QuickBite - Menu Management System

Problem Definition:
Restaurants often struggle with managing digital menus efficiently. Existing systems can be complex, limited in functionality, or not tailored for fast, user-friendly interaction.

QuickBite addresses this by offering a responsive platform for restaurant managers to maintain their menus, collect customer feedback, and manage key settings in a simple, centralized interface.

Scope:

This project focuses on the **admin-side digital menu system**, allowing restaurant owners to:

- Add, update, and remove menu items
- Submit and store restaurant setup details
- View and manage feedback from customers
- Log in and manage restaurant data securely

Team Members:

- Noor Hatem Kadry, ID: 222300477
- Ali Sherif Rashed, ID: 222300474
- Karim Hatem, ID: 222300475

Date: 27th May 2025

Github Link: https://github.com/noorhatem123/QuickBite.git

Table of Contents

Architecture Overview

Introduction:
QuickBite uses a **microservices-based architecture** to isolate each functionality into its own service. This improves modularity, scalability, and fault tolerance.

System Components:

1. Frontend (React + MUI)

- Built with **React.js** using functional components and React Router.
- Styled with **Material UI** for consistent and responsive design.
- Handles routing, user input, form validation, and API interaction.

2. Gateway Service (Node.js + Express)

- Acts as the central router for all microservices.
- Routes incoming requests to the appropriate service (menu, auth, feedback, restaurant).
- Manages centralized routing under `/api/*` paths.

3. Authentication Service

- Handles user registration and login.
- Issues JWTs for authenticated access.
- Provides endpoints:
  ```
  POST /auth/register
  POST /auth/login
  ```

4. Menu Service

- Provides full CRUD capabilities for menu items.
- Endpoints:
  ```
  GET /menu
  POST /menu
  PUT /menu/:id
  DELETE /menu/:id
  ```

5. Restaurant Service

- Accepts restaurant onboarding info through a form.
- Endpoint:
  ```
  POST /api/restaurants
  ```

6. Feedback Service

- Allows admins to collect customer feedback.

- Endpoint:
  `POST /api/feedback`

Service Communication Flow:

- The **frontend** sends HTTP requests to the **gateway** using Axios.
- The **gateway** forwards each request to the appropriate microservice.
- Each microservice connects to its own MongoDB collection.
- Services are isolated but work together via RESTful HTTP.

API Specifications

The QuickBite system uses RESTful APIs across all microservices to interact with data models such as users, menu items, restaurants, and feedback.

1. Authentication API

Method	Endpoint	Description
POST	`/auth/register`	Register a new user
POST	`/auth/login`	Authenticate user and return JWT token

Register – POST /auth/register

```
{
  "username": "johnDoe",
  "password": "password123"
}
```

Response:

```
{
  "message": "User registered successfully",
  "token": "jwt_token_here"
}
```

2. Menu API

Create Menu Item – POST /menu

Method	Endpoint	Description
GET	`/menu`	Retrieve all menu items
POST	`/menu`	Add a new menu item
PUT	`/menu/:id`	Update a specific menu item
DELETE	`/menu/:id`	Delete a menu item

```
{
  "name": "Burger",
  "description": "Beef with cheese",
  "price": 9.99,
  "category": "Main",
  "available": true
}
```

3. Restaurant API

Request:

Method	Endpoint	Description
POST	`/api/restaurants`	Submit restaurant setup info

```
{
  "name": "Pizza Planet",
  "location": "Cairo",
  "category": "Italian"
}
```

4. Feedback API

Method	Endpoint	Description
POST	`/api/feedback`	Submit customer feedback

Request:

```
{
  "message": "Loved the UI!"
}
```

All APIs return appropriate status codes:

- `201` for successful creation
- `200` for success
- `400` for bad requests
- `500` for server errors

Data Models and Schema Definitions

QuickBite uses **Mongoose** to define and interact with MongoDB collections. Each service has its schema tailored to its function.

User Model (`auth-service`)
Used for authentication and login.

```
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true, minlength: 3 },
  password: { type: String, required: true, minlength: 6 }
});

userSchema.pre("save", async function (next) {
  if (!this.isModified("password")) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

userSchema.methods.comparePassword = async function (password) {
  return await bcrypt.compare(password, this.password);
};

module.exports = mongoose.model("User", userSchema);
```

Menu Model (menu-service)
Represents menu items.

```
const mongoose = require("mongoose");

const menuSchema = new mongoose.Schema({
  name: { type: String, required: true, minlength: 3 },
  description: { type: String, required: true },
  price: { type: Number, required: true },
  category: { type: String, required: true },
  available: { type: Boolean, default: true }
});

module.exports = mongoose.model("Menu", menuSchema);
```

Restaurant Model (restaurant-service)
Stores restaurant setup details.

```
const mongoose = require("mongoose");

const restaurantSchema = new mongoose.Schema({
  name: { type: String, required: true },
  location: String,
  category: String,
  ownerId: String,
  logo: String,
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model("Restaurant", restaurantSchema);
```

Feedback Model (feedback-service)
Stores customer comments and suggestions.

```
const mongoose = require("mongoose");

const feedbackSchema = new mongoose.Schema({
  message: { type: String, required: true },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model("Feedback", feedbackSchema);
```

Each model is isolated by service, following the microservices principle of **separation of data and responsibilities**.

