

Most Essential UML Diagrams for This Project (Mandatory)

These diagrams ensure that every module—scraping, cloud backend, local backend, AI pipeline, admin system, tenant UI, multi-tenant DB—is clearly understood.

1. Use Case Diagram (HIGH PRIORITY)

Covers:

- Tenants
- Group Observers
- Group Admins
- System Admin
- Scraper Engine
- AI Engine / RAG Pipeline

Why needed:

It defines *all system behaviors* based on the features outlined in the project proposal and architecture files (review collection, analysis, response generation, dashboard insights, group comparisons, competitor management).

You should draw **separate use-case diagrams** for:

- Tenant app (local + cloud interactions)
 - Admin system
 - Scraper subsystem
 - AI pipeline subsystem
-

2. High-Level System Architecture Diagram (ALREADY IN YOUR DOCS, but refine it)

Your current architecture diagram (seen in *Architecture Report, page 5*) shows the Local App \leftrightarrow Cloud App communication.

You must convert this into a **formal UML Deployment Diagram**.

This includes:

- Nodes: Tenant Local Machine, Cloud Backend, Database Server, Qdrant Server, Scraper Container, Admin Frontend
 - Artifacts: FastAPI app, Electron app, Embedding service, Scheduler, SQL DB, Qdrant DB
-

3. UML Deployment Diagram

This is essential because your system has:

- A local environment (Electron + FastAPI + MiniLM + SQLite)
- A cloud environment (FastAPI backend + Playwright scrapers + Qdrant + MS SQL + Celery + Redis)
- Admin frontend & backend

Deployment diagram shows:

- Which components run on which machines
 - How they communicate
 - Network boundaries, queues, APIs, DBs
-

4. UML Component Diagram

Shows modular separation of:

- Local Frontend (React)
- Local Backend (FastAPI)
- Cloud Backend API modules
- Admin Backend
- Admin Frontend
- Scraper Engine
- AI pipeline components (Chunker, Embedding Engine, RAG Engine, LLM Gateway)

Why needed:

Your system has **many interacting modules**; this diagram clarifies the dependency boundaries and communication channels.

5. Class Diagram (ABSOLUTELY REQUIRED — Based on Your SQL Schema)

The SQL schema you provided defines **the full domain model**.

So the class diagram should include:

Core Classes

- Tenant
- Plan
- Group, Group_Tenant, Sub_Group
- Domain
- Organization

- Source, Organization_Source, Source_Link
- Raw_Review
- Processed_Review
- Review_Category
- Review_Type
- Knowledgebase_Item
- Entity
- Embedding_Job
- Review_Analysis

This is crucial because:

- Your DB schema is rich and multi-relational
 - It informs your backend models and ORM design
 - It clarifies relationships to developers before coding starts
-

6. Sequence Diagrams (MULTIPLE REQUIRED)

Given your system's complexity, you need at least 5 key sequence diagrams:

(1) Review Scraping Sequence

Scraper → Cloud API → raw_review → processed_review → embedding_job → Qdrant
Matches data flow on *Architecture Report page 10*.

(2) Tenant Opening Dashboard

Local App → Local FastAPI → Cloud Backend → SQL → Local Cache → Dashboard UI

(3) AI Reply Generation

Tenant UI → Cloud API → RAG Retrieval → LLM Gateway → Review Analysis → Response returned
Matches AI Pipeline steps on *page 10* (MiniLM → embedding → RAG → LLM).

(4) Group Comparison Flow

Tenant → Cloud API → Group Data → Analytics Engine → Charts

(5) Competitor Evaluation Flow

Tenant → Cloud → Domain/Competitor Analytics → Redis cache → UI
(Referenced in Proposal V3 pages 7–8 regarding domain & competitor analytics).

Sequence diagrams ensure developers understand micro-level interactions.

7. Activity Diagrams (Recommended)

Create activity diagrams for the following flows:

(1) User Login + Organization Selection

Matches your described tenant onboarding workflow.

(2) Data Ingestion Pipeline

Scraping → Cleaning → Storage → Processing → Embedding → Sync

Based on the step-by-step sequence shown in Architecture Report pages 10–11.

(3) AI Analysis Workflow

Raw review → Local model → Embedding → Qdrant retrieval → LLM → Store → Display

(4) Group Invite + Join Flow

Tenant joins via invite link, admin approval, role assignment.

8. State Machine Diagrams (Optional but Useful)

Use for objects whose state changes frequently:

Best candidates:

- Review object (Raw → Processed → Embedded → Analyzed → Replied)
 - Embedding job (Pending → Running → Completed → Failed)
 - Organization scraper state (Active → Paused → Error)
-

9. ER Diagram (Already covered by your SQL schema but still required)

This is essentially **your DB schema visualized**, which you have already provided but should redraw with crow's feet notation to include:

- All FK constraints
- Cardinality
- Multi-tenant relationships

Reflect tables shown in your SQL script in structured ER format.