Product Development Specification (PDS) for GeoVision AI Miner

---

1. Project Overview

1.1. Purpose
GeoVision AI Miner is a cloud-based, AI-driven mineral exploration platform that integrates geological, geophysical, geochemical, and satellite data into a single collaborative environment. Its goal is to help exploration teams across Africa (initially South Africa, Namibia, Mozambique, Zimbabwe, Zambia, DRC, and Malawi) rapidly identify and validate high-potential mineral targets via predictive modeling, interactive 2D/3D visualization, and decision-support reporting.

1.2. Scope

- Build a modern web application with a responsive front-end for mapping, modeling, and dashboards.
- Develop back-end microservices for data ingestion, AI analytics, user management, and reporting.
- Integrate machine learning models that process multi-layer geoscientific data into prospectivity maps.
- Support multiple user roles (geologist, geophysicist, driller, environmental officer, QA/QC, manager, executive) each with tailored interfaces and permissions.
- Provide 2D GIS maps and 3D subsurface visualization (drillholes, geological surfaces, AI-predicted ore bodies).
- Implement reporting templates and analytics dashboards for technical and managerial decision-making.
- Ensure robust security measures (encryption, RBAC, auditing) to protect sensitive exploration and proprietary data.

---

2. Objectives & Success Criteria

1. **Data Integration & AI Modeling**
   - Ingest geological maps, geophysical survey grids, geochemical assays, and satellite imagery into a unified geospatial database.
   - Train and deploy supervised machine learning models (e.g., random forest, gradient boosting, convolutional neural nets) that generate prospectivity heatmaps for copper, gold, and other key commodities.
   - Provide explainable AI outputs (feature importance, confidence metrics) to support geoscientist review.
   - **Success Criteria:** AI models correctly flag ≥ 80 % of known deposit locations in retrospective tests; prospectivity maps load in < 5 seconds on average.
2. **Multi-User Collaboration & Role-Based Access**

   - Define and enforce at least seven user roles with least-privilege access (Administrator, Geologist, Geophysicist, Drilling Manager, QA/QC, Environmental Officer, Executive).
   - Implement project-level data isolation: users see only the datasets and projects they are granted.
   - Provide real-time commenting, annotation, and notification features so that, for example, a geologist's data upload immediately notifies the driller or QA/QC team.
   - **Success Criteria:** Each role's interface hides or disables unauthorized features; onboarding a new user takes < 15 minutes with basic training.
3. **2D/3D Visualization & Mapping**
   - Build an interactive 2D map module (Mapbox GL or OpenLayers) that can layer multiple raster/vector datasets with transparency controls, legends, and point/line/polygon interactions.
   - Create a 3D visualization engine (CesiumJS or Three.js) to render drillholes in correct collar coordinates, geological surfaces (interpolated from drill data), and AI-derived orebody isosurfaces.
   - Enable users to draw polygons, measure distances/areas, and slice through 3D models (arbitrary cross-section).
   - **Success Criteria:** 2D map loads base layers in < 3 seconds; 3D scene (with up to 1,000 drillholes) maintains ≥ 30 FPS on a modern laptop GPU.
4. **Reporting & Analytics**
   - Implement templated technical report generation in PDF/Word format (e.g., "Exploration Target Report," "AI Prospectivity Summary," "QA/QC Data Validation Report").
   - Provide in-app dashboards with charts (assay distributions, model performance, drilling progress) and executive KPIs (e.g., "Cost per target," "Projected resource tonnage × grade vs. exploration budget").
   - Allow users to export raw data (CSV, GeoJSON) and model outputs (GeoTIFF, shapefiles) for offline analysis in GIS or mine planning software.
   - **Success Criteria:** Reports generate (compile and export) in < 20 seconds; dashboards update in real time as new data are added.
5. **Security & Compliance**
   - Enforce TLS 1.2+ (HTTPS only) with HSTS.
   - Require multi-factor authentication (MFA) for all accounts, implement strict password policy and optional SSO integration for enterprise clients.
   - Encrypt data at rest (database TDE, object storage via KMS) and in transit.
   - Maintain detailed audit logs of user activity (logins, data uploads, AI runs, administrative changes).
   - Perform regular vulnerability scans, apply OWASP Top 10 mitigations in the codebase, and schedule periodic third-party penetration tests.
   - Comply with local data residency requirements when partnering with government agencies (allow hosting in country-specific regions if requested).
   - **Success Criteria:** Zero critical vulnerabilities remain unpatched for more than 72 hours; 100 % of user activity is logged and reviewable; audit reports available on demand.

