Ruchikooth Admin Portal: Complete User Manual

Welcome to the **Ruchikooth Admin Portal** — your complete command centre for managing the online store, processing orders, handling customer support, and analysing business performance. This manual covers every screen, feature, and workflow available to administrators.

📑 Table of Contents

1. Accessing the Admin Portal
2. Dashboard Overview
3. Menu Management
4. Product Management
5. Order Management
6. WhatsApp Ordering
7. Party & Catering Orders
8. Split Orders (Pre-booking + Normal)
9. Categories
10. Store Settings
11. Customer Support
12. Customer Management
13. Analytics & Reports
14. Delivery Management
15. Staff Management
16. Back Office
17. Notifications System
18. System Architecture Overview
19. Security Architecture
20. Deployment & Updates
21. Troubleshooting & Known Issues
22. Frequently Asked Questions

---

1. Getting Started & Dashboard

Logging In

1. Navigate to the Admin Portal URL:
   - **Production:** `https://ruchikooth.com/admin/`
   - **Local Development:** `http://localhost:8081/`
2. You will see the **Admin Login** screen.
3. Enter your **Email** and **Password**.
4. Click **Login**.

Development credentials (if applicable):

- URL: `http://localhost:8081/`
- Username: `admin@ruchikooth.com`
- Password: `Test@1234`

> *[!IMPORTANT] Only user accounts with the `admin` role in Firestore can access the admin portal. If you see an "You do not have admin access" message, contact your system administrator.*

Dashboard Layout

Once logged in, the **Dashboard** is your command centre. It is organised as follows:

- **Top Bar:**

    - **Logged-in email** displayed on the left
    - **Notification Bell** 🔔 with unread count badge (red circle)
    - **Logout** button on the right
- **Tab Navigation:** Four main tabs — **Menus**, **Orders**, **Categories**, **Support**.

What You Can See on the Dashboard

The dashboard is rendered via `admin-dashboard.js` and displays four metric cards:

Metric	Description
Total Orders	Count of all orders ever placed in the system
Total Revenue	Sum of all order totals (includes all statuses)
Active Products	Number of products with `status = active`
Total Customers	Count of registered users who are not admins

Numbers animate smoothly from 0 to their final value on page load (1-second animation via `requestAnimationFrame`).

Inventory Alerts

The dashboard checks all products and displays **Low Stock Alerts** for any item with stock ≤ 5 units:

- Item name and remaining stock quantity shown
- "Update Stock" button (links to the Products section)
- Alerts displayed in a red-bordered card with exclamation-triangle icon
- If no items are low on stock, the alerts section is hidden

Recent Orders Table

Shows the **last 10 orders** placed, sorted by date descending. Each row includes:

Column	Details
Order ID	Truncated ID (last 8 characters in uppercase, e.g., `#ABC123XY`)
Customer	Customer email
Location	Delivery address / location
Amount	₹ formatted total
Status	Colour-coded badge (Pending=Yellow, Confirmed=Green, etc.)
Date	Order creation date
Actions	View button + inline status dropdown for quick updates

The status dropdown allows changing order status directly from the dashboard without opening the order details modal.

> *[!TIP] Keep an eye on the **Top Right notification bell**. If a new order comes in or a support ticket receives a reply, a small alert will pop up so you never miss a sale or a customer inquiry!*

---

2. Dashboard Overview

The dashboard is rendered into the `#admin-content` div by `loadDashboardPage()` in `admin-dashboard.js`.

Architecture Note

The dashboard is the first page loaded after login. It calls `adminDashboard.loadDashboardData()` which:

1. Fetches all products (to find low stock items with threshold ≤ 5)
2. Counts all orders (for total orders + revenue)
3. Counts active products (status = `active`)
4. Counts non-admin users (for customer count)
5. Loads last 10 orders
6. Renders everything into the DOM

Dashboard Stats Grid

```
<div class="admin-stats-grid">
    <div class="admin-stat-card">
        <div class="admin-stat-value" id="total-orders">0</div>
        <div class="admin-stat-label">Total Orders</div>
    </div>
    <!-- ... three more cards ... -->
</div>
```

---

3. Menu Management

The **Menus** tab (`admin-menu.js`) is where you manage all items visible on the customer storefront. This is a full-featured module with real-time updates via Firestore `onSnapshot()` listeners.

Accessing Menu Management

Click the **Menus** tab at the top of the dashboard.

Statistics Bar

Four stat cards at the top give you an instant overview:

- **Total Items** — All menu items in the database
- **Available Items** — Items marked as available
- **Unavailable Items** — Items marked as unavailable
- **Categories** — Number of unique categories used

Menu Items Grid

Items are displayed as a responsive card grid (auto-fill, min 350px width). Each card shows:

- **Item Image** — Lazy-loaded with fallback to `kerala-meals.jpg`; loading state includes CSS classes for smooth transitions
- **Item Name** — Bold heading
- **Availability Toggle** — Switch ON/OFF (green/red slider that saves immediately)
- **Category Badge** — Coloured tag
- **Detailed Info Grid:**
  - Track Stock — Yes/No with current stock count (green if >0, red if 0)
  - Pre-Booking — Yes/No indicator
  - Next Available — Shows when item will be back (if currently unavailable)
  - Bulk Threshold — Minimum qty for bulk pricing
  - Bulk Price — Discounted unit price for bulk orders
  - Best Before — Expiry date/time (red if expired, with label)
- **Description** — Clamped to 2 lines using CSS `-webkit-line-clamp: 2`
- **Price** — Large ₹ amount
- **Action Buttons:**

- **View in Store** — Opens item detail page in customer store
- **Edit** — Opens modal to edit item details
- **Delete** — Removes item after confirmation

Search & Filters

Three filter controls at the top:

1. **Search Box** — Real-time search by item name or description (debounced 300ms)
2. **Category Dropdown** — Filter by category (populated from Firestore `categories` collection; auto-seeds Breakfast, Lunch, Dinner, Beverages, Desserts if empty)
3. **Availability Filter** — All / Available / Unavailable

Pagination

- Items are paginated (20 per page)
- Scroll-based lazy loading: when user scrolls near the bottom of the container, the next page loads automatically
- Page info shows "Showing X-Y of Z items"
- Previous/Next buttons with current page indicator

Adding a New Menu Item

1. Click the **"Add New Item"** button in the card header.
2. A modal form appears with the following sections:

Basic Information

- **Item Name** (required) — The display name on the storefront
- **Description** — Detailed item description (textarea, rows=3)
- **Price (₹)** (required) — Selling price
- **Category** (required) — Populated from existing categories
- **Menu Item Image** — Two ways to upload:
  - Upload a file from your computer (auto-compresses to ≤900KB JPEG)
  - Enter an image URL directly
  - Live preview updates as you choose/type

Availability & Scheduling

- **Next Available** — Text field (e.g., "Tomorrow, 2 PM") shown when item is unavailable
- **Available for Order** — Master toggle
- **Today's Special** — Highlights the item as a daily special
- **Available for Delivery / Pickup / Dine-in** — Three separate checkboxes for channel-specific availability

Bulk Order Pricing

- **Min Qty for Bulk** — Minimum quantity to qualify for bulk pricing
- **Bulk Unit Price (₹)** — Discounted price per unit when bulk threshold is met
- Displayed in a bordered section with green background hint

Inventory Management

- **Track Stock** — Enable stock counting (disables ordering when stock hits 0)
- **Current Stock Count** — Enter current inventory level
- **Allow Pre-booking** — Let customers order even when stock is 0 (for future delivery)
- **Best Before** — Datetime picker for freshness expiry; system warns customers if expired

Dietary Type

- **Veg** — Green leaf icon (default)
- **Non-Veg** — Red drumstick icon

Custom Variations (Optional)

Enable **"Has Custom Variations"** to configure product customisation:

- **Variant Options** — Enter one option per line in `Name:Price` format (e.g., `0.5kg:500`, `1kg:900`, `Small:100`). The price is the **additional cost** over the base price.
- **Add-on 1 (e.g., Flavor)** — Name + comma-separated options + required toggle
- **Add-on 2 (e.g., Frosting)** — Name + comma-separated options + required toggle
- **Requires Personalized Text** — E.g., for cake inscriptions or gift messages

> *[!TIP] Use variations for items like cakes (different sizes), pickles (different weights), or any product with customisable options.*

Saving

- Click the **"Save Item"** button at the bottom of the modal.
- The grid updates in real time via Firestore snapshot listener.

Editing an Item

- Click the **Edit (pencil)** icon on any card.
- The same modal opens pre-populated with all existing values.
- Make changes and click **"Update Item"**.

