Workflow Studio — Mini Resumen (Índice técnico)

Actualizado: 11-Nov-2025 (AR)

---

1) Esquema de base de datos (actual)

WF_Definicion - `Id` (int, PK, identity) - `Codigo` (nvarchar(50)) - `Nombre` (nvarchar(200)) - `Version` (int) - `Activo` (bit) - `FechaCreacion` (datetime) - `CreadoPor` (nvarchar(100)) - `JsonDef` (nvarchar(max))

WF_Instancia - `Id` (bigint, PK, identity) - `WF_DefinicionId` (int, FK → WF_Definicion.Id) - `Estado` (nvarchar(50)) — *EnCurso, Finalizado, Error, etc.* - `FechaInicio` (datetime) - `FechaFin` (datetime, null) - `DatosEntrada` (nvarchar(max)) - `DatosContexto` (nvarchar(max)) - `CreadoPor` (nvarchar(100))

WF_InstanciaLog - `Id` (bigint, PK, identity) - `WF_InstanciaId` (bigint, FK → WF_Instancia.Id) - `FechaLog` (datetime) - `Nivel` (nvarchar(20)) — *Info, Error, Debug* - `Mensaje` (nvarchar(max)) - `NodoId` (nvarchar(50), null) - `NodoTipo` (nvarchar(100), null)

---

2) Páginas WebForms (UI)

WorkflowUI.aspx (editor) - Canvas con nodos/aristas, inspector de propiedades, *Export JSON*, *Export C#*, *Clear*. - Botón **Guardar en SQL** → inserta en *WF_Definicion* (usa `__doPostBack('WF_SAVE','')`). - Panel “Ejecutar en servidor”: *solo pruebas* (no negocio).

WF_Definiciones.aspx - Grilla de definiciones activas (filtrar por código). - Acciones: **JSON** (ver `JsonDef`), **Instancias** (navega a WF_Instancias).

WF_Instancias.aspx - Selección de definición y grilla de instancias. - Acciones por instancia: **Datos** (entrada + contexto), **Log**, **Reejecutar**.

Poliza_Nueva.aspx (negocio) - Form simple (NroPoliza, Asegurado, etc.). - Ejecuta **WorkflowRuntime.CrearInstanciaYEjecutarAsync(defId, jsonEntrada, usuario)**. - Mensaje con Id de instancia y link a *WF_Instancias*.

---

3) Runtime / Motor

Archivo: `App_Code/MotorFlujoMinimo.cs` - *Contracts*: `NodeDef`, `EdgeDef`, `WorkflowDef`, `ResultadoEjecucion`, `IManejadorNodo`. - *Infra*: `ContextoEjecucion` (incluye `Estado:Dictionary`, `Log:Action<string>`, `Http:HttpClient` con `BaseAddress`). - *Motor*: `MotorFlujo` (valida y recorre; selecciona arista por `Etiqueta`). - *Handlers básicos*: `HStart (util.start)`, `HEnd (util.end)`, `HLogger (util.logger)`, `HIf (control.if)`,

`HHttpRequest (http.request)`, `HDocEntrada (doc.entrada)`. - *Helper*: `MotorDemo.CrearHandlersPorDefecto()` → `List<IManejadorNodo>`.

Carpeta Runtime (`Intranet.WorkflowStudio.Runtime`) - `WorkflowRuntime`
- `CrearInstanciaYEjecutarAsync(defId, datosEntradaJson, usuario)` → crea en *WF_Instancia*, ejecuta motor, persiste logs en *WF_InstanciaLog*, cierra instancia con `DatosContexto`. - `ReejecutarInstanciaAsync(instId, usuario)` → lee definición previa y crea nueva instancia. - `ManejadorSql (data.sql)`
- Parámetros: `connectionStringName` **o** `connection` (directa), `commandText` / `query`, `parameters` (objeto).
- Ejecuta `ExecuteNonQueryAsync`; guarda `ctx.Estado["sql.rows"]` y loguea filas.

---

4) Convenciones del JSON

```
{
  "StartNodeId": "nStart",
  "Nodes": {
    "nStart": { "Id":"nStart", "Type":"util.start", "Parameters":{} },
    "nSql":   { "Id":"nSql",   "Type":"data.sql",   "Parameters":{ /* ...
*/ } },
    "nIf":    { "Id":"nIf",    "Type":"control.if", "Parameters":
{"expression":"${sql.rows} == 1"} },
    "nLogOk": { "Id":"nLogOk", "Type":"util.logger","Parameters":
{"level":"Info","message":"SQL OK"} },
    "nLogErr":{ "Id":"nLogErr","Type":"util.logger","Parameters":
{"level":"Error","message":"SQL sin filas"} },
    "nEnd":   { "Id":"nEnd",   "Type":"util.end",   "Parameters":{} }
  },
  "Edges": [
    {"From":"nStart","To":"nSql","Condition":"always"},
    {"From":"nSql","To":"nIf","Condition":"always"},
    {"From":"nIf","To":"nLogOk","Condition":"true"},
    {"From":"nIf","To":"nLogErr","Condition":"false"},
    {"From":"nLogOk","To":"nEnd","Condition":"always"},
    {"From":"nLogErr","To":"nEnd","Condition":"always"}
  ]
}
```

Etiquetas de salida: `always`, `true`, `false`, `error` (según handler).
IF evalúa expresiones simples; p.ej. `${payload.status} == 200`, `${sql.rows} == 1`.

Params `data.sql` (ejemplo):

```
{
  "connectionStringName": "DefaultConnection",
  "commandText": "UPDATE Polizas SET Asegurado=@Asegurado WHERE
NroPoliza=@NroPoliza",
```

```
  "parameters": { "Asegurado": "${input.Asegurado}", "NroPoliza": "$
{input.NroPoliza}" }
}
```

El motor resuelve `${input.*}` desde `DatosEntrada` y expone `sql.rows` para `IF`.

---

5) Flujo de prueba recomendado (end-to-end)

1) **Editor (WorkflowUI.aspx)** - Nodos: `Start → data.sql → if → (logger OK|logger Error) → End`. - `data.sql` con `connectionStringName=DefaultConnection`, `commandText` y `parameters`. - `control.if` con `expression = "${sql.rows} == 1"`. - **Export JSON** y **Guardar en SQL** (crea *WF_Definicion*). 2) **Negocio (Poliza_Nueva.aspx)** - Completar datos y elegir la definición guardada. - Enviar → se crea *WF_Instancia* y se ejecuta. 3) **Monitoreo (WF_Instancias.aspx)** - Ver **Datos** (entrada + contexto) y **Log**. - **Reejecutar** para retry o regresión.

---

6) Configuración & notas

- `web.config` → `<connectionStrings><add name="DefaultConnection" ... /></connectionStrings>`.
- Páginas que usan tareas: `Async="true"` en la directiva Page.
- Mantener **Bootstrap local** (sin CDN) por intranet.
- `Newtonsoft.Json` referenciado en WebForms y Runtime.
- Logs de ejecución persistidos en *WF_InstanciaLog* por `WorkflowRuntime`.

---

7) Próximos pasos sugeridos

- **Persistir posiciones** del canvas en `JsonDef` para respetar layout.
- **Aristas con flecha** y selección estable (UI polida del editor).
- **Más handlers**: email real, chat/Slack, redis, file I/O, subflujo.
- **Secrets/config** centralizados (no embebidos en JSON).
- **Validaciones** previas a guardar (1 Start, ≥1 End, aristas válidas, params JSON válido).
- **Pruebas**: botón de *Run Tests* (ya en editor React) + tests de motor.