Vehicle Service Management System – LINQ Usage Overview

1. Service Request Management

1.1. Filter Service Requests by Priority

File: `Application/Features/Reports/GetByPriority.cs`
Method: `GetByPriorityHandler.Handle`
Purpose: Filter service requests by priority (Normal/Urgent/High) for reporting.

Representative LINQ:

```
var result = _context.ServiceRequests
    .AsNoTracking()
    .AsEnumerable()
    .Where(s => !string.IsNullOrWhiteSpace(s.Priority) &&
                s.Priority.Equals(priority, StringComparison.OrdinalIgnoreCase))
    .OrderByDescending(s => s.RequestedDate)
    .Select(s => new ServiceRequestDto
    {
        Id = s.Id,
        VehicleModel = s.Vehicle != null ? s.Vehicle.Model : "Unknown",
        Description = s.Description ?? string.Empty,
        Status = s.Status ?? "Requested",
        Priority = s.Priority ?? "Normal",
        RequestedDate = s.RequestedDate
    })
    .ToList();
```

Used by: Service Manager Reports page – Priority-based filtering.

1.2. Get Customer's Service Requests (My Bookings)

File: `Application/Features/Services/GetMyBookings.cs`
Method: `GetMyBookingsHandler.Handle`
Purpose: Retrieve all service requests for a specific customer, ordered by date.

Representative LINQ:

```
return await _context.ServiceRequests
    .Include(s => s.Vehicle)
    .Include(s => s.Technician)
    .Where(s => s.Vehicle != null && s.Vehicle.UserId == request.UserId)
    .OrderByDescending(s => s.RequestedDate)
    .ThenByDescending(s => s.Id)
```

```
    .Select(s => new ServiceRequestDto
    {
        Id = s.Id,
        Description = s.Description,
        Status = s.Status,
        Priority = s.Priority,
        RequestedDate = s.RequestedDate,
        VehicleModel = s.Vehicle != null ? $"{s.Vehicle.Make} {s.Vehicle.Model}" :
"Unknown",
        TechnicianName = s.Technician != null ? s.Technician.FullName : null
    })
    .ToListAsync(cancellationToken);
```

Used by: Customer "My Bookings" page – View all their service requests.

1.3. Get Technician's Assigned Tasks

File: `Application/Features/Services/GetTechnicianTasks.cs`
Method: `GetTechnicianTasksHandler.Handle`
Purpose: Get active tasks assigned to a technician, prioritized by urgency.

Representative LINQ:

```
return await _context.ServiceRequests
    .Include(s => s.Vehicle)
    .Where(s => s.TechnicianId == request.TechnicianId &&
               (s.Status == "Assigned" || s.Status == "In Progress"))
    .OrderByDescending(s => s.Priority == "Urgent")
    .ThenByDescending(s => s.Priority == "High")
    .ThenByDescending(s => s.RequestedDate)
    .Select(s => new ServiceRequestDto
    {
        Id = s.Id,
        Description = s.Description ?? string.Empty,
        Status = s.Status ?? "Assigned",
        Priority = s.Priority ?? "Normal",
        RequestedDate = s.RequestedDate,
        VehicleModel = s.Vehicle != null ? $"{s.Vehicle.Make} {s.Vehicle.Model}" :
"Unknown Vehicle"
    })
    .ToListAsync(cancellationToken);
```

Used by: Technician Task List page – View assigned and in-progress tasks.

1.4. Get Technician Service History

File: `Application/Features/Services/GetTechnicianServiceHistory.cs`
Method: `GetTechnicianServiceHistoryHandler.Handle`

Purpose: Retrieve completed services for a technician, ordered by completion date.

