README: How to use our web application

BEFORE DOING ANYTHING ELSE (STEP 1):

1. Download HW4 Folder
2. Load Folder into VS Code
3. Head over to server.js and adjust the database information to match you local machine:
    - Lines 14-20
    - Match to your own database and its information

```
const pool = new Pool({
   user: 'postgres',
   host: 'localhost',
   database: 'hw4',
   password: 'xxxxx',
   port: '5432',
});
```

4. Open new terminal
5. Type: npm install
6. Type: node server.js
7. Open localhost:3000
8. Click on Create Database: If your pool information is correct, then the database tables are created and you are alerted that you successfully inserted tables into the database. If your pool information is incorrect, then you will be alerted that the database tables were not able to be created, please check that your pool information is correct.
9. After successfully creating a database, click on Populate Database: If your pool information is correct, then the database table values are inserted and you are alerted that you successfully inserted values into the tables in your database. If your pool information is incorrect, then you will be alerted that the values were not able to be inserted into the table, please check that your pool information is correct.
10. Congratulations the database has been successfully implemented, now you can do any of the below tasks

Run Concurrent Simulation:

- After completing Step 1, we can now run our concurrent simulation
    1. Click on Run Simulation
    2. Click on Run Random Simulation
    3. 100 transactions are committed concurrently, and each transaction's key data and its status is displayed.
    4. Database Tables are now updated to show all transactions that are successful out of the 100 transactions.

5. Simulation successfully completed. Now you can click on “Run Random Simulation” again, to run another 100 customers processing, or you can click on one of the other buttons to see key data or perform actions with the updated database.

Order Food Transaction:

- After completing step 1, you can now order from McDonalds
  1. Click on “Order Food Transaction” / “Order” to order from McDonalds
  2. Pick 1 of the 25 customers by clicking select on row corresponding with name
  3. Pick 1 of the 10 restaurants by clicking select on row corresponding with restaurant name
  4. Select a payment method: Either card or cash
  5. Now the customers' bank information and the employee information for the selected restaurant are displayed.
  6. Select what items you want to purchase by clicking add and/or remove. Item stock is also displayed depending on restaurant selected
  7. If a customer is a member, then cost is incremented with member price. If not a member, then cost is incremented with normal price.
  8. Total cost is displayed, and you have the option to type or increment a tip for your order
  9. Tax of 8 % is added
  10. Order is placed. You are alerted if the order is successful. Alerted order failed if there is a problem in order such as bank balance or stock.
  11. If the order is successful, then the database is updated with new transactions and you are sent to the view tables page, where you can see the transaction you committed.

View Tables:

- After completing step 1, you can now view the tables in the database
  1. Click on “View Tables”
  2. Displayed are:
     - Restaurant List
     - Menu List
     - Customer List
     - Bank List
     - Transaction List
     - Order List
     - Stock List
     - Employee List
     - UPDATE: Update Item Price

        - INSERT: Add New Customer
        - DELETE: Restaurant
        - INSERT: Restaurant
    3. The 8 tables displayed are straight from the database
    4. UPDATE: Update Item Price -> Updates the cost and member exclusive cost for a specific item in menu
    5. INSERT: Add new customer -> Adds new customer if all inputted information is correct
    6. DELETE: Restaurant -> Deletes a restaurant if there is no transaction data for the restaurant. Will not delete restaurant if that restaurant_id has transaction data, as that would result in data loss
    7. INSERT: Restaurant -> Inserts a restaurant into restaurant table

View Restaurant Analytics:

- Works best when simulation is ran at least once
- Provides many unique queries that show key data for managers
    1. Top Selling Product Per Restaurant: Returns highest selling item for each restaurant
    2. Least Selling Product Per Restaurant: Returns lowest selling item for each restaurant
    3. Top Ordering Customers: Returns top 5 customers in term of orders
    4. Top Performing Restaurants Based On Quantity Sold: Returns top 5 restaurants in terms of quantity sold
    5. Top Performing Restaurants Based On Orders Placed: Returns top 5 restaurants in terms of orders placed
    6. Restaurants Ranked Based on Gross and Net Profit: Returns each restaurants Gross Profit and Net Profit assuming 50% margins. Ranked by profit

Transaction Logic (Single Order):

- Customer selected and stored as variable
- Bank Account of customer also stored as variable
- Restaurant ID for selected restaurant stored as variable
- Payment method: card or cash save
- Items that are added are stored in a hashmap that hold item_id as a key and quantity as value
- Cost is changed based on items added and removed
- Cost changed by tip amount
- 8% Tax added on
- When confirm is clicked, the transaction is ready to begin
- Transaction Steps in POST:

    1. BEGIN transaction
    2. Insert Transaction data to the transaction: If error rollback
    3. Insert order data to order_data: If error rollback
    4. Update stock to decrease by the amount bought by customer: If error such as not enough stock, then rollback
    5. Get account balance: if not found rollback
    6. Update bank account to become the old balance - cost: if error rollback
    7. If no errors, then commit
    8. Client released