Deleting an Item

- Click the **Trash** icon on any card.
- Confirm the deletion in the popup dialog.
- The item is removed from Firestore immediately.

Toggling Availability

- Use the **green/red slider** on any card to instantly enable or disable an item.
- Changes are saved immediately without reloading the page.

---

4. Product Management

The **Products** section (`admin-products.js`) is an alternative product management interface with a table-based layout and additional features like branch-level unavailability.

> *[!NOTE] Products and Menu Items are kept in sync — creating/updating/deleting in one section automatically updates the other (via `products` and `menu_items` collections using the same document ID).*

Loading the Products Page

Products is loaded via the sidebar navigation (click a "Products" link if available) or by calling `loadProductsPage()`.

Products Table

Displays all products with columns:

Column	Details
Image	50×50px thumbnail with rounded corners
Name	Bold item name
Category	Category name (or N/A)
Price	₹ formatted
Stock	Current stock quantity
Status	Active / Inactive / Out of Stock (colour-coded badges)
Featured	Yes/No badge

Actions	Edit + Delete buttons

Filters

- **Category Dropdown** — Filter by category
- **Status Dropdown** — All / Active / Inactive / Out of Stock
- **Search Box** — Real-time search by name or description (debounced 500ms)

Pagination

- Products are paginated (10 per page)
- Previous / Next buttons with page number buttons

Adding a Product

Click **"Add New Product"** to open a modal with:

- **Product Name** (required)
- **Category** (required) — Plus "Add New Category" option (shows a new text field when selected)
- **Description**
- **Price** and **Stock Quantity**
- **Product Image** — File upload (auto-compresses) or URL input with live preview
- **Status** — Active / Inactive / Out of Stock
- **Featured Product** — Checkbox
- **Branch Unavailability** (if branches exist) — Select which branches should NOT offer this product (shown as a grid of checkboxes)

Image Upload Modes

Two modes are supported, configured in Store Settings:

1. **Firebase Storage** — Uploads go to Firebase Cloud Storage (requires Blaze plan; recommended for production)
2. **Inline Data URL** — Images are compressed (max 900KB at JPEG quality 0.35-0.82) and stored directly in the document (no extra storage costs; suitable for testing)

Syncing with Menu Items

When a product is saved, the system also attempts to sync the data to the `menu_items` collection:

- **Create:** `db.collection('menu_items').doc(docRef.id).set(productData)`
- **Update:** `db.collection('menu_items').doc(form.dataset.productId).update(productData)`
- **Delete:** `db.collection('menu_items').doc(productId).delete()`
- Failures in syncing are logged as warnings but do not block the operation

Editing a Product

- Click **Edit** to open the pre-populated modal
- Make changes and click **Update Product**

Deleting a Product

- Click **Delete** and confirm
- Removes from both `products` and `menu_items` collections

---

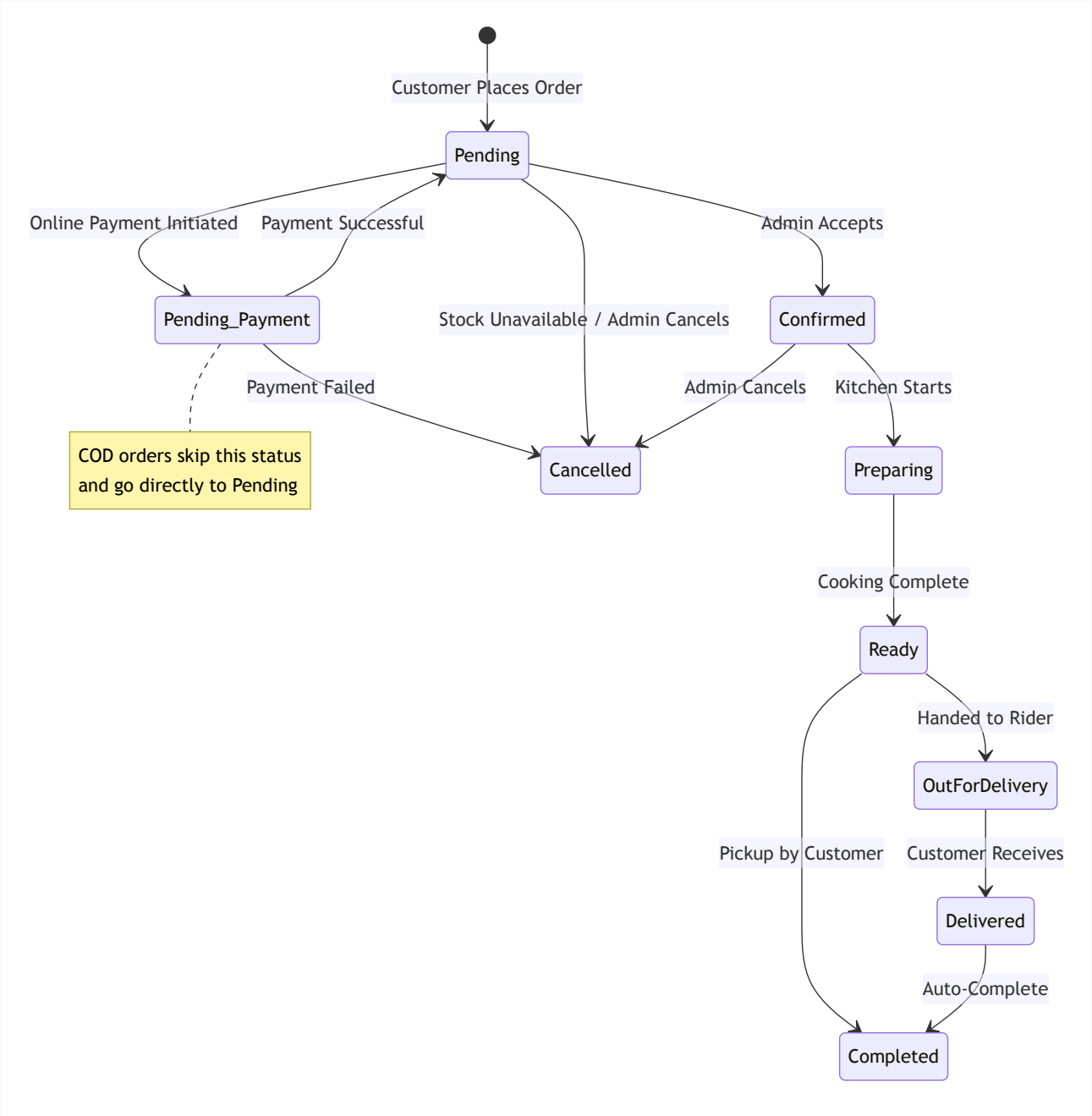
5. Order Management & Processing

The **Orders** tab (`admin-orders.js` and `admin.js`) is the most feature-rich section. It provides a complete lifecycle management system for every customer purchase.

Accessing Orders

Click the **Orders** tab in the dashboard navigation, or if using the enhanced sidebar, click **Orders**.

The Order Lifecycle



Order List View

The enhanced order management (loaded via `adminOrders.loadOrdersPage()`) provides:

Summary Stats Bar

Compact pills showing key metrics:

- **Total orders** count with ₹ total revenue (filter-aware)
- **Pending** orders count (gold background)
- **Paid** orders count (green)
- **COD** orders count (purple)
- **Cancelled** orders count (red)

Filters

Filter	Options
Search	Search by Order ID, customer email, name, or phone (debounced 500ms)
Date	All Dates / Today / This Week / This Month
Payment	All / Paid / Pending / COD
Status	All / Pending / Confirmed / Preparing / Ready / Out for Delivery / Delivered / Cancelled
Delivery Agent	All / Unassigned / [Individual Agents]
My Deliveries Only	Toggle for delivery-only staff (checkbox)

All filtering is performed client-side after fetching all orders sorted by `createdAt desc`. This avoids the need for composite Firestore indexes.

