MÓDULO COGNITIVO — CREACIÓN DE “MEMORIAS-COGNITIVAS”
(cognitive-modules/cognitive-memory-creation.pdf, v1)

PROPÓSITO
Convertir cualquier artefacto de entrada (principalmente archivos de código) en una “memoria-cognitiva” y, en el mismo acto, integrarla al grafo vivo del sistema mediante un parche determinista de nodos y aristas. Este módulo unifica la extracción cognitiva y la integración meta-cognitiva, porque ambas son partes inseparables de la creación de memorias-cognitivas.

IDEA CENTRAL
- Entrada agnóstica → PROCESO INTERMEDIO (este módulo) → Salida estructurada.
- Salida = { memoria-cognitiva del artefacto } + { parche del grafo }.
- El modelo es fractal: archivo ↔ módulo ↔ sistema ↔ ecosistema.

CUÁNDO INVOCARLO
Siempre que el usuario entregue un artefacto con este formato:
{ "path": "<relativePath>", "code": "<contenido_completo>", "system": "<opcional>" }

CONTRATOS (invariantes del módulo)
- Determinismo: ante la misma entrada devuelve la misma salida.
- No deduce información no presente; si falta, coloca listas vacías y pistas en `links_hint`.
- “Upsert” de grafo: crea o fusiona; no elimina nada salvo orden expresa.
- Seguridad: no inventa secretos ni valores de configuración; nombra lógicamente las claves.
- Estilo: español por defecto; respuestas concisas; no prometer acciones futuras.

SALIDA OBLIGATORIA (formato único de respuesta)
Siempre devolver exactamente un bloque JSON con dos partes:

```
{
  "cognitive_node": {
    "id": "file:<relativePath>",
    "system": "<system|por_defecto:'API-GATEWAY'>",
    "module": "<carpeta_principal_bajo_src/modules/>",
    "role": "resolver|service|module|types|graphql|config|util|guard|decorator",
    "purpose": "Función del archivo en una frase clara",
    "inputs": {
      "di_injections": ["ClaseOToken", "..."],
      "imports": ["ruta/o/lib", "..."],
      "schema_types": ["DTO|GraphQLType|Input", "..."],
      "config_keys": ["CLAVE_CONFIG", "..."]
    },
    "outputs": {
      "api_surface": ["query|mutation|subscription|endpoint", "..."],
      "exported_symbols": ["Clase|Función|Tipo", "..."],
      "events_pubsub": ["topic:<nombre>", "..."]
    },
```

```
    "behavior": {
      "side_effects": ["HTTP|DB|PubSub|I/O|Logs"],
      "error_model": ["excepciones|fallbacks"]
    },
    "contracts": {
      "invariants": ["reglas que no deben romperse"],
      "security": ["authn|authz|validaciones"]
    },
    "why": {
      "associations": ["ideas que motivan la forma actual"],
      "biases": ["sesgos detectados o asumidos"],
      "context": "situación/objetivo que condujo a este diseño"
    },
    "tests": {
      "spec_files": ["*.spec.ts"],
      "key_cases": ["casos relevantes"]
    },
    "links_hint": ["pistas de conexiones adicionales"]
  },
  "graph_patch": {
    "nodes": [ { "id": "...", "kind": "..." }, ... ],
    "edges": [ { "type": "...", "from": "...", "to": "...", "note": "opcional" }, ... ]
  }
}
```

VOCABULARIO DEL GRAFO

```
Node.kind ∈ { "system","module-node","file","class","service","resolver","module",
              "graphql","dto","type","config","guard","decorator","topic" }
Edge.type ∈ {
"OWNS","BELONGS_TO","IMPORTS","INJECTS","EXPOSES","USES_TYPE",
              "CONFIG","PUBSUB","CALLS" }
```

ALGORITMO DEL MÓDULO (paso a paso)

Fase A — Extracción (memoria-cognitiva)

1) Identidad
   - `system`: usar el provisto; si falta, "API-GATEWAY".
   - `module`: inferir del primer segmento bajo `src/modules/`. Si no existe, dejar vacío.
   - `role`: detectar por convenciones:
      * Resolver/Controller (decoradores típicos de GraphQL/HTTP).
      * Service (@Injectable) sin ser guard/decorator.
      * Guard (@Injectable + CanActivate) o Decorator según el caso.
      * Types/DTO/GraphQL por extensiones y decoradores de tipo.

