560 Project Report

Great Eats Restaurant Manager

Created By: John Chapple, Alex Varenhorst, Jordan Greene

Introduction

Overview

Our team created an online management application that allows restaurants access to different aspects of the business. Our team wanted to allow the users one central location to be able to communicate between coworkers and look at different types of analytics. Some larger restaurants that are using similar software include Kansas State's Kramer Dining Center, Culver's, and Kona Ice.

Targeted Users

The intended users of the application would include restaurants that are looking for a way to manage their business, being able to see their employees as well as up to date analytics to better understand how their business is operating. The everyday users of this application would be the restaurant managers and employees.

System Importance

One of the benefits of an application like ours is having multiple facets of the business all in one location. Being able to see your entire staff list as well as being able to communicate with staff through the bulletin board is great to have all together so you don't have to juggle all those types of things with various apps.

System Expansion

When it comes to expanding this application, we plan on adding in tables and tabs to help with inventory management. Currently, the restaurant uses another system for inventory management, so our systems are independent from each other. But we would

like to deduct from the system's inventory when we have sales. That way the restaurant can see not just their sales, but also how much inventory they have left.
With adding this to the system also comes other smaller implementations. Such as getting a notification when inventory for a certain item is lower than expected. Also with inventory deduction we can use more analytics to better forecast for when we need to order more inventory. Overall, we would like to incorporate more of the restaurant's inventory into the analytics. We would like to design into our current system a way to keep track of inventory and also be able to update inventory when new shipments of food, silverware, etc come in. That way we can merge two independent systems into one and make life easier for the managers and employees of the restaurant.

Technical Description

Front-end Web Application

- Node.js - I decided to use Node.js for the front-end because it allows us to access powerful and essential packages or a front-end app. It makes it easy to organize a project, how to run it, and what packages are along with it.
- Vue.js - Vue.js is a front-end framework for elegantly combining compartmentalized html templates, code, and data that can work together. These compartmentalized templates are called components. For example, I could make a page for showing the employees, which would be a component. Inside that component I could handle the fetching of the employee data while also rendering a single Employee component for each employee object gathered from our database.
- TailwindCSS - TailwindCSS is an upcoming utility framework that comes with pre-built CSS classes that are intuitive to use, allowing for a quick iteration process of styling a webpage.
- Date-fns - This is a package that comes with a comprehensive list of functions that allow easy parsing and displaying of dates in any fashion or format you

could imagine. It also follows a tree structure. Or, to put it simply, I only need to import the specific commands that I need in a certain instance instead of importing the whole set of functions.

- Axios - Axios is a promise based HTTP client for the front-end. It allows me to easily make a request to the back-end. It turned my previous HTTP fetch from eight lines to one. I can specify the endpoint route, and the data I want to send. Then either what to do on success of the request, or what to do in case of an error.
- Highcharts - For the ability to show the information determined by report queries, we decided to use Highchart for its ease of use in our Vue front-end. It also is appealing to look at.

Back-end Server

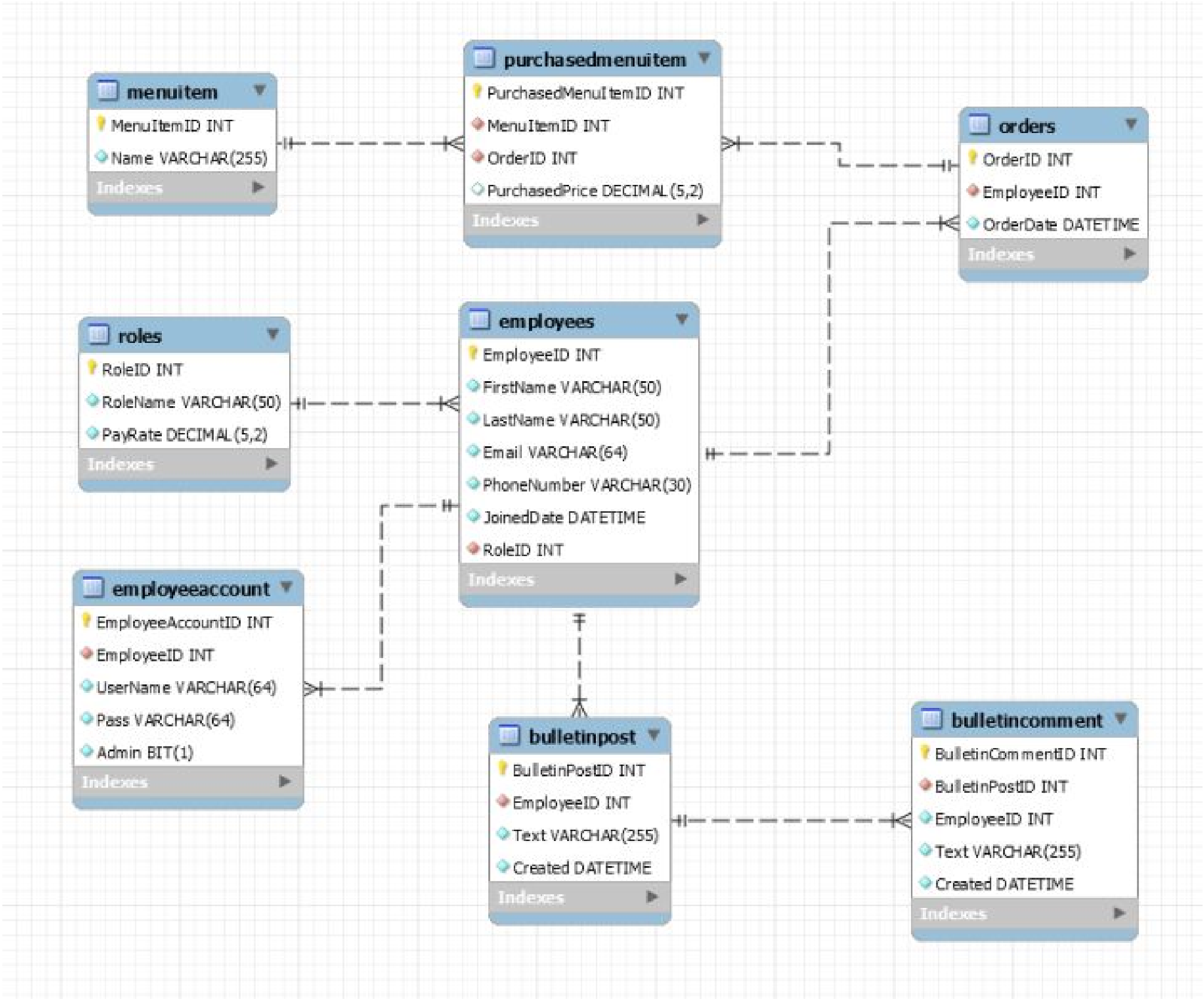
- Node.js - Again, Node.js gives us access to the npm database, giving us access to powerful packages. In addition, node.js is basically a javascript client separated from a browser, so it allows me to use javascript to create a back-end.
- Express.js - The Express package allows us to organize endpoints on the backend easier. I can define an endpoint and what type of CRUD request I am expecting, then send it to a specified file or function. It's easy to grab info from the request, and send it back.
- MySQL npm package - In order for our back-end to connect and query our MySQL server, we needed this package. These two actions are its only purpose.

Database

- MYSQL - A mysql server was used in this project to both create and query the database. As aforementioned, we made these connections and queries to the mysql server with a mysql npm package. Mysql queries were made in the front end in the form of a string. We simply just pass the mysql query as a string to this npm package to make the query.
- MYSQL Workbench - To manage and create the database, we all used mysql workbench. Mysql workbench is a client that facilitates both the local server, and

the database to be used for the project. This tool was vital for us in looking up new information, and seeing whether our queries on the front-end were correct and affecting the database as we should expect.