Pagination

- Orders are paginated (10 per page)
- Page navigation with prev/next, page numbers, and "Showing X-Y of Z" summary
- Ellipsis (...) for large page ranges

Order Table Columns

Column	Details
Order ID	Truncated ID (last 8 chars) in bold
Customer	Customer name
Total	₹ formatted in bold
Status	Colour-coded pill badge with dot indicator
Payment	Payment method + status badge + action buttons
Date	Order creation date
Actions	View button + Status update dropdown

Payment status badges:

- `paid` → Green "Paid" badge
- `refund_pending` → Yellow "Refund Pending" badge
- `refunded` → Blue "Refunded" badge
- Default → Red "Pending" badge

Viewing Order Details

Click the **View** button (eye icon) on any order row to open a comprehensive modal with these sections:

1. Order Information

- Order ID (truncated)
- Current Status (with editable dropdown)
- Order Date & Time
- Payment Method

2. Customer Information

- Name, Email, Phone

- Delivery Address (full address with city and PIN code)

3. Order Summary

- Subtotal
- Discount (if any coupon applied)
- Tax (default 5%, configurable in Settings)
- Delivery Fee
- **Total Amount** (highlighted in green, larger font)

4. Order Items

Visually listed items with name, quantity, and line total. Scrollable container (max-height 250px) for large orders. Individual item cards show:

- Item name with quantity
- Line total (price × qty)

5. Kitchen Management (Internal)

- **Assigned Cook** — Dropdown to select from staff with `cook` or `manager` roles; "Assign" button
- **Kitchen Status** — Dropdown with stages: Pending → In Preparation → Garnishing → Packed & Ready
- Kitchen controls are disabled (opacity 0.6, pointer-events none) for:
  - `pending_payment` orders (with yellow info box explaining why)
  - `cancelled` orders
- If payment is pending, admin can click **"Convert to Cash on Delivery"** to bypass

6. Internal Notes

- Free-text textarea for internal staff notes (saved on blur/change)
- Visible to all admin staff with access

7. Delivery Dispatch

- **Delivery Agent** — Dropdown to assign from `delivery` / `manager` / `superadmin` staff; "Dispatch" button
- **Live Tracking URL** — Text field for Google Maps / any tracking link
- Both controls disabled for `kitchen_only` role

8. Payment Actions

- **COD Cash Collection** — Green-bordered panel shown for undelivered COD orders; "Mark Cash Collected" button
- **Refund Processing** — Yellow panel shown when a paid order is cancelled; "Process Refund" button
- **Refund Confirmation** — Blue panel showing refund details (who processed it and when)

9. Support Tickets (if any)

If the order has linked support tickets, a red-bordered section shows:

- Ticket issue type and status
- "View Chat" button to jump to the support conversation

Changing Order Status

There are several ways to update status:

1. **From the order list table** — Use the inline status dropdown in the Actions column
2. **From the order details modal** — Use the status dropdown at the top
3. **From the dashboard recent orders** — Inline dropdown in the dashboard table

Automatic Status Transitions

- Setting a COD order to **Delivered** → auto-marks payment as `paid` with timestamp and admin email logged
- Setting any order to **Delivered** for COD → auto-sets `paymentStatus = 'paid'`, `paymentCollectedAt`, `paymentCollectedBy`
- Cancelling a **paid** order → auto-sets payment to `refund_pending`

- Cancelled orders have the status dropdown disabled in the list view

Editing Order Items

If you have `all` or `orders` privileges, you can modify the items in an order:

1. Open the order details modal
2. Click **"Edit Items"** button (visible unless order is cancelled/delivered)
3. A new modal appears showing:
    - Current items table with:
        - Adjustable quantity input fields
        - Delete (trash) buttons per item
        - Line totals auto-calculated
    - "Add Product" section with:
        - Product dropdown (filtered to exclude already-added items)
        - Quantity input
        - "Add" button
    - Live recalculation of:
        - Subtotal
        - Coupon discount (preserved from original)
        - Tax (5% of discounted subtotal)
        - Delivery fee (preserved from original)
        - **New Total** (highlighted in blue)
4. Click **"Save Changes"** to persist

> *[!IMPORTANT] All edits are logged in an `editHistory` array on the order document, recording:*
>
> - *Who made the change (admin email)*
> - *When the change was made*
> - *Previous total vs. new total*

Exporting Orders

Click the **"Export CSV"** button at the top of the orders page. This downloads a CSV file with 11 columns:

- Order ID (formatted as `#ABCD1234`)
- Customer Email, Phone, Location
- Items Count, Total Amount
- Status, Payment Method, Payment Status
- Can Convert to COD (Yes/No)
- Order Date

The CSV includes all currently filtered orders, not just the current page.

Converting Orders to COD

For orders stuck in `pending_payment` (failed online payments), you can:

1. Click **"Convert to COD"** in the payment column or the modal (blue underlined link)
2. Confirm the conversion in the dialog
3. The order is updated to `cash_on_delivery` payment method and status set back to `pending`

Processing Refunds

When a paid order is cancelled:

1. The payment status auto-changes to `refund_pending`
2. Click **"Process Refund"** button (yellow, with undo icon)
3. Confirm the refund action

4. Payment status updates to `refunded` with timestamp and admin email logged
5. Refund metadata saved: `refundedAt`, `refundedBy`

> *[!NOTE] This is an internal system action; actual money transfer to the customer must be done manually through your payment gateway (Razorpay/direct bank).*

Role-Based Status Access

The system respects user privileges when showing status options:

Role	Allowed Status Transitions
Super Admin / Manager (`privs: all`)	All statuses
Cook / Kitchen Staff (`privs: kitchen_only`)	Preparing, Ready (and current status)
Delivery Staff (`privs: delivery_only`)	Out for Delivery, Delivered (and current status)

Unauthorized status options are hidden from the dropdown entirely to keep the UI clean.

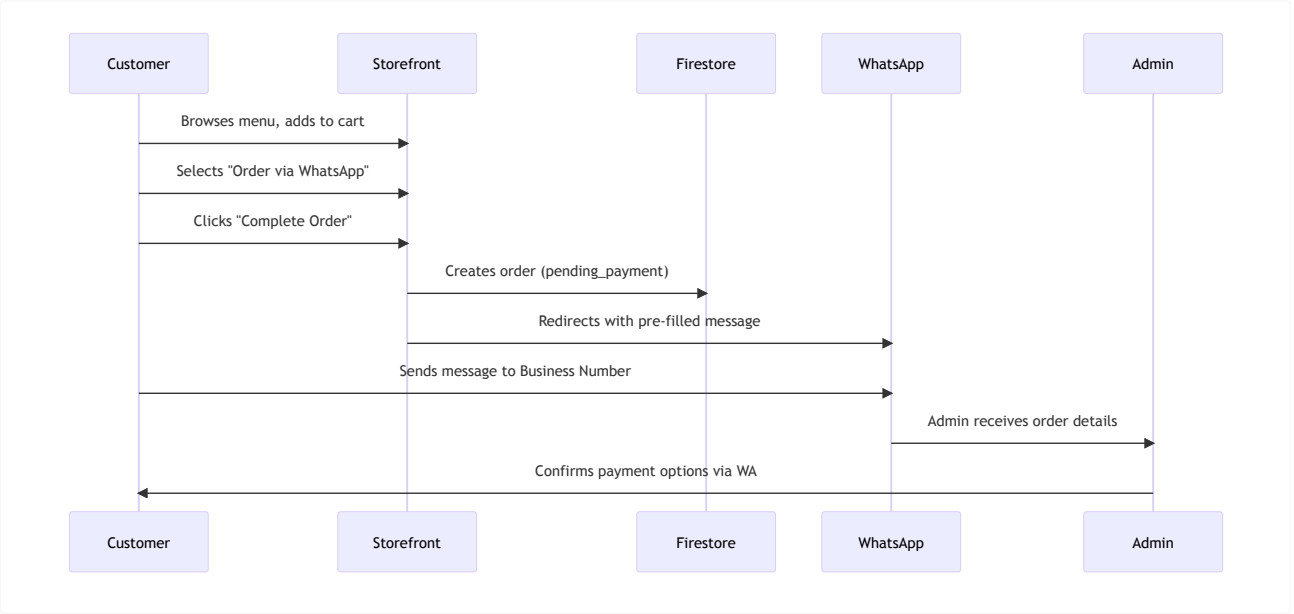
---

6. WhatsApp Ordering

The **WhatsApp Ordering Integration** seamlessly connects the Ruchikooth online storefront with WhatsApp, allowing customers to browse the menu, add items to their cart, and submit their final order directly to the restaurant's business WhatsApp number.

How It Works

1. **Customer on Storefront:** Customer browses the menu, adds items to cart, and proceeds to checkout.
2. **Payment Selection:** On the Payment Step of the order wizard, the customer selects **"Order via WhatsApp"** payment method.
3. **Order Created:** Clicking "Complete Order" securely creates the order in Firestore with:
   - `paymentMethod: "whatsapp"`
   - `status: "pending_payment"` (by default)
4. **WhatsApp Redirect:** The customer is redirected to WhatsApp with a pre-filled message containing:
   - Order ID
   - All ordered items with quantities and prices
   - Subtotal, Delivery Fee, Total Amount
   - Customer name, phone, delivery address
   - Tracking link
   - A request for payment confirmation
5. **Admin Receives Message:** You receive the WhatsApp message instantly on your business number with all order details.



