
Defines project metadata and dependencies.

- **Name**: `alaris-assignment`
- **Version**: `1.0.0`
- **Scripts**: `test` placeholder
- **Dependencies**
  - `@google/generative-ai` for Gemini model calls
  - `express` for API server
  - `pdf-parse` , `pdfjs-dist` for PDF text extraction
  - `pg` for PostgreSQL interactions
  - `typescript` & `ts-node` for development
- **DevDependencies** include type definitions for Express and pg

---

Configures TypeScript compiler options.

- **Target**: ES2021, CommonJS modules
- **Strict** mode enabled
- **Paths**: Root `src/` directory
- **Emits**: Source maps for debugging

Defines core data structures for ingested papers.

```
export interface PaperMetadata { title: string; authors?: string[]; year?: number; venue?: string; canonicalId?: string; }
export interface PaperSection   { heading: string; level: number; text: string; }
export interface PaperReference { raw: string; title?: string; authors?: string[]; year?: number; doi?: string; arxivId?: string; }
export interface PaperDocument  { metadata: PaperMetadata; abstract?: string; sections: PaperSection[]; rawText: string; references: PaperReference[]; }
```

- **PaperDocument** is the central unit passed through ingestion, extraction, graph building, and storage.

Utility to extract raw text from PDFs.

```
const pdfParse = resolvePdfParse();
export async function extractTextFromPdf(filePath: string): Promise<string> { ... }
```

- **resolvePdfParse()** finds the correct export of `pdf-parse`.
- Returns a single string of the PDF's text content.

Converts text or PDF into `PaperDocument`.

- **extractTitle** chooses the first non-boilerplate line ≥10 chars.
- **extractAbstract | extractSections** split raw text into metadata and sections.
- **ingestFromTextFile** reads a `.txt` and builds `PaperDocument`.
- **ingestFromPdfFile** wraps PDF parsing then reuses text ingestion logic.

Extracts authors via Gemini model.

```
export interface Author { name: string; affiliation?: string; email?: string; }
export async function extractAuthorsFromText(rawText: string): Promise<Author[]> { ... }
```

- **Model**: `gemini-2.5-flash`
- **Prompt**: Asks Gemini to parse header for names, affiliations, emails.
- Returns an array of `Author`.

Identifies key technical concepts via Gemini.

```
export interface ConceptNode { id: string; name: string; description?: string; sectionIndex?: number; }
export async function extractConceptsFromText(text: string): Promise<ConceptNode[]> { ... }
```

- **Model** configurable via `GEMINI_MODEL` env var.
- Returns up to 30 concepts with optional section indices.

Parses a raw text blob for numbered reference list.

```
export interface Reference { id: string; raw: string; }
export function extractReferences(text: string): Reference[] { ... }
```

- Splits by lines, finds “References” or “Bibliography”, then matches `1. Some text`.
- Outputs an array of `Reference` objects.

Converts `PaperDocument` into an initial graph of nodes & edges.

```
export interface GraphNode { id: string; type: string; data: any; }
export interface GraphEdge { from: string; to: string; type: string; }
export function extractGraph(doc: PaperDocument): Graph { ... }
```

- **Nodes**: one `Paper` and multiple `Section`
- **Edges**: `has_section` from paper → section

Integrates concepts into the base graph.

- Creates **Concept** nodes for each `ConceptNode`.
- Adds `discusses` edges from paper → concept.
- Adds `mentions` edges from section → concept when `sectionIndex` exists.

Integrates authors into the graph.

- Generates stable IDs via `author:<slug>`
- Creates **Author** nodes with `name`, `affiliation`, `email`.
- Adds bidirectional edges:
  - `authored_by`: paper → author
  - `author_of` : author → paper

Defines PostgreSQL schema for graph storage.

```
CREATE TABLE nodes ( id TEXT PRIMARY KEY, type TEXT NOT NULL, title TEXT, data JSONB, created_at TIMESTAMPTZ );
CREATE TABLE edges ( id BIGSERIAL PRIMARY KEY, from_id TEXT REFERENCES nodes, to_id TEXT REFERENCES nodes, type TEXT NOT NULL, data JSONB, created_at TIMESTAMPTZ );
CREATE INDEX idx_edges_from_id ON edges(from_id);
CREATE INDEX idx_edges_to_id   ON edges(to_id);
CREATE INDEX idx_edges_type    ON edges(type);
CREATE INDEX idx_nodes_type    ON nodes(type);
```

- **nodes** holds all entities (Paper, Section, Concept, Author).
- **edges** stores relationships with metadata support.

Shared PostgreSQL pool for queries.

```
export const pool = new Pool({ connectionString: process.env.DATABASE_URL });
```

