Aldeia: Data Strategy Document

# Overview

This document outlines the foundational data architecture and database schema for the Aldeia platform, which supports post-disaster property insights, damage assessments, and rebuild workflows. It provides a scalable structure to support mapping interfaces, user interactions, property valuations, user preferences, architect/contractor engagement, and AI-powered recommendations.

#### **1. AI, LLM & System Intelligence Infrastructure**

To support continuous learning and system robustness, Aldeia should integrate:

#### **Machine Learning Feedback Loops**

* Feedback on design matches, zoning summaries, and insurance calculators is stored in feedback and model\_training\_signals
* Labeled examples power periodic retraining of core models (recommendation, classifier, regression)

#### **LLM Chat Assistant**

* Logs captured in chat\_interactions
* Integrates with vector database (for RAG) or rules (for cost/zoning)
* Intent and topic analytics used for platform improvements

#### **Error Handling & System Logging**

* All exceptions, warnings, and performance issues are logged to system\_logs
* User-facing alerts displayed via notifications

#### **Key AI Modules**

|  |  |
| --- | --- |
| **Module** | **Description** |
| style\_recognition\_model | Yolov8 trained style recognition model |
| design\_match\_model | Random forests algorithm to match lot details and user inputs to design catalog |
| faq\_llm\_assistant | LLM chatbot with RAG architecture & brand-voice fine-tuning |

#### **2. Key Data Entities**

* **Property**: Core parcel and residential details.
* **Damage Assessment**: Post-disaster structural status.
* **Valuation Estimate**: Estimated land and building value.
* **Parcel Geometry**: Geospatial polygon boundaries and overlays.
* **User Actions**: Events triggered by users on the platform.
* **Reports**: Downloadable insights for users.
* **Contact Messages**: Submitted forms and inquiries from users.
* **User Profile**: Stores identity and login status.
* **Rebuild Preferences**: Captured inputs for matching house designs.
* **Upgrade Options**: User-selected rebuild enhancements or additions.
* **Budget & Insurance**: Preferences for rebuild spending and coverage planning.
* **Style Preferences**: Original design details and visual style data.
* **Design Inspiration**: User-uploaded or selected images that guide the AI matching process.
* **Design Matches**: AI-recommended rebuild options personalized to user inputs.
* **Design Details**: Information about selected designs, their architects, and cost/certification breakdown.
* **Architects**: Profiles and activity of participating licensed architects.
* **Contractors**: Profiles and availability of partner contractors.
* **Consultation Records**: Logs of architect-client project activity.
* **Planning & Zoning Rules**: Governing regulations for rebuild feasibility.
* **Calculation Rules**: System rules that determine budget bands, incentives, or eligibility logic.
* **Imagery Metadata**: Processed aerial, 3D, or street view data.
* **Feedback**: Platform-wide user feedback tied to data quality, features, or recommendations.
* **Rebuild Recommendations**: Finalized AI-assisted rebuild plan, compiled from matches and user input.
* **Insurance Estimates**: Calculated estimates of coverage needed for reconstruction.
* **LLM Chat History:** Logs of user conversations with the AI assistant.
* **Error Logs & Notifications:** Real-time tracking of system messages and user-facing alerts.
* **Model Training Signals:** Feedback-derived annotations for AI improvement.

#### **3. Core Database Tables**

**A. LOT DETAILS**

#### properties

id, apn, address, city, state, zip\_code, use\_type, lot\_size\_sqft, bedrooms, bathrooms, building\_sqft, created\_at

#### damage\_assessments

id, property\_id, assessment\_status, damage\_percent, assessed\_by, assessed\_at

#### valuations

id, property\_id, total\_value, land\_value, improvement\_value, valuation\_date, source

#### parcel\_geometry

id, property\_id, geometry\_type, coordinates, map\_layer\_type, source

#### imagery\_metadata

id, property\_id, image\_type, capture\_date, source, file\_url, resolution, orientation, processed (BOOLEAN)

#### reports (optional)

id, property\_id, report\_url, status, generated\_at

**B. USER INPUTS & UPLOADS**

#### user\_profiles

id, name, email, is\_registered, created\_at

#### user\_actions

id, user\_id, property\_id, action\_type, timestamp

#### contact\_messages

id, full\_name, email, message, submitted\_at

#### rebuild\_preferences

id, user\_id, lot\_id, bedroom\_count, bathroom\_count, story\_count, preferred\_style, max\_budget, submitted\_at

