Department of Computer Science

Report of Exam

Sector: AI and System distributed

Design and Development of Restaurant Management Application

Year: 2025-2026

Realized By: Abdelkebir Bouchti

Delivered to: Loubna Aminou

ENSET, Avenue Hassan II

Mohammedia – Maroc

Site Web: www.enset-media.ac.ma

E-Mail : enset-media@enset-media.ac.ma

Content :

Introduction

In the modern era of digital transformation, the restaurant industry is increasingly leveraging technology to enhance customer experience and streamline operations. One of the key advancements in this domain is the development of customized ordering systems that allow customers to personalize their meals according to their preferences. This project aims to design and implement a comprehensive application for restaurant ordering terminals, focusing on the customization of ingredients and supplements for ordered dishes. As the lead software developer, the task involves the complete conception and implementation of this application.

The application will be divided into several key components, each responsible for managing different aspects of the restaurant's operations. The first part of the project focuses on the management of customized dishes, which includes the following:

1. **Client Management**:
   - A `Client` class will be implemented to model customer information.
   - A `ClientDAO` class will handle CRUD (Create, Read, Update, Delete) operations for clients.
   - Each client will have the ability to place multiple orders.

2. **Main Dishes Management**:
   - A `PlatPrincipal` class will be created to model the information of a main dish.
   - A `PlatPrincipalDAO` class will manage CRUD operations for main dishes.
   - The `PlatPrincipal` class will include functionality to calculate the price of a dish based on its composition.

3. **Ingredients Management**:
   - An `Ingredient` class will be implemented to model ingredient information.
   - An `IngredientDAO` class will handle CRUD operations for ingredients.
   - Ingredients can be used in multiple dishes, with specific quantities defined for each dish.

4. **Supplements Management**:
   - A `Supplement` class will be created to model supplement information.
   - A `SupplementDAO` class will manage CRUD operations for supplements.
   - A meal can include multiple supplements.

5. **Meals and Orders Management**:

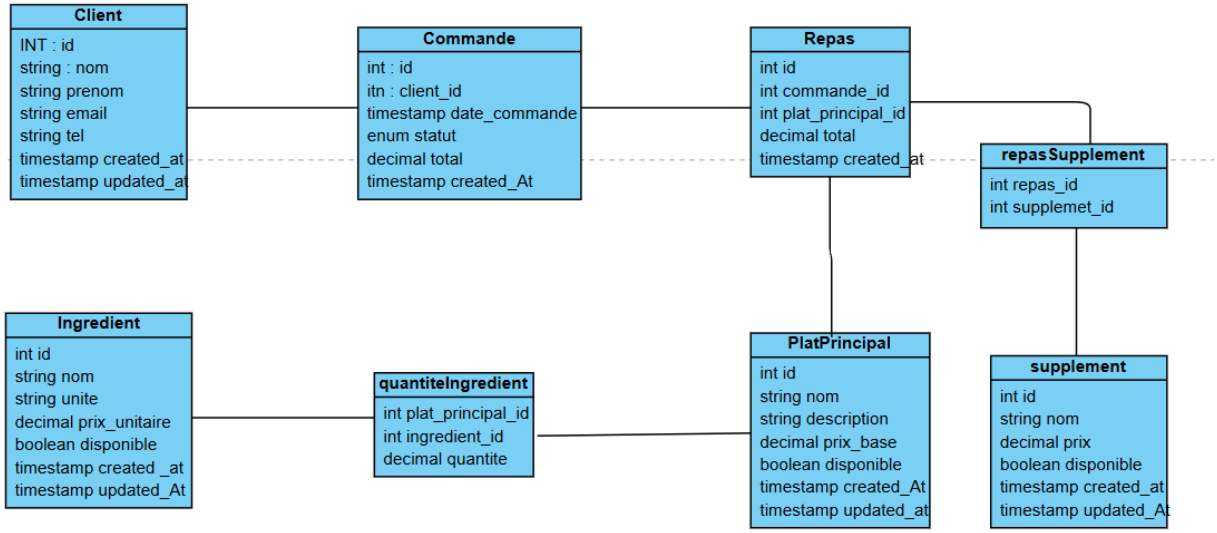
- A `Repas` class will be implemented to model a meal, which consists of a main dish, ingredients, and supplements.
- The `Repas` class will calculate the total cost of the meal, including ingredients and supplements.
- A `Commande` class will model order information.
- The `Commande` class will calculate the total cost of the order by summing the prices of all meals.

To enhance the user experience, the application will also include graphical user interfaces (GUIs) developed using JavaFX. These interfaces will allow customers to:

1. Select a meal from a graphical menu.
2. Customize their main dish by choosing ingredients and supplements.
3. View real-time updates of the total cost as ingredients and supplements are added or removed.
4. Save their customized meal and receive a printed ticket upon completion.

This project not only aims to improve the efficiency of restaurant operations but also to provide a personalized and interactive experience for customers, ultimately leading to higher satisfaction and loyalty. The following sections will delve into the detailed design and implementation of each component, ensuring a robust and user-friendly application.

Diagram de class



MLD

Table Name	Attributes	Constraints/Relationships
Client	`id` (INT, PK, Auto-Increment), `nom` (VARCHAR(50), NOT NULL), `prenom` (VARCHAR(50), NOT NULL), `email` (VARCHAR(100), UNIQUE, NOT NULL), `telephone` (VARCHAR(20)), `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP), `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)	- Primary Key: `id`.
Commande	`id` (INT, PK, Auto-Increment), `client_id` (INT, FK), `date_commande` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP), `statut` (ENUM, NOT NULL, DEFAULT 'en_attente'), `total` (DECIMAL(10,2), NOT NULL, DEFAULT 0.00), `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)	- Primary Key: `id`. - Foreign Key: `client_id` references `Client(id)`.

PlatPrincipal	`id` (INT, PK, Auto-Increment), `nom` (VARCHAR(100), NOT NULL), `description` (TEXT), `prix_base` (DECIMAL(10,2), NOT NULL), `disponible` (BOOLEAN, DEFAULT TRUE), `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP), `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)	- Primary Key: `id`.
Ingredient	`id` (INT, PK, Auto-Increment), `nom` (VARCHAR(100), NOT NULL), `unite` (VARCHAR(20), NOT NULL), `prix_unitaire` (DECIMAL(10,2), NOT NULL), `disponible` (BOOLEAN, DEFAULT TRUE), `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP), `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)	- Primary Key: `id`.

QuantiteIngredient	`plat_principal_id` (INT, FK), `ingredient_id` (INT, FK), `quantite` (DECIMAL(10,3), NOT NULL)	- Composite Primary Key: (`plat_principal_id`, `ingredient_id`). - Foreign Key: `plat_principal_id` references `PlatPrincipal(id)`, `ingredient_id` references `Ingredient(id)`.
Supplement	`id` (INT, PK, Auto-Increment), `nom` (VARCHAR(100), NOT NULL), `prix` (DECIMAL(10,2), NOT NULL), `disponible` (BOOLEAN, DEFAULT TRUE), `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP), `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)	- Primary Key: `id`.

Supplement	`id` (INT, PK, Auto-Increment), `nom` (VARCHAR(100), NOT NULL), `prix` (DECIMAL(10,2), NOT NULL), `disponible` (BOOLEAN, DEFAULT TRUE), `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP), `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)	- Primary Key: `id`.
Repas	`id` (INT, PK, Auto-Increment), `commande_id` (INT, FK), `plat_principal_id` (INT, FK), `total` (DECIMAL(10,2), NOT NULL, DEFAULT 0.00), `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)	- Primary Key: `id`. - Foreign Keys: `commande_id` references `Commande(id)`, `plat_principal_id` references `PlatPrincipal(id)`.
RepasSupplement	`repas_id` (INT, FK), `supplement_id` (INT, FK)	- Composite Primary Key: (`repas_id`, `supplement_id`). - Foreign Keys: `repas_id` references `Repas(id)`, `supplement_id` references `Supplement(id)`.

Justification of Design Choices

1. Nature of Associations Between Classes

- **Client → Commande (Order):**
  - **Type:** Simple association (1:n).
  - **Justification:** A client can place multiple orders (1:n relationship). This association is directional because an order must know its client, but a client doesn't necessarily need to know all their orders.
- **Commande → Repas (Meal):**
  - **Type:** Simple association (1:n).
  - **Justification:** An order can include multiple meals (1:n relationship). The association is directional from Commande to Repas, as an order needs to track the meals it contains.
- **PlatPrincipal (Main Dish) ↔ Ingredient:**
  - **Type:** Association class (QuantiteIngredient).
  - **Justification:** This is an n:n relationship with an additional attribute (quantite), requiring an association class. A main dish can include multiple ingredients, and an ingredient can be used in multiple dishes.
- **Repas ↔ Supplement:**

    - **Type:** Association class (RepasSupplement).
    - **Justification:** This is an n:n relationship because multiple supplements can be added to a meal, and a supplement can be part of multiple meals.
- **Commande → Client and Repas → PlatPrincipal:**
    - **Type:** Simple associations.
    - **Justification:** Directional, as an order belongs to a client, and a meal is based on a main dish, but the inverse relationships are not necessary.

---

2. Role of Attributes and Methods in Each Class

- **Client:**
    - **Attributes:** nom, prenom, email, telephone (to identify and contact the client).
    - **Methods:** placeOrder() to create an order linked to the client.
- **Commande (Order):**
    - **Attributes:** date_commande, statut, total (to track the order and its status).
    - **Methods:** addMeal(), updateStatus() to manage meals and order status.
- **PlatPrincipal (Main Dish):**
    - **Attributes:** nom, description, prix_base, disponible (characteristics of the dish).
    - **Methods:** addIngredient(), calculatePrice() to handle ingredients and determine the final price.
- **Ingredient:**
    - **Attributes:** nom, unite, prix_unitaire, disponible (basic ingredient properties).
    - **Methods:** updateAvailability() to adjust ingredient availability.
- **Repas (Meal):**
    - **Attributes:** commande_id, plat_principal_id, total (links to an order and manages costs).
    - **Methods:** addSupplement(), calculateTotal() to include supplements and compute the total.
- **Supplement:**
    - **Attributes:** nom, prix, disponible (additional options for meals).
    - **Methods:** updateAvailability() to manage availability.

---

3. Directional or Bidirectional Relationships

- **Directional (most cases):**
    - Simplifies the design and reflects business requirements, such as:
        - **Commande → Client:** The order needs to know its client.
        - **Repas → PlatPrincipal:** The meal depends on a main dish.

- **Bidirectional:**
  - Used in n:n relationships requiring interaction in both directions:
    - **PlatPrincipal ↔ Ingredient** and **Repas ↔ Supplement** via association classes.