Database Design

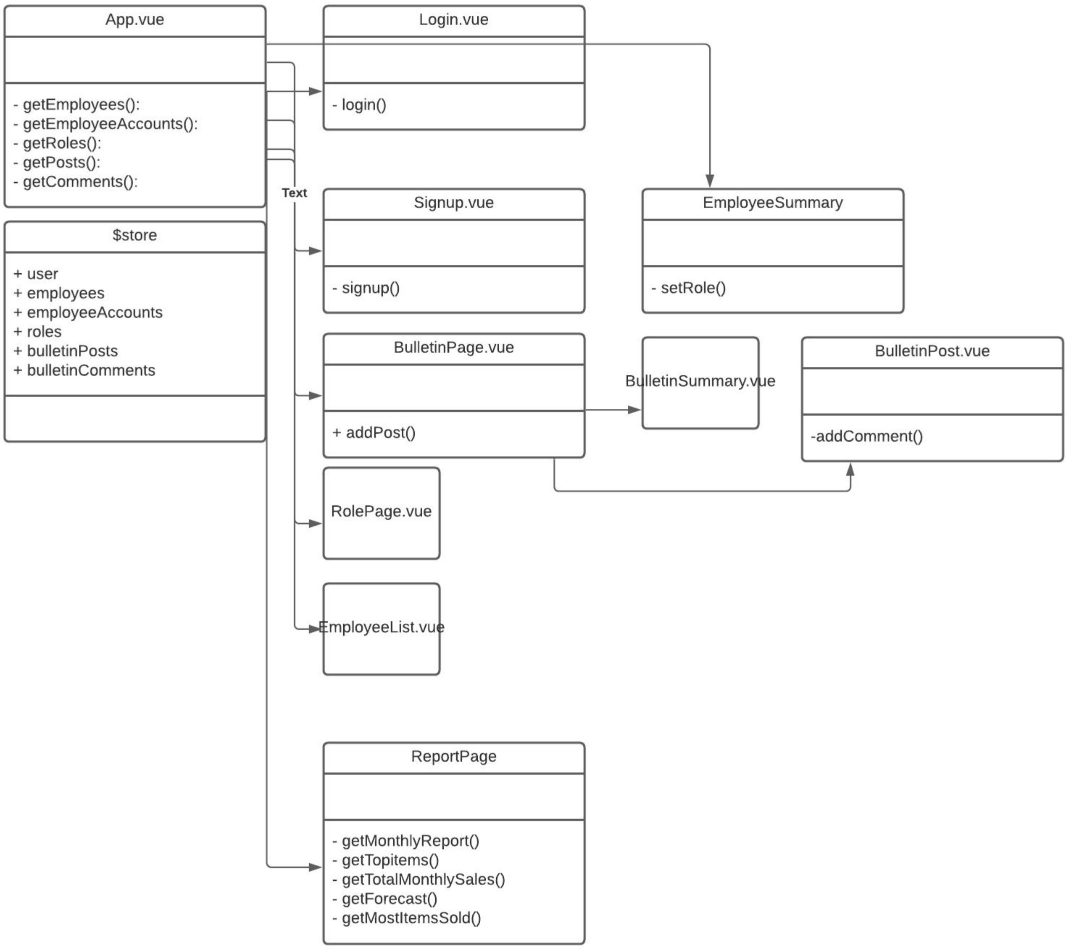


The design of this database is such so that the user can post bulletins, comments, and also receive information regarding their colleagues, performance, and other interesting reports. Each Employee is assigned a single role, and this role is essentially for administrative purposes. Those with RoleID 1, for example, are managers, and are

therefore allowed to make bulletin posts. Those with RoleID 2 though will not have this ability, and will only be able to make comments to those posts.
The reports generated are derived from multiple tables of the database, and we designed this database so that reports could be more easily made and it would also open the system to more improvements and extensions down the road.

For the relationship between employees and employeeaccount it actually is a 1-1 relationship. MySQL server wouldn’t recognize it as such. But it was confirmed to be built correctly by John Keller during Office Hours on 11/19/2020.

System Design



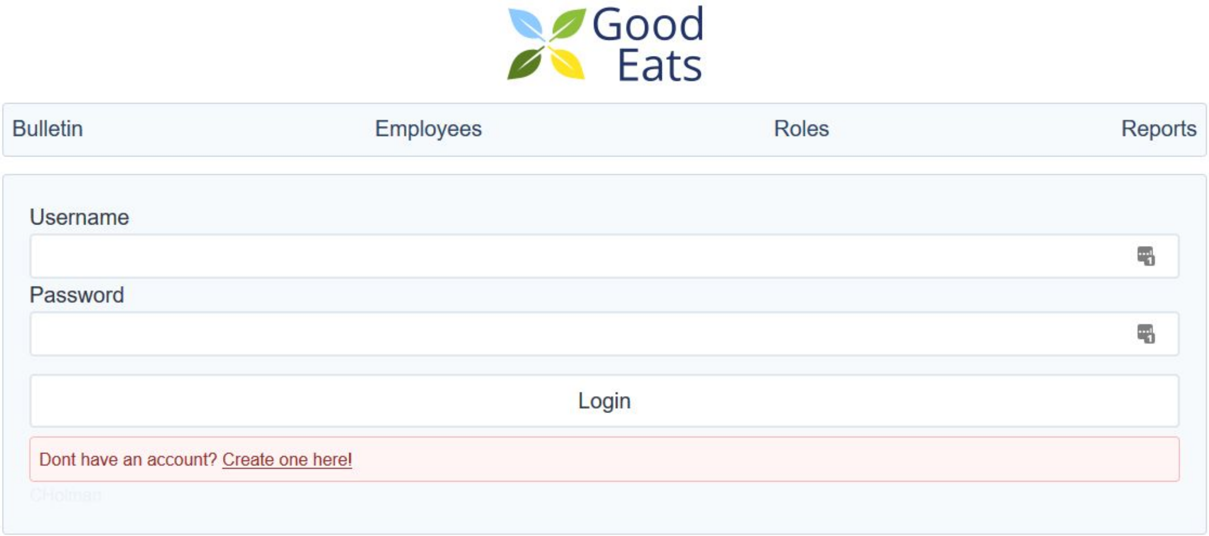
This is a basic UML diagram for the Vue front-end app. The app.vue component is the highest level, and then is composed of multiple page components, which a few contain smaller components such as the BulletinPage component containing BulletinPosts and BulletinSummary.

Since the composition of our vue web app and node server were not as traditionally structured as other software patterned in UML, it may be best to supplement the explanation through text.

Each component would locally hold data relevant to the actions of the component. For example, the bulletin page had a variable to determine which post to display, and if we were going to create a new post. The login component keeps track of what is typed into the inputs to send to the back-end.

The node back-end served as a listener for certain endpoints, such as a post request to bulletinPosts, which upon the request would run specified code. This could have been all written in the main app.js file, but was organized by file in the routes directory. Each route would take information from the body of the request, perform a SQL query, and return the desired data.

