Branch-wise Consolidated Report

Branch-wise Implementation Guide (End-to-End)

Date: 12 Feb 2026

Goal

Make the entire application **branch-aware** so that: - Every page loads data **branch-wise** (no cross-branch leakage by default) - All transactions (Orders → Items → Kitchen/Bar tickets → Payments → Closing → Reports) are **scoped to the active branch** - Master pages (Menu, Categories, Tables, Counters, Users, Settings, etc.) are either: - Branch-owned and filtered by branch, or - Global but optionally overridable per branch (explicitly designed)

This guide describes the recommended approach, design decisions, schema changes, rollout steps, and validation checklist.

---

1) Decide the Branch Model (must be finalized first)

1.1 Users and Branch access

Choose one: 1) **Single-branch user** (simplest): each user belongs to exactly one branch. 2) **Multi-branch user**: user can access multiple branches and must select an **Active Branch**. 3) Add **privileged roles** that can view multiple or all branches.

Recommended for growth: **multi-branch access with a required Active Branch**.

1.2 What is branch-owned vs global?

Create a table (design contract) for every major entity:

Typically branch-owned (recommended) - Tables, Table Sections - Counters / POS terminals - Orders, OrderItems, KitchenTickets, Bar tickets (BOT) - Payments, PaymentSplits, Refunds, Complimentary - Day closing / cash closing / collection register - Audit trail (if it contains branch-sensitive data)

Typically global (optional) - Menu master (Categories, MenuItems, Modifiers) - Tax rules, configuration defaults

Hybrid pattern (common) - Menu master global, but **price / availability / tax** can be overridden per branch.

1.3 Branch scope rules

Write these rules and keep them consistent: - “If active branch is X, show ONLY branch X data” - “Admin can switch branch; normal users may be restricted” - “Any access by ID (e.g., /Order/Details?id=123) must validate that record belongs to active branch unless privileged”

---

2) Database Strategy (recommended)

2.1 Add a `Branches` table

Create `Branches` with at least: - `Id` (PK) - `BranchCode` (unique) - `BranchName` - `IsActive` - Address/phone fields (optional)

2.2 Add `BranchId` to branch-owned tables

Add a `BranchId` foreign key column and index it.

Recommended minimum (core transactions): - `Orders(BranchId)` - `OrderItems(BranchId)` (optional if always joined via Orders, but recommended for performance) - `KitchenTickets(BranchId)` - `KitchenTicketItems(BranchId)` - `Payments(BranchId)` - `Tables(BranchId)` - `TableTurnovers(BranchId)` - `Counters(BranchId)`

If there are additional operational tables (discount approvals, void logs, etc.), add `BranchId` there too.

2.3 Constraints + indexes

For each transactional table: - `BranchId` should become `NOT NULL` after backfill - Add FK: `BranchId → Branches(Id)` - Index patterns: - `IX_Orders_BranchId_CreatedAt` - `IX_Orders_BranchId_Status_CreatedAt` - `IX_Payments_BranchId_PaymentDate`

2.4 Stored procedure contract

For every stored procedure that reads/writes branch-owned data: - Add parameter: `@BranchId INT` - Enforce scope: - For reads: `WHERE BranchId = @BranchId` - For writes: set `BranchId = @BranchId` - Defensive validation: - If procedure takes a record ID, confirm the record belongs to `@BranchId`

Example rule: - If `@OrderId` is passed, validate: `Orders.Id=@OrderId AND Orders.BranchId=@BranchId`.

---

3) Application Strategy (end-to-end)

3.1 Establish an “Active Branch” source of truth

Recommended: - Store Allowed Branch IDs in DB (mapping) - On login, set an `ActiveBranchId` in Session - Optionally also store in auth claims for easier access

Create a single helper method used everywhere: - `GetActiveBranchId()`

3.2 Branch switch UI

Add a Branch selector in top navigation: - For multi-branch users: dropdown to switch Active Branch - On switch: - Update session ActiveBranchId - Clear branch-dependent caches (POS menu cache, counter selection cache) - Redirect to dashboard