The Console Statements

In the exercise, we want something like this:

The console Statements :

In the exercise we want something like this :

```
Bienvenue Ali baba
-------------------------------------
------------TICKET------------------
Nom:Ali baba


nombre de repas:2
Repas N°:1 Tajine de viande & Pruneaux
Ingrédient:
Viande: 250  gramme
Pruneaux: 1  gramme
Suppléments:
Frites 11.0
Boisson 12.0
********
Repas N°:2 Tajine de poulet & légumes
Ingrédient:
Poisson: 250  gramme
Carrote: 1  gramme
Pomme de terre: 1  gramme
Olive: 1  gramme
Suppléments:
Jus d'orange 13.0
Salade marocaine 14.0
********
-------Total:125.24000000000001
-------------------------------------
```

So after I code the requirements I made this exemple to show the ticket :

Main :

```
package com.example.exam;

import java.math.BigDecimal;
import java.sql.SQLException;

public class Main {
    public static void main(String[] args) throws SQLException {
        try {
            // Initialize DAOs
            ClientDAO clientDAO = new ClientDAO();
            CommandeDAO commandeDAO = new CommandeDAO();
            IngredientDAO ingredientDAO = new IngredientDAO();
            PlatPrincipalDAO platPrincipalDAO = new PlatPrincipalDAO();
            RepasDAO repasDAO = new RepasDAO();
            SupplementDAO supplementDAO = new SupplementDAO();

            // Step 1: Create and Save a Client
            Client client = new Client("jsj", "sad",
"ddda.Bouchti@example.com", "0623456789");
            clientDAO.save(client);

            // Step 2: Create and Save Ingredients
            Ingredient viande = new Ingredient("Viande", "gramme", new
BigDecimal("10"));
            Ingredient pruneaux = new Ingredient("Pruneaux", "gramme", new
BigDecimal("5"));
            Ingredient poisson = new Ingredient("Poisson", "gramme", new
BigDecimal("12"));
            Ingredient carotte = new Ingredient("Carotte", "gramme", new
BigDecimal("3"));
            Ingredient pommeDeTerre = new Ingredient("Pomme de terre",
"gramme", new BigDecimal("2"));
            Ingredient olive = new Ingredient("Olive", "gramme", new
BigDecimal("1"));

            ingredientDAO.save(viande);
            ingredientDAO.save(pruneaux);
            ingredientDAO.save(poisson);
            ingredientDAO.save(carotte);
            ingredientDAO.save(pommeDeTerre);
            ingredientDAO.save(olive);

            // Step 3: Create and Save Supplements
            Supplement frites = new Supplement("Frites", new
BigDecimal("11"));
            Supplement boisson = new Supplement("Boisson", new
BigDecimal("12"));
            Supplement jusOrange = new Supplement("Jus d'orange", new
BigDecimal("13"));
            Supplement saladeMarocaine = new Supplement("Salade marocaine",
new BigDecimal("14"));

            supplementDAO.save(frites);
            supplementDAO.save(boisson);
            supplementDAO.save(jusOrange);
            supplementDAO.save(saladeMarocaine);

            // Step 4: Create and Save PlatPrincipal with Ingredients
```

```
            PlatPrincipal tajineViande = new PlatPrincipal("Tajine de
viande & Pruneaux", "Delicious tajine", new BigDecimal("50"));
            tajineViande.addIngredient(viande, new BigDecimal("250"));
            tajineViande.addIngredient(pruneaux, new BigDecimal("1"));
            platPrincipalDAO.save(tajineViande);

            PlatPrincipal tajinePoulet = new PlatPrincipal("Tajine de
poulet & légumes", "Healthy tajine", new BigDecimal("45"));
            tajinePoulet.addIngredient(poisson, new BigDecimal("250"));
            tajinePoulet.addIngredient(carotte, new BigDecimal("1"));
            tajinePoulet.addIngredient(pommeDeTerre, new BigDecimal("1"));
            tajinePoulet.addIngredient(olive, new BigDecimal("1"));
            platPrincipalDAO.save(tajinePoulet);

            // Step 5: Create and Save Commande
            Commande commande = new Commande(client);
            commandeDAO.save(commande); // Save the Commande first to
generate its ID

            // Step 6: Create and Save Repas
            Repas repas1 = new Repas(tajineViande);
            repas1.setCommandeId(commande.getId()); // Associate Repas with
the saved Commande
            repas1.addSupplement(frites);
            repas1.addSupplement(boisson);
            repasDAO.save(repas1);

            Repas repas2 = new Repas(tajinePoulet);
            repas2.setCommandeId(commande.getId()); // Associate Repas with
the saved Commande
            repas2.addSupplement(jusOrange);
            repas2.addSupplement(saladeMarocaine);
            repasDAO.save(repas2);

            // Step 7: Add Repas to Commande and Update Commande
            commande.addRepas(repas1);
            commande.addRepas(repas2);
            //commandeDAO.update(commande); // Update the Commande to
reflect the added Repas

            // Step 8: Retrieve Commande and Generate Ticket
            Commande retrievedCommande =
commandeDAO.findById(commande.getId());
            if (retrievedCommande != null) {
                System.out.println("Bienvenue " +
retrievedCommande.getClient().getNom());
                System.out.println("--------------------------------");
                System.out.println("------------TICKET-------------");
                System.out.println("--------------------------------");
                System.out.println("Nom: " +
retrievedCommande.getClient().getNom());
                System.out.println("Nombre de repas: " +
retrievedCommande.getRepas().size());

                int repasNumber = 1;
                for (Repas repas : retrievedCommande.getRepas()) {
                    System.out.println("Repas N°:" + repasNumber + " " +
repas.getPlatPrincipal().getNom());
                    System.out.println("Ingrédients:");
                    for (var entry :
repas.getPlatPrincipal().getIngredients().entrySet()) {
```

```
                        System.out.println(entry.getKey().getNom() + ": " +
entry.getValue() + " " + entry.getKey().getUnite());
                    }
                    System.out.println("Suppléments:");
                    for (Supplement supplement : repas.getSupplements()) {
                        System.out.println(supplement.getNom() + ": " +
supplement.getPrix());
                    }
                    System.out.println("*********");
                    repasNumber++;
                }
                System.out.println("Total: " +
retrievedCommande.getTotal());
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
Bienvenue jsj
--------------------------------
------------TICKET-------------
--------------------------------
Nom: jsj
Nombre de repas: 2
Repas N°:1 Tajine de viande & Pruneaux
Ingrédients:
Suppléments:
Frites: 11.00
Boisson: 12.00
*********
Repas N°:2 Tajine de poulet & légumes
Ingrédients:
Suppléments:
Jus d'orange: 13.00
Salade marocaine: 14.00
*********
Total: 122.33

Process finished with exit code 0
```

The Implementation using javaFX and nysql

Class Clien:

```
package com.example.exam;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class Client {
    private int id;
    private String nom;
    private String prenom;
    private String email;
    private String telephone;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    private List<Commande> commandes;

    // Default constructor (required for frameworks like Hibernate)
    public Client() {
        this.commandes = new ArrayList<>();
        this.createdAt = LocalDateTime.now();
        this.updatedAt = LocalDateTime.now();
    }

    // Parameterized constructor
    public Client(String nom, String prenom, String email, String
telephone) {
        this.nom = Objects.requireNonNull(nom, "Nom cannot be null");
        this.prenom = Objects.requireNonNull(prenom, "Prenom cannot be
null");
        this.email = validateEmail(email);
        this.telephone = validateTelephone(telephone);
        this.commandes = new ArrayList<>();
        this.createdAt = LocalDateTime.now();
        this.updatedAt = LocalDateTime.now();
    }

    // Getters and setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = Objects.requireNonNull(nom, "Nom cannot be null");
    }

    public String getPrenom() {
        return prenom;
```

```
    }

    public void setPrenom(String prenom) {
        this.prenom = Objects.requireNonNull(prenom, "Prenom cannot be
null");
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = validateEmail(email);
    }

    public String getTelephone() {
        return telephone;
    }

    public void setTelephone(String telephone) {
        this.telephone = validateTelephone(telephone);
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = Objects.requireNonNull(createdAt, "CreatedAt
cannot be null");
    }

    public LocalDateTime getUpdatedAt() {
        return updatedAt;
    }

    public void setUpdatedAt(LocalDateTime updatedAt) {
        this.updatedAt = Objects.requireNonNull(updatedAt, "UpdatedAt
cannot be null");
    }

    public List<Commande> getCommandes() {
        return commandes;
    }

    public void setCommandes(List<Commande> commandes) {
        this.commandes = Objects.requireNonNull(commandes, "Commandes
cannot be null");
    }

    // Helper methods
    public void addCommande(Commande commande) {
        if (commande != null) {
            commandes.add(commande);
        }
    }

    private String validateEmail(String email) {
        if (email == null || email.trim().isEmpty()) {
            throw new IllegalArgumentException("Email cannot be null or
empty");
```

```
        }
        if (!email.matches("^[A-Za-z0-9+_.-]+@(.+)$")) {
            throw new IllegalArgumentException("Invalid email format");
        }
        return email;
    }

    private String validateTelephone(String telephone) {
        if (telephone == null || telephone.trim().isEmpty()) {
            throw new IllegalArgumentException("Telephone cannot be null or
empty");
        }

        return telephone;
    }

    @Override
    public String toString() {
        return "Client{" +
                "id=" + id +
                ", nom='" + nom + '\'' +
                ", prenom='" + prenom + '\'' +
                ", email='" + email + '\'' +
                ", telephone='" + telephone + '\'' +
                ", createdAt=" + createdAt +
                ", updatedAt=" + updatedAt +
                ", commandes=" + commandes +
                '}';
    }
}
```

Client DAO

```
package com.example.exam;

import java.sql.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class ClientDAO {

    public void save(Client client) throws SQLException {
        String query = "INSERT INTO Client (nom, prenom, email, telephone,
created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?)";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, client.getNom());
            ps.setString(2, client.getPrenom());
            ps.setString(3, client.getEmail());
            ps.setString(4, client.getTelephone());
            ps.setTimestamp(5, Timestamp.valueOf(client.getCreatedAt()));
            ps.setTimestamp(6, Timestamp.valueOf(client.getUpdatedAt()));
            ps.executeUpdate();

            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                client.setId(rs.getInt(1));
            }
        }
```

```
    }

    public Client findById(int id) throws SQLException {
        String query = "SELECT * FROM Client WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                Client client = new Client(
                        rs.getString("nom"),
                        rs.getString("prenom"),
                        rs.getString("email"),
                        rs.getString("telephone")
                );
                client.setId(rs.getInt("id"));

client.setCreatedAt(rs.getTimestamp("created_at").toLocalDateTime());

client.setUpdatedAt(rs.getTimestamp("updated_at").toLocalDateTime());
                return client;
            }
        }
        return null;
    }

    public List<Client> findAll() throws SQLException {
        List<Client> clients = new ArrayList<>();
        String query = "SELECT * FROM Client";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query);
             ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                Client client = new Client(
                        rs.getString("nom"),
                        rs.getString("prenom"),
                        rs.getString("email"),
                        rs.getString("telephone")
                );
                client.setId(rs.getInt("id"));

client.setCreatedAt(rs.getTimestamp("created_at").toLocalDateTime());

client.setUpdatedAt(rs.getTimestamp("updated_at").toLocalDateTime());
                clients.add(client);
            }
        }
        return clients;
    }

    public void update(Client client) throws SQLException {
        String query = "UPDATE Client SET nom = ?, prenom = ?, email = ?,
telephone = ?, updated_at = ? WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setString(1, client.getNom());
            ps.setString(2, client.getPrenom());
            ps.setString(3, client.getEmail());
            ps.setString(4, client.getTelephone());
            ps.setTimestamp(5, Timestamp.valueOf(client.getUpdatedAt()));
            ps.setInt(6, client.getId());
```

```
            ps.executeUpdate();
        }
    }

    public void delete(int id) throws SQLException {
        String query = "DELETE FROM Client WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ps.executeUpdate();
        }
    }
}
```