System Features and Usage



Username

Password

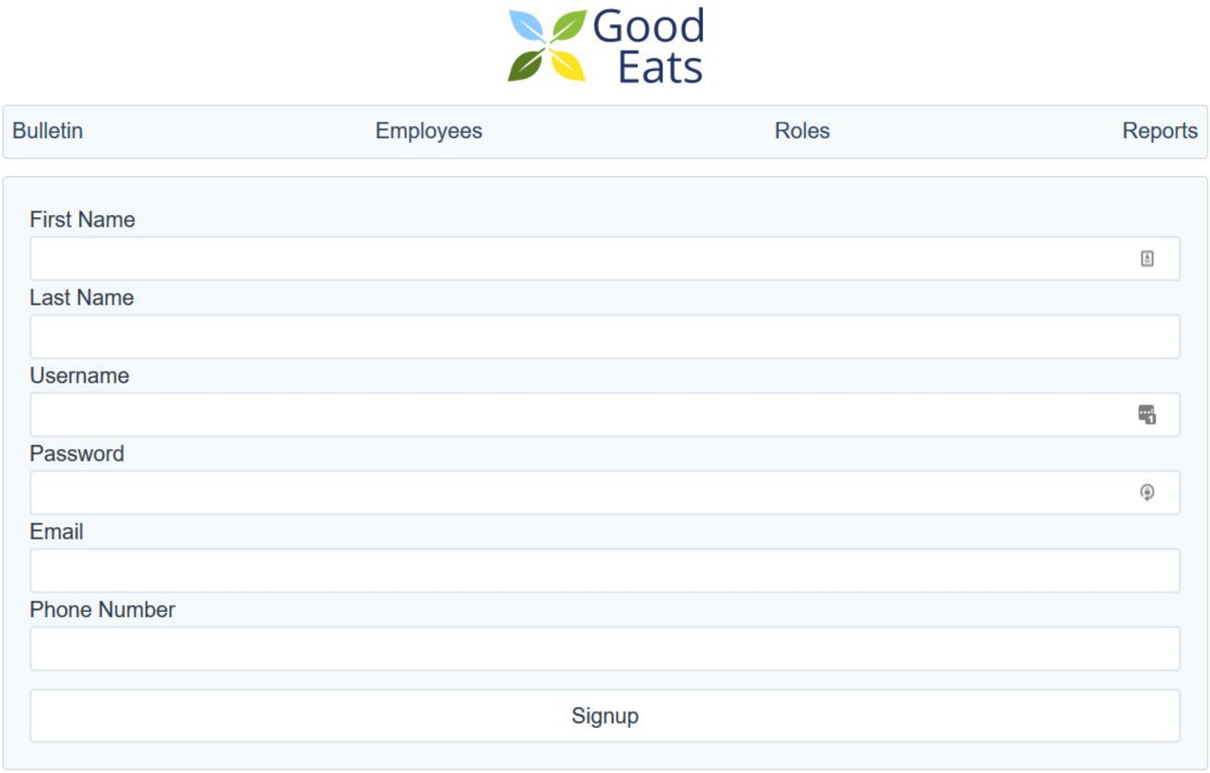
Dont have an account? Create one here!

CHolman

This is the page that the user is brought to on open. They can see what tabs are available, but won't be able to navigate to them until they login. If the user doesn't have a login yet, they can register themself with the company by creating an account next.

Logging in takes the username and password and finds a matching item on the EmployeeAccounts table.

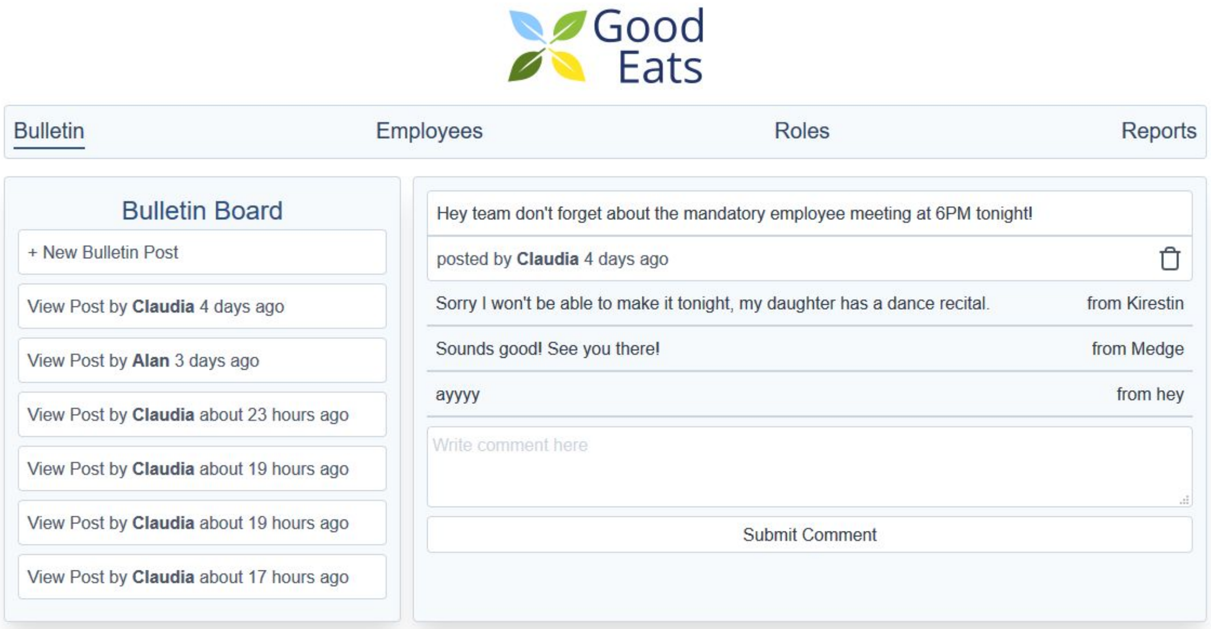
By default, the account username for generated employees is the capitalized first letter of their first name, and their last name capitalized. The account's password is "Password".



This is the page that the user can go to if they don’t have a login, and therefore are not registered with the system. When they click Sign Up the data they filled out gets sent to the database using these two statements.

```
"INSERT INTO `Employees`(`FirstName`,`LastName`,`Email`,`PhoneNumber`) VALUES ('" +
firstname + "','" + lastname + "','" + email + "','" + phonenumber + "')"

"INSERT INTO `EmployeeAccount`(`EmployeeID`,`UserName`,`Pass`) VALUES ('" + empid +
"','" + username + "','" + password + "')"
```



View Post by **Claudia** 4 days ago

View Post by **Alan** 3 days ago

View Post by **Claudia** about 23 hours ago

View Post by **Claudia** about 19 hours ago

View Post by **Claudia** about 19 hours ago

View Post by **Claudia** about 17 hours ago

Hey team don't forget about the mandatory employee meeting at 6PM tonight!

posted by **Claudia** 4 days ago

Sorry I won't be able to make it tonight, my daughter has a dance recital. from Kirestin

Sounds good! See you there! from Medge

ayyyy from hey

Upon login, the first page that the user is brought to is the bulletin board page. This is where they can see all the posts created by managers, and can select them on the left to view the contents and the comments. Anyone can post a comment on a bulletin post, but only the manager has access to creating a new post. You can see that button at the top of the left column. If you own a post, you are able to delete it.

For the Bulletin tab we have it so that Managers can post to the board. When they post they use this statement.