6. **Scalability & Performance**
   - Architect using microservices (containerized with Docker, orchestrated via Kubernetes or managed container service).
   - Use horizontally scalable data store (PostgreSQL/PostGIS for spatial, object store for rasters, NoSQL for indexing).
   - Implement caching layers (Redis or Memcached) for frequently accessed tiles and model results.
   - **Success Criteria:** System can handle 200 concurrent active users (viewing maps, running AI jobs) without degradation; AI service autoscaling triggers within 60 seconds of increased load.
7. **Usability & Adoption**
   - Develop an intuitive, responsive UI (React or Angular) with role-specific dashboards, inline help tooltips, and a guided onboarding wizard.
   - Offer in-app tutorials and a searchable knowledge base.
   - Provide formal training materials (video tutorials, user manual PDFs) and conduct at least one regional live workshop in each target country within the first year.
   - **Success Criteria:** ≥ 85 % of pilot users rate usability as “Good” or “Excellent”; average onboarding time < 30 minutes; < 10 % of support tickets due to UI confusion.
8. **Maintenance & Support**
   - Set up CI/CD pipeline with automated tests (unit, integration, security scans) to deliver weekly incremental updates and monthly feature releases.
   - Offer 24 × 7 monitoring and a support helpdesk (SLA: 1 hour response for critical incidents, 8 hours for high-prio issues).
   - Maintain a public changelog and versioned API documentation with examples.
   - **Success Criteria:** Released updates cause < 1 % downtime (planned or unplanned); 95 % of high-priority tickets resolved within SLA; user retention ≥ 80 % at six months.

---

3. Stakeholders & Roles

- **Project Sponsor:** GeoVision AI Miner Founders / Executive Team
- **Steering Committee:** Representatives from pilot partners (e.g., Geological Surveys, academic collaborators, mining company pilots)
- **Product Owner:** Platform Product Manager
- **Development Team:**
  - Front-End Engineers (React/Angular, WebGL/GIS)
  - Back-End Engineers (Python/Node.js, API design, microservices)
  - Data Engineers (PostgreSQL/PostGIS, ETL pipelines, object storage)
  - Data Scientists/ML Engineers (model training, inference, explainability)
  - DevOps/Cloud Engineers (container orchestration, CI/CD, monitoring)
- **UX/UI Designer:** Responsible for wireframes, prototypes, and usability testing
- **QA & Security Engineers:** Automated/manual testing, vulnerability scanning, pen testing

- **Technical Writers & Trainers:** Documentation, user manuals, training videos
- **Support Team:** Tier 1 helpdesk, Tier 2 technical support

---

4. Functional Requirements

4.1. Data Ingestion & Management

- **4.1.1. Geological Data Upload**
  - Users can upload vector (shapefiles, GeoJSON) and raster (GeoTIFF) layers.
  - System automatically georeferences and validates coordinate systems; provides feedback on mismatches.
  - Metadata capture (source, date, commodity focus, licensing) required at upload.
- **4.1.2. Geophysical & Geochemical Data Import**
  - Bulk CSV/XLS import for sample tables (with columns: sample_id, easting, northing, element columns).
  - Ability to ingest standard geophysical grid formats (e.g., .grd, .xyz) and associate them with a survey parameter (magnetic, gravity).
  - Automated preview of data distribution (histograms of assay values, summary stats).
- **4.1.3. Satellite Imagery & Remote Sensing Layers**
  - Integration with third-party APIs (e.g., Digital Earth Africa) to fetch analysis-ready Landsat/Sentinel tiles for selected bounding boxes.
  - Users can upload custom UAV orthomosaics (GeoTIFF).
  - System pre-loads base imagery for focus countries (public, high-resolution where available).

4.2. Machine Learning & Analytics

- **4.2.1. Model Training Pipeline**
  - A data labeling interface where geologists mark known deposit and non-deposit locations (for supervised learning).
  - ETL pipelines to generate training features:
    - Raster feature extraction (e.g., local window statistics of magnetic intensity, spectral indices from satellite).
    - Categorical features (rock type, structural zone).
    - Distance features (to known faults, streams, alteration zones).
  - Support for multiple commodity-specific models (e.g., copper, gold, lithium).
  - Automated hyperparameter tuning (e.g., grid search or Bayesian optimization) for model accuracy.
- **4.2.2. Model Inference & Prospectivity Maps**
  - Users can select a project area (polygon) and trigger a “Run AI Prospectivity Analysis” job.