Representative LINQ:

```
return await _context.ServiceRequests
    .Include(s => s.Vehicle)
    .Where(s => s.TechnicianId == request.TechnicianId &&
               (s.Status == "Completed" || s.Status == "Closed"))
    .OrderByDescending(s => s.CompletionDate ?? s.RequestedDate)
    .Select(s => new ServiceRequestDto
    {
        Id = s.Id,
        Description = s.Description ?? string.Empty,
        Status = s.Status ?? "Completed",
        Priority = s.Priority ?? "Normal",
        RequestedDate = s.RequestedDate,
        CompletionDate = s.CompletionDate,
        VehicleModel = s.Vehicle != null ? $"{s.Vehicle.Make} {s.Vehicle.Model}" :
"Unknown Vehicle"
    })
    .ToListAsync(cancellationToken);
```

Used by: Technician Dashboard – View completed service history.

---

2. Technician Workload & Assignment

2.1. Calculate Technician Workload

File: `Application/Features/Reports/GetTechnicianWorkload.cs`
Method: `GetTechnicianWorkloadHandler.Handle`
Purpose: Group active service requests by technician to calculate workload.

Representative LINQ:

```
var serviceRequests = await _context.ServiceRequests
    .Where(s => s.Status == "Assigned" || s.Status == "In Progress")
    .Include(s => s.Vehicle)
    .Include(s => s.Technician)
    .ToListAsync(ct);

var workload = serviceRequests
    .GroupBy(s => new {
        TechnicianId = s.TechnicianId ?? "Unassigned",
        TechnicianName = s.Technician != null ? s.Technician.FullName :
"Unassigned"
    })
    .Select(g => new TechnicianWorkloadDto
    {
        TechnicianId = g.Key.TechnicianId,
```

```
        TechnicianName = g.Key.TechnicianName,
        ActiveTasksCount = g.Count(),
        CurrentVehicleModels = g
            .Where(sr => sr.Vehicle != null)
            .Select(sr => sr.Vehicle!.Model)
            .Distinct()
            .DefaultIfEmpty("Unknown")
            .ToList()
    })
    .OrderByDescending(res => res.ActiveTasksCount)
    .ToList();
```

Used by: Service Manager Dashboard – Monitor technician workload distribution.

2.2. Technician Performance Report

File: `Application/Features/Reports/GetTechPerformance.cs`
Method: `GetTechPerformanceHandler.Handle`
Purpose: Group completed paid services by technician and calculate total revenue and job count.

Representative LINQ:

```
var performance = await _context.Invoices
    .Include(i => i.ServiceRequest)
        .ThenInclude(sr => sr!.Technician)
    .Where(i => i.PaymentStatus == "Paid"
        && i.ServiceRequest != null
        && i.ServiceRequest.TechnicianId != null
        && i.ServiceRequest.Technician != null
        && (i.ServiceRequest.Status == "Completed" || i.ServiceRequest.Status ==
"Closed"))
    .GroupBy(i => new {
        i.ServiceRequest!.TechnicianId,
        i.ServiceRequest.Technician!.FullName
    })
    .Select(g => new TechPerformanceDto
    {
        TechnicianName = g.Key.FullName,
        TotalRevenueGenerated = g.Sum(x => x.TotalAmount),
        JobsCompleted = g.Count()
    })
    .OrderByDescending(res => res.TotalRevenueGenerated)
    .ToListAsync(ct);
```

Used by: Service Manager Reports – Technician performance metrics.

---

3. Reports & Dashboards

3.1. Pending vs Completed Services

File: `Application/Features/Reports/GetPendingVsCompleted.cs`
Method: `GetPendingVsCompletedHandler.Handle`
Purpose: Count pending and completed services, with status breakdown.

Representative LINQ:

```
var pendingCount = await _context.ServiceRequests
    .Where(s => s.Status == "Requested" || s.Status == "Assigned" || s.Status ==
"In Progress")
    .CountAsync(ct);

var completedCount = await _context.ServiceRequests
    .Where(s => s.Status == "Completed" || s.Status == "Closed")
    .CountAsync(ct);

var statusBreakdown = await _context.ServiceRequests
    .GroupBy(s => s.Status)
    .Select(g => new StatusCountDto
    {
        Status = g.Key,
        Count = g.Count()
    })
    .ToListAsync(ct);
```

Used by: Service Manager Dashboard – Pending vs completed services summary.