#### upgrade\_options

id, rebuild\_pref\_id, option\_name, is\_selected

#### budget\_insurance

id, rebuild\_pref\_id, budget\_min, budget\_max, insurance\_coverage\_percent, efficiency\_incentives, submitted\_at

#### style\_preferences

id, rebuild\_pref\_id, image\_url, architecture\_character, roof\_type, detected\_by, confirmed\_match, submitted\_at

#### design\_inspiration

id, rebuild\_pref\_id, image\_url, image\_type, selected\_by\_user, submitted\_at

**C. PARTNER & DESIGN CATALOG**

#### design\_details

id, design\_id, description, floorplan\_url, image\_url, video\_url, estimated\_cost\_min, estimated\_cost\_max, insurance\_estimate, preapproved\_cert, electric\_savings\_note, architect\_name, architect\_org, contact\_email

#### architect\_profiles

id, name, organization, email, certifications, availability\_status, login\_credentials

#### contractor\_profiles

id, name, organization, email, specialties, availability\_status, login\_credentials

#### consultations

id, user\_id, architect\_id, design\_id, status, calendar\_event\_url, notes, created\_at

**D. RECOMMENDATIONS**

#### design\_matches

id, rebuild\_pref\_id, design\_id, design\_name, design\_provider, main\_style, sub\_styles, bedroom\_count, bathroom\_count, sqft, match\_score, image\_url, suggested\_at

#### rebuild\_recommendations

id, rebuild\_pref\_id, selected\_design\_id, ai\_notes, user\_notes, final\_choice\_confirmed (BOOLEAN), submitted\_at

#### insurance\_estimates

id, property\_id, design\_id, estimated\_coverage (FLOAT), provider\_notes, discount\_applied (BOOLEAN), submitted\_at

#### feedback

id, user\_id, source\_table, source\_id, feedback\_text, sentiment, submitted\_at

**E. RULES**

#### zoning\_rules

id, zoning\_code, permitted\_use, max\_stories, setback\_rules, lot\_coverage\_limit, source\_jurisdiction, updated\_at

#### calculation\_rules

id, rule\_type, trigger\_condition, formula\_description, output\_variable, version, created\_at

**F. SYSTEM LEVEL**

#### system\_logs

id, user\_id, error\_type, message, severity, timestamp, route\_or\_screen

#### notifications

id, user\_id, type, message, is\_read, created\_at

#### model\_training\_signals

id, source\_table, source\_id, label\_type, label\_value, source\_user\_id, timestamp

#### **4. Analytics & ML Stack**

* **Data Warehouse**: Supabase (for integration with parcel + zoning overlays)
* **BI Tools**: Freely available for internal dashboards
* **LLM Integration**: miniLM via Hugging Face, chat logs to chat\_interactions
* **Vector DB**: FAISS or Weaviate
* **Model Monitoring**: Use model\_training\_signals, feedback, and rebuild\_recommendations to generate retraining datasets

# LLM Chatbot & RAG Architecture

This section extends the Database Strategy to fully support integration of a Retrieval-Augmented Generation (RAG) pipeline powering the LLM-based chatbot. It introduces new data structures, privacy enhancements, and architectural amendments aligned with responsible AI design and user consent.

#### **1. Data Tables**

#### chat\_interactions

id, user\_id, session\_id, user\_input, llm\_response, intent\_detected, source\_context, timestamp

#### source\_documents

(Stores original, indexed files used by the RAG pipeline for retrieval)

id, title, doc\_type, source\_url, jurisdiction, version, uploaded\_by, upload\_date

#### vector\_store\_metadata

(Metadata and vector embeddings for each chunked passage)

id, source\_document\_id, chunk\_id, embedding\_vector, page\_number, chunk\_text, created\_at

#### consent\_log

(Tracks user consent related to chatbot use, training, and personalization)

id, user\_id, consent\_type, consent\_status, timestamp

#### **2. Chat Log Structure**

Updates to the existing chat\_interactions table:

