MarkAny* GaneshaIT*

Civil Construction App - Technical Documentation

Version: 1.0.0

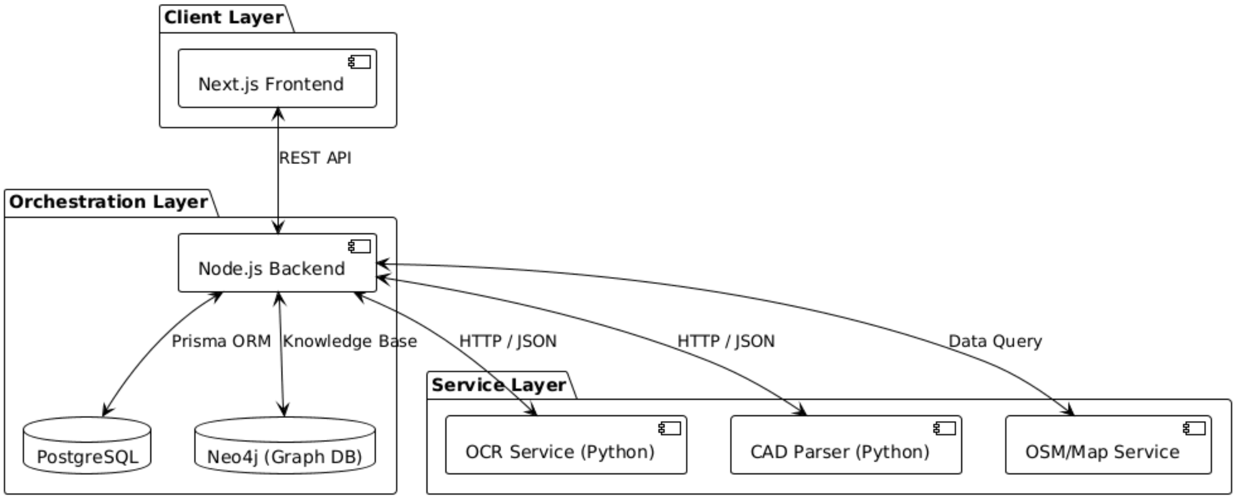
Date: February 21, 2026

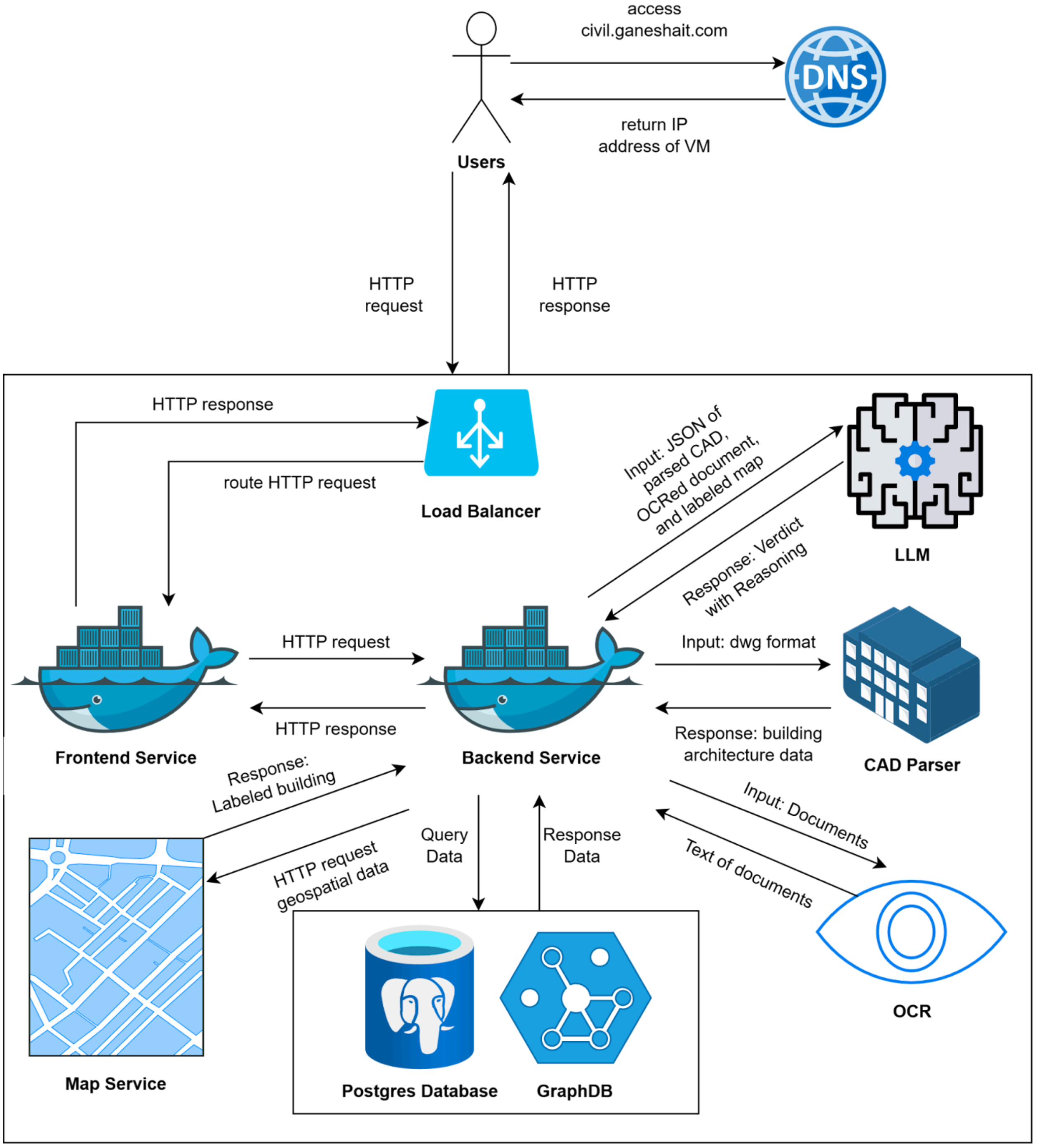
1. System Overview

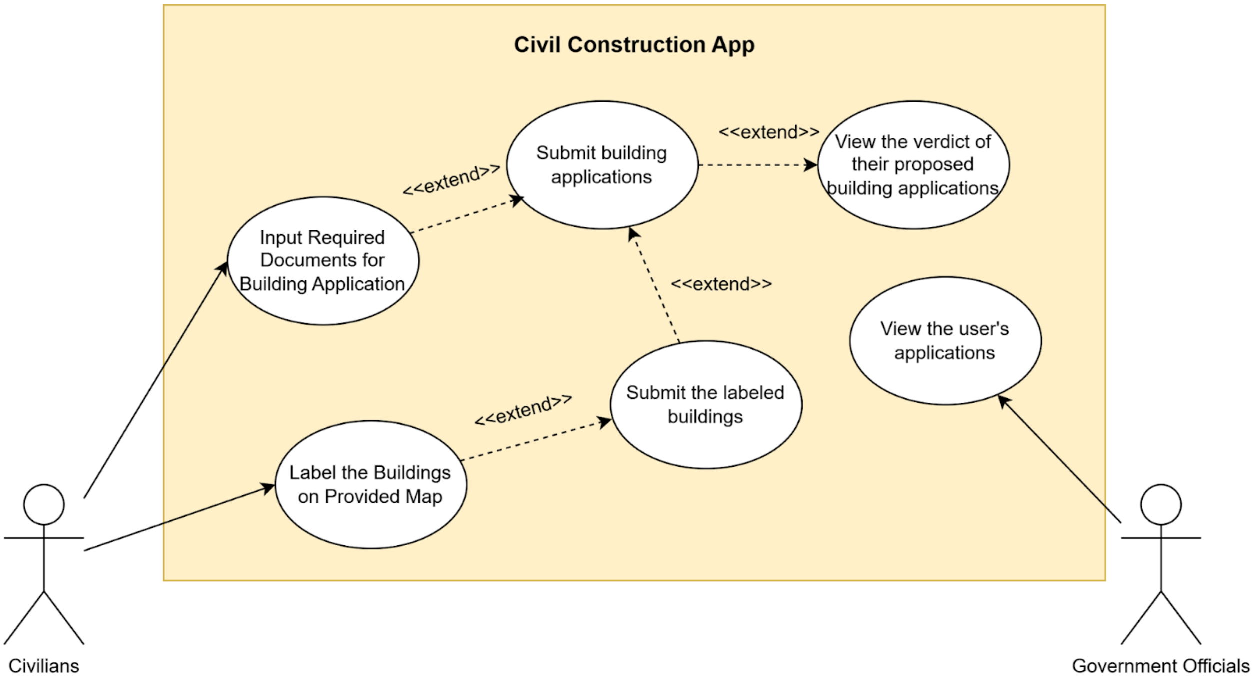
The Civil Construction App is a modular platform designed to assist civil engineers and construction professionals. It integrates various specialized services—including CAD parsing, optical character recognition (OCR), regulatory knowledge bases, and geospatial mapping—into a unified interface.

