**CRAVE NEST**

## **Table of Contents**

1. **Introduction**
2. **System Architecture**
3. **Technology Stack**
4. **Database Design**
   1. User Table
   2. Customer Table
   3. Restaurant Table
   4. MenuItem Table
   5. Category Table
   6. Order Table
   7. OrderItem Table
   8. Payment Table
5. **API Endpoints**
   1. Authentication & Authorization
   2. Restaurant Management
   3. Customer Management
   4. Menu Item Management
   5. Order Management
   6. Orderitems Management
   7. Category Management
   8. Payment Management
6. **Unit of Work Pattern**
7. **Frontend (Angular) Overview**
8. **Security Considerations**

### **1. Introduction**

The Online Food Delivery Management System is a comprehensive solution for restaurants and customers, facilitating the management of food orders and restaurant profiles. The system includes an API built with **ASP.NET Core** and a frontend built with **Angular**. It allows restaurants to create and manage their profiles, and customers can browse menus, place orders, and manage their orders. Authentication and authorization are handled using **JWT (JSON Web Tokens)** for secure access to the system. The backend uses **MSSQL** for data storage, and **Unit of Work Pattern** is used to manage transactions and database operations.

### **2. System Architecture**

The system is divided into two main parts:

* **Backend (ASP.NET Core API)**: Responsible for handling business logic, database interactions, authentication, and authorization.
* **Frontend (Angular)**: Provides the user interface for customers and restaurant owners, enabling them to interact with the system.

#### **Components:**

* **Authentication & Authorization**: JWT tokens for user authentication and role-based access control.
* **Database**: MSSQL for persistent data storage.
* **Unit of Work Pattern**: Ensures that database operations are completed successfully or rolled back in case of errors.

### **3. Technology Stack**

* **Backend**:
  + **ASP.NET Core**: Used to build the RESTful API.
  + **MSSQL**: Database for storing data.
  + **JWT**: Used for securing the API and handling user authentication and authorization.
  + **Unit of Work**: Manages database transactions and operations.
* **Frontend**:
  + **Angular**: Used to build the user interface for customers and restaurants.
* **Authentication**: JWT-based authentication and role management (Admin, Restaurant Owner, Customer).

### **4. Database Design**

The database design includes the following tables:

#### **User Table**

This table stores information about the users who can be customers or restaurant owners.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| UserId | int | Primary key, auto-increment |
| Username | string | Username for login |
| Password | string | Encrypted password |
| Email | string | Email address |
| Role | string | User role (Admin, Restaurant Owner, Customer) |
| ProfilePicture | string | URL to profile picture (optional) |

#### **Customer Table**

Stores information related to customers.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| CustomerId | int | Primary key, auto-increment |
| UserId | int | Foreign key referencing the User table |
| Name | string | Customer's name |
| Phone | string | Customer's phone number |
| Address | string | Customer's address |

#### **Restaurant Table**

Stores restaurant-related information. Each restaurant is associated with a user.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| RestaurantId | int | Primary key, auto-increment |
| Name | string | Name of the restaurant |
| Description | string | Brief description of the restaurant |
| Address | string | Physical address of the restaurant |
| City | string | City of the restaurant |
| State | string | State of the restaurant |
| Phone | string | Restaurant contact number |
| Email | string | Restaurant contact email |
| Image | string | Restaurant image (optional) |
| UserId | int | Foreign key referencing the user who owns the restaurant |

#### **MenuItem Table**

Stores details about the menu items available at the restaurant.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| MenuItemId | int | Primary key, auto-increment |
| CategoryId | int | Category of the menu item |
| Name | string | Name of the menu item |
| Description | string | Description of the menu item |
| Price | decimal | Price of the menu item |
| ImageUrl | string | Image URL of the menu item (optional) |
| RestaurantId | int | Foreign key referencing the restaurant |

#### **Categories Table**

Stores details about the categories of menu items available at the restaurant.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| CategoryId | int | Primary key, auto-increment |
| Name | varchar(50) | Name of the category, required field |
| Description | varchar(200) | Detailed description of the category |

#### **Order Table**

Stores details about each order placed by a customer.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| OrderId | int | Primary key, auto-increment |
| RestaurantId | int | Foreign key referencing the restaurant |
| CustomerId | int | Foreign key referencing the customer |
| OrderDate | DateTime | Date when the order was placed |
| TotalAmount | decimal | Total amount for the order |
| Status | string | Order status (e.g., Pending, Delivered) |
| DeliveryAddress | string | Delivery address |

