Restaurant Management System

Requirements Specification

Version 1.0

March 27, 2025

**Table of Contents**

[**RESTORANT MANAGEMENT SYSTEM REQUIREMENTS SPECIFICATION 1**](#_heading=h.19c6y18)

[**VERSION 1.0 1**](#_heading=h.3tbugp1)

[**APRIL 19, 2021 1**](#_heading=h.28h4qwu)

[**1.**](#_heading=h.nmf14n) **EXECUTIVE SUMMARY 3**

[1.1](#_heading=h.37m2jsg) Project Overview 3

[1.2](#_heading=h.1mrcu09) Purpose and Scope of this Specification 3

[**2.**](#_heading=h.46r0co2) **PRODUCT/SERVICE DESCRIPTION 3**

[2.1](#_heading=h.2lwamvv) Product Context 3

[2.2](#_heading=h.111kx3o) User Characteristics 3

[2.3](#_heading=h.3l18frh) Assumptions 3

[2.4](#_heading=h.206ipza) Constraints 3

[2.5](#_heading=h.4k668n3) Dependencies 4

[**3.**](#_heading=h.2zbgiuw) **REQUIREMENTS 4**

[3.1](#_heading=h.1egqt2p) Functional Requirements 5

[3.2](#_heading=h.3ygebqi) Non-Functional Requirements 5

[*3.2.1*](#_heading=h.2dlolyb) *Product Requirements 5*

[**3.2.1.1**](#_heading=h.44sinio) **User Interface Requirements** 6

[**3.2.1.2**](#_heading=h.2jxsxqh) **Usability** 6

[**3.2.1.3**](#_heading=h.z337ya) **Efficiency** 6

[3.2.1.3.1](#_heading=h.3j2qqm3) Performance Requirements 6

[3.2.1.3.2](#_heading=h.1y810tw) Space Requirements 6

[**3.2.1.4**](#_heading=h.4i7ojhp) **Dependability** 6

[**3.2.1.5**](#_heading=h.2xcytpi) **Security** 7

[*3.2.2*](#_heading=h.sqyw64) *Organizational Requirements 7*

[**3.2.2.1**](#_heading=h.3whwml4) **Environmental Requirements** 7

[**3.2.2.2**](#_heading=h.2bn6wsx) **Operational Requirements** 7

[**3.2.2.3**](#_heading=h.qsh70q) **Development Requirements** 7

[*3.2.3*](#_heading=h.3cqmetx) *External Requirements 7*

[**3.2.3.1**](#_heading=h.1pxezwc) **Regulatory Requirements** 7

[**3.2.3.2**](#_heading=h.49x2ik5) **Ethical Requirements** 7

[**3.2.3.3**](#_heading=h.2p2csry) **Legislative Requirements** 7

[3.2.3.3.1](#_heading=h.147n2zr) Accounting Requirements 7

[3.2.3.3.2](#_heading=h.3o7alnk) Security Requirements 7

[3.3](#_heading=h.1rvwp1q) Domain Requirements 7

[**4.**](#_heading=h.4bvk7pj) **USER SCENARIOS/USE CASES 7**

# Executive Summary

## Project Overview

The Restaurant Management System is a web-based application that allows customers to browse the menu, make reservations, and place orders; waiters to manage orders and billing; and managers to oversee restaurant operations, including inventory, menu items, and tables.

## Purpose and Scope of this Specification

This document outlines the requirements for the restaurant management system. It is intended for developers, testers, and stakeholders to align on the system’s goals.

In Scope:

-Role-based features for Customers, Waiters, and Managers

-Menu, ordering, and reservation features

-Payment handling and table management

Out of Scope:

-Kitchen display systems

-Loyalty/reward program

# Product/Service Description

## Product Context

This is a standalone application with a user-friendly interface. It may connect with external payment gateways for card payments.

## User Characteristics

Customers: General users with no technical expertise

Waiters: Basic computer skills

Managers: Intermediate technical skills for dashboard use

## Assumptions

-Users have internet access

-A browser-based interface is sufficient

-Admin roles are predefined in the system

## Constraints

-The app must work on standard modern browsers

-Must support up to 100 concurrent users

-Payment integration must be PCI compliant

## Dependencies

-Third-party payment API

-Database system (e.g., MySQL or PostgreSQL)

-Authentication services

# Requirements

* Describe all system requirements in enough detail for designers to design a system satisfying the requirements and testers to verify that the system satisfies requirements.
* Organize these requirements in a way that works best for your project. See Appendix DAppendix D, Organizing the Requirements for different ways to organize these requirements.
* Describe every input into the system, every output from the system, and every function performed by the system in response to an input or in support of an output. (Specify what functions are to be performed on what data to produce what results at what location for whom.)
* Each requirement should be numbered (or uniquely identifiable) and prioritized.

