FINAL REPORT

Group Name: **TOGETHER ICT**

**NAMES**

**1.KALEBO EMANUEL MPENDA 24101133370060**

**2.ALIKO OMARY MKONDYA 24101133370010**

## 1. INTRODUCTION

### 1.1 Problem description

Cafeterias today have a hard time keeping up with orders and making sure customers are happy. Using paper tickets or waiting in line can cause delays, mistakes, and doesn't really connect with customers. A simple digital system would make things easier for workers and makes things clearer and faster for customers.

### 1.2.Motivation

Were building a web app called "Welcome to Our Cafeteria" to fix the problems of traditional cafeterias. The idea is to give customers an easy way to check the menu and order food online from their phones. At the same time, it helps staff manage orders and menus more smoothly. Since most people will use it on mobile, we're focusing on making it work really well on smartphones first.

### 1.3.Project objectives

- Build a web app that works great on mobile devices for the cafeteria.
- Let staff easily add, update, view, and delete menu items and orders.
- Add secure login and signup so users can manage their profiles.
- Design a simple, clean interface that looks good and works well on any device.

## 2. SYSTEM REQUIREMENT

### 2.1 Functional Requirements

| **Requirement** | **Description** |
|---|---|

| | |
|---|---|
| User Authentication | System allow user to register, login and manage their profiles securely |
| Menu browsing | All users can view the digital menu, including item names ,description and real time pricing |
| Menu management | Admin can create, read ,update and delete food items and availability status |
| Order placement | Users can add items and submit orders |
| Payment processing | System support basic payment simulation to finalize orders |

2.2 Non-Functional Requirements

| **Requirement** | **description** |
|---|---|
| Mobile first design | UI is optimized for small screens first, ensuring usabilityon smartphones |
| Clean and consistent UI | Application maintain a umiform look and feel a intuitive navigation |
| Organized code | Source code follows best practices with clear separation of concerns |
| Input validation | System validates user inputs and provides graceful error massages |
| performance | Application load quickly and responds without delays |

## 3. **SYSTEM ARCHITECTURE**

### 3.1 Overall System Design

- **The Frontend (The Face):** Built with **HTML, CSS, and JavaScript**. It handles everything from displaying the lunch menu to letting users click "Order".

- **The Backend (The Brain):** Using **Node.js and Express**, It receives "orders"

- **The Database (The Memory):** We use **MongoDB** to store all the important information. This includes things like the list of food items, prices, and who has already paid for their meal.

3.2 Client-Server Architecture Diagram

| **Customer's Mobile Device** | **→ The Connection →** | **Backend Server** | **→ The Storage →** | **MongoDB Database** |
|---|---|---|---|---|
| What it does: Displays the menu and buttons. | How it talks: Sends a secure HTTPS message. | What it does: Processes the order logic via Node.js. | How it talks: Sends a Query (a question). | What it does: Safely stores the order in the Cloud. |
| Format: Uses JSON (a simple list format). | Path: Travels through the RESTful API. | Action: Performs (Create, Read, Update, Delete). | Tool: Uses MongoDB | Data: Remembers all customer info. |

3.3 Database Schema (ERD)

How data is stored

1. Users

This folder keeps track of everyone who uses the app.

- **Name & Email:** To identify the customer.
- **Password:** Kept as secret code for security.
- **Role:** Tells the system if the person is a regular **Customer** or a Cafeteria **Admin**.

2.menu items

This is a list of everything the cafeteria sells.

- **Details:** Includes the food name, description, and an image
- **Price:** The cost of the item
- **Availability:** A simple show if the kitchen has run out of that meal.

3.orders

Every time a customer buys something, a new receipt is created here.

- **Customer Link:** Connects the order to the specific user who bought it.
- **Items List:** A list of exactly what was bought, how many, and the price at that moment.
- **Status:** Shows if the the food is **Pending** or if it's finished

3.4 Authentication Mechanism

Authentication is handled using JSON Web Tokens (JWT). When a user logs in successfully, the server generates a signed JWT and sends it to the client. The client stores this token and includes it in the Authorization header for all subsequent requests to protected API endpoints.

