BusinessLogic.Extensions

```
using System;
using System.Collections.Generic;
using System.Linq;
using HMS_API.Models; // Reference your API's entity models

namespace MyApp.BusinessLogic.Extensions
{
   // Generic extension method for filtering any collection.
   public static class CollectionExtensions
   {
      public static List<T> FilterBy<T>(this IEnumerable<T> source, Func<T, bool>
predicate)
      {
         return source.Where(predicate).ToList();
      }
   }

   // Extension method for Appointment that returns a summary string.
   public static class AppointmentExtensions
   {
      public static string ToSummary(this Appointment appointment)
      {
         var patientName = appointment.Patient?.AppUser?.FullName ?? "Unknown
Patient";
         var doctorName = appointment.Doctor?.AppUser?.FullName ?? "Unknown
Doctor";
         return $"[ID: {appointment.EventId}] {patientName} with {doctorName} on
{appointment.AppointmentDate:d} - Status: {appointment.Status}";
      }
   }
}
```

ReportFormatter.cs

```
namespace MyApp.BusinessLogic.Utilities
{
   public sealed class ReportFormatter
   {
      public static string Format(string content)
      {
         return $"===== REPORT =====\n{content}\n===== END REPORT =====";
      }
   }
}
```

Manager1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using HMS_API.DB;
using HMS_API.Models;
using Microsoft.EntityFrameworkCore;

namespace MyApp.BusinessLogic
{
    // Business-level enum representing appointment statuses.
    public enum BusinessAppointmentStatus
    {
        Scheduled,
        Completed,
        Cancelled
    }

    // Delegate and event args for notifying when an appointment is processed.
    public delegate void AppointmentProcessedEventHandler(object sender,
AppointmentProcessedEventArgs e);
    public class AppointmentProcessedEventArgs : EventArgs
    {
        public int AppointmentId { get; }
        public BusinessAppointmentStatus Status { get; }
        public AppointmentProcessedEventArgs(int appointmentId, BusinessAppointmentStatus
status)
        {
            AppointmentId = appointmentId;
            Status = status;
        }
    }

    public partial class HospitalBusinessManager
    {
        private readonly AppDBContext _context;

        // Cache appointments grouped by doctor's full name.
        private Dictionary<string, List<Appointment>> _appointmentsByDoctor;
        // Keep track of processed appointment IDs to avoid duplicate processing.
        private HashSet<int> _processedAppointmentIds;

        // Event fired when an appointment is processed.
        public event AppointmentProcessedEventHandler? AppointmentProcessed;

        public HospitalBusinessManager(AppDBContext context)
        {
            _context = context;
            _appointmentsByDoctor = new Dictionary<string, List<Appointment>>();
            _processedAppointmentIds = new HashSet<int>();
        }

        /// <summary>
        /// Processes all pending appointments (those with Status = "Scheduled") by:
        /// - Fetching them from the database,
        /// - Simulating asynchronous processing,
        /// - Updating their status to "Completed",
        /// - Caching them,
        /// - Raising an event to notify subscribers.
        /// This method encapsulates the state change rather than allowing direct updates.
        /// </summary>
        public async Task ProcessPendingAppointmentsAsync()
        {
            try
            {
                // Use LINQ to load pending appointments from the database.
                var pendingAppointments = await _context.Appointments
                    .Where(a => a.Status == "Scheduled")
                    .ToListAsync();

                foreach (var appt in pendingAppointments)
                {
                    // Simulate processing work asynchronously.
                    await Task.Delay(100);

                    // Enforce encapsulation: change status only via this method.
                    appt.Status = "Completed";

                    // If not already processed, record and fire event.
                    if (_processedAppointmentIds.Add(appt.EventId))
                    {
                        AppointmentProcessed?.Invoke(this, new
AppointmentProcessedEventArgs(appt.EventId, BusinessAppointmentStatus.Completed));
                    }

                    // Cache appointment by doctor's name.
                    string doctorName = appt.Doctor?.AppUser?.FullName ?? "Unknown Doctor";
                    if (!_appointmentsByDoctor.ContainsKey(doctorName))
                        _appointmentsByDoctor[doctorName] = new List<Appointment>();
                    _appointmentsByDoctor[doctorName].Add(appt);
                }
                // Save changes to the database.
                await _context.SaveChangesAsync();
            }
            catch (Exception ex)
            {
                // In production, log the exception using a logging framework.
                throw new ApplicationException("Error processing appointments in the Business
Logic Layer.", ex);
```

```
            }
        }
    }
}
```

Manager2

```
using System;
using System.Linq;
using System.Threading.Tasks;
using MyApp.BusinessLogic.Extensions;
using MyApp.BusinessLogic.Utilities;
using Microsoft.EntityFrameworkCore;

namespace MyApp.BusinessLogic
{
    public partial class HospitalBusinessManager
    {
        /// <summary>
        /// Generates a daily report from the database for the specified date.
        /// Uses LINQ to query appointments, arrays and loops to build the report,
        /// and an extension method to format each appointment's summary.
        /// </summary>
        /// <param name="reportDate">The date for which the report is generated.</param>
        /// <returns>A formatted report string.</returns>
        public async Task<string> GenerateDailyReportAsync(DateTime reportDate)
        {
            try
            {
                var dailyAppointments = await _context.Appointments
                    .Where(a => a.AppointmentDate.Date == reportDate.Date)
                    .ToArrayAsync();