Example Pre-filled WhatsApp Message

```
Hello Ruchikooth! 👋

I would like to place an order.

*Order ID:* #ABC123XY

*Items:*
1x Chicken Biryani - ₹250
2x Masala Dosa - ₹160

*Subtotal:* ₹410
*Delivery Fee:* ₹40
*Total Amount:* ₹450

*Delivery Details:*
John Doe (9876543210)
123 Main Street
Bangalore, 560001

*Track Order:*
https://ruchikooth.com/tracking.html?id=ABC123XY

Please confirm my order and let me know the payment options! Thank you! 🍲
```

Admin Workflow for WhatsApp Orders

1. **Order Appears in Admin:** The order appears in the Orders page with `paymentMethod: "whatsapp"` and `status: "pending_payment"`.
2. **Confirm Payment via WhatsApp:** Reply to the customer on WhatsApp to provide UPI payment links, QR codes, or confirm a Cash on Delivery arrangement.
3. **Update Order Status:**
   - Once payment is confirmed, click **"Convert to Cash on Delivery"** in the order details
   - Or update status from `pending_payment` to `pending` and process normally

Configuring the WhatsApp Number

The WhatsApp number used for receiving orders is taken from the **Store Phone Number** setting:

1. Go to **Settings** → **Store Settings**
2. Update **"Contact Phone"** with the country code (e.g., `919876543210`)
3. The checkout system automatically uses this number for generating `wa.me` redirect links

Troubleshooting WhatsApp Orders

Issue	Solution
Blank WhatsApp message	Customer's browser may be blocking URL redirects; advise them to check browser settings
Order missing in admin	Order only generates if customer clicks "Complete Order"; abandoned checkouts don't create orders
Wrong number receiving messages	Update store phone number in Settings (must include country code without +)

---

7. Party & Catering Orders

Ruchikooth includes a dedicated **Party Orders** feature for catering and bulk event bookings.

Customer Experience

Customers can access the party ordering page at `/party-orders.html` to submit catering requests with custom event details.

Admin Management

Party orders appear as a special tab/section in the admin dashboard. Each party order includes:

- Event type and date
- Estimated guest count
- Menu preferences
- Contact information
- Status tracking

Managing Party Orders

1. Navigate to the Party Orders section in the admin dashboard
2. View incoming catering requests
3. Update status as needed (e.g., Inquiry, Confirmed, Completed, Cancelled)
4. Use the WhatsApp quick-chat feature to contact customers directly

---

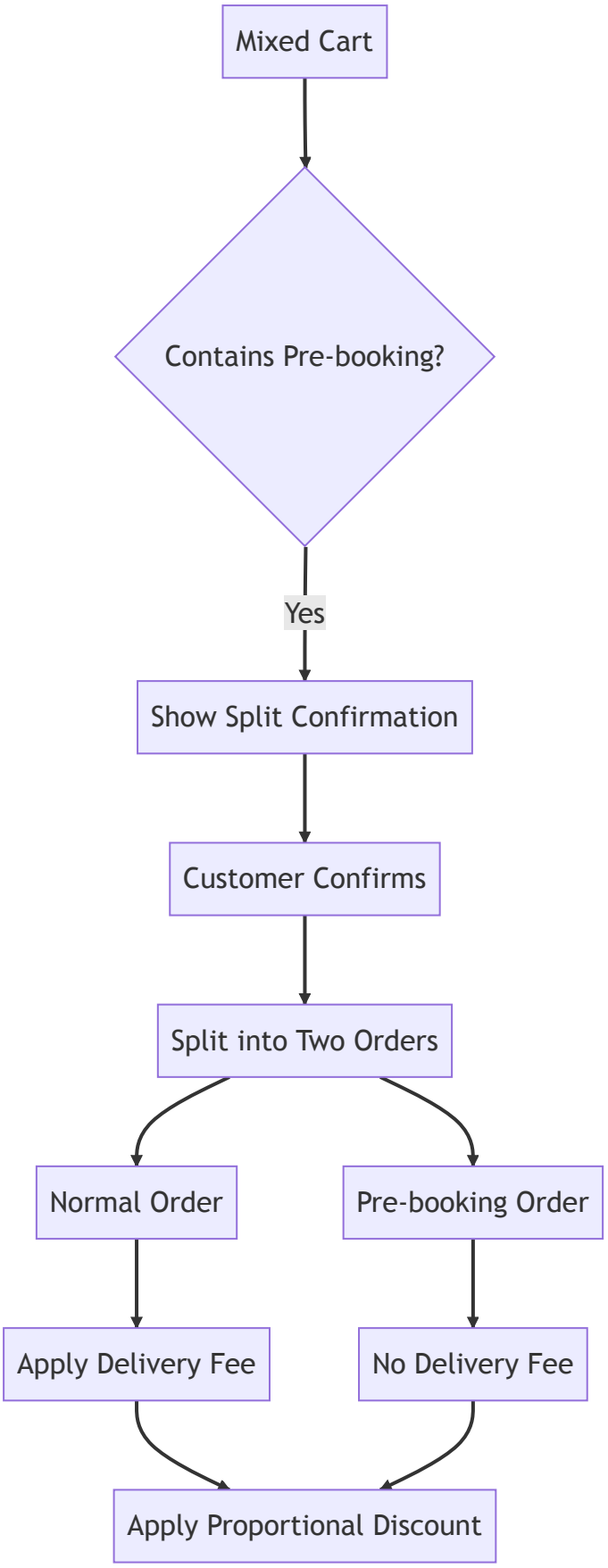
8. Split Orders (Pre-booking + Normal)

When a customer's cart contains a mix of **pre-booking items** (items that need to be scheduled for future delivery) and **normal items** (available for immediate delivery), the system can split the order into two separate orders.

How Split Orders Work

1. **Detection:** During checkout, the system detects items marked for pre-booking.
2. **Customer Confirmation:** A branded modal (styled to match Ruchikooth aesthetics) shows:
    - Normal order items (for immediate delivery)
    - Pre-booking order items (scheduled for future date)
3. **Order Creation:** Two separate Firestore order documents are created:
    - One for normal items (immediate delivery)

    - One for pre-booking items (scheduled delivery)
4. **Delivery Fee:** Charged only on the immediate delivery order
5. **Coupon Discount:** Proportionally applied to both orders based on their subtotals



Managing Split Orders in Admin

- Both orders appear independently in the Orders list

- Each order has its own Order ID and status lifecycle
- Pre-booking orders show the scheduled delivery date/time in the order details
- Normal orders show "As soon as possible" or an estimated delivery time

---

9. Categories

The **Categories** tab lets you organise your menu into logical groups.

Category Management

Adding a Category

1. Click the **Categories** tab
2. Fill in:
    - **Category Name** (required)
    - **Description** (optional)
    - **Active** checkbox (enabled by default)
3. Click **"Add Category"**

Category Table

Displays:

- **Name** — Category label
- **Description** — Optional description
- **Active** — Active/Inactive status badge (green/red)
- **Item Count** — Number of menu items in this category
- **Actions** — Toggle (Enable/Disable) and Delete buttons

Toggling a Category

- Disabling a category hides it from the menu form dropdown
- Does not delete the menu items within it
- Active categories populate the category dropdown in the Menu form

Deleting a Category

- Click **Delete** and confirm (action cannot be undone)
- The category is removed from Firestore immediately

> *[!TIP] Use categories like "Breakfast", "Desserts", "Specials" to keep your menu organised. You can temporarily disable a category to hide an entire section from the website without deleting individual items.*

---

10. Store Settings

The **Settings** page (`admin-settings.js`) lets you configure global store preferences. Settings are stored in Firestore at `settings/store_config` .

Accessing Settings

Settings is loaded via the sidebar navigation (if applicable) or by calling `loadSettingsPage()` .

Configuration Sections

10.1 Store Profile

- **Store Name** — Business name displayed across the platform
- **Contact Email** — Primary support email (used in email links across customer site)
- **Contact Phone** — Store phone number (used for WhatsApp ordering and tel links)
- **Store Physical Address** — Business location

10.2 Operations & Delivery

Setting	Description
Store Online Status	Master toggle. When OFF, customers can browse but cannot place orders.
Customer Testing Message	Banner text shown to customers when store is in testing/offline mode.
Standard Delivery Fee (₹)	Default delivery charge applied to all orders.
Free Delivery Threshold (₹)	Orders above this amount get free delivery. Enter 0 to disable free delivery.
Minimum Order Amount (₹)	Minimum cart subtotal to place an order online. Enter 0 to disable.
Est. Delivery Time	Displayed to customers as the default delivery time estimate (e.g., "30-45 mins").
Use Firebase Storage for Images	Toggle between Firebase Storage and inline Data URL for product images (superadmin only).