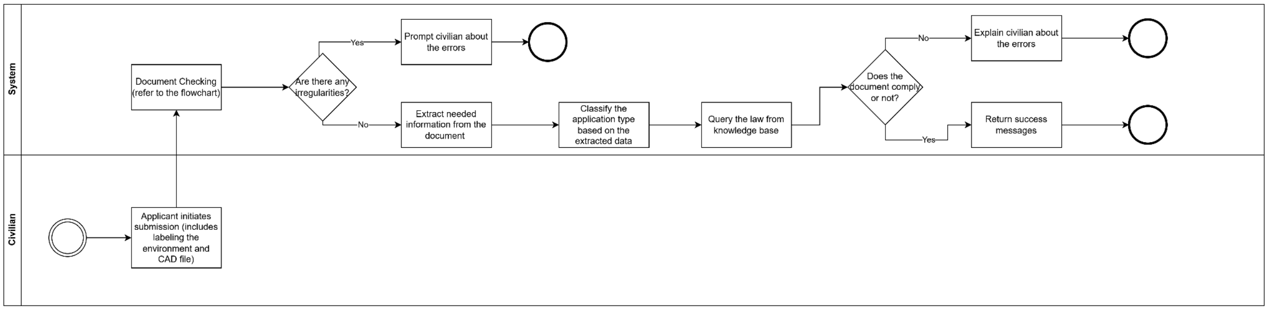
1.1 High-Level Architecture

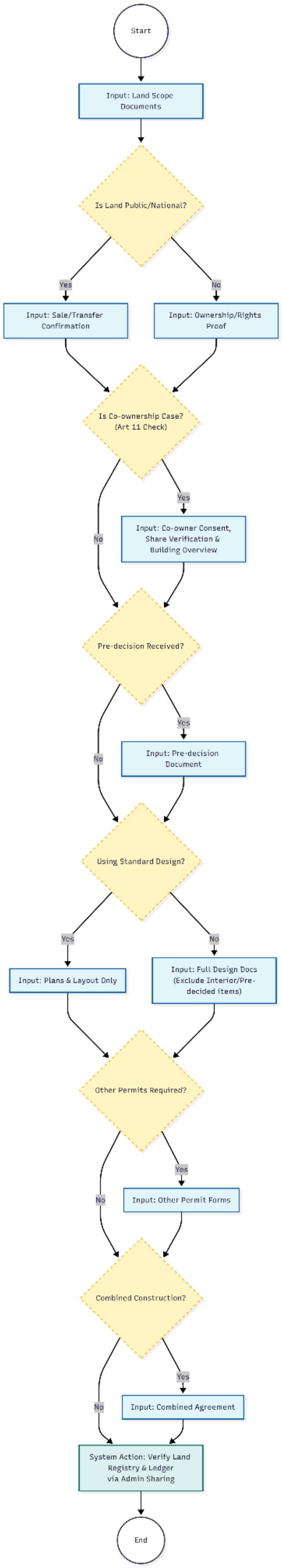
The system follows a monolith-based architecture, orchestrated by a central Node.js backend which serves a Next.js frontend. Specialized computational tasks (OCR, CAD processing) are offloaded to dedicated Python services.




access
civil.ganeshait.com
return IP
address of VM
Users
HTTP
request
HTTP
response
HTTP response
route HTTP request
Load Balancer
Input: JSON of
parsed CAD,
OCRed document,
and labeled map
Response: Verdict
with Reasoning
LLM
HTTP request
HTTP response
Input: dwg format
Response: building
architecture data
CAD Parser
Frontend Service
Backend Service
Response:
Labeled building
HTTP request
geospatial data
Query
Data
Response
Data
Input: Documents
Text of documents
OCR
Map Service
Postgres Database
GraphDB


Civil Construction App
Input Required Documents for Building Application
Submit building applications
View the verdict of their proposed building applications
Label the Buildings on Provided Map
Submit the labeled buildings
View the user's applications
<<extend>>
<<extend>>
<<extend>>
<<extend>>
Civilians
Government Officials


System
Civilian
Applicant initiates submission (includes labeling the environment and CAD file)
Document Checking (refer to the flowchart)
Are there any irregularities?
Yes
Prompt civilian about the errors
No
Extract needed information from the document
Classify the application type based on the extracted data
Query the law from knowledge base
Does the document comply or not?
No
Explain civilian about the errors
Yes
Return success messages


Start
Input: Land Scope Documents
Is Land Public/National?
Yes
No
Input: Sale/Transfer Confirmation
Input: Ownership/Rights Proof
Is Co-ownership Case? (Art 11 Check)
Yes
No
Input: Co-owner Consent, Share Verification & Building Overview
Pre-decision Received?
Yes
No
Input: Pre-decision Document
Using Standard Design?
Yes
No
Input: Plans & Layout Only
Input: Full Design Docs (Exclude Interior/Pre-decided items)
Other Permits Required?
Yes
No
Input: Other Permit Forms
Combined Construction?
Yes
No
Input: Combined Agreement
System Action: Verify Land Registry & Ledger via Admin Sharing
End

2. Module: OSM Geospatial Map

2.1 Overview

The OSM Geospatial Map System is an interactive visualization module designed to render real-world infrastructure data within the Civil Construction Application. Unlike static map images, this system dynamically queries **OpenStreetMap (OSM)** data to generate editable, classified geospatial features.

2.2 Technology Stack

- **Mapping Engine: Leaflet.js** - *Handles tile rendering, coordinate projection, and user interaction layers.*
- **Data Interchange: GeoJSON** - *Standard format for representing vector features (Points, Polygons, LineStrings).*
- **Geocoding Service: Nominatim API** - *Translates text queries (e.g., "Seoul Station") into Lat/Lon coordinates.*
- **Vector Data Service: Overpass API** - *Executes complex QL (Query Language) scripts to retrieve specific infrastructure nodes and ways.*

2.3 Data Schema (GeoJSON)

The system transforms raw OSM XML/JSON into a **Strict GeoJSON Feature Schema**. This structure separates the "Geometry" (Shape) from the "Properties" (Metadata) and "Style" (Visuals).

Data Definitions

A. Feature Collection (Root)

The container for all map objects.

- `type`: "FeatureCollection"
- `features`: Array of `GeoJSONFeature` objects.

B. GeoJSON Feature (The Object)

Represents a single physical entity (e.g., a specific building).