3.3 Enforce branch in every endpoint

For any controller action that loads or modifies data: - Derive `branchId = GetActiveBranchId()` - Pass `branchId` into stored procedures - If performing a direct SQL query, add `BranchId` filter - If endpoint accepts an ID (OrderId, PaymentId, TableId): validate it belongs to Active Branch

3.4 Don’t trust client input

Never accept BranchId from browser without validation. - Browser can only *request* a branch switch - Server validates the user can access the requested branch

---

4) Master Pages (branch-wise from Users to Menu)

4.1 Users

Pick one: - **User has BranchId** (single-branch) - **UserBranchAccess(UserId, BranchId)** (multi-branch)

Recommended: mapping table for multi-branch.

UI changes: - Create/edit user should include: - Default branch - Allowed branches list - Role-based privilege (can view all branches)

4.2 Tables, Sections, Reservations

These should almost always be branch-owned: - Filter list pages by ActiveBranchId - Enforce BranchId during create/update

4.3 Counters / POS setup

Counters should be branch-owned: - When selecting POS counter, only show counters for ActiveBranchId - When loading POS dashboard/orders, filter by ActiveBranchId

4.4 Menu and pricing model

Choose one design:

Option A: Fully branch-owned menu (simpler, but duplicates data) - Menu tables include BranchId and are filtered

Option B: Global menu + branch overrides (recommended) - Global: `MenuItems`, `Categories`, `Modifiers` - Branch override tables: - `BranchMenuItemSettings(BranchId, MenuItemId, PriceOverride, IsAvailable, TaxOverride, ...)`

This keeps master data consistent and allows per-branch differences.

4.5 Settings / GST / Taxes

Decide: - Global settings with optional branch overrides (recommended) - Or strictly per-branch settings

Implement as: - `Settings` table global - `BranchSettings(BranchId, Key, Value)` override

---

5) Transaction Flows (Food/Bar/POS/Kitchen/Payment)

5.1 Order creation

When creating an order: - Always stamp `Orders.BranchId = ActiveBranchId` - If the order is tied to a Table/Turnover/Counter: - Validate those belong to the same branch

5.2 Add items / kitchen tickets

- Item inserts must match order's branch
- Any "kitchen ticket creation" must be filtered/scoped to branch

5.3 Dashboards

Order dashboard, Bar dashboard, Kitchen dashboard: - Add `BranchId = ActiveBranchId` filter to the queries

5.4 Payments

For all payment actions: - Validate the order belongs to ActiveBranchId - Store `BranchId` in payment tables

5.5 Closing / day closing

Every closing summary must: - Filter only orders/payments from ActiveBranchId - Store `BranchId` on closing tables

---

6) Reports (branch-wise)

For every report stored procedure: - Add `@BranchId` (or `@BranchIds` for privileged roles) - Add `WHERE BranchId = @BranchId` - Ensure exports (CSV/Excel/PDF) use the same query path

Be careful with: - “Totals” cards on dashboards - Financial summaries - GST breakup - Collection register

---

7) Data Migration Plan (safe rollout)

7.1 Phase 1: Add schema (nullable)

- Add `Branches`
- Add `BranchId` columns as NULLABLE
- Add indexes (optional at this stage)

7.2 Phase 2: Backfill

Populate BranchId for existing data using deterministic rules: - Prefer: `Orders.CounterId → Counters.BranchId` - Else: `Orders.TableTurnoverId → Tables.BranchId` - Else: default to “Main” branch

Backfill dependent tables: - `OrderItems.BranchId = Orders.BranchId` - `Payments.BranchId = Orders.BranchId` (or from payment source)

7.3 Phase 3: Enforce NOT NULL + FK

After verification: - Convert BranchId columns to NOT NULL - Add foreign keys to Branches - Add required indexes

7.4 Phase 4: Deploy app changes

- Login sets ActiveBranchId
- All queries filter by branch

7.5 Phase 5: Cleanup

- Remove legacy code paths that don’t pass BranchId
- Add monitoring / audit for cross-branch attempts

---

8) Hardening (prevent cross-branch access by ID)

8.1 Always validate record ownership

For endpoints like: - `/Order/Details?id=...` - `/Payment/Manage?orderId=...` - `/Kitchen/Ticket?id=...`

Server must validate: - `record.BranchId == ActiveBranchId` unless privileged.