> *[!NOTE] The "Use Firebase Storage for Images" toggle is only editable by users with `superadmin` role. All other users see a note: "Only superadmins can change this setting."*

10.3 Discounts & Promotions

- **Minimum Order for Discount (₹)** — Cart value needed to qualify for flat discount. Set to 0 to disable.
- **Flat Discount Percentage (%)** — Percentage off applied when minimum order amount is met.

10.4 Payments & Billing

- **Business UPI ID** — For generating UPI QR codes (e.g., `merchant@upi`)
- **Tax Rate (%)** — Global tax percentage applied to all orders (default: 5%). Shown as "GST" in order summaries.
- **Enable Cash on Delivery** — Master toggle for COD availability at checkout.

Settings Sync Mechanism

When settings are saved, they are immediately available to the customer storefront through the `settings-sync.js` module:

1. Admin saves settings to Firestore `settings/store_config`
2. Every customer page loads and calls `syncStoreSettings()`
3. The sync script reads `store_config` from Firestore
4. It patches the DOM: replaces `mailto:` and `tel:` links, updates banners, etc.
5. Customers always see the correct contact details, even if changed minutes ago

Saving Settings

Click **"Save Settings"** at the bottom. Settings persist to Firestore with:

- `updatedAt` timestamp (server timestamp)
- `updatedBy` field (admin email who made the change)

---

11. Customer Support

The **Support** tab (`admin-support.js`) handles all customer inquiries submitted via the "Contact Us" form.

Accessing Support Tickets

Click the **Support** tab. A badge on the tab shows the count of **unattended tickets** (tickets needing admin reply).

Ticket List

Column	Details
ID	Truncated ID with ⚠️ warning icon for unattended tickets
Customer	Name + Email/Phone
Subject/Message	Truncated message
Related Order	Linked order ID (clickable to view order)
Status	Colour-coded badge (Open=Green, In Progress=Orange, Resolved/Closed=Grey)
Date	Creation date
Actions	"View" button

Unattended rows have:

- Bold text for customer name and message
- Orange left border (4px)
- Light orange background (`#fffaf0`)

Filter Options

- **All Tickets**
- **Unattended (Needs Reply)** — Tickets where the last response was from the customer, or status is `open`
- **Open / In Progress / Resolved / Closed**

Real-Time Updates

The ticket list uses Firestore `onSnapshot` for real-time updates:

- No page refresh needed to see new tickets
- When a customer replies, the ticket automatically moves to "unattended"
- Badge count updates instantly
- If the admin is viewing the ticket modal, they may need to reopen it to see new messages (500ms delay for refresh)

Notification on New Activity

When a customer replies to a ticket:

1. The existing notification system detects the new response
2. A notification appears in the bell icon with title and customer name
3. A floating toast notification appears (if no notification module loaded) — auto-dismisses after 6 seconds
4. Clicking the toast navigates to the support tab and opens the ticket

Viewing & Responding to a Ticket

Click **"View"** to open a modal with:

Header Section

- Customer name, email, and phone
- Related order link (if applicable)
- Status dropdown (can change: Open → In Progress → Resolved → Closed)

Linked Order Refund Panel (if applicable)

If the ticket is linked to an order:

- Shows order ID, payment status, and amount
- Green check for refunded, grey for COD unpaid, orange for paid
- **"Issue Refund"** button for eligible orders (paid orders)
- Refunding from here:

1. Updates order `paymentStatus` to `refunded`
2. Sets order `status` to `cancelled`
3. Posts a detailed refund confirmation note in the conversation
4. Auto-resolves the support ticket
5. Logs refund processor email and timestamp

Original Message

The customer's initial message in a bordered box with grey background

Conversation History

Scrollable thread (max-height 300px) of all responses:

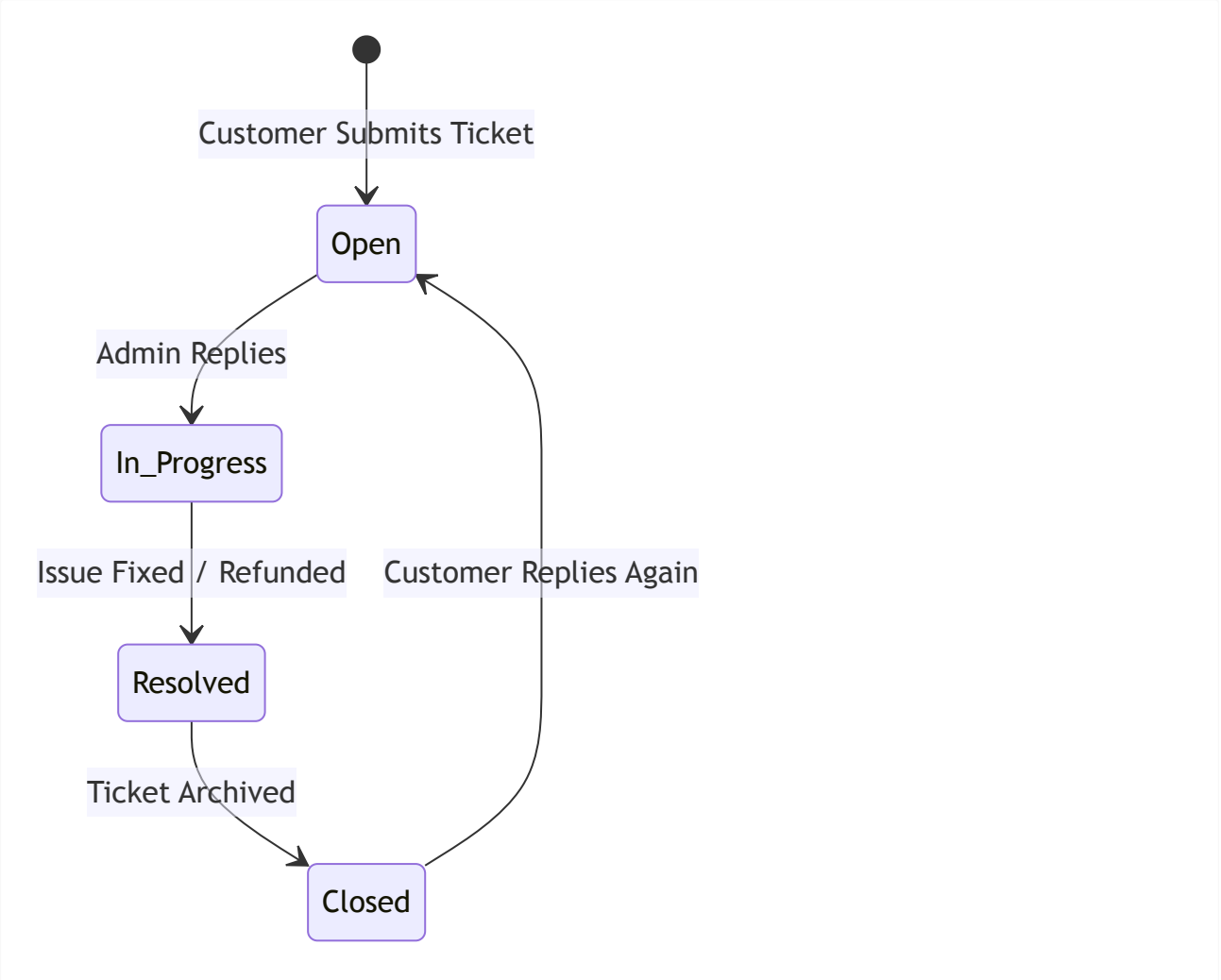
- **Support Team** messages — Blue left border, light blue background
- **Customer** messages — Grey left border, light grey background
- Each message shows sender name and timestamp

Reply Box

- Textarea with placeholder "Type your reply here..."
- **"Send Reply"** button (right-aligned)
- Replying auto-changes ticket status to `in_progress`
- The full response array is saved to Firestore with current timestamp

Ticket Status Progression

`Open` → `In Progress` → `Resolved` → `Closed`



- Tickets can be reopened from Closed back to Open if needed
- Replying to a ticket auto-sets status to `in_progress`

> *[!TIP] Resolve tickets only after you've fully addressed the customer's concern. Use the refund functionality within the ticket modal when appropriate so the entire interaction history is preserved.*

---

12. Customer Management

The **Customers** page (`admin-dashboard.js` — `loadCustomersPage()`) provides an overview of all registered customers.

Accessing Customers

Click **"Customers"** in the sidebar navigation (if available) or call `loadCustomersPage()` .