Commande

```
package com.example.exam;

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class Commande {
    private int id;
    private Client client;
    private LocalDateTime dateCommande;
    private String statut;
    private double total;
    private List<Repas> repas;
    private LocalDateTime createdAt;

    public Commande(Client client) {
        this.client = client;
        this.dateCommande = LocalDateTime.now();
        this.statut = "en_attente"; // Default status
        this.repas = new ArrayList<>();
        this.total = 0;
        this.createdAt = LocalDateTime.now();
    }

    public void addRepas(Repas repas) {
        if (repas != null) {
            this.repas.add(repas);
            calculateTotal();
        }
    }

    public void removeRepas(Repas repas) {
        if (repas != null) {
            this.repas.remove(repas);
            calculateTotal();
        }
    }

    public void calculateTotal() {
        // Initialize total to 0
```

```java
        total = 0;

        // Check if the repas list is null or empty
        if (repas == null || repas.isEmpty()) {
            return; // If the list is null or empty, total remains 0
        }

        // Iterate through the repas list and sum the totals
        for (Repas r : repas) {
            if (r != null && r.getTotal() != null) { // Ensure Repas and
its total are not null
                total += r.getTotal().doubleValue(); // Add the Repas total
to the commande total
            }
        }
    }

    // Getters and setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Client getClient() {
        return client;
    }

    public void setClient(Client client) {
        this.client = client;
    }

    public LocalDateTime getDateCommande() {
        return dateCommande;
    }

    public void setDateCommande(LocalDateTime dateCommande) {
        this.dateCommande = dateCommande;
    }

    public String getStatut() {
        return statut;
    }

    public void setStatut(String statut) {
        // Validate statut to ensure it is one of the allowed values
        if (statut != null && (statut.equals("en_attente") ||
statut.equals("en_preparation") ||
                statut.equals("pret") || statut.equals("livre") ||
statut.equals("annule"))) {
            this.statut = statut;
        } else {
            throw new IllegalArgumentException("Invalid statut value");
        }
    }

    public double getTotal() {
        return total;
    }
```

```
    public void setTotal(double total) {
        this.total = total;
    }

    public List<Repas> getRepas() {
        return repas;
    }

    public void setRepas(List<Repas> repas) {
        this.repas = repas;
        calculateTotal(); // Recalculate total when repas list is updated
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }
}
```

Commande DAO

```
package com.example.exam;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class CommandeDAO {
    private ClientDAO clientDAO = new ClientDAO();
    private RepasDAO repasDAO = new RepasDAO();

    public void save(Commande commande) throws SQLException {
        String query = "INSERT INTO Commande (client_id, date_commande,
statut, total, created_at) VALUES (?, ?, ?, ?, ?)";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setInt(1, commande.getClient().getId());
            ps.setTimestamp(2,
Timestamp.valueOf(commande.getDateCommande()));
            ps.setString(3, commande.getStatut());
            ps.setDouble(4, commande.getTotal());
            ps.setTimestamp(5, Timestamp.valueOf(commande.getCreatedAt()));
            ps.executeUpdate();

            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                commande.setId(rs.getInt(1));
            }

            // Save repas
            for (Repas repas : commande.getRepas()) {
                repas.setCommandeId(commande.getId()); // Set the
commande_id in Repas
```

```
                repasDAO.save(repas); // Save the Repas object
            }
        }
    }

    public Commande findById(int id) throws SQLException {
        String query = "SELECT * FROM Commande WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                Client client = clientDAO.findById(rs.getInt("client_id"));
                Commande commande = new Commande(client);
                commande.setId(id);

commande.setDateCommande(rs.getTimestamp("date_commande").toLocalDateTime()
);
                commande.setStatut(rs.getString("statut"));
                commande.setTotal(rs.getDouble("total"));

commande.setCreatedAt(rs.getTimestamp("created_at").toLocalDateTime());
                commande.getRepas().addAll(findRepasByCommandeId(id));
                return commande;
            }
        }
        return null;
    }

    public List<Commande> findAll() throws SQLException {
        List<Commande> commandes = new ArrayList<>();
        String query = "SELECT * FROM Commande";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query);
             ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                Client client = clientDAO.findById(rs.getInt("client_id"));
                Commande commande = new Commande(client);
                commande.setId(rs.getInt("id"));

commande.setDateCommande(rs.getTimestamp("date_commande").toLocalDateTime()
);
                commande.setStatut(rs.getString("statut"));
                commande.setTotal(rs.getDouble("total"));

commande.setCreatedAt(rs.getTimestamp("created_at").toLocalDateTime());

commande.getRepas().addAll(findRepasByCommandeId(commande.getId()));
                commandes.add(commande);
            }
        }
        return commandes;
    }

    private List<Repas> findRepasByCommandeId(int commandeId) throws
SQLException {
        List<Repas> repasList = new ArrayList<>();
        String query = "SELECT * FROM Repas WHERE commande_id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, commandeId);
```

```
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                Repas repas = repasDAO.findById(rs.getInt("id")); // Fetch
the full Repas object
                repasList.add(repas);
            }
        }
        return repasList;
    }

    public void delete(int id) throws SQLException {
        // Delete related repas first
        deleteRepasRelations(id);

        // Delete the commande
        String query = "DELETE FROM Commande WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ps.executeUpdate();
        }
    }

    private void deleteRepasRelations(int commandeId) throws SQLException {
        String query = "DELETE FROM Repas WHERE commande_id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, commandeId);
            ps.executeUpdate();
        }
    }
}
```

Ingredient

```
package com.example.exam;

import java.math.BigDecimal;
import java.time.LocalDateTime;

public class Ingredient {
    private int id;
    private String nom;
    private String unite;
    private BigDecimal prixUnitaire;
    private boolean disponible;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    public Ingredient(String nom, String unite, BigDecimal prixUnitaire) {
        this.nom = nom;
        this.unite = unite;
        this.prixUnitaire = prixUnitaire;
        this.disponible = true;
        this.createdAt = LocalDateTime.now();
        this.updatedAt = LocalDateTime.now();
    }

    // Getters and setters
    public int getId() { return id; }
```

```
    public void setId(int id) { this.id = id; }
    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
    public String getUnite() { return unite; }
    public void setUnite(String unite) { this.unite = unite; }
    public BigDecimal getPrixUnitaire() { return prixUnitaire; }
    public void setPrixUnitaire(BigDecimal prixUnitaire) {
this.prixUnitaire = prixUnitaire; }
    public boolean isDisponible() { return disponible; }
    public void setDisponible(boolean disponible) { this.disponible =
disponible; }
    public LocalDateTime getCreatedAt() { return createdAt; }
    public LocalDateTime getUpdatedAt() { return updatedAt; }
}
```

In DAO

```
package com.example.exam;


import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class IngredientDAO {
    public void save(Ingredient ingredient) throws SQLException {
        String query = "INSERT INTO ingredient (nom, unite, prix_unitaire,
disponible, created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?)";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, ingredient.getNom());
            ps.setString(2, ingredient.getUnite());
            ps.setBigDecimal(3, ingredient.getPrixUnitaire());
            ps.setBoolean(4, ingredient.isDisponible());
            ps.setTimestamp(5,
Timestamp.valueOf(ingredient.getCreatedAt()));
            ps.setTimestamp(6,
Timestamp.valueOf(ingredient.getUpdatedAt()));
            ps.executeUpdate();

            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                ingredient.setId(rs.getInt(1));
            }
        }
    }

    public Ingredient findById(int id) throws SQLException {
        String query = "SELECT * FROM ingredient WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                Ingredient ingredient = new Ingredient(
                        rs.getString("nom"),
                        rs.getString("unite"),
                        rs.getBigDecimal("prix_unitaire")
                );
```

```
                ingredient.setId(rs.getInt("id"));
                ingredient.setDisponible(rs.getBoolean("disponible"));
                return ingredient;
            }
        }
        return null;
    }

    public List<Ingredient> findAll() throws SQLException {
        List<Ingredient> ingredients = new ArrayList<>();
        String query = "SELECT * FROM ingredient";
        try (Connection conn = SingletonConnexionDB.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()) {
                Ingredient ingredient = new Ingredient(
                        rs.getString("nom"),
                        rs.getString("unite"),
                        rs.getBigDecimal("prix_unitaire")
                );
                ingredient.setId(rs.getInt("id"));
                ingredient.setDisponible(rs.getBoolean("disponible"));
                ingredients.add(ingredient);
            }
        }
        return ingredients;
    }

    public void update(Ingredient ingredient) throws SQLException {
        String query = "UPDATE ingredient SET nom = ?, unite = ?,
prix_unitaire = ?, disponible = ?, updated_at = ? WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setString(1, ingredient.getNom());
            ps.setString(2, ingredient.getUnite());
            ps.setBigDecimal(3, ingredient.getPrixUnitaire());
            ps.setBoolean(4, ingredient.isDisponible());
            ps.setTimestamp(5,
Timestamp.valueOf(ingredient.getUpdatedAt()));
            ps.setInt(6, ingredient.getId());
            ps.executeUpdate();
        }
    }

    public void delete(int id) throws SQLException {
        String query = "DELETE FROM ingredient WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ps.executeUpdate();
        }
    }
}
```

PlatPrincipal

```
package com.example.exam;
```

```java
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;

public class PlatPrincipal {
    private int id;
    private String nom;
    private String description;
    private BigDecimal prixBase;
    private boolean disponible;
    private Map<Ingredient, BigDecimal> ingredients;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    // Default constructor (required for frameworks like Hibernate)
    public PlatPrincipal() {
        this.ingredients = new HashMap<>();
        this.createdAt = LocalDateTime.now();
        this.updatedAt = LocalDateTime.now();
    }

    // Parameterized constructor
    public PlatPrincipal(String nom, String description, BigDecimal
prixBase) {
        this.nom = Objects.requireNonNull(nom, "Nom cannot be null");
        this.description = Objects.requireNonNull(description, "Description
cannot be null");
        this.prixBase = Objects.requireNonNull(prixBase, "PrixBase cannot
be null");
        this.disponible = true;
        this.ingredients = new HashMap<>();
        this.createdAt = LocalDateTime.now();
        this.updatedAt = LocalDateTime.now();
    }

    public BigDecimal calculatePrix() {
        BigDecimal total = prixBase;
        for (Map.Entry<Ingredient, BigDecimal> entry :
ingredients.entrySet()) {
            Ingredient ingredient = entry.getKey();
            BigDecimal quantite = entry.getValue();
            total =
total.add(ingredient.getPrixUnitaire().multiply(quantite));
        }
        return total;
    }

    public void addIngredient(Ingredient ingredient, BigDecimal quantite) {
        if (ingredient != null && quantite != null &&
quantite.compareTo(BigDecimal.ZERO) > 0) {
            ingredients.put(ingredient, quantite);
            updatedAt = LocalDateTime.now();
        }
    }

    public void removeIngredient(Ingredient ingredient) {
        if (ingredient != null) {
            ingredients.remove(ingredient);
            updatedAt = LocalDateTime.now();
```

```
        }
    }

    // Getters and setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = Objects.requireNonNull(nom, "Nom cannot be null");
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = Objects.requireNonNull(description, "Description
cannot be null");
    }

    public BigDecimal getPrixBase() {
        return prixBase;
    }

    public void setPrixBase(BigDecimal prixBase) {
        this.prixBase = Objects.requireNonNull(prixBase, "PrixBase cannot
be null");
    }

    public boolean isDisponible() {
        return disponible;
    }

    public void setDisponible(boolean disponible) {
        this.disponible = disponible;
    }

    public Map<Ingredient, BigDecimal> getIngredients() {
        return ingredients;
    }

    public void setIngredients(Map<Ingredient, BigDecimal> ingredients) {
        this.ingredients = Objects.requireNonNull(ingredients, "Ingredients
cannot be null");
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = Objects.requireNonNull(createdAt, "CreatedAt
```

```
cannot be null");
    }

    public LocalDateTime getUpdatedAt() {
        return updatedAt;
    }

    public void setUpdatedAt(LocalDateTime updatedAt) {
        this.updatedAt = Objects.requireNonNull(updatedAt, "UpdatedAt
cannot be null");
    }

    @Override
    public String toString() {
        return "PlatPrincipal{" +
                "id=" + id +
                ", nom='" + nom + '\'' +
                ", description='" + description + '\'' +
                ", prixBase=" + prixBase +
                ", disponible=" + disponible +
                ", ingredients=" + ingredients +
                ", createdAt=" + createdAt +
                ", updatedAt=" + updatedAt +
                '}';
    }
}
```