8.2 Central authorization rule

Add a helper: - `UserCanAccessBranch(branchId)` - `UserCanViewAllBranches()`

---

9) Testing Plan (mandatory)

Create 2 branches: A and B.

9.1 Access control tests

- Branch A user cannot see Branch B orders/tables/payments
- Branch switch changes data immediately

9.2 Transaction tests

- Create order in A, ensure it never appears in B dashboards
- Payment in A cannot be applied from B UI

9.3 Report tests

- Reports totals match dashboard totals for same branch
- Exports match on-screen data

9.4 Performance

- Ensure indexes exist for `(BranchId, CreatedAt)` style filters

---

10) Recommended Implementation Order (step-by-step)

1. Finalize branch ownership matrix (transaction tables + master tables)
2. Create `Branches` + user-branch mapping tables
3. Add ActiveBranchId in login + UI selector
4. Add `BranchId` to Tables + Counters + TableTurnovers first

5. Add `BranchId` to Orders and enforce it end-to-end
6. Propagate to OrderItems, KitchenTickets, Payments
7. Update all dashboards to filter by branch
8. Update all reports SPs + exports
9. Update master pages (Menu/Categories/Modifiers) according to chosen model
10. Hardening + test suite + rollout

Notes / Common pitfalls

- If “MenuItems are global” but “availability is branch-wise”, you must have a branch override table; otherwise you will constantly fight inconsistent behavior.
- If you allow multi-branch users, always store an ActiveBranchId; showing mixed data across branches is the most common source of report mismatches.
- Ensure POS cached data (menu, counters) is scoped by branch; otherwise users will see wrong price/availability.

Deliverables Checklist (what you should end up with)

- `Branches` master + `UserBranchAccess` mapping (if multi-branch)
- BranchId columns + FKs + indexes on all branch-owned tables
- Updated stored procedures: all take `@BranchId`
- Login + session stores ActiveBranchId
- Branch switch UI
- Every controller action filters by ActiveBranchId
- All reports and exports filter by ActiveBranchId
- Migration scripts + backfill scripts + validation SQL

Branch-wise Implementation (Repo + DB Scan Report)

This document is a repo-specific assessment for adding a **Restaurant Branch** concept across login → all pages → all transactions, with **admin consolidated (all-branches) reporting**.

It is based on scanning this workspace (C# controllers/views + SQL scripts). It does **not** implement anything; it’s a step-by-step workaround / rollout plan.

---

1) What the scan shows (current state)

Repo size (quick stats) - C# files: ~197 - Controllers: ~49 - Razor views: ~166 - SQL scripts: ~153

Where data access lives - A lot of SQL is written inline inside controllers (not a centralized repository layer). - Stored procedures are also used (at least ~91 call sites found), so branch filtering must be done in both: - controller inline SQL - stored procedures / views

High-traffic transactional areas (most likely to need strict branch filtering) - Orders / POS: `OrderController`, `OrderController_NewItems`, and related SQL procs. - Payments / split bills / settlement: `PaymentController` + `Payments_Setup.sql`. - Bar (BOT module): `BOTController` + `BOT_Setup.sql`. - Kitchen tickets: `KitchenController` + `Kitchen_Setup.sql`. - Reports: `ReportsController` + multiple report stored procedures. - Table service & reservations: `TableServiceController`, `ReservationController` + setup scripts.

Important discovery: "BranchID" already exists but for Hotel Room Service This repo already uses a *hotel* branch concept: - `Orders.H_BranchID` (and other hotel-related columns like booking/room) - View: `vw_RoomService_PendingSettlement` maps `o.H_BranchID AS BranchID` - Procedure: `usp_GetRoomServicePendingSettlementDetails(@BranchID)` filters on `o.H_BranchID = @BranchID` - UI: `Views/Order/Create.cshtml` has `hotelBranch` dropdown + `/Order/GetCheckedInOccupiedRooms?branchId=...`

Implication: - Do **not** reuse `H_BranchID` for restaurant branches. - Introduce a **separate restaurant branch key**, e.g. `Orders.BranchId` (recommended), and keep hotel integration intact.