- `type`: "Feature"
- `id`: String (e.g., "way/12345678")

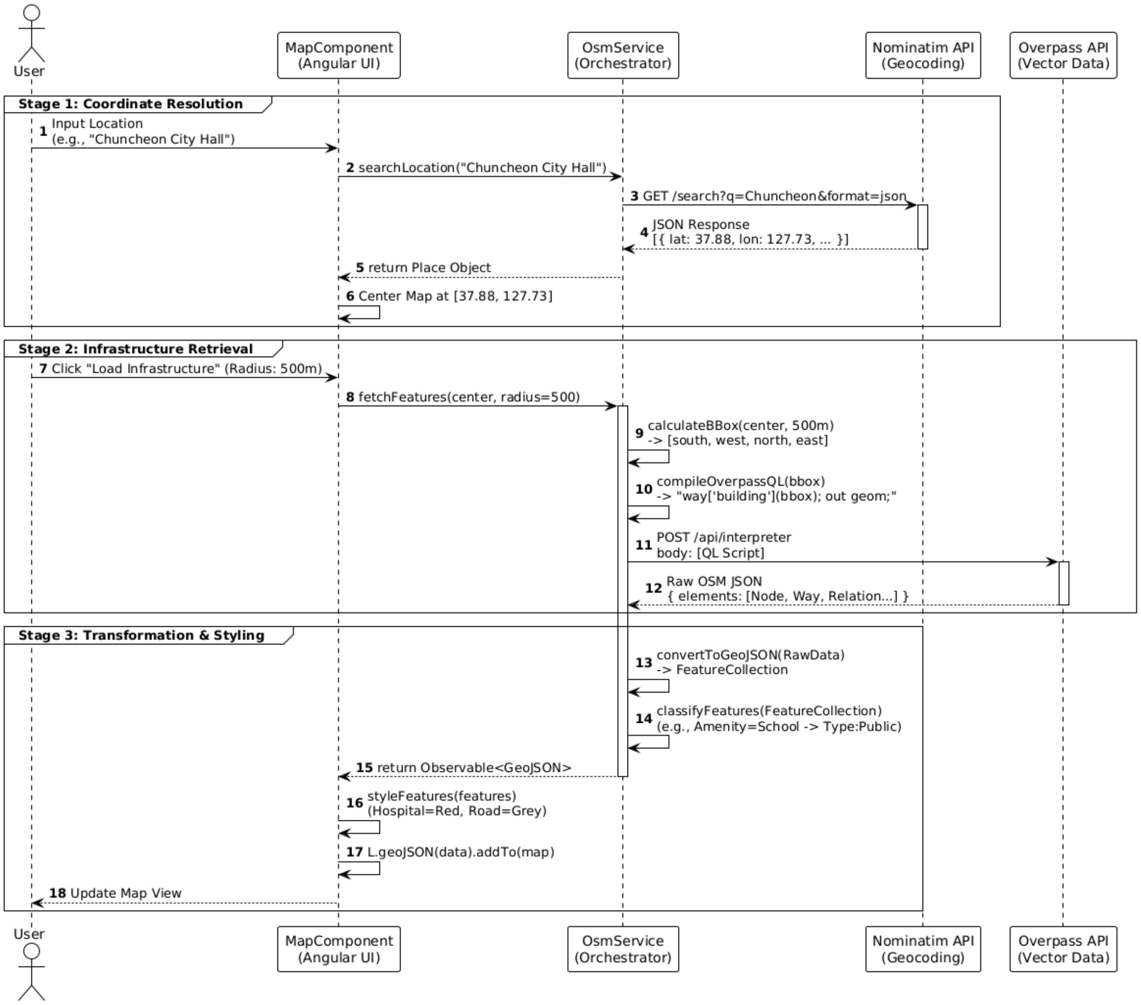
- **`geometry`**:
    - `type`: "Polygon" (Buildings) | "LineString" (Roads) | "Point" (Amenities)
    - `coordinates`: Array of Latitude/Longitude pairs.
- **`properties`**:
    - `osm_id`: Unique Identifier.
    - `amenity`: Type (e.g., "school", "hospital").
    - `building`: Levels, Height, Name.
    - `category`: **Inferred Field** (e.g., "Critical Infrastructure").

2.4 Geospatial Retrieval Workflow

The system chains different geospatial APIs to deliver a clean map view. This logic is orchestrated by the system.

Retrieval Strategy

1. **Stage 1: Coordinate Resolution (Geocoding)**
    - **Trigger:** User inputs text (e.g., "Gangnam District").
    - **Mechanism:** Queries Nominatim API.
    - **Output:** Center Point `[lat, lon]` and Bounding Box.
2. **Stage 2: Bounding Box Calculation**
    - **Trigger:** Center Point determined.
    - **Mechanism:** Calculates a square bounding box based on the selected radius (e.g., 500m).
    - **Formula:** `lat ± (radius / 111km)`, `lon ± (radius / (111km * cos(lat)))`.
3. **Stage 3: Overpass QL Execution (Vector Fetch)**
    - **Trigger:** Bounding Box established.
    - **Mechanism:** Sends a compiled **Overpass QL** script to the Interpreter.
4. **Stage 4: Classification & Styling**
    - **Mechanism:** Client-side mapping of tags to colors.


User
MapComponent (Angular UI)
OsmService (Orchestrator)
Nominatim API (Geocoding)
Overpass API (Vector Data)
Stage 1: Coordinate Resolution
1 Input Location (e.g., "Chuncheon City Hall")
2 searchLocation("Chuncheon City Hall")
3 GET /search?q=Chuncheon&format=json
4 JSON Response [{ lat: 37.88, lon: 127.73, ... }]
5 return Place Object
6 Center Map at [37.88, 127.73]
Stage 2: Infrastructure Retrieval
7 Click "Load Infrastructure" (Radius: 500m)
8 fetchFeatures(center, radius=500)
9 calculateBBox(center, 500m) -> [south, west, north, east]
10 compileOverpassQL(bbox) -> "way['building'](bbox); out geom;"
11 POST /api/interpreter body: [QL Script]
12 Raw OSM JSON { elements: [Node, Way, Relation...] }
Stage 3: Transformation & Styling
13 convertToGeoJSON(RawData) -> FeatureCollection
14 classifyFeatures(FeatureCollection) (e.g., Amenity=School -> Type:Public)
15 return Observable<GeoJSON>
16 styleFeatures(features) (Hospital=Red, Road=Grey)
17 L.geoJSON(data).addTo(map)
18 Update Map View
User
MapComponent (Angular UI)
OsmService (Orchestrator)
Nominatim API (Geocoding)
Overpass API (Vector Data)

2.5 API Specifications

1. Nominatim Geocoding

- **GET** `https://nominatim.zopenstreetmap.org/search`
- **Params:** `{ q: string, format: 'json' }`
- **Purpose:** Resolves human-readable addresses to coordinates.

2. Overpass Interpreter

- **POST** `https://overpass-api.de/api/interpreter`
- **Body:** `[out:json][timeout:25]; (way["building"](...);); out geom;`
- **Purpose:** Fetches raw vector data for buildings, roads, and waterways.

3. Module: OCR Service

3.1 Overview

The OCR (Optical Character Recognition) module extracts text and spatial layout information from construction documents, blueprints, and regulatory papers. It runs as **GPU-accelerated serverless workers on RunPod**, split into two independent endpoints:

- **OCR Endpoint** — Traditional OCR engines (Surya + PaddleOCR) on RTX 4090
- **VLM Endpoint** — Vision Language Model (Qwen2.5-VL-7B) on A6000