Customer Stats

- **Total Customers** — Count of all non-admin users
- **New Customers (30 days)** — Users registered in the last 30 days

Customers Table

Column	Details
Name	User avatar (first letter initial) + name + "Customer" role label
Email	Customer email
Phone	Contact number
Location	Saved address/location
Total Orders	Dynamically loaded per customer (may show 0 before load)
Total Spent	Dynamically loaded per customer (may show ₹0 before load)
Joined Date	Registration date
Actions	View (eye) + Edit (pencil) buttons

> *[!NOTE] Order counts and total spent are loaded asynchronously per customer by querying orders where `customer.email` matches. The table initially shows placeholder values (0/₹0) which update as each query completes. Customers without an email address are skipped during order loading.*

Search

Real-time client-side search by any customer field (name, email, phone, address, etc.) using JavaScript `textContent.includes(searchTerm)` .

Viewing Customer Details

Click the **View (eye)** button to open a modal with:

- **Customer header** — Large avatar, name, role badge, active/inactive badge
- **Stats** — Total orders count, total spent (₹), joined date
- **Contact Information** — Email, phone
- **Saved Addresses** — All saved addresses (if multiple; each shown in a bordered card with name, phone, full address, city, pincode)
- **Recent Orders** — Last 5 orders with order ID, date, status, and amount
- **View all orders** link for extended history

Editing a Customer

Click the **Edit (pencil)** button to open a form with:

- Name, Email, Phone
- Address, City, PIN Code
- Active status toggle (checkbox)

Changes are saved via `db.collection('users').doc(customerId).set(updatedData, { merge: true })` with a logged `updatedBy` field.

> *[!NOTE] The email field may load as missing for some customers depending on how their data was stored (possible fields: `email`, `authEmail`, `profileEmail`).*

Export Customers

The **"Export Data"** button is currently a placeholder and will download customer data as a spreadsheet in a future update.

---

13. Analytics & Reports

The **Analytics** page (`admin-analytics.js`) provides powerful, real-time business intelligence using Chart.js v4.4.3 visualisations. All data is fetched directly from Firestore (read-only).

Accessing Analytics

Click **"Analytics"** in the sidebar or call `loadAnalyticsPage()` .

Period Filter

Choose the analysis period at the top:

- **Today** — Hourly breakdown (24 data points)
- **7 Days** — Daily breakdown
- **30 Days** (default)
- **90 Days**
- **All Time**

When you change the period, all KPIs and charts refresh automatically.

KPI Cards (8 metrics)

KPI	Icon	Description
Revenue	₹	Total sales in period
Orders	🛍️	Total order count
Avg Order Value	📈	Revenue ÷ Orders
New Customers	👤	New non-admin users in period
Open Tickets	🎧	Tickets with status = open
Bulk Orders	📦	Orders with bulk-qualifying items
Cancellation Rate	❌	(Cancelled ÷ Total) × 100

KPIs show "..." while loading.

Revenue & Orders Line Chart

- Dual-axis chart (Chart.js):
  - **Revenue (₹)** — Blue line, left Y-axis, filled area
  - **Orders** — Green line, right Y-axis, filled area
- **Hourly breakdown** when viewing "Today" (24 hours with `0:00` to `23:00` labels)
- **Daily breakdown** for other periods (chronologically sorted)
- Responsive; point radius reduces to 0 when >30 data points

Order Status Doughnut Chart

Breakdown of orders by status:

- Pending (amber), Confirmed (blue), Preparing (purple), Ready (cyan), Delivered (green), Cancelled (red)
- 60% cutout; legend positioned on the right
- Tooltip shows exact count on hover

Top 10 Products by Revenue

Horizontal bar chart showing the highest-revenue products:

- Calculated from all `items` or `cartItems` arrays in orders
- Sorted by revenue descending, limited to top 10
- Blue bars with 6px border radius

Advanced Analytics Heatmaps

Ruchikooth provides granular visualization for order operations and kitchen capacities:

- **Order Volume Heatmap**: Displays the frequency of orders placed by day and hour, helping to identify peak operational hours and staffing needs.
- **Booking Capacity Heatmap**: Visualizes the booked kitchen capacity against maximum limits on a calendar grid, highlighting days that are fully booked (red) versus days with available capacity (green). This is crucial for managing pre-booking limits and vendor allocations.

Order Volume Heatmap

A 7×24 grid (days of week × hours):

- Rows: Sun → Sat
- Columns: 0 → 23 (hours)
- Cell colour intensity: lighter → darker blue based on order count relative to max
- Hover tooltip shows: "Day Hour:00 — N orders"
- Empty cells have minimal shading

Top Customers by Spend

Table ranking the top 10 customers by total spend:

- Rank number, avatar (first letter) + name + email
- Order count
- Total amount spent (₹ formatted for Indian locale)

Detailed Reports Section

Six specialised export options as CSV files, accessible via buttons:

Report	Columns	Use Case
Operations Report	Order ID, Date, Status, Delivery Location, Branch, Items Count, Prep Time	Kitchen workflow analysis
Sales Report	Date, Order ID, Items Ordered, Payment Method, Discount (₹), Total Paid (₹)	Revenue analysis

Profit Report	Order ID, Total Revenue, Estimated Cost (60%), Estimated Profit	Profitability estimation
Work Burnout Report	Date, Hour, Orders Processed, Complexity Score	Staffing optimisation
Bulk Ordering Report	Order ID, Customer Name, Customer Email, Phone, Bulk Items, Total Amount, Date	Bulk order tracking
General CSV	Order ID, Customer Email, Customer Name, Status, Amount, Date	General data export

> *[!TIP] Use the **Burnout Report** combined with the **Heatmap** to identify peak hours where kitchen staff may need additional support. The Complexity Score metric (orders × 1 + items × 0.5 per hour) helps quantify workload.*

How Analytics Data is Fetched

The analytics module:

1. Fetches ALL orders from Firestore (no date filter on the query to avoid composite index requirements)
2. Filters orders client-side by the selected period
3. Fetches ALL users (for new customer count)
4. Fetches ALL support tickets (for open ticket count)
5. Stores orders in `window._analyticsOrders` for CSV export
6. Renders all KPIs and charts

---

14. Delivery Management

The **Delivery Management** page (`admin-delivery.js`) is a dedicated workspace for tracking and managing all delivery operations.

Accessing Delivery

Click **"Delivery"** in the sidebar or call `adminDelivery.load()` .

KPI Cards

Four KPI cards at the top:

KPI	Description
Pending Dispatch	Orders confirmed/preparing but not yet assigned to a rider
Out for Delivery	Orders currently with a rider
COD Cash to Collect	Total ₹ value of unpaid COD orders
Completed Today	Deliveries marked complete today

Filters

Three filters in a card above the table:

1. **Delivery Status** — Active (default) / Pending Dispatch / Out for Delivery / Completed / All
2. **Delivery Agent** — All / Unassigned / [Individual agents from staff list]
3. **Search** — Search by customer name, phone, address, or landmark

Delivery Table

Column	Details

Order ID	Clickable (blue, underlined) to view order details via `viewOrderDetails()`
Customer & Address	Name + phone (bold), address with truncation (250px max, tooltip on hover)
Scheduled/Date	Scheduled delivery time + order placement time
Assign Agent	Dropdown of delivery staff + "Assign" button
Payment Info	Payment method badge + amount (bold)
Status	Colour-coded status badge (Confirmed=Indigo, Preparing=Amber, Out for Delivery=Orange with pulse animation, Delivered=Green)
Actions	Context-sensitive buttons

Actions Available

Current Status	Available Action	Description
Confirmed / Preparing	🚚 **Dispatch**	Sets status to `out_for_delivery` (orange button)
Out for Delivery	✅ **Delivered**	Sets status to `completed`; auto-marks COD as `paid` (green button)
Completed (COD unpaid)	💰 **Collect Cash**	Records cash as collected (red button)

Assigning a Delivery Agent

1. Select a delivery agent from the dropdown (populated from staff list filtered by `delivery` role)
2. Changes are saved immediately to Firestore
3. The order's `deliveryAgentEmail`, `deliveryAgentName`, and `deliveryAgentId` fields are updated
4. The table refreshes automatically

Auto-Payment on Delivery

When marking a COD order as delivered:

- `paymentStatus` is automatically set to `paid`
- `cashCollectedAt` is recorded with current timestamp
- `cashCollectedBy` is recorded with the admin's email

---

15. Staff Management

The **Staff Management** module (`admin-staff.js`) lets super admins and managers control access to the admin system.

Accessing Staff

Click **"Staff"** in the sidebar or call `adminStaff.loadStaff()`.