---

2) Design decision you must confirm (before coding)

You need to decide these 4 items up-front, because they change both schema and UI behavior:

1. **Branch scope model**
   - **Strict branch isolation**: every transactional table row belongs to exactly one branch.
   - Admin may read all branches; normal users only their branch.
2. **Branch selection UX**
   - Option A: user is tied to a single branch (no switch UI).
   - Option B (your request): user can **switch branch** (session-based active branch).
3. **Master data (Menu/Prices) strategy**
   - Global masters shared across branches (simpler) vs.
   - Branch-specific menu/pricing (more complex but common for chains)
4. **Counter strategy**
   - `Counters` are almost always branch-specific (recommended), because POS counter belongs to a physical location.

---

3) Recommended branch data model (DB)

3.1 Core tables

Create: - `Branches (Id, BranchCode, BranchName, IsActive, CreatedAt, UpdatedAt, ...)` - `UserBranches (UserId, BranchId, IsDefault, IsActive, ...)`

Add: - `Users.DefaultBranchId` (optional but practical)

3.2 Transactional tables that should get `BranchId`

At minimum: - `Orders` (most important)
- `OrderItems` (optional if always joined from Orders, but recommended for indexing / safety) - `Payments`, `SplitBills`, `SplitBillItems` - `TableTurnovers`, `ServerAssignments` - `Reservations`, `Waitlist`, `Tables` (or at least tables/turnovers) - BOT module: `BOT_Header`, `BOT_Detail`, `BOT_Bills`, `BOT_Payments` - Kitchen module: `KitchenTickets`, `KitchenTicketItems` (and related) - Day closing / cashier: `CashierDayOpening`, `CashierDayClose`, `DayLockAudit` - Online ordering: `OnlineOrders`, `OnlineOrderItems` (and related)

3.3 Foreign keys and indexes (performance + correctness)

- Add FK: `... BranchId → Branches.Id`
- Add indexes:
  - `Orders(BranchId, CreatedAt)`
  - `Orders(BranchId, Status)`
  - `Payments(BranchId, CreatedAt)` or `Payments(OrderId)` + join filter
  - `KitchenTickets(BranchId, Status, CreatedAt)`
  - `BOT_Header(BranchId, Status, CreatedAt)`

---

4) Application approach (ASP.NET Core MVC)

4.1 Where to store the active branch

This app already uses Session for POS counter selection (`POS.SelectedCounterId`). Reuse the same pattern: - Session keys: - `BRANCH.ActiveBranchId` - `BRANCH.ActiveBranchName` (optional)

4.2 How to pick the branch at login

Recommended flow: - After successful login: - fetch allowed branches for user - if only 1 branch → set session active branch automatically - if multiple branches → show a “Select Branch” screen once, then store in session

4.3 Enforcing branch scope

Because SQL is often inline in controllers, the safest pattern is: - Make a **single helper** that resolves active branch id (or throws) - Every query that reads/writes transactions must include: - `... WHERE BranchId = @BranchId` - or `JOIN Orders o ... WHERE o.BranchId = @BranchId`

For Admin consolidated reporting: - allow `@BranchId = NULL` meaning “all branches” - BUT only if user has an `Admin`/`SuperAdmin` role/claim

---

5) Repo impact map (what you will need to touch)

5.1 Controllers that are branch-sensitive (highest priority)

These controllers were found referencing Orders/Payments and are the main risk for cross-branch leakage: - `OrderController.cs` - `OrderController_NewItems.cs` - `PaymentController.cs` - `ReportsController.cs` - `BOTController.cs` - `KitchenController.cs` - `OnlineOrderController.cs` - `ReservationController.cs` - `TableServiceController.cs` - `AuditTrailController.cs`

5.2 SQL scripts / procedures (examples)

You will need to branch-scope: - Order creation / item add / dashboards / recent orders - Payment info + payment processing - BOT dashboard / listing / settlement - Kitchen dashboards / tickets by status - Reports procedures (sales/discount/GST/collection register/order report/etc.) - Day closing procedures