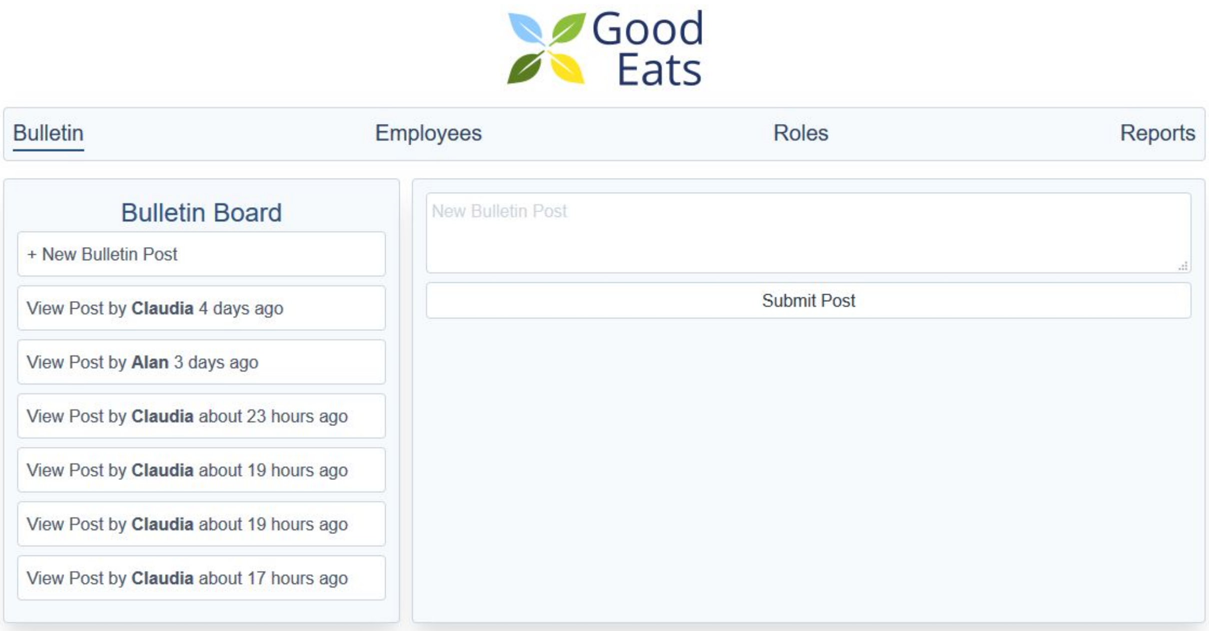
```
"INSERT INTO `BulletinPost`(`Text`,`EmployeeID`) VALUES ('" + Text + "','" +
EmployeeID + "')";
```

When loading the Bulletin tab we grab all of the bulletin posts that are available using this statement.

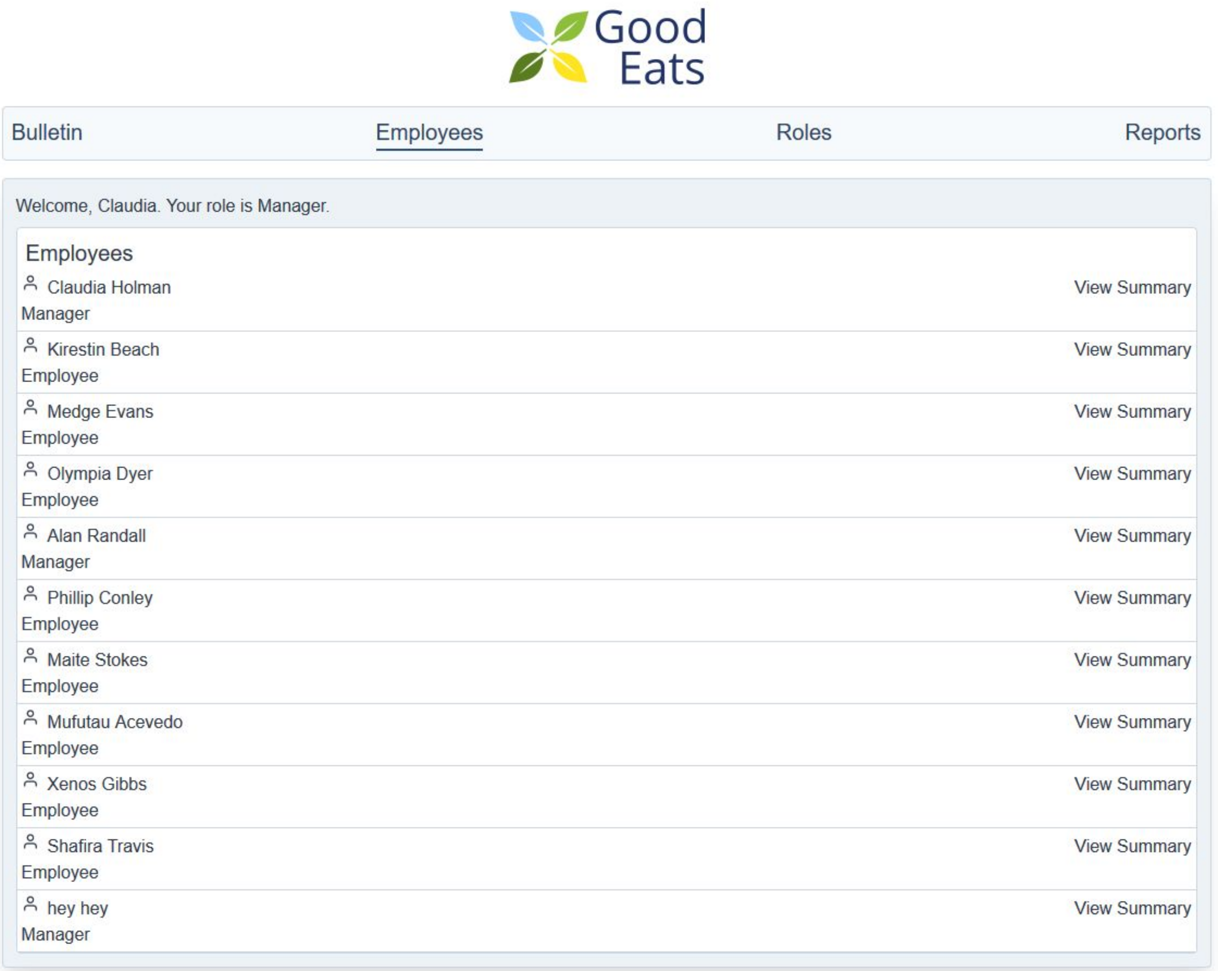
```
"SELECT * FROM BulletinPost"
```

When selecting a certain bulletin post we will use this statement. To load the comments we load in all the comments that match the BulletinPostID and show them beneath the parent BulletinPost.

```
`SELECT * FROM BulletinPost WHERE BulletinPostID = ${insertId}`
```


Good Eats
Bulletin
Employees
Roles
Reports
Bulletin Board
+ New Bulletin Post
View Post by Claudia 4 days ago
View Post by Alan 3 days ago
View Post by Claudia about 23 hours ago
View Post by Claudia about 19 hours ago
View Post by Claudia about 19 hours ago
View Post by Claudia about 17 hours ago
New Bulletin Post
Submit Post

Here is the view of the bulletin board when the new bulletin post is pressed. There is a text area input and a button to create the new post.



Welcome, Claudia. Your role is Manager.

Employees

Claudia Holman View Summary
Manager

Kirestin Beach View Summary
Employee

Medge Evans View Summary
Employee

Olympia Dyer View Summary
Employee

Alan Randall View Summary
Manager

Phillip Conley View Summary
Employee

Maite Stokes View Summary
Employee

Mufutau Acevedo View Summary
Employee

Xenos Gibbs View Summary
Employee

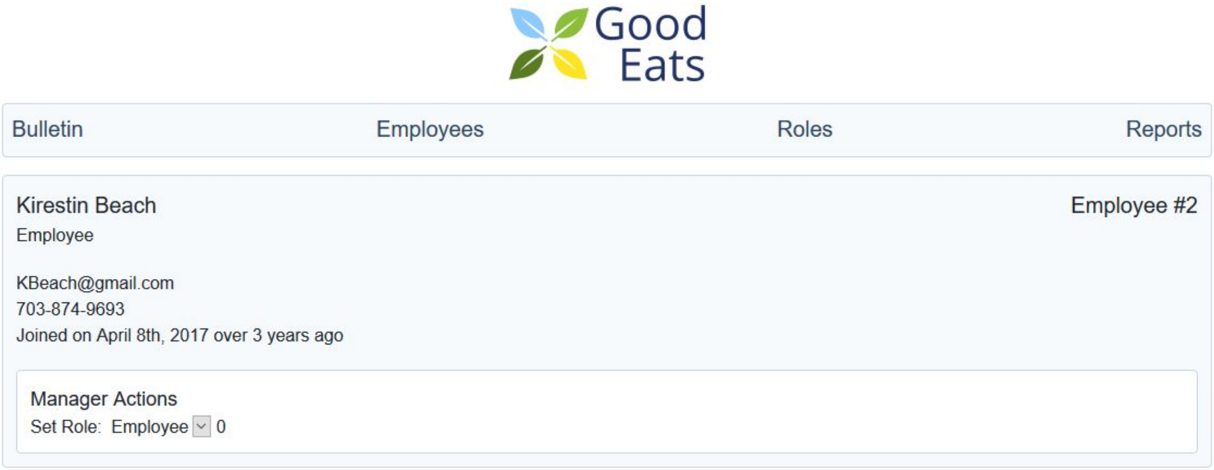
Shafira Travis View Summary
Employee

hey hey View Summary
Manager

The next tab is the employees list. Here, any user can see the other employees. On this page, you can only see the name of the employee and their role, but you can click on "View Summary" to go to the employee summary page to see more details.

