ChatGPT

System Requirements Specification (SRS) for Cloud-Based Restaurant POS System

1. Introduction

The purpose of this document is to specify requirements for a modern, cloud-based restaurant Point-of-Sale (POS) system. This SaaS application will support multiple restaurants (multi-tenant), providing a unified interface for billing, order management, inventory, and analytics. The system uses a JavaScript-based tech stack: Frontend built with React (bundled via Vite) and styled with Tailwind CSS for responsive UI; Backend built with Node.js and Prisma ORM for database access. Major stakeholders include restaurant owners/managers, cashiers, waitstaff, and kitchen staff. The POS must be accessible on desktop, tablet, and mobile browsers and support key features such as role-based authentication, offline operation, SMS integration for digital bills, and integrations with online food-ordering platforms.

2. Overall Description

- **Product Perspective:** This POS is a cloud-native, multi-tenant SaaS platform. Multiple restaurants (tenants) share one software instance with isolated data [1]. Each restaurant can have multiple outlets ("outlets" or "branches") with consolidated management. The system offers modular features for billing, orders, inventory, reporting, CRM, etc., exposed via web interfaces and APIs.

- **User Classes and Roles:** Four primary user roles are supported: **Administrator**, **Cashier**, **Waiter**, and **Kitchen Staff**. Administrators manage outlets, users, menus, and settings. Cashiers handle billing, payments, and day-end reporting. Waiters take orders (dine-in/takeaway/delivery) and send KOTs (Kitchen Order Tickets) to the kitchen. Kitchen staff receive KOTs, update order status (e.g. "in progress" or "done"). Each role has distinct privileges (see Table 1) controlled by role-based access.

- **Operating Environment:** The POS is a web application optimized for modern browsers on desktop and mobile platforms. It must be mobile-responsive or available as a Progressive Web App (PWA) to function on smartphones/tablets. An internet connection is required, but critical operations (order-taking, billing) can function offline and sync later (see Sections 3.4 and 4.2). The backend runs on a cloud server or container environment, using Node.js and a relational database (accessed via Prisma).

- **Assumptions and Dependencies:** We assume availability of APIs or gateways for SMS delivery (e.g. Twilio) and for integrating with food delivery aggregators (Zomato, Swiggy, etc.). Payment gateway APIs will be used for card/digital payments. The system must comply with data protection (e.g., GDPR) and payment security (PCI DSS).

3. Functional Requirements

3.1 Authentication and User Management

- **Login/Authentication:** Users must log in via email/password, OTP (sent by SMS), or social login (e.g. Google/Facebook OAuth). The system should support two-factor authentication (2FA) via SMS. SMS-based verification can be implemented using SMS APIs (e.g. Twilio) to send OTP codes [2].
- **Role-Based Access Control:** Each user account is assigned a role (admin, cashier, waiter, kitchen) that determines accessible features. The system provides an **Admin Dashboard** where administrators can create/manage user accounts and roles. Privileges include (see Table 1): creating orders, printing bills/KOTs, editing menu/inventory, viewing reports, and managing outlets. Role-based login ensures, for example, kitchen staff only see pending KOTs and not payment screens [3].

Role	Order Entry/ Billing	KOT Management	Menu/ Inventory	Reports/ Analytics	Admin Controls
Admin	Full access	Full access	Full access	Full access	Full access
Cashier	Create/close bills, refunds	Generate/print KOTs	View/Edit items	View (sales reports)	View only
Waiter	Take orders (dine-in/ takeaway)	Send KOTs (print)	View only	Basic view (daily)	No
Kitchen Staff	Update order status only	Update/mark KOT done	No access	No access	No

Table 1: User roles and their primary permissions.

3.2 Billing and Order Management

- **Order Types:** Support for multiple order channels – dine-in, takeaway, delivery, and counter sales. The UI should allow quick switching between these (e.g. tabs labeled “Dine-in”, “Delivery”, “Takeaway”) as shown below. Staff can assign orders to tables or new tickets. Menu items can be quickly selected, quantities adjusted, and special instructions added.

