Refined Logging Design — Narrative (workflow.jsonl) + Fan■out Lenses

Objective: Make a single per■run file that teaches a new engineer (or GPT■5) the system by reading it top■to■bottom, while also providing focused files for API/DB/SignalR deep■dives.

1) What goes into workflow.jsonl (curated)

• Envelope: WorkflowRunId, TraceId/SpanId, RequestId, SessionId, Actor, Component, Event, Level, Timestamp, DurationMs?, RowCount?, SizeBytes?, Count?.

• Always include: Workflow.StepEntered/Exited, Workflow.TransitionTriggered, Workflow.TethysStatusRun, Workflow.BddMatchBatch, Workflow.SignalRProgress, Api.RequestSummary, Db.CommandExecuted, Cache.Hit/Miss, Exceptions (minimal context).

• Payloads: shapes/sizes instead of full bodies; SQL text only via debug sampling ≤ 2%.

Example event

```
{
  "Timestamp":"2025-09-13T09:15:42.183Z",
  "WorkflowRunId":"9f9e6f3a5a934a3b9ccb3e2c1f0d1a9f",
  "TraceId":"4eaa02b1f2b3f6c9a5c0d1e2f3a4b5c6",
  "SessionId":"b2c1-78fe",
  "Actor":"worker",
  "Component":"BddMatchService",
  "Event":"Workflow.BddMatchBatch",
  "RowsIn":1000,
  "RowsMatched":843,
  "DurationMs":86.7,
  "Level":"Information",
  "Message":"BDD matched 843/1000 in 86.7 ms"
}
```

2) Fan■out files (per■run lenses)

• workflow.errors.jsonl — only Warning/Error/Fatal • workflow.api.jsonl — only Api.* • workflow.db.jsonl — only Db.* (with sampled SQL if enabled) • workflow.signalr.jsonl — only Workflow.SignalR*

Serilog fan■out config

```
builder.Host.UseSerilog((ctx, cfg) =>
{
  var compact = new Serilog.Formatting.Compact.CompactJsonFormatter();
  cfg.MinimumLevel.Information()
    .MinimumLevel.Override("Microsoft", Serilog.Events.LogEventLevel.Warning)
    .Enrich.FromLogContext()

    // Global
    .WriteTo.Async(a => a.File(compact, "logs/app/app.jsonl", rollingInterval: RollingInterval.Day, shar

    // Per-run: narrative
    .WriteTo.Async(a => a.Map("WorkflowRunId","no-run",
        (runId, wt) => wt.File(compact, $"logs/workflows/{runId}/workflow.jsonl", rollingInterval: Rolli

    // Errors
    .WriteTo.Logger(lc => lc
        .Filter.ByIncludingOnly("@l in ['Warning','Error','Fatal']")
        .WriteTo.Map("WorkflowRunId","no-run",
            (runId, wt) => wt.File(compact, $"logs/workflows/{runId}/workflow.errors.jsonl", rollingInte
```

```
    // API
    .WriteTo.Logger(lc => lc
        .Filter.ByIncludingOnly("Event like 'Api.%'")
        .WriteTo.Map("WorkflowRunId","no-run",
            (runId, wt) => wt.File(compact, $"logs/workflows/{runId}/workflow.api.jsonl", rollingInterva

    // DB
    .WriteTo.Logger(lc => lc
        .Filter.ByIncludingOnly("Event like 'Db.%'")
        .WriteTo.Map("WorkflowRunId","no-run",
            (runId, wt) => wt.File(compact, $"logs/workflows/{runId}/workflow.db.jsonl", rollingInterval

    // SignalR
    .WriteTo.Logger(lc => lc
        .Filter.ByIncludingOnly("Event like 'Workflow.SignalR%'")
        .WriteTo.Map("WorkflowRunId","no-run",
            (runId, wt) => wt.File(compact, $"logs/workflows/{runId}/workflow.signalr.jsonl", rollingInt
});
```

3) Documentation artifacts per run

• summary.md — timeline, top endpoints, top DB, N+1, SignalR chatter, warnings/errors by TraceId, plus a Mermaid state diagram generated from Transition events. • legend.md — dictionary of Event types and key fields (one-time or per-run copy).

4) SSE/SignalR safety (do not regress)

UnifiedLoggingMiddleware must skip WebSockets, SSE, and negotiate/hub paths. Never buffer response bodies for streaming endpoints.

5) Where to find & download

```
logs/workflows/{WorkflowRunId}/
  ├─ workflow.jsonl
  ├─ workflow.errors.jsonl
  ├─ workflow.api.jsonl
  ├─ workflow.db.jsonl
  ├─ workflow.signalr.jsonl
  ├─ summary.md
  └─ legend.md

GET /api/logs/current/stream
GET /api/logs/current/tail?lines=500
GET /api/logs/{runId}/download
```

6) To■do for UI/Copilot

Implement correlation + per■run sinks, domain events, EF interceptor, filtered lenses, LogsController, summary generator, legend.md; verify SSE/SignalR handshake unaffected; add admin page tail/download.