REDMIND — Contracts Bridge (JSON Schema v1)

Contrato único para Front (TS) + Back (Python) trabajando en paralelo sin romperse

Fecha: 2025-12-30

1. Objetivo

Permitir que frontend y backend se desarrollen en repos separados sin divergencias. La fuente de verdad es un **JSON Schema v1** versionado (repo *redmind-contracts*). Ambos lados deben validar contra el mismo schema y aplicar reglas de integridad adicionales.

Reglas de oro

• El contrato (schema) es el puente: front/back NO inventan campos fuera del schema.
• Cambios breaking → subir versión mayor del contrato (v2) o incrementar Node.typeVersion.
• IDs estables: no regenerar ids al importar/exportar.
• La semántica del flujo NO depende de ui.x/ui.y (solo presentación).

2. Repo recomendado: redmind-contracts

Estructura mínima

```
redmind-contracts/
schemas/
graph.schema.json
examples/
hello-agent.json
route-intent.json
CHANGELOG.md
README.md
```

Versionado

• Taggear releases: **v1.0.0**, **v1.0.1**…
• Cambios no breaking (agregar campos opcionales) → patch/minor.
• Cambios breaking (renombrar/eliminar, cambiar required) → major (v2).

3. JSON Schema v1 (graph.schema.json)

Este schema define el formato del grafo (GraphDefinition) y los nodos del MVP. La validación de integridad (IDs únicos, start existe, edges apuntan a nodos) se hace fuera del schema.

```
{
"$schema": "https://json-schema.org/draft/2020-12/schema",
"$id": "https://redmind.local/schemas/graph.schema.json",
"title": "REDMIND GraphDefinition v1",
"type": "object",
"additionalProperties": false,
"required": ["id", "version", "start", "nodes", "edges"],
"properties": {
"id": { "type": "string", "minLength": 1 },
"version": { "type": "integer", "minimum": 1 },
"start": { "type": "string", "minLength": 1 },
"nodes": { "type": "array", "minItems": 1, "items": { "$ref": "#/$defs/Node" } },
"edges": { "type": "array", "items": { "$ref": "#/$defs/Edge" } }
},
"$defs": {
"UI": {
"type": "object",
"additionalProperties": false,
"required": ["x", "y"],
"properties": { "x": { "type": "number" }, "y": { "type": "number" }, "w": { "type": "number"
}, "h": { "type": "number" } }
},
"Edge": {
"type": "object",
"additionalProperties": false,
"required": ["id", "source", "target"],
"properties": {
"id": { "type": "string", "minLength": 1 },
"source": { "type": "string", "minLength": 1 },
"target": { "type": "string", "minLength": 1 },
"label": { "type": "string" }
}
},
"NodeBase": {
"type": "object",
"additionalProperties": false,
"required": ["id", "type", "typeVersion", "config"],
"properties": {
"id": { "type": "string", "minLength": 1 },
"typeVersion": { "type": "integer", "minimum": 1, "default": 1 },
"label": { "type": "string" },
"ui": { "$ref": "#/$defs/UI" },
"config": { "type": "object" }
}
},
"Node": {
"oneOf": [
{ "$ref": "#/$defs/TriggerManualNode" },
{ "$ref": "#/$defs/TriggerWebhookNode" },
{ "$ref": "#/$defs/AgentCoreNode" },
{ "$ref": "#/$defs/ConditionExprNode" },
{ "$ref": "#/$defs/MemoryKVNode" },
{ "$ref": "#/$defs/ModelLLMNode" },
{ "$ref": "#/$defs/ToolHttpNode" },
{ "$ref": "#/$defs/ToolPostgresNode" },
{ "$ref": "#/$defs/ResponseChatNode" }
]
```

```
},

"TriggerManualNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
"properties": {
"type": { "const": "trigger.manual" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["message"],
"properties": { "message": { "type": "string" } }
}
}
}
]
},

"TriggerWebhookNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
"properties": {
"type": { "const": "trigger.webhook" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["path", "method"],
"properties": {
"path": { "type": "string", "pattern": "^/" },
"method": { "type": "string", "enum": ["GET", "POST", "PUT", "PATCH", "DELETE"] }
}
}
}
}
]
},

"AgentCoreNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
"properties": {
"type": { "const": "agent.core" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["strategy", "instructions"],
"properties": {
"strategy": { "type": "string", "enum": ["reactive"] },
"instructions": { "type": "string" }
}
}
}
}
]
```

```
},

"ConditionExprNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
"properties": {
"type": { "const": "condition.expr" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["engine", "rules"],
"properties": {
"engine": { "type": "string", "enum": ["jexl", "jmespath"] },
"rules": {
"type": "array",
"minItems": 1,
"items": {
"type": "object",
"additionalProperties": false,
"required": ["if", "to"],
"properties": { "if": { "type": "string", "minLength": 1 }, "to": { "type": "string",
"minLength": 1 } }
}
}
}
}
}
}
]
},

"MemoryKVNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
"properties": {
"type": { "const": "memory.kv" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["mode", "scope", "backend"],
"properties": {
"mode": { "type": "string", "enum": ["load", "save"] },
"scope": { "type": "string", "enum": ["conversation", "run"] },
"backend": { "type": "string", "enum": ["postgres", "memory"] }
}
}
}
}
]
},

"ModelLLMNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
```

```
"properties": {
"type": { "const": "model.llm" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["provider", "model"],
"properties": {
"provider": { "type": "string", "enum": ["azure", "openai", "local"] },
"model": { "type": "string" },
"temperature": { "type": "number", "minimum": 0, "maximum": 2, "default": 0.2 }
}
}
}
}
]
},

"ToolHttpNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
"properties": {
"type": { "const": "tool.http" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["method", "url"],
"properties": {
"method": { "type": "string", "enum": ["GET", "POST", "PUT", "PATCH", "DELETE"] },
"url": { "type": "string", "format": "uri" },
"headers": { "type": "object", "additionalProperties": { "type": "string" } },
"body": {}
}
}
}
}
]
},

"ToolPostgresNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
"properties": {
"type": { "const": "tool.postgres" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["connectionRef", "query"],
"properties": {
"connectionRef": { "type": "string", "minLength": 1 },
"query": { "type": "string", "minLength": 1 }
}
}
}
}
]
},
```

```
"ResponseChatNode": {
"allOf": [
{ "$ref": "#/$defs/NodeBase" },
{
"type": "object",
"required": ["type", "config"],
"properties": {
"type": { "const": "response.chat" },
"config": {
"type": "object",
"additionalProperties": false,
"required": ["format"],
"properties": {
"format": { "type": "string", "enum": ["text", "json"] },
"template": { "type": "string" }
}
}
}
}
]
}
}
}
```