To grab all of the employees we just use this statement and load this in. Then based on what the user clicks we then show/hide data from the user. So if we clicked "View Summary" we could see a more in depth look at each employee. But the statement we use to grab the employees is.

```
"SELECT * FROM Employees"
```



Kirestin Beach

Employee #2

Employee

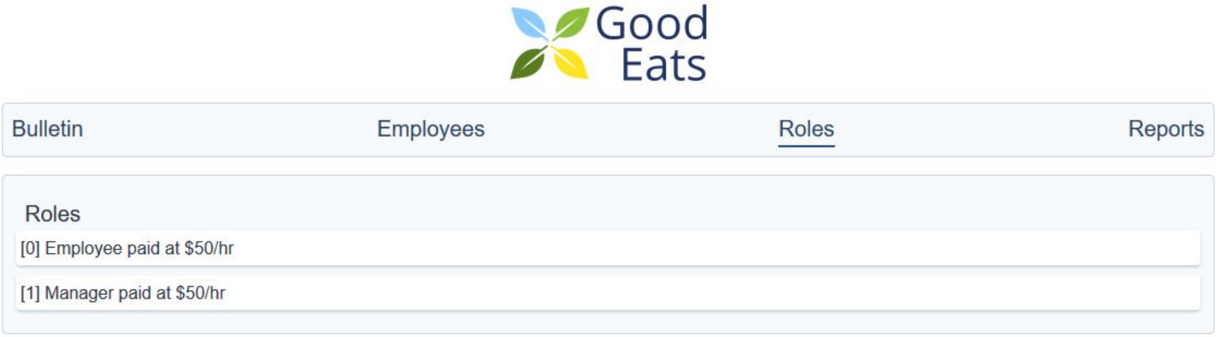
KBeach@gmail.com
703-874-9693
Joined on April 8th, 2017 over 3 years ago

Manager Actions

Set Role: Employee 0

Once a user has clicked "View Summary," they are taken to this page where they can see other relevant details about employees such as their email, phone number, and join date. Only managers are able to change the role of any of the employees. But when we do change the role of an employee we use this statement.

```
`UPDATE restaurant.employees SET RoleID = ${req.body.RoleID} WHERE EmployeeID =
${req.params.id}`
```

