Restaurant Management Suite - Capstone Project

Short Description

This is a 5-day capstone project designed to consolidate and demonstrate all the technical skills acquired throughout the 11-week software engineering course. You will build a real-world restaurant management application that incorporates user authentication, menu management, staff scheduling, and basic analytics. The project emphasizes practical application of programming fundamentals, web development, backend APIs, security practices, and deployment using Docker. This comprehensive exercise serves as a portfolio piece that showcases your full-stack development capabilities and modern software engineering practices.

Project Overview

You will build a comprehensive restaurant operations system that handles menu management and staff scheduling. This capstone project allows you to demonstrate all the skills learned throughout the course by creating a full-stack application with proper architecture, security, and deployment practices.

Core Functional Requirements

1. User Authentication & Authorization System

User Management

- User Registration: Implement a secure registration form that collects email, password, name, and other relevant information
- User Login/Logout: Create a secure authentication system with proper session management
- Role-Based Access Control: Design and implement different permission levels for different user types
- Password Management: Include password reset functionality with email verification (can be simulated) and change password capability
- Profile Management: Allow users to update their personal information

Required User Roles

- Customer: Can view menu only
- Staff/Server: Can view their schedules and update their availability

- Manager: Can access all features, manage staff schedules, and view reports
- Admin: Can manage users, system settings, and all operations

Technical Requirements

- Implement secure password hashing
- Store authentication tokens securely
- Create middleware for role-based route protection
- Implement proper validation for all user inputs
- Handle authentication errors gracefully

2. Menu Management System

Menu Features

- Menu Display: Create an organized menu interface with categories (appetizers, mains, desserts, beverages)
- Item Management: Implement complete CRUD operations for menu items with descriptions and prices
- Image Upload: Add support for food photography and menu item images (manager+ permission only)
- Availability Control: Create functionality to mark items as available/unavailable (manager+ permission only)
- Pricing Management: Implement dynamic pricing with the ability to set specials and discounts (manager+ permission only)
- Multi-language Support (Optional): Add support for displaying menu in multiple languages

Technical Requirements

- Design a flexible database schema for menu items and categories
- Implement secure file upload for images with proper validation
- Create a responsive menu display for different device sizes
- Implement search and filtering capabilities
- Add proper error handling for all operations

3. Staff Scheduling System

Scheduling Features

- Weekly Schedule Creation: Allow managers to create and manage weekly staff schedules
- Staff Assignment: Implement functionality to assign staff members to specific shifts and roles (server, host, cleaner, etc.)

- Shift Management: Create interface for creating, editing, and deleting shifts with specific time slots
- Schedule Viewing: Provide staff with ability to view their assigned schedules
- Shift Coverage: Implement tracking for which shifts are covered and which need staff assignment
- Weekly Overview: Create a visual calendar showing all staff assignments for the week

Technical Requirements

- Design a database schema that supports recurring shifts
- Implement proper date and time handling
- Create a visual calendar interface
- Add conflict detection for double-booking prevention
- Implement proper validation for all scheduling operations
- Add notification system for schedule changes (can be simulated)

4. Reporting & Analytics Dashboard

Analytics Features

- Popular Items: Create analytics on most viewed or ordered menu items
- Staff Scheduling Analytics: Implement tracking for schedule coverage and staff utilization
- Menu Analytics: Add tracking for pricing and theoretical profits
- System Usage Reports: Include tracking for user activity and system performance

Technical Requirements

- Design efficient queries for aggregating data
- Create reusable chart components for data visualization
- Implement filtering and date range selection
- Ensure optimized performance for analytics calculations
- Add export functionality for reports (CSV/PDF)

Technical Requirements

Backend Technology Options

Choose ONE of the following backend technologies:

- Node.js + Express (JavaScript/TypeScript)
- Python + Flask (Python)

Frontend Technology Options

Choose ONE of the following frontend technologies:

- HTML/CSS/JavaScript (vanilla or with a framework like jQuery)
- React.js

Database Options

Choose ONE of the following database solutions:

- SQLite (recommended for simplicity and portability)
- File-based JSON (static files)

Authentication Options

Choose ONE of the following authentication methods:

- JWT (JSON Web Tokens)
- Session-based authentication

System Architecture Requirements

API Design

- RESTful API: Design clean, consistent API endpoints following REST principles
- Error Handling: Implement proper error responses with appropriate status codes
- Input Validation: Validate all user inputs on both client and server sides
- Rate Limiting: Implement basic rate limiting for API protection
- Documentation: Document all API endpoints and their usage