Plat DAO

```
package com.example.exam;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class PlatPrincipalDAO {
    public void save(PlatPrincipal plat) throws SQLException {
        String query = "INSERT INTO platprincipal (nom, description,
prix_base, disponible, created_at, updated_at) VALUES (?, ?, ?, ?, ?, ?)";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, plat.getNom());
            ps.setString(2, plat.getDescription());
            ps.setBigDecimal(3, plat.getPrixBase());
            ps.setBoolean(4, plat.isDisponible());
            ps.setTimestamp(5, Timestamp.valueOf(plat.getCreatedAt()));
            ps.setTimestamp(6, Timestamp.valueOf(plat.getUpdatedAt()));
            ps.executeUpdate();

            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                plat.setId(rs.getInt(1));
            }
        }
    }

    public PlatPrincipal findById(int id) throws SQLException {
        String query = "SELECT * FROM platprincipal WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
```

```
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                PlatPrincipal plat = new PlatPrincipal(
                        rs.getString("nom"),
                        rs.getString("description"),
                        rs.getBigDecimal("prix_base")
                );
                plat.setId(rs.getInt("id"));
                plat.setDisponible(rs.getBoolean("disponible"));

plat.setCreatedAt(rs.getTimestamp("created_at").toLocalDateTime());

plat.setUpdatedAt(rs.getTimestamp("updated_at").toLocalDateTime());
                return plat;
            }
        }
        return null;
    }

    public List<PlatPrincipal> findAll() throws SQLException {
        List<PlatPrincipal> plats = new ArrayList<>();
        String query = "SELECT * FROM platprincipal";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query);
             ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                PlatPrincipal plat = new PlatPrincipal(
                        rs.getString("nom"),
                        rs.getString("description"),
                        rs.getBigDecimal("prix_base")
                );
                plat.setId(rs.getInt("id"));
                plat.setDisponible(rs.getBoolean("disponible"));

plat.setCreatedAt(rs.getTimestamp("created_at").toLocalDateTime());

plat.setUpdatedAt(rs.getTimestamp("updated_at").toLocalDateTime());
                plats.add(plat);
            }
        }
        return plats;
    }

    public void update(PlatPrincipal plat) throws SQLException {
        String query = "UPDATE platprincipal SET nom = ?, description = ?,
prix_base = ?, disponible = ?, updated_at = ? WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setString(1, plat.getNom());
            ps.setString(2, plat.getDescription());
            ps.setBigDecimal(3, plat.getPrixBase());
            ps.setBoolean(4, plat.isDisponible());
            ps.setTimestamp(5, Timestamp.valueOf(plat.getUpdatedAt()));
            ps.setInt(6, plat.getId());
            ps.executeUpdate();
        }
    }

    public void delete(int id) throws SQLException {
```

```
        String query = "DELETE FROM platprincipal WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ps.executeUpdate();
        }
    }
}
```

Repas

```
package com.example.exam;

import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class Repas {
    private int id;
    private PlatPrincipal platPrincipal;
    private List<Supplement> supplements;
    private BigDecimal total;
    private LocalDateTime createdAt;
    private int commandeId; // Corrected field name

    public Repas(PlatPrincipal platPrincipal) {
        this.platPrincipal = platPrincipal;
        this.supplements = new ArrayList<>();
        this.createdAt = LocalDateTime.now();
        this.commandeId = 0; // Initialize to a default value
        calculateTotal();
    }

    public void addSupplement(Supplement supplement) {
        if (supplement.isDisponible()) {
            supplements.add(supplement);
            calculateTotal();
        }
    }

    public void removeSupplement(Supplement supplement) {
        supplements.remove(supplement);
        calculateTotal();
    }

    public void calculateTotal() {
        total = platPrincipal.calculatePrix();
        for (Supplement supplement : supplements) {
            total = total.add(supplement.getPrix());
        }
    }

    // Getters and setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
```

```
    }

    public PlatPrincipal getPlatPrincipal() {
        return platPrincipal;
    }

    public void setPlatPrincipal(PlatPrincipal platPrincipal) {
        this.platPrincipal = platPrincipal;
        calculateTotal();
    }

    public List<Supplement> getSupplements() {
        return supplements;
    }

    public BigDecimal getTotal() { // Corrected return type
        return total;
    }

    public void setTotal(BigDecimal total) { // Added setter for total
        this.total = total;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(LocalDateTime createdAt) { // Added setter for
createdAt
        this.createdAt = createdAt;
    }

    public int getCommandeId() { // Corrected implementation
        return commandeId;
    }

    public void setCommandeId(int commandeId) { // Added setter for
commandeId
        this.commandeId = commandeId;
    }
}
```

```
package com.example.exam;

import java.math.BigDecimal;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class RepasDAO {
    private PlatPrincipalDAO platPrincipalDAO = new PlatPrincipalDAO();
    private SupplementDAO supplementDAO = new SupplementDAO();

    public void save(Repas repas) throws SQLException {
        String query = "INSERT INTO Repas (commande_id, plat_principal_id,
total, created_at) VALUES (?, ?, ?, ?)";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setInt(1, repas.getCommandeId());
```

```
            ps.setInt(2, repas.getPlatPrincipal().getId());
            ps.setBigDecimal(3, repas.getTotal());
            ps.setTimestamp(4, Timestamp.valueOf(repas.getCreatedAt()));
            ps.executeUpdate();

            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                repas.setId(rs.getInt(1));
            }

            // Save supplements
            for (Supplement supplement : repas.getSupplements()) {
                saveSupplementRelation(repas.getId(), supplement.getId());
            }
        }
    }

    private void saveSupplementRelation(int repasId, int supplementId)
throws SQLException {
        String query = "INSERT INTO RepasSupplement (repas_id,
supplement_id) VALUES (?, ?)";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, repasId);
            ps.setInt(2, supplementId);
            ps.executeUpdate();
        }
    }

    public Repas findById(int id) throws SQLException {
        String query = "SELECT * FROM Repas WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                PlatPrincipal platPrincipal =
platPrincipalDAO.findById(rs.getInt("plat_principal_id"));
                Repas repas = new Repas(platPrincipal);
                repas.setId(id);
                repas.setCommandeId(rs.getInt("commande_id"));
                repas.setTotal(BigDecimal.valueOf(rs.getDouble("total")));

repas.setCreatedAt(rs.getTimestamp("created_at").toLocalDateTime());

repas.getSupplements().addAll(findSupplementsByRepasId(id));
                return repas;
            }
        }
        return null;
    }

    private List<Supplement> findSupplementsByRepasId(int repasId) throws
SQLException {
        List<Supplement> supplements = new ArrayList<>();
        String query = "SELECT supplement_id FROM RepasSupplement WHERE
repas_id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, repasId);
            ResultSet rs = ps.executeQuery();
```

```
            while (rs.next()) {

supplements.add(supplementDAO.findById(rs.getInt("supplement_id")));
            }
        }
        return supplements;
    }

    public void delete(int id) throws SQLException {
        String query = "DELETE FROM Repas WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ps.executeUpdate();

            // Delete related supplements
            deleteSupplementRelations(id);
        }
    }

    private void deleteSupplementRelations(int repasId) throws SQLException
{
        String query = "DELETE FROM RepasSupplement WHERE repas_id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, repasId);
            ps.executeUpdate();
        }
    }
}
```

Suplement

```
package com.example.exam;

import java.math.BigDecimal;
import java.time.LocalDateTime;

public class Supplement {
    private int id;
    private String nom;
    private BigDecimal prix;
    private boolean disponible;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    public Supplement(String nom, BigDecimal prix) {
        this.nom = nom;
        this.prix = prix;
        this.disponible = true;
        this.createdAt = LocalDateTime.now();
        this.updatedAt = LocalDateTime.now();
    }

    // Getters and setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
    public BigDecimal getPrix() { return prix; }
```

```
    public void setPrix(BigDecimal prix) { this.prix = prix; }
    public boolean isDisponible() { return disponible; }
    public void setDisponible(boolean disponible) { this.disponible =
disponible; }
    public LocalDateTime getCreatedAt() { return createdAt; }
    public LocalDateTime getUpdatedAt() { return updatedAt; }
}
```

Supplement DAO

```
package com.example.exam;



import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SupplementDAO {
    public void save(Supplement supplement) throws SQLException {
        String query = "INSERT INTO supplement (nom, prix, disponible,
created_at, updated_at) VALUES (?, ?, ?, ?, ?)";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, supplement.getNom());
            ps.setBigDecimal(2, supplement.getPrix());
            ps.setBoolean(3, supplement.isDisponible());
            ps.setTimestamp(4,
Timestamp.valueOf(supplement.getCreatedAt()));
            ps.setTimestamp(5,
Timestamp.valueOf(supplement.getUpdatedAt()));
            ps.executeUpdate();

            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                supplement.setId(rs.getInt(1));
            }
        }
    }

    public Supplement findById(int id) throws SQLException {
        String query = "SELECT * FROM supplement WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                Supplement supplement = new Supplement(
                        rs.getString("nom"),
                        rs.getBigDecimal("prix")
                );
                supplement.setId(rs.getInt("id"));
                supplement.setDisponible(rs.getBoolean("disponible"));
                return supplement;
            }
        }
        return null;
    }
```

```java
    public List<Supplement> findAll() throws SQLException {
        List<Supplement> supplements = new ArrayList<>();
        String query = "SELECT * FROM supplement";
        try (Connection conn = SingletonConnexionDB.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()) {
                Supplement supplement = new Supplement(
                        rs.getString("nom"),
                        rs.getBigDecimal("prix")
                );
                supplement.setId(rs.getInt("id"));
                supplement.setDisponible(rs.getBoolean("disponible"));
                supplements.add(supplement);
            }
        }
        return supplements;
    }

    public void update(Supplement supplement) throws SQLException {
        String query = "UPDATE supplement SET nom = ?, prix = ?, disponible
= ?, updated_at = ? WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setString(1, supplement.getNom());
            ps.setBigDecimal(2, supplement.getPrix());
            ps.setBoolean(3, supplement.isDisponible());
            ps.setTimestamp(4,
Timestamp.valueOf(supplement.getUpdatedAt()));
            ps.setInt(5, supplement.getId());
            ps.executeUpdate();
        }
    }

    public void delete(int id) throws SQLException {
        String query = "DELETE FROM supplement WHERE id = ?";
        try (Connection conn = SingletonConnexionDB.getConnection();
             PreparedStatement ps = conn.prepareStatement(query)) {
            ps.setInt(1, id);
            ps.executeUpdate();
        }
    }
}
```