See the sample requirements in Functional Requirements, and System Interface/Integration, as well as these example priority definitions:

**Priority Definitions**

The following definitions are intended as a guideline to prioritize requirements.

* Priority 1 – The requirement is a “must have” as outlined by policy/law
* Priority 2 – The requirement is needed for improved processing, and the fulfillment of the requirement will create immediate benefits
* Priority 3 – The requirement is a “nice to have” which may include new functionality

It may be helpful to phrase the requirement in terms of its priority, e.g., "The value of the employee status sent to DIS **must be** either A or I" or "It **would be nice** if the application warned the user that the expiration date was 3 business days away". Another approach would be to group requirements by priority category.

* A good requirement is:
* Correct
* Unambiguous (all statements have exactly one interpretation)
* Complete (where TBDs are absolutely necessary, document why the information is unknown, who is responsible for resolution, and the deadline)
* Consistent
* Ranked for importance and/or stability
* Verifiable (avoid soft descriptions like “works well”, “is user friendly”; use concrete terms and specify measurable quantities)
* Modifiable (evolve the Requirements Specification only via a formal change process, preserving a complete audit trail of changes)
* Does not specify any particular design
* Traceable (cross-reference with source documents and spawned documents).

## Functional Requirements

| **Req#** | **Requirement** | **Priority** |
| --- | --- | --- |
| FR-CUST-01 | Customers shall be able to register and log in. | 1 |
| FR-CUST-02 | Customers shall view the menu without logging in. | 1 |
| FR-CUST-03 | Customers shall place orders and make payments online. | 1 |
| FR-CUST-04 | Customers shall book table reservations. | 2 |
| FR-CUST-05 | Customers shall see a homepage with featured items and announcements. | 3 |
| FR-WAIT-01 | Waiters shall take orders on behalf of customers. | ~~1~~ |
| FR-WAIT-02 | Waiters shall view but not edit table information. | ~~2~~ |
| FR-WAIT-03 | Waiters shall generate bills and process payments. | ~~1~~ |
| FR-MGR-01 | Managers shall manage menu items (add/edit/delete). | ~~1~~ |
| FR-MGR-02 | Managers shall view and update inventory. | ~~2~~ |
| FR-MGR-03 | Managers shall view daily revenue reports. | ~~2~~ |
| FR-MGR-04 | Managers shall edit and delete table information. | ~~1~~ |

## Non-Functional Requirements

### Product Requirements

-User Interface Requirements:

Responsive UI for mobile and desktop

Clear navigation with role-based dashboards

-Usability:

Learnable within 10 minutes for new users

Tooltips and contextual help for forms and dashboards

-Efficiency:

95% of operations must complete within 1 second

Application supports 100 simultaneous users

-Space Requirements:

Max database size: 5 GB for the first year

Application codebase should not exceed 200 MB

-Dependability:

System availability: 99.5%

Uptime tracking and error logging enabled

-Security:

Login via secure protocol (HTTPS)

Role-based access control

Sensitive data encrypted at rest and in transit

#### **User Interface Requirements**

The application shall provide separate dashboards for Customers, Waiters, and Managers with clear visual distinctions.

The interface shall be responsive and function correctly on both desktop and mobile browsers.

Menu screens must include food categories, item names, prices, and images.

Reservation and order forms shall use dropdowns, calendars, and real-time input validation.

Error messages must be clearly displayed next to input fields with user-friendly explanations.

Managers shall have access to summary reports with tabular and chart-based data presentation.

Users should be able to navigate using a sidebar or top navigation bar, with breadcrumb trails on deeper pages.

#### **Usability**

The system shall be easy to use and require no more than 10 minutes to learn basic tasks.

Online documentation shall be provided for each user role and linked within the system.

Help icons should be available next to complex inputs to show task-specific tooltips.

The system shall offer a guided walkthrough on first login per role.

Common tasks like placing orders or managing tables should be no more than 3 clicks away from the dashboard.