#### **OrderItem Table**

Stores information about each item in an order.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| OrderItemId | int | Primary key, auto-increment |
| RestaurantId | int | Foreign key referencing the restaurant |
| OrderId | int | Foreign key referencing the order |
| MenuItemId | int | Foreign key referencing the menu item |
| Quantity | int | Quantity of the menu item ordered |
| UnitPrice | decimal | Unit price of the menu item |
| Subtotal | decimal | Subtotal for this menu item (Quantity \* UnitPrice) |

#### **Payments Table**

Stores details about the payment details of the customers.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| PaymentId | int | Primary key, auto-increment |
| OrderId | int | Foreign key referencing the associated order |
| PaymentDate | datetime | Date and time of payment, defaults to current date and time |
| Amount | decimal(10,2) | Total payment amount, required field |
| PaymentMethod | varchar(50) | Payment method with constraint (Credit Card, Debit Card, UPI, Cash) |
| Status | varchar(20) | Payment status with constraint (Pending, Completed, Failed, Refunded) |
| TransactionId | varchar(100) | Unique transaction identifier |

### **5. API Endpoints**

The API provides various endpoints for managing users, restaurants, menu items, and orders.

#### **Authentication & Authorization**

1. **POST /api/auth/login**: Logs a user in and returns a JWT token.
2. **POST /api/auth/register**: Registers a new user (customer or restaurant owner).

#### **Restaurant Management**

1. **GET /api/restaurants**: Gets a list of all restaurants.
2. **GET /api/restaurants/{id}**: Gets details of a specific restaurant by ID.
3. **POST /api/restaurants**: Adds a new restaurant.
4. **PUT /api/restaurants/{id}**: Updates restaurant information.
5. **DELETE /api/restaurants/{id}**: Deletes a restaurant.

#### **Menu Management**

1. **GET /api/menuitems**: Gets a list of all menu items.
2. **GET /api/menuitems/{id}**: Gets details of a specific menu item.
3. **POST /api/menuitems**: Adds a new menu item.
4. **PUT /api/menuitems/{id}**: Updates a menu item.
5. **DELETE /api/menuitems/{id}**: Deletes a menu item.

#### **Customer Management**

1. **GET /api/customers/{id}**: Gets customer details.
2. **PUT /api/customers/{id}**: Updates customer information.
3. **POST /api/orders**: Places a new order.

#### **Order Management**

1. **GET /api/orders**: Gets a list of all orders.
2. **GET /api/orders/{id}**: Gets details of a specific order.
3. **PUT /api/orders/{id}**: Updates order status.

## ***OrderItem Management***

1. **GET /api/orderitems**: Gets a list of all order items
2. **GET /api/orderitems/{id}**: Gets details of a specific order item
3. **POST /api/orderitems**: Creates a new order item
4. **PUT /api/orderitems/{id}**: Updates an existing order item
5. **DELETE /api/orderitems/{id}**: Deletes an order item

## ***Categories Management***

1. **GET /api/categories**: Gets a list of all categories
2. **GET /api/categories/{id}**: Gets details of a specific category
3. **POST /api/categories**: Creates a new category
4. **PUT /api/categories/{id}**: Updates an existing category
5. **DELETE /api/categories/{id}**: Deletes a category

## ***Payments Management***

1. **GET /api/payments**: Gets a list of all payments
2. **GET /api/payments/{id}**: Gets details of a specific payment
3. **POST /api/payments**: Creates a new payment record
4. **PUT /api/payments/{id}**: Updates an existing payment
5. **DELETE /api/payments/{id}**: Deletes a payment record

### **6. Unit of Work Pattern**

The **Unit of Work** pattern is used to group multiple database operations into a single transaction. It ensures that all operations are committed or rolled back as one unit, ensuring data consistency.

* **IUnitOfWork** interface defines methods for accessing repositories.
* **UnitOfWork** class implements this interface and manages the commit or rollback of transactions.
* **Repositories** for each table (User, Restaurant, MenuItem, etc.) implement CRUD operations.

### **7. Frontend (Angular) Overview**

The frontend is developed using Angular to provide an interactive and dynamic user interface. The app has the following key components:

* **Login/Registration Pages**: For user authentication.
* **Restaurant Management Pages**: For restaurant owners to manage their profiles and menu items.
* **Menu Browsing**: For customers to view restaurant menus and place orders.
* **Order Management**: For customers to track and manage their orders.

### **8. Security Considerations**

* **JWT Authentication**: Secures API endpoints and ensures that only authorized users (customers or restaurant owners) can access certain functionalities.
* **Role-Based Access Control (RBAC)**: Restricts access to certain API endpoints based on user roles (e.g., only restaurant owners can manage their menu items).
* **Data Encryption**: Sensitive data such as passwords are encrypted in the database.