Controller

```
package com.example.exam;

import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.layout.VBox;
import javafx.scene.control.cell.PropertyValueFactory;

import java.math.BigDecimal;
import java.sql.SQLException;
import java.time.LocalDateTime;

public class RestaurantController {

    @FXML private TextField clientNomField;
    @FXML private TextField clientPrenomField;
    @FXML private TextField clientEmailField;
    @FXML private TextField clientTelephoneField;
    @FXML private Button addClientButton;
    @FXML private Button deleteClientButton;

    @FXML private ComboBox<Client> commandeClientComboBox;
    @FXML private TextField commandeStatutField;
    @FXML private TextField commandeTotalField;
    @FXML private Button addCommandeButton;
    @FXML private Button deleteCommandeButton;

    @FXML private TableView<Client> clientTable;
    @FXML private TableColumn<Client, Integer> clientIdColumn;
    @FXML private TableColumn<Client, String> clientNomColumn;
    @FXML private TableColumn<Client, String> clientPrenomColumn;
    @FXML private TableColumn<Client, String> clientEmailColumn;
    @FXML private TableColumn<Client, String> clientTelephoneColumn;

    @FXML private TableView<Commande> commandeTable;
    @FXML private TableColumn<Commande, Integer> commandeIdColumn;
    @FXML private TableColumn<Commande, String> commandeClientColumn;
    @FXML private TableColumn<Commande, String> commandeStatutColumn;
    @FXML private TableColumn<Commande, Double> commandeTotalColumn;
    @FXML private TableColumn<Commande, LocalDateTime> commandeDateColumn;


    @FXML private TextField platNomField;
    @FXML private TextField platDescriptionField;
    @FXML private TextField platPrixBaseField;
    @FXML private Button addPlatButton;
    @FXML private Button deletePlatButton;
    @FXML private TableView<PlatPrincipal> platTable;
    @FXML private TableColumn<PlatPrincipal, Integer> platIdColumn;
    @FXML private TableColumn<PlatPrincipal, String> platNomColumn;
    @FXML private TableColumn<PlatPrincipal, String> platDescriptionColumn;
    @FXML private TableColumn<PlatPrincipal, BigDecimal>
platPrixBaseColumn;
    @FXML private TableColumn<PlatPrincipal, Boolean> platDisponibleColumn;

    @FXML private TextField ingredientNomField;
    @FXML private TextField ingredientUniteField;
    @FXML private TextField ingredientPrixUnitaireField;
```

```
    @FXML private Button addIngredientButton;
    @FXML private Button deleteIngredientButton;
    @FXML private TableView<Ingredient> ingredientTable;
    @FXML private TableColumn<Ingredient, Integer> ingredientIdColumn;
    @FXML private TableColumn<Ingredient, String> ingredientNomColumn;
    @FXML private TableColumn<Ingredient, String> ingredientUniteColumn;
    @FXML private TableColumn<Ingredient, BigDecimal>
ingredientPrixUnitaireColumn;
    @FXML private TableColumn<Ingredient, Boolean>
ingredientDisponibleColumn;

    @FXML private VBox platSection;
    @FXML private VBox ingredientSection;

    private PlatPrincipalDAO platDAO = new PlatPrincipalDAO();
    private IngredientDAO ingredientDAO = new IngredientDAO();
    private ObservableList<PlatPrincipal> plats =
FXCollections.observableArrayList();
    private ObservableList<Ingredient> ingredients =
FXCollections.observableArrayList();
    @FXML private VBox clientSection;
    @FXML private VBox commandeSection;

    private ClientDAO clientDAO = new ClientDAO();
    private CommandeDAO commandeDAO = new CommandeDAO();
    private ObservableList<Client> clients =
FXCollections.observableArrayList();
    private ObservableList<Commande> commandes =
FXCollections.observableArrayList();

    @FXML
    public void initialize() {

        loadClients();
        loadCommandes();
        loadPlats();
        loadIngredients();
        // Initialize Plat Table Columns
        platIdColumn.setCellValueFactory(new PropertyValueFactory<>("id"));
        platNomColumn.setCellValueFactory(new
PropertyValueFactory<>("nom"));
        platDescriptionColumn.setCellValueFactory(new
PropertyValueFactory<>("description"));
        platPrixBaseColumn.setCellValueFactory(new
PropertyValueFactory<>("prixBase"));
        platDisponibleColumn.setCellValueFactory(new
PropertyValueFactory<>("disponible"));

        // Initialize Ingredient Table Columns
        ingredientIdColumn.setCellValueFactory(new
PropertyValueFactory<>("id"));
        ingredientNomColumn.setCellValueFactory(new
PropertyValueFactory<>("nom"));
        ingredientUniteColumn.setCellValueFactory(new
PropertyValueFactory<>("unite"));
        ingredientPrixUnitaireColumn.setCellValueFactory(new
PropertyValueFactory<>("prixUnitaire"));
        ingredientDisponibleColumn.setCellValueFactory(new
PropertyValueFactory<>("disponible"));

        // Load plats and ingredients from the database
```

```
        // Bind the Tables to the ObservableLists
        platTable.setItems(plats);
        ingredientTable.setItems(ingredients);

        // Set up button actions
        addPlatButton.setOnAction(e -> addPlat());
        deletePlatButton.setOnAction(e -> deletePlat());
        addIngredientButton.setOnAction(e -> addIngredient());
        deleteIngredientButton.setOnAction(e -> deleteIngredient());
        // Initialize Client Table Columns
        clientIdColumn.setCellValueFactory(new
PropertyValueFactory<>("id"));
        clientNomColumn.setCellValueFactory(new
PropertyValueFactory<>("nom"));
        clientPrenomColumn.setCellValueFactory(new
PropertyValueFactory<>("prenom"));
        clientEmailColumn.setCellValueFactory(new
PropertyValueFactory<>("email"));
        clientTelephoneColumn.setCellValueFactory(new
PropertyValueFactory<>("telephone"));

        // Initialize Commande Table Columns
        commandeIdColumn.setCellValueFactory(new
PropertyValueFactory<>("id"));
        commandeClientColumn.setCellValueFactory(new
PropertyValueFactory<>("client"));
        commandeStatutColumn.setCellValueFactory(new
PropertyValueFactory<>("statut"));
        commandeTotalColumn.setCellValueFactory(new
PropertyValueFactory<>("total"));
        commandeDateColumn.setCellValueFactory(new
PropertyValueFactory<>("dateCommande"));

        // Load clients and commandes from the database


        // Bind the Tables to the ObservableLists
        clientTable.setItems(clients);
        commandeTable.setItems(commandes);

        // Populate the ComboBox with clients
        commandeClientComboBox.setItems(clients);

        // Set up button actions
        addClientButton.setOnAction(e -> addClient());
        deleteClientButton.setOnAction(e -> deleteClient());
        addCommandeButton.setOnAction(e -> addCommande());
        deleteCommandeButton.setOnAction(e -> deleteCommande());
    }

    @FXML
    private void showPlatSection() {
        platSection.setVisible(true);
        ingredientSection.setVisible(false);
        clientSection.setVisible(false);
        commandeSection.setVisible(false);
    }

    @FXML
```

```
    private void showIngredientSection() {
        platSection.setVisible(false);
        ingredientSection.setVisible(true);
        clientSection.setVisible(false);
        commandeSection.setVisible(false);
    }

    private void loadPlats() {
        try {
            plats.clear();
            plats.addAll(platDAO.findAll());
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to load plats: " +
e.getMessage());
        }
    }

    private void loadIngredients() {
        try {
            ingredients.clear();
            ingredients.addAll(ingredientDAO.findAll());
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to load ingredients: " +
e.getMessage());
        }
    }

    @FXML
    private void addPlat() {
        String nom = platNomField.getText();
        String description = platDescriptionField.getText();
        BigDecimal prixBase = new BigDecimal(platPrixBaseField.getText());

        if (nom.isEmpty() || description.isEmpty() || prixBase == null) {
            showAlert("Input Error", "All fields are required.");
            return;
        }

        PlatPrincipal plat = new PlatPrincipal(nom, description, prixBase);
        try {
            platDAO.save(plat);
            plats.add(plat);
            clearPlatFields();
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to save plat: " +
e.getMessage());
        }
    }

    @FXML
    private void deletePlat() {
        PlatPrincipal selectedPlat =
platTable.getSelectionModel().getSelectedItem();
        if (selectedPlat == null) {
            showAlert("Selection Error", "No plat selected.");
            return;
        }

        try {
            platDAO.delete(selectedPlat.getId());
            plats.remove(selectedPlat);
```

```
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to delete plat: " +
e.getMessage());
        }
    }

    @FXML
    private void addIngredient() {
        String nom = ingredientNomField.getText();
        String unite = ingredientUniteField.getText();
        BigDecimal prixUnitaire = new
BigDecimal(ingredientPrixUnitaireField.getText());

        if (nom.isEmpty() || unite.isEmpty() || prixUnitaire == null) {
            showAlert("Input Error", "All fields are required.");
            return;
        }

        Ingredient ingredient = new Ingredient(nom, unite, prixUnitaire);
        try {
            ingredientDAO.save(ingredient);
            ingredients.add(ingredient);
            clearIngredientFields();
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to save ingredient: " +
e.getMessage());
        }
    }

    @FXML
    private void deleteIngredient() {
        Ingredient selectedIngredient =
ingredientTable.getSelectionModel().getSelectedItem();
        if (selectedIngredient == null) {
            showAlert("Selection Error", "No ingredient selected.");
            return;
        }

        try {
            ingredientDAO.delete(selectedIngredient.getId());
            ingredients.remove(selectedIngredient);
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to delete ingredient: " +
e.getMessage());
        }
    }

    private void clearPlatFields() {
        platNomField.clear();
        platDescriptionField.clear();
        platPrixBaseField.clear();
    }

    private void clearIngredientFields() {
        ingredientNomField.clear();
        ingredientUniteField.clear();
        ingredientPrixUnitaireField.clear();
    }

    @FXML
    private void showClientSection() {
```

```
        clientSection.setVisible(true);
        commandeSection.setVisible(false);
        platSection.setVisible(false);
        ingredientSection.setVisible(false);
    }

    @FXML
    private void showCommandeSection() {
        clientSection.setVisible(false);
        commandeSection.setVisible(true);
        platSection.setVisible(false);
        ingredientSection.setVisible(false);
    }

    private void loadClients() {
        try {
            clients.clear();
            clients.addAll(clientDAO.findAll());
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to load clients: " +
e.getMessage());
        }
    }

    private void loadCommandes() {
        try {
            commandes.clear();
            for (Commande commande : commandeDAO.findAll()) {
                commandes.add(commande);
            }
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to load commandes: " +
e.getMessage());
        }
    }

    @FXML
    private void addClient() {
        String nom = clientNomField.getText();
        String prenom = clientPrenomField.getText();
        String email = clientEmailField.getText();
        String telephone = clientTelephoneField.getText();

        if (nom.isEmpty() || prenom.isEmpty() || email.isEmpty() ||
telephone.isEmpty()) {
            showAlert("Input Error", "All fields are required.");
            return;
        }

        Client client = new Client(nom, prenom, email, telephone);
        try {
            clientDAO.save(client);
            clients.add(client);
            clearClientFields();
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to save client: " +
e.getMessage());
        }
    }

    @FXML
```

```
    private void deleteClient() {
        Client selectedClient =
clientTable.getSelectionModel().getSelectedItem();
        if (selectedClient == null) {
            showAlert("Selection Error", "No client selected.");
            return;
        }

        try {
            clientDAO.delete(selectedClient.getId());
            clients.remove(selectedClient);
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to delete client: " +
e.getMessage());
        }
    }

    @FXML
    private void addCommande() {
        Client selectedClient =
commandeClientComboBox.getSelectionModel().getSelectedItem();
        String statut = commandeStatutField.getText();
        double total = Double.parseDouble(commandeTotalField.getText());

        if (selectedClient == null || statut.isEmpty() || total < 0) {
            showAlert("Input Error", "Invalid input for commande.");
            return;
        }

        Commande commande = new Commande(selectedClient);
        commande.setStatut(statut);
        commande.setTotal(total);
        commande.setCreatedAt(LocalDateTime.now());

        try {
            commandeDAO.save(commande);
            commandes.add(commande);
            clearCommandeFields();
        } catch (SQLException e) {
            showAlert("Database Error", "Failed to save commande: " +
e.getMessage());
        }
    }




    @FXML
    private void deleteCommande() {
        Commande selectedCommande =
commandeTable.getSelectionModel().getSelectedItem();
        if (selectedCommande == null) {
            showAlert("Selection Error", "No commande selected.");
            return;
        }

        try {
            commandeDAO.delete(selectedCommande.getId());
            commandes.remove(selectedCommande);
        } catch (SQLException e) {
```

```
            showAlert("Database Error", "Failed to delete commande: " +
e.getMessage());
        }
    }

    private void clearClientFields() {
        clientNomField.clear();
        clientPrenomField.clear();
        clientEmailField.clear();
        clientTelephoneField.clear();
    }

    private void clearCommandeFields() {
        commandeClientComboBox.getSelectionModel().clearSelection();
        commandeStatutField.clear();
        commandeTotalField.clear();
    }

    private void showAlert(String title, String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }
}
```

