Alexander Ogay 6380727
GitHub Username: Alexander6380727
Minn Swan Pai 6480756
GitHub Username: swanduckquack
https://github.com/Alexander6380727/Database-FastFood-Project-.git

About the business: Our fast-food website is dedicated to providing customers with a seamless ordering experience, offering a diverse menu delivered straight to their chosen location from any of our numerous branches. Through our website, patrons can browse our selection, place orders, and enjoy their favorite meals from the comfort of their preferred setting.

Purposes: The system with these features:

Menu Management: Our system meticulously organizes details of all menu items, including names, descriptions, prices, ingredients, and customizable options, ensuring customers have a comprehensive view of our offerings.

Order Management: We diligently record incoming orders, monitor their status throughout the fulfillment process (such as In The Kitchen, Delivering, Delivered), and maintain a comprehensive order history for each customer, ensuring efficiency and transparency in our operations.

Customer Management: Central to our operations is the management of customer information, encompassing names, contact details, delivery addresses, and comprehensive order histories. This enables personalized service and facilitates seamless interactions with our valued patrons.

Delivery Management: Our system optimizes delivery routes, assigns delivery personnel, tracks delivery statuses in real-time, and efficiently manages delivery schedules, ensuring prompt and reliable service to our customers.

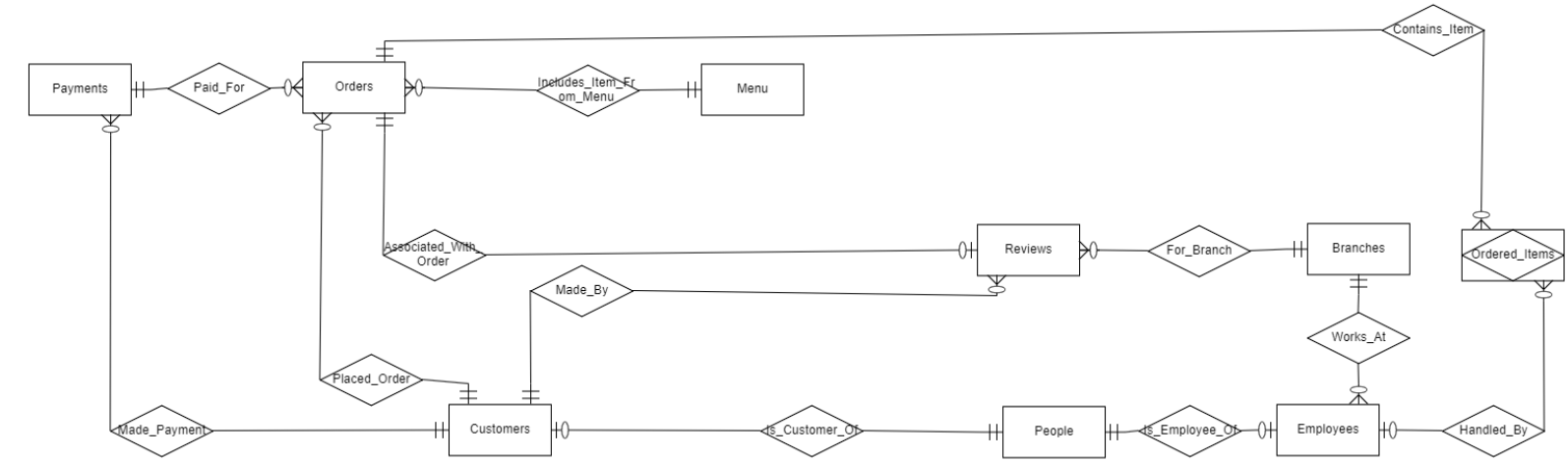
Payment Processing: With a focus on security and convenience, we meticulously record payment details, offer a variety of payment methods (such as credit cards, online wallets), and ensure the integrity of all transactions, providing customers with a hassle-free payment experience.

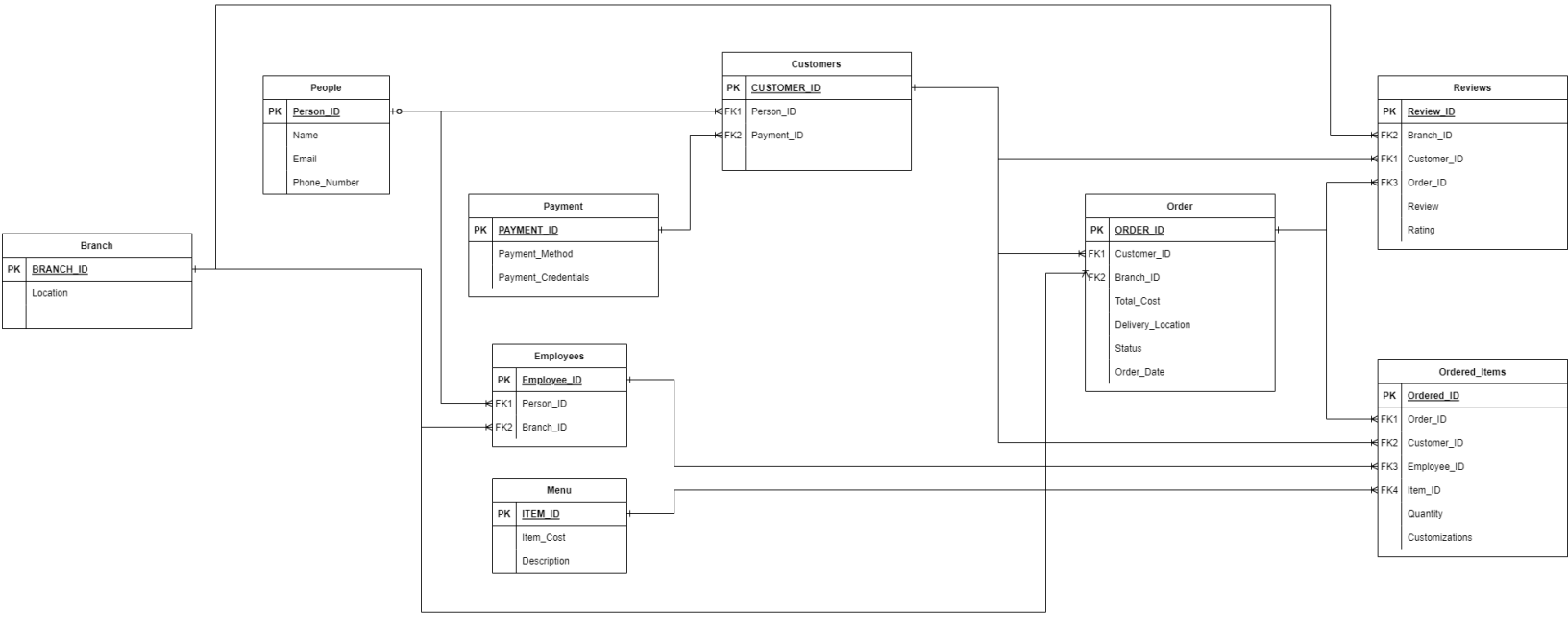
Branch Management: Across our diverse network of branches, our system streamlines operations, facilitating efficient management of each location, ensuring consistency in service quality and operational excellence across all branches.