5.3 Existing Hotel Room Service branch fields (do not break)

Files that currently depend on hotel branch (`H_BranchID`) and should remain separate: - `RestaurantManagementSystem/RestaurantManagementSystem/SQL/usp_GetRoomServicePendingSettlementDetails.sql` - `RestaurantManagementSystem/RestaurantManagementSystem/SQL/vw_RoomService_PendingSettlement.sql` - `RestaurantManagementSystem/RestaurantManagementSystem/Views/Order/Create.cshtml` - `RestaurantManagementSystem/RestaurantManagementSystem/Controllers/OrderController.cs` (room-service flow) - `RestaurantManagementSystem/RestaurantManagementSystem/Controllers/PaymentController.cs` (room-service settlement references)

Recommendation: - Keep hotel-related parameter naming as `HotelBranchId` or `HBranchId` in C# models to reduce confusion. - Introduce restaurant `BranchId` for restaurant module.

---

6) Step-by-step workaround / rollout plan (safe phases)

Phase 0 — Finalize rules (1–2 days)

- Confirm master data strategy: global vs branch-specific menu/prices.
- Confirm whether order numbers are global across branches or per-branch.
- Confirm whether admin can switch branch and/or view all.

Phase 1 — DB foundation (2–4 days)

- Create `Branches` and `UserBranches`.
- Add nullable `BranchId` columns to **top-level** transaction tables first (Orders, Payments, BOT, Kitchen, Reservations).
- Backfill `BranchId` for existing rows:
  - If you only have one branch today: set all existing rows to that branch.
  - If multiple branches already exist implicitly (by Counter or hotel branch): define mapping rules explicitly.
- Add indexes on `(BranchId, CreatedAt/Status)`.

Phase 2 — App "active branch" plumbing (2–4 days)

- Add a branch selector after login (only when user has >1 allowed branch).
- Store active branch in session (`BRANCH.ActiveBranchId`).
- Add a small helper (single source of truth) to get active branch in controllers.

Phase 3 — Enforce branch on write paths (3–7 days)

Do this before read paths to prevent new cross-branch records. - Order creation: insert `Orders.BranchId = @BranchId`. - Add/update order items: ensure `OrderId` belongs to the active branch. - Payments: ensure payment is applied only to an order in the active branch. - BOT/Kitchen: ensure created rows inherit branch from the order.

Phase 4 — Enforce branch on read paths (5–10 days)

- Dashboards (food + bar + kitchen): filter by active branch.
- POS views and active order lists: filter by branch.
- Reports:
  - Normal users: filter by branch only.
  - Admin: add optional branch filter + "All branches" option.

Phase 5 — Make BranchId NOT NULL + add FKs (1–2 days)

After app is fully writing correct BranchId: - Make `BranchId` required on core transactional tables. - Add strict foreign keys.

Phase 6 — Hardening + QA (ongoing)

- Add automated checks (or at least manual test cases) for:

- user from Branch A cannot view Branch B orders
- payment cannot be applied across branches
- reports totals match per-branch and all-branches

---

7) High-risk areas / gotchas (specific to this repo)

1. **Two branch concepts**
   - Hotel room-service uses `H_BranchID` already.
   - Restaurant branch must be separate to avoid logic conflicts.
2. **Inline SQL scattered across controllers**
   - Branch conditions are easy to miss; expect iterative hardening.
3. **Counters & POS**
   - POS uses selected counter in session.
   - Recommended: `Counters` should have `BranchId`, and selected counter must belong to active branch.
4. **Reports stored procedures**
   - Many reports are SQL procedures; each needs either:
     - mandatory `@BranchId` parameter, or
     - optional `@BranchId` with admin-only all-branch mode.
5. **Backfill rules**
   - If historical data cannot be mapped to a branch, you'll need a "Legacy/Unknown" branch.