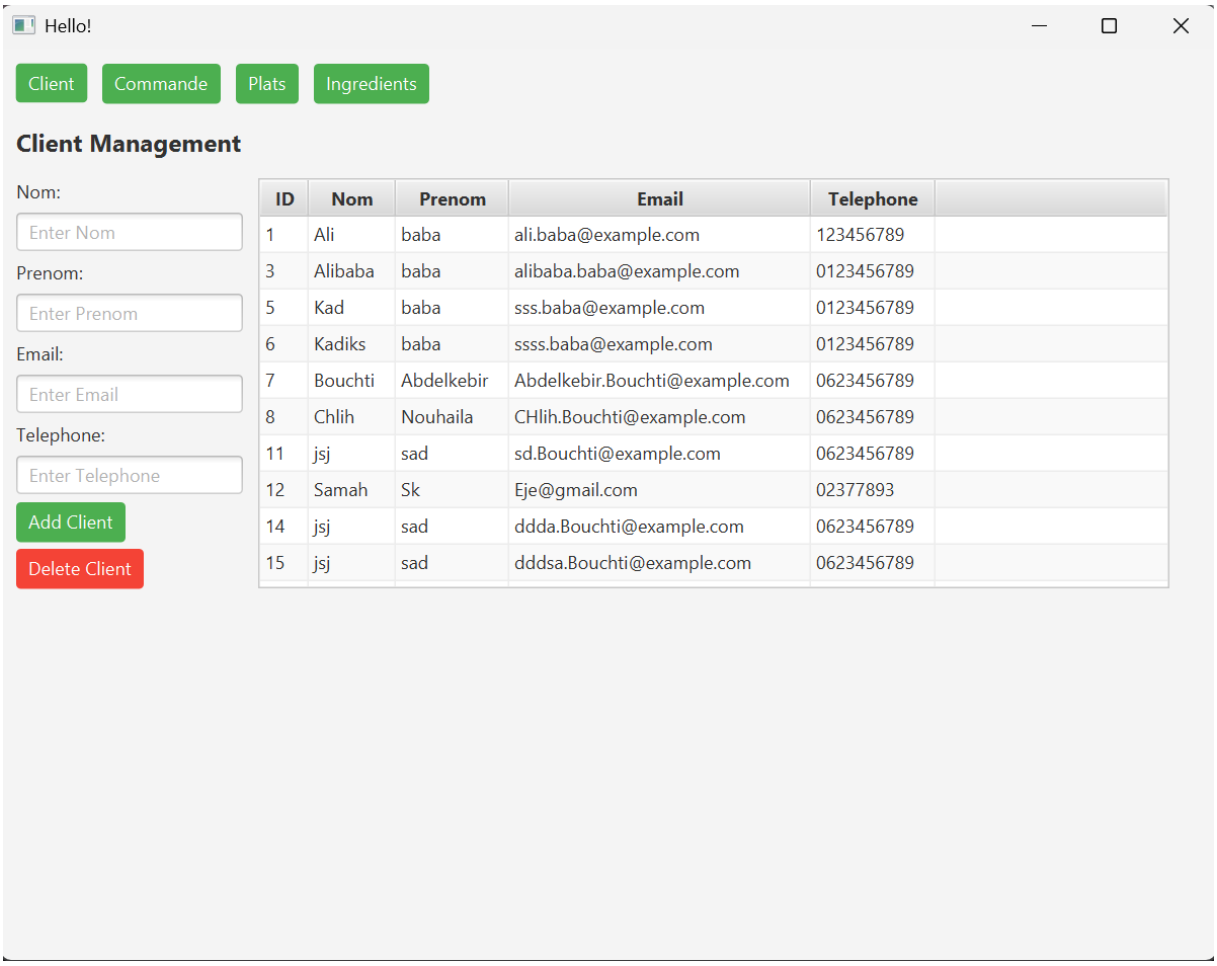
FXML

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ComboBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.layout.StackPane?>

<AnchorPane xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.example.exam.RestaurantController">
    <!-- Top Buttons -->
    <HBox spacing="10" AnchorPane.topAnchor="10.0"
AnchorPane.leftAnchor="10.0">
        <Button text="Client" fx:id="showClientButton" style="-fx-
background-color: #4CAF50; -fx-text-fill: white;"
onAction="#showClientSection"/>
        <Button text="Commande" fx:id="showCommandeButton" style="-fx-
background-color: #4CAF50; -fx-text-fill: white;"
onAction="#showCommandeSection"/>
        <Button text="Plats" fx:id="showPlatButton" style="-fx-background-
color: #4CAF50; -fx-text-fill: white;" onAction="#showPlatSection"/>
        <Button text="Ingredients" fx:id="showIngredientButton" style="-fx-
background-color: #4CAF50; -fx-text-fill: white;"
```

```
onAction="#showIngredientSection"/>
    </HBox>

    <!-- StackPane to switch between sections -->
    <StackPane AnchorPane.topAnchor="50.0" AnchorPane.leftAnchor="10.0"
AnchorPane.rightAnchor="10.0" AnchorPane.bottomAnchor="10.0">
        <!-- Client Section -->
        <VBox fx:id="clientSection" spacing="10">
            <Label text="Client Management" style="-fx-font-size: 16px; -
fx-font-weight: bold;"/>
            <HBox spacing="10">
                <VBox spacing="5">
                    <Label text="Nom:"/>
                    <TextField fx:id="clientNomField" promptText="Enter
Nom"/>
                    <Label text="Prenom:"/>
                    <TextField fx:id="clientPrenomField" promptText="Enter
Prenom"/>
                    <Label text="Email:"/>
                    <TextField fx:id="clientEmailField" promptText="Enter
Email"/>
                    <Label text="Telephone:"/>
                    <TextField fx:id="clientTelephoneField"
promptText="Enter Telephone"/>
                    <Button fx:id="addClientButton" text="Add Client"
style="-fx-background-color: #4CAF50; -fx-text-fill: white;"
onAction="#addClient"/>
                    <Button fx:id="deleteClientButton" text="Delete Client"
style="-fx-background-color: #f44336; -fx-text-fill: white;"
onAction="#deleteClient"/>
                </VBox>
                <TableView fx:id="clientTable" prefWidth="600"
prefHeight="200">
                    <columns>
                        <TableColumn fx:id="clientIdColumn" text="ID"/>
                        <TableColumn fx:id="clientNomColumn" text="Nom"/>
                        <TableColumn fx:id="clientPrenomColumn"
text="Prenom"/>
                        <TableColumn fx:id="clientEmailColumn"
text="Email"/>
                        <TableColumn fx:id="clientTelephoneColumn"
text="Telephone"/>
                    </columns>
                </TableView>
            </HBox>
        </VBox>

        <!-- Commande Section -->
        <VBox fx:id="commandeSection" spacing="10" visible="false">
            <Label text="Commande Management" style="-fx-font-size: 16px; -
fx-font-weight: bold;"/>
            <HBox spacing="10">
                <VBox spacing="5">
                    <Label text="Client:"/>
                    <ComboBox fx:id="commandeClientComboBox"
promptText="Select Client"/>
                    <Label text="Statut:"/>
                    <TextField fx:id="commandeStatutField"
promptText="Enter Statut"/>
                    <Label text="Total:"/>
                    <TextField fx:id="commandeTotalField" promptText="Enter
```

```
Total"/>
                    <Button fx:id="addCommandeButton" text="Add Commande"
style="-fx-background-color: #4CAF50; -fx-text-fill: white;"
onAction="#addCommande"/>
                    <Button fx:id="deleteCommandeButton" text="Delete
Commande" style="-fx-background-color: #f44336; -fx-text-fill: white;"
onAction="#deleteCommande"/>
                </VBox>
                <TableView fx:id="commandeTable" prefWidth="600"
prefHeight="200">
                    <columns>
                        <TableColumn fx:id="commandeIdColumn" text="ID"/>
                        <TableColumn fx:id="commandeClientColumn"
text="Client"/>
                        <TableColumn fx:id="commandeStatutColumn"
text="Statut"/>
                        <TableColumn fx:id="commandeTotalColumn"
text="Total"/>
                        <TableColumn fx:id="commandeDateColumn"
text="Date"/>
                    </columns>
                </TableView>
            </HBox>
        </VBox>