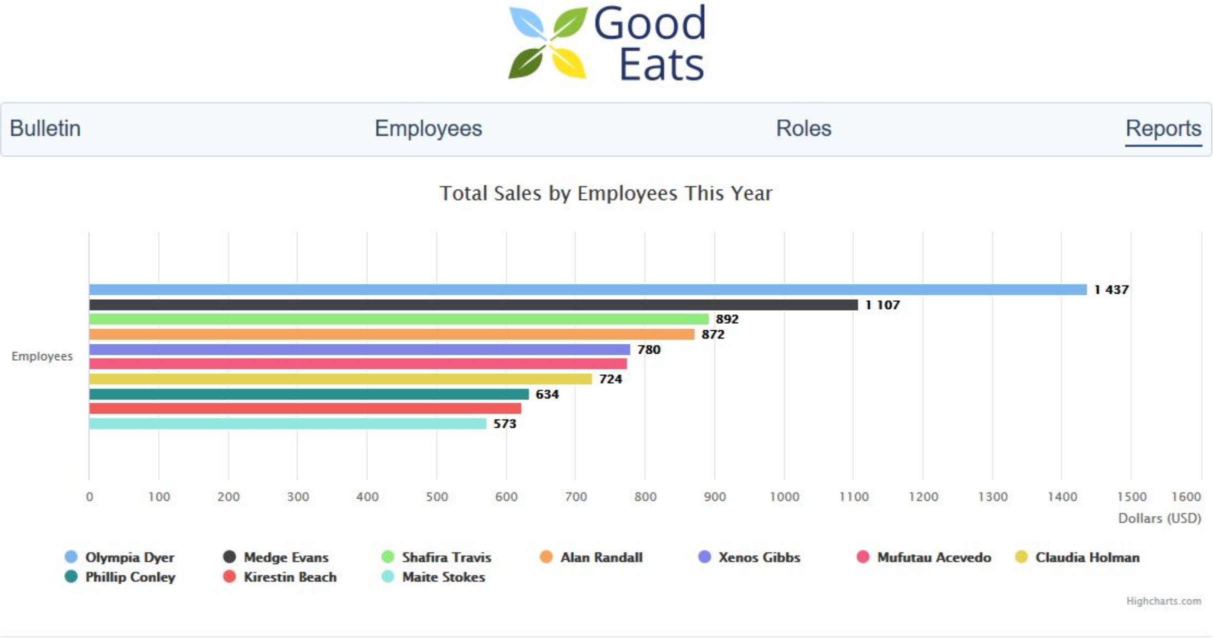


Roles

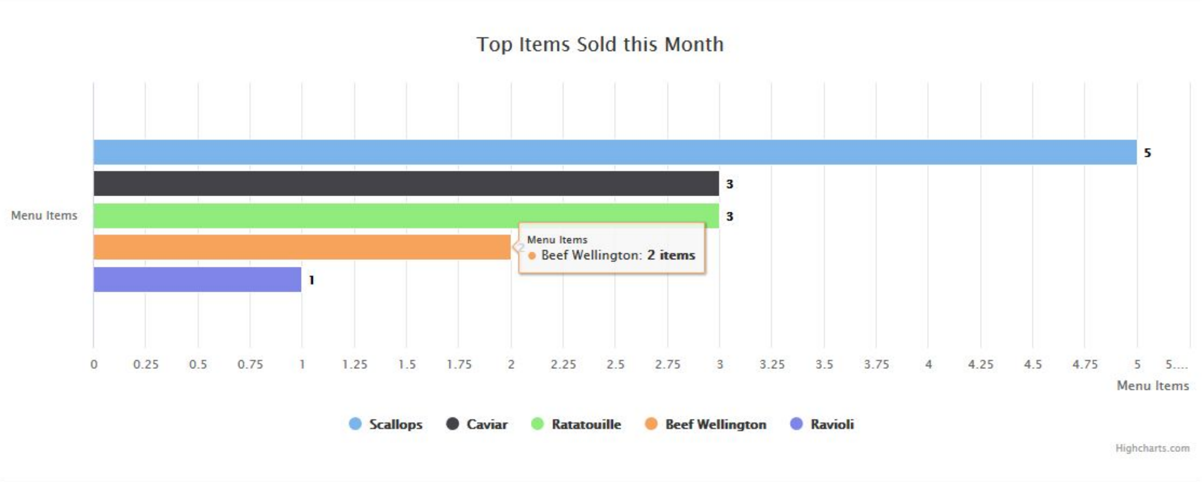
[0] Employee paid at $50/hr

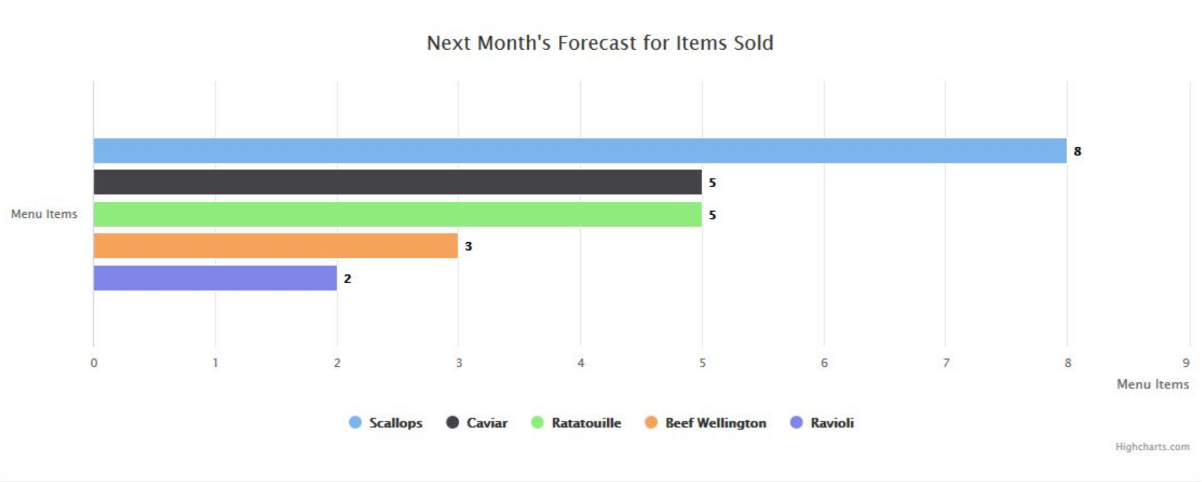
[1] Manager paid at $50/hr

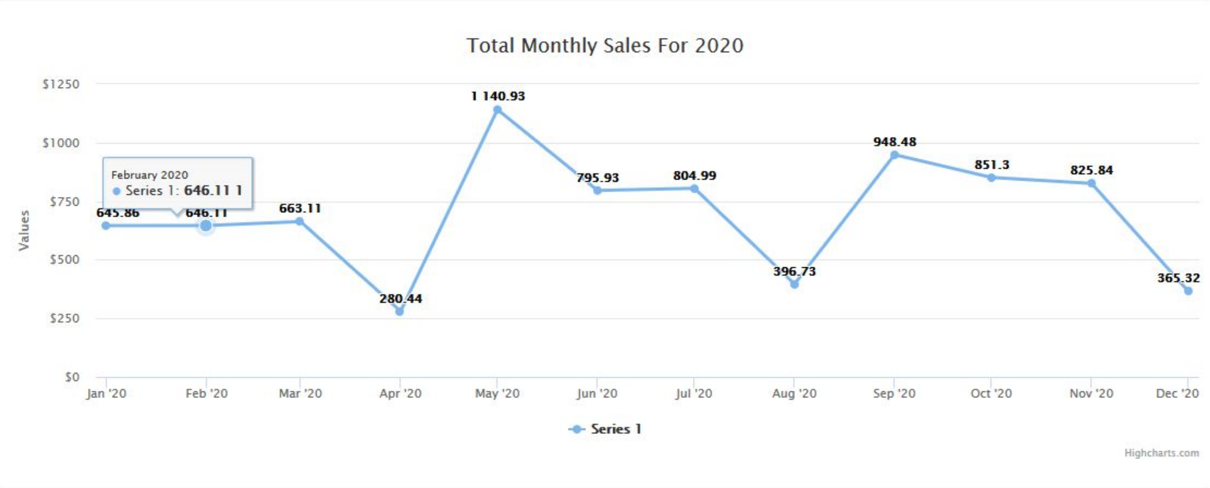
The role page contains a list of the roles that employees can have. In our example, we only have two.



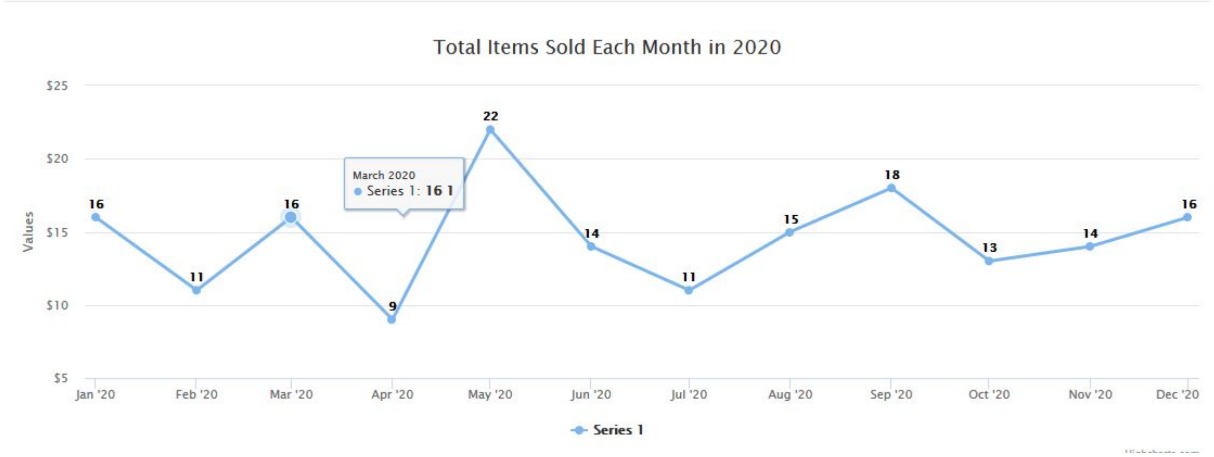
The last tab is the reports page, where users can see a variety of reports detailing the performance of employees and items for sale.







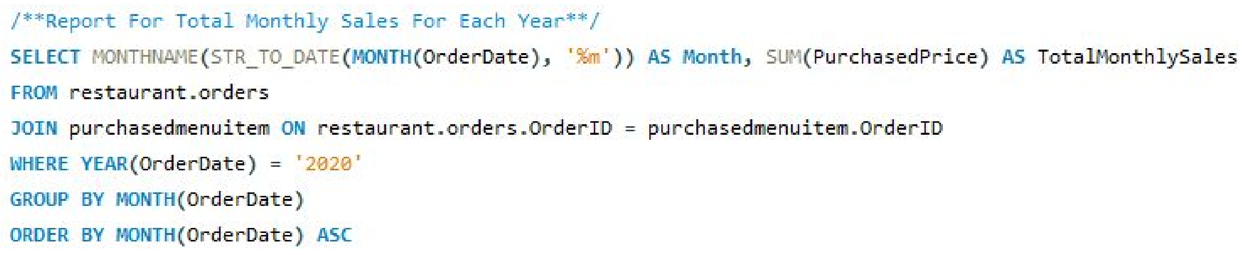
Here are screenshots of the other example reports. Users are able to hover over different parts of the charts to see more relevant information.



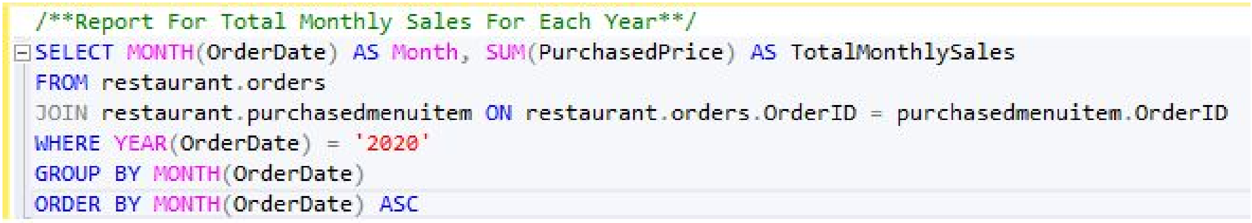
Report Queries:

For the screenshots I will include MySQL first, then SQL Server second.