  - Backend runs inference, producing a georeferenced raster (GeoTIFF) of deposit probability (0–100 %).
  - Automatically generate a confidence or uncertainty map alongside.
  - Provide feature importance summary for that run (e.g., "Magnetic anomaly accounted for 35 % of predictive weight").
- **4.2.3. Feature Correlation Tools**
  - Interactive correlation matrix interface: users select up to 10 variables (e.g., arsenic in soil, proximity to shear zone, elevation) and view scatter-plot grids and correlation coefficients.
  - Ability to overlay correlations on map (e.g., show where two variables co-occur above threshold).
- **4.2.4. 3D Geological Modeling**
  - Import drillhole logs (with collar coords, downhole depths, intervals, assays).
  - Automatically interpolate 3D surfaces for lithological contacts or ore envelopes using standard algorithms (e.g., IDW, kriging) or feed into AI-driven implicit modeling.
  - Visualize drillholes as color-coded cylinders in 3D space, with toggles to display/hide intervals by assay thresholds.
- **4.2.5. Reporting & Dashboards**
  - Predefined report templates that compile maps, charts, tables, and narrative text into a single document.
  - Interactive dashboards:
    - "Project Overview" (project status, # targets, # drillholes, project value estimate)
    - "Data Quality" (duplicates detected, QA/QC flags)
    - "Model Performance" (precision/recall on validated targets, training set metrics)
    - "Environmental Risk" (overlay of targets on protected areas).
  - Export to PDF, Word, Excel as needed.

4.3. User Interface & Experience

- **4.3.1. Authentication & Dashboard**
  - Central login page with MFA prompt.
  - Landing dashboard customized per role:
    - Geologist: "My Projects," "Upload Data," "Run AI Analysis."
    - Executive: "Portfolio View" with KPIs across all active projects.
  - Quick access to "Recent Projects" and "Notifications" (e.g., "AI run complete," "New comment on your project").
- **4.3.2. Project & Data Management UI**
  - Project creation wizard: define project name, geographic extent (draw on map or upload polygon), commodity focus, partner organization.
  - Data library interface: list of uploaded datasets (with icons for map preview, metadata, download link). Search/filter by type (geology, geophysics, geochem, satellite), date, uploader.
- **4.3.3. Map & Visualization Controls**

  - Layer panel (show/hide, reorder, adjust opacity).
  - Legend panel that dynamically updates based on visible layers.
  - Tools for drawing (point, line, polygon) with metadata input (e.g., draw a polygon and name it "Target Area A").
  - Measurement tool (distance, area) in both 2D map and 3D scene.
  - If user hovers or clicks on a feature (e.g., sample point), display a popup with attribute details (assay values, QA/QC flags).
- **4.3.4. 3D Scene Controls**
  - Toggle switch ("2D Map" vs. "3D Model") in map header.
  - 3D scene displays terrain with draped raster layers; drillholes rendered as vertical cylinders with color bars.
  - Floating control panel:
    - Layer visibility (e.g., toggle "Drillholes," "Orebody Isosurfaces," "Lithology Surfaces").
    - Transparency sliders.
    - Cross-section drawer: click start and end points; 3D viewer automatically generates a vertical slice plane.
  - Camera controls: Orbit, Zoom, Pan (mouse or touch gestures).
- **4.3.5. Collaboration & Notifications**
  - In-map commenting: user selects a location or feature, types a comment, and tags another user (@username).
  - Notification center (bell icon) with real-time updates: "@Alice commented on Project Z: 'Looks promising near that fault.'"
  - Ability to subscribe/unsubscribe to project events (e.g., AI run completions, data uploads by others).

4.4. Security & Permissions

- **4.4.1. Authentication & Authorization**
  - Support for user registration (with email verification) and SSO (OAuth2) for enterprise clients.
  - Enforce MFA via TOTP (Google Authenticator or Authy) or SMS.
  - Role-based permissions defined per project. Admins can assign and revoke roles.
- **4.4.2. Data Encryption**
  - All data in transit encrypted via TLS 1.2+.
  - Data at rest encrypted using KMS-managed keys (for both databases and object storage).
- **4.4.3. Audit & Logging**
  - Record every user action (login, data import, AI run, comment posted).
  - Keep immutable logs (write-once storage) for at least seven years to satisfy audit requirements.
  - Provide an audit UI for Admins to filter logs by user, date, action type.
- **4.4.4. Infrastructure Security**
  - Deploy back-end services in private subnets; expose only APIs via secure load balancers.
  - Apply WAF rules to the HTTP endpoints.

- Require VPN or bastion host for administrative SSH/RDP access to servers.
- Regularly scan servers with vulnerability scanners (e.g., Nessus) and remediate high-severity findings within 48 hours.