Employee Management: Ensuring efficient handling of employee data, task assignments, and scheduling across branches. It facilitates seamless coordination and enables performance tracking

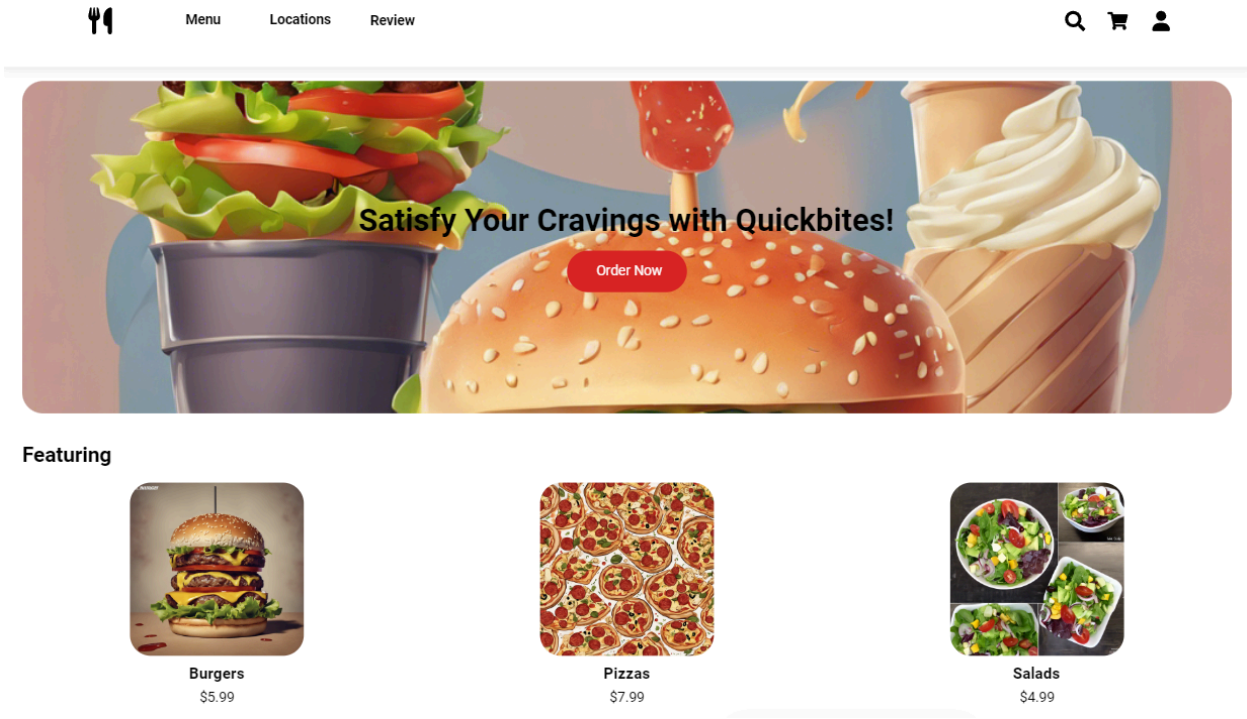
Review Management: enabling customers to provide feedback on their experiences. It tracks reviews linked to specific orders, branches, and customers, fostering transparency and accountability.

Will serve the business as mentioned above.





Front Page




```
SELECT item_id, item_cost, item_description FROM menu WHERE item_id IN (1, 2, 3);
```

item_id	item_cost	item_description
1	120	Garlic Chicken
2	220	Margarita Pizza
3	160	Chicken Burger

This is the front page of the website that customers first see when they enter the page and want to order food.
The owners would be able to put up featured foods onto the page in order to attract customers

Menu Page

Menu Locations Review

Burgers

View Items

Pizzas

View Items

Salads

View Items

Classic Cheeseburger

A juicy beef patty with melted cheese

$6.99

Add to Cart

Margherita Pizza

Classic pizza with tomato, mozzarella, and basil

$8.99

Add to Cart

Caesar Salad

Crisp romaine lettuce with Caesar dressing

$5.99

Add to Cart


```sql
CREATE TABLE menu (
    item_id SERIAL PRIMARY KEY,
    item_cost INTEGER,
    item_description CHAR(100) UNIQUE
);
SELECT * FROM menu;
SELECT *
FROM menu
WHERE item_description LIKE '%Burger%';
```

	item_id	item_cost	item_description
1	1	120	Garlic Chicken
2	2	220	Margarita Pizza
3	3	160	Chicken Burger
4	4	200	Beef Burger
5	5	180	Pork Burger
6	6	100	French Fries