Roles & Privileges

Role	ID	Privileges	Description
Super Admin	`superadmin`	`['all']`	Full system access including settings
Manager	`manager`	`['orders', 'products', 'customers', 'promotions', 'support']`	Operaional management

Cook / Kitchen Staff	`cook`	`['kitchen_only']`	Can only see kitchen order management
Delivery Manager	`delivery`	`['delivery_only']`	Can only see delivery workspace

These privileges control:

- Which tabs/pages are visible in the sidebar navigation
- Which status updates are allowed in order management (see Order Management → Role-Based Status Access)
- Which settings can be modified (e.g., image storage mode requires `superadmin`)

Staff List Table

Column	Details
Name	Staff member's display name
Email	Login email (not editable after creation)
Role	Colour-coded badge (blue)
Joined	Account creation date
Actions	Edit (pencil) + Delete (trash) buttons

Adding a New Staff Member

1. Click the **"Add Staff"** button
2. Fill in:
   - **Name** — Display name
   - **Email** — Login email (used for Firebase Auth)
   - **Password** — Initial password (required for new accounts)
   - **Role** — Select from role dropdown
3. Click **"Add Staff"**
4. The system:
   - Creates a Firebase Auth account via `createUserWithEmailAndPassword` (using a secondary Firebase app instance)
   - Creates a Firestore document in `admins` collection with the new user's UID
   - Signs out and deletes the secondary app instance

Editing a Staff Member

- Click **Edit** (pencil icon) on any row
- Modify Name and Role
- Email is read-only (cannot be changed; input is disabled)
- Password field is hidden (passwords cannot be edited without Admin SDK)

Removing a Staff Member

- Click **Delete** (trash icon) on any row
- Confirm the removal in the dialog
- The Firestore `admins` document is deleted

> *[!CAUTION] Deleting a staff member removes their access immediately. The Firebase Auth account persists to avoid orphaned accounts. To fully remove access, a super admin should also delete the user from Firebase Authentication Console.*

15.1 Inventory & Stock Management

The Inventory Management module allows admins to track real-time stock availability and handle restocking.

- **Stock Dashboard**: View current stock levels for all tracked products.
- **Automated Replenishment**: Upon the completion of vendor work orders, stock levels are automatically updated and replenished in the inventory system, reducing manual entry errors.

15.2 Vendor Purchase Orders

Vendor Purchase Orders provide a comprehensive way to distribute production workloads.

- **Consolidated Orders**: Admins can pool active work orders and assign them to specific vendors.
- **Communication integration**: Send consolidated purchase orders to vendors via Email or WhatsApp.
- **Vendor Specific Pricing**: Track production costs effectively by applying individual vendor pricing to each requested item.

---

16. Back Office

The **Back Office** (`backoffice/index.html`) is a separate, simplified interface for operational staff (kitchen, inventory managers, etc.) without full admin access.

Accessing Back Office

Navigate to the Back Office URL:

- **Production:** `https://ruchikooth.com/backoffice/`
- **Local Development:** `http://localhost:8080/backoffice/`

Login

- Separate login page styled differently from the main admin portal
- Uses the same Firebase Auth system as the admin portal
- Same credentials work across both portals

Layout

Two-column layout:

- **Left Sidebar (260px, green background):** Navigation links with icons
  - Ruchikooth logo (white version with brightness filter)
  - Orders (default active module)
  - Inventory
  - Branches
  - Reports
  - Feedback/Customer Ratings
  - Promotions
  - Logout (separated by a divider)
- **Right Content Area (white/light grey background):** Module-specific content
  - Header with module title and logged-in user email

Sidebar Navigation

Module	Title	Description
Orders	Kitchen Orders	Simplified order view focused on preparation and kitchen workflow
Inventory	Inventory Management	Stock management for ingredients and supplies
Branches	Branch Management	Multi-branch configuration and management

Reports	Admin Reports	Operational reports and summaries
Feedback	Customer Feedback	Customer ratings and reviews
Promotions	Promotions Management	Discount and promotional campaign management

Each module is loaded on demand via dedicated JS scripts:

- `backoffice-orders.js` — `loadOrdersModule()`
- `backoffice-inventory.js` — `loadInventoryModule()`
- `backoffice-branches.js` — `loadBranchesModule()`
- `backoffice-reports.js` — `loadReportsModule()`
- `backoffice-feedback.js` — `loadFeedbackModule()`
- `backoffice-promotions.js` — `loadPromotionsModule()`

Auth Flow

```
auth.onAuthStateChanged(user => {
    if (user) {
        loginView.style.display = 'none';
        mainView.style.display = 'grid';
        document.getElementById('user-display').textContent = user.email;
        loadModule('orders'); // Default module
    } else {
        loginView.style.display = 'flex';
        mainView.style.display = 'none';
    }
});
```

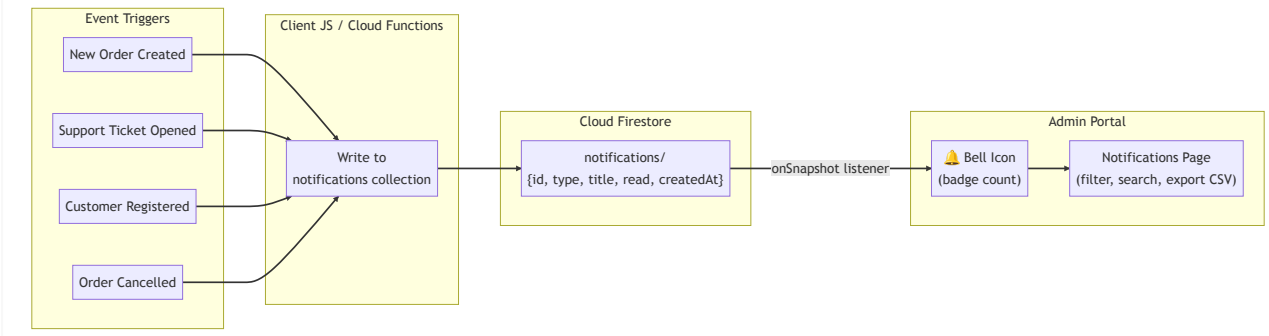
---

17. Notifications System

The admin portal includes a real-time notification system designed to alert staff about important events requiring attention.

Architecture

The notification system follows a **push model** using Firestore's `onSnapshot()` listener:



Notification Bell

Located at the top-right of the dashboard header:

- Shows a **red badge** with the count of unread notifications
- Badge is hidden when count is 0
- Click to open the notification popup panel

Notification Panel

The sliding popup panel (300px wide, right-aligned) shows:

- **Header:** "Notifications" title + "Clear All" button (red text)
- **Notification List:** Scrollable (max-height 400px)
  - Each notification shows:
    - Title (bold, dark)
    - Message (grey, smaller)
    - Blue dot indicator for unread (left-aligned)
  - Click a notification to mark it as read and trigger any associated action
  - Empty state: "No new notifications" centred text
  - "Mark all read" link at the top of the list when notifications exist

When Notifications Fire

Notifications are added automatically in these scenarios:

1. **New order created** — When a customer places an order
2. **Support ticket activity** — When a customer replies to an existing ticket
3. **New customer registered** — When a new user creates an account
4. **Order cancelled** — When an order is cancelled by the customer

Dismiss Behaviour

- Clicking anywhere outside the notification wrapper auto-closes the popup
- Clicking a notification marks it as read, closes the popup, and executes any `onClick` handler
- "Mark all read" marks all current notifications as read in the UI
- "Clear All" removes all notifications and closes the popup

Storage

Notifications are stored in a JavaScript array (in-memory) within the admin portal. They persist only during the current browser session. For persistent notifications across sessions, the system also supports Firestore-backed notifications.

---

18. System Architecture Overview

Understanding the underlying architecture helps admin users appreciate how the system works and why certain behaviours occur.

Technology Stack

Layer	Technology	Purpose
Frontend	Vanilla JS + Custom Web Components	UI rendering, interaction; no framework dependency
Database	Cloud Firestore (NoSQL)	Real-time data sync, role-based access control
Auth	Firebase Authentication	Customer & admin login, session management
Hosting	Firebase Hosting (CDN)	Global edge delivery via CDN
Storage	Firebase Storage	Product images, file uploads
Functions	Firebase Cloud Functions	Server-side logic, payment webhooks
Payments	Razorpay / WhatsApp	Order payment collection
Notifications	Firestore + Cloud Functions	Real-time admin alerts

Why Vanilla JS?