                // Build report lines using a loop over the array.
                string[] reportLines = new string[dailyAppointments.Length];
                for (int i = 0; i < dailyAppointments.Length; i++)
                {
                    reportLines[i] = dailyAppointments[i].ToSummary();
                }

                string reportContent = string.Join(Environment.NewLine, reportLines);
                return ReportFormatter.Format(reportContent);
            }
            catch (Exception ex)
            {
                throw new ApplicationException("Error generating daily report in the Business Logic
Layer.", ex);
            }
        }
        /// <summary>
        /// A generic method that filters an array of items using a given predicate.
        /// Demonstrates generics and extension method usage.
        /// </summary>
        public T[] FilterArray<T>(T[] items, Func<T, bool> predicate)
        {
            return items.FilterBy(predicate).ToArray();
        }

        /// <summary>
        /// Runs a CPU-bound operation in the background using a dedicated Thread.
        /// </summary>
        public void RunBackgroundOperation(Action action)
        {
            var thread = new System.Threading.Thread(new System.Threading.ThreadStart(action))
            {
                IsBackground = true
            };
            thread.Start();
        }
    }
}
```

Notif

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace MyApp.BusinessLogic.Services
{
    public sealed class NotificationService
    {
        public event EventHandler<string>? NotificationSent;

        /// <summary>
        /// Sends a notification asynchronously.
        /// In a real-world scenario, this could trigger email/SMS notifications.
        /// </summary>
        public async Task SendNotificationAsync(string message)
        {
            await Task.Run(() =>
            {
                // Simulate sending notification with delay.
                Thread.Sleep(200);
                OnNotificationSent(message);
            });
        }

        private void OnNotificationSent(string message)
        {
            NotificationSent?.Invoke(this, message);
        }
    }
}
builder.Services.AddScoped<HospitalBusinessManager>();
builder.Services.AddScoped<MyApp.BusinessLogic.Services.NotificationService>();
[ApiController]
[Route("api/[controller]")]
public class ReportController : ControllerBase
{
    private readonly HospitalBusinessManager _blManager;
    public ReportController(HospitalBusinessManager blManager)
    {
        _blManager = blManager;
    }

    [HttpGet("daily")]
    public async Task<IActionResult> GetDailyReport(DateTime? date)
    {
        var report = await _blManager.GenerateDailyReportAsync(date ?? DateTime.UtcNow);
        return Ok(new { Report = report });
    }
}

using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using HMS_API.DB;
using System.Windows.Forms;
using Microsoft.EntityFrameworkCore;

static class Program
{
    public static IServiceProvider ServiceProvider { get; private set; }
    [STAThread]
    static void Main()
    {
        Application.SetHighDpiMode(HighDpiMode.SystemAware);
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);

        var host = Host.CreateDefaultBuilder()
            .ConfigureServices((context, services) =>
            {
                services.AddDbContext<AppDBContext>(options =>
                    options.UseSqlServer("YourConnectionStringHere"));
                services.AddScoped<HospitalBusinessManager>();
                services.AddTransient<MainForm>(); // Your main WinForms form.
            })
            .Build();

        ServiceProvider = host.Services;
        Application.Run(ServiceProvider.GetRequiredService<MainForm>());
    }
}

public partial class MainForm : Form
{
    private readonly HospitalBusinessManager _blManager;
    public MainForm(HospitalBusinessManager blManager)
    {
        _blManager = blManager;
        InitializeComponent();
    }

    private async void btnGenerateReport_Click(object sender, EventArgs e)
    {
        try
        {
            string report = await _blManager.GenerateDailyReportAsync(DateTime.UtcNow);
            txtReport.Text = report; // Display report in a multi-line TextBox.
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Error: {ex.Message}");
        }
    }
}

using System;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;
using HMS_API.DB;

namespace MyApp.BusinessLogic.AzureFunctions
{
    public class AzureFunctionsIntegration
    {
        private readonly HospitalBusinessManager _manager;

        public AzureFunctionsIntegration(AppDBContext context)
        {
            _manager = new HospitalBusinessManager(context);
        }

        [FunctionName("DailyReportFunction")]
        public async Task Run(
            [TimerTrigger("0 0 8 * * *")] TimerInfo myTimer,
            ILogger log)
        {
            try
            {
                // Process pending appointments using live data.
                await _manager.ProcessPendingAppointmentsAsync();

                // Generate and log the daily report.
                string report = await _manager.GenerateDailyReportAsync(DateTime.UtcNow);
                log.LogInformation("Daily Report Generated:");
                log.LogInformation(report);
            }
            catch (Exception ex)
            {
                log.LogError(ex, "Error executing DailyReportFunction in the Business Logic Layer.");
            }
        }
    }
}
```