<u>Run Simulation Logic (100 concurrent orders):</u>

- Loop done 100 times
- For each loop:
    1. Get customer id which is random number 1-25 and save as cid
    2. Get bank account number for that cid and save as ba
    3. Get restaurant id which is random number 1-10 and save as rid
    4. Payment method is cash for when i is even, and card when i is odd
    5. itemDict, which stores order data, is empty hash right now
    6. Items is random number 1-5, dictating the number of items the customer will buy in this transaction
    7. Cost is total cost of order
    8. Member is true if i is divisible by 3
    9. Loop j times, where j is items number:
        - Pick a random number 1-15
        - Add one to its value in itemDict
        - If member, then add memberCost of item to cost. Else add normal cost of item to cost
    10. Cost = cost * 1.08. Adds tax
    11. Run Transaction: Done serializable, to keep integrity of data
    12. If any one of the queries fails, the whole query is rolled back.
    13. When loop is done, all results are displayed to user.
    14. Time is displayed as time when transaction finished, subtracted by the time the user clicked the simulation button

<u>Assumptions:</u>

1. The Menu for all McDonalds branches are the same
2. In the simulation, we assume the simulated customers do not tip, as tipping is very rare in McDonalds
3. Assumption is Tax is 8%
4. Currency is USD
5. Item price and member item price is same across all branches
6. Only payment methods are cash and card

7. Each employee has predefined roles
8. Card numbers and Bank Account Numbers are unique in the bank
9. There is one universal banking system
10. No returns

Decisions:

1. **Only 8 Tables:** We valued referential integrity and normalization in our project. As a result, we went with only 8 tables to make sure that data was clean, accurate, and could work together in transactions. Moreover, we made sure to make our app as precise as possible, so that we could do a deep dive into the data analytics portion of the project. This allowed us to get top performing and least performing data, as well as key trends and profit margins. If we had more tables, the logic would get more complicated, but the information would stay relatively the same. By normalizing data in carefully designed tables. We have reduced redundancy and ensured that data integrity is preserved by using primary foreign key constraints. Each table focuses on a specific aspect of the system. Make the relationship clear and manageable.The chosen planning structure corresponds to the core operations of the system, such as customer, transaction, and inventory management. without creating unnecessary complexity This clear organization simplifies debugging. With a well designed table, queries will be more efficient. This reduces the computational cost for aggregated searches. Helps ensure fast data recovery. This is important for real-time systems such as ordering food from a restaurant. By keeping the format concise we ensure that analysis is easy. For example, a logical and general database design allows for trend analysis. Performance indicators And the profit margin is easy. Introducing additional tables can separate the data. This makes the analysis more complex without providing important new insights.Although the current plan supports the needs of the application. But it also allows for future scalability. If there is a need for new features or information The database can be expanded without affecting the existing structure.The plan directly supports business operations, such as inventory management. Employee tracking and customer transactions and avoid too much complexity. Adding more tables can create redundant or redundant layers that add little to the cost but increase the complexity.

2. **Not hiding important data:** Instead of hiding important data like the customers info when selecting a customer, we decided to focus more on the queries and complexity of our site, per Mr. Ordonez's instructions. By putting more focus into our queries and transactions, we were able to do more complicated queries and have a better overall product. A product that would stand out from the competition

3. **Why we did not make tip or tax a database value:** We as a group valued normalization above everything else. We wanted our data to be clean, have referential integrity, and be normalized. As a result, we kept only totalCost as a database value, and instead calculated tax and tip as the transaction went on. This allowed normalization and better functionality than saving the tips or tax as database value.

4. **Why we kept a universal menu:** We decided to implement a universal menu across all branches to ensure consistency in customer experience, so customers always know what to expect regardless of location. This approach streamlines inventory management by standardizing stock requirements across branches, simplifying procurement and reducing operational risks like overstocking or stockouts. It also enables unified data collection and analysis, making it easier to identify top-performing and least-performing items across the chain while analyzing trends for better decision-making. Operational efficiency is enhanced, as training employees becomes simpler, allowing for easier transfers between branches and standardizing order preparation and quality control processes. A universal menu is quite common in McDonalds, as a lot of the same items are available in all of the branches. Lastly, scalability of the project is easier with a universal menu

New Feature:

- Added a time section for each transaction. When the transaction is called, the time starts. For each transaction, their respective time ends when their transaction is complete. The difference between the end and start time is displayed. You will see that concurrency based on serializable is achieved as the time increases as you go down the table.

Files:

1. Index.html: holds home page (replacement for home.html)
2. Simulation.html: holds the page to run simulation
3. Analytics.html: holds the analytics for the transactions
4. Transaction.html: page for doing single transaction
5. Tables.html: page that shows tables and INSERT,DELETE,UPDATE functions
6. server.js: handles all the queries and advanced logic
7. Query.sql: holds all the sql for the advanced queries in Analytics.html
8. Transaction.sql: holds all sql for the transaction that is done in application
9. tableCreation.sql: holds all sql for the creation of tables for the database
10. tableValues.sql: holds all sql for the values inputted into the tables for the database

GUI Button:

1. Create Tables: Creates the tables in the database
2. Populate Data: Adds data rows for each table in database
3. Run Simulation: Takes you to simulation page
4. Run Random Simulation: Runs concurrent 100 simulations