Prompt — Implement Next■Level, Per■Run Logging (ASP.NET Core + EF Core + SignalR + Vue)

Primary goals: (1) Reconstruct a full workflow run (UI → API → worker → DB → SignalR); (2) Identify optimization hotspots (repeats, slow queries, chatty SignalR, unused paths).

Note: You may deviate if you can better satisfy the goals, but explain why.

A) Design decisions (big■company style)

• Structured JSON logs + correlation envelope. • Domain events with metrics. • Per■run partitioned logs (narrative + optional lenses). • SSE/SignalR■safe middleware. • Tail/stream/download endpoints. • Redaction & sampling for bodies/SQL. • Optional OpenTelemetry.

B) What to log (content contract)

```
Envelope fields:
WorkflowRunId, TraceId, SpanId, RequestId, SessionId, Actor(ui|api|worker),
Component, Event, Level, Timestamp, DurationMs?, RowCount?, SizeBytes?, Count?

Event types:
Workflow.StepEntered/StepExited
Workflow.TransitionTriggered
Workflow.TethysStatusRun
Workflow.BddMatchBatch
Workflow.SignalRProgress
Api.RequestSummary
Db.CommandExecuted
Cache.Hit / Cache.Miss
Ui.Action
```

Log **shapes & sizes**, not full bodies. Sample heavy payloads/SQL ≤ 2% in debug.

C) File strategy (single vs multiple)

Global: logs/app/app.jsonl (rolling). **Per■run**: logs/workflows/{WorkflowRunId}/workflow.jsonl (+ summary.md, optional sampled-sql.jsonl). Endpoints: GET /api/logs/current/tail, GET /api/logs/current/stream, GET /api/logs/{runId}/download. Reset → new WorkflowRunId → new folder.

D) Implementation plan (code)

1) Correlation & propagation

```
// IWorkflowRunIdProvider + WorkflowRunIdProvider
public interface IWorkflowRunIdProvider { string GetCurrentRunId(); string NewRunId(); }
public class WorkflowRunIdProvider : IWorkflowRunIdProvider {
  private string _current = Guid.NewGuid().ToString("n");
  public string GetCurrentRunId() => _current;
  public string NewRunId() => _current = Guid.NewGuid().ToString("n");
}

// CorrelationMiddleware
public sealed class CorrelationMiddleware {
  private readonly RequestDelegate _next;
  public CorrelationMiddleware(RequestDelegate next) => _next = next;
  public async Task Invoke(HttpContext ctx, IWorkflowRunIdProvider run) {
    var runId = ctx.Request.Headers["X-Workflow-Run-Id"].FirstOrDefault() ?? run.GetCurrentRunId();
    var sessionId = ctx.Request.Headers["X-Session-Id"].FirstOrDefault() ?? ctx.TraceIdentifier;
    using (Serilog.Context.LogContext.PushProperty("WorkflowRunId", runId))
    using (Serilog.Context.LogContext.PushProperty("SessionId", sessionId))
    using (Serilog.Context.LogContext.PushProperty("Actor", "api")) {
      ctx.Response.Headers["X-Workflow-Run-Id"] = runId;
```

```
      await _next(ctx);
    }
  }
}

// src/api/http.ts
import axios from 'axios';
const http = axios.create();
http.interceptors.request.use(cfg => {
  const runId = localStorage.getItem('workflowRunId') || '';
  const s = sessionStorage.getItem('sessionId') || (sessionStorage.setItem('sessionId', crypto.randomUUI
  cfg.headers['X-Workflow-Run-Id'] = runId;
  cfg.headers['X-Session-Id'] = s!;
  return cfg;
});
export default http;

// src/signalr.ts
import * as signalR from '@microsoft/signalr';
export function connect(runId: string) {
  return new signalR.HubConnectionBuilder().withUrl(`/workflowHub?runId=${encodeURIComponent(runId)}`).w
}
```

2) Serilog sinks & enrichers

```
builder.Host.UseSerilog((ctx, cfg) =>
{
  var compact = new Serilog.Formatting.Compact.CompactJsonFormatter();
  cfg.MinimumLevel.Information()
    .MinimumLevel.Override("Microsoft", Serilog.Events.LogEventLevel.Warning)
    .Enrich.FromLogContext()
    .WriteTo.Async(a => a.File(compact, "logs/app/app.jsonl", rollingInterval: RollingInterval.Day, shar
    .WriteTo.Async(a => a.Map("WorkflowRunId","no-run",
      (runId, wt) => wt.File(compact, $"logs/workflows/{runId}/workflow.jsonl", rollingInterval: Rolling
});
app.UseSerilogRequestLogging();
```

