EXPLAIN: Prototype 1 â  Login to POS
Author: Canteen System (Student Prototype)
Date: 2025-09-15

## Overview

This document explains what we accomplished in the first prototype, focusing on the end-to-end flow from Login to the POS (Point of Sale) module. It also lists every file we added or modified and describes the purpose of each change. The goal is to show a clear understanding of how authentication, permissions, POS operations, images/media, order queue, and order history now work together.

## Authentication and Access Control

Purpose: Ensure only authorized users can use POS features and that UI controls respect role-based permissions (admin/manager/staff).

Key Files

- src/components/AuthContext.jsx (modified):
- Centralizes login, logout, token persistence, refresh token handling, and permission checks via ‘can()‘ / ‘hasRole()‘ helpers.
- Wires the API client to always attach the bearer token and auto-refresh or sign-out on 401.
- src/lib/permissions.js (added):
- Mirrors backend default permissions to gate UI controls.
- Staff and Manager can handle order queue and update statuses by default.

Flow

1) User logs in; we persist the token in local/session storage based on the remember preference.
2) ‘AuthContext‘ configures the API client so every request carries the token.
3) Components use ‘useAuth().can(’permission.code’)‘ to show/hide controls.

## POS â  Menu, Images, and Categories

Purpose: Make the POS menu usable and polished: clear category tabs (with an "All" tab), searchable items, and reliable images.

Key Files

- src/hooks/usePOSData.js (modified):
- Loads categories and items, normalizes category names, and injects a first "All" tab that shows every item.
- Normalizes item.image from multiple possible API fields (image, imageUrl, photo, picture).
- Listens for a global ‘menu.items.updated‘ event so POS refreshes after menu changes (e.g., image upload or item edit) without a manual page refresh.
- src/components/pos/MenuSelection.jsx (modified):
- Renders category tabs correctly (no more "[object Object]").
- Shows images if available; supports search across categories.
- src/api/services/menuService.js (modified):
- Normalizes image fields from the backend and builds absolute media URLs using ‘VITE_MEDIA_BASE_URL‘ (preferred) or ‘VITE_API_BASE_URL‘ as a fallback.
- After image upload, returns a cache-busted URL so new images appear immediately.
- src/hooks/useMenuManagement.js (modified):
- After creating/updating items or uploading an image, dispatches the ‘menu.items.updated‘ event so POS refreshes the menu automatically.

## Backend Media and Image Uploads

Purpose: Make images reliably upload and display in Menu Management and POS.

Key Files
- backend/config/urls.py (modified):
- Serves ‘/media/‘ files in development when DEBUG is true.
- backend/api/views_menu.py (modified):
- Image upload endpoint validates content-type and size (now configurable via DJANGO_MENU_IMAGE_MAX_MB, default 25MB). Accepts image/jpg.
- backend/api/views_diag.py (added):
- Diagnostics endpoints including ‘/api/diagnostics/media‘ to verify that DB-referenced images exist on storage.
- backend/api/management/commands/verify_media.py (added):
- ‘python manage.py verify_media [--fix]‘ helps audit or clear broken image references.
Order Queue â  Walk-in and Online Orders
Purpose: Make order queue functional and easy to operate by staff.
Key Files
- src/components/pos/OrderQueue.jsx (modified):
- Fixed date crash by guarding time parsing; shows "time ago" safely.
- Shows the correct status chips.
- For Walk-in orders, action buttons progress status: Pending â  Start Preparing â  Mark Ready â  Complete. These buttons are permission-gated via ‘can(’order.status.update’)‘.
- src/api/services/orderService.js (modified):
- Normalizes backend statuses so the UI logic is consistent:
â¢ in_queue â  pending
â¢ in_progress â  preparing
- Keeps dates normalized to ISO strings.
Order Creation and Payment in POS
Purpose: Ensure a smooth flow from building an order to payment.
Key Files
- src/components/POS.jsx (modified):
- Wires together MenuSelection, CurrentOrder, and OrderQueue tabs.
- Polls order queue and refreshes on status updates.
- Opens the Payment modal and Order History modal.
- src/hooks/usePOSLogic.js (modified):
- Adds items, updates quantities, applies discounts, computes totals.
- Creates orders via the backend and then processes payment.
- src/components/pos/PaymentModal.jsx (modified):
- Uses permission checks for payment processing.
Backend Order and Payment Behavior
Purpose: Keep server-side data accurate and aligned with the UI flow.
Key Files
- backend/api/views_orders.py (added/modified):
- When creating orders, defaults ‘payment_method=’cash’‘ for walk-in orders.
- Enforces state transitions on the server: pending/in_queue â  in_progress â  ready â  completed/cancelled.
- backend/api/views_payments.py (modified):
- After successful payment, updates the parent orderâ s payment_method to the actual method used.
- backend/api/management/commands/set_walkin_default_cash.py (added):
- Backfills existing walk-in orders with empty payment_method to ’cash’.

