# LLM-SRE Site: Directory Structure, Wiring Diagram, and Threat Model

This document captures a practical reference architecture for the llm-sre-site project: how the repo is organized, how UI/infra/backend are wired in AWS, and a lightweight threat model focused on the /api/runbooks/ask RAG flow.

## 1) Recommended Repository Directory Structure

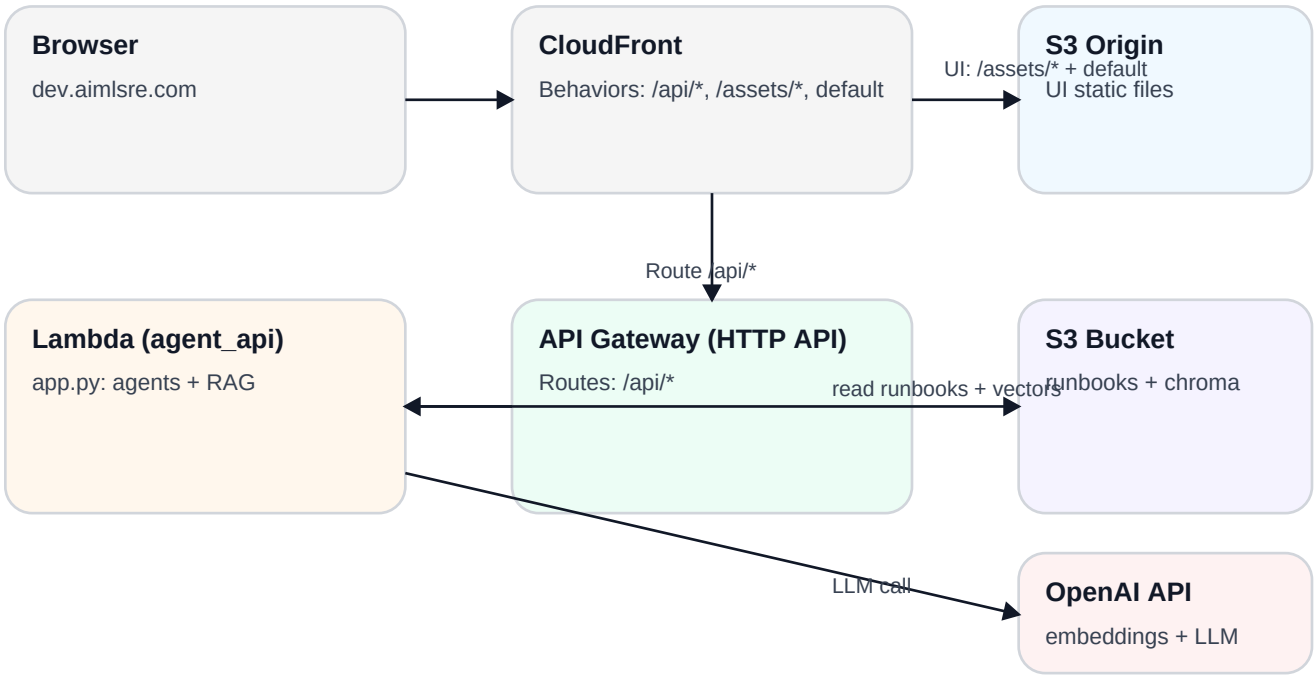
```
llm-sre-site/
├── ui/                              # Vite + React frontend (static)
│   ├── src/
│   │   ├── components/
│   │   │   └── AskRunbooks.jsx          # RAG UI (question -> /api/runbooks/ask -> answer + sou
│   │   ├── pages/
│   │   │   ├── Agents.jsx               # location-based agents (weather/travel)
│   │   │   ├── Rag.jsx                  # RAG tab (wires to /api/runbooks/ask)
│   │   │   └── Runbooks.jsx             # runbook list / UX helpers
│   │   ├── lib/
│   │   │   └── api.js                   # apiGet/apiPost helpers; JSON parsing + error handling
│   │   └── App.jsx                      # tab shell + tabs
│   ├── public/
│   ├── package.json
│   └── vite.config.js
│
├── services/
│   ├── agent_api/
│   │   ├── app.py                       # Lambda handler (agents + runbooks/ask)
│   │   ├── requirements.txt             # openai + chroma + sqlite shim deps
│   │   └── Dockerfile                   # Lambda container image build
│
├── modules/
│   ├── agent_api/
│   │   ├── main.tf                      # Lambda (Image), API Gateway v2 routes, IAM, CORS
│   │   ├── variables.tf
│   │   └── outputs.tf
│
├── scripts/
│   ├── index_runbooks_chroma.py       # builds local chroma_store/ from PDFs + embeddings
│   └── (optional) runbook_manifest.py # incremental diffing + dry-run (hash/etag)
│
├── infra/                             # root Terraform (calls modules/*)
│   ├── main.tf
│   ├── env/dev.tfvars
│   └── ...
│
└── README.md
```

**Notes**

- Keep secrets (OpenAI key) out of the repo. Use Lambda environment variables or AWS Secrets Manager.
- Keep the Chroma index as an artifact in S3 under **knowledge/vectors/<env>/chroma/** and download to **/tmp** at runtime.
- scripts/ is a good location for the indexing job; run it locally or in CI, then upload the resulting chroma_store/ to S3.

