Software Design Specification (SDS)

Restaurant Management System (RMS)

Prepared for Academic Submission

December 2025

Muhammad Saad Bilal

Muhammad Hashim Mughal

Eisha-Tur-Raazia

Contents

1 Introduction **3**

1.1 Purpose . . . 3

1.2 Scope . . . 3

1.3 Definitions and Acronyms . . . 3

1.4 References . . . 3

2 System Overview **4**

3 Architectural Design **4**

3.1 High-Level Architecture . . . 4

3.2 Subsystem Descriptions . . . 4

4 Detailed Design **4**

4.1 Component Design . . . 4

4.2 Interface Design . . . 5

4.3 Data Design . . . 5

4.4 Behavior Modeling . . . 5

4.5 Algorithm Design . . . 6

5 User Interface Design **6**

6 External Interface Design **6**

7 Non-Functional Requirements **6**

8 Design Constraints **6**

9 Assumptions and Dependencies **6**

10 Design Decisions and Trade-offs **7**

11 Appendices **7**

11.1 Future Enhancements . . . 7

1 Introduction

1.1 Purpose

This Software Design Specification (SDS) describes the architectural and detailed design of the Restaurant Management System (RMS). It serves as a technical realization of the Software Requirements Specification (SRS) by defining how system components, data structures, interfaces, and workflows are constructed and interact. The document provides developers, testers, and evaluators with a clear and consistent design reference for implementation and validation.

1.2 Scope

The RMS is a web-based system intended to digitize restaurant operations by enabling QR-based ordering, managerial approval, kitchen order tracking, and customer feedback. This SDS covers the system architecture, component design, database schema, interface definitions, behavioral models, and key algorithms. Features such as authentication, POS integration, and multi-branch support are excluded from the current scope and identified as future enhancements.

1.3 Definitions and Acronyms

RMS	Restaurant Management System
SRS	Software Requirements Specification
SDS	Software Design Specification
API	Application Programming Interface
ERD	Entity Relationship Diagram
KDS	Kitchen Display System
REST	Representational State Transfer

1.4 References

- Software Requirements Specification (RMS v1.0)
- Assignment 3 – Software Design Specification Guidelines
- Express.js Documentation
- React Documentation
- Supabase / PostgreSQL Documentation

2 System Overview

The Restaurant Management System is a modular, web-based application designed to streamline restaurant workflows. Customers access the system through mobile browsers to view menus and place orders. Managers approve and monitor orders through a dashboard, while kitchen staff track preparation status on a real-time display. A centralized backend coordinates all communication and ensures data consistency across subsystems.

The system is intended for deployment within a restaurant's local network, with optional cloud synchronization for persistence and backup.

3 Architectural Design

3.1 High-Level Architecture

The RMS follows a three-tier architecture to promote scalability and separation of concerns. The presentation layer consists of a React-based frontend providing interfaces for customers, managers, and kitchen staff. The application layer is implemented using Node.js with the Express framework and encapsulates all business logic and workflow rules. The data layer is backed by a Supabase-hosted PostgreSQL database, ensuring persistent storage and relational integrity. Communication between layers occurs via stateless RESTful APIs over HTTPS.

Architecture diagram placeholder

3.2 Subsystem Descriptions

The Customer Subsystem handles menu display, cart management, order submission, and order status visualization. The Manager Subsystem enables order approval, menu management, and operational monitoring. The Kitchen Subsystem displays approved orders and allows kitchen staff to update preparation status. The Backend Subsystem enforces business rules and mediates access to the database. The Database Subsystem provides durable storage and supports analytical queries and auditing.

4 Detailed Design

4.1 Component Design

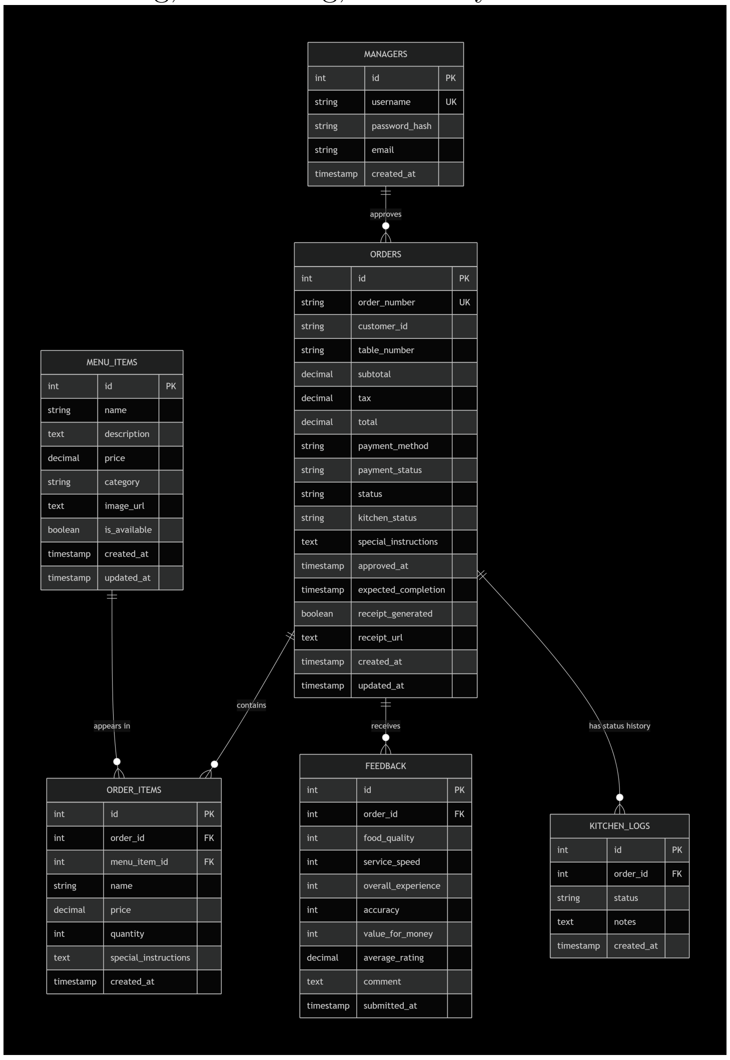
The backend is organized into controller-based components aligned with business domains. The Order Controller manages the lifecycle of orders, including creation, approval, and status transitions. The Menu Controller manages retrieval and modification of menu data. The Feedback Controller processes customer feedback after order completion. A Receipt Service generates HTML-based receipts and stores their references for retrieval. The frontend consists of reusable React components corresponding to distinct screens and workflows.