Order History (Modal)
Purpose: Allow staff to review past orders.
Key Files
- src/hooks/useOrderManagement.js (modified):
- Stabilized fetches; added an in-flight guard to avoid overlapping requests and rate limits.
- History loads on-demand when the modal opens to reduce background load.
- Provides a fallback to recent orders if no completed/cancelled/refunded orders exist yet, so the modal never looks empty during early testing.
- src/components/pos/OrderHistoryModal.jsx (modified):
- Renders robustly (handles missing fields), shows loading/empty state, and includes a Refresh button.
Login â  POS Flow (Narrative)
1) Login: User signs in, token is stored. AuthContext wires token to API calls, and ‘can()‘ exposes effective permissions for UI gating.
2) Menu & Categories: POS fetches categories/items; an "All" tab gives quick access to all items. Images load from absolute media URLs; after image updates, POS auto-refreshes.
3) Build Order: Staff selects items, adjusts quantities, and applies optional discounts. Totals are computed live.
4) Queue & Status: Placing an order enqueues it. Walk-in orders start as pending (server in_queue). Staff progresses states via buttons to preparing, ready, and completed.
5) Payment: Processing payment records a PaymentTransaction and syncs the orderâ s payment_method. Walk-in orders default to cash; card/mobile requires tokens per provider.
6) History: Staff can open Order History to view completed/cancelled/refunded orders. Fetch is on-demand, with loading/empty state and optional refresh.
List of Added/Modified Files
Frontend
- src/components/AuthContext.jsx (modified) â  Auth/token/permissions flow.
- src/components/POS.jsx (modified) â  Main POS container and tabs.
- src/components/pos/MenuSelection.jsx (modified) â  Category tabs, search, images.
- src/components/pos/OrderQueue.jsx (modified) â  Time-safe rendering and action buttons.
- src/components/pos/OrderHistoryModal.jsx (modified) â  History list with loading/empty and refresh.
- src/components/pos/PaymentModal.jsx (modified) â  Payment with permission checks.
- src/hooks/usePOSData.js (modified) â  Menu/categories loader, All tab, image normalization, auto-refresh.
- src/hooks/usePOSLogic.js (modified) â  Cart/discounts/totals, order creation, payment call.
- src/hooks/useMenuManagement.js (modified) â  Dispatch POS refresh events on item/image updates.
- src/hooks/useOrderManagement.js (modified) â  Stable, guarded fetches for orders/queue/history.
- src/api/services/menuService.js (modified) â  Image URL normalization, absolute URLs, cache-busting.
- src/api/services/orderService.js (modified) â  Status and date normalization, queue/history fetch.
- src/lib/permissions.js (added) â  Default role permissions for UI gating.
Backend

- backend/config/urls.py (modified) â  Serve /media in DEBUG.
- backend/api/views_menu.py (modified) â  Larger uploads; stricter validation.
- backend/api/views_diag.py (added) â  /api/diagnostics/media for media health checks.
- backend/api/management/commands/verify_media.py (added) â  Verify/fix broken image references.
- backend/api/views_orders.py (added/modified) â  Default walk-in payment_method='cash'; order creation.
- backend/api/views_payments.py (modified) â  Sync order payment_method after payment.
- backend/api/management/commands/set_walkin_default_cash.py (added) â  Backfill existing walk-in orders.

Demo Checklist (for Adviser)

1) Login with a staff account; confirm POS menu loads.
2) Upload a menu item image in Menu Management; verify it appears immediately in both Menu Management and POS (thanks to absolute URLs + auto-refresh).
3) Add items to order in POS, apply a discount, and process a cash payment.
4) Check Order Queue: progress a Walk-in order through Pending â  Preparing â  Ready â  Completed.
5) Open Order History; click Refresh to fetch results on demand.
6) (Optional) Visit /api/diagnostics/media to validate image files.

Environment Notes

- Frontend: set VITE_MEDIA_BASE_URL to your backend origin (e.g., http://localhost:8000). VITE_API_BASE_URL can be http://localhost:8000/api.
- Backend: ensure DEBUG=True in dev so /media is served; in production, serve /media via web server (e.g., Nginx alias).

End of document.