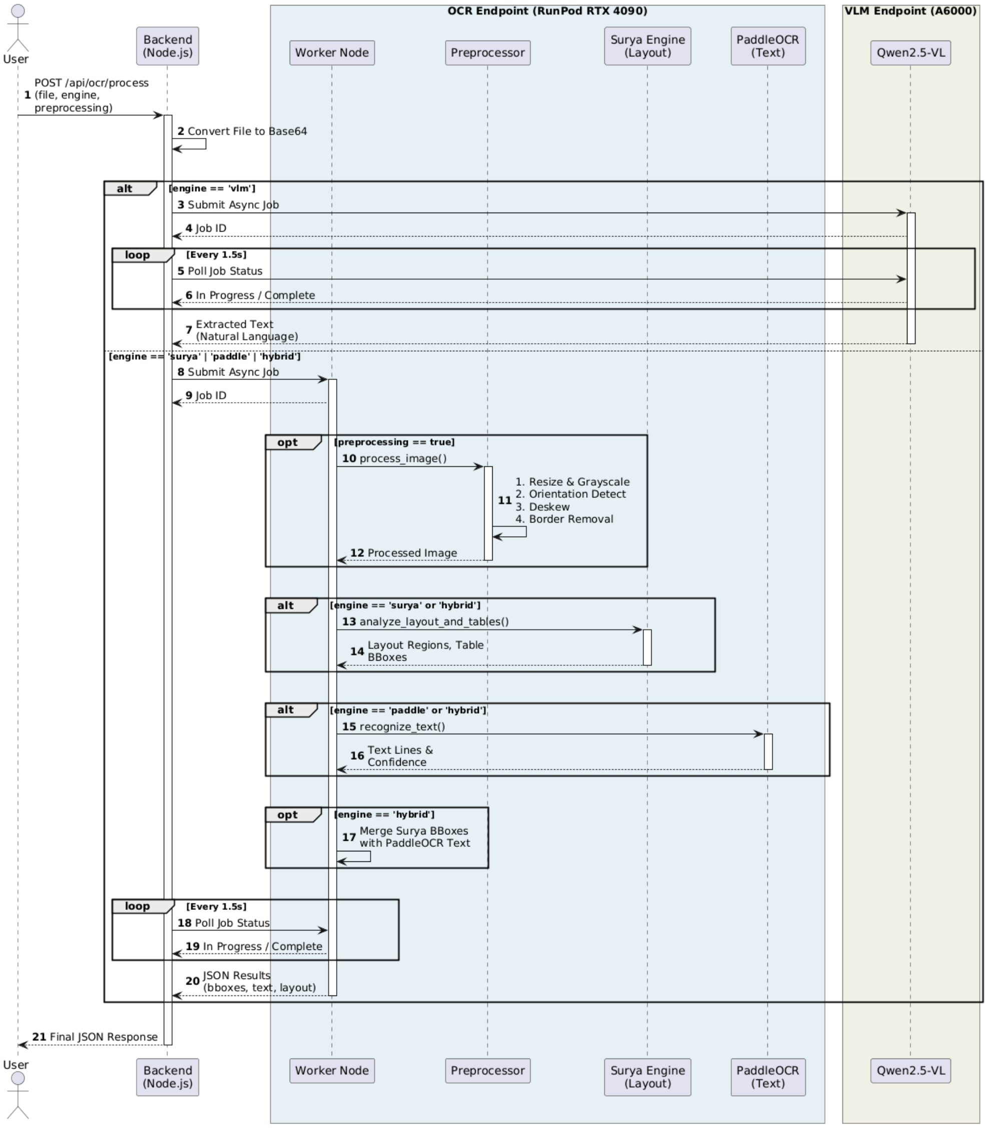
The backend (Node.js) acts as a proxy: it receives file uploads, converts them to base64, submits async jobs to RunPod, and polls for results.

3.2 Technology Stack

Component	**Technology**
Backend API	Express.js (Node.js/TypeScript)
OCR Layout & Tables	Surya-OCR 0.17.0
OCR Text Recognition	PaddleOCR >=3.1.0 (PP-OCRv5)
Vision Language Model	Qwen2.5-VL-7B-Instruct via vLLM
Image Processing	OpenCV (cv2), Pillow (PIL)
Orientation Detection	PaddleOCR DocImgOrientationClassification
Deep Learning Frameworks	PyTorch 2.8.0, PaddlePaddle GPU 3.3.0
Deployment	RunPod Serverless (GPU), Docker
OCR GPU	NVIDIA RTX 4090 (24GB VRAM, Ada Lovelace)
VLM GPU	NVIDIA A6000 (48GB VRAM, Ampere)

3.3 Architecture & Workflow

The architecture is designed to handle high-latency GPU inference through asynchronous job submission and polling. Below is the workflow sequence:


OCR Endpoint (RunPod RTX 4090)
VLM Endpoint (A6000)
User
Backend (Node.js)
Worker Node
Preprocessor
Surya Engine (Layout)
PaddleOCR (Text)
Qwen2.5-VL
1 POST /api/ocr/process (file, engine, preprocessing)
2 Convert File to Base64
alt [engine == 'vlm']
3 Submit Async Job
4 Job ID
loop [Every 1.5s]
5 Poll Job Status
6 In Progress / Complete
7 Extracted Text (Natural Language)
[engine == 'surya' | 'paddle' | 'hybrid']
8 Submit Async Job
9 Job ID
opt [preprocessing == true]
10 process_image()
11 1. Resize & Grayscale 2. Orientation Detect 3. Deskew 4. Border Removal
12 Processed Image
alt [engine == 'surya' or 'hybrid']
13 analyze_layout_and_tables()
14 Layout Regions, Table BBoxes
alt [engine == 'paddle' or 'hybrid']
15 recognize_text()
16 Text Lines & Confidence
opt [engine == 'hybrid']
17 Merge Surya BBoxes with PaddleOCR Text
loop [Every 1.5s]
18 Poll Job Status
19 In Progress / Complete
20 JSON Results (bboxes, text, layout)
21 Final JSON Response
User
Backend (Node.js)
Worker Node
Preprocessor
Surya Engine (Layout)
PaddleOCR (Text)
Qwen2.5-VL

Workflow Steps

1. User uploads a file via POST /api/ocr/process with engine selection
2. Backend converts the file to base64 and submits an async job to the appropriate RunPod endpoint
3. Backend polls the RunPod status API until the job completes or times out

4. The RunPod worker processes the file:
   - **PDF**: Converted to images at 300 DPI (one per page) using pdf2image/poppler
   - **DOCX**: Text extracted directly using python-docx (no OCR needed)
   - **Image**: Optionally preprocessed, then passed to the selected OCR engine
5. Results are returned as JSON with extracted text, bounding boxes, and layout information

3.4 Component Details

3.4.1 OCR Engines

1. **Surya Engine (engine: surya)**

   Full Surya pipeline with layout analysis, table detection, text detection, and text recognition. Best for documents where structural understanding (columns, tables, headers) is important.

   Pipeline: **FoundationPredictor** → **LayoutPredictor** → **TableRecPredictor** → **DetectionPredictor** → **RecognitionPredictor**

2. **PaddleOCR Engine (engine: paddle)**

   PaddleOCR PP-OCRv5 for text detection and recognition. Optimized for Korean and Latin scripts. Faster than Surya but does not provide layout/table analysis.

3. **Hybrid Engine (engine: hybrid) — Default**

   Combines Surya for structural analysis (layout regions, table detection, text bounding boxes) with PaddleOCR for text recognition. This gives the best of both: Surya's layout understanding with PaddleOCR's text accuracy for Korean documents.

4. **VLM Engine (engine: vlm)**

   Qwen2.5-VL-7B-Instruct vision-language model served via vLLM. Processes the entire image as a visual prompt and extracts text using natural language understanding. Best for complex or handwritten documents where traditional OCR fails.

   - Model is cached on a RunPod Network Volume (20GB) — downloaded once on first request
   - Served via vLLM OpenAI-compatible API (/v1/chat/completions)
   - Supports custom prompts via the prompt input field

3.4.2 Preprocessing Pipeline

Step	Technique	Details
1/5	Resizing	Downscale if any dimension exceeds 2000px (INTER_AREA interpolation)
2/5	Grayscale Conversion	BGR to grayscale to reduce channel complexity
3/5	Orientation Detection	PaddleOCR DocImgOrientationClassification (PP-LCNet_x1_0_doc_ori) detects 0/90/180/270 degree rotation. Double-checks 90/270 for ambiguity.
4/5	Skew Detection & Correction	Hough Transform (Canny edge detection + HoughLinesP) detects fine skew angles. Corrects if skew > 0.5 degrees.
5/5	Border Removal	Otsu thresholding + contour detection to crop to document content area

3.4.3 Device Management

The service runs exclusively on GPU (RunPod serverless workers):

- **PyTorch** (Surya): CUDA via torch.cuda
- **PaddlePaddle** (PaddleOCR): CUDA via paddlepaddle-gpu
- No CPU or MPS fallback — GPU-only deployment

3.5 Implementation Parameters & Configuration

This section details the specific configuration parameters utilized within the OCR service. These values are explicitly defined in the implementation to handle civil engineering drawings.