* retrieved\_doc\_ids (ARRAY)
* retrieval\_scores (ARRAY of FLOAT)
* response\_type (ENUM: 'rag', 'fallback', 'faq')
* source\_routing\_notes (TEXT)

#### **3. RAG Data Pipeline Structure**

* **Document Ingestion:** PDFs, HTML guides, links?
* **Chunking:** Split into ~500-token sections
* **Embedding:** Use OpenAI/Groq/Instructor model for vector generation (miniLM in our case???)
* **Storage:** Indexed via FAISS or Weaviate backend
* **Query Flow:**
  + User sends natural language input
  + Query is embedded and matched with top-k vectors
  + Matched text passed to LLM with prompt formatting
  + Response stored with document traceability

# Data Privacy and Security

#### **1. PII & Data Privacy Compliance Strategy**

#### **Principles**

* Data minimization (only collect what is essential)
* Purpose limitation (use data strictly for user-facing services)
* Privacy by Design and Default

#### **Actions**

* **Anonymize user data** upon collection wherever possible (e.g., hashed IDs for analytics).
* **Tokenize or mask sensitive fields** (e.g., email, name, address) in storage and internal tools.
* **Role-based access control (RBAC)** to limit access to PII fields.
* **Secure Data Transmission**: TLS 1.3 for all web and backend communications.
* **Audit Trails** for all admin access to PII fields.

#### **For Chatbot RAG:**

* Chunking service pre-screens for PII using regex + LLM filters
* All chat\_interactions redact names/emails before storage
* source\_documents tagged with access scope: public, internal, admin-only
* Weekly audit pipeline scans and purges sensitive info

#### **2. PII Masking at Account Creation & Storage**

#### **Masked Field Examples**

| **Field** | **Masking Strategy** |
| --- | --- |
| Name | Partial masking (e.g., H\*\*\*\* Amin) |
| Email | Hashed + masked display (e.g., h***@g***.com) |
| Phone Number | Encrypted at rest + masked view (e.g., +1 (\*\*\* ) \*\*\*-7890) |
| Address | Geohash or zip-only storage for analytics |

#### 

#### **Implementation**

* Use database encryption functions and masking views in PostgreSQL/MySQL.
* Set frontend display rules that use masked fields.

#### **3. Chatbot Safety & User Data Protection**

#### **Consent Design in Front-End**

* Checkbox at start of chat: *“Do not share personal details in this chat. By continuing, you agree to anonymized use for learning.”*

#### **Chatbot Controls**

* **LLM Prompt Filtering**: Strip or reject any prompts that contain name, email, or address patterns.
* **Contextual Anonymization**: Replace input PII with placeholders using regex before sending to LLM.
* **No Context Memory**: Default stateless interactions unless user logs in and opts in.
* **Audit Logging**: Log each LLM interaction with timestamp and metadata (not user content).

#### **4. Responsible AI & Controls for Learning from Data**

#### **Feedback-Informed Learning**

* **User Feedback Table** (already in schema) used for retraining only after PII is stripped.
* **Feedback-Only Learning** - Train recommendation models only on feedback table records and design\_matches selections. - Apply scoring algorithms that exclude raw chat content.
* **Vectorized embeddings** generated from abstracted user traits, not full identities.
* **Anonymized Training Pipelines** - Strip all user identifiers before creating model training datasets. - Use synthetic or aggregated embeddings for training logic.

#### **For Chatbot RAG:**

* Feedback captured in feedback table (source = ‘chatbot’)
* Negative feedback flags hallucinated or misaligned responses
* Labeled logs stored in model\_training\_signals for prompt refinement

#### **Guardrails for ML Models**

* Drift detection when recommendation distribution changes suddenly
* Abuse detection for repeated abnormal queries
* Sandbox retraining before deployment

#### **5. Additional Safeguards**

* **PII Retention Policy**: Define automatic deletion windows (e.g., 180 days for unused accounts).
* **Redaction Pipeline**: Scheduled task that checks and removes residual PII from logs or exports.
* **LLM Fine-Tuning Rules**: Never include raw chat transcripts with personal data in training sets.

#### **6. Access Control & Compliance**

**RBAC Implementation** - Define roles: User, Architect, Contractor, Admin, ML Analyst. - Restrict email, notes, insurance\_estimates to permitted roles only.