3.5 Security Measures Used

1. Password Hashing: User passwords are hashed using bcrypt before being saved in the database.
2. JWT for Authorization: Provides secure, stateless authentication.
3. HTTPS: Encrypts all data transmitted between client and server.
4. Environment Variables: Sensitive information stored in environment variables.

**4. IMPLENTATION DETAILS**

4.1 Technologies Used

| Component | Technologies |
|---|---|
| Frontend | Html, css ,javascript |
| Backend | Node.js |
| Database | MongoDB |
| Authentication | JSON Web Tokens(JWT),bcrypt |
| Tools | Git, VS code |

4.2 Frontend Implementation

The frontend is built with a mobile-first approach. The landing page showcases the cafeteria's brand value propositions with four key features:

- Fresh & Quality Food
- Quick Service
- Event Catering
- Modern Experience

4.3 Backend Implementation

| Functional Area | Method | Endpoint | Description | Access Level |
|---|---|---|---|---|
| Authentication | POST | /api/auth/register | Allows new users to create a secure account. | Public |
| | POST | /api/auth/login | Authenticates existing users for system entry. | Public |
| Menu Management | GET | /api/menu | Retrieves all current food items, descriptions, and prices. | Public |
| | POST | /api/menu | Adds a new food item to the digital menu. | Admin |
| | PUT | /api/menu/:id | Updates details of an existing menu item by its ID. | Admin |
| | DELETE | /api/menu/:id | Permanently removes a food item from the menu. | Admin |
| Order System | POST | /api/orders | Processes a new food order submitted by a user. | User |
| | GET | /api/orders/:userId | Retrieves the full order history for a specific user. | User |

| Functional Area | Method | Endpoint | Description | Access Level |
|---|---|---|---|---|
| Reservations | POST | /api/reservations | Records a new table booking in the system. | User |
| | GET | /api/reservations/:userId | Displays all upcoming bookings for a specific user. | User |

4.4 Database Design

MongoDB was chosen for its flexibility with JSON-like documents. Mongoose schemas enforce consistent data structure and provide built-in validation.

4.5 Challenges Faced

1. Managing State on Frontend: Solved by implementing a simple JavaScript state object.
2. Secure Token Storage: Used localStorage for simplicity.
3. Time :More time in preparing codes

4.6 Assumptions Made

1. Cafeteria staff have separate interface for menu/order management.
2. Application designed for single location.
3. Payment processing handled offline (cash/card at pickup).

**5.SECURITY CONSIDERATIONS**

5.1 Authentication Handling

- JWT tokens issued upon successful login

5.2 Password Storage

- Passwords are not saved as normal text for safety

- The system uses bcrypt to change passwords into secure coded form

- Only the coded password is saved in the database and real password is not kept.

5.3 Data Protection

- Input validation on both client and server sides

- MongoDB uses safe query method to stop injection attacks

5.4 Access Control Implementation

The system uses role-based access control to manage user permissions. This means users can only access features that match their roles. For example, an admin has more permissions than a normal user.

The system checks the user's role before allowing any action. If the user does not have the correct role, the request is blocked and an "Access denied" message is shown.

This approach helps protect important parts of the system by ensuring that only authorized users can perform sensitive tasks. For instance, only an admin is allowed to delete menu items, and other users cannot perform this action.