#### **Efficiency**

##### Performance Requirements

The system shall support at least 100 concurrent users during peak hours.

95% of transactions (e.g., order submission, reservation confirmation) shall complete in less than 1 second.

The system shall be capable of handling at least 1,000 transactions per day.

Waiters and Managers should not experience more than 1-second delay for inventory or bill generation tasks.

##### Space Requirements

The application shall not exceed 200 MB for the deployed server-side code.

The database shall support up to 5 GB of data storage in the first operational year.

Image uploads for menu items shall be compressed to a maximum of 1 MB per image.

#### **Dependability**

**Availability**

The system shall be available 24/7, with downtime restricted to scheduled maintenance between 2–4 AM local time.

System uptime must be at least 99.5% monthly.

The system must be usable across all geographic areas with a stable internet connection.

Downtime must trigger a system-wide maintenance banner and notify the admin dashboard.

**Reliability**

Mean Time Between Failures (MTBF) must exceed 500 hours.

The system shall allow retry for failed transactions such as payment processing.

Transaction logs shall include time, user, and action for auditing and debugging.

**Monitoring**

All critical services must include logging for errors and warnings.

Admin shall receive email alerts for server crashes or failed login attempts.

**Maintenance**

The codebase shall be modular and documented to support future feature additions.

System logs must be retained for 6 months and accessible via the Manager’s dashboard.

**Integrity**

Order, payment, and reservation data must include checksums or versioning to prevent corruption or overwrites.

Role-based permissions will restrict access to CRUD operations on sensitive data.

#### **Security**

All communication shall be encrypted using HTTPS.

User passwords shall be hashed and stored securely.

The system shall implement role-based access control for Customers, Waiters, and Managers.

Activity logs shall be generated for login attempts, data edits, and admin actions.

External payments shall be processed via a PCI-DSS-compliant payment provider.

Data validation must be enforced both client-side and server-side to prevent injection or spoofing attacks.

### Organizational Requirements

Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc

#### **Environmental Requirements**

The system shall be developed and deployed on cloud-based environments compatible with Linux.

The client interface must run on all major modern browsers (Chrome, Firefox, Safari, Edge).

No specialized hardware shall be required—desktop or tablet with internet access is sufficient.

#### **Operational Requirements**

The system shall support real-time operations (order taking, billing) with a backend API that supports live data interactions.

Daily backups must occur automatically at 3:00 AM and be restorable within 1 hour.

Logs and reports must be downloadable from the Manager dashboard for audit or analysis.

#### **Development Requirements**

The system shall be built using standard web technologies: HTML, CSS, JavaScript (React for frontend), Node.js or Django for backend.

All code must follow industry standard documentation and use Git for version control.

Testing frameworks (e.g., Jest, Selenium) shall be used for automated testing.

### External Requirements

These are factors driven by external entities such as governments, regulators, or industry standards.

#### **Regulatory Requirements**

Payment processes must comply with PCI DSS standards.

The system must support user requests to export or delete their personal data, as mandated by GDPR.

#### **Ethical Requirements**

User data shall only be used for the intended restaurant functionality (e.g., order tracking, reservations).

The system shall not store or use customer data for marketing without explicit consent.

#### **Legislative Requirements**

All system logs, especially financial and administrative actions, must be recorded and retained for at least 6 months.

Sensitive fields such as card numbers, passwords, and personal IDs shall never be stored in plaintext.

##### Accounting Requirements

The system shall produce detailed, itemized receipts for each transaction, with tax information when applicable.

Daily revenue reports must include total sales, breakdown by payment method, and total taxes collected.

##### Security Requirements

Two-Factor Authentication (2FA) must be enabled for Manager logins.

Session expiration shall be enforced after 30 minutes of inactivity.

All endpoints must validate user permissions to prevent unauthorized access.

## Domain Requirements

Each food item belongs to a category (e.g., appetizer, main course, dessert).

A reservation is associated with a table, a customer, a date and time, and a party size.

Waiters serve specific tables; this association may be updated by the Manager.

Inventory must track quantities of raw ingredients used per dish.

Menu items may be marked as temporarily unavailable if inventory is insufficient.

# User Scenarios/Use Cases

**Use Case 1**: Customer Places an Order

Actor: Customer

Goal: To order food online and pay via card.