**Consent Logging** - Store time-stamped user consent records on data use. - Enable opt-in for chatbot training and opt-out for full deletion.

**GDPR/CCPA Toolkit** - Build a data export function triggered from user dashboard. - Implement deletion cascade for user account removal requests. - Maintain access logs for all data views or modifications.

#### **7. System Monitoring & Incident Response**

**Audit Trails** - Log access to all tables containing PII or feedback. - Review logs weekly for unusual access patterns.

**Breach Protocol** - Establish a data breach response checklist (notification, patch, review). - Include an SLA for breach response based on severity.

# Governance & Access Framework

* **Data Classification**:
  + PII: email, name, address
  + Sensitive but non-PII: match\_score, insurance\_estimate
  + Public/analytics-safe: sub\_styles, feedback\_text
* **Retention Policy**:
  + PII removed or anonymized after 6 months of inactivity
  + Logs purged after 12 months unless flagged for audit
* **Consent Logging**:
  + All opt-ins (chatbot, data usage) timestamped in a consent\_log table (planned)
* **For Chatbot RAG:**
  + RAG responses always include traceable citations (retrieved\_doc\_ids)
  + Users must opt in to personalization or log-based improvements (via consent\_log)
  + RAG inputs and outputs excluded from model fine-tuning unless explicitly consented

# APPENDIX: Fine-tuning the LLM AI Model

This section outlines the strategy to fine-tune the MiniLM model for Aldeia’s chatbot, ensuring responses reflect the brand’s tone: empathetic, action-oriented, and disaster-recovery aware. The strategy includes dataset preparation, tone modeling, Hugging Face integration, and data governance enhancements.

#### **1. Objectives**

* Adapt MiniLM to Aldeia’s brand voice
* Improve retrieval relevance in RAG pipeline
* Enable semantic search aligned with user emotion and rebuild intent
* Ensure responses are technically correct and emotionally sensitive

#### **2. Brand Tone & Domain Goals**

* **Empathetic**: Supportive tone toward disaster-affected users
* **Directive**: Clear next steps and process guidance
* **Context-aware**: Understand architectural, zoning, and rebuild terms

#### **3. Dataset Design**

#### training\_prompts

Sample training schema for prompt-response pairs:

id, prompt\_text, response\_text, tone\_label, domain\_tag, created\_by, created\_at

#### Labeling Guidelines

* tone\_label: empathetic, technical, directive, neutral
* domain\_tag: zoning\_law, design\_selection, insurance, budgeting, user\_support

#### **4. Fine-Tuning MiniLM (Hugging Face)**

MiniLM is an encoder model, ideal for sentence embeddings. Use the sentence-transformers library for fine-tuning.

#### Python Training Script

from sentence\_transformers import SentenceTransformer, InputExample, losses  
from torch.utils.data import DataLoader  
import pandas as pd  
  
# Load and preprocess training data  
df = pd.read\_csv('training\_prompts.csv')  
train\_examples = [  
 InputExample(texts=[row['prompt\_text'], row['response\_text']])  
 for \_, row in df.iterrows()  
]  
  
# Create dataloader  
dataloader = DataLoader(train\_examples, shuffle=True, batch\_size=16)  
  
# Load pre-trained MiniLM  
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')  
  
# Define loss and train  
train\_loss = losses.CosineSimilarityLoss(model)  
model.fit(  
 train\_objectives=[(dataloader, train\_loss)],  
 epochs=3,  
 warmup\_steps=100  
)  
  
# Save fine-tuned model  
model.save('./aldeia-minilm-v1')

#### **5. Deployment**

* Save final model to Hugging Face hub or private registry
* Update vector embedding service in RAG to use new model checkpoint
* Add model\_version tracking in logs

#### fine\_tuning\_logs

id, model\_version, dataset\_version, training\_date, accuracy\_metrics, notes

#### **6. Feedback Loop Enhancements**

* Extend feedback table:

+ tone\_match\_rating (INT), semantic\_fit\_rating (INT)

* Use feedback to trigger model updates or prompt tuning

#### **7. Governance & Privacy**

* Anonymize training data
* Log consent in consent\_log if using chat history
* Label dataset sources in metadata
* Allow user opt-out from future training use