4. Integridad del grafo (validación adicional)

Además del schema, el backend (y opcionalmente el frontend) valida integridad:

• IDs únicos: node.id y edge.id sin duplicados.
• start existe y apunta a un nodo válido.
• edges: source/target deben existir.
• No self-loops (source==target) en v1.
• Condition rules: rule.to debe existir.
• Ciclos: en v1 bloquear ciclos (salvo futura bandera explícita).

Formato de error recomendado

```
{
"ok": false,
"errors": [
{ "code": "START_NOT_FOUND", "message": "start 't0' no existe", "path": "start" },
{ "code": "EDGE_TARGET_NOT_FOUND", "message": "Edge e4 target 'r_chat' no existe", "path":
"edges[3].target" }
]
}
```

5. Validación en Front (TypeScript) con AJV

Instalación

```
npm i ajv ajv-formats

import Ajv from "ajv";
import addFormats from "ajv-formats";
import schema from "./schemas/graph.schema.json";

const ajv = new Ajv({ allErrors: true, strict: false });
addFormats(ajv);

const validate = ajv.compile(schema);

export function validateGraph(graph: unknown) {
const ok = validate(graph);
return { ok: !!ok, errors: validate.errors ?? [] };
}
```

Recomendación: mapear validate.errors a mensajes amigables y (si se puede) resaltar el node/edge implicado.

6. Validación en Back (Python) con fastjsonschema

Instalación

```
pip install fastjsonschema

import json
import fastjsonschema

with open("schemas/graph.schema.json", "r", encoding="utf-8") as f:
schema = json.load(f)

validate = fastjsonschema.compile(schema)

def validate_schema(graph: dict) -> None:
validate(graph) # levanta exception si falla
```

Después de validate_schema(), correr validate_integrity() para IDs únicos, start, edges, cycles.

7. Ejemplos (deben pasar en front y back)

7.1 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" }
]
}
```

7.2 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" }
]
}
```

8. Checklist para ambos devs (ejecución)

■ Clonar/consumir repo redmind-contracts (submodule o dependencia git).
■ Front: validar local con AJV al exportar/importar; mostrar errores.
■ Back: validar schema con fastjsonschema y luego integridad; devolver errores con path.
■ Mantener ejemplos sincronizados: cualquier cambio del schema requiere actualizar examples.
■ No romper compatibilidad: cambios breaking → v2 o typeVersion.