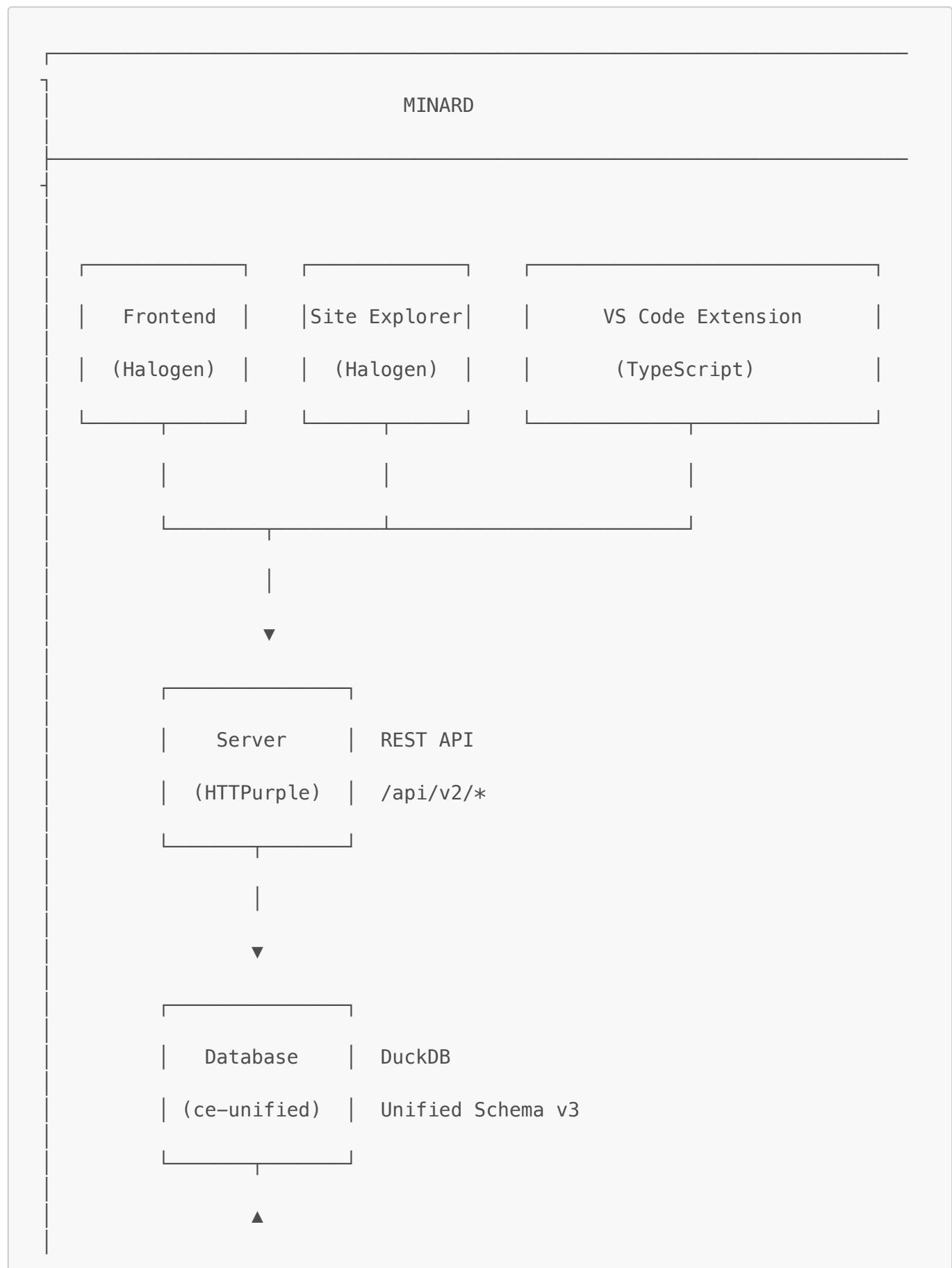


# Minard Architecture

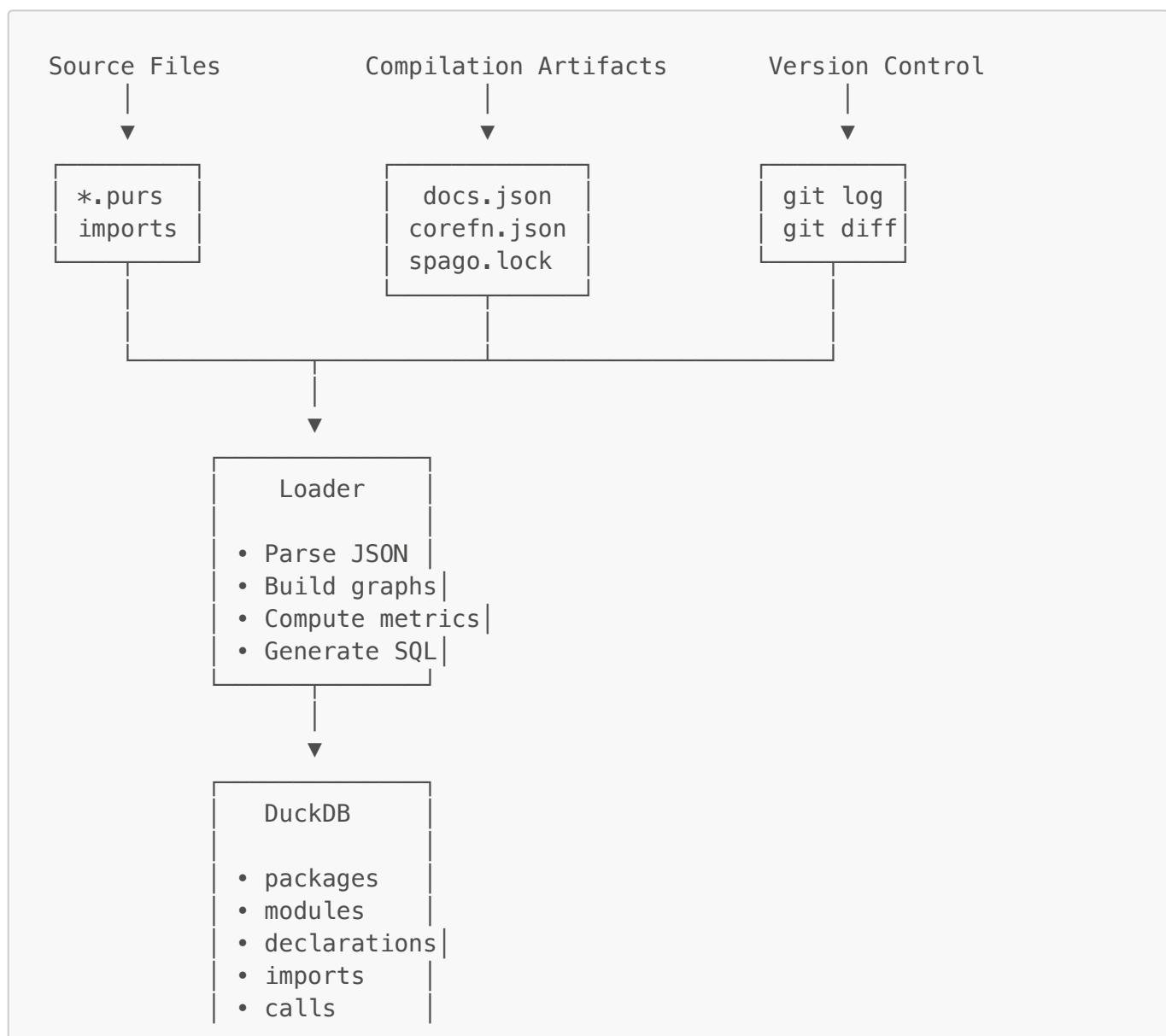
## System Overview





## Data Flow

### 1. Ingestion (Loader → Database)



- git metrics

## Data Sources:

Source	What We Extract	Tables Populated
spago.lock	Package versions, dependencies	package_versions, package_dependencies
output/*/docs.json	Module names, declarations, types	modules, declarations, child_declarations
output/*/corefn.json	Function calls, imports	function_calls, module_imports
git log	Commits, authors, dates	commits, module_commits
*.purs (src)	LOC, raw source	modules.loc, declarations.source_code