---

8) Practical next step (if you want me to continue the scan)

If you want an even more actionable deliverable, I can generate a second document that lists: - each controller action that needs `BranchId` filter - each stored procedure that queries `Orders/Payments` and needs a `@BranchId` input

Tell me your decision for these 2 items and I'll tailor that list: 1) Menu data is **global** or **branch-specific**? 2) Reports: Admin needs **All branches at once** or **switch branch only**?

Branch-wise Implementation — Exact Touch List (Menu/Pricing Branch-wise + Admin All-Branches Reporting)

Decisions confirmed (12 Feb 2026): - **Menu/Pricing is branch-wise** - **Admin reporting shows all branches at once** (consolidated)

This is a repo-specific "what to change" list (controllers + SQL procedures) based on scanning this workspace.

A) DB changes (must-do)

A1) New tables

1. `dbo.Branches`
   - `Id INT IDENTITY PK`
   - `BranchCode NVARCHAR(20) UNIQUE`
   - `BranchName NVARCHAR(120)`
   - `IsActive BIT`, `CreatedAt`, `UpdatedAt`
2. `dbo.UserBranches`
   - `UserId INT FK -> Users.Id`
   - `BranchId INT FK -> Branches.Id`
   - `IsDefault BIT`, `IsActive BIT`, timestamps
   - Unique constraint `(UserId, BranchId)`

A2) BranchId columns (recommended minimum)

Restaurant transactional - `Orders.BranchId` (do NOT reuse `H_BranchID`) - `Payments.BranchId` (or enforce via join from Orders; still recommended) - `OrderItems.BranchId` (recommended for performance & safety) - `SplitBills.BranchId`, `SplitBillItems.BranchId` - `TableTurnovers.BranchId`, `ServerAssignments.BranchId` - `Reservations.BranchId`, `Waitlist.BranchId`, `Tables.BranchId` - Kitchen: `KitchenTickets.BranchId`, `KitchenTicketItems.BranchId` - Bar/BOT: `BOT_Header.BranchId`, `BOT_Detail.BranchId`, `BOT_Bills.BranchId`, `BOT_Payments.BranchId` - Day closing: `CashierDayOpening.BranchId`, `CashierDayClose.BranchId`, `DayLockAudit.BranchId` - Online ordering: `OnlineOrders.BranchId`, `OnlineOrderItems.BranchId`, `WebhookEvents.BranchId` (if needed)

Menu + Pricing (branch-wise requirement) At least one of these models must be chosen. Recommended is “branch-scoped menu rows”. - `Categories.BranchId` - `SubCategories.BranchId` - `MenuItems.BranchId` - `Modifiers.BranchId` - Any mapping tables used by menu/kitchen (e.g. `MenuItemModifiers`, `MenuItemKitchenStations`) should either: - inherit from `MenuItems`, or - store `BranchId` for faster filtering.

A3) Indexes (performance)

- `Orders(BranchId, CreatedAt DESC)`
- `Orders(BranchId, Status, CreatedAt DESC)`
- `MenuItems(BranchId, IsAvailable, CategoryId)`
- `Payments(BranchId, CreatedAt DESC)`
- `KitchenTickets(BranchId, Status, CreatedAt DESC)`
- `BOT_Header(BranchId, Status, CreatedAt DESC)`

A4) Important: existing Hotel Room-Service branch stays separate

This repo already uses **hotel** branch fields (`Orders.H_BranchID`) for room-service settlement. - Keep those flows unchanged. - Restaurant branch must be `Orders.BranchId` (new).

B) App plumbing (must-do)

B1) Session keys

Add active branch selection (similar to existing POS counter session approach): - `BRANCH.ActiveBranchId` - `BRANCH.ActiveBranchName` (optional)

B2) Login / switch branch

Implement: - After login: load allowed branches for user. - If 1 branch → set it. - If many → force select branch (once) then proceed. - Provide a “Switch Branch” option (clears cached lists where needed).

B3) Enforcement rule

For non-admin users: - Every read/write must be filtered by `BranchId = ActiveBranchId`.

For admin: - Most pages still work in single active branch mode. - Reports allow **All branches** consolidated: - pass `@BranchId = NULL` (or `0`) to report procedures. - only if admin role/permission.