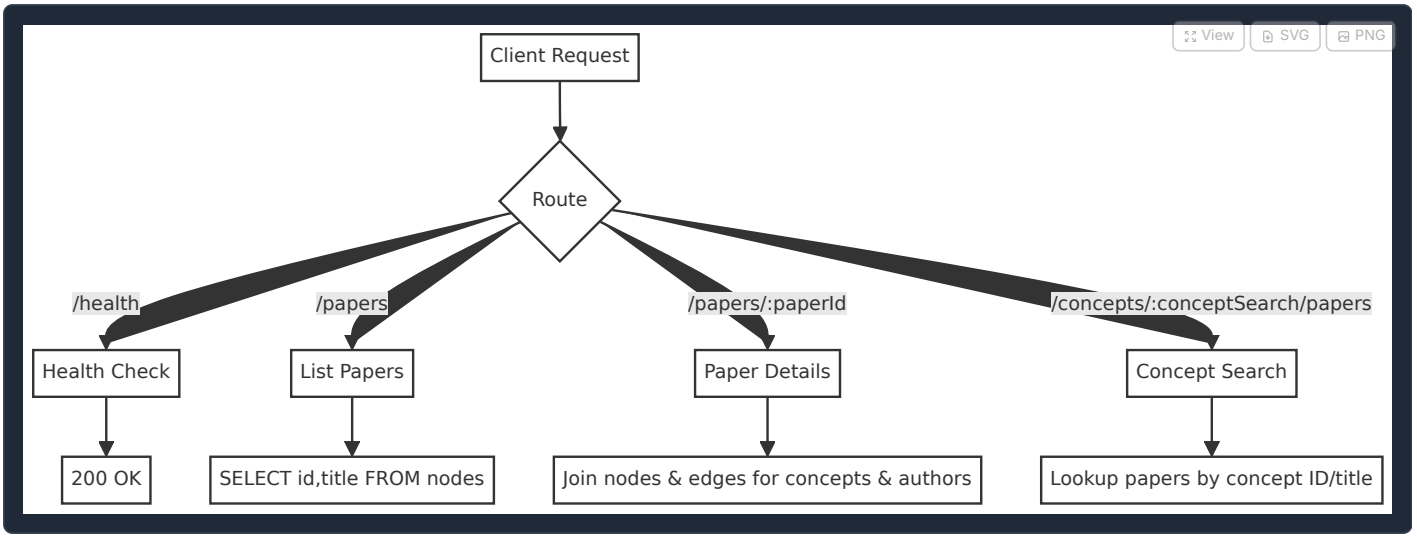
Reusable queries for Graph data retrieval.

- **getPaperByTitle(title)** returns one paper node.
- **getAnyPaper()** useful for smoke tests.
- **getAuthorsForPaper(paperId)** returns author nodes via `authored_by`.
- **getConceptsForPaper(paperId)** returns concept nodes via `discusses`.
- **closePool()** ends the pool in tests.

Upserts a full graph into the database.

- **inferTitle** extracts human-readable titles per node type.
- **saveGraphToDatabase(graph)**
  i. Begins transaction
  ii. Upserts nodes (`ON CONFLICT DO UPDATE`)
  iii. Inserts edges (`ON CONFLICT DO NOTHING`)
  iv. Commits or rolls back on error
- **closePool()** for clean shutdown

Express-based REST API exposing graph data.



Health Check

Export to Postman

Verifies server status.

GET `http://localhost:3000/health`

Code examples

```
curl -X GET "http://localhost:3000/health"
```

Responses

OK

```
{"status":"ok"}
```

List Papers

Export to Postman

Returns all papers in graph.

`GET` `http://localhost:3000/papers`

Code examples

```
curl -X GET "http://localhost:3000/papers"
```

Responses

A list of paper IDs and titles

```
{"count":2, "papers":[{"id":"paper:Foo","title":"Foo"},…]}
```

Paper Details

Export to Postman

Fetches a paper with its concepts and authors.

`GET` `http://localhost:3000/papers/:paperId`

Path parameters

paperId string • path required

Paper ID

Code examples

```
curl -X GET "http://localhost:3000/papers/:paperId"
```

Responses

200 404

Paper node, concepts & authors


```
{"paper":{…},"concepts":[…],"authors":[…]}
```


Not Found


```
{"error":"Paper not found"}
```


Concept Search

Export to Postman

Finds papers that discuss a given concept.

GET `http://localhost:3000/concepts/:conceptSearch/papers`

Path parameters

conceptSearch string • path required

Concept ID or title

Code examples

```
curl -X GET "http://localhost:3000/concepts/:conceptSearch/papers"
```

Responses

Grouped concept → papers mapping


```
{"conceptSearch":"NeRF","results":[{"conceptId":"concept:nerf","papers":[…]}]}
```

Standalone script to find papers sharing concepts.

- Connects via `pg.Client`
- Uses CTEs to compute concept overlap counts
- Outputs titles sorted by shared-concept count

Plain-text test fixtures.

- `sample.txt` is short sample paper text for ingestion tests.
- `mypdf.txt` is converted PDF text of “3D Gaussian Splatting…” for end-to-end pipeline tests.

Each verifies portions of the pipeline without a testing framework.

- **src/testIngestion.ts**: logs `PaperDocument` from sample.txt
- **src/testPdfIngestion.ts**: logs ingestion from mypdf.txt
- **src/testGraph.ts**: prints graph nodes & edges from a document
- **src/testConcepts.ts**: extracts concepts from the “Introduction” section
- **src/testConceptGraph.ts**: builds and logs full concept graph
- **src/testAuthor.ts**: extracts and logs authors
- **src/testReferences.ts**: ingests PDF then extracts references
- **src/testQueries.ts**: runs DB queries for paper, then prints authors & concepts
- **src/testExplainPaper.ts**: fetches graph from DB and asks Gemini to explain
- **src/testSaveGraph.ts**: exercises full pipeline and saves to DB
- **src/testFullGraphWithAuthors.ts**: like `testSaveGraph` but also closes pool

---

Note:

- Ingestion → Extraction → Graph building → Database saving form the core pipeline.
- The API server wraps DB queries, exposing the graph via REST.
- Test scripts allow manual verification at each stage.