Security Requirements

- Data Validation: Sanitize all user inputs to prevent injection attacks
- Password Security: Hash passwords using secure algorithms (bcrypt/Argon2)
- XSS Prevention: Implement measures to prevent cross-site scripting
- CORS Configuration: Properly configure cross-origin resource sharing
- Secure Headers: Implement security headers for web protection
- Session Management: Ensure secure session handling and timeout

DevOps & Deployment Requirements

Version Control

- Git Repository: Use Git for version control with meaningful commit messages

- Branching Strategy: Implement feature branches and proper merge practices
- Code Review: Use pull requests for code review process (can be self-reviews)
- Documentation: Maintain README and setup instructions

Containerization

- Docker: Containerize the application using Docker
- Docker Compose: Set up multi-container development environment if needed
- Environment Variables: Use environment variables for configuration
- Volume Management: Implement proper data persistence in containers

Deployment Instructions

You must provide straightforward explanation of how to run the backend and the frontend using Docker:

- Clear step-by-step instructions
- Environment setup guidance
- Container build and run commands
- Troubleshooting tips

User Interface Requirements

Responsive Design

- Desktop Interface: Create a full-featured desktop experience
- Mobile Responsiveness: Ensure the application works on various screen sizes
- Cross-Browser Compatibility: Test with major web browsers

User Experience

- Error Messages: Implement clear, helpful error messages
- Confirmation Dialogs: Add confirmation for destructive actions
- Real-time Updates: Implement live updates for schedules and menu changes when possible
- Loading States: Add proper loading indicators
- Intuitive Navigation: Create a logical information architecture

Data Management Requirements

Data Models

Design appropriate data structures for:

- Users and authentication
- Menu items and categories
- Staff information and availability
- Weekly schedules and shift assignments

Data Integrity

- Validation Rules: Implement business logic validation
- Referential Integrity: Maintain proper relationships between data
- Migration Support: Handle database schema changes gracefully
- Data Persistence: Ensure data is properly saved and retrieved

Testing Requirements

Testing Types

- Unit Testing: Test individual functions and components
- Integration Testing: Test API endpoints and database interactions
- Manual Testing: Document test cases for critical features

Test Coverage

- Core Features: All main features should have tests
- Edge Cases: Test error conditions and edge cases
- Security Testing: Test authentication and authorization

Documentation Requirements

Required Documentation

- README: Provide clear setup and running instructions
- API Documentation: Document all endpoints with examples
- Database Schema: Document data models and relationships
- Deployment Guide: Include step-by-step deployment instructions
- Feature Guide: Explain how to use each feature

Agile Development Practices

Required Agile Practices

- Maintain a task board (Trello, GitHub Projects, etc.)
- Break down work into small, manageable tasks
- Track progress daily
- Conduct personal daily stand-ups (written format is acceptable)
- Prioritize features based on core requirements
- Adjust plan as needed based on progress

Documentation of Agile Process

- Include screenshots of your task board at different stages
- Document daily progress in a log
- Note any challenges and how they were addressed

Evaluation Criteria

Functionality (40%)

- All core features implemented and working
- Proper error handling and edge cases
- User experience and interface quality
- Feature completeness according to requirements

Technical Implementation (30%)

- Code quality and organization
- Proper use of chosen technologies
- Security implementation
- Database design and optimization
- Adherence to software engineering principles

DevOps & Deployment (20%)

- Docker implementation
- Git usage and commit history
- Successful deployment
- Environment configuration
- Documentation quality

Documentation & Presentation (10%)

- Quality of written documentation

- Clear setup instructions
- Project demonstration
- Code comments and readability

Hints

- Plan thoroughly before coding: Spend adequate time on Day 1 planning your architecture
- Focus on core functionality first: Implement the essential features before adding enhancements
- Commit code frequently: Make small, regular commits with meaningful messages
- Test continuously: Don't leave all testing to the end
- Document as you go: Update documentation regularly as you implement features
- Use time management: Allocate specific time blocks for different aspects of the project
- Ask for help when stuck: Don't spend too long on a single problem
- Leverage Docker early: Set up your containerization at the beginning

To Submit

Code Deliverables

- Complete source code with proper organization
- Docker configuration files
- Configuration files and environment examples

Documentation Deliverables

- All required documentation as specified above
- Demo video (5-10 minutes) showcasing the application

Deployment Deliverables

- Working dockerized application
- Deployment scripts and instructions
- Environment configuration guide
- Testing scripts and instructions

This capstone project is your opportunity to showcase everything you've learned throughout the course and create a substantial portfolio piece that demonstrates your full-stack development capabilities. Good luck!