REDMIND — MVP1 (Frontend)

Semana 1: Editor base de flujos (React Flow) + contratos + export/import

Fecha: 2025-12-30 | Alcance: Frontend (dev 1) — Entregable semanal

1. Objetivo de la semana

Construir el **Editor mínimo** para definir y persistir grafos de agentes. El editor debe permitir: **(a)** crear nodos/edges, **(b)** editar configuración por nodo, **(c)** exportar/importar JSON, **(d)** validar contra contratos compartidos, y **(e)** aplicar auto-layout (botón). No se requiere ejecución real del flujo aún (eso vive en backend/kernel), pero sí una integración inicial con *Validate API*.

Definition of Done (Semana 1)

• Editor con canvas (React Flow) + palette de nodos + panel de propiedades.
• Export JSON / Import JSON funcionando con IDs estables.
• Validación local (Zod) + validación remota (POST /graphs/validate).
• Auto-layout on demand (ELK) para grafos pequeños/medianos.
• 2 grafos de ejemplo importables: “Hello Agent” y “Route Intent”.

2. Dependencias y herramientas (Frontend)

Librería	Uso
react, react-dom	UI base
reactflow	Canvas de nodos/edges (editor)
elkjs	Auto-layout (organizar nodos automáticamente)
zod	Validación runtime de contratos (configs + graph)
zustand	Store simple para estado del editor (nodos, edges, selección, UI)
uuid o ulid	IDs estables para nodes/edges (recomendado ULID para orden temporal)
@tanstack/react-query (opcional)	Fetch y caching para /validate, /runs (futuro)
react-hook-form (opcional)	Forms de propiedades por nodo (mejor DX)
monaco-editor (opcional)	Editor JSON/expresiones con highlight (puede ir en Semana 2)

Nota: Mantener el front sin dependencias pesadas de “agent frameworks”. El editor produce JSON estable; la ejecución vive en backend/Temporal.

3. Estructura de proyecto (recomendada)

Se recomienda monorepo (pnpm workspaces) para compartir contratos entre front y back sin duplicación.

Estructura mínima

```
apps/
editor/ (React + React Flow)
src/
app/
components/
nodes/
services/
state/
styles/
api/ (Backend - fuera del alcance semana 1)
packages/
contracts/ (Tipos + Zod Schemas compartidos)
kernel/ (Runner - fuera del alcance semana 1)
```

Regla de oro: el frontend importa los contratos desde *packages/contracts* y no inventa campos nuevos. Si falta algo, se agrega al contrato y se versiona.

4. Contratos (contracts) que el Front debe respetar

El editor debe trabajar con un contrato estable. Abajo se muestra una versión mínima para Semana 1. En código, esto vive en *packages/contracts* como tipos TS + Zod schemas.

4.1 Tipos base

```
export type NodeId = string;
export type EdgeId = string;

export type NodeType =
| "trigger.webhook"
| "trigger.manual"
| "agent.core"
| "condition.expr"
| "memory.kv"
| "model.llm"
| "tool.http"
| "tool.postgres"
| "response.chat";

export interface BaseNode {
id: NodeId;
type: NodeType;
typeVersion: number; // default 1
label?: string;
config: TConfig;
ui?: { x: number; y: number; w?: number; h?: number }; // posición opcional persistible
}

export interface Edge {
id: EdgeId;
source: NodeId;
target: NodeId;
label?: string;
}

export interface GraphDefinition {
id: string;
version: number; // contract version
start: NodeId;
nodes: BaseNode[];
edges: Edge[];
}
```

4.2 Convenciones importantes

• IDs estables: usar ULID/UUID al crear nodos/edges; no regenerar al guardar.
• typeVersion: iniciar en 1; si cambian campos de config, incrementar typeVersion.
• ui.x/ui.y: el editor puede persistir posiciones; si no existen, aplicar auto-layout (ELK).
• Graph.start: nodo inicial (por defecto el primer trigger creado).

5. Nodos del MVP (Semana 1) — UI + Config

En Semana 1 se recomienda implementar UI completa para los nodos base del MVP. El front debe renderizar cada tipo y su panel de propiedades.

Tipo	Propósito	Config mínima (campos)
trigger.manual	Entrada para pruebas locales	message: string
trigger.webhook	Entrada HTTP (solo definición)	path: string; method: 'POST'|'GET'
agent.core	Decisión semántica (intent/params)	strategy: 'reactive'; instructions: string
condition.expr	Ruteo por expresión	rules: Array<{ if: string; to: nodeId }>; engine: 'jexl'|'jmespath'
memory.kv	Cargar/guardar contexto	scope: 'conversation'|'run'; backend: 'postgres' (placeholder)
model.llm	LLM provider (placeholder)	provider: 'azure'|'openai'|'local'; model: string; temperature?: number
tool.http	Tool HTTP (placeholder)	method, url, headers?, body?
tool.postgres	Tool DB read-only (placeholder)	connectionRef, query (read-only)
response.chat	Salida al usuario	format: 'text'|'json'; template?: string

5.1 Reglas UI por nodo (Semana 1)

• Cada nodo muestra: icono/tipo + label editable (opcional) + badges de estado (futuro).
• Panel de propiedades: formulario simple; al guardar, valida con Zod y actualiza el store.
• Campos sensibles (headers/tokens) deben marcarse como secretos (no mostrar en export si se elige “redact”).