## 2. Query (Server → Database)

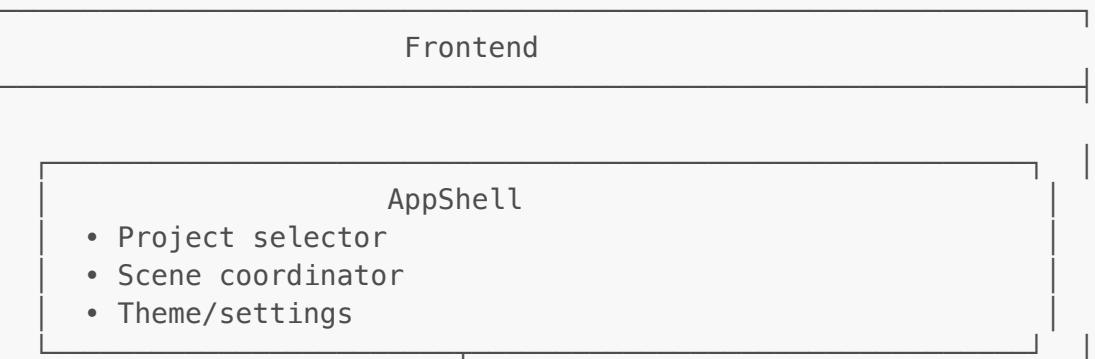
The server provides a REST API that translates HTTP requests to SQL queries:

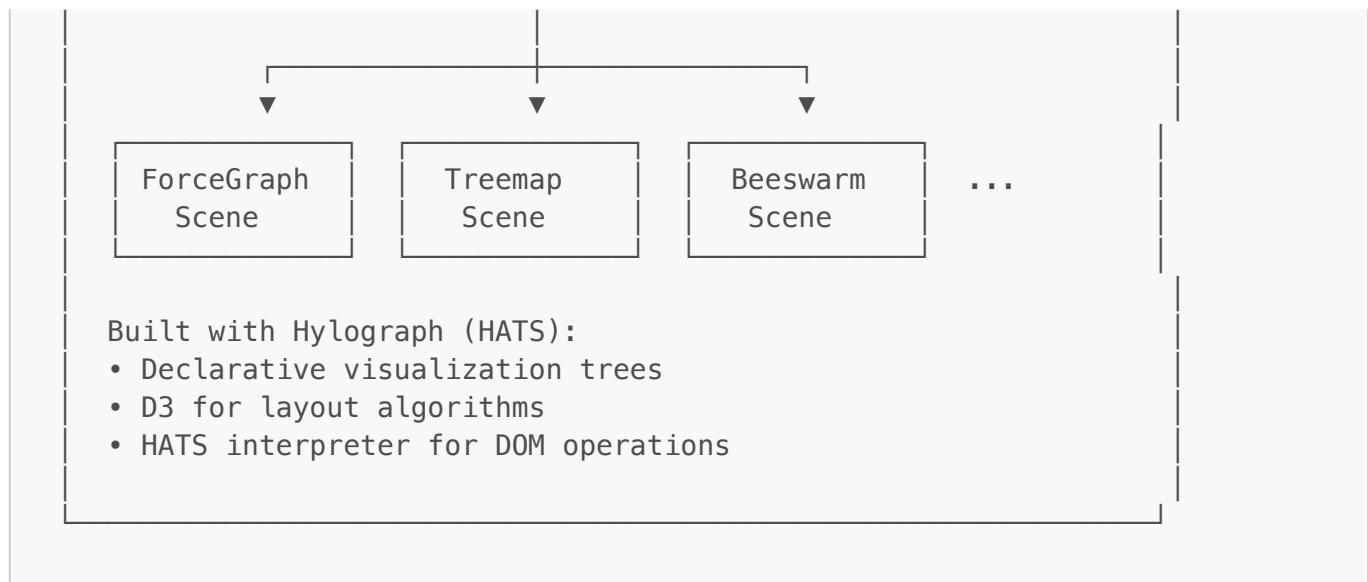
```
GET /api/v2/packages
↓
SELECT * FROM package_versions WHERE source IN ('workspace', 'extra')
↓
JSON response with moduleCount, declarationCount computed via subquery
```