        <!-- Plat Section -->
        <VBox fx:id="platSection" spacing="10" visible="false">
            <Label text="Plat Management" style="-fx-font-size: 16px; -fx-
font-weight: bold;"/>
            <HBox spacing="10">
                <VBox spacing="5">
                    <Label text="Nom:"/>
                    <TextField fx:id="platNomField" promptText="Enter
Nom"/>
                    <Label text="Description:"/>
                    <TextField fx:id="platDescriptionField"
promptText="Enter Description"/>
                    <Label text="Prix Base:"/>
                    <TextField fx:id="platPrixBaseField" promptText="Enter
Prix Base"/>
                    <Button fx:id="addPlatButton" text="Add Plat" style="-
fx-background-color: #4CAF50; -fx-text-fill: white;" onAction="#addPlat"/>
                    <Button fx:id="deletePlatButton" text="Delete Plat"
style="-fx-background-color: #f44336; -fx-text-fill: white;"
onAction="#deletePlat"/>
                </VBox>
                <TableView fx:id="platTable" prefWidth="600"
prefHeight="200">
                    <columns>
                        <TableColumn fx:id="platIdColumn" text="ID"/>
                        <TableColumn fx:id="platNomColumn" text="Nom"/>
                        <TableColumn fx:id="platDescriptionColumn"
text="Description"/>
                        <TableColumn fx:id="platPrixBaseColumn" text="Prix
Base"/>
                        <TableColumn fx:id="platDisponibleColumn"
text="Disponible"/>
                    </columns>
                </TableView>
            </HBox>
        </VBox>
```