The first report finds the Total Monthly Sales For A Designated Year. For our report we showed the results for the year 2020, just so we can keep our data relevant with the website.

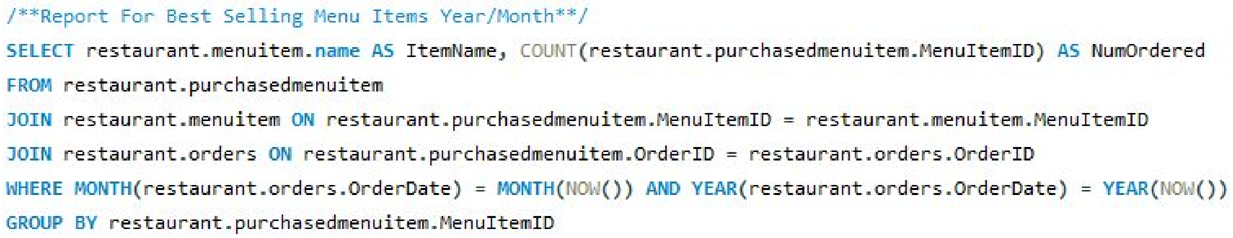


```
/**Report For Total Monthly Sales For Each Year**/
SELECT MONTHNAME(STR_TO_DATE(MONTH(OrderDate), '%m')) AS Month, SUM(PurchasedPrice) AS TotalMonthlySales
FROM restaurant.orders
JOIN purchasedmenuitem ON restaurant.orders.OrderID = purchasedmenuitem.OrderID
WHERE YEAR(OrderDate) = '2020'
GROUP BY MONTH(OrderDate)
ORDER BY MONTH(OrderDate) ASC
```

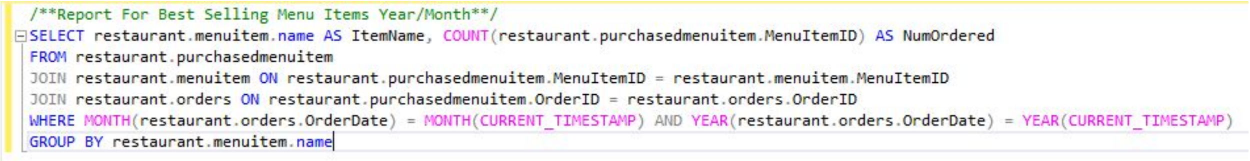


```
/**Report For Total Monthly Sales For Each Year**/
SELECT MONTH(OrderDate) AS Month, SUM(PurchasedPrice) AS TotalMonthlySales
FROM restaurant.orders
JOIN restaurant.purchasedmenuitem ON restaurant.orders.OrderID = purchasedmenuitem.OrderID
WHERE YEAR(OrderDate) = '2020'
GROUP BY MONTH(OrderDate)
ORDER BY MONTH(OrderDate) ASC
```

The second report shows the best selling menu items for each year or month. We had it show November 2020 results so we can see what the best selling items were for this month.

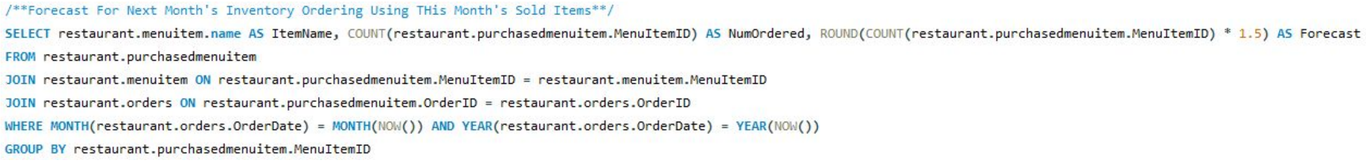


```
/**Report For Best Selling Menu Items Year/Month**/
SELECT restaurant.menuitem.name AS ItemName, COUNT(restaurant.purchasedmenuitem.MenuItemID) AS NumOrdered
FROM restaurant.purchasedmenuitem
JOIN restaurant.menuitem ON restaurant.purchasedmenuitem.MenuItemID = restaurant.menuitem.MenuItemID
JOIN restaurant.orders ON restaurant.purchasedmenuitem.OrderID = restaurant.orders.OrderID
WHERE MONTH(restaurant.orders.OrderDate) = MONTH(NOW()) AND YEAR(restaurant.orders.OrderDate) = YEAR(NOW())
GROUP BY restaurant.purchasedmenuitem.MenuItemID
```

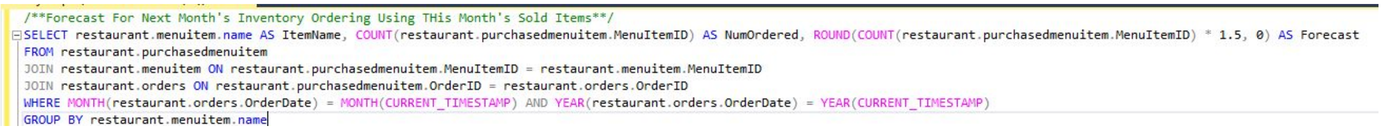


```
/**Report For Best Selling Menu Items Year/Month**/
SELECT restaurant.menuitem.name AS ItemName, COUNT(restaurant.purchasedmenuitem.MenuItemID) AS NumOrdered
FROM restaurant.purchasedmenuitem
JOIN restaurant.menuitem ON restaurant.purchasedmenuitem.MenuItemID = restaurant.menuitem.MenuItemID
JOIN restaurant.orders ON restaurant.purchasedmenuitem.OrderID = restaurant.orders.OrderID
WHERE MONTH(restaurant.orders.OrderDate) = MONTH(CURRENT_TIMESTAMP) AND YEAR(restaurant.orders.OrderDate) = YEAR(CURRENT_TIMESTAMP)
GROUP BY restaurant.menuitem.name
```

The third report shows a forecast for what to order for next month based on how well certain items are selling this month. For this particular restaurant we didn't have 100's of orders per month. We had between 5-20 sold each month so we used a constant of 1.5 times then rounded to the nearest integer. For future expansions we can create variables to make it easier for a user to change what the forecast could be.



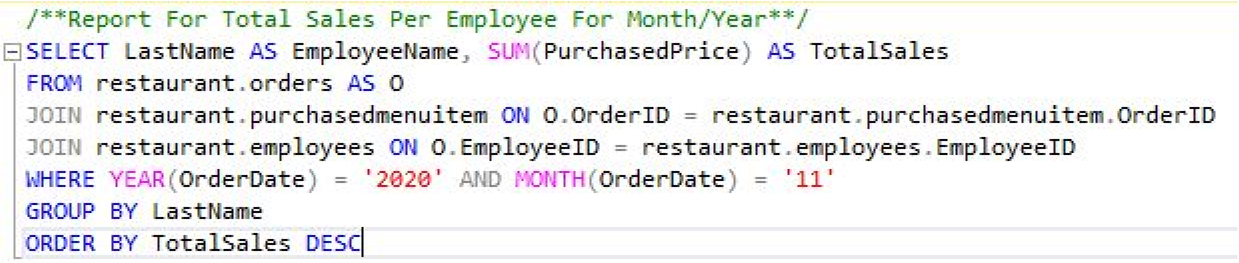
```
/**Forecast For Next Month's Inventory Ordering Using THis Month's Sold Items**/
SELECT restaurant.menuitem.name AS ItemName, COUNT(restaurant.purchasedmenuitem.MenuItemID) AS NumOrdered, ROUND(COUNT(restaurant.purchasedmenuitem.MenuItemID) * 1.5) AS Forecast
FROM restaurant.purchasedmenuitem
JOIN restaurant.menuitem ON restaurant.purchasedmenuitem.MenuItemID = restaurant.menuitem.MenuItemID
JOIN restaurant.orders ON restaurant.purchasedmenuitem.OrderID = restaurant.orders.OrderID
WHERE MONTH(restaurant.orders.OrderDate) = MONTH(NOW()) AND YEAR(restaurant.orders.OrderDate) = YEAR(NOW())
GROUP BY restaurant.purchasedmenuitem.MenuItemID
```



```
/**Forecast For Next Month's Inventory Ordering Using THis Month's Sold Items**/
SELECT restaurant.menuitem.name AS ItemName, COUNT(restaurant.purchasedmenuitem.MenuItemID) AS NumOrdered, ROUND(COUNT(restaurant.purchasedmenuitem.MenuItemID) * 1.5, 0) AS Forecast
FROM restaurant.purchasedmenuitem
JOIN restaurant.menuitem ON restaurant.purchasedmenuitem.MenuItemID = restaurant.menuitem.MenuItemID
JOIN restaurant.orders ON restaurant.purchasedmenuitem.OrderID = restaurant.orders.OrderID
WHERE MONTH(restaurant.orders.OrderDate) = MONTH(CURRENT_TIMESTAMP) AND YEAR(restaurant.orders.OrderDate) = YEAR(CURRENT_TIMESTAMP)
GROUP BY restaurant.menuitem.name
```

