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