Communication Flow Between Microservices

QuickBite uses a microservices architecture where each service is responsible for a specific function. Communication is handled through **HTTP RESTful APIs**, and all requests are routed via the **API Gateway**.

How Services Communicate:

1. Frontend ↔ Gateway

- The React frontend sends all API requests to a single entry point:
  e.g., `http://localhost:8000/api/...`
- The Gateway routes each request to the correct internal service.

Example Requests:

Service	Endpoint	Description
Auth Service	`POST /auth/login`	Handles user login
Menu Service	`GET /menu`	Retrieves menu items
Restaurant Service	`POST /api/restaurants`	Submits restaurant info
Feedback Service	`POST /api/feedback`	Submits user feedback

Microservice Flow Example (Add Menu Item):

1. Admin logs in → JWT is issued.
2. Admin navigates to menu manager → `GET /menu` is called.
3. Admin adds item → `POST /menu` with JWT auth.
4. Menu service stores item in MongoDB.
5. Frontend re-fetches list → renders updated menu.

Summary of Flow:

```
[Frontend] → [API Gateway] → [Target Microservice] → [MongoDB]
```

- All communication is **stateless** using REST.
- **Each service** uses **its own database collection**.
- **Frontend** uses **Axios** to handle HTTP requests.

Database Selection & Justification

Database Used:

MongoDB (NoSQL)

Why MongoDB?

1. *Schema Flexibility*

QuickBite deals with documents that vary in structure — such as menu items, restaurant setups, and feedback. MongoDB allows storing these as flexible, JSON-like documents without enforcing a rigid schema.

2. *Scalability*

MongoDB supports horizontal scaling through **sharding**, making it easy to scale out as the app grows (e.g., more restaurants, menus, users, and orders in the future).

3. *Speed*

MongoDB provides fast **read/write operations**, which is essential for real-time menu updates and quick user interaction.

4. *Developer-Friendly*

Data is stored in a JSON-like format, which aligns perfectly with the frontend's data structure and makes backend–frontend integration seamless.

5. *Service Isolation*

Each microservice (auth, menu, feedback, restaurant) can operate independently on its own MongoDB collection or even its own database.

Collections Used:

Service	Collection	Description
Auth Service	`users`	Stores user credentials (hashed)
Menu Service	`menu`	Stores all menu items
Restaurant Service	`restaurants`	Stores restaurant setup details
Feedback Service	`feedbacks`	Stores customer feedback messages

Testing

QuickBite uses a combination of **manual testing**, **unit tests**, and **integration tests** to ensure that both the backend and frontend components function as expected.

1. Manual Testing

- Performed throughout development via the browser and Postman.
- Includes verifying form inputs, submission behavior, navigation, and feedback.
- Frontend UI is tested for responsiveness and interaction on various screen sizes.

2. API Testing with Postman

- A **Postman collection** is used to test all API endpoints:
  - Auth: `/auth/register`, `/auth/login`
  - Menu: `/menu`, `/menu/:id`
  - Restaurants: `/api/restaurants`
  - Feedback: `/api/feedback`
- Each request is verified for correct status codes and expected responses.

Sample Manual Test Cases:

Scenario	Expected Result
Register with valid credentials	Account created + JWT token returned
Login with wrong password	Error message displayed
Add new menu item	Item saved and displayed in UI
Submit feedback	Message saved and modal closes
Click “Try Demo”	Redirect to /menu with demo data

Tools and Technologies

QuickBite uses a modern tech stack built around JavaScript and designed for rapid development, scalability, and modular architecture.

Frontend

React.js

- Builds the interactive admin dashboard
- Uses React Hooks (`useState`, `useEffect`, etc.)
- Handles routing with **React Router**

Material UI (MUI)

- Pre-styled components for consistent design
- Used for buttons, cards, modals, typography, etc.

Axios

- Handles HTTP requests to the backend

Backend

Node.js + Express

- Backend server and routing for each microservice
- RESTful API design
- JSON parsing, error handling, and middleware configuration

MongoDB + Mongoose

- NoSQL database for flexible schema design
- Mongoose schemas define and validate document structure

JWT (JSON Web Token)

- Stateless authentication and route protection
- Tokens issued on login and stored on the client

bcryptjs

- Hashes passwords securely before saving
- Compares passwords for login authentication

Testing

Postman

- Used to test and validate all REST API endpoints
- Simulates requests to services like auth, menu, restaurant, and feedback

Dev Tools

Tool	**Purpose**
Git & GitHub	Version control and collaboration
VS Code	Main development environment
Figma (optional)	UI planning and layout
Dotenv	Manages environment variables

Conclusion

QuickBite successfully delivers a scalable, user-friendly platform that allows restaurant managers to take full control of their menu operations. The system is designed around modular, microservice-based architecture, which ensures maintainability, separation of concerns, and ease of future expansion.

By combining a responsive React frontend with independently managed backend services, QuickBite enables:

- Secure user authentication
- Fast and flexible menu management
- Easy restaurant onboarding
- Direct feedback collection from users

The system is backed by MongoDB, chosen for its schema flexibility and seamless integration with Node.js and Express. All API interactions were thoroughly tested using Postman and verified manually through the frontend UI.

QuickBite is now ready for deployment or further development, such as adding order placement or customer-facing features.

It stands as a reliable solution to simplify digital restaurant management — built with performance, simplicity, and scalability in mind.