The fourth report shows the total sales per employee for each month or for the entire year depending on how you set up the where clause. We had it show the highest sellers for this month and year. But these can easily be changed with front end inputs.

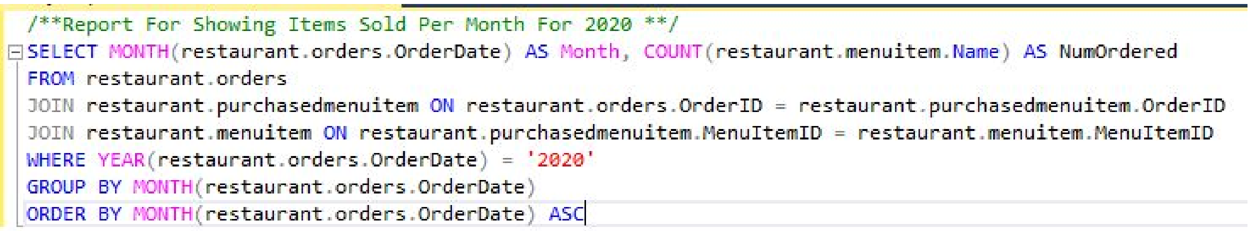
```
/**Report For Total Sales Per Employee For Month/Year**/
SELECT CONCAT(FirstName, " ", LastName) AS EmployeeName, SUM(PurchasedPrice) AS TotalSales
FROM restaurant.orders AS O
JOIN restaurant.purchasedmenuitem ON O.OrderID = restaurant.purchasedmenuitem.OrderID
JOIN restaurant.employees ON O.EmployeeID = restaurant.employees.EmployeeID
WHERE YEAR(OrderDate) = '2020' AND MONTH(OrderDate) = '11'
GROUP BY O.EmployeeID
ORDER BY TotalSales DESC
```



```
/**Report For Total Sales Per Employee For Month/Year**/
SELECT LastName AS EmployeeName, SUM(PurchasedPrice) AS TotalSales
FROM restaurant.orders AS O
JOIN restaurant.purchasedmenuitem ON O.OrderID = restaurant.purchasedmenuitem.OrderID
JOIN restaurant.employees ON O.EmployeeID = restaurant.employees.EmployeeID
WHERE YEAR(OrderDate) = '2020' AND MONTH(OrderDate) = '11'
GROUP BY LastName
ORDER BY TotalSales DESC
```

The fifth report shows the items sold per month for a given year. The year we have selected is 2020. That way we can keep the front end relevant with data.

```
/**Report For Showing Items Sold Per Month For 2020 **/
SELECT MONTH(restaurant.orders.OrderDate) AS Month, COUNT(restaurant.menuitem.Name) AS NumOrdered
FROM restaurant.orders
JOIN restaurant.purchasedmenuitem ON restaurant.orders.OrderID = restaurant.purchasedmenuitem.OrderID
JOIN restaurant.menuitem ON restaurant.purchasedmenuitem.MenuItemID = restaurant.menuitem.MenuItemID
WHERE YEAR(restaurant.orders.OrderDate) = '2020'
GROUP BY MONTH(restaurant.orders.OrderDate)
ORDER BY MONTH(restaurant.orders.OrderDate) ASC
```



```
/**Report For Showing Items Sold Per Month For 2020 **/
SELECT MONTH(restaurant.orders.OrderDate) AS Month, COUNT(restaurant.menuitem.Name) AS NumOrdered
FROM restaurant.orders
JOIN restaurant.purchasedmenuitem ON restaurant.orders.OrderID = restaurant.purchasedmenuitem.OrderID
JOIN restaurant.menuitem ON restaurant.purchasedmenuitem.MenuItemID = restaurant.menuitem.MenuItemID
WHERE YEAR(restaurant.orders.OrderDate) = '2020'
GROUP BY MONTH(restaurant.orders.OrderDate)
ORDER BY MONTH(restaurant.orders.OrderDate) ASC
```

Summary and Discussion

John - The original project only had a few endpoints and a single page that contained bulletin posts and employees that you could access after logging in. Now, the user can login or signup, in which they are taken by default to the bulletin page. The web app has been cut up into different pages, depending on what information you are looking for. For the backend, the relevant endpoints have been fleshed out to match what actions we wanted on the front end.

I learned the problems with rushing a web app. Not necessarily because of time constraint, but the lack of thinking things through and making it organized. I was hopping from task to task, and not cleaning things as I went.

For future work of this app, I would likely extend the app through other data interactions, to make sure all use cases for restaurants could be covered.

Alex- There are many changes we made when comparing to the project we presented. We updated the database to include tables for bulletin post comments, and orders. Additionally, we created a signup page, report page, and an employee summary page.

From this project, I learned a lot about using front-end tools to create a website that utilizes a database. For some time, I've had issues with this, and It was because I was not using the proper tools. This project made me understand the importance of time. In the outset of this project, I assumed it would not take very long to complete, but as soon as we got 'into the weeds of it', that was not true. If I were doing this project over again, I think I would start a little earlier to fully encapsulate the ideas that everyone had.

Further improvements to the website include the ability to delete accounts from a manager account, other reports, and more concepts revolving around the restaurant business, such as tips. I would have also liked to add the ability to edit, and place orders through this. I think this would have been interesting.

Jordan - For the most part with our database a lot of it is very similar to what we submitted at the beginning. We added more information into the Employees table and then got rid of the Shifts table. We originally were going to have it generate shifts based on availability but when looking into how to accomplish that it wouldn't have been feasible in the time frame for this project so we scrapped that idea. It turned into a Discrete Math problem trying to balance everyone's shifts and availability so we focused more on the user being able to log into the system and being able to create an account.

Through making this application I have learned a lot more about how to make reports and how to connect these reports through SQL. Also I have found it quite amazing being able to display SQL data in other ways than a table. I think it was a cool idea designing a project like this, because in the real world SQL is used as the database for many projects and seeing how it all works together was very cool.

Some of the improvements or additions to this project would definitely be adding in an inventory system so that you can have the entire restaurant ecosystem in one place. You can see what you have in stock as well as when you need to restock. I think this would also make forecasts a lot more accurate.

If I were to go back and change how we approached this project I would have used SQL Server for our database in the beginning instead of trying to migrate from MySQL to SQL Server.

Overall, we have added more functionality since our presentation, such as different reports in an easy to read interactive chart, a way to change user roles, and also a creation page to become a user. So I am pleased with our end product.