4.2 Interface Design

The backend exposes RESTful API endpoints that exchange JSON-formatted data. These interfaces support order placement, approval, status updates, menu management, and feedback submission. The stateless nature of the APIs enables scalability and loose coupling between frontend and backend.

4.3 Data Design

The database schema is derived from the ERD and implemented using PostgreSQL via Supabase. Core entities include orders, menu items, order items, kitchen logs, feedback, and managers. Referential integrity is enforced using primary and foreign keys. Timestamp attributes enable auditing, monitoring, and analytics.



MANAGERS		
int	id	PK
string	username	UK
string	password_hash	
string	email	
timestamp	created_at	

ORDERS		
int	id	PK
string	order_number	UK
string	customer_id	
string	table_number	
decimal	subtotal	
decimal	tax	
decimal	total	
string	payment_method	
string	payment_status	
string	status	
string	kitchen_status	
text	special_instructions	
timestamp	approved_at	
timestamp	expected_completion	
boolean	receipt_generated	
text	receipt_url	
timestamp	created_at	
timestamp	updated_at	

MENU_ITEMS		
int	id	PK
string	name	
text	description	
decimal	price	
string	category	
text	image_url	
boolean	is_available	
timestamp	created_at	
timestamp	updated_at	

ORDER_ITEMS		
int	id	PK
int	order_id	FK
int	menu_item_id	FK
string	name	
decimal	price	
int	quantity	
text	special_instructions	
timestamp	created_at	

FEEDBACK		
int	id	PK
int	order_id	FK
int	food_quality	
int	service_speed	
int	overall_experience	
int	accuracy	
int	value_for_money	
decimal	average_rating	
text	comment	
timestamp	submitted_at	

KITCHEN_LOGS		
int	id	PK
int	order_id	FK
string	status	
text	notes	
timestamp	created_at	

ERD

4.4 Behavior Modeling

System behavior follows a controlled order lifecycle. A customer submits an order, which remains pending until approved by a manager. Approved orders are forwarded to the kitchen, where staff update preparation states until completion. A finite state machine governs valid transitions, ensuring system consistency.

Sequence and state machine diagram placeholders

4.5 Algorithm Design

Order placement involves validating cart data, computing pricing, persisting order records, and returning confirmation. Order approval triggers receipt generation and kitchen notification. These algorithms are deterministic and designed to prevent inconsistent states.

5 User Interface Design

The customer interface is mobile-first and optimized for simplicity. The manager dashboard presents dense operational information for decision-making. The kitchen display prioritizes high-contrast visuals and minimal interaction to support fast-paced environments.

6 External Interface Design

The system interfaces with customer smartphones, managerial tablets, and kitchen displays through modern web browsers. Supabase provides database services, while payment processing is handled by an abstracted third-party gateway. All communication uses HTTPS with JSON payloads.

7 Non-Functional Requirements

The design emphasizes near real-time performance, secure communication, high reliability, and scalability through stateless services. Maintainability is achieved through modular design and clear separation of concerns.

8 Design Constraints

The current system does not include authentication or POS integration. It relies on third-party payment gateways and assumes reliable local network connectivity.

9 Assumptions and Dependencies

It is assumed that customers possess smartphones, staff are familiar with touch interfaces, and Supabase and payment services remain available. The system depends on modern browser compatibility and stable network infrastructure.

10 Design Decisions and Trade-offs

Several design decisions were made to balance simplicity and extensibility. The selection of Express.js and React prioritizes rapid development and ecosystem support over stricter compile-time guarantees. Supabase reduces database administration effort at the cost of lower-level configuration control. Authentication was deferred to focus development on core ordering workflows, acknowledging reduced security in early versions. Real-time updates are handled through lightweight mechanisms rather than persistent WebSockets to minimize infrastructure complexity. These trade-offs favor clarity, modularity, and academic suitability.

11 Appendices

11.1 Future Enhancements

Future work includes role-based authentication, POS integration, multi-branch deployment, and advanced analytics dashboards.