3.5.1 Text Recognition Configuration (PaddleOCR)

The OCR engine is initialized with specific arguments to tune the DB (Differentiable Binarization) algorithm and language support.

Parameter	Value	Explanation
OCR Version	PP-OCR v5	Latest version for improved accuracy
Angle Classification	False	Disabled — orientation handled by preprocessing pipeline
Language	korean	Korean + Latin character support
Detection Threshold	0.3	Binarization threshold for text detection (DB algorithm)
Box Threshold	0.6	Predicted boxes below this confidence are discarded
Unclip Ratio	1.8	Expands detected text regions for full character coverage

3.5.2 Layout & Text Detection Configuration (Surya)

The layout analysis pipeline utilizes specific predictors to identify document structure.

Component	Purpose
Foundation Predictor	Base model for initializing layout and recognition tasks
Layout Predictor	Identifies document regions (paragraphs, headers, figures, etc.)

Table Rec Predictor	Detects table structures and bounding boxes
Detection Predictor	Detects text line bounding boxes
Recognition Predictor	Performs text recognition on detected lines (surya engine only)

3.5.3 VLM Configuration

The configuration of the VLM used:

Parameter	**Value**	**Explanation**
Model	Qwen2.5-VL-7B-Instruct	7B parameter vision-language model
Serving Framework	vLLM	OpenAI-compatible API server
Max Model Length	4096 tokens	Context window limit
GPU Memory Utilization	0.85	85% of VRAM allocated to model
Max New Tokens	2048	Maximum output length
Temperature	0.1	Low temperature for deterministic extraction
Model Storage	RunPod Network Volume	~15GB model cached at /runpod-volume/models/
Parameter	**Value**	**Explanation**
Model	Qwen2.5-VL-7B-Instruct	7B parameter vision-language model
Serving Framework	vLLM	OpenAI-compatible API server

Max Model Length	4096 tokens	Context window limit
GPU Memory Utilization	0.85	85% of VRAM allocated to model
Max New Tokens	2048	Maximum output length
Temperature	0.1	Low temperature for deterministic extraction
Model Storage	RunPod Network Volume	~15GB model cached at /runpod-volume/models/

3.5.4 Preprocessing Configuration

Technique	**Parameter**	**Value**
Resizing	Max Dimension	2000 pixels
Resizing	Interpolation	INTER_AREA
Denoising	Filter Diameter	5
Denoising	Sigma Color	30
Denoising	Sigma Space	30
Skew Detection (Edge)	Lower Threshold	50
Skew Detection (Edge)	Upper Threshold	150
Skew Detection (Line)	Vote Threshold	100
Skew Detection (Line)	Min Line Length	100
Skew Detection (Line)	Max Line Gap	10
Shadow Removal	Dilate Kernel Size	(7, 7)
Shadow Removal	Median Blur Size	21

Contrast (CLAHE)	Clip Limit	3.0
Contrast (CLAHE)	Tile Grid Size	8
Orientation	Model Name	PP-LCNet_x1_0_doc_ori

3.6 API Specification

Extracts text and layout information from an uploaded file.

Request: multipart/form-data

Field	**Type**	**Required**	**Description**
image	file	Yes	Image (JPG, PNG, BMP, TIFF), PDF, or DOCX file. Max 50MB.
engine	string	No	OCR engine: surya, paddle, hybrid (default), or vlm
preprocessing	string	No	"true" or "false" (default: "true" for OCR engines, ignored for VLM)

Response (200):

```
{
  "success": true,
  "textContent": "SECTION A-A\nCONCRETE SCALE 1:50...",
  "preprocessedImage": "data:image/png;base64,...",
  "preprocessingMetadata": {
    "steps_completed": ["resize", "grayscale", "rotation", "deskew", "border_removal"],
    "rotation_applied": -90.0,
    "original_size": [3000, 2000, 3],
    "final_size": [2000, 1500]
  },
  "results": {
    "layout": {
      "regions": [
        { "bbox": [100, 100, 500, 300], "type": "Paragraph" }
      ]
    },
    "tables": [
      { "bbox": [50, 400, 600, 800], "confidence": 0.95 }
    ],
    "text_lines": [
      {
        "text": "SECTION A-A",
        "bbox": [100, 100, 200, 120],
        "confidence": 0.99,
        "region_type": "Header",
        "page": 1
      }
    ]
  }
}
```

Error Response (400/500):

```
{
  "success": false,
  "error": "Description of the error"
}
```

4. Module: CAD Parser

4.1 Overview

The CAD Parser Service is a dedicated microservice designed to process Computer-Aided Design (CAD) files, specifically DXF (Drawing Exchange Format). It extracts geometric data, layer information, and calculates building compliance metrics required for permit applications. It operates in two distinct modes:

- **Manual Mode**: Extracts geometry from specific, user-selected layers.
- **Automated Mode**: Uses Python logic to automatically detect site boundaries, building footprints, and floors to calculate Building-to-Land Ratio (BTL) and Floor Area Ratio (FAR).