**6. TESTING AND RESULTS**

6.1 Screenshots of the Application

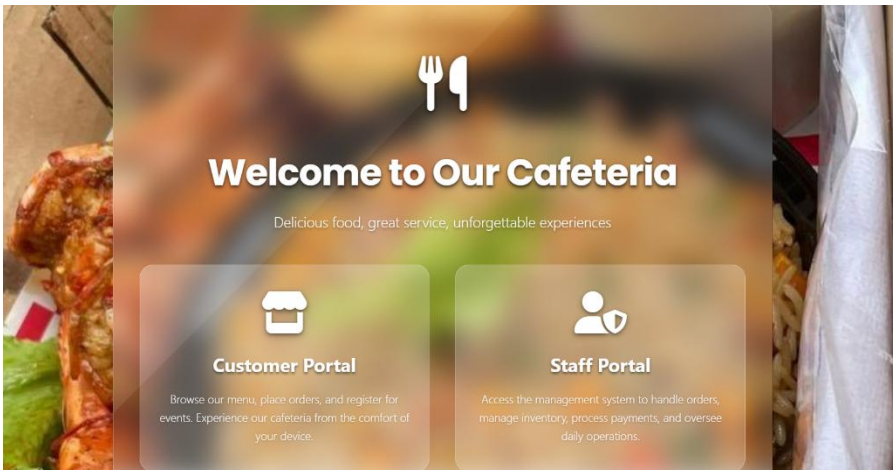


Figure 1:Landing Page (Mobile View) - Shows the "Welcome to Our Cafeteria" homepage.

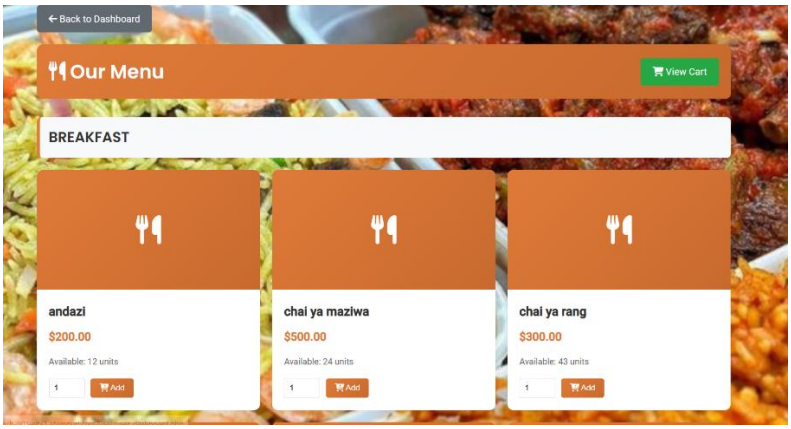


Figure 2:Menu Page - Displays food items with descriptions, prices, and "Add to Order" buttons.

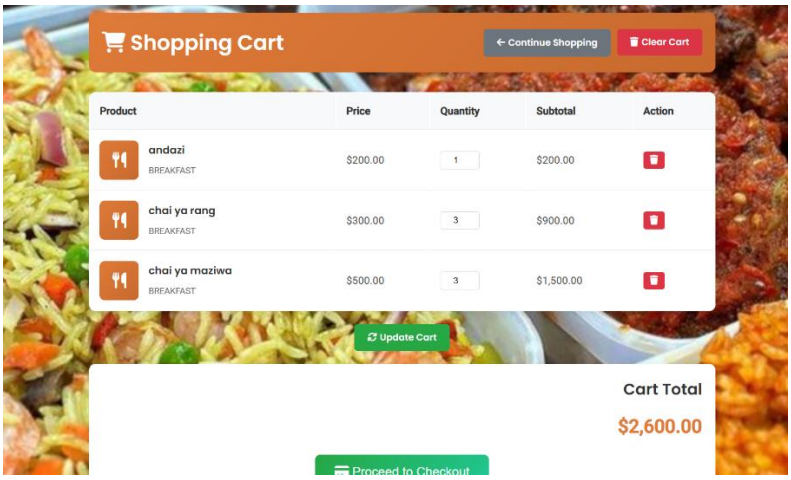


Figure 3:Order Cart - Shows selected items, quantities and total price

6.2 Explanation of Major Features Working

Feature 1: User Authentication

Users can successfully register and log in.

Feature 2: Menu Management (Admin)

Admin users can access an interface to add new items which immediately appear in the public menu. They can also update prices or mark items as unavailable.

Feature 3: Order Placement

Authenticated users can add items to their cart and place orders. Each order is saved in the database, linked to the user ID, and appears in their order history

7. Future Work

7.1 Possible Improvements

1. Real-time Features
2. Full Admin Dashboard .
3. Online Payment Integration
4. Push Notifications.
5. User Profile Management
6. Rating and Reviews

7.2 Scalability Consideration

Scalability methods

-Database optimization

-caching

-load balancing

-indexing: searching for orders remain fast as database grows