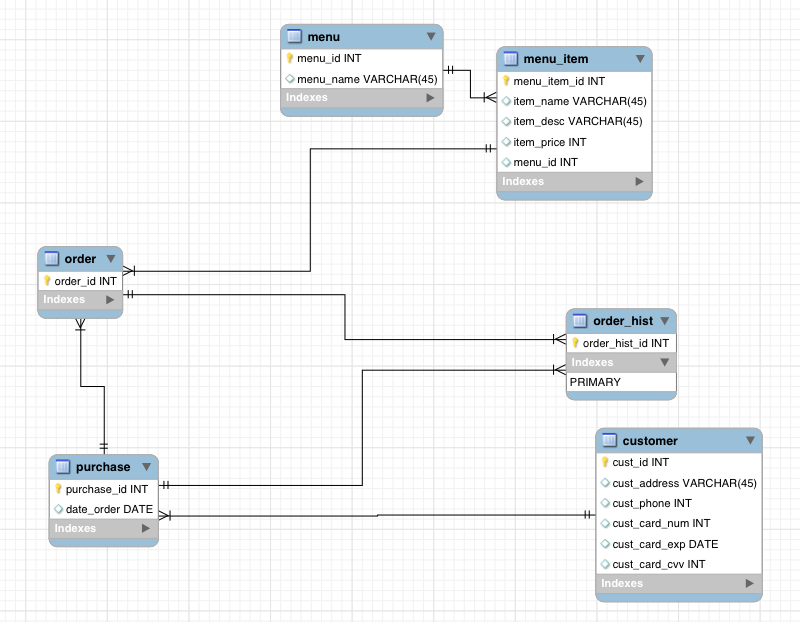
Andrew Smith

Jay Shome

Project Final Report

Before this project was started, we were reached out to by a friend’s father and Andrews’s uncle to create an online ordering application. The application would be better then usual online ordering applications for a few reasons. The first is that it would be dynamic for multiple restaurants to save multiple menus in the application. The second difference is that it would have a more unique person to person sales feel, as in at the end of the order it will say, for example thank you for ordering, last time you ordered you purchased French fries would you like to add French fries to your order? These types of questions are what make fast food places like McDonalds millions of dollars a year, and if integrated online would help companies get significantly more of a return from online ordering. For this project specifically, we dealt with how to add the items into an application from the menu owner’s perspective. This is the first step into completing the goal of an online ordering system, but it is the largest one since from here all that needs to be added is styling and security features that do not allow users other then the owners to write.



Here is the ERD for our database system.

The way we set up our project was to have as little logic being done on our frontend application as possible. The project setup is a SQL database, with a python Flask application, and a ReactJS frontend application. This is done by having all SQL operations stored as MySQL procedures and having backend python scripts call to the SQL database. The way this works too keep it light is that by pressing a button or making a change on the frontend view, it sends a post request to the corresponding python script. The python script then calls whatever CRUD operation is necessary to load the information requested. The SQL database is updated, and the python script then calls procedures that read all of the tables from the page. The information from all of the tables on the page is then converted into JSON and saved onto a server. The ReactJS frontend then picks up the JSON from the server and parses it on the page. This way all the frontend ReactJS application is doing is picking up the JSON from the server and formatting it correctly on the page. This is what we believed to be the best possible solution for this project, but in terms of class time and knowledge to complete it we fell slightly short.

CREATE🡪 All tables are created in SQL, insert statements are included in MySQL Procedures, in the Python Script add\_item.py and in the React App.js file.

READ🡪 Read operations are included in MySQL Procedures, in all of the Python Scripts, and in the React App.js file.

UPDATE🡪 Update operations are includes in MySQL Procedures, in update\_item.py, and in the React App.js file.

DELETE🡪 Delete operations are includes in MySQL Procedures, in removeItem.py, and in the React App.js file.

The first thing we learned during this project was how to create a Flask application and connect it to a SQL database. The second thing we learned is how complicated it really is to connect JavaScript to Python. The issues with connecting that impair the final results of our project are apparent once run, but it is really this one issue that keeps our project from working very well. The first problem that this brings us, is that we do not read the JSON directly from the server, this is not good and keeps us from dynamically changing the frontend with database changes. The second issue that comes with this problem for our application is that the app cannot actually dynamically send adding, removing, and updating requests to the python backend. This is seemingly solved in everything but the remove feature since we cannot actually remove from the frontend, but the request to python would send a request to SQL that would delete the object, then the request from python would run again and new JSON would be populated and replace the current state in the frontend app. After conducting research, the reason why we cannot connect to our Flask application is because we would actually need to deploy both applications onto an application like Heroku in order to see both running at the same time and working with each other. Because we did not have the resources and time available to learn Heroku we were unable to complete this part of our goals in time and thus are turning in a slightly incomplete assignment. But, all of these features individually work on there respectful applications which is why I believe we still have made a very good platform to build off of after this class ends.

In the future we plan to fix this issue by pushing our application onto a Heroku server, debugging, and then styling the application. The use of this application will then be sent to our family and friends that requested its development and released on web stores for companies to use free of cost.