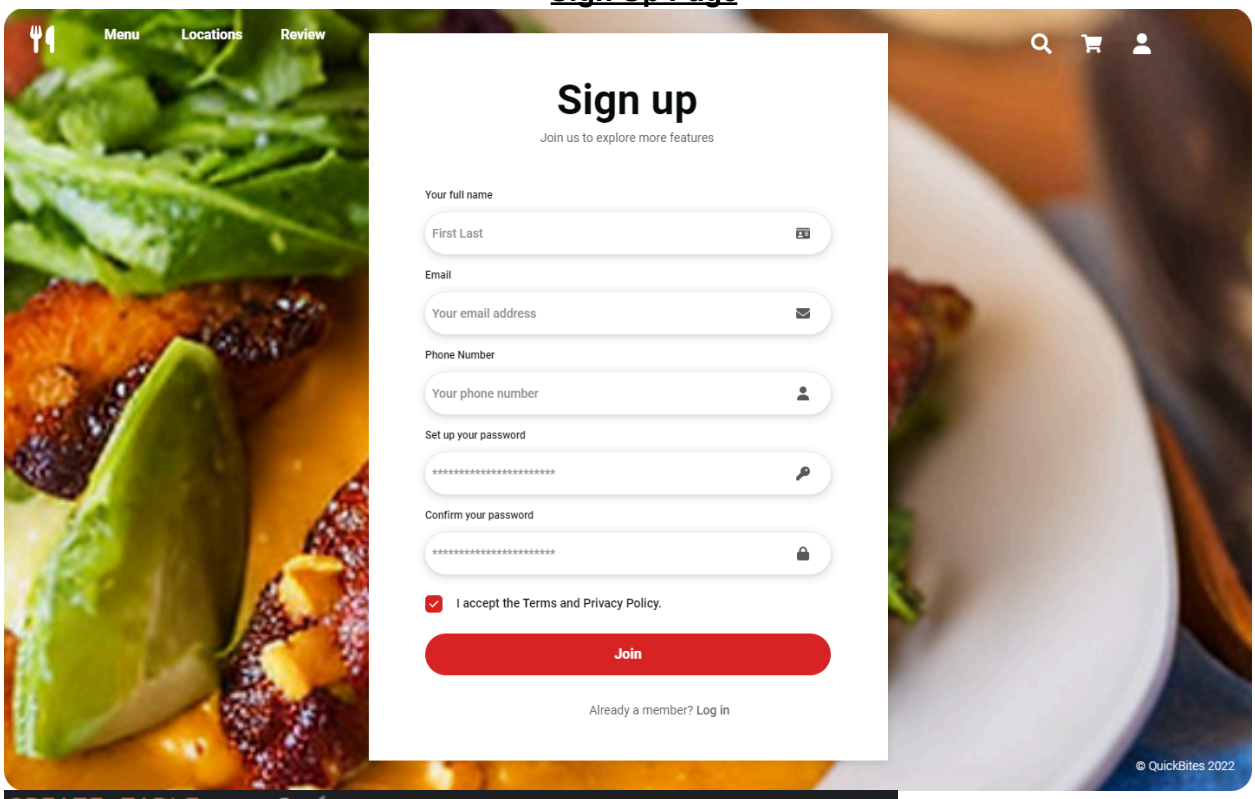
	item_id	item_cost	item_description
1	3	160	Chicken Burger
2	4	200	Beef Burger
3	5	180	Pork Burger

```
SELECT item_id, item_cost, item_description
FROM menu
WHERE item_cost > 200;
```

	item_id	item_cost	item_description
1	2	220	Margarita Pizza

This webpage allows customers to see what the restaurant is offering for food as well as the price for each meal.
The owners can browse the food on the menu and change the details of each menu item, as well as removing them or adding new items to the menu.

Sign Up Page



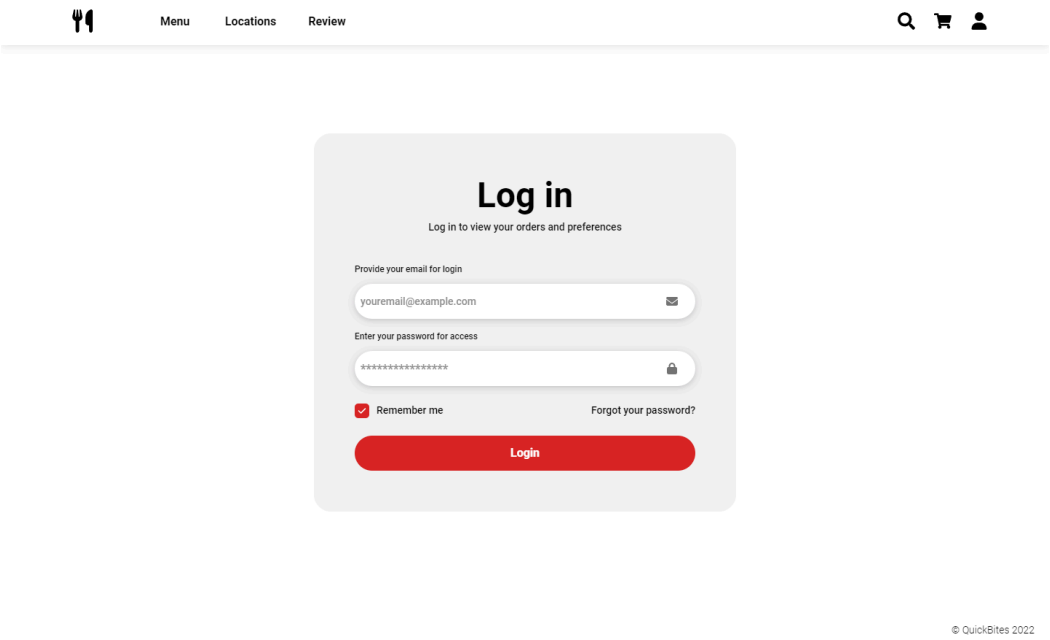

```sql
CREATE TABLE people(
    person_id SERIAL PRIMARY KEY,
    name CHAR(100),
    email VARCHAR(100) UNIQUE ,
    phone_number VARCHAR(20) UNIQUE
);

CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    person_id INTEGER REFERENCES people(person_id),
    payment_id INTEGER REFERENCES payments(payment_id)
);
SELECT *
FROM people
WHERE person_id IN (SELECT person_id FROM customers);
```

	person_id	name	email	phone_number
1	1	John Wick	1@gmail.com	0000000000
2	2	Paul Atreides	2@gmail.com	0000000001
3	3	Will Smith	3@gmail.com	0000000002
4	4	John Cena	4@gmail.com	0000000003
5	5	Michael Jackson	5@gmail.com	0000000004
6	6	Hulk Hogan	6@gmail.com	0000000005
7	7	Thomas Wayne	7@gmail.com	0000000006
8	8	Toby Maguire	8@gmail.com	0000000012

This webpage allows for any new customers to make an account and order food for themselves. The owners would be able to see the addition of new customers to their database and view their data.

Login page

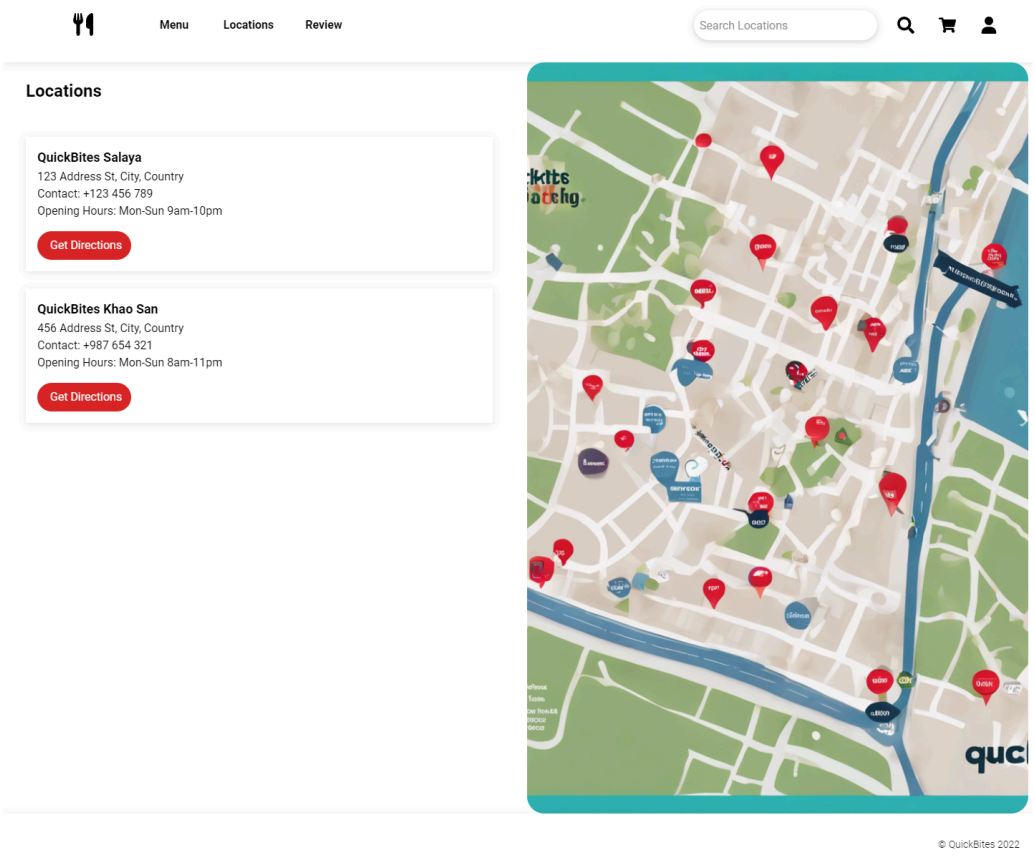



```
SELECT name,email FROM people
WHERE person_ID IN (SELECT person_id FROM CUSTOMERS);
```

	name	email
1	John Wick	1@gmail.com
2	Paul Atreides	2@gmail.com
3	Will Smith	3@gmail.com
4	John Cena	4@gmail.com
5	Michael Jackson	5@gmail.com
6	Hulk Hogan	6@gmail.com
7	Thomas Wayne	7@gmail.com
8	Toby Maguire	8@gmail.com
9	Jackie Chan	9@gmail.com
10	Robute Gulliman	10@gmail.com

Allows for customers that made an account to be able to log into the website and have all their saved information.
Owners can view the details of each customer by id.

Restaurant Branch Finder Page




```
SELECT * FROM branches;
```

	branch_id	branch_location
1	1	Tungsin
2	2	Salaya
3	3	Khao San
4	4	Phuket

```
SELECT order_id, customer_id, total_cost, delivery_location, status, branch_id, order_date
FROM orders
WHERE branch_id = 1;

SELECT review_id, customer_id, order_id, review, rating
FROM reviews
WHERE branch_id = 1;

SELECT e.employee_id, p.name, p.email, p.phone_number
FROM employees e
JOIN people p ON e.person_id = p.person_id
WHERE e.branch_id = 1;
```

	order_id	customer_id	total_cost	delivery_location	status	branch_id	order_date
1	1	1	900.00	123 Roadhouse	Delivered	1	2024-03-20
2	4	4	100.00	Spongebobs House	Delivered	1	2024-03-23
3	8	8	800.00	Home	In the Kitchen	1	2024-03-25

	review_id	customer_id	order_id	review	rating
1	1	1	1	Nice	5
2	4	4	4	Meh	2

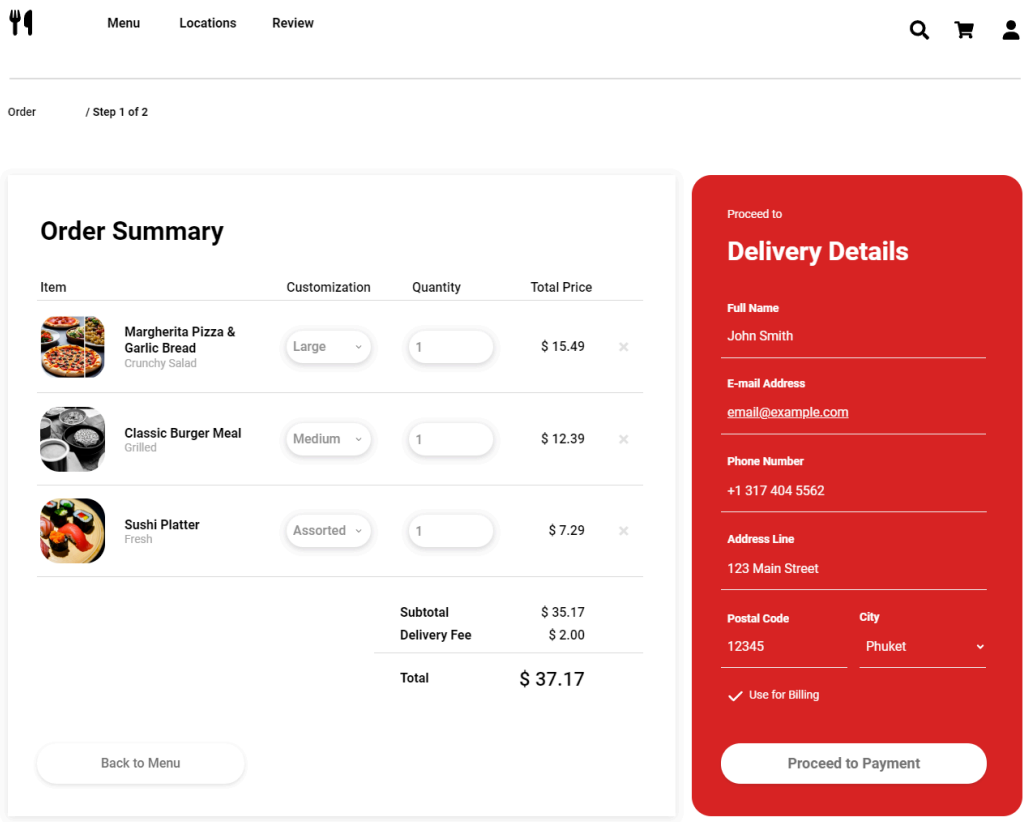
	employee_id	name	email	phone_number
1	1	Bruce Wayne	1@quickbites.com	0000000007

```
CREATE TABLE branches (
    branch_id SERIAL PRIMARY KEY,
    branch_location VARCHAR(200) UNIQUE NOT NULL
);

CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    person_id INTEGER NOT NULL REFERENCES people(person_id),
    branch_id INTEGER NOT NULL REFERENCES branches(branch_id)
);
```

Allows the customers to search for branches in their area.
Owners can keep track of the orders, reviews, employees for each of their branches
All the data they want can be seen for all branches or just one.

Order Summary Page




```sql
SELECT o.order_id, c.customer_id, p.name AS customer_name, p.email AS customer_email, p.phone_number,
       oi.item_id, m.item_description, oi.quantity, oi.customizations,
       o.total_cost, o.delivery_location, o.status, o.branch_id, b.branch_location, o.order_date
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN people p ON c.person_id = p.person_id
JOIN ordered_items oi ON o.order_id = oi.order_id
JOIN menu m ON oi.item_id = m.item_id
JOIN branches b ON o.branch_id = b.branch_id;
```

	order_id	customer_id	customer_name	customer_email	phone_number	item_id	item_description
1	1	1	John Wick	1@gmail.com	0000000000	1	Garlic Chicken
2	1	1	John Wick	1@gmail.com	0000000000	2	Margarita Pizza
3	2	2	Paul Atreides	2@gmail.com	0000000001	3	Chicken Burger
4	3	3	Will Smith	3@gmail.com	0000000002	4	Beef Burger
5	3	3	Will Smith	3@gmail.com	0000000002	5	Pork Burger
6	4	4	John Cena	4@gmail.com	0000000003	6	French Fries
7	5	5	Michael Jackson	5@gmail.com	0000000004	1	Garlic Chicken
8	6	6	Michael Jackson	5@gmail.com	0000000004	2	Margarita Pizza
9	7	7	Hulk Hogan	6@gmail.com	0000000005	3	Chicken Burger
10	8	8	Thomas Wayne	7@gmail.com	0000000006	4	Beef Burger
11	9	9	Toby Maguire	8@gmail.com	0000000012	5	Pork Burger
12	10	10	Jackie Chan	9@gmail.com	0000000013	6	French Fries

quantity	customizations	total_cost	delivery_location	status	branch_id	branch_location	order_date
2	None	900.00	123 Roadhouse	Delivered	1	Tungsin	2024-03-20
3	None	900.00	123 Roadhouse	Delivered	1	Tungsin	2024-03-20
1	None	160.00	Ram Ranch	Delivered	2	Salaya	2024-03-21
2	None	1120.00	Electric Avenue	Delivered	3	Khao San	2024-03-22
4	None	1120.00	Electric Avenue	Delivered	3	Khao San	2024-03-22
1	None	100.00	Spongebobs House	Delivered	1	Tungsin	2024-03-23
2	None	240.00	Hell	Delivered	2	Salaya	2024-03-24
2	None	440.00	Antarctica	Delivering	3	Khao San	2024-03-25
1	None	160.00	Electric Boogaloo	Delivering	3	Khao San	2024-03-25
4	None	800.00	Home	Delivered	1	Tungsin	2024-03-25
2	None	360.00	Space	In the Kitchen	2	Salaya	2024-03-25
1	None	100.00	Ocean	In the Kitchen	3	Khao San	2024-03-25

```sql
SELECT ordered_id, order_id, item_id, quantity, customizations
FROM ordered_items
WHERE quantity > 1;
```

	ordered_id	order_id	item_id	quantity	customizations
1	1	1	1	2	None
2	2	1	2	3	None
3	4	3	4	2	None
4	5	3	5	4	None
5	7	5	1	2	None
6	8	6	2	2	None
7	10	8	4	4	None

```sql
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    customer_id INTEGER NOT NULL REFERENCES customers(customer_id),
    total_cost DECIMAL(10, 2),
    delivery_location VARCHAR(200) NOT NULL,
    status CHAR(50) NOT NULL,
    branch_id INTEGER NOT NULL REFERENCES branches(branch_id),
    order_date DATE NOT NULL
);
```

```sql
CREATE TABLE ordered_items (
    ordered_id SERIAL PRIMARY KEY,
    order_id INTEGER NOT NULL REFERENCES orders(order_id),
    customer_id INTEGER NOT NULL REFERENCES customers(customer_id),
    employee_id INTEGER NOT NULL REFERENCES employees(employee_id),
    item_id INTEGER NOT NULL REFERENCES menu(item_id),
    quantity INTEGER NOT NULL,
    customizations VARCHAR(50)
);
```

```sql
UPDATE orders o
SET total_cost = (
    SELECT SUM(oi.quantity * m.item_cost)
    FROM ordered_items oi
    JOIN menu m on oi.item_id = m.item_id
    WHERE oi.order_id = o.order_id
    );
```

The page is used by the customers to see all the information in their cart, their orders and the total amount they need to pay as well as the place where they can type in their location details. Owners can see the quantity of food per customer and what food they ordered, and update the final cost as necessary.

Payment Page

Menu Locations Review

Payment Details

Name on Card | Card Number

Expiration Date | CVV

Billing Address

☐ Save card information for future orders

Select Payment Method:

Credit/Debit Card | PayPal | Apple Pay

Order Summary

Item		Qty
Price	Burger	2
$10.99		

Subtotal:	$21.98
Taxes:	$1.50
Delivery Fee:	$3.00
Total:	**$26.48**

Complete Transaction

Complete Order

Total Amount: $26.48
Estimated Delivery Time: 30 minutes

```sql
SELECT o.order_id, c.customer_id, p.name AS customer_name,
       oi.item_id, m.item_description, oi.quantity, oi.customizations,
    o.delivery_location, o.status, o.order_date,
       py.payment_id, py.payment_method, py.payment_credentials, o.total_cost
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN people p ON c.person_id = p.person_id
JOIN ordered_items oi ON o.order_id = oi.order_id
JOIN menu m ON oi.item_id = m.item_id
JOIN payments py ON c.payment_id = py.payment_id;
```

order_id	customer_id	customer_name	item_id	item_description	quantity	customizations	delivery_location	status	order_date
1	1	John Wick	1	Garlic Chicken	2	None	123 Roadhouse	Delivered	2024-03-20
1	1	John Wick	2	Margarita Pizza	3	None	123 Roadhouse	Delivered	2024-03-20
2	2	Paul Atreides	3	Chicken Burger	1	None	Ram Ranch	Delivered	2024-03-21
3	3	Will Smith	4	Beef Burger	2	None	Electric Avenue	Delivered	2024-03-22
3	3	Will Smith	5	Pork Burger	4	None	Electric Avenue	Delivered	2024-03-22
4	4	John Cena	6	French Fries	1	None	Spongebobs House	Delivered	2024-03-23
5	5	Michael Jackson	1	Garlic Chicken	2	None	Hell	Delivered	2024-03-24
6	6	Michael Jackson	2	Margarita Pizza	2	None	Antarctica	Delivering	2024-03-25
7	7	Hulk Hogan	3	Chicken Burger	1	None	Electric Boogaloo	Delivering	2024-03-25
8	8	Thomas Wayne	4	Beef Burger	4	None	Home	Delivered	2024-03-25
9	9	Toby Maguire	5	Pork Burger	2	None	Space	In the Kitchen	2024-03-25
10	10	Jackie Chan	6	French Fries	1	None	Ocean	In the Kitchen	2024-03-25

payment_id	payment_method	payment_credentials	total_cost
1	QR Payment	gsfdbvcFGVC	900.00
1	QR Payment	gsfdbvcFGVC	900.00
2	Credit Card	asfddgs@ES	160.00
3	Debit Card	DGFVS$#%	1120.00
3	Debit Card	DGFVS$#%	1120.00
4	PayPal	@#$%@E2345	100.00
5	Credit Card	sfge=45-6	240.00
5	Credit Card	sfge=45-6	440.00
6	QR Payment	346thdfgh	160.00
7	PayPal	@#%$$Wdf	800.00
8	Credit Card	hdgfs#$@!	360.00
9	Debit Card	sdfesf!@#@!	100.00

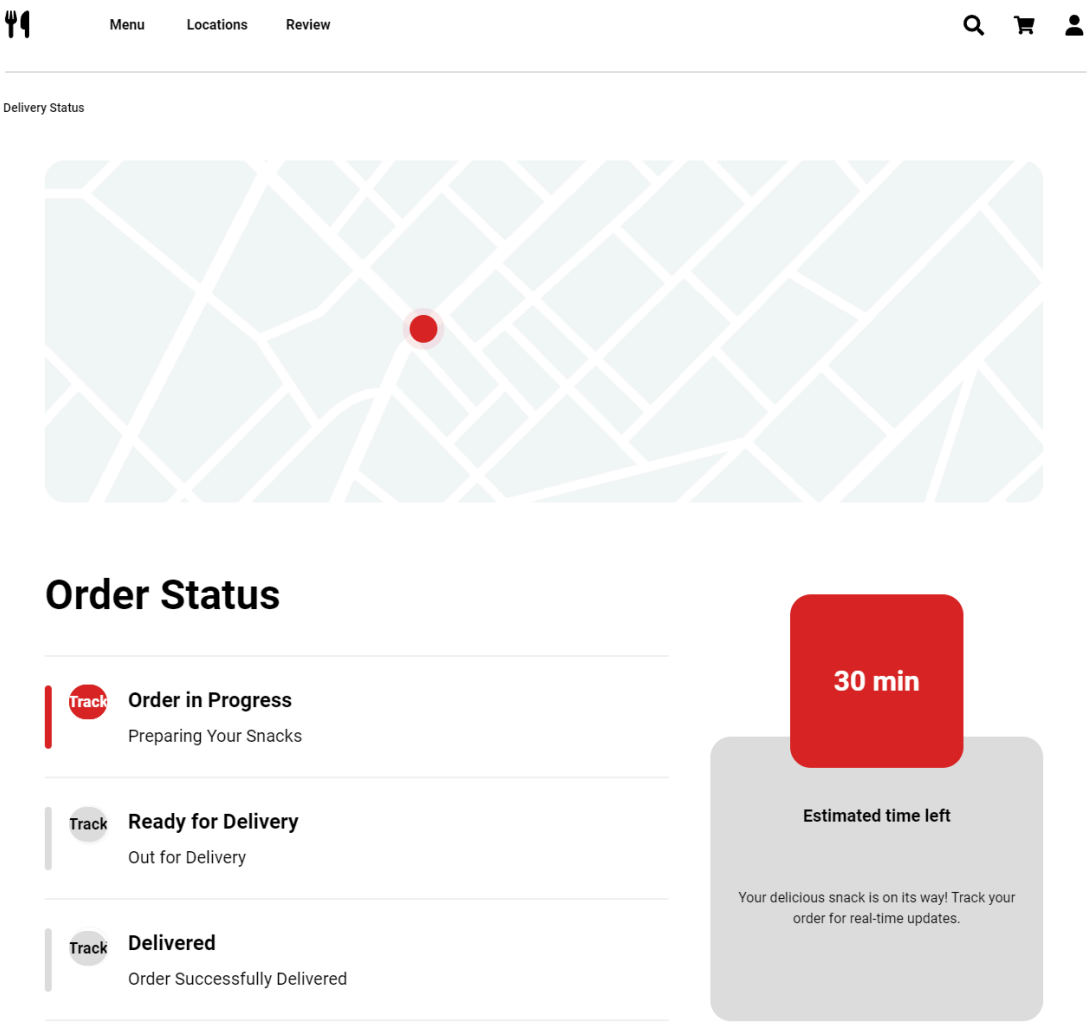
```sql
SELECT p.payment_method, p.payment_credentials
FROM customers c
JOIN people pe ON c.person_id = pe.person_id
JOIN payments p ON c.payment_id = p.payment_id
WHERE c.customer_id = 1;
```

	payment_method	payment_credentials
1	QR Payment	gsfdbvcFGVC

```sql
CREATE TABLE payments (
    payment_id SERIAL PRIMARY KEY,
    payment_method CHAR(50) NOT NULL,
    payment_credentials VARCHAR(100) NOT NULL
);
```

Allows the customers to enter their payment details to make their orders.
The owners would be able to see the payment details for customers as well as what payment was used for what order.

Delivery Page




```sql
SELECT o.order_id, c.customer_id, p.name AS customer_name, p.email AS customer_email, p.phone_number,
       o.total_cost, o.delivery_location, o.status, o.order_date
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN people p ON c.person_id = p.person_id;
```

order_id	customer_id	customer_name	customer_email	phone_number	total_cost	delivery_location	status	order_date
1	1	John Wick	1@gmail.com	0000000000	900.00	123 Roadhouse	Delivered	2024-03-20
2	2	Paul Atreides	2@gmail.com	0000000001	160.00	Ram Ranch	Delivered	2024-03-21
3	3	Will Smith	3@gmail.com	0000000002	1120.00	Electric Avenue	Delivered	2024-03-22
4	4	John Cena	4@gmail.com	0000000003	100.00	Spongebobs House	Delivered	2024-03-23
5	5	Michael Jackson	5@gmail.com	0000000004	240.00	Hell	Delivered	2024-03-24
6	6	Michael Jackson	5@gmail.com	0000000004	440.00	Antarctica	Delivering	2024-03-25
7	7	Hulk Hogan	6@gmail.com	0000000005	160.00	Electric Boogaloo	Delivering	2024-03-25
9	9	Toby Maguire	8@gmail.com	0000000012	360.00	Space	In the Kitchen	2024-03-25
10	10	Jackie Chan	9@gmail.com	0000000013	100.00	Ocean	In the Kitchen	2024-03-25
8	8	Thomas Wayne	7@gmail.com	0000000006	800.00	Home	Delivered	2024-03-25

```sql
SELECT order_id, delivery_location, status
FROM orders
WHERE status = 'Delivered';

SELECT oi.ordered_id, oi.order_id, oi.item_id, oi.quantity, oi.customizations
FROM ordered_items oi
JOIN orders o ON oi.order_id = o.order_id
WHERE oi.order_id = 1;
```

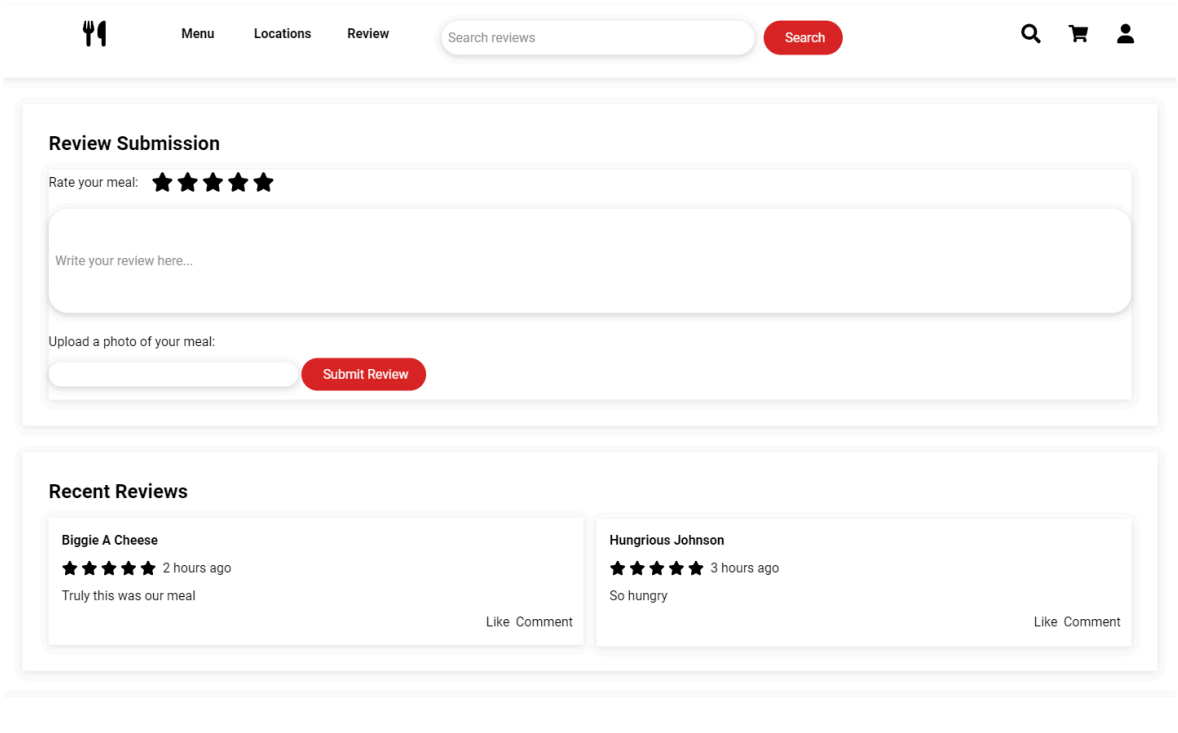
	order_id	delivery_location	status
1	1	123 Roadhouse	Delivered
2	2	Ram Ranch	Delivered
3	3	Electric Avenue	Delivered
4	4	Spongebobs House	Delivered
5	5	Hell	Delivered