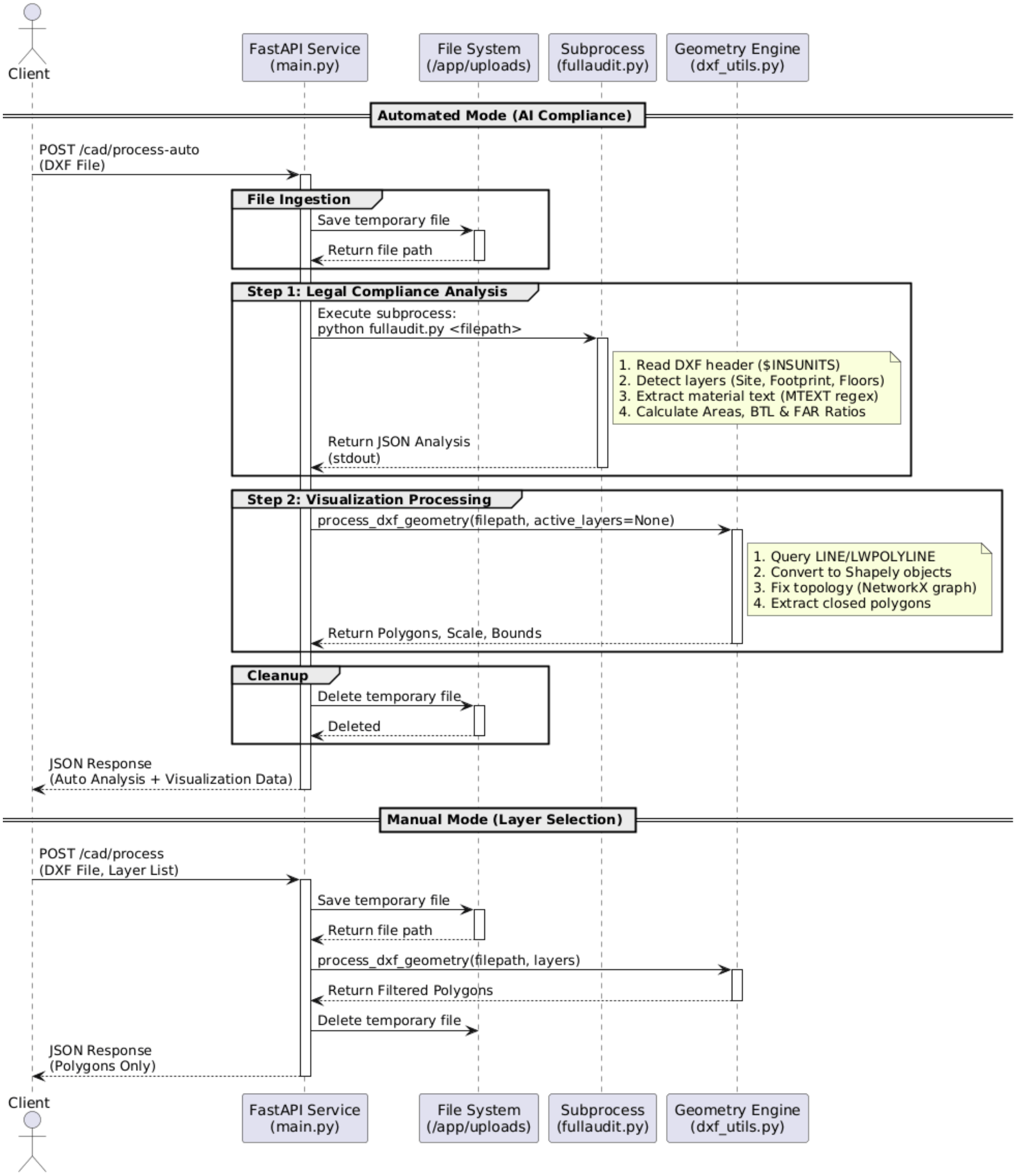
4.2 Technology Stack

- **Runtime:** Python 3.11
- **CAD Engine:** ezdxf (v1.1.0+) - DXF parsing and entity extraction
- **Geometry:** Shapely (v2.0.0+) - Polygon creation and area calculation
- **Topology:** NetworkX (v3.0.0+) - Graph analysis to repair disconnected lines
- **Analysis:** Pandas (v2.0.0+) - Data aggregation and filtering
- **Container:** Docker - Deployment and isolation

4.3 Architecture & Workflow

The service follows a layered architecture separating API handling from core geometric processing and legal auditing.

CAD Parser Service - Workflow Sequence


Client
FastAPI Service (main.py)
File System (/app/uploads)
Subprocess (fullaudit.py)
Geometry Engine (dxf_utils.py)
Automated Mode (AI Compliance)
POST /cad/process-auto (DXF File)
File Ingestion
Save temporary file
Return file path
Step 1: Legal Compliance Analysis
Execute subprocess: python fullaudit.py <filepath>
1. Read DXF header ($INSUNITS)
2. Detect layers (Site, Footprint, Floors)
3. Extract material text (MTEXT regex)
4. Calculate Areas, BTL & FAR Ratios
Return JSON Analysis (stdout)
Step 2: Visualization Processing
process_dxf_geometry(filepath, active_layers=None)
1. Query LINE/LWPOLYLINE
2. Convert to Shapely objects
3. Fix topology (NetworkX graph)
4. Extract closed polygons
Return Polygons, Scale, Bounds
Cleanup
Delete temporary file
Deleted
JSON Response (Auto Analysis + Visualization Data)
Manual Mode (Layer Selection)
POST /cad/process (DXF File, Layer List)
Save temporary file
Return file path
process_dxf_geometry(filepath, layers)
Return Filtered Polygons
Delete temporary file
JSON Response (Polygons Only)
Client
FastAPI Service (main.py)
File System (/app/uploads)
Subprocess (fullaudit.py)
Geometry Engine (dxf_utils.py)

4.4 Component Details

4.4.1 Main Application

The entry point for the service. It handles HTTP requests, file uploads, and lifecycle management.

- **File Handling**: Saves uploads to `/app/uploads` and performs auto-cleanup in a `finally` block.
- **Subprocess**: Executes the `process-auto` logic in a separate process to ensure isolation and stability.

4.4.2 Geometry Engine

Handles the conversion of raw CAD entities into usable GIS-like polygons.

Key Algorithm: Topology Repair CAD drawings often contain "dead ends" or lines that almost touch but don't quite close.

1. **Graph Construction**: Converts lines to a `NetworkX` graph.
2. **Dead End Detection**: Identifies nodes with a degree of 1.
3. **Extension**: Extends dead ends to the nearest geometry within a `1.5` unit tolerance.
4. **Polygonization**: Uses `shapely.ops.polygonize` to find closed loops from the repaired lines.

Unit Conversion Strategy: Based on the `$INSUNITS` header variable:

- `4` (Millimeters) → Scale factor `0.001`
- `6` (Meters) → Scale factor `1.0`
- `0` (Unitless) → Defaults to inches (`0.0254`) or user-defined.

4.4.3 Compliance Auditor

An intelligent parser class `FinalComplianceAuditor` designed for automated legal checks.

Detection Logic:

- **Site Boundary**: Looks for layers containing keywords: `['지적', 'SITE', '대지', 'LND', 'BOUNDARY']`. Defaults to the largest polygon if no keyword matches.
- **Building Footprint**: Looks for layers containing: `['HH', 'FOOTPRINT', '건축면적']`.

- **Floor Detection**: Uses Regex `(B?\d+)(F|층|FLR|FLOOR|ND|ST|RD|TH)` to identify floors (e.g., "1F", "2nd", "B1").
    - *Safety Feature*: Explicitly ignores single-digit layers ("1", "2") as these usually represent AutoCAD colors, not floors.
- **Material Audit**: Scans `MTEXT` entities for keywords like `["마감", "유리", "콘크리트", "THK"]`.

4.5. API Specification

4.5.1 Extract Layers

- **Endpoint**: `POST /cad/layers`
- **Body**: `multipart/form-data` (`file`: .dxf/.dwg)

Response:

JSON

```
{
  "layers": ["0", "A-WALL", "SITE-BNDY", "DIMENSIONS"]
}
```

4.5.2 Manual Processing

- **Endpoint**: `POST /cad/process`
- **Body**:
    - `file`: Binary file
    - `layers`: JSON string (e.g., `["A-WALL", "SITE"]`)
- **Response**: Returns geometric polygons and SVG bounds.

4.5.3 Automated Analysis

- **Endpoint**: `POST /cad/process-auto`
- **Body**: `file`: Binary file

Response:

JSON

```
{
  "polygons": [...],
  "scale": 0.001,
  "bounds": { "min_x": 0, "max_x": 100, ... },
  "auto_analysis": {
    "site_area": 500.0,
    "footprint_area": 300.0,
    "total_floor_area": 600.0,
    "btl": 60.0,
    "far": 120.0,
    "materials_count": 15
  },
  "mode": "automated"

}
```

4.6. CAD Document Standarization

To ensure the automated parser correctly interprets geometry and units, the following global settings must be applied to every DWG/DXF file.

- **Drawing Units:** Millimeters (mm).
- **System Variable:** Set INSUNITS to **4** (Millimeters).
- **Geometry Type:** All areas (Site, Footprint, Floors) must be drawn using **Closed Polylines** (LWPOLYLINE).
- **Prohibited Layer Names:** Do **not** use single digits (1, 2, 3 ... 8) as layer names. These are reserved for AutoCAD colors and will be ignored by the floor detection logic.

4.6.1 Mandatory Layer Naming Convention

The automated auditor uses **Keyword Matching** and **Regex Patterns**. Layer names **must** contain the specific English or Korean keywords listed below to be detected.

A. **Site Boundary (Article 55 Check)**

Defines the legal scope of the land.

- **Required Keyword:** Must contain one of: SITE, BOUNDARY, LND, 대지, 지적
- **Recommended Layer Name:** A-SITE-BNDY
- **Geometry:** Single closed polyline representing the cadastral boundary.

B. Building Footprint (Building-to-Land Ratio)

Defines the area covering the ground, used for the numerator in BCR calculation.

- **Required Keyword:** Must contain one of: FOOTPRINT, HH, 건축면적
- **Recommended Layer Name:** A-AREA-FOOTPRINT or A-HH-FOOTPRINT
- **Geometry:** Closed polyline of the building's exterior wall line at ground level.

C. Floor Area Layers (FAR Calculation)

To calculate Total Floor Area (FAR), layers must strictly follow a numbering pattern detected by the system's Regex.