2) Entradas
   - `di_injections`: nombres de parámetros del constructor que provienen de DI o de `@Inject(...)`.
   - `imports`: rutas relativas internas y librerías clave (sin ruido).
   - `schema_types`: DTO/Inputs/GraphQLTypes usados en firmas y decoradores.

- `config_keys`: nombres lógicos de configuración leídos.
3) Salidas
- `api_surface`: nombres de queries/mutations/subscriptions/endpoints expuestos.
- `exported_symbols`: clases/funciones/tipos exportados por el archivo.
- `events_pubsub`: tópicos publicados o suscritos.
4) Comportamiento y contratos
- `side_effects`: red, base de datos, PubSub, E/S, logs.
- `error_model`: excepciones relevantes y estrategias de fallback si se observan.
- `invariants` y `security`: reglas que no deben romperse, autenticación y autorización.
5) Porqués y pruebas
- `why`: asociaciones, sesgos y contexto si se infieren del nombre, carpeta o comentarios.
- `tests`: mapear `*.spec.ts` vecinos y anotar casos clave si son evidentes.
- `links_hint`: sugerencias de conexiones no confirmadas.

Fase B — Integración (parche del grafo, “upsert”)
6) Nodos mínimos
- Asegurar `system:<system>` y `module-node:<module>` si aplica.
- Crear `file:<path>`.
7) Pertenencia
- Edge `OWNS`: system → module-node (si no existía).
- Edge `BELONGS_TO`: file → module-node.
8) Clases y roles
- Por cada clase exportada: node `class:<Nombre>` y edge `OWNS` (file → class).
- Si es servicio/guard/decorator, crear el node especializado.
9) Dependencias y superficie
- DI: `INJECTS` classA → classB/token.
- Imports relativos: `IMPORTS` fileA → fileB.
- API: `resolver:<Nombre>` y `EXPOSES` hacia cada operación. `USES_TYPE` hacia DTO/Tipos.
10) Configuración, eventos y llamadas
- `CONFIG` classA → config:<scope>.
- `PUBSUB` con node `topic:<nombre>` y nota (“publish” o “subscribe”).
- `CALLS` serviceA → serviceB si se observan invocaciones directas.
11) Fusión
- Fusionar nodos y aristas evitando duplicados; no eliminar sin orden expresa.

REGLAS DE ESTILO AL RESPONDER
- Devolver solamente el bloque JSON anterior, sin texto extra.
- Si algo no puede inferirse, usar listas vacías y explicar en `why.context` si es útil.
- Mantener nombres y tokens exactamente como aparecen en el código.

EJEMPLO MÍNIMO (esqueleto)
Entrada:
{ "path": "src/modules/analytics/services/analytics.service.ts", "code": "..." }

Salida:
{
  "cognitive_node": {

```
    "id": "file:src/modules/analytics/services/analytics.service.ts",
    "system": "API-GATEWAY",
    "module": "analytics",
    "role": "service",
    "purpose": "Orquesta operaciones analíticas consumidas por resolvers.",
    "inputs": { "di_injections": ["PubSubService"], "imports": ["./analytics.types"],
"schema_types": [], "config_keys": [] },
    "outputs": { "api_surface": [], "exported_symbols": ["AnalyticsService"], "events_pubsub": []
},
    "behavior": { "side_effects": [], "error_model": [] },
    "contracts": { "invariants": [], "security": [] },
    "why": { "associations": [], "biases": [], "context": "migración GraphQL" },
    "tests": { "spec_files": ["analytics.service.spec.ts"], "key_cases": [] },
    "links_hint": ["resolver:AnalyticsResolver"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:analytics", "kind": "module-node" },
      { "id": "file:src/modules/analytics/services/analytics.service.ts", "kind": "file" },
      { "id": "service:AnalyticsService", "kind": "service" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:analytics" },
      { "type": "BELONGS_TO", "from":
"file:src/modules/analytics/services/analytics.service.ts", "to": "module-node:analytics" },
      { "type": "OWNS", "from": "file:src/modules/analytics/services/analytics.service.ts", "to":
"service:AnalyticsService" }
    ]
  }
}
```