## Key API Patterns:

- **List endpoints** return summary data (counts, not full objects)
- **Detail endpoints** return nested data (package → modules → declarations)
- **Bulk endpoints** ([/all-imports](#)) return denormalized data for graph building
- **Search endpoints** support type signature queries

## 3. Visualization (Frontend ← Server)

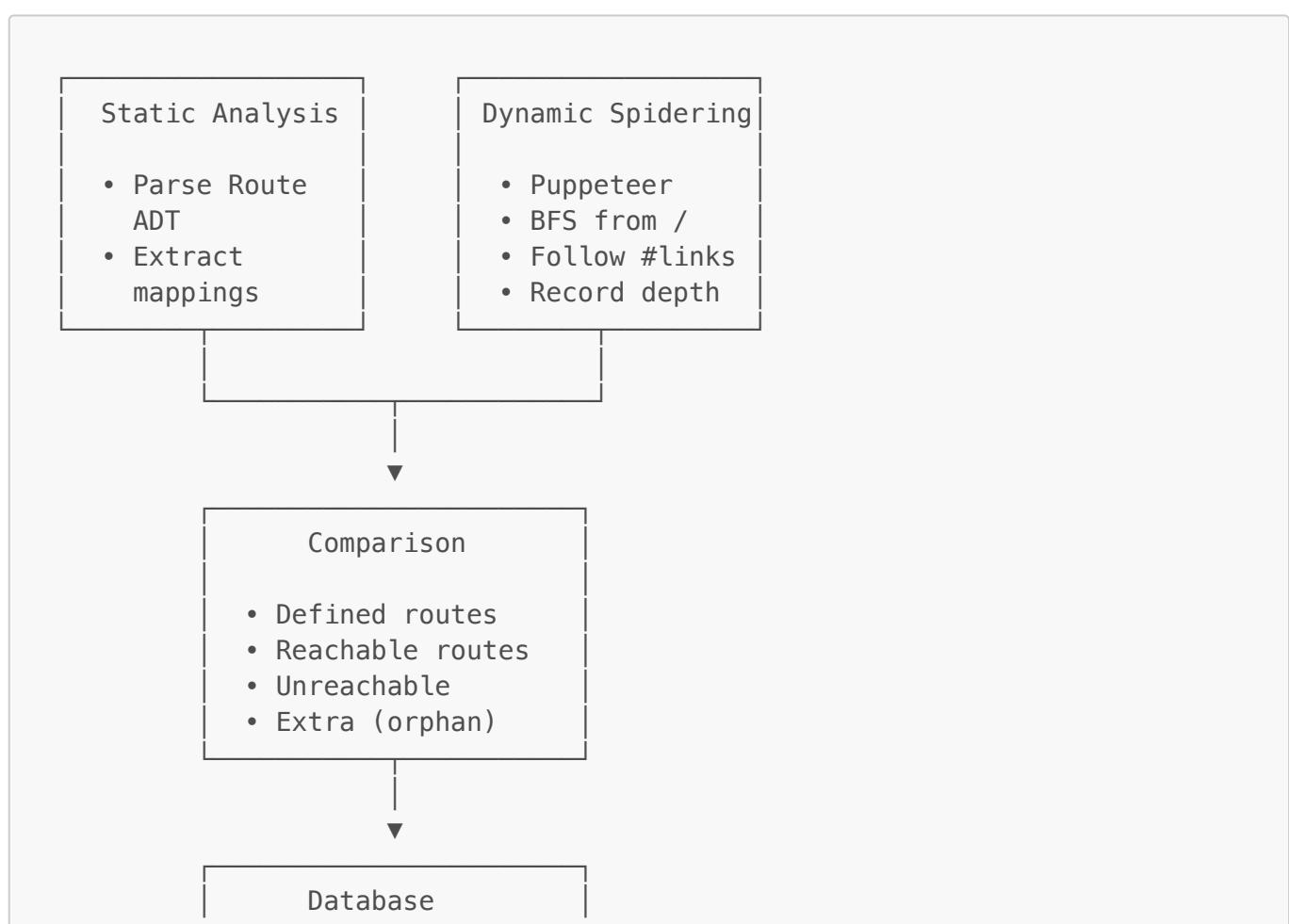




### Scene Architecture:

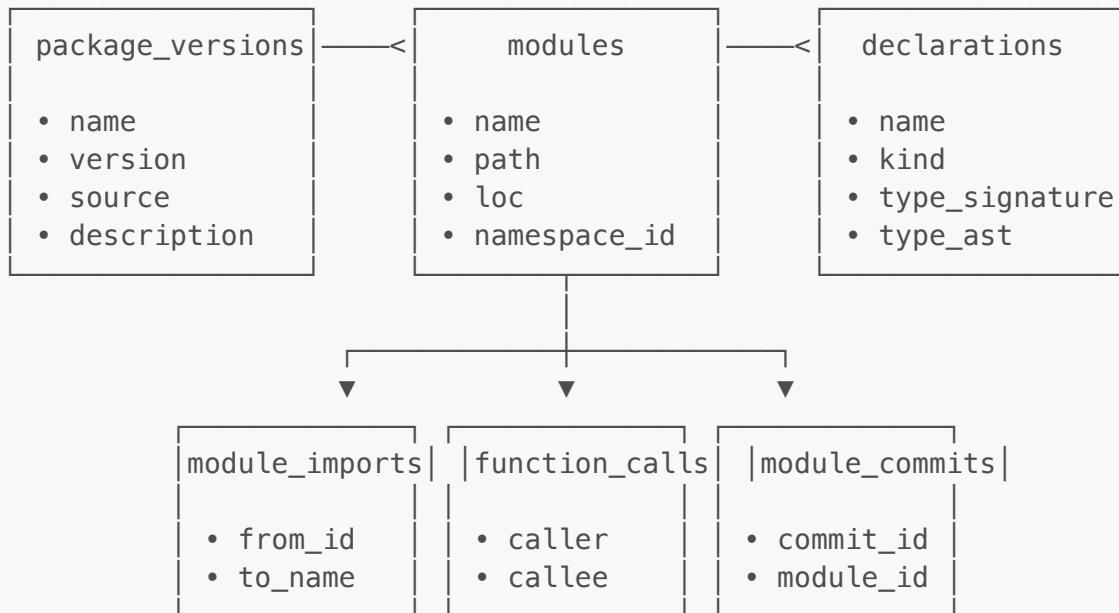
Each visualization scene follows this pattern:

1. **Data Loading:** Fetch from server, transform to visualization model
  2. **Layout Computation:** Use D3 algorithms (force, hierarchy, etc.)
  3. **HATS Tree Building:** Declarative specification of SVG structure
  4. **Rendering:** HATS interpreter produces DOM operations
  5. **Interaction:** Event handlers update state, trigger re-render
4. Site Explorer Data Flow