Naming Pattern: [Any Prefix]-[Number][Suffix]

- **Allowed Suffixes:** F, FLR, FLOOR, 층
- **Allowed Numbers:** B1, B2... 1, 2... 9 (and higher if regex updated)

Floor Level	Standard Layer Name (Recommended)	Alternative Valid Name	Invalid Name (Will Fail)
1st Floor	A-AREA-01F	WALL-1F, 1층-바닥	1 (Ignored as color)
2nd Floor	A-AREA-02F	WALL-2F, 2FLOOR	Level-Two (No digit)
3rd Floor	A-AREA-03F	WALL-3FLR	Third-Floor
Basement	A-AREA-B1F	WALL-B1	Basement

D. Material Specifications (Article 11 Audit)

The system scans TEXT and MTEXT entities for specific keywords. The text content itself is more important than the layer name, but keeping them on a dedicated layer helps organization.

- **Recommended Layer Name:** A-ANNO-MATL or A-ANNO-TEXT
- **Content Requirement:** The text string **must** contain at least one of these keywords to be captured:
  - THK (Thickness)
  - 유리 (Glass)
  - 콘크리트 (Concrete)
  - 마감 (Finish)
  - 단열재 (Insulation)
  - 방수 (Waterproofing)

Example Valid Text: "THK24 복층유리" (Contains 'THK' and '유리')

4.6.2 Quick Reference Table for Architects

Element	Standard Layer Name	Trigger Keyword (In Logic)	Legal Purpose
Site Boundary	A-SITE-BNDY	SITE	Building-to-Land Ratio (Denominator)
Bldg Footprint	A-AREA-FOOTPRINT	FOOTPRINT	Building-to-Land Ratio (Numerator)
1st Floor Area	A-AREA-01F	1F (Regex)	Floor Area Ratio (FAR)
2nd Floor Area	A-AREA-02F	2F (Regex)	Floor Area Ratio (FAR)
Roof	A-ROOF	(None - Excluded)	N/A
Dimensions	A-ANNO-DIM	(None - Excluded)	Visual Check Only

6. Module: Knowledge Base

6.1 Overview

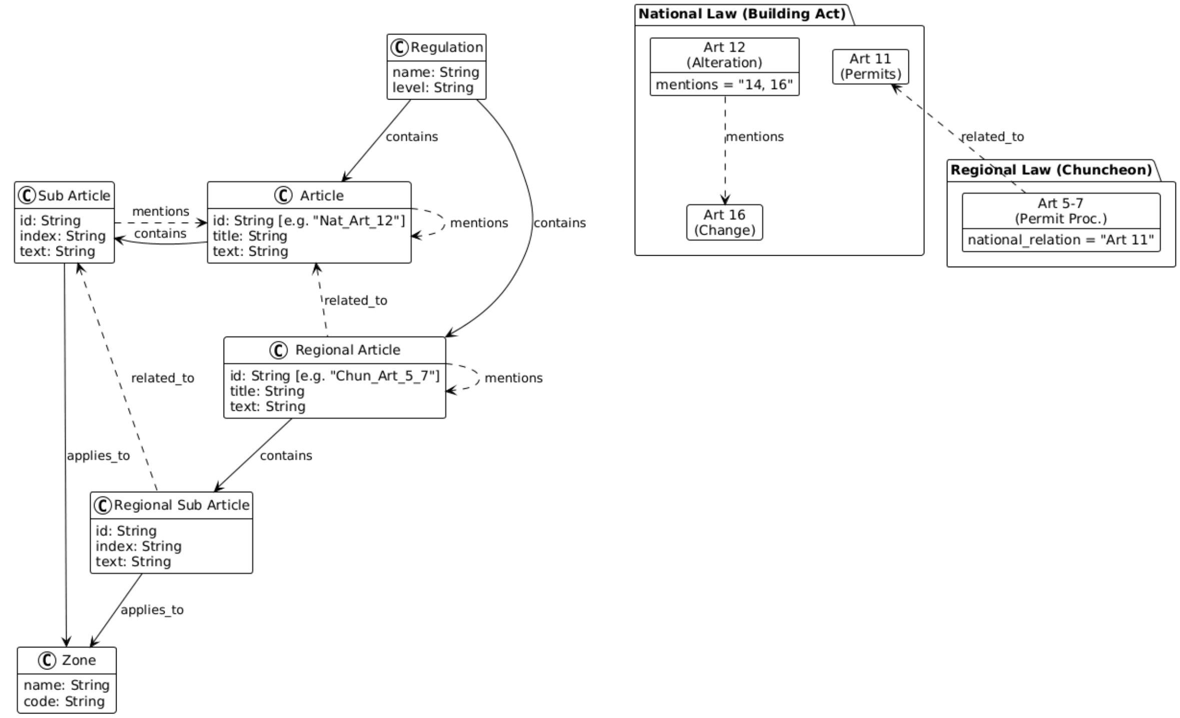
The Legal Compliance Knowledge Base is a graph-based Retrieval-Augmented Generation (RAG) system designed to automate building code compliance checking. It utilizes a **Recursive Tree Structure** to model the depth of legal documents, strictly separating National Standards from Regional Overrides.

6.2 Technology Stack

- **Database: Neo4j** (Graph Database)
- **Backend:** Python (FastAPI).
- **Query Language: Cypher** (Graph queries)

6.3 Knowledge Base Schema

The service splits the workload into two distinct capabilities: **Detection** (identifying where text is) and **Extraction** (reading what the text says).



Node Definitions

A. Authority Nodes

- `Regulation`
  - `name` (String): e.g., "Building Act Enforcement Decree".
  - `level` (String): "National" | "Regional".
  - **Role:** The root container for all articles.

B. National Content Nodes

- `Article`
  - `id` (String): e.g., "Nat_Art_12".
  - `title` (String): e.g., "Alteration of Permitted Matters".
  - `text` (String): Full text content.
- `Sub Article`
  - `id` (String): e.g., "Nat_Art_12_1".
  - `index` (String): e.g., "①", "1.", "(a)".
  - `text` (String): Specific rule text.
  - **Logic Fields (Leaf Nodes Only):** `topic`, `value`, `operator`, `unit`.

C. Regional Content Nodes

- `Regional Article`
  - `id` (String): e.g., "Chun_Art_5_7".
  - `title` (String): e.g., "Permit Application Procedures".
  - `national_relation` (String): "Building Act Art 11" (Text reference).
- `Regional Sub Article`
  - `id` (String): e.g., "Chun_Art_5_7_1".
  - `index` (String): e.g., "1.".
  - **Logic Fields:** `topic`, `value`, `operator`.

D. Filter Nodes

- `Zone`
  - `name` (String): e.g., "General Residential", "Mountainous Territory".
  - `code` (String): e.g., "RES-02".
  - **Role:** Constraint filter.

Source Node	Relationship	Target Node	Description
Regulation	**contains**	Article / Regional Article	Hierarchy: Document owns the Article.
Article	**contains**	Sub Article	Hierarchy: Level 0 → Level 1.
Regional Article	**contains**	Regional Sub Article	Hierarchy: Level 0 → Level 1.
Regional Article	**related_to**	Article	**Explicit Link:** Regional rule implements/overrides National rule.
Article	**mentions**	Article	**Citation:** Text explicitly cites another article.
Sub Article	**applies_to**	Zone	**Filter:** Logic applies to specific location only.

6.4 RAG Details

The system employs a **Multi-Strategy Retrieval Engine** that combines the semantic understanding of Vector Search with the structural precision of Graph Queries.

6.4.1 Retrieval Strategy

To ensure zero-result queries are minimized, the system attempts three distinct retrieval methods in a fallback sequence:

1. **Primary: Vector Similarity Search**
   - **Mechanism:** Converts user query into a 1536-dimensional vector using `text-embedding-004`.
   - **Target:** Queries the Neo4j Vector Index.
   - **Metric:** Cosine Similarity (0–1 scale).
   - **Use Case:** Conceptual queries (e.g., "What are the rules for open spaces?").

2. **Secondary: LLM-Generated Cypher (Dynamic)**
   - **Trigger:** Activates if Vector Search returns low-confidence scores.
   - **Mechanism:** The LLM receives the Graph Schema and the user query, generating a valid Cypher query string.
   - **Target:** Direct Graph traversal.
   - **Use Case:** Complex relational queries (e.g., "Find all articles in Chuncheon ordinance that mention 'Safety'").

3. **Fallback: Deterministic Lookup**
   - **Trigger:** Activates if both semantic and graph queries fail.
   - **Mechanism:** Regex extraction of article numbers or specific keywords.
   - **Target:** `MATCH (n) WHERE n.id CONTAINS...`
   - **Use Case:** Specific navigational queries (e.g., "Show me Article 52").

6.4.2 LLM Integration Logic

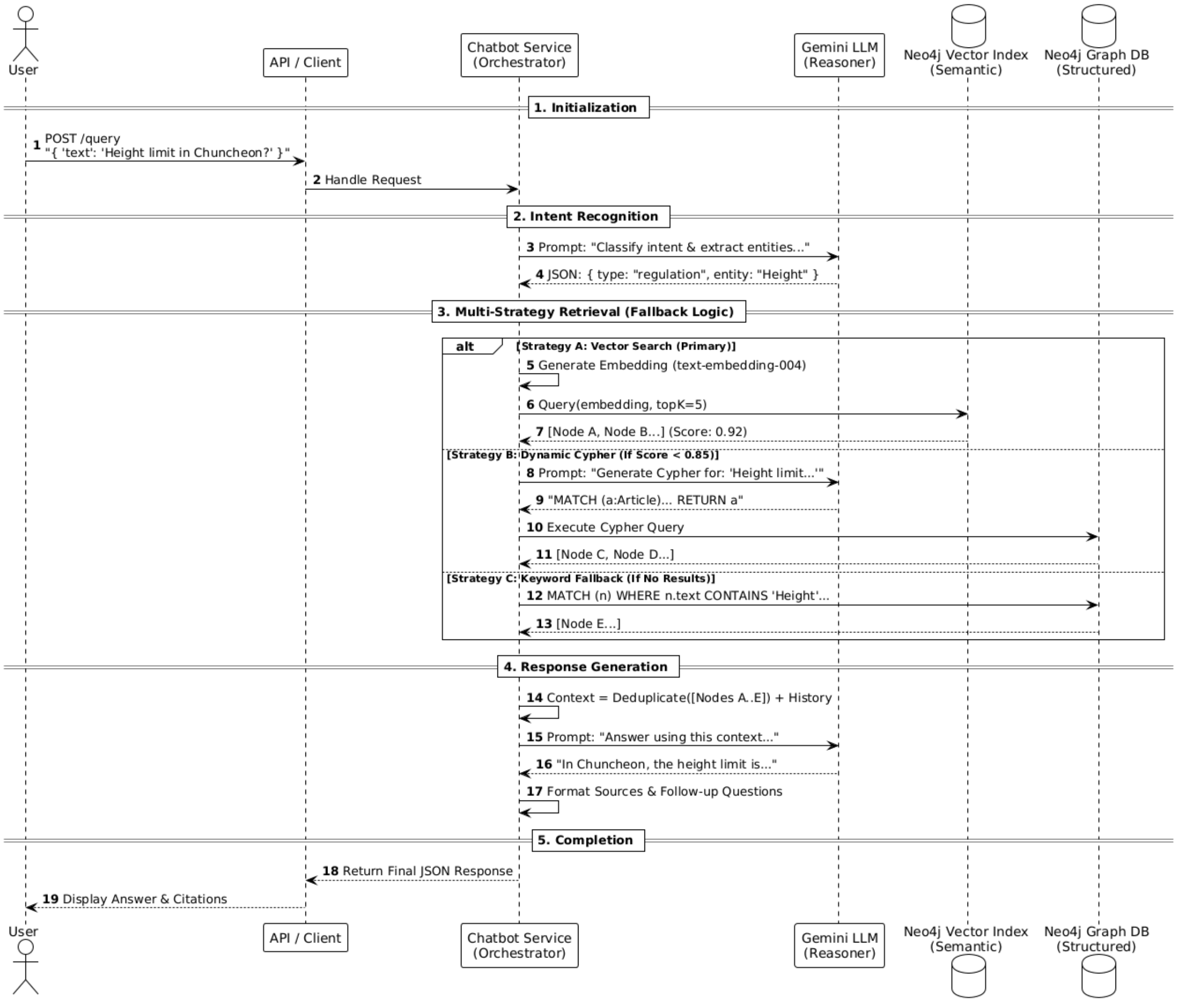
The system utilizes Google **Gemini** models for both embedding and reasoning.

Component	Model	Configuration	Purpose
Reasoner	gemini-2.5-flash-lite	Temp: 0.3, TopK: 30	Intent classification, Cypher generation, Final answer synthesis.

Embedder	text-embedding-004	1536 Dimensions	Semantic indexing of legal text.

The Prompt Chain:

1. **Intent Recognition:** Classifies input into `greeting`, `capabilities`, or `regulation_query`. Extracts entities (e.g., `{ articleNumber: "52" }`).
2. **Query Generation:** (If needed) Generates specific Cypher or Vector parameters.
3. **Synthesis (RAG):** "Using these {8 Retrieved Articles} as context, answer {User Query}. Explicitly cite National vs. Regional differences."
4. **Engagement:** Generates 3 context-aware follow-up questions.


User
API / Client
Chatbot Service (Orchestrator)
Gemini LLM (Reasoner)
Neo4j Vector Index (Semantic)
Neo4j Graph DB (Structured)
1. Initialization
1 POST /query "{ 'text': 'Height limit in Chuncheon?' }"
2 Handle Request
2. Intent Recognition
3 Prompt: "Classify intent & extract entities..."
4 JSON: { type: "regulation", entity: "Height" }
3. Multi-Strategy Retrieval (Fallback Logic)
alt
[Strategy A: Vector Search (Primary)]
5 Generate Embedding (text-embedding-004)
6 Query(embedding, topK=5)
7 [Node A, Node B...] (Score: 0.92)
[Strategy B: Dynamic Cypher (If Score < 0.85)]
8 Prompt: "Generate Cypher for: 'Height limit...'"
9 "MATCH (a:Article)... RETURN a"
10 Execute Cypher Query
11 [Node C, Node D...]
[Strategy C: Keyword Fallback (If No Results)]
12 MATCH (n) WHERE n.text CONTAINS 'Height'...
13 [Node E...]
4. Response Generation
14 Context = Deduplicate([Nodes A..E]) + History
15 Prompt: "Answer using this context..."
16 "In Chuncheon, the height limit is..."
17 Format Sources & Follow-up Questions
5. Completion
18 Return Final JSON Response
19 Display Answer & Citations
User
API / Client
Chatbot Service (Orchestrator)
Gemini LLM (Reasoner)
Neo4j Vector Index (Semantic)
Neo4j Graph DB (Structured)

6.5 API Specifications

Core Endpoints

1. **Query Chatbot**
    - POST `/api/chatbot/query`
    - Body: `{ "query": "What are the height limits?", "sessionId": "abc-123" }`

      Response:
      JSON
      ```
      {
        "message": "Height limits are determined by...",
        "sources": [{ "regulation": "Building Act", "articleId": "52" }],
        "suggestedQuestions": ["What about mountainous zones?"]
      }
      ```

2. **Session Management**
    - GET `/api/chatbot/history/:sessionId` - Retrieve last 20 exchanges.
    - DELETE `/api/chatbot/history/:sessionId` - Clear context.

3. **Knowledge Graph Exploration**
    - GET `/api/chatbot/article/:articleId` - Fetches specific node details and its immediate relationships (Visualizer support).

4. **Administration**
    - POST `/api/chatbot/admin/reingest` - Triggers the ETL pipeline to rebuild the Graph and Vector indices from source files.