3.2. Monthly Revenue Report (Aggregation)

File: `Application/Features/Reports/GetMonthlyStats.cs`
Method: `GetMonthlyStatsHandler.Handle`
Purpose: Group paid invoices by year/month and sum revenue.

Representative LINQ:

```
var stats = await _context.Invoices
    .Where(i => i.PaymentStatus == "Paid")
    .GroupBy(i => new { i.IssuedDate.Year, i.IssuedDate.Month })
    .Select(g => new MonthlyRevenueDto
    {
        Year = g.Key.Year,
        Month = g.Key.Month,
        Revenue = g.Sum(i => i.TotalAmount),
        ServiceCount = g.Count()
    })
    .OrderByDescending(res => res.Year)
    .ThenByDescending(res => res.Month)
    .ToListAsync(ct);
```

Used by: Service Manager Reports – Monthly revenue chart.

3.3. Revenue by Service Type (Grouping)

File: `Application/Features/Reports/GetServiceTypeRevenue.cs`
Method: `GetServiceTypeRevenueHandler.Handle`
Purpose: Group paid invoices by service category and calculate revenue per type.

Representative LINQ:

```
var report = await _context.Invoices
    .Include(i => i.ServiceRequest)
        .ThenInclude(sr => sr!.ServiceCategory)
    .Where(i => i.PaymentStatus == "Paid" &&
                i.ServiceRequest != null &&
                i.ServiceRequest.ServiceCategory != null)
    .GroupBy(i => i.ServiceRequest!.ServiceCategory!.Name)
    .Select(g => new ServiceTypeRevenueDto
    {
        ServiceTypeName = g.Key,
        TotalRevenue = g.Sum(i => i.TotalAmount),
        ServiceCount = g.Count()
    })
    .OrderByDescending(res => res.TotalRevenue)
    .ToListAsync(ct);
```

Used by: Service Manager Reports – Revenue breakdown by service category.

3.4. Service History per Vehicle (Grouping)

File: `Application/Features/Reports/GetVehicleServiceHistoryReport.cs`
Method: `GetVehicleServiceHistoryReportHandler.Handle`
Purpose: Group completed services by vehicle and calculate service statistics.

Representative LINQ:

```
var report = await _context.ServiceRequests
    .Include(sr => sr.Vehicle)
    .Where(sr => sr.Status == "Completed" || sr.Status == "Closed")
    .GroupBy(sr => new
    {
        sr.VehicleId,
        sr.Vehicle!.Make,
        sr.Vehicle.Model,
        sr.Vehicle.LicensePlate,
        sr.Vehicle.Year
```

```
    })
    .Select(g => new VehicleServiceHistoryReportDto
    {
        VehicleId = g.Key.VehicleId,
        Make = g.Key.Make ?? "Unknown",
        Model = g.Key.Model,
        LicensePlate = g.Key.LicensePlate,
        Year = g.Key.Year,
        TotalServices = g.Count(),
        LastServiceDate = g
            .OrderByDescending(sr => sr.CompletionDate ?? sr.RequestedDate)
            .Select(sr => sr.CompletionDate ?? sr.RequestedDate)
            .FirstOrDefault()
    })
    .OrderByDescending(v => v.LastServiceDate)
    .ToListAsync(ct);
```

Used by: Service Manager Reports – Vehicle service history summary.

3.5. Get Vehicle Service History (Detailed)

File: `Application/Features/Reports/GetVehicleServiceHistory.cs`
Method: `GetVehicleServiceHistoryHandler.Handle`
Purpose: Get detailed service history for a specific vehicle with role-based filtering.