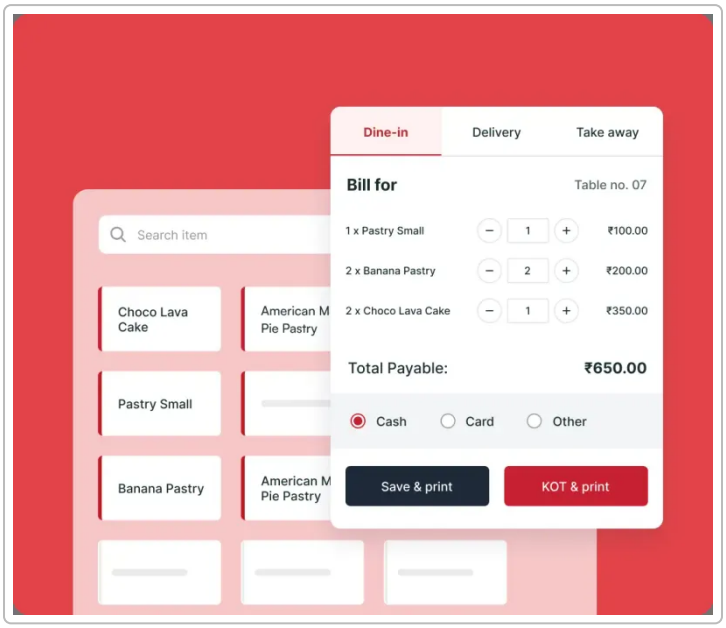

Figure: Billing interface with order-type tabs (Dine-in, Delivery, Takeaway). The UI is touch-optimized and keyboard-friendly, allowing staff to rapidly select items, adjust quantities, and generate bills. The billing module must provide quick "3-click billing" workflows: select items, apply modifiers/discounts, and print the invoice [4]. It should support splitting or merging bills (e.g. split by guest or combine tables) and applying coupons or discounts at checkout [5]. All common payment methods (cash, card, UPI/wallets) should be processed, and receipts printed or sent digitally.

- **KOT (Kitchen Order Ticket) Generation:** When orders are placed, the system generates KOTs for the kitchen. KOTs should be sent automatically to kitchen printers or displays in the appropriate section (e.g. Bar, Grill). The system may support *group KOT printing* (split by preparation area) or remote KOTs for different kitchens [6]. Waiters can generate or reprint KOTs as needed. Kitchen staff can mark items as "prepared" or "ready", updating the system in real-time. This digital KOT workflow reduces errors and speeds up service [7] [6].

- **Offline Billing Support:** The POS must work offline when the internet is unavailable. In offline mode, it should store orders, payments, and inventory changes locally (e.g., in browser storage or a local database). Once connectivity is restored, all pending transactions and data must sync back to the cloud server. According to industry best practices, offline POS modes create a local copy of inventory, customer, and transaction data so businesses can continue processing sales during outages [8]. The system should handle sync conflicts gracefully (e.g., using timestamps or retries) and alert the admin if any data failed to sync.

3.3 Menu and Inventory Management

- **Menu Configuration:** Administrators (or managers) can create and organize menu items into categories. Each item can have variations (size, flavor) and optional add-ons [9]. The menu can be customized by outlet or time-based (e.g. breakfast vs dinner menu). Area-wise or section-wise pricing is supported, allowing different prices per dining area or branch [10] [9]. For example, an AC seating area may have a higher price than a non-AC area, managed via the menu settings.

- **Recipe and Inventory:** Each menu item can be linked to a recipe of raw ingredients. The inventory module tracks raw material stock levels and automatically deducts quantities as items are sold [11]. For example, selling one "Pasta" reduces the stock of pasta noodles, sauce, etc. The

system allows manual inventory entries (purchases, wastage) and supports recipe-based inventory deduction and batch updates [12]. It should generate low-stock alerts and provide day-end stock and wastage reports for inventory control. Bulk actions (import stocks, close stock) should be supported for ease of management (see Fig. 2).