C) Controllers to modify (repo scan result)

These controllers were identified as touching Orders/Payments and will need branch enforcement.

C1) Orders & POS

File:
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/OrderController.cs Actions seen in scan (non-exhaustive): - `Dashboard`, `Create`, `Details`, `GetOrderDashboard` - POS: `POSOrder`, `SetPOSCounter`, `CreatePOSOrder`, `CreatePOSOrderAjax`, `UpdateMultipleOrderItems` - Items: `AddItem`, `QuickAddMenuItem`, `FireItems`, `SubmitOrder` - Hotel-only: `GetHotelBranches`, `GetCheckedInOccupiedRooms` (keep using `H_BranchID` path)

What to change: - On create order: set `Orders.BranchId = ActiveBranchId` (restaurant orders) - On add/update items: validate the order belongs to branch - On listing dashboards: filter by branch - Keep room-service (OrderType=4) flow separate from restaurant branches.

File:
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/OrderController_NewItems.cs - Same branch rules as above for any order-item write APIs.

C2) Payments

File:
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/PaymentController.cs Actions seen: - `Index`, `ProcessPayment`, `ProcessSplitPayments` - `VoidPayment`, approvals (`ApprovePayment`, `RejectPayment`, `ApprovePaymentAjax`, `RejectPaymentAjax`) - Dashboards: `Dashboard`, `BarDashboard` - Print flows: `Print`, `PrintBill`, `PrintPOS`

What to change: - Any payment operation must confirm the `OrderId` belongs to active branch. - Payment listing/dashboard queries must filter by branch. - If you add `Payments.BranchId`, always populate it from the order's branch.

C3) Bar / BOT

File:
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/BOTController.cs Actions seen: - `Dashboard`, `Tickets`, `Details`, `UpdateStatus`, `Void` - `BarOrderCreate`, `BarOrderDashboard`

What to change: - Ensure BOT header/detail rows are created with `BranchId` (inherit from Orders). - Filter BOT dashboards and reports by branch.

C4) Kitchen

File:
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/KitchenController.cs Actions seen: - `Dashboard`, `Tickets`, `TicketDetails` - status updates + station management (`Stations`, `SaveStation`, etc.)

What to change: - Filter kitchen ticket lists/dashboard by branch. - Station management: decide whether kitchen stations are branch-wise (almost always yes).

C5) Reports (Admin all-branches)

File:
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/ReportsController.cs Actions seen: - `Sales`, `Orders`, `Menu`, `Customers`, `Financial` - `Kitchen`, `Bar` - `DiscountReport`, `GSTBreakup`, `CollectionRegister`, `CashClosing`, `FeedbackSurveyReport`

What to change: - Add branch selector in report UI: - Normal users: fixed to active branch (no "All"). - Admin: allow `All branches`. - Pass `@BranchId` into stored procedures: - `@BranchId = ActiveBranchId` normally - `@BranchId = NULL` for admin consolidated

C6) Tables/Reservations

Files: -
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/ReservationController.cs -

RestaurantManagementSystem/RestaurantManagementSystem/Controllers/TableServiceController.cs

What to change: - Tables/Reservations/Waitlist/Turnovers must be branch-scoped. - All lists and lookups filtered by branch.

C7) Online ordering

File:
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/OnlineOrderController.cs Actions seen: - `Dashboard`, `Index`, `Details`, `SyncOrder`, webhook receiver

What to change: - Decide whether an online source is branch-wise. Usually: **yes**. - Store `BranchId` on online orders and enforce during sync.

C8) Menu + Pricing (branch-wise requirement)

Files: -
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/MenuManagementController.cs -
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/MenuController.cs -
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/CategoryController.cs -
RestaurantManagementSystem/RestaurantManagementSystem/Controllers/SubCategoryController.cs

What to change: - Every list/create/edit/delete must use active branch. - If Admin needs to manage multiple branches: - either switch branch while managing menu, or - provide branch dropdown on menu pages.