Representative LINQ:

```
var baseQuery = _context.ServiceRequests
    .Where(s => s.VehicleId == request.VehicleId &&
               (s.Status == "Completed" || s.Status == "Closed"));

if (role == "customer")
{
    baseQuery = baseQuery
        .Where(s => _context.Vehicles
            .Any(v => v.Id == s.VehicleId && v.UserId == request.UserId));
}
else if (role == "technician")
{
    baseQuery = baseQuery.Where(s => s.TechnicianId == request.UserId);
}

var history = await baseQuery
    .Include(s => s.ServiceCategory)
    .Include(s => s.Vehicle)
    .Include(s => s.UsedParts)
        .ThenInclude(up => up.Part)
    .OrderByDescending(s => s.CompletionDate ?? s.RequestedDate)
    .Select(s => new VehicleHistoryDto
    {
```

```
        ServiceId = s.Id,
        ServiceName = s.ServiceCategory != null ? s.ServiceCategory.Name :
"General Service",
        CompletionDate = s.CompletionDate ?? s.RequestedDate,
        PartsReplaced = s.UsedParts.Select(up => up.Part != null ? up.Part.Name :
"Unknown Part").ToList(),
        TotalCost = _context.Invoices
            .Where(i => i.ServiceRequestId == s.Id)
            .Select(i => i.TotalAmount)
            .FirstOrDefault()
    })
    .ToListAsync(ct);
```

Used by: Vehicle Details page – Detailed service history with parts and costs.

3.6. Revenue Report Summary

File: `Application/Features/Admin/GetRevenueReport.cs`
Method: `GetRevenueReportHandler.Handle`
Purpose: Calculate total revenue, completed services count, and recent invoices.

Representative LINQ:

```
var totalRevenue = await _context.Invoices
    .Where(i => i.PaymentStatus == "Paid")
    .SumAsync(i => i.TotalAmount, cancellationToken);

var totalCompleted = await _context.Invoices
    .CountAsync(i => i.ServiceRequest != null &&
                    (i.ServiceRequest.Status == "Completed" ||
i.ServiceRequest.Status == "Closed"),
                    cancellationToken);

var recentInvoices = await _context.Invoices
    .Include(i => i.ServiceRequest)
        .ThenInclude(s => s!.Vehicle)
    .OrderByDescending(i => i.IssuedDate)
    .Take(10)
    .Select(i => new InvoiceSummaryDto
    {
        InvoiceId = i.Id,
        CustomerVehicle = i.ServiceRequest != null && i.ServiceRequest.Vehicle !=
null
            ? i.ServiceRequest.Vehicle.Make + " " + i.ServiceRequest.Vehicle.Model
            : "Unknown Vehicle",
        Amount = i.TotalAmount,
        Date = i.IssuedDate,
        Status = i.PaymentStatus
    })
    .ToListAsync(cancellationToken);
```

Used by: Admin Dashboard – Revenue summary and recent transactions.

---

4. Inventory Management

4.1. Low Stock Alert

File: `Application/Features/Inventory/InventoryFeatures.cs`
Method: `GetLowStockHandler.Handle`
Purpose: Filter parts with stock quantity below threshold.

Representative LINQ:

```
var parts = await _context.Parts
    .Where(p => p.StockQuantity < 5 && p.IsActive)
    .ToListAsync(ct);
```

Used by: Admin Dashboard – Low stock alerts.

4.2. Filter Parts by Role (Price Visibility)

File: `Application/Features/Inventory/InventoryFeatures.cs`
Method: `GetAllPartsHandler.Handle`
Purpose: Hide pricing from technicians while showing to others.

Representative LINQ:

```
var parts = await _context.Parts.ToListAsync(ct);
var isTechnician = request.UserRole?.Equals("Technician",
StringComparison.OrdinalIgnoreCase) == true;

return parts.Select(p => new PartDto
{
    Id = p.Id,
    Name = p.Name,
    UnitPrice = isTechnician ? null : p.UnitPrice,
    StockQuantity = p.StockQuantity,
    IsActive = p.IsActive
}).ToList();
```

Used by: Inventory Management page – Role-based price visibility.

---

5. Billing & Payments

5.1. Calculate Parts Cost for Invoice

File: `Application/Features/Admin/ApproveService.cs`
Method: `ApproveServiceHandler.Handle`
Purpose: Sum the cost of all parts used in a service.

Representative LINQ:

```
decimal partsCost = service.UsedParts
    .Sum(up => (up.Part?.UnitPrice ?? 0) * up.QuantityUsed);
```

Used by: Invoice generation – Calculate total parts cost.

5.2. Get Customer Invoices

File: `Application/Features/Invoices/GetMyInvoices.cs`
Method: `GetMyInvoicesHandler.Handle`
Purpose: Retrieve all invoices for a customer with parts details, ordered by date.

Representative LINQ:

```
var invoices = await _context.Invoices
    .Include(i => i.ServiceRequest)
        .ThenInclude(s => s!.ServiceCategory)
    .Include(i => i.ServiceRequest)
        .ThenInclude(s => s!.UsedParts)
            .ThenInclude(sp => sp.Part)
    .Where(i => i.ServiceRequest != null
        && i.ServiceRequest.CustomerId == request.UserId)
    .OrderByDescending(i => i.IssuedDate)
    .Select(i => new InvoiceDto
    {
        InvoiceId = i.Id,
        ServiceRequestId = i.ServiceRequestId,
        TotalAmount = i.TotalAmount,
        LabourFee = i.ServiceRequest!.ServiceCategory != null
            ? i.ServiceRequest.ServiceCategory.BasePrice : 0,
        IssuedDate = i.IssuedDate,
        PaymentStatus = i.PaymentStatus ?? "Pending",
        PartsUsed = i.ServiceRequest!.UsedParts.Select(sp => new
PartUsageDetailDto
        {
            PartName = sp.Part != null ? sp.Part.Name : "Unknown Part",
            Quantity = sp.QuantityUsed,
            PricePerUnit = sp.Part != null ? sp.Part.UnitPrice : 0
        }).ToList()
    })
    .ToListAsync(cancellationToken);
```

Used by: Customer Billing page – View invoices with parts breakdown and payment history.

Summary of LINQ Operations

Filtering Operations

- ☑ Filter by **Status** (Requested, Assigned, In Progress, Completed, Closed)
- ☑ Filter by **Priority** (Normal, High, Urgent)
- ☑ Filter by **Date** (OrderByDescending on RequestedDate, CompletionDate)
- ☑ Filter by **User/Customer** (Vehicle.UserId, CustomerId)
- ☑ Filter by **Technician** (TechnicianId)
- ☑ Filter by **Payment Status** (Paid, Pending)
- ☑ Filter by **Stock Quantity** (Low stock alerts: StockQuantity < 5)
- ☑ **Any()** – Check existence (vehicle ownership, role validation)

Grouping Operations

- ☑ Group by **Technician** (Workload, Performance)
- ☑ Group by **Service Category/Type** (Revenue by type)
- ☑ Group by **Vehicle** (Service history per vehicle)
- ☑ Group by **Month/Year** (Monthly revenue)
- ☑ Group by **Status** (Status breakdown)

Aggregation Operations

- ☑ **Count()** / **CountAsync()** – Service counts, task counts, invoice counts
- ☑ **Sum()** / **SumAsync()** – Total revenue, parts cost, outstanding amounts
- ☑ **FirstOrDefault()** / **FirstOrDefaultAsync()** – Latest service date, most recent invoice, find specific records
- ☑ **Take()** – Limit results (recent invoices, top N records)
- ☑ **Distinct()** – Remove duplicates (vehicle models, part names)
- ☑ **Any()** – Check existence conditions

Ordering Operations

- ☑ **OrderByDescending** – Most recent first (date, revenue)
- ☑ **ThenByDescending** – Secondary sorting (priority, ID)
- ☑ **OrderBy** – Alphabetical (technician name, vehicle model)

Total LINQ Operations Demonstrated: 18+ distinct query patterns across 5 feature areas

Key LINQ Techniques Used

- **Include() / ThenInclude()** – Eager loading related entities
- **AsNoTracking()** – Read-only queries for better performance
- **AsEnumerable()** – Client-side evaluation when needed
- **Conditional Filtering** – Dynamic query building based on role/user
- **Nested Queries** – Subqueries for related data (invoice totals, parts lists)
- **Null Coalescing** – Safe navigation with `??` operator