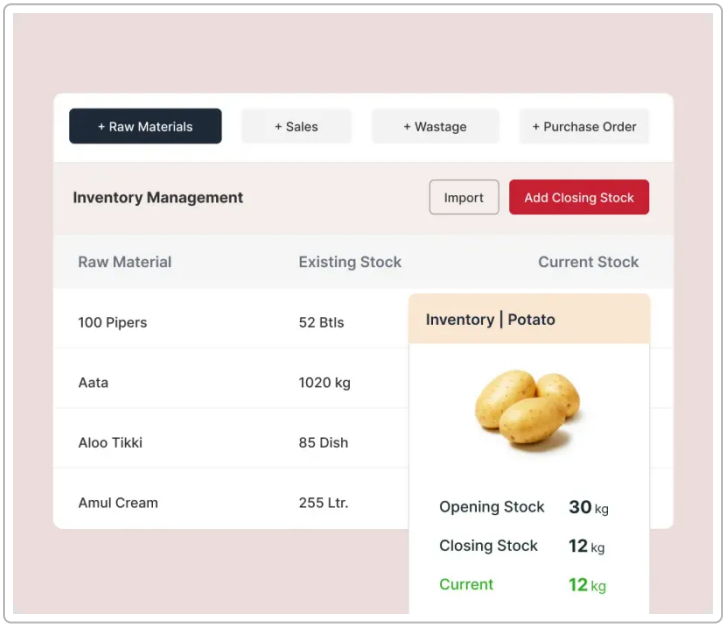

Figure: Inventory management screen showing raw materials, opening/closing stock, and stock adjustments. The inventory dashboard lets managers add new raw materials, record purchases, waste, and transfers. A dedicated "Inventory Report" can display opening vs current stock for each ingredient, highlighting any variances.

- **Costing and Recipe Management:** The system should allow defining standard costs for ingredients and recipes. Managers can update ingredient prices, and the system will automatically recalculate menu item costs and profit margins. Recipe management ensures consistency (e.g. fixed ratio of ingredients) and supports scaling recipes by servings.

3.4 Table and Floor Management

- **Seating Layout:** For dine-in orders, the POS provides a visual floor plan of the restaurant with tables and zones. Each table shows its status (free, occupied, bill pending). Waiters can tap a table to start a new order or view an existing one. The interface supports multiple floors or sections, reflecting area-wise pricing differences.

- **Live Status Updates:** The system shows live updates of table occupancy and orders. When a table's order is sent to the kitchen, the table's icon is marked accordingly. After checkout, the table is cleared automatically. Table bookings/reservations (optional) may be integrated to block tables at scheduled times.

- **QR/Contactless Ordering (Optional):** Tables can have QR codes linking to a digital menu to complement POS operations. While not mandatory, this feature can be considered for future development.

3.5 Online Ordering Integration

- **Aggregator Integration:** The POS integrates with popular online food delivery platforms (e.g. Zomato, Swiggy, Talabat). Orders from these services must flow into a unified "Online Orders Dashboard" within the POS. Staff can accept or reject each order, view details (items, customer info), and dispatch a KOT to the kitchen. Integration should be done via APIs or webhooks (the goal is that the staff need not juggle multiple apps). LimeTray notes that modern POS solutions integrate orders from multiple platforms into one system to manage them in real-time [13].

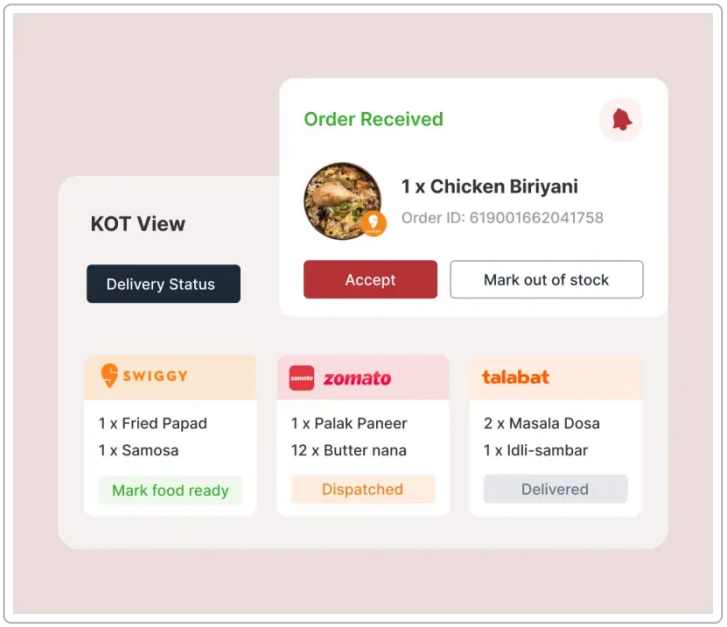

Figure: Online orders dashboard showing orders from Swiggy, Zomato, etc. The dashboard lists incoming orders with their source and status. Staff click "Accept" to send KOTs to the kitchen or "Mark Out of Stock" if items are unavailable. Once prepared, orders are marked "Ready" or "Dispatched." Integration may also support printing a consolidated receipt for delivery partners, if required.

- **Online Menu Sync:** Menus and item availability should sync between the POS and online platforms. Changes in menu (price, availability) should reflect on delivery channels. Stock availability management can optionally disable items online if inventory is low (to prevent over-selling).

- **Self-Delivery and Third-Party:** In addition to third-party apps, the POS should handle the restaurant's own delivery orders (online website or call-in orders) under a "Delivery" order type, ensuring all delivery flows are centralized.

3.6 Reporting and Analytics

- **Sales Reporting:** The system must produce comprehensive sales reports: daily/weekly/monthly sales by outlet, by order type (dine-in/delivery/takeaway), by menu item, and by cashier. For example, reports can show "Time-slot wise sales bifurcation" and breakdown by order category (as in Fig. 3). Petpooja highlights that POS systems offer error-free, automated day-end sales and order reports [14]. Reports should support filtering by date, outlet, or category.

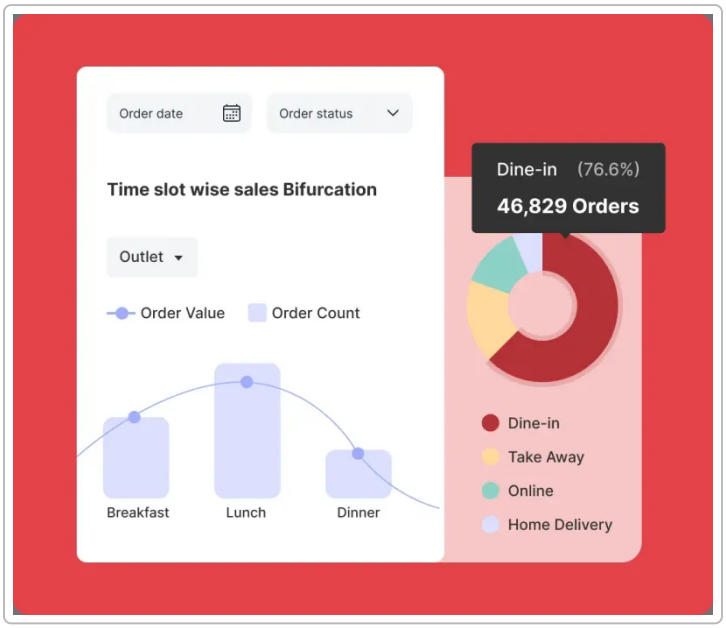

Figure: Real-time analytics dashboard (example). Charts show sales by time-slot and order-type distribution.
In addition to end-of-day reports, the POS provides a real-time dashboard for managers, displaying key metrics (total sales, order counts, avg. ticket) and charts (e.g. pie chart of dine-in vs takeaway). The dashboard must update as orders are processed, enabling managers to react quickly during service.

- **Inventory Reports:** Generate stock usage and wastage reports. At day-end, the system should compare expected vs actual inventory (based on sales vs actual counts) to flag discrepancies. Automated reports (80+ types) are useful [14], including stock valuation, stock-out alerts, item-wise consumption, and supplier purchase summaries.

- **Cash & Payment Reports:** Track all payment types (cash, card, wallet) with a unified payment report. Reconciliation reports should match sales against payment gateway settlements. The system should allow exporting transaction lists for accounting.

- **Multi-Outlet Aggregation:** For restaurants with multiple outlets, the system consolidates data across outlets. Admins can view group-level reports (total revenue, best-selling items chain-wide, top-performing outlets). LimeTray observes that a unified dashboard can combine data from all outlets for real-time insights [15].

- **Customer Insights (CRM Analytics):** Collect customer data (e.g. phone numbers, visit frequency, total spend). The POS can generate analytics on customer behavior (e.g. repeat rate, favorite dishes). These insights support targeted marketing. Modern POS systems embed CRM to gather customer preferences and spending habits for personalized loyalty programs [16]. For example, if an email/phone is captured, the system can track loyalty points or send promotions via SMS/email.

3.7 CRM and Loyalty

- **Customer Profiles:** Ability to save customer profiles (name, contact, preferences). At checkout, cashiers can search or add customers for loyalty or invoicing. A history of past orders should be retrievable.

- **Loyalty Program:** Optionally, implement a point-based loyalty program or membership tiers. Points can accrue per bill and be redeemed for discounts. The POS should allow rule-based point calculation (e.g. 1 point per ₹100 spent).

- **Marketing & Communication:** Integrated SMS/email campaign tools (or integrations with marketing platforms) for promotions. Using collected customer data, the system can send birthday offers or mass promotions. The POS's CRM can interface with SMS gateways to automate this (e.g. sending an SMS coupon).

3.8 Pricing, Taxation, and Expense Tracking

- **Area/Section Pricing:** As noted, different zones or seating areas can have different base prices [10]. The POS menu setup allows price variants by area. Tax settings (GST, VAT, Service Charge) can be configured per region. It supports multi-tax on items (e.g. CGST+SGST) and inclusive/exclusive pricing.

- **Expense Management:** Administrators can record overhead expenses (rent, utilities, salaries, raw material purchases, etc.). The system should link these expenses to outlets and show daily/weekly expense reports. Profit & Loss statements are generated by offsetting sales against expenses.

3.9 Administration and Multi-Outlet Control

- **Outlet Management:** As a SaaS multi-tenant system, each restaurant (tenant) can operate multiple outlets. The admin portal allows adding outlets with settings (shift timings, taxes, printers). Menus and inventory can be synced across outlets or customized per outlet.

- **Global Admin Controls:** A master admin (for multi-brand or chain) can manage all outlets under one account, with a global dashboard for summary statistics. This includes aggregate sales, outlet comparisons, and centralized user management. For example, head office can view combined sales graphs across cities.

- **Integrations:** Administrators can configure integrations: payment gateway credentials, SMS gateway API keys, and online ordering platforms. For SMS integration, the system will use a third-party SMS API (e.g. Twilio) to send transactional messages [2]. For example, after billing, an SMS invoice (e-bill) with the bill amount and a QR code can be sent automatically [17].

4. Non-Functional Requirements

- **Usability:** The UI must be clean, intuitive, and fast-loading. It should work well with touch (larger buttons) and keyboard shortcuts (for fast entry by cashiers). Consistent styling (via Tailwind CSS) ensures cross-browser compatibility and mobile responsiveness [18]. Training should be minimal due to familiar UI patterns.

- **Performance:** The POS should handle high transaction volumes (e.g. 100 orders/hour or more) with minimal latency. Search/filter on menu and customer database should be instantaneous. Backend APIs must respond within a few hundred milliseconds under normal load.

- **Reliability/Availability:** The system should be highly available (target 99.9% uptime). Automatic failover and backups are required in the cloud architecture. Offline mode ensures local operation

during network outages, as discussed in Section 3.2 [8]. Data sync mechanisms ensure no loss of orders.

- **Scalability:** Designed for SaaS, the system scales horizontally to support many restaurants and users. Multi-tenant architecture shares resources efficiently [1], and additional compute/database nodes can be added for load.

- **Security:** Role-based authentication with secure password hashing (e.g. bcrypt) and encryption of sensitive data (e.g. SSL for network). Compliance with PCI DSS for payment data. Audit trails must record user actions (login, voids, overrides). OTP login and session management prevent unauthorized access.

- **Maintainability:** Code should be modular (React components, Express routes, Prisma models). APIs should have documentation. Automated testing (unit, integration) is expected. Logging and monitoring (e.g. error reporting, usage metrics) should be in place.

- **Localization:** Support for multiple currencies and languages (English/Hindi, etc.) depending on region. Currency and number formats should adapt based on outlet locale.

- **Compliance:** If handling GST (India) or VAT (other regions), include features for tax invoice generation and relevant reports (GSTR-1, GSTR-3B as needed) [19]. Data privacy (GDPR) requires ability to delete/modify personal customer data on request.

5. Modules Summary

The following table summarizes the major system modules, their responsibilities, and key sub-features:

Module	Description
Authentication & Security	User login (password, OTP, social). Role-based access control. OTP via SMS APIs (e.g. Twilio) for 2FA. Password reset functionality.
POS Terminal (Billing/Orders)	Order entry UI (dine-in/takeaway/delivery). Quick billing, table assignment, split bills, discounts. Integration with KOT module. Offline data capture with sync.
KOT Management	Automatic KOT generation for kitchen. Configurable printer routing and group KOTs [6]. Kitchen order queue, status updates (Cooking → Ready) by staff.
Menu & Inventory	CRUD menu items (with variations, addons) and categories. Area-wise pricing [10]. Inventory of raw materials with stock tracking, auto-deduction on sales [11], wastage entry, purchase orders. Recipe management linking ingredients.
Table Management	Visual table map, section zoning. Real-time table status updates. Reservation slot blocking (optional).
Online Ordering Integration	Sync menus with aggregators. Unified order dashboard for Swiggy/Zomato/Foodpanda/etc [13]. Accept/reject orders, print delivery KOTs. Consolidated revenue tracking from third-party.

Module	Description
Reporting & Analytics	Sales reports by time, outlet, item. Inventory and wastage reports. Cashier shift reports. Real-time dashboard charts (see Fig. 3). Multi-outlet summary. 80+ report types [14] [15].
Customer Management (CRM)	Database of customers with visit history. Loyalty points and membership. Automated SMS/e-billing. Data analytics on customer behavior [16].
Finance & Payments	Payment gateway integration (card, UPI). Cash drawer management. Invoicing (digital receipts via SMS/email [2]). Expense entry and P&L. Unified payment reconciliation across outlets.
Administration	Outlet/user/configuration management. Role and permission settings [3]. Subscription and billing for SaaS plan. API keys for SMS, payments, third-party integrations. Activity logs.

The system's modular design ensures each component can be developed and tested independently. APIs connect frontend and backend, and a multi-tenant database scheme isolates each restaurant's data.

6. References

- Petpooja Restaurant POS features: quick billing, inventory auto-deduction, real-time reports [4] [11] [14].
- SaaS Multitenancy: "multiple customers or tenants can share a single software instance running on a cloud infrastructure" [1].
- Offline POS mode: local backup of inventory and transactions during network outages [8].
- SMS API (Twilio): "Send and receive text messages with just a few lines of code" enabling transactional SMS [2].
- Digital KOT & Integrations: unified online ordering, multi-outlet analytics, and CRM insights [13] [15] [16].
- Additional POS features (area pricing, KOT, e-bill on SMS) from industry sources [9] [6] [17] [10] [16].

---

[1] **How to Build & Scale a Multi-Tenant SaaS Application ☑ Best Practices**
https://acropolium.com/blog/build-scale-a-multi-tenant-saas/

[2] **SMS API for business text messaging | Twilio**
https://www.twilio.com/en-us/messaging/channels/sms

[3] [6] [9] [12] [17] [19] **Pricing & Features of Restaurant POS Software**
https://www.ere4u.in/feature&pricing.php

[4] [5] [11] [14] **Restaurant POS and Management System | Petpooja**
https://www.petpooja.com/poss

[7] [13] [15] **Importance of Kitchen Order Ticket (KOT) in the Restaurant Industry – Limetray**
https://limetray.com/blog/what-is-kot-restaurant-industry/

[8] **POS Offline Mode: How To Run Your Business Without Connection**
https://www.magestore.com/blog/pos-offline-mode/

10 **Bar POS System with Inventory, Billing - Gofrugal**
https://www.gofrugal.com/restaurant/bar-pos/

16 **How does a POS system manage customer loyalty programs?**
https://www.tryotter.com/resource/wiki/how-pos-systems-manage-customer-loyalty-programs

18 **Responsive design - Core concepts - Tailwind CSS**
https://tailwindcss.com/docs/responsive-design