## 2) End-to-End Wiring: UI -> CloudFront -> API Gateway -> Lambda -> S3/OpenAI

UI is served from an S3 static site origin behind CloudFront. API traffic is routed to an API Gateway v2 HTTP API origin. Lambda handles /api/agents, /api/agent/run, and /api/runbooks/ask (RAG).




## Key AWS resources

| Layer | Primary resource | Purpose / Notes |
|---|---|---|
| Edge | CloudFront Distribution | Routes UI + API; ordered behaviors: /assets/* -> S3, /api/* -> API GW, default -> SP |
| UI origin | S3 bucket (static website) | Hosts Vite build output (dist/) |
| API | API Gateway v2 (HTTP API) | Routes: GET /api/health, GET /api/agents, POST /api/agent/run, POST /api/runbook |
| Compute | Lambda (container image) | services/agent_api Docker image includes chromadb + sqlite shim |
| Knowledge | S3 bucket (runbooks + vectors) | knowledge/runbooks/*.pdf + knowledge/vectors/<env>/chroma/* |
| LLM | OpenAI API | Embeddings + Responses API for answer generation |

## 3) RAG Flow (Runbooks Q&A;)

1. User enters a question in the UI (AskRunbooks component).
2. UI POSTs JSON to /api/runbooks/ask via CloudFront -> API Gateway -> Lambda.
3. Lambda downloads Chroma store from S3 to /tmp/chroma_store (cold start or cache-miss).
4. Lambda creates an embedding for the question (OpenAI embeddings).
5. Lambda queries Chroma for top_k chunks and generates an answer using only the excerpts.
6. UI renders answer + sources (file + chunk).

## Mermaid sequence diagram (copy into your docs/wiki)

````
```mermaid
sequenceDiagram
  autonumber
  participant U as User (Browser)
  participant CF as CloudFront
  participant APIGW as API Gateway (HTTP API)
  participant L as Lambda (agent_api)
  participant S3 as S3 (runbooks + vectors)
  participant OAI as OpenAI API

  U->>CF: POST /api/runbooks/ask {question, top_k}
  CF->>APIGW: Forward /api/* behavior
  APIGW->>L: Invoke Lambda proxy
  L->>S3: Download vectors prefix to /tmp (if needed)
  L->>OAI: Create embedding(question)
  L->>L: Chroma query top_k
  L->>OAI: Generate answer from excerpts
  L-->>APIGW: {answer, sources[]}
  APIGW-->>CF: Response
  CF-->>U: Render answer in UI
```
````

## 4) Threat Model (Practical, lightweight)

Scope: public site + API endpoints. Focus on protecting credentials, controlling cost, and preventing data exfiltration or prompt injection via untrusted content.

| Threat | Example | Mitigation (recommended) |
|---|---|---|
| Secrets leakage | OpenAI API key exposed in repo/UI, logs, or client-side code. | Keep OPENAI_API_KEY server-side only. Store in Lambda env or Secrets Manager; never bundle into UI. Scrub logs. |
| Prompt injection via runbooks | Malicious content inside PDFs attempts to override instructions. | Use strict system prompt: answer only from excerpts. Strip/limit context length; consider corpus allowlist. |
| Data exfiltration | Model outputs sensitive runbook details to unauthorized users. | Add AuthN/AuthZ if needed (Cognito/JWT). Rate limit/WAF. Consider redaction for secrets in runbooks. |
| Cost abuse / DoS | Attackers spam /api/runbooks/ask to burn tokens and Lambda time. | Throttle (API GW), WAF rate-based rules, top_k cap, max tokens, and per-IP limits. |
| S3 abuse / path traversal | Attacker forces Lambda to download unexpected S3 objects. | Do not accept arbitrary S3 keys from client for RAG. Keep vectors prefix fixed in env; validate inputs. |
| Supply chain risks | Compromised dependencies in container image. | Pin versions, scan ECR images, enable Dependabot, and restrict outbound egress where possible. |

## Mermaid data-flow diagram (DFD-lite)

````
```mermaid
flowchart LR
  U[User Browser] -->|HTTPS| CF[CloudFront]
  CF -->|/assets/*, default| S3UI[S3 UI Bucket]
  CF -->|/api/*| APIGW[API Gateway HTTP API]
  APIGW --> L[Lambda agent_api]
  L -->|GetObject/List| S3KB[S3 Knowledge Bucket]
  L -->|Embeddings + Responses| OAI[OpenAI API]
```
````

Next hardening steps: add authentication (Cognito), restrict origins, enable WAF rate limiting, add request size limits, and optionally move OpenAI key into Secrets Manager with rotation.