4.5. Integration & Extensibility

- **4.5.1. Third-Party Data Services**
  - Integrate with Digital Earth Africa API for on-the-fly satellite tile retrieval (user provides bounding box and time range).
  - Allow connections to external databases (e.g., ESRI ArcGIS Server, S3 buckets) for direct data ingestion if user has credentials.
- **4.5.2. External Software Interoperability**
  - Support export of data in common formats: GeoTIFF, shapefile, KML, CSV.
  - Offer a RESTful API (versioned) so external systems (e.g., internal GIS, resource modeling software) can query project metadata, retrieve prospectivity rasters, or post back drilling results programmatically.
- **4.5.3. Plugin/Extension Framework**
  - Define an SDK pattern so that advanced users can write custom analytics or data transformations (e.g., a geophysicist wants to run a bespoke inversion algorithm on imported EM data). These plugins run in a sandboxed environment in the back end.
  - Document plugin API endpoints and data models.

---

5. Non-Functional Requirements

5.1. Performance & Scalability

- **5.1.1. System Load & Concurrency**
  - Support 200 concurrent active users (viewing maps, running AI jobs) with < 5 seconds page load times.
  - AI/Analytics microservice autoscaling:
    - Baseline: 2 CPU, 4 GB RAM for light loads.
    - Scale out to 10 nodes (8 CPU, 32 GB RAM each) under heavy ML job queue.
  - Use caching (Redis) for frequently requested map tiles and repeated AI inference requests on the same area.
- **5.1.2. Latency**
  - 2D map tile load: < 2 seconds.
  - AI inference job scheduling acknowledgment: ≤ 2 seconds.
  - AI result raster available to download within 10 minutes for a 100 km^2 area (depending on data complexity).

5.2. Reliability & Availability

- **5.2.1. Uptime**
  - Target 99.9 % uptime (less than 9 hours of downtime per year).
  - Deploy in multi-availability-zone clusters; automatically failover if a zone becomes unavailable.
- **5.2.2. Disaster Recovery**
  - Daily backups of all databases and object storage.
  - Recovery Point Objective (RPO): 4 hours; Recovery Time Objective (RTO): 12 hours.
  - Quarterly DR drills to validate backup integrity and restore procedures.

5.3. Usability & Accessibility

- **5.3.1. Response Time**
  - Interactive tasks (pan, zoom, data layer toggle) respond within 300 ms.
  - Form validations (data upload) within 1 second.
- **5.3.2. Accessibility Compliance**
  - Conform to WCAG 2.1 AA guidelines: keyboard navigation, screen-reader labels, sufficient color contrast in UI elements.
  - Provide alt text for non-text content (e.g., map thumbnails).
- **5.3.3. Internationalization (i18n)**
  - UI text externalized for translation.
  - Support English by default; prepare for French and Portuguese language packs for later release.

5.4. Maintainability & Extensibility

- **5.4.1. Code Quality**
  - Enforce linter rules and code style guidelines (ESLint/Prettier for front end; Pylint/Black or ESLint for back end).
  - Achieve ≥ 80 % unit test coverage for all services; integration tests for critical workflows (data upload → AI run → map display).
- **5.4.2. Modular Architecture**
  - Design microservices so they can be updated or replaced independently (e.g., upgrade the AI service to a new framework without touching the map service).
  - Use semantic versioning for APIs.
- **5.4.3. Documentation**
  - Maintain a developer wiki with architecture diagrams, data schema descriptions, API reference, and deployment guides.
  - Keep user documentation (user guides, training videos) current with each release.

---

6. System Architecture Overview

1. **Client-Side (Front End)**

- **Framework:** React (TypeScript), with Redux or Context API for state management.
- **Mapping Libraries:** Mapbox GL JS (for 2D), CesiumJS (for 3D) or Three.js for custom scenes.
- **UI Component Library:** Material UI or Ant Design (customized for branding).
- **Authentication Flow:** OAuth2/JWT integration, redirect to central Auth microservice.
- **Communication:** RESTful or GraphQL calls over HTTPS to back end.

2. **Server-Side (Back End)**
   - **API Gateway:** A layer (e.g., Kong or AWS API Gateway) to route requests to microservices and enforce rate limits.
   - **Microservices** (all containerized, deployed in Kubernetes or similar):
     - **Auth Service:** Manages user accounts, roles, MFA enforcement, token issuance, audit logs.
     - **Data Service:** CRUD operations for datasets; spatial queries; metadata management.
     - **AI/Analytics Service:** