CS 387 PROJECT

RESTAURANT MANAGEMENT SUITE

180050002 Abhinav Goud
180050030 Dorna Vineeth
180050042 Jatoth Shiva Tarun
180050050 Kondra Rishyanth

Requirements:

Our system has 5 different roles - Manager, Chef, Receptionist, Delivery Boy and Customer. They have to login using their google accounts. Manager has control over all the employees in the restaurant. Chef has control over the menu related stuff. All the employees can switch mode from customer to employee in the app.

Customer Requirements:

- Can signup or login into the app and logout from the app
- Can add/remove dishes to the cart from the menu
- Can order dishes from the cart by selecting the address and completing the payment
- Can see their previous orders and status of pending orders
- Can reserve tables at the restaurant online
- Can switch to employee mode if he works in the restaurant
- Can provide feedback on the service and rate the dishes ordered
- Can view and update his/her details and add/update addresses connected with his/her account
- Can see recommendations based on previous orders

Manager Requirements:

- Can view the menu
- Can see the successful, pending and failed transactions between a start and end date
- Can see the successful, pending and failed purchases between a start and end date
- Can see the statistics of dishes ordered and ingredients used between a start and end date
- Can recruit new employees and give them a role and wage
- Can see all the employees working in the restaurant and edit employee details except addresses

Chef Requirements:

- Can add or delete dishes to the menu
- Can add new ingredients to the list of ingredients
- Can update the price of a dish
- Can view or update the ingredients of a dish
- Can view the pending and current orders at that moment
- Can confirm or reject orders placed by the customers

Receptionist Requirements:

- Can view the menu
- Can see the bookings for today

Delivery Boy Requirements:

- Can see the orders to be delivered by him

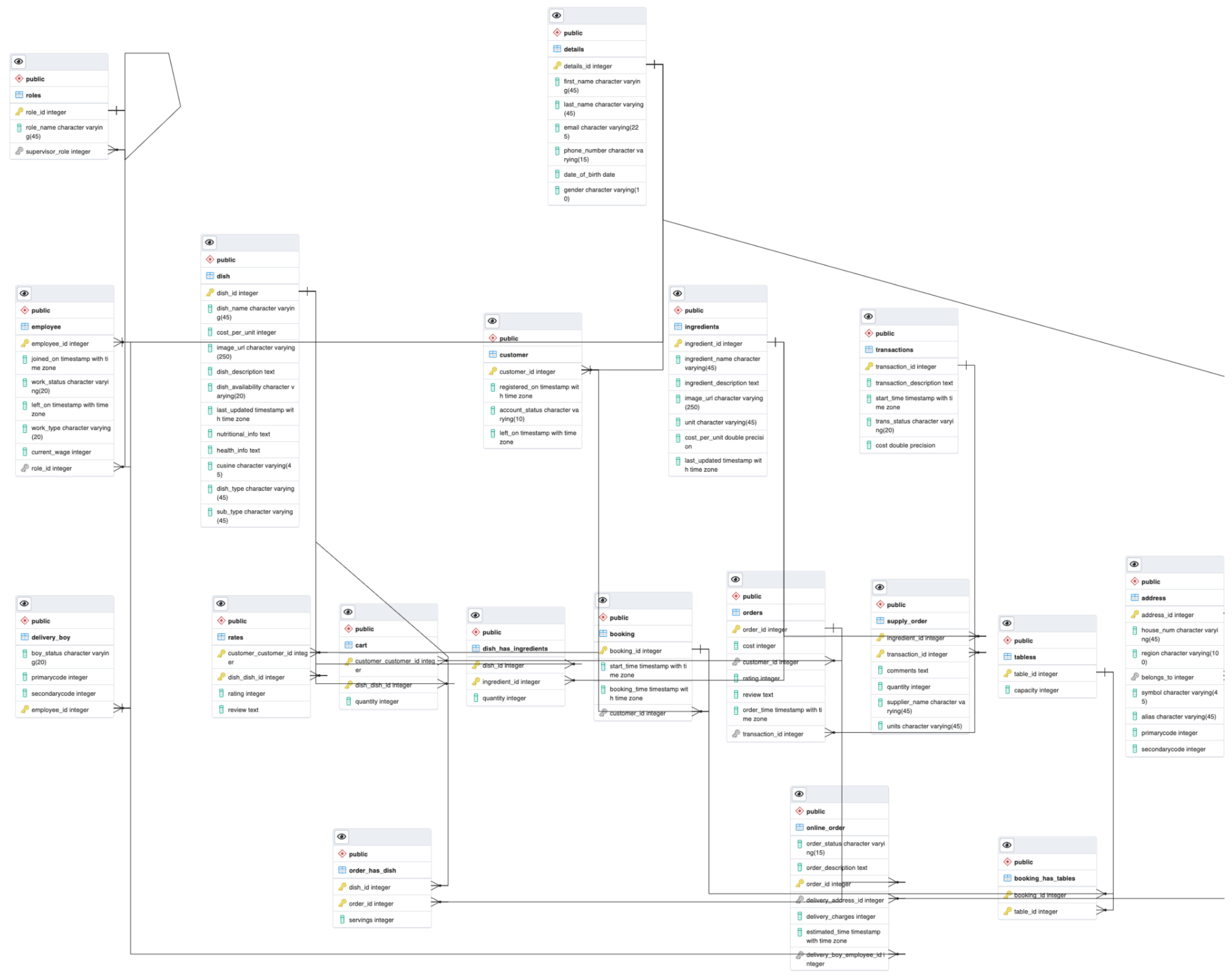
Use Cases:

1. **Logging in :** Framework for signup, login and logout for the user
2. **Browsing Menu :** The user(Customer, Chef, Receptionist and Manager) can browse the menu and use the special buttons (based on the role of user) on requirement
3. **Booking Tables :** The customer can book vacant tables by mentioning the date and time. Once booked, he cannot book the tables again. Maximum capacity for booking is 20. Booking slots are 1 Hr intervals that start between 11:00AM and 09:00PM. (payment process is dropped)
4. **Rating Services :** The customer can give feedback on the service of a particular order in the previous orders page
5. **Rating Dishes :** The customer can give feedback on the dish and rate the dishes that the customer ordered in the previous orders page
6. **Browsing Orders :** The customer can see all his pending and previous orders
7. **Adding Address :** The customer can add the address to which the online order has to be delivered while placing the order
8. **Viewing User Details :** The customer can view all his details, update them except for his email; can add, delete and update many addresses which are used for deliveries.
9. **Adding to Cart :** The customer can add/remove dishes from cart
10. **Confirming the order:** The chef can confirm the order placed by a customer before payment
11. **Rejecting the order:** The chef can reject the order placed by a customer before payment
12. **Allotting delivery boys to orders:** The chef can allot the order to a delivery boy for delivering it

13. **Recruitment of employees :** The manager can recruit new employees, set his role, wage, work type and create an employee account for him
14. **Editing dishes in Menu :** The chef can add new dishes or delete or update the price of existing dishes in the menu
15. **Editing dish Ingredients :** The chef can add new ingredients or update the count of existing ingredient
16. **Cooking and Usage Stats :** The manager can see the stats of the dishes cooked and ingredients used between a start and end date
17. **Ingredients Purchased :** The manager can see successful, pending and failed purchases between a start and end date
18. **Transactions :** The manager can see successful, pending and failed transactions between a start and end date
19. **Editing Employee details :** The manager can edit the employee details such as set his role, wage, work type
20. **Viewing Booked tables :** The receptionist can see the bookings for that day
21. **Viewing orders :** The delivery boy can see the orders to be delivered by him

Design:

Database Design:



public
roles
role_id integer
role_name character varying(45)
supervisor_role integer
public
details
details_id integer
first_name character varying(45)
last_name character varying (45)
email character varying(225)
phone_number character varying(15)
date_of_birth date
gender character varying(10)
public
dish
dish_id integer
dish_name character varying(45)
cost_per_unit integer
image_url character varying (250)
dish_description text
dish_availability character varying(20)
last_updated timestamp with time zone
nutritional_info text
health_info text
cusine character varying(45)
dish_type character varying (45)
sub_type character varying (45)
public
employee
employee_id integer
joined_on timestamp with time zone
work_status character varying(20)
left_on timestamp with time zone
work_type character varying (20)
current_wage integer
role_id integer
public
customer
customer_id integer
registered_on timestamp with time zone
account_status character varying(10)
left_on timestamp with time zone
public
ingredients
ingredient_id integer
ingredient_name character varying(45)
ingredient_description text
image_url character varying (250)
unit character varying(45)
cost_per_unit double precision
last_updated timestamp with time zone
public
transactions
transaction_id integer
transaction_description text
start_time timestamp with time zone
trans_status character varying(20)
cost double precision
public
delivery_boy
boy_status character varying(20)
primarycode integer
secondarycode integer
employee_id integer
public
rates
customer_customer_id integer
dish_dish_id integer
rating integer
review text
public
cart
customer_customer_id integer
dish_dish_id integer
quantity integer
public
dish_has_ingredients
dish_id integer
ingredient_id integer
quantity integer
public
booking
booking_id integer
start_time timestamp with time zone
booking_time timestamp with time zone
customer_id integer
public
orders
order_id integer
cost integer
customer_id integer
rating integer
review text
order_time timestamp with time zone
transaction_id integer
public
supply_order
ingredient_id integer
transaction_id integer
comments text
quantity integer
supplier_name character varying(45)
units character varying(45)
public
tabless
table_id integer
capacity integer
public
address
address_id integer
house_num character varying(45)
region character varying(100)
belongs_to integer
symbol character varying(45)
alias character varying(45)
primarycode integer
secondarycode integer
public
order_has_dish
dish_id integer
order_id integer
servings integer
public
online_order
order_status character varying(15)
order_description text
order_id integer
delivery_address_id integer
delivery_charges integer
estimated_time timestamp with time zone
delivery_boy_employee_id integer
public
booking_has_tables
booking_id integer
table_id integer

Normalization:

Same as above

Final Schema:

```
-- -----------------------------------------------------
-- Schema public
-- -----------------------------------------------------
SET timezone = 'Asia/Kolkata';

DROP TABLE IF EXISTS Supply_Order ;
DROP TABLE IF EXISTS Order_has_dish ;
DROP TABLE IF EXISTS Online_order ;
DROP TABLE IF EXISTS Delivery_Boy ;
DROP TABLE IF EXISTS Orders ;
DROP TABLE IF EXISTS Rates ;
DROP TABLE IF EXISTS Cart ;
DROP TABLE IF EXISTS Dish_has_Ingredients ;
DROP TABLE IF EXISTS Ingredients ;
DROP TABLE IF EXISTS Dish ;
DROP TABLE IF EXISTS Booking_has_Tables ;
DROP TABLE IF EXISTS Booking ;
DROP TABLE IF EXISTS Transactions ;
DROP TABLE IF EXISTS Tabless ;
DROP TABLE IF EXISTS Address ;
DROP TABLE IF EXISTS Employee ;
DROP TABLE IF EXISTS Roles ;
DROP TABLE IF EXISTS Customer ;
DROP TABLE IF EXISTS Details ;
-- -----------------------------------------------------
-- Table public.Details : BCNF, fixed
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Details (
  details_id serial PRIMARY KEY,
  first_name VARCHAR(45) NOT NULL,
  last_name VARCHAR(45) NOT NULL,
  email VARCHAR(225) NOT NULL UNIQUE,
  phone_number VARCHAR(15),
  date_of_birth DATE,
  gender VARCHAR(10) CHECK(gender IN ('other','male','female'))
);

-- -----------------------------------------------------
-- Table public.Customer : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Customer (
  customer_id INT PRIMARY KEY,
  registered_on TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  account_status VARCHAR(10) NOT NULL DEFAULT 'active',
  left_on TIMESTAMPTZ,
  CONSTRAINT Customer_Details
    FOREIGN KEY (customer_id)
    REFERENCES public.Details (details_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Account_Status_Check
    CHECK (account_status IN ('active','inactive','deleted'))
```

```
);

-- -----------------------------------------------------
-- Table public.Roles : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Roles (
  role_id serial PRIMARY KEY,
  role_name VARCHAR(45) NOT NULL,
  supervisor_role INT,
  CONSTRAINT Role_Supervisor
    FOREIGN KEY (supervisor_role)
    REFERENCES public.Roles (role_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- CREATE INDEX Role_Supervisor_idx ON public.Role (supervisor_role ASC) VISIBLE;

-- -----------------------------------------------------
-- Table public.Employee : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Employee ( -- change this make details as details_id and make p.k remove employee_id
  employee_id  INT PRIMARY KEY,
  joined_on TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  work_status VARCHAR(20) NOT NULL,
  left_on TIMESTAMPTZ , -- change this
  work_type VARCHAR(20) NOT NULL,
  current_wage INT NOT NULL,
  role_id INT NOT NULL,
  CONSTRAINT Employee_Details
    FOREIGN KEY (employee_id)
    REFERENCES public.Details (details_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Employee_Role
    FOREIGN KEY (role_id)
    REFERENCES public.Roles (role_id)
    ON DELETE SET NULL
    ON UPDATE CASCADE,
  CONSTRAINT Work_Status_Check
    CHECK (work_status IN ('suspended','active','leave','vacation','fired','reserve')),
  CONSTRAINT Work_Type_Check
    CHECK (work_type IN ('permanent','temporary','internship'))
);


-- -----------------------------------------------------
-- Table public.Address : BCNF
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Address (
  address_id serial PRIMARY KEY,
  house_num VARCHAR(45) NOT NULL,
  region VARCHAR(100) NOT NULL, -- Eg: Colony and Area,etc
  belongs_to INT NOT NULL,
  symbol VARCHAR(45) NOT NULL, -- redundant predifined symbols
  alias VARCHAR(45) NOT NULL,
  primaryCode INT,
  secondaryCode INT,
  CONSTRAINT Address_Details
    FOREIGN KEY (belongs_to)
    REFERENCES public.Details (details_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE INDEX "Address_Details_idx" ON public.Address USING btree(belongs_to);

-- -----------------------------------------------------
-- Table public.Tabless : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Tabless (
  table_id serial PRIMARY KEY,
  capacity INT NOT NULL
);
-- -----------------------------------------------------
-- Table public.Transactions : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Transactions (
  transaction_id serial PRIMARY KEY,
  transaction_description TEXT NOT NULL DEFAULT '',
  start_time TIMESTAMPTZ NOT NULL,
  trans_status VARCHAR(20) NOT NULL,
  cost DOUBLE PRECISION NOT NULL,
  CONSTRAINT Trans_Status_Check
    CHECK (trans_status IN ('failed','pending','successful'))
);

-- -----------------------------------------------------
-- Table public.Booking : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Booking (
  booking_id serial PRIMARY KEY,
  start_time TIMESTAMPTZ NOT NULL,
  booking_time TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  customer_id INT NOT NULL,
  CONSTRAINT Booking_Customer
    FOREIGN KEY (customer_id)
    REFERENCES public.Customer (customer_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- CREATE INDEX Booking_Customer_idx ON public.Booking (customer_id ASC) VISIBLE;

-- CREATE INDEX Booking_Transaction_idx ON public.Booking (transaction_id ASC) VISIBLE;

-- -----------------------------------------------------
-- Table public.Booking_has_Tables : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Booking_has_Tables (
  booking_id INT NOT NULL,
  table_id INT NOT NULL,
  PRIMARY KEY (booking_id, table_id),
  CONSTRAINT Booking_has_Tables_Booking
    FOREIGN KEY (booking_id)
    REFERENCES public.Booking (booking_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Booking_has_Tables_Tables
    FOREIGN KEY (table_id)
    REFERENCES public.Tabless (table_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- CREATE INDEX Booking_has_Tables_Tables_idx ON public.Booking_has_Tables (table_id ASC) VISIBLE;

-- CREATE INDEX Booking_has_Tables_Booking_idx ON public.Booking_has_Tables (booking_id ASC) VISIBLE;

-- -----------------------------------------------------
-- Table public.Dish : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Dish (
  dish_id serial PRIMARY KEY,
  dish_name VARCHAR(45) NOT NULL,
  cost_per_unit INT NOT NULL,
  --currency VARCHAR(10) NOT NULL, -- Redundancy
  image_url VARCHAR(250) NOT NULL,
  dish_description TEXT NOT NULL,
  dish_availability VARCHAR(20) NOT NULL DEFAULT 'available',
  last_updated TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  nutritional_info TEXT,
  health_info TEXT,
  cusine VARCHAR(45) NOT NULL,
  dish_type VARCHAR(45) NOT NULL,
  sub_type VARCHAR(45) NOT NULL,
  CONSTRAINT Dish_Availability_Check
    CHECK (dish_availability IN ('available','out of stock','unavailable')),
  CONSTRAINT Cusine_Check
    CHECK (cusine IN ('Indian','Italian','American','Fusion')),
  CONSTRAINT Dish_Type_Check
    CHECK (dish_type IN ('Vegetarian','Non Vegetarian','Vegan'))
);

-- -----------------------------------------------------
-- Table public.Ingredients : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Ingredients (
  ingredient_id serial PRIMARY KEY,
  ingredient_name VARCHAR(45) NOT NULL,
  ingredient_description TEXT NOT NULL,
  image_url VARCHAR(250) NOT NULL,
  unit VARCHAR(45) NOT NULL,
  cost_per_unit DOUBLE PRECISION NOT NULL,
  last_updated TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

-- -----------------------------------------------------
-- Table public.Dish_has_Ingredients : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Dish_has_Ingredients (
  dish_id INT NOT NULL,
  ingredient_id INT NOT NULL,
  quantity INT NOT NULL,
  PRIMARY KEY (dish_id, ingredient_id),
  CONSTRAINT Dish_has_Ingredients_Dish
    FOREIGN KEY (dish_id)
    REFERENCES public.Dish (dish_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Dish_has_Ingredients_Ingredients
    FOREIGN KEY (ingredient_id)
    REFERENCES public.Ingredients (ingredient_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Quantity_Check
    CHECK (quantity >= 0)
);

CREATE INDEX "Dish_has_Ingredients_Ingredients_idx" ON public.Dish_has_Ingredients USING btree(ingredient_id,dish_id);

-- -----------------------------------------------------
-- Table public.Cart : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Cart (
  Customer_customer_id INT NOT NULL,
  Dish_dish_id INT NOT NULL,
  Quantity INT NOT NULL,
  PRIMARY KEY (Customer_customer_id, Dish_dish_id),
  CONSTRAINT Customer_has_Dish_Customer
    FOREIGN KEY (Customer_customer_id)
    REFERENCES public.Customer (customer_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Customer_has_Dish_Dish
    FOREIGN KEY (Dish_dish_id)
    REFERENCES public.Dish (dish_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Quantity_Check
    CHECK (quantity > 0)
);


-- -----------------------------------------------------
-- Table public.Rates : BCNF, looks good
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS public.Rates (
  customer_customer_id INT NOT NULL, -- changed name here
  dish_dish_id INT NOT NULL, -- here too
  rating INT,
  review TEXT,
  PRIMARY KEY (customer_customer_id,dish_dish_id),
  CONSTRAINT Customer_has_Dish_Customer
    FOREIGN KEY (customer_customer_id)
    REFERENCES public.Customer (customer_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Customer_has_Dish_Dish
    FOREIGN KEY (dish_dish_id)
    REFERENCES public.Dish (dish_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Rating_Check
    CHECK (rating <= 5 OR rating >= 0)
);

CREATE INDEX "Rates_Customer_has_Dish_Dish_idx" ON public.Rates USING btree(dish_dish_id,customer_customer_id);

-- -----------------------------------------------------
-- Table public.Order : BCNF, looks good
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Orders (
  order_id serial PRIMARY KEY,
  cost INT NOT NULL,
  customer_id INT NOT NULL,
  rating INT NULL,
  review TEXT NULL,
  order_time TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  transaction_id INT NOT NULL,
  CONSTRAINT Order_Customer
    FOREIGN KEY (customer_id)
    REFERENCES public.Customer (customer_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Order_Transaction
    FOREIGN KEY (transaction_id)
    REFERENCES public.Transactions (transaction_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Rating_Check
    CHECK (rating <= 5 OR rating >= 0)
);

CREATE INDEX "Order_Customer_idx" ON public.Orders USING btree(customer_id);

-- CREATE INDEX Order_Transaction_idx ON public.Orders (transaction_id ASC) VISIBLE;

-- -----------------------------------------------------
-- Table public.Delivery_Boy : BCNF, looks good
-- specialization of Employee
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS public.Delivery_Boy (
  boy_status VARCHAR(20) NOT NULL,
  primaryCode INT NOT NULL,
  secondaryCode INT,
  employee_id INT NOT NULL,
  PRIMARY KEY (employee_id),
  CONSTRAINT Delivery_Boy_Employee
    FOREIGN KEY (employee_id)
    REFERENCES public.Employee (employee_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Boy_Status_Check
    CHECK (boy_status IN ('On delivery','Free'))
);

-- -----------------------------------------------------
-- Table public.Online_order : BCNF, looks good
-- specialization of Orders
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS public.Online_order (
  order_status VARCHAR(15) NOT NULL,
  order_description TEXT,
  order_id INT NOT NULL,
  delivery_address_id INT NOT NULL,
  delivery_charges INT NOT NULL,
  estimated_time TIMESTAMPTZ ,
  delivery_boy_employee_id INT,
  PRIMARY KEY (order_id),
  CONSTRAINT Online_order_Order
    FOREIGN KEY (order_id)
    REFERENCES public.Orders (order_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Online_order_Address
    FOREIGN KEY (delivery_address_id)
    REFERENCES public.Address (address_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Online_order_Delivery_Boy
    FOREIGN KEY (delivery_boy_employee_id)
    REFERENCES public.Delivery_Boy (employee_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Order_Status_Check
    CHECK (order_status IN ('confirmed','rejected','pending','paid','delivered','on_way'))
);

CREATE INDEX "Online_order_Address_idx" ON public.Online_order USING  btree(delivery_address_id,delivery_boy_employee_id);

-- CREATE INDEX Offline_order_Employee_idx ON public.Offline_order (waiter_employee_id ASC) VISIBLE;

-- -----------------------------------------------------
-- Table public.Order_has_dish : BCNF, looks good
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS public.Order_has_dish (
  dish_id INT NOT NULL,
  order_id INT NOT NULL,
  servings INT NOT NULL,
  PRIMARY KEY (dish_id, order_id),
  CONSTRAINT Dish_has_Order_Dish
    FOREIGN KEY (dish_id)
    REFERENCES public.Dish (dish_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Dish_has_Order_Order
    FOREIGN KEY (order_id)
    REFERENCES public.Orders (order_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Servings_Check
    CHECK (servings > 0)
);

CREATE INDEX "Dish_has_Order_Order_idx" ON public.Order_has_dish USING btree(order_id,dish_id ASC);
-- -----------------------------------------------------
-- Table public.Supply_Order : BCNF, looks good
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS public.Supply_Order (
  ingredient_id INT NOT NULL,
  transaction_id INT NOT NULL,
  comments TEXT NULL,
  quantity INT NOT NULL,
  supplier_name VARCHAR(45) NOT NULL,
  units VARCHAR(45) NOT NULL,
  PRIMARY KEY (ingredient_id, transaction_id),
  CONSTRAINT Ingredients_has_Transaction_Ingredients
    FOREIGN KEY (ingredient_id)
    REFERENCES public.Ingredients (ingredient_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Ingredients_has_Transaction_Transaction
    FOREIGN KEY (transaction_id)
    REFERENCES public.Transactions (transaction_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT Quantity_Check
    CHECK (quantity > 0));
```

Technology Choices:

We used nodejs framework for the backend and ejs for the front end. Our database model is relational so we used Postgresql for database management since over the duration of the course, we became familiar with psql in relational types.

Performance Improvement Measures:

1. Denormalization:

The data was already in the denormalized form. So we didn't have to denormalize it again.

2. Indexing :

```
CREATE INDEX "Address_Details_idx" ON public.Address USING btree(belongs_to);
CREATE INDEX "Dish_has_Ingredients_Ingredients_idx" ON public.Dish_has_Ingredients USING
btree(ingredient_id,dish_id);
CREATE INDEX "Rates_Customer_has_Dish_Dish_idx" ON public.Rates USING
btree(Dish_dish_id,Customer_customer_id);
CREATE INDEX "Order_Customer_idx" ON public.Orders USING btree(customer_id);
CREATE INDEX "Online_order_Address_idx" ON public.Online_order USING
btree(delivery_address_id,delivery_boy_employee_id);
CREATE INDEX "Dish_has_Order_Order_idx" ON public.Order_has_dish USING btree(order_id,dish_id ASC);
```

Test Results

Use Case id	**Use case**	**Input**	**Output**	**Results**
8.	Adding to cart	Quantity of dish (positive number)	Successfully added to cart message	PASS
8	Adding to cart	Quantity of dish (0)	Redirect to same menu	PASS
12	Allotting delivery boy for an order By manager	Delivery boy (available)	Redirect to Orders pending to allot page	PASS

12	Allotting delivery boy for an order By manager	Delivery boy (not available)	Error message showing delivery boy busy with an order	HANDLED BY SHOWING STATUS OF DELIVERY BOYS
14	Recruiting employees	Email id, Role, Wage, area_code (All valid fields)	Successfully added employee	PASS
14	Recruiting employees	Name, Phno, DOB, Email id, Role, Wage, area_code (At least one in valid field)	Error message of invalid field	PASS
14	Removing employee	NA	Successfully removed employee	NOT IMPLEMENTED
2	View menu	NA	Menu page displayed	PASS
5	View previous orders	NA	Previous orders page displayed	PASS
1.	Logging in	Email and password	NA	PASS
4.	Providing feedback	feedback->text and/or Rating -> stars	Thank you for your feedback message	PASS
4.	Providing feedback	No input	Error message asking to enter feedback	PASS
6.	Adding address	House number, region, primary code, secondary code	User details page displayed which contains all addresses	PASS
6.	Adding address	House number, region, primary code, secondary code	Error message displayed	PASS
7.	Viewing user details	NA	User details page displayed	PASSS

9.	Viewing announcements and offers	NA	Announcements page displayed	NOT IMPLEMENTED
10.	Paying for the order	Card number, cvv, expiry date with balance required for transaction	Payment successful message	PASS
10.	Paying for the order	Invalid information	Error message displayed	PASS
11.	Cancelling the order	NA	Amount refunded and order cancelled	NOT IMPLEMENTED
20.	Can view all the transactions online and offline.	NA	Transaction details page displayed	OFFLINE ORDERS NOT IMPLEMENTED, ONLINE ORDERS PASS
15.	Updating occupancy in realtime Displaying tables based on occupancy	Start date-time and Duration. (Tables available)	Display all the tables available with their capacities during duration at the given date-time.	NOT IMPLEMENTED
15.	Updating occupancy in realtime Displaying tables based on occupancy	Start date-time and Duration. (Tables are not available)	“Sorry all tables are reserved” message is displayed.	NOT IMPLEMENTED
3.	Confirming selected Tables and paying	Number of people, tables chosen, start date and time, duration Successful payement on api	Redirect to payement gateway (wait till payement is confirmed and display successful/ error after payment). Message displayed “booking success”	PAYMENT NOT IMPLEMENTED, BOOKING PASS
3.	Confirming selected Tables and paying	Number of people, tables chosen, start date and time, duration, Check failed (table capacity is less than people)	Message displayed “failure” based on check	PAYMENT NOT IMPLEMENTED, BOOKING PASS
3.	Confirming selected Tables and paying	Number of people, tables chosen, start date and time, duration, Payement failed on api	Message displayed “failure” based on payement message	PAYMENT NOT IMPLEMENTED, BOOKING PASS

16.	Adding a new dish	Dish_name, cost_per_unit, image_url, dish_description Dropdown : cuisine , type, sub_type. Ingredients: (addable like a cart of ingredients) Optional: nutritional_info, health_info, Defaults: Last_updated_on, dish_availablity (VALID INPUT)	Message “Dish successfully added”	PASS
16.	Adding a new dish	Dish_name, cost_per_unit, image_url, dish_description Dropdown : cuisin , type, sub_type. Ingredients: (addable like a cart of ingredients) Optional: nutritional_info, health_info, Defaults: Last_updated_on, dish_availablity (INVALID INPUT - repeating dish_name or check constarints fail)	Message “Dish cannot be added + {error message}”	PASS
16.	Deleting a dish (Not suggested - update availability instead. Delete only when completely useless dish with no relations)	NA (on clicking delete button)	Message “Successfully deleted the dish”	PASS
16.	Updating a dish	Dish_name, cost_per_unit, image_url, dish_description Dropdown : cuisine , type, sub_type. Ingredients: (editable like a cart of ingredients) Optional: nutritional_info, health_info, Defaults: Last_updated_on, dish_availablity (VALID INPUT)	Message “Dish successfully updated”	PASS
16.	Updating a dish	Dish_name, cost_per_unit, image_url, dish_description Dropdown : cuisin , type, sub_type. Ingredients: (editable like a cart of ingredients) Optional: nutritional_info, health_info, Defaults: Last_updated_on, dish_availablity	Message “Dish cannot be updated + {error message}”	PASS

		(INVALID INPUT - repeating dish_name or check constarints fail)		
17.	Adding a new ingredient	Ingredient_name,ingredient_description,image_url,cost_per_unit Dropdown: unit Default : Last_updated (VALID INPUT)	Message “Ingredient successfully added”	PASS
17.	Adding new ingredient	Ingredient_name,ingredient_description,image_url,cost_per_unit, unit Default : Last_updated (INVALID INPUT repeating name)	Message “Ingredient cannot be added + {error message}”	PASS
17.	Deleting an existing ingredient	NA (on clicking delete button) (No dish has this ingredient)	Message “Successfully deleted the ingredient”	NOT IMPLEMENTED
17.	Deleting an existing ingredient	NA (on clicking delete button) (Fails because a dish has this ingredient)	Message “Ingredient cannot be deleted”	NOT IMPLEMENTED
17.	Updating an existing ingredient	Ingredient_name,ingredient_description,image_url,cost_per_unit Dropdown: unit Default : Last_updated (VALID INPUT)	Message “Ingredient successfully updated”	NOT IMPLEMENTED
17.	Updating an existing ingredient	Ingredient_name,ingredient_description,image_url,cost_per_unit, unit Default : Last_updated (INVALID INPUT repeating name)	Message “Ingredient cannot be updated + {error message}”	NOT IMPLEMENTED
18.	View dishes cooked on the day(s)	start date, end date	Dishes prepared and corresponding quanitities	PASS
18.	View Ingredients used on the day(s)	Start date, End date	Ingredients used and corresponding quanities	PASS

19.	View Ingredients purchased on the day(s)	Start date, End date	Ingredients purchased and corresponding quantity	PASS

Conclusion

The above mentioned **use cases and test cases** which have “PASS” in results were successfully implemented and tested.

The above mentioned **test cases** which have “NOT IMPLEMENTED” in results were successfully not-implemented and not-tested.

The following use cases were not implemented:

Usecaseid. Title - Reason

9. Viewing announcements and offers - Reason: Didn’t get much of an interest in displaying them

13. Area code allotment - Reason: Focused mainly on restaurant site part, so didn’t care much about area codes which are a focus of delivery part

15. Updating table occupancy in real time - Not implemented because we focused on online orders rather than offline setting. To avoid the table booking clash with offline customers, online customers should book tables at least one-day beforehand.

20. Can view all the transactions online and offline - Haven’t implemented offline orders because we focused on online orders

Link

https://github.com/rishyanthkondra/restaurantApp