3) SSE/SignalR■safe logging middleware

```
public class UnifiedLoggingMiddleware {
  private readonly RequestDelegate _next;
  private static readonly string[] ExcludeStartsWith = new[] { "/workflowHub" };
  public UnifiedLoggingMiddleware(RequestDelegate next) => _next = next;
  public async Task Invoke(HttpContext ctx) {
    if (ctx.WebSockets.IsWebSocketRequest) { await _next(ctx); return; }
    var accept = ctx.Request.Headers["Accept"].ToString();
    if (!string.IsNullOrEmpty(accept) && accept.Contains("text/event-stream")) { await _next(ctx); retur
    var path = ctx.Request.Path.Value ?? "";
    if (path.EndsWith("/negotiate") || ExcludeStartsWith.Any(p => path.StartsWith(p, StringComparison.Or
      await _next(ctx); return;
    }
    await _next(ctx);
  }
}
// Order:
app.UseSerilogRequestLogging();
app.UseMiddleware<CorrelationMiddleware>();
app.UseMiddleware<UnifiedLoggingMiddleware>();
app.UseWebSockets();
app.MapHub<WorkflowHub>("/workflowHub");
```

4) EF Core interceptor (durations, row counts, optional sampled SQL)

```
public sealed class DbCommandLoggingInterceptor : DbCommandInterceptor
{
  private static readonly Random _rng = new Random();
  public override async ValueTask<DbDataReader> ReaderExecutedAsync(
    DbCommand cmd, CommandExecutedEventData evt, DbDataReader res, CancellationToken ct = default)
  {
    var ms = evt.Duration.TotalMilliseconds;
    var sampled = _rng.NextDouble() < 0.02;
    Log.ForContext("Event","Db.CommandExecuted")
       .ForContext("DurationMs", ms)
       .ForContext("Database", cmd.Connection.Database)
       .ForContext("Command", evt.CommandSource.ToString())
       .ForContext("RowCount", res?.RecordsAffected >= 0 ? res.RecordsAffected : null)
       .ForContext("SampledSql", sampled ? cmd.CommandText : null)
       .Information("DB command executed");
    return await base.ReaderExecutedAsync(cmd, evt, res, ct);
  }
}
```

5) Typed domain events

```
public static class LogWorkflow
{
  public static void StepEntered(ILogger log, string step) =>
    log.ForContext("Event","Workflow.StepEntered").ForContext("Step", step).Information("Step entered: {
  public static void TransitionTriggered(ILogger log, string src, string trg, string dst) =>
    log.ForContext("Event","Workflow.TransitionTriggered").ForContext("Source", src).ForContext("Trigger
       .Information("State changed: {Source} --{Trigger}--> {Destination}", src, trg, dst);
  public static void SignalRProgress(ILogger log, string chan, int processed, int total) =>
    log.ForContext("Event","Workflow.SignalRProgress").ForContext("Channel", chan).ForContext("Processed
       .Information("SignalR progress {Processed}/{Total} on {Channel}", processed, total, chan);
}
```

6) Logs controller (tail, stream, download)

```
[ApiController]
[Route("api/logs")]
public class LogsController : ControllerBase
{
  private readonly IWorkflowRunIdProvider _run;
  public LogsController(IWorkflowRunIdProvider run) => _run = run;

  [HttpGet("current/tail")]
  public IActionResult Tail([FromQuery]int lines = 500) {
    var path = Path.Combine("logs","workflows", _run.GetCurrentRunId(), "workflow.jsonl");
    if (!System.IO.File.Exists(path)) return NotFound();
    var all = System.IO.File.ReadAllLines(path);
    var take = all.Length <= lines ? all : all[^lines..];
    return Content(string.Join('\n', take), "text/plain");
  }

  [HttpGet("current/stream")]
  public async Task Stream() {
    Response.Headers.Add("Content-Type", "text/event-stream");
    var path = Path.Combine("logs","workflows", _run.GetCurrentRunId(), "workflow.jsonl");
    Directory.CreateDirectory(Path.GetDirectoryName(path)!);
    using var fs = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Read, FileShare.ReadWrite);
    using var reader = new StreamReader(fs);
    while (!HttpContext.RequestAborted.IsCancellationRequested) {
```

```
      var line = await reader.ReadLineAsync();
      if (line is null) { await Task.Delay(300, HttpContext.RequestAborted).ContinueWith(_=>{}); continu
      await Response.WriteAsync($"data: {line}\n\n");
      await Response.Body.FlushAsync();
    }
  }

  [HttpGet("{runId}/download")]
  public IActionResult Download(string runId) {
    var folder = Path.Combine("logs","workflows", runId);
    if (!Directory.Exists(folder)) return NotFound();
    var zipPath = Path.Combine(Path.GetTempPath(), $"logs-{runId}.zip");
    if (System.IO.File.Exists(zipPath)) System.IO.File.Delete(zipPath);
    System.IO.Compression.ZipFile.CreateFromDirectory(folder, zipPath);
    return PhysicalFile(zipPath, "application/zip", Path.GetFileName(zipPath));
  }
}
```

7) Summary generator & Security (redaction/sampling) — see companion doc for details.

Acceptance: new run ⇒ new folder; SSE live tail works; typed events present with durations/counters; download bundle works; no PII; sampled SQL only in debug.