Integrated Geoscience & Geoengineering Modeling Platform

Purpose

A clear, developer-facing architecture and stack for a **desktop-first, no-code** subsurface modeling platform (MVP → Cloud / AI phases). This document arranges the entire system by **Frontend, Backend, Database, Visualization**, plus supporting systems (workflow engine, storage, CI/CD, security, offline mode, plugin system) and a deployment & milestone roadmap.

High-level Architecture Diagram (flow)

```
[ User Desktop App ] <---> [Electron Shell (React UI + Blockly) ]
         |                                  |
         | REST / WebSocket                  | Embeds Dash / Plotly panels
         v                                  v
[ Local Backend (FastAPI) ]  <---->  [Workflow Engine / Orchestrator]
         |                                  |
         | Celery/Task Queue (Redis)          | Plugin Manager (Python APIs)
         v                                  v
[ Database Layer ]  <----->  [Object Storage (MinIO or S3)]
(Postgres + PostGIS)         (Seismic volumes, grids, models)
         |
         v
[ Visualization Engine ]
(PyVista / VTK) -> Render -> Canvas in Electron (WebGL / Native bridge)

Optional Cloud Mode:
Local Backend  <-->  Cloud API Gateway  <-->  Cloud Compute / GPU / AI Services
```

Layers & Components (Arranged)

1) Frontend (Desktop-first no-code client)

- **Electron**: Desktop shell, packaging, cross-platform distribution.
- **React**: Main UI framework (component-driven, performance). Use TypeScript.
- **Blockly / Node-RED style canvas**: Visual drag-and-drop workflow builder for non-programmers.
- **Plotly Dash (embedded if needed)**: For complex interactive dashboards without extra JS.
- **Three.js / deck.gl**: Fast WebGL 3D rendering for interactive scenes in the browser UI.
- **Tailwind CSS / Radix UI**: Design system and components.

- **IPC / WebSocket**: Bi-directional comms between Electron UI and local backend for realtime updates.

Key frontend responsibilities:

- No-code workflow creation, node properties editing, visual data mapping.
- Upload UI for LAS, SEG-Y metadata, CSV, GeoJSON.
- Interactive 2D/3D viewers embedded in the same app window.
- Role-based UI controls and offline-first project sync.

2) Backend (Local & Cloud-capable)

- **FastAPI**: Primary API server (async, lightweight, auto-docs).
- **Celery** + **Redis**/**RabbitMQ**: Asynchronous task queue for heavy compute (gridding, inversion, large transforms).
- **NumPy / pandas**: Tabular and numerical processing.
- **Cython / Numba / PyO3 (Rust)**: Accelerate hotspots (interpolation, gridding, numerical solvers).
- **Geoscience libs**: `lasio`, `welly`, `segpy` (SEG-Y handling), `pyproj`, `rasterio`, `geopandas`.
- **Plugin API (Python)**: A clear SDK so domain experts can add modules (pre- & post-processors, custom models).
- **Auth & RBAC**: Token-based auth for cloud mode; local user profiles and encrypted vault for offline projects.

Backend responsibilities:

- Data ingestion, parsing, normalization, validation.
- Orchestrating workflows from the no-code canvas.
- Running domain models as background jobs and returning results.
- Exposing WebSocket updates for long-running jobs.

3) Database & Storage

- **PostgreSQL + PostGIS**: Primary relational + spatial store (wells, horizons, annotations, metadata).
- **MinIO (S3-compatible)**: Local object storage for large binary datasets (SEG-Y, model grids, seismic volumes).
- **SQLite**: Lightweight local project DB for strictly offline mode (or encrypted project bundle).
- **SQLAlchemy / Alembic**: ORM + migrations.

Data partitioning guidance:

- Keep metadata and indexes in Postgres/PostGIS.
- Store large binaries in object storage and reference them by ID + checksum.
- Use delta uploads for cloud sync to reduce bandwidth.

4) Visualization & Modeling Engine

- **PyVista (VTK)**: 3D mesh & volumetric rendering, iso-surfaces, well trajectories, gridding visualization.
- **Mayavi** (optionally) for advanced scientific visualizations.
- **Matplotlib / Plotly / Seaborn**: 2D plots, histograms, crossplots, petrophysical logs.

- **Custom WebGL Bridge**: For extremely interactive 3D visualizations in Electron, stream precomputed geometry or use `vtk.js`.

Visualization responsibilities:

- Render interactive 3D scenes with overlays (horizons, faults, wells).
- Provide linked views: selecting an item in table highlights in 3D viewer.
- Export figures and printable reports (PDF).

