**M&A Unstructured → Store Normalizer**

Production-ready FastAPI service that **normalizes Unstructured.io JSON** (raw extracted PDF elements) into a **stable, auditable “store” format** you can validate, index, and later lift into a **knowledge graph**.

**Why this exists**

* Unstructured output is flexible but inconsistent (mixed element shapes, optional fields, parent pointers).
* M&A documents are litigation-sensitive: you need **verbatim text**, **layout coordinates**, **stable IDs**, and a **clear chain-of-custody**.
* This service creates a **canonical store**:
  + sections[\*].text always present (no lost text)
  + section\_id (deterministic URN) for graph edges and citations
  + parent\_element\_id + sequence to rebuild hierarchy and order
  + spans with page/bbox/polygon for precise localization
  + raw\_element preserved verbatim for audit
  + Optional schema block to validate the store with JSON Schema

**Endpoints**

**POST /api/structure**

Normalize an **uploaded .json file** (from Unstructured or similar) into the store.

**Form fields (multipart):**

* file *(required)*: the JSON file
* include\_schema *(bool, default: true)*: include JSON Schema alongside the store
* index\_text *(bool, default: false)*: include **full text** in a lightweight index (topology.section\_index). This **does not** affect sections[\*].text, which is always present.
* snippet\_chars *(int, default: 280)*: when index\_text=false, controls snippet length in the index

**Responses:**

* 200 OK → {"store": {...}, "schema": {...}}
* 400 → validation/parse errors (you’ll see a clear message)

**Swagger demo steps:**

1. Go to /docs
2. Open **POST /api/structure** → **Try it out**
3. Upload your raw .json
4. (Optional) toggle index\_text or include\_schema
5. **Execute** → inspect the response

**POST /api/structure/rawjson**

Normalize an **inline JSON payload** (e.g., when you post {"elements":[...]} directly).

**Body (application/json):**

* Either an **array** of unstructured elements, or an **object** with an elements array.

**Query params:**

* include\_schema *(bool, default: true)*
* index\_text *(bool, default: false)*
* snippet\_chars *(int, default: 280)*

**Responses:**

* same as /api/structure

**Examples in Swagger:** Look for the built-in example payloads.

**store.document**

Stable header for audit & indexing.

* doc\_id *(URN)*, filename, filetype, hash
* extracted\_with (e.g., unstructured.io), extracted\_at, version
* Optional legal metadata: governing\_law, jurisdiction, source\_url

**store.sections[] — the important bit**

Each item is a **canonical section** built from one unstructured element. This is where your **TEXT lives**.

Key fields you’ll use most:

* section\_id — deterministic URN (use this as your KG node ID)
* element\_id — original unstructured id (for traceability)
* parent\_element\_id — build hierarchy (title → subclause → bullet)
* sequence — order under the same parent
* text — ✅ canonical text (guaranteed present)
* element\_type — Title, NarrativeText, Header, etc.
* page\_start / page\_end — page bounds
* spans[] — { page, bbox, polygon } in PDF coordinates (PixelSpace 612×792)
* raw\_element — verbatim original object (full fidelity audit)
* Diagnostics: text\_source, text\_candidates, text\_length, missing\_text

You’ll notice items like "- 2 -" with element\_type: "Header". That’s a **page header/footer** preserved for audit. Filter them out for semantic tasks.

**store.definitions[] *(optional)***

Populated by later passes when you extract defined terms ("Exchange Ratio" etc.).

**store.cross\_references[] *(optional)***

For “Section 1.5(a)(iii)” style links. Fill in later, mapping source\_section\_id → resolved\_section\_id.

**store.topology *(optional)***

Lightweight indices and read-order helpers. Controlled by index\_text and snippet\_chars.

**store.provenance**

Toolchain details (who built it, when, counts). Extend with your pipeline versions.

**JSON Schema**

When include\_schema=true, we return a JSON Schema describing the store. Use it to **validate** incoming data and catch regressions in CI.

**Using the store to build a Knowledge Graph**

**Nodes** (typical):

* Headings (Title) and clause bodies (NarrativeText / UncategorizedText)
* Always carry: section\_id, text, element\_type, page\_\*

**Edges**:

* PARENT\_OF: use parent\_element\_id (map to the child’s section\_id)
* NEXT\_IN\_SEQUENCE: order siblings by sequence
* REFERS\_TO: from cross\_references (when you populate them)

**How the normalizer works (pipeline)**

1. **Load** raw JSON (load\_any\_shape)
   * Accepts either [...] or { "elements": [...] }
   * Normalizes to a list of element dicts
2. **Build** store (StoreBuilder)
   * Creates doc\_id, computes file hash, copies header metadata
   * For each element:
     + Creates a new **section**
     + Copies **verbatim text** → section.text
     + Derives/normalizes **spans** (bbox + polygon)
     + Transfers **parent id** → parent\_element\_id
     + Sets **sequence** per parent
     + Preserves the **raw\_element**
   * Populates diagnostics (text\_length, etc.)
   * (Optional) builds topology.section\_index with text or snippets
3. **Schema** (optional)
   * Builds JSON Schema matching the output, with evolution-friendly additionalProperties: true
   * Requires sections[\*].text

**Validation & QA tips**

* **CI**: validate the response store against the returned schema with jsonschema.
* **Sanity**: assert missing\_text == False for most sections; alert when true.
* **Noise**: filter Header element\_type for semantic text processing.
* **Stitching**: use parent\_element\_id+sequence to combine multi-line clauses.

**Performance & size**

* PDFs with thousands of elements produce large JSON. Consider gzip on the API.
* index\_text=true duplicates text into topology; keep it false unless you need that index for search.

**Troubleshooting**

* **Import "app.services.loaders" could not be resolved**
  + Ensure \_\_init\_\_.py files exist.
  + Run pip install -e . (editable install) **from repo root**.
  + Confirm your editor uses the same venv as uvicorn.
* **NameError: name 'Query' is not defined**
  + Add it to your imports:  
    from fastapi import APIRouter, UploadFile, File, HTTPException, Query
* **Pylance unresolved symbols**
  + VS Code → Command Palette → *Python: Select Interpreter* → pick your venv
  + Reload window.
* **400 Bad Request**
  + The file isn’t JSON or the payload is not an array or {elements: [...]} object.
  + Check that your JSON is UTF-8.

**Security notes**

* Validate/limit upload size.
* Treat input as untrusted—never eval fields.
* Consider redacting PII at ingest if your compliance requires it.
* Enable HTTPS and auth in production (reverse proxy or FastAPI deps).

**Extending the pipeline**

* **Definition extraction**: scan sections[\*].text] for “is referred to as” patterns → populate store.definitions.
* **Cross-refs**: regex “Section \d+(.\d+)*([a−z]+[a-z]+[a−z]+)*” → resolve by label → store.cross\_references.
* **Clause classification**: add section.tags = ["tax", "governance"] based on ML/rules (schema permits extras).

**FAQ**

**Q: Why keep raw\_element if we already have text?**  
A: Audit, reproducibility, and fallback. If a later pass wants coordinates/language/filetype nuances, the verbatim object is there.

**Q: Does index\_text affect sections[\*].text?**  
A: No. It only controls what goes into topology.section\_index. sections[\*].text is always present.

**Q: How do I ignore page headers like “- 2 -”?**  
A: Filter element\_type == "Header".