D) Stored procedures to update (from SQL scan)

D1) Orders

- `dbo.usp_CreateOrder` → add `@BranchId` and insert to `Orders.BranchId`
- `dbo.usp_AddOrderItem` → validate order branch; optionally set `OrderItems.BranchId`
- `dbo.usp_FireOrderItems` → filter/validate by branch
- `usp_GetRecentOrdersForDashboard` / dashboards → filter by branch

D2) Payments

- `dbo.usp_GetOrderPaymentInfo` → enforce that requested order belongs to branch
- `dbo.usp_ProcessPayment` → enforce/derive branch
- `dbo.usp_CreateSplitBill` → enforce/derive branch
- `dbo.usp_VoidPayment` → enforce/derive branch

D3) BOT

- `dbo.GetBOTDashboardStats`, `dbo.GetBOTsByStatus`, `dbo.GetBOTDetails` → filter by branch
- `dbo.GetNextBOTNumber` → decide if sequence is per-branch (recommended)
- `dbo.UpdateBOTStatus`, `dbo.VoidBOT` → validate branch
- `dbo.usp_GetBarBOTReport` → add `@BranchId` (nullable for admin) and filter.

D4) Kitchen

- `GetKitchenDashboardStats`, `GetKitchenTicketsByStatus`, `GetFilteredKitchenTickets`, `GetKitchenTicketDetails` → add `@BranchId`
- `UpdateKitchenTicketsForOrder` / `dbo.UpdateKitchenTicketsForOrder` → validate branch
- `MarkAllTicketsReady` → branch filter

D5) Reports (admin consolidated)

Add `@BranchId INT = NULL` to each report SP and use pattern: - `WHERE (@BranchId IS NULL OR o.BranchId = @BranchId)`

Procedures found in scan: - `usp_GetSalesReport` - `usp_GetOrderReport` - `usp_GetDiscountReport` - `dbo.usp_GetGSTBreakupReport` - `dbo.usp_GetCollectionRegister` - `usp_GetFinancialSummary` - `usp_GetCashClosingReport` - `usp_GetHomeDashboardStats` / `usp_GetHomeDashboardStatsEnhanced` (if totals should be branch-wise) - `dbo.usp_GetCustomerAnalysis` - `dbo.usp_GetMenuAnalysis`

D6) Menu + Pricing (branch-wise)

Procedures found in scan: - `dbo.sp_GetAllMenuItems` - `dbo.sp_GetMenuItemById` - `dbo.sp_CreateMenuItem` - `dbo.sp_UpdateMenuItem` - `dbo.sp_PublishMenuItem` - `dbo.sp_GetAllModifiers` - `dbo.sp_ApprovePriceChange` - plus recipe-related SPs (`dbo.sp_ManageRecipe`, etc.)

What to change: - Add `@BranchId` to all menu/proc read/write. - Ensure any uniqueness constraints consider branch (e.g., item codes/names).

D7) Tables/Reservations

- `dbo.usp_SeatGuests`, `dbo.usp_AssignServerToTable`, `dbo.usp_StartTableTurnover`, `dbo.usp_UpdateTableTurnoverStatus`
- `dbo.usp_UpsertReservation`, `dbo.usp_UpsertWaitlist`, `dbo.usp_UpsertTable`

Add branch enforcement and prevent cross-branch table assignment.

D8) Hotel room-service (leave as-is)

- `dbo.usp_GetRoomServicePendingSettlementDetails`
- `dbo.vw_RoomService_PendingSettlement`

These are hotel-specific and depend on `Orders.H_BranchID`.

E) Recommended implementation order (workable sequence)

1. DB: create `Branches`, `UserBranches`.
2. Add `BranchId` to menu tables first (because menu/pricing is branch-wise).
3. Add `BranchId` to `Orders` and enforce on create/add items.
4. Add branch enforcement to payments.
5. Add branch enforcement to BOT + Kitchen.
6. Reports: add `@BranchId` param (nullable) and enable admin consolidated.
7. Backfill existing rows to a default branch.
8. Make `BranchId` NOT NULL + FK constraints.

F) Next optional deliverable

If you want, I can generate a ready-to-run SQL migration skeleton that: - creates `Branches`, `UserBranches` - adds `BranchId` columns - adds indexes - shows safe backfill template