Workflow / No-Code Engine

- **Node model**: Each workflow node maps to a Python plugin: `Input -> Transform -> Model -> Output`.
- **Execution modes**:
- `Local Sync` for small jobs (FastAPI direct call).
- `Background` for heavy compute (enqueue Celery task).
- `Cloud` for offloading to remote compute resources (k8s jobs / GPU instances).
- **Versioning**: Workflow versioning and reproducibility metadata saved with the project.
- **Error handling & logs**: Streams back to UI with rich context for each node.

Extensibility: Plugin System

- **Python-based plugin interface** exposing:
- `metadata()` (name, inputs, outputs, params, version)
- `validate(inputs)`
- `run(inputs, params, progress_callback)`
- `ui_schema()` (optional JSON schema for node property editor)
- Plugin discovery via a plugin folder in project or via pip-installed packages.
- Support sandboxing: run untrusted plugins inside isolated processes or containers.

Offline-first & Cloud Modes

- **Default**: Desktop app with local backend + local Postgres/SQLite + MinIO.
- **Cloud integration options**:
- Sync projects to a cloud service (object storage + managed Postgres or PostGIS).
- Remote compute: submit workflows to a Kubernetes cluster with autoscaling.
- Authentication through OAuth2 / OpenID Connect for corporate deployments.
- **Sync mechanics**: Conflict resolution by timestamps and user choice; chunked uploads for large files.

Security & Compliance

- **Encryption at rest** for sensitive project data (AES-256 for local vaults and S3 buckets).
- **Transport security**: TLS for any cloud communication; secure WebSocket (wss).
- **Role-based access control (RBAC)**: Admin / Engineer / Viewer roles.
- **Audit logs**: Track changes, exports, and model runs.
- **Plugin sandboxing** to avoid arbitrary code execution risks in shared/cloud mode.

DevOps / CI-CD / Testing

- **CI**: GitHub Actions (or GitLab CI) for linting, unit tests, type checks (mypy), and building Electron artifacts.
- **Integration tests**: Run headless integration tests (Playwright) for the UI and API contract tests for FastAPI.
- **Packaging**: Electron-builder for Windows/macOS/Linux installers; Docker images for cloud services.
- **Monitoring**: Prometheus + Grafana for server metrics; Sentry for app error monitoring.

Suggested Project Structure (mono-repo)

```
/monorepo
  /client-electron (React + TypeScript)
  /client-dash (optional Plotly dash panels)
  /backend
    /app (FastAPI app)
    /plugins
    /workers (Celery tasks)
  /services
    /minio (local deploy scripts)
    /postgres (migrations)
  /infra (k8s manifests, helm)
  /docs
  /scripts (build, package, release)
```

MVP Feature Set (Desktop)

1. Project creation, import LAS/CSV/GeoJSON, basic validation.
2. Drag-and-drop workflow builder with nodes for parsing, simple transforms, and visualization.
3. 2D log viewer + crossplots + basic 3D viewer (mesh + wells).
4. Background job runner (local Celery) and progress streaming.
5. Export reports (PDF) and project bundles.
6. Plugin SDK and one sample plugin (e.g., simple gridding or material-balance model).

Roadmap & Milestones (6 months suggested MVP timeline)

- **Month 0–1**: Scaffolding: monorepo, Electron shell, FastAPI skeleton, Postgres + MinIO local deploy.
- **Month 2**: Basic ingestion (LAS, CSV), simple UI, and 2D plots; plugin SDK draft.
- **Month 3**: No-code canvas (Blockly-like) and node execution plumbing; Celery worker integration.
- **Month 4**: 3D visualization integration (PyVista + vtk.js bridge) and exporting capabilities.
- **Month 5**: Offline project bundles, project sync basics, RBAC minimal implementation.
- **Month 6**: Testing, packaging, stakeholder demo, and roadmap for cloud + AI features.

CI/CD & Release Strategy

- Build and test on PRs.
- Nightly artifacts: packaged Electron for QA.
- Release channels: `alpha` (internal), `beta` (partners), `stable` (public).
- Maintain detailed release notes with migration steps and plugin compatibility.

Next steps I can help with (pick one)

- Generate a visual architecture diagram (SVG/PNG) from this layout.
- Create the Electron + FastAPI starter repo with basic IPC and an example plugin.
- Design the plugin SDK spec and a sample plugin (material balance model).
- Draft the UI screens for the no-code canvas (wireframes).

Prepared for: Integrated Geoscience & Geoengineering Modeling Platform — MVP plan