- routes
- spider\_runs
- discovered\_pages
- annotations

## Database Schema (Simplified)



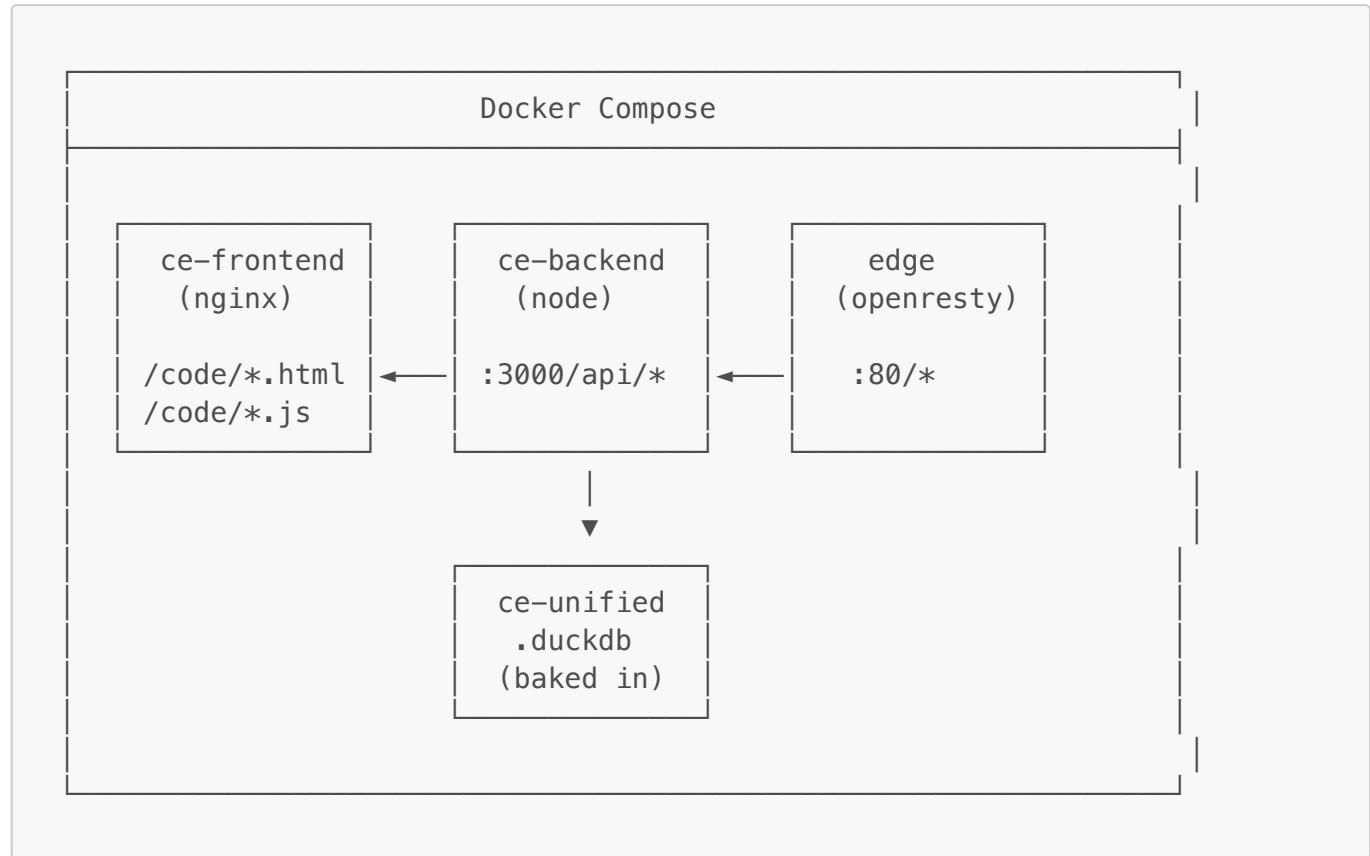
### Design Principles:

- 1. Package versions are identity:** `halogen@7.0.0` ≠ `halogen@8.0.0`
- 2. `spago.lock` is truth:** Versions come from lock file, not guessing
- 3. Types are first-class:** Both rendered (`String -> Int`) and AST (queryable JSON)
- 4. Namespaces independent:** `Data.Array` namespace exists even without that module

## Technology Stack

Component	Technology	Why
Frontend	PureScript + Halogen	Type safety, declarative UI
Visualization	Hylograph (HATS)	Declarative, composable
Server	PureScript + HTTPurple	Type-safe HTTP, same language as frontend
Database	DuckDB	Fast analytics, embedded, SQL
Loader	Node.js (→ Rust/Go)	Easy start, performance rewrite planned
Site Spider	Puppeteer	Headless Chrome, reliable SPA crawling
VS Code	TypeScript	Required by VS Code extension API

## Deployment

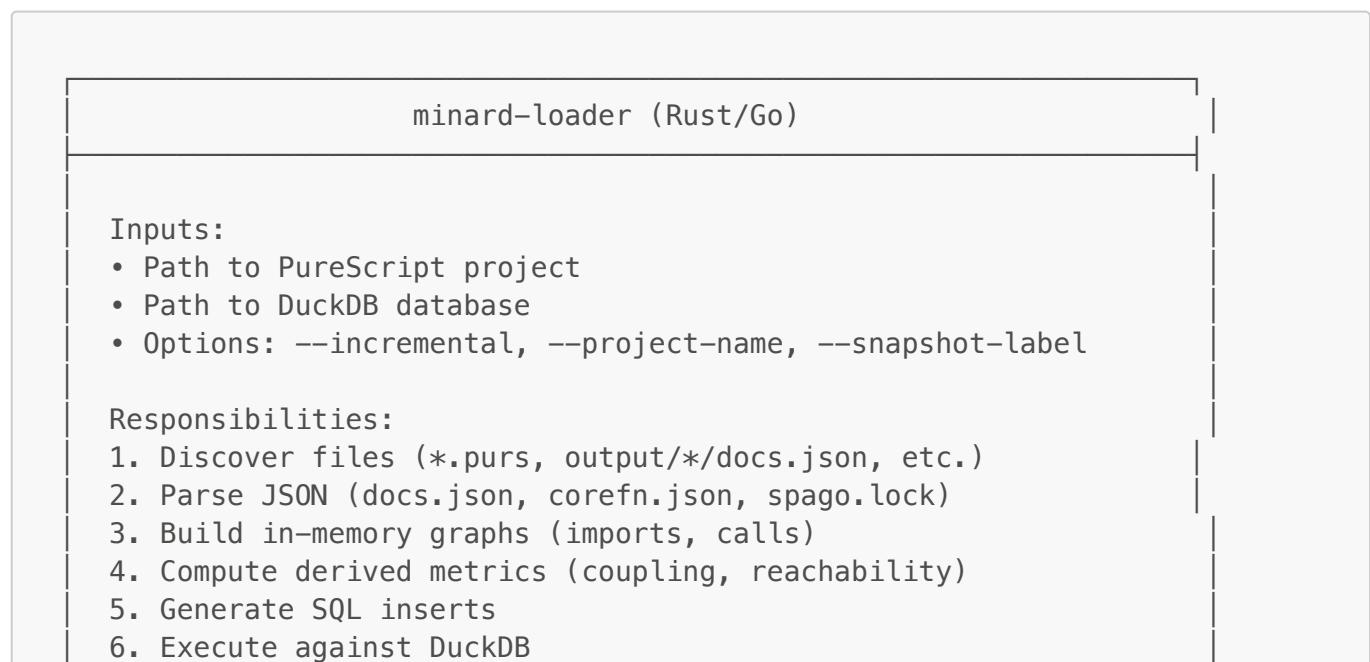


### Edge Router:

- OpenResty (nginx + Lua)
- Routes `/code/api/*` → ce-backend
- Routes `/code/*` → ce-frontend
- URL rewriting: strips `/code` prefix

### Future: Loader Rewrite

The current loader (`ce-loader.js`, ~800 lines Node.js) works but is slow. Plan:



Performance target: <5s for 1000 modules (currently ~30s)

Key constraint: Run on every compile without friction

See [docs/LOADER-SPEC.md](#) for full specification.