6. Editor (React Flow) — componentes mínimos

Componentes sugeridos

Componente	Responsabilidad
<EditorCanvas/>	React Flow wrapper; maneja nodes/edges + onConnect + selection
<NodePalette/>	Lista de nodos disponibles para drag/drop o click-add
<PropertiesPanel/>	Form dinámico según node.type; valida con Zod
<Toolbar/>	Export, Import, Validate, Organizar (ELK)
<JsonModal/>	Ver/pegar JSON; importar/exportar
<Toast/Notifications/>	Errores de validación y feedback

Store (Zustand) — shape recomendada

```
type EditorState = {
graph: GraphDefinition;
selectedNodeId?: string;
setGraph: (g: GraphDefinition) => void;
upsertNode: (node: BaseNode) => void;
updateNodeConfig: (id: string, patch: unknown) => void;
addEdge: (edge: Edge) => void;
removeNode: (id: string) => void;
validateLocal: () => { ok: boolean; errors: string[] };
};
```

Integración con API: crear un servicio *validateGraph()* que haga POST /graphs/validate. En Semana 1 basta con mostrar errores y resaltar nodos involucrados si aplica.

7. Auto-layout (ELK) — uso recomendado

Implementar un botón “Organizar” que calcule posiciones. El layout solo afecta ui.x/ui.y; no modifica semántica (nodes/edges).

```
import ELK from "elkjs/lib/elk.bundled.js";

const elk = new ELK();

async function layoutGraph(nodes, edges) {
const elkGraph = {
id: "root",
layoutOptions: {
"elk.algorithm": "layered",
"elk.direction": "RIGHT",
"elk.spacing.nodeNode": "40",
"elk.layered.spacing.nodeNodeBetweenLayers": "60"
},
children: nodes.map(n => ({ id: n.id, width: 220, height: 60 })),
edges: edges.map(e => ({ id: e.id, sources: [e.source], targets: [e.target] }))
};

const res = await elk.layout(elkGraph);

return nodes.map(n => {
const ln = res.children.find(x => x.id === n.id);
return { ...n, ui: { ...(n.ui ?? {}), x: ln.x, y: ln.y, w: ln.width, h: ln.height } };
});
}
```

Tip: aplicar auto-layout automáticamente al importar un grafo sin posiciones (*ui* ausente).

8. Ejemplos de grafos (importables en Semana 1)

8.1 Hello Agent

```
{
"id": "hello-agent",
"version": 1,
"start": "t1",
"nodes": [
{ "id": "t1", "type": "trigger.manual", "typeVersion": 1, "config": { "message": "Hola" }
},
{ "id": "a1", "type": "agent.core", "typeVersion": 1, "config": { "strategy": "reactive",
"instructions": "Return intent=chat" } },
{ "id": "r1", "type": "response.chat", "typeVersion": 1, "config": { "format": "text",
"template": "Hola, soy REDMIND." } }
],
"edges": [
{ "id": "e1", "source": "t1", "target": "a1" },
{ "id": "e2", "source": "a1", "target": "r1" }
]
}
```

8.2 Route Intent

```
{
"id": "route-intent",
"version": 1,
"start": "t1",
"nodes": [
{ "id": "t1", "type": "trigger.manual", "typeVersion": 1, "config": { "message": "reporte
ventas" } },
{ "id": "a1", "type": "agent.core", "typeVersion": 1, "config": { "strategy": "reactive",
"instructions": "If message includes 'reporte' return intent=report else intent=chat" } },
{ "id": "c1", "type": "condition.expr", "typeVersion": 1, "config": { "engine": "jexl",
"rules": [
{ "if": "intent == 'report'", "to": "r_report" },
{ "if": "intent == 'chat'", "to": "r_chat" }
] } },
{ "id": "r_report", "type": "response.chat", "typeVersion": 1, "config": { "format":
"text", "template": "OK, generaré el reporte." } },
{ "id": "r_chat", "type": "response.chat", "typeVersion": 1, "config": { "format": "text",
"template": "OK, conversemos." } }
],
"edges": [
{ "id": "e1", "source": "t1", "target": "a1" },
{ "id": "e2", "source": "a1", "target": "c1" },
{ "id": "e3", "source": "c1", "target": "r_report" },
{ "id": "e4", "source": "c1", "target": "r_chat" }
]
}
```

9. Checklist operativo (Semana 1)

■ Crear nodos desde palette y conectarlos con edges.
■ Editar configs desde PropertiesPanel (forms + Zod).
■ Exportar JSON y re-importarlo sin perder IDs.
■ Validar local + remoto; mostrar lista de errores.
■ Auto-layout con ELK (botón).
■ Guardar 2 ejemplos importables en /examples.

Notas finales

Semana 1 prioriza **contratos** y **editor**. La ejecución real del agente se integra en Semana 2 (Temporal/Kernel). Mantener el front agnóstico de la ejecución: solo produce y valida grafos.