```
        <!-- Ingredient Section -->
        <VBox fx:id="ingredientSection" spacing="10" visible="false">
            <Label text="Ingredient Management" style="-fx-font-size: 16px;
-fx-font-weight: bold;"/>
            <HBox spacing="10">
                <VBox spacing="5">
                    <Label text="Nom:"/>
                    <TextField fx:id="ingredientNomField" promptText="Enter
Nom"/>
                    <Label text="Unite:"/>
                    <TextField fx:id="ingredientUniteField"
promptText="Enter Unite"/>
                    <Label text="Prix Unitaire:"/>
                    <TextField fx:id="ingredientPrixUnitaireField"
promptText="Enter Prix Unitaire"/>
                    <Button fx:id="addIngredientButton" text="Add
Ingredient" style="-fx-background-color: #4CAF50; -fx-text-fill: white;"
onAction="#addIngredient"/>
                    <Button fx:id="deleteIngredientButton" text="Delete
Ingredient" style="-fx-background-color: #f44336; -fx-text-fill: white;"
onAction="#deleteIngredient"/>
                </VBox>
                <TableView fx:id="ingredientTable" prefWidth="600"
prefHeight="200">
                    <columns>
                        <TableColumn fx:id="ingredientIdColumn" text="ID"/>
                        <TableColumn fx:id="ingredientNomColumn"
text="Nom"/>
                        <TableColumn fx:id="ingredientUniteColumn"
text="Unite"/>
                        <TableColumn fx:id="ingredientPrixUnitaireColumn"
text="Prix Unitaire"/>
                        <TableColumn fx:id="ingredientDisponibleColumn"
text="Disponible"/>
                    </columns>
                </TableView>
            </HBox>
        </VBox>
    </StackPane>
</AnchorPane>
```



Description of the JavaFX Application:

Client Management Section:

Fields for Client Information:

Nom (Name): A text field to enter the client's name.

Prenom (First Name): A text field to enter the client's first name.

Email: A text field to enter the client's email address.

Telephone: A text field to enter the client's phone number.

Client Table:

The table displays a list of clients with the following columns:

ID: Unique identifier for each client.

Nom: The name of the client.

Prenom: The first name of the client.

Email: The email address of the client.

Telephone: The phone number of the client.

Buttons:

Add Client: A button to add a new client to the table using the information entered in the fields.

Delete Client: A button to remove a selected client from the table.

Sample Data:

The table contains sample client data, such as:

ID: 1, Nom: Ali, Prenom: baba, Email: ali.baba@example.com, Telephone: 123456789

ID: 3, Nom: Alibaba, Prenom: baba, Email: alibaba.baba@example.com, Telephone: 0123456789

ID: 5, Nom: Kad, Prenom: baba, Email: sss.baba@example.com, Telephone: 0123456789

And so on.

Functionality:

Adding a Client:

The user can enter the client's details (Nom, Prenom, Email, Telephone) in the respective fields and click the "Add Client" button to add the client to the table.

Deleting a Client:

The user can select a client from the table and click the "Delete Client" button to remove the client from the list.

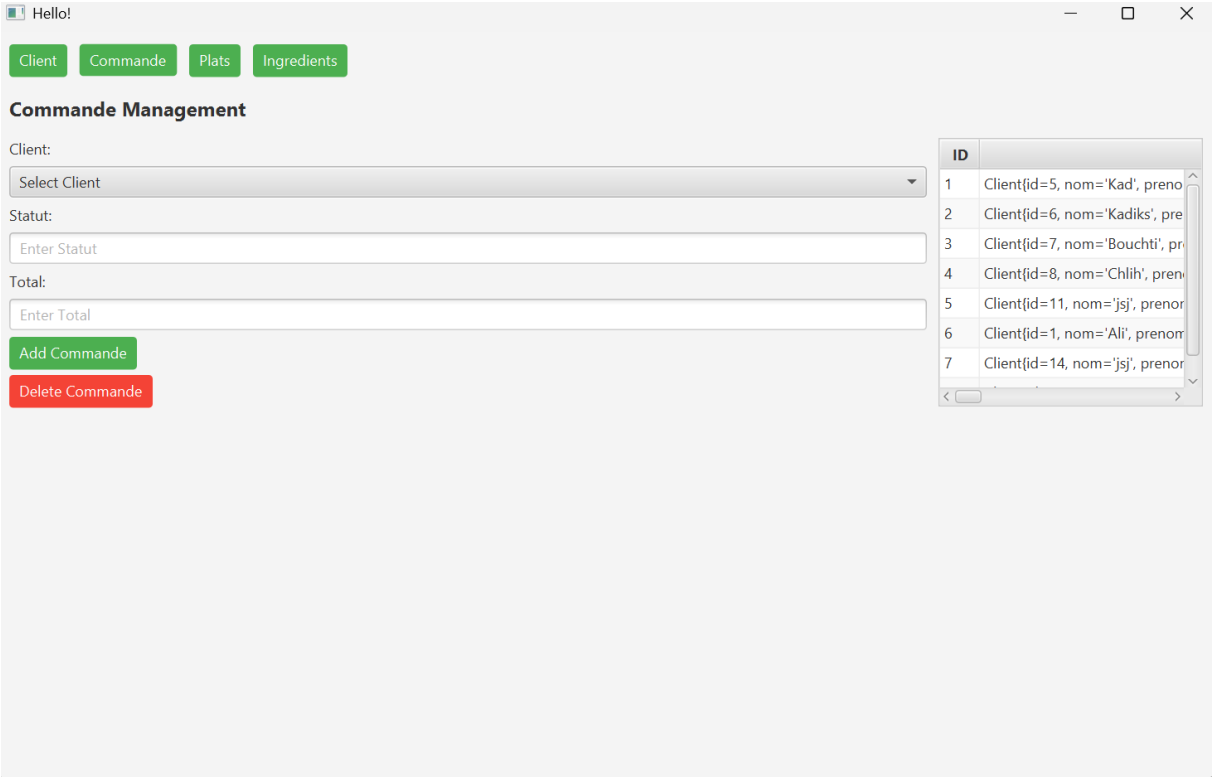
JavaFX Components Used:

Text Fields: For inputting client details.

Table View: To display the list of clients.

Buttons: For adding and deleting clients.

Table Columns: To organize and display client information.



Description of the JavaFX Application:

Commande Management Section:

Fields for Commande Information:

Client: A dropdown or selection field to choose a client for the order.

Statut (Status): A text field to enter the status of the order.

Total: A text field to enter the total amount for the order.

Client Selection:

The application allows selecting a client from a list of existing clients. The clients are displayed with their IDs and details, such as:

ID: 5, Nom: Kad, Prenom: baba

ID: 6, Nom: Kadiks, Prenom: baba

ID: 7, Nom: Bouchti, Prenom: Abdelkebir

ID: 8, Nom: Chihi, Prenom: Nouhaila

And so on.

Buttons:

Add Commande: A button to add a new order using the selected client and entered details.

Delete Commande: A button to remove a selected order from the list.

Functionality:

Adding a Commande:

The user can select a client from the dropdown, enter the status and total amount, and click the "Add Commande" button to add the order.

Deleting a Commande:

The user can select an order and click the "Delete Commande" button to remove it from the list.

JavaFX Components Used:

ComboBox or ChoiceBox: For selecting a client from the list.

Text Fields: For inputting order status and total amount.

Buttons: For adding and deleting orders.

Labels: To display the IDs and details of clients.

Integration with Client Management:

This section is likely integrated with the client management section described earlier. The clients added in the client management section can be selected here to create orders.

Sample Data:

The client selection list contains sample client data, such as:

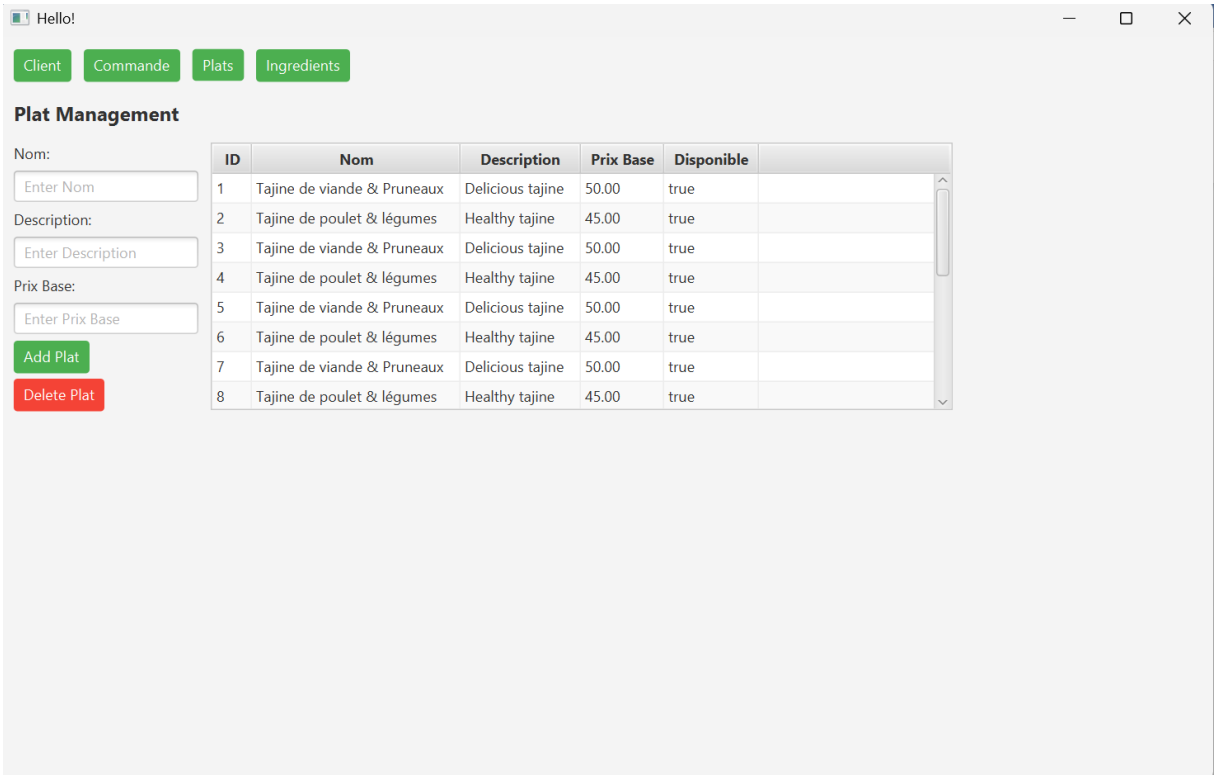
ID: 5, Nom: Kad, Prenom: baba

ID: 6, Nom: Kadiks, Prenom: baba

ID: 7, Nom: Bouchti, Prenom: Abdelkebir

ID: 8, Nom: Chihi, Prenom: Nouhaila

And so on.



Description of the JavaFX Application:

Plat Management Section:

Fields for Plat Information:

Nom (Name): A text field to enter the name of the dish.

Description: A text field to enter the description of the dish.

Prix Base (Base Price): A text field to enter the base price of the dish.

Disponible (Available): A checkbox or toggle to indicate if the dish is available.

Plat Table:

The table displays a list of dishes with the following columns:

ID: Unique identifier for each dish.

Nom: The name of the dish.

Description: The description of the dish.

Prix Base: The base price of the dish.

Disponible: Indicates whether the dish is available (true/false).

Buttons:

Add Plat: A button to add a new dish to the table using the information entered in the fields.

Delete Plat: A button to remove a selected dish from the table.

Sample Data:

The table contains sample dish data, such as:

ID: 1, Nom: Tajine de viande & Pruneaux, Description: Delicious tajine, Prix Base: 50.00, Disponible: true

ID: 2, Nom: Tajine de poulet & legumes, Description: Healthy tajine, Prix Base: 45.00, Disponible: true

ID: 3, Nom: Tajine de viande & Pruneaux, Description: Delicious tajine, Prix Base: 50.00, Disponible: true

And so on.

Functionality:

Adding a Plat:

The user can enter the dish's details (Nom, Description, Prix Base, Disponible) in the respective fields and click the "Add Plat" button to add the dish to the table.

Deleting a Plat:

The user can select a dish from the table and click the "Delete Plat" button to remove the dish from the list.

JavaFX Components Used:

Text Fields: For inputting dish details.

Table View: To display the list of dishes.

Buttons: For adding and deleting dishes.

Table Columns: To organize and display dish information.

Integration with Other Sections:

This section is likely integrated with the commande management section described earlier. The dishes added here can be associated with orders (commandes) in the commande management section.

Sample Data:

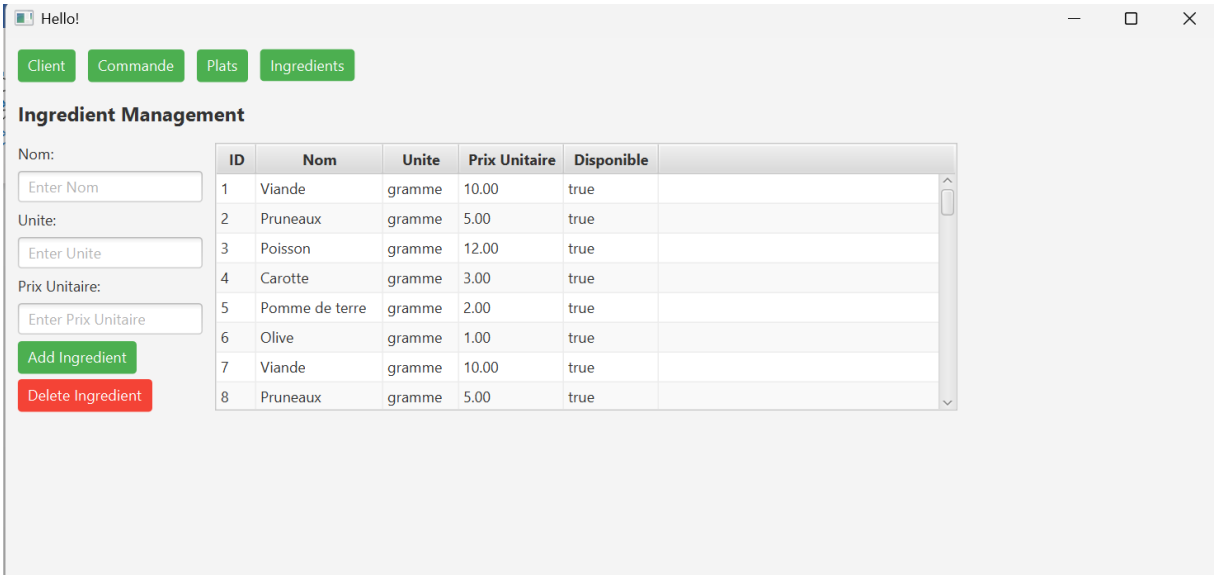
The table contains sample dish data, such as:

ID: 1, Nom: Tajine de viande & Pruneaux, Description: Delicious tajine, Prix Base: 50.00, Disponible: true

ID: 2, Nom: Tajine de poulet & legumes, Description: Healthy tajine, Prix Base: 45.00, Disponible: true

ID: 3, Nom: Tajine de viande & Pruneaux, Description: Delicious tajine, Prix Base: 50.00, Disponible: true

And so on.



Description of the JavaFX Application:

Ingredient Management Section:

Fields for Ingredient Information:

Nom (Name): A text field to enter the name of the ingredient.

Unite (Unit): A text field to enter the unit of measurement for the ingredient.

Prix Unitaire (Unit Price): A text field to enter the unit price of the ingredient.

Disponible (Available): A checkbox or toggle to indicate if the ingredient is available.

Ingredient Table:

The table displays a list of ingredients with the following columns:

ID: Unique identifier for each ingredient.

Nom: The name of the ingredient.

Unite: The unit of measurement for the ingredient.

Prix Unitaire: The unit price of the ingredient.

Disponible: Indicates whether the ingredient is available (true/false).

Buttons:

Add Ingredient: A button to add a new ingredient to the table using the information entered in the fields.

Delete Ingredient: A button to remove a selected ingredient from the table.

Sample Data:

The table contains sample ingredient data, such as:

ID: 1, Nom: Viande, Unite: gramme, Prix Unitaire: 10.00, Disponible: true

ID: 2, Nom: Pruneaux, Unite: gramme, Prix Unitaire: 5.00, Disponible: true

ID: 3, Nom: Poisson, Unite: gramme, Prix Unitaire: 12.00, Disponible: true

ID: 4, Nom: Carotte, Unite: gramme, Prix Unitaire: 3.00, Disponible: true

ID: 5, Nom: Pomme de terre, Unite: gramme, Prix Unitaire: 2.00, Disponible: true

ID: 6, Nom: Olive, Unite: gramme, Prix Unitaire: 1.00, Disponible: true

And so on.

Functionality:

Adding an Ingredient:

The user can enter the ingredient's details (Nom, Unite, Prix Unitaire, Disponible) in the respective fields and click the "Add Ingredient" button to add the ingredient to the table.

Deleting an Ingredient:

The user can select an ingredient from the table and click the "Delete Ingredient" button to remove the ingredient from the list.

JavaFX Components Used:

Text Fields: For inputting ingredient details.

Table View: To display the list of ingredients.

Buttons: For adding and deleting ingredients.

Table Columns: To organize and display ingredient information.

Integration with Other Sections:

This section is likely integrated with the plat management section described earlier. The ingredients added here can be associated with dishes (plats) in the plat management section.

Sample Data:

The table contains sample ingredient data, such as:

ID: 1, Nom: Viande, Unite: gramme, Prix Unitaire: 10.00, Disponible: true

ID: 2, Nom: Pruneaux, Unite: gramme, Prix Unitaire: 5.00, Disponible: true

ID: 3, Nom: Poisson, Unite: gramme, Prix Unitaire: 12.00, Disponible: true

ID: 4, Nom: Carotte, Unite: gramme, Prix Unitaire: 3.00, Disponible: true

ID: 5, Nom: Pomme de terre, Unite: gramme, Prix Unitaire: 2.00, Disponible: true

ID: 6, Nom: Olive, Unite: gramme, Prix Unitaire: 1.00, Disponible: true

And so on.

Project Summary: Restaurant Management System

Project Overview:

The Restaurant Management System is a JavaFX-based application designed to manage the operations of a restaurant. It provides a user-friendly interface for handling clients, orders (commandes), dishes (plats), and ingredients. The system is built using a CRUD (Create, Read, Update, Delete) architecture, allowing users to efficiently manage data across different sections of the restaurant.

Key Features:

Client Management:

Add, view, update, and delete client information.

Fields: Nom (Name), Prenom (First Name), Email, Telephone.

Table displays: ID, Nom, Prenom, Email, Telephone.

Buttons: Add Client, Delete Client.

Commande Management:

Manage orders (commandes) associated with clients.

Fields: Client (dropdown), Statut (Status), Total.

Table displays: ID, Client, Statut, Total.

Buttons: Add Commande, Delete Commande.

Plat Management:

Manage dishes (plats) available in the restaurant.

Fields: Nom (Name), Description, Prix Base (Base Price), Disponible (Available).

Table displays: ID, Nom, Description, Prix Base, Disponible.

Buttons: Add Plat, Delete Plat.

Ingredient Management:

Manage ingredients used in dishes.

Fields: Nom (Name), Unite (Unit), Prix Unitaire (Unit Price), Disponible (Available).

Table displays: ID, Nom, Unite, Prix Unitaire, Disponible.

Buttons: Add Ingredient, Delete Ingredient.

Integration Between Sections:

Clients are associated with Commandes (orders).

Plats (dishes) are associated with Ingredients.

Commandes can include multiple Plats, and Plats can include multiple Ingredients.

Functionality:

Add: Users can add new clients, orders, dishes, and ingredients.

View: Data is displayed in tables for easy viewing.

Update: Users can modify existing records (e.g., update client details, change order status).

Delete: Users can remove records (e.g., delete a client, remove a dish).

Sample Data:

Clients:

Example: ID: 1, Nom: Ali, Prenom: baba, Email: ali.baba@example.com, Telephone: 123456789.

Commandes:

Example: ID: 1, Client: Kad, Statut: en_attente, Total: 100.00.

Plats:

Example: ID: 1, Nom: Tajine de viande & Pruneaux, Description: Delicious tajine, Prix Base: 50.00, Disponible: true.

Ingredients:

Example: ID: 1, Nom: Viande, Unite: gramme, Prix Unitaire: 10.00, Disponible: true.

Technical Details:

Frontend: JavaFX for the user interface.

Backend: Java for business logic and data management.

Database: Likely a relational database (e.g., MySQL) to store client, order, dish, and ingredient data.

CRUD Operations: Implemented for all sections (Client, Commande, Plat, Ingredient).

Use Case:

A restaurant can use this system to:

Manage customer information.

Track orders and their statuses.

Maintain a menu of dishes and their ingredients.

Ensure ingredients are available for dish preparation.

Benefits:

Efficiency: Streamlines restaurant operations by centralizing data management.

User-Friendly: Intuitive interface with tables and buttons for easy navigation.

Scalability: Can be extended to include additional features (e.g., billing, inventory management).

Conclusion:

The Restaurant Management System is a comprehensive JavaFX application designed to simplify restaurant operations. It provides a seamless experience for managing clients, orders, dishes, and ingredients, ensuring that all aspects of the restaurant are well-organized and easily accessible. The

system is ideal for small to medium-sized restaurants looking to digitize their operations and improve efficiency.

Future Enhancements:

Adding the other class missed.

Use an interactive Console.