LINQ Queries Inventory

1) AdminService.cs

- Users.ToListAsync(): retrieves all users from Identity store.
- GetRolesAsync(user): fetches role list per user for building response objects.
- Purpose: Build user-with-roles listing and update user roles.

2) AuthService.cs

- No LINQ aggregation; uses Identity role retrieval and JWT generation.
Purpose: Authentication/authorization workflow (token issuance).

3) AvailabilityService.cs

- ServiceRequests.Where(... Assigned/InProgress ...).ToListAsync(): filters busy requests.
- Grouping not used; simple overlap check on filtered set.
- Purpose: Determine technicians busy for a given time window.

4) BillingService.cs

- Invoices.Include(i => i.ServiceRequest).FirstOrDefaultAsync(): load invoice with related request.
- Invoices.Where(...).ToListAsync(): fetch invoices filtered by user/role.
- SumAsync(i => i.Amount): revenue/payment calculations.
- Purpose: Invoice retrieval, payment processing, revenue sums.

5) CategoryService.cs

- ServiceCategories.OrderBy(c => c.Name).ToListAsync(): sorted category list.
- FindAsync / Where by Id: single category fetch/update/delete.
- Purpose: CRUD for service categories with ordering for display.

6) DashboardService.cs

- ServiceRequests.GroupBy(r => r.Status).Select(Count): status distribution.
- ServiceRequests.Where(r => r.TechnicianId != null).Include(Technician).GroupBy(...).Select(Count): technician workload.
- ServiceRequests.Include(Category).GroupBy(Category.Name).Select(Count): category distribution.
- Invoices.Where(Status == "Paid" && PaidAt != null).GroupBy(PaidAt month/year).Select(Sum): monthly revenue.
- ServiceRequests.Where(WorkStartedAt & WorkEndedAt).Average(duration): average resolution hours.
- Invoices.Where(Status == "Paid").Sum(Amount): total revenue.
- Purpose: Dashboard KPIs (counts, workload, revenue, averages).

7) ServiceRequestService.cs

- Multiple Where filters (status, role, technician/customer ownership).
- GroupBy(r => r.Status).Select(Count): status summary.
- Workload: Where(technician active) GroupBy(technician).Select(Count).
- CategoryCounts: GroupBy(category).Select(Count).
- SLA compliance and averages: Average, Count on filtered sets.
- Revenue: invoices.Where(PaidAt month/year).Sum(Amount); fallback Sum on requests.
- Revenue report: invoices.Where(PaidAt).GroupBy(month/year).Select(Sum).ToList / fallback GroupBy on requests with OrderByDescending.
- Pagination: OrderByDescending(CreatedAt).Skip().Take().ToListAsync(); CountAsync for totals.

Purpose: Core business logic—request lifecycle, assignments, dashboards, reporting, pagination.

8) TechnicianService.cs

- ServiceRequests.Include(Category).Where(TechnicianId == id).OrderByDescending(CreatedAt).ToListAsync(): fetch technician tasks.
- Workload stats: filtering completed requests, summing durations; Sum on invoices via Dictionary; LINQ projections to DTOs.
- Users.Where(customerIds.Contains).ToDictionaryAsync(): bulk customer lookup.
- Purpose: Technician workload, earnings, task history.

9) EmailBackgroundService.cs / INotificationQueue.cs

No LINQ queries; handles queuing and background email processing.

10) BillingService.cs

- Invoices.Include(i => i.ServiceRequest).FirstOrDefaultAsync(): load invoice with related request.
- Invoices.Where(...).ToListAsync(): fetch invoices filtered by user/role.
- SumAsync(i => i.Amount): revenue/payment calculations.
- Purpose: Invoice retrieval, payment processing, revenue sums.

Notes:

- All queries use EF Core LINQ with async execution (ToListAsync, SumAsync, CountAsync, Average).
- Overlap/availability logic uses LINQ filtering followed by in-memory checks for time conflicts.