	ordered_id	order_id	item_id	quantity	customizations
1	1	1	1	2	None
2	2	1	2	3	None

```sql
CREATE OR REPLACE FUNCTION update_order_status(p_order_id INT, new_status TEXT)
RETURNS VOID AS $$
BEGIN
    UPDATE orders
    SET status = new_status
    WHERE order_id = p_order_id;
END;
$$ LANGUAGE plpgsql;
```

Allows the customer to see the status of their order, whether it is in the kitchen, being delivered or already delivered.
Owners are able to see the status of the orders going on, as well as the customers that are working on what order and the item that is going into each order overall or by branch if needed.

Review Page




```sql
SELECT r.review_id, c.customer_id, p.name AS customer_name, p.email AS customer_email, p.phone_number,
       r.branch_id, b.branch_location, r.order_id, r.review, r.rating
FROM reviews r
JOIN customers c ON r.customer_id = c.customer_id
JOIN people p ON c.person_id = p.person_id
JOIN branches b ON r.branch_id = b.branch_id;
```

review_id	customer_id	customer_name	customer_email	phone_number
1	1	John Wick	1@gmail.com	0000000000
2	2	Paul Atreides	2@gmail.com	0000000001
3	3	Will Smith	3@gmail.com	0000000002
4	4	John Cena	4@gmail.com	0000000003
5	5	Michael Jackson	5@gmail.com	0000000004

branch_id	branch_location	order_id	review	rating
1	Tungsin	1	Nice	5
2	Salaya	2	Alright	4
3	Khao San	3	Mid	3
1	Tungsin	4	Meh	2
2	Salaya	5	Terrible	1

```sql
SELECT r.review_id, r.customer_id, r.branch_id, r.order_id, r.review, r.rating
FROM reviews r
JOIN customers c ON r.customer_id = c.customer_id
WHERE r.rating = 4;
```

	review_id	customer_id	branch_id	order_id	review	rating
1	2	2	2	2	Alright	4

```sql
SELECT r.review_id, r.customer_id, r.branch_id, r.order_id, r.review, r.rating
FROM reviews r
JOIN customers c ON r.customer_id = c.customer_id
WHERE r.rating = 4;

SELECT review_id, customer_id, branch_id, order_id, review, rating
FROM reviews
WHERE customer_id = 1;

SELECT review_id, customer_id, branch_id, order_id, review, rating
FROM reviews
WHERE branch_id = 1;
```

```sql
CREATE TABLE reviews (
    review_id SERIAL PRIMARY KEY,
    customer_id INTEGER NOT NULL REFERENCES customers(customer_id),
    branch_id INTEGER NOT NULL REFERENCES branches(branch_id),
    order_id INTEGER NOT NULL REFERENCES orders(order_id),
    review TEXT NOT NULL,
    Rating INTEGER CHECK (Rating >= 1 AND Rating <= 5) NOT NULL
);
```

This page allows the customer to send their review in.
The database allows for the owners to view the reviews given by all customers across multiple branches.
They are able to view reviews per customer, per branch, per rating, and specifically single out specific orders by the order id.

The provided SQL script creates a database schema for managing a restaurant system. Here's a summary of each table and the script's purpose:

People: Stores information about individuals interacting with the system, such as customers and employees. It includes fields for a unique identifier (person_id), name, email, and phone number.

Payments: Stores information about payment methods and their credentials. It includes fields for a unique identifier (payment_id), payment method, and payment credentials.

Customers: Associates people with their payment methods. It includes fields for a unique identifier (customer_id), references to the person and payment tables (person_id, payment_id).

Menu: Represents the items available on the restaurant's menu. It includes fields for a unique identifier (item_id), item cost, and item description.

Branches: Represents the branches or locations of the restaurant. It includes fields for a unique identifier (branch_id) and branch location.

Employees: Stores information about restaurant employees. It includes fields for a unique identifier (employee_id), references to the person and branch tables (person_id, branch_id).

Orders: Stores information about customer orders. It includes fields for a unique identifier (order_id), references to the customer, branch, and ordered items tables (customer_id, branch_id, ordered_items), total cost, delivery location, order status, and order date.

Ordered Items: Represents the items ordered by customers in each order. It includes fields for a unique identifier (ordered_id), references to the order, customer, employee, and menu tables (order_id, customer_id, employee_id, item_id), quantity, and customizations.

Reviews: Stores reviews given by customers for their orders. It includes fields for a unique identifier (review_id), references to the customer, branch, and order tables (customer_id, branch_id, order_id), review text, and rating.

Additionally, the script includes an SQL UPDATE statement to calculate the total cost of each order. This calculation is based on the sum of the quantities of ordered items multiplied by their respective item costs. The total cost is then updated in the orders table.

Overall, the script sets up a comprehensive database schema for managing various aspects of a restaurant system, including customer and employee information, menu items, orders, payments, and reviews.

Criticism & Suggestion:
The database schema could benefit from further normalization to reduce redundancy and improve data integrity. For example, the ordered_items table could include only the order_id and item_id, with a separate table for item details. This would avoid duplication of item information and ensure consistency.
Data Integrity, ensuring that the same customer cannot be associated with multiple payment methods simultaneously could improve data quality.
Data Types, using VARCHAR for phone numbers may not be ideal, as it allows for non-numeric characters. Using numeric types like INTEGER may be more suitable.
Scalability, ensure that the database schema is designed to handle increasing data volumes efficiently.
Security: Have to be more careful when storing the payment credentials. It would be better not to store sensitive information like this directly in the database.

In our current system, each customer is connected to one associated payment method, which might not accurately represent how the system would actually work (in the case that one person has multiple payment methods)