Steps:

1.Login or continue as guest

2.Browse menu and add items to cart

3.Proceed to checkout

4.Enter payment details and confirm

5.Receive confirmation page with estimated delivery time1.

**Use Case 2:** Waiter Takes a Table Order

Actor: Waiter

Goal: To place an order for a customer dining in

Steps:

1.Login as Waiter

2.Select an assigned table

3.Add menu items to the order

4.Submit to kitchen

5.Print or view bill when the customer requests

**Use Case 3**: Manager Edits the Menu

Actor: Manager

Goal: To add a new seasonal item to the menu

Steps:

1.Login as Manager

2.Navigate to “Menu Management”

3.Click “Add Item”, fill in name, price, image, category

4.Submit and verify the item appears on the customer menu

APPENDIX

1. **Definitions, Acronyms, and Abbreviations**

RMS: Restaurant Management System

CRUD: Create, Read, Update, Delete

PCI DSS: Payment Card Industry Data Security Standard

2FA: Two-Factor Authentication

MTBF: Mean Time Between Failures

GDPR: General Data Protection Regulation

1. **References**

PCI DSS Guidelines: https://www.pcisecuritystandards.org

Usability Standards: http://www.usabilitynet.org

GDPR Compliance: https://gdpr.eu/

React Documentation: https://reactjs.org/docs/getting-started.html

Django Documentation: <https://docs.djangoproject.com/en/>

1. **Requirements Traceability Matrix**

The following matrix demonstrates how each business requirement (BR) connects to design and testing deliverables. Naming follows the pattern:

FunctionalArea-DocType-Number (e.g., ORD-UCD-01 = Order Use Case Diagram 01).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BizReqID** | **Area** | **ID** | **Deliverable Name** | **Status** |
| BR\_CUST\_01 | ORD | ORD-UCD-01 | Customer Login & Registration Use Case Diagram | Accepted |
| BR\_CUST\_02 | MENU | MENU-PF-01 | Menu Browsing Process Flow | Accepted |
| BR\_WAIT\_01 | ORD | ORD-UCD-02 | Waiter Order Management Use Case Diagram | In review |
| BR\_MGR\_01 | INV | INV-UCD-01 | Inventory and Menu Admin Use Case Diagram | Accepted |
| BR\_MGR\_02 | REP | REP-UI-01 | Revenue Report UI Mockup | ReadyForReview |
| BR\_ALL\_01 | SEC | SEC-TC-01 | Login & Role Access Test Case | ReadyForReview |

1. **Organizing the Requirements**

This section is provided to aid in understanding and preparing detailed requirements. Requirements can be extensive and should be organized in a meaningful and scalable way. The following organization techniques may be considered:

**By System Mode**

Although the Restaurant Management System does not operate in different system modes (e.g., training vs. emergency), modular flows like customer mode, waiter mode, and manager mode can be conceptually considered as modes for organizational clarity.

**By User Class**

This is the primary method used for this project. The system clearly defines three roles:

-Customer: Access to menu, ordering, payment, and reservations

-Waiter: Order-taking, billing, and viewing table

-Manager: Menu, inventory, table, and revenue management

Each role has distinct access and interface requirements.

**By Objects**

Key system objects include:

Menu Item

Reservation

Order

Table

Inventory Item

User (Customer, Waiter, Manager)

Each object supports various CRUD operations and interactions depending on user roles.

**By Feature**

Features such as "Make Reservation", "Process Order", and "Generate Bill" are defined and broken down into functional requirements and use cases for clarity.

**By Stimulus**

Certain system functions respond to direct user actions or external events:

A user clicking "Confirm Order" triggers inventory adjustment and order confirmation

A table becoming available after payment is processed

A manager updating a menu item prompts an update in the live menu feed

**By Response**

Functional areas are also structured around expected outcomes, like:

Viewing a reservation list

Receiving a payment confirmation

Generating a bill receipt

**By Functional Hierarchy**

Functions are grouped under logical modules such as:

User Authentication

Order Management

Reservation Handling

Menu Administration

Inventory Control

Reporting and Revenue Tracking

These groupings align with both user roles and system architecture.

**Additional Comments**

For this project, a combination of User Class, Feature, and Functional Hierarchy was used to organize the requirements effectively. This ensures clarity for developers, testers, and stakeholders, and supports scalability for future extensions (e.g., mobile app, kitchen display system, loyalty programs).