The platform uses Vanilla JavaScript + Custom Web Components instead of frameworks like React, Angular, or Vue:

- **No framework dependency:** Works in any modern browser without compilation
- **Lightweight:** Smaller bundle size, faster page loads
- **Immune to framework churn:** No need to rewrite for new framework versions
- **Easy to hand off:** Simple code structure, no build pipeline complexity

Why Firebase?

Firebase was chosen over traditional databases because:

- **Real-time updates:** Firestore pushes data to all connected clients instantly via `onSnapshot()` — no polling needed
- **Zero server management:** No servers to provision, patch, or scale
- **Automatic scaling:** Handles 1 to 100,000+ concurrent users automatically
- **Built-in authentication:** Handles JWT token generation, session management, password resets
- **Global CDN:** Firebase Hosting replicates files to servers worldwide

Monorepo Structure

```
ruchikooth-ecom/
├── apps/
│   ├── store/         ← Customer Storefront (products.html, order-wizard.html, etc.)
│   ├── admin/         ← Admin Portal (admin.html)
│   └── mobile/        ← iOS & Android (Capacitor wrapper)
└── packages/
    └── shared/
        ├── components/ ← Web Components (<rk-header>, <rk-footer>, <rk-layout>)
        ├── js/         ← Business Logic Modules (order-wizard.js, products.js, etc.)
        ├── css/        ← Global Styles (admin-styles.css, customer-styles.css)
        └── assets/     ← Images, favicon, etc.
```

Data Model (Firestore Collections)

The database uses these main collections:

Collection	Description	Key Fields
`users`	Customer and admin accounts	`name`, `email`, `role`, `addresses[]`, `defaultBranch`
`orders`	All customer orders	`items[]`, `customer{}`, `status`, `paymentMethod`, `total`, `branchId`
`menu_items`	Food products / menu items	`name`, `price`, `category`, `available`, `branchAvailability{}`, `isVeg`
`products`	Product catalog (synced with menu_items)	Same as menu_items + `featured`, `unavailableBranches[]`
`categories`	Menu categories	`name`, `description`, `active`, `itemCount`
`branches`	Physical store locations	`name`, `address`, `isOnline`, `operatingHours{}`, `unavailableProducts[]`
`support_tickets`	Customer support inquiries	`userId`, `orderId`, `issueType`, `message`, `status`, `responses[]`
`notifications`	Admin notifications	`type`, `title`, `message`, `read`, `createdAt`
`settings`	Global store configuration	Single document `store_config` with all settings

`admins`	Admin staff accounts	`name`, `email`, `role`, `privileges[]`
`party_orders`	Catering and party bookings	Event details, date, menu preferences

How Everything Connects

```
Customer places order
    → order-wizard.js creates order in Firestore
        → Cloud Function fires, writes a notification document
            → Admin portal's onSnapshot listener fires
                → Bell badge increments, toast appears
                    → Admin opens order, updates status
                        → Customer's tracking page updates in real time
                            → Customer sees "Delivered" ✅
```

Every interaction follows this pattern: **Client action → Firestore write → real-time propagation → UI update**

Settings Sync Flow

When you update settings, they propagate to the customer site:

```
Admin saves settings in Settings page
    → Firestore settings/store_config updated
        → Customer page loads, calls syncStoreSettings()
            → Reads settings from Firestore
                → Patches DOM: updates mailto/tel links, banner text
```

---

19. Security Architecture

Security is implemented at three layers:

Layer 1 — Firebase Auth (Identity)

- Every user has a cryptographically signed JWT token
- Token is automatically included in every Firestore request
- Cannot be forged or tampered with from the browser

Layer 2 — Firestore Security Rules (Authorization)

Server-side rules inside Google's infrastructure that cannot be bypassed:

```
users collection:      You can only read/write your own document
orders collection:     Any authenticated user can create; you can only read your own orders
settings collection:   Read by all, write by admins only
admin collections:     Requires role == admin
```

> *[!IMPORTANT] The role check in JavaScript is a UX convenience. The real security lives in Firestore Security Rules — server-side rules enforced by Google's infrastructure.*

Layer 3 — Cloud Functions (Server-Side Validation)

Sensitive operations are handled server-side:

- Payment verification via Razorpay webhooks
- Admin role assignment
- Order document validation

What This Means in Practice

Even if a user opens the browser console and tries:

```
updateDoc(db, 'orders/someOrderId', {total: 0})
```

Firestore security rules reject the request with a permission denied error. The client-side code is never the last line of defence.

Deployment Security

- One `firebase deploy` command updates hosting, functions, and rules simultaneously
- Security Rules apply globally the moment they are deployed — no cache invalidation delay
- All communications use HTTPS

---

20. Deployment & Updates

From Code to Production

The entire platform is deployed with a single command:

```
firebase deploy
```

This single command:

1. Uploads all HTML/CSS/JS to Firebase Hosting
2. Deploys Cloud Functions to Google's serverless runtime
3. Applies updated Firestore Security Rules

CDN Replication

Firebase automatically copies your files to edge servers globally:

- Users always hit the geographically nearest server
- Pages load in under 1 second worldwide
- Zero configuration needed

Cloud Functions

- Functions sit dormant until triggered
- Google spins up the function in milliseconds when triggered
- Runs the function, then shuts it down
- No servers to manage, patch, or scale

Development vs Production

Aspect	Local Development	Production
URL	`http://localhost:8080/` (store) / `http://localhost:8081/` (admin)	`https://ruchikooth.com/`
Payment Gateway	Simulated/disabled	Razorpay live
Email	Console logging	Real email delivery
Firebase Storage	May not be available	CDN-backed
CORS	Localhost restrictions apply	Fully configured

Updating the WhatsApp Number

The WhatsApp number is configured through Settings (see Section 10) and syncs automatically.

---

21. Troubleshooting & Known Issues

Login Issues

Issue	Cause	Solution
"You do not have admin access"	User not in `users` collection with role `admin`	Contact super admin to add admin role
Login fails with wrong credentials	Incorrect email or password	Use "Forgot Password" via Firebase Auth UI, or contact super admin

Order Issues

Issue	Cause	Solution
Order not appearing in admin list	Firestore listener delay or filter applied	Click "Refresh" button or clear filters
Split orders not showing both orders	Confirmation page bug (fixed in v2)	Check Orders page for both order IDs
Stock not deducting on order	Track Stock not enabled or pre-booking is on	Verify Track Stock is checked in menu item settings
Payment pending but customer paid	Webhook delay or failure	Click "Convert to COD" to bypass
Cannot change order status	Role restrictions (cook/delivery) or order is cancelled	Only super admins/managers have full status access

Menu & Product Issues

Issue	Cause	Solution
Product not showing in store	Item marked unavailable or category inactive	Check availability toggle and category status
Image not loading	Corrupt URL or Firebase Storage not enabled	Update or re-upload the image
Changes not saving	Firestore permission denied	Check admin privileges

WhatsApp Ordering Issues

Issue	Cause	Solution
WhatsApp message is blank	Browser blocking popups/redirects	Advise customer to check browser settings
Order missing in admin	Customer abandoned checkout before clicking "Complete Order"	No order is created in this case
Wrong number receiving messages	Store phone number incorrect in Settings	Update Settings → Contact Phone (with country code)

Known Items from Active Work Tracker

The following items are noted from the active work tracker as deliberate design decisions or known limitations:

- **Notifications are session-only:** Notifications stored in the browser's JavaScript array are cleared on page refresh. Firestore-backed notification history is available for persistent storage.
- **Settings sync delay:** Changes to settings may take a few seconds to propagate to the customer storefront (dependent on Firestore read timing).
- **Email field variance:** Customer email may be stored in `email`, `authEmail`, or `profileEmail` fields depending on registration method.
- **Export customers:** The "Export Data" button in Customer Management is a placeholder and not yet functional.
- **CORS on localhost:** Cloud Functions may not work from `http://localhost` due to CORS policies. They work correctly on the production domain.

---

22. Frequently Asked Questions

▶ **How do I log in if I forgot my password?**
▶ **Why can't I see certain tabs or buttons?**
▶ **How do I mark an item as out of stock?**
▶ **What happens when a customer pays online but the order is cancelled?**
▶ **Can I edit an order after it's been placed?**
▶ **How do I temporarily close the store?**
▶ **What is the difference between Menu Management and Products?**
▶ **How do pre-bookings work?**
▶ **Can I assign multiple delivery agents to one order?**
▶ **How do I export data for my accountant?**
▶ **How does the WhatsApp ordering work for admins?**
▶ **What should I do if the store is in testing mode?**
▶ **How do split orders work when a customer has pre-booking items?**
▶ **Where is the back office and who should use it?**

Generated for Ruchikooth E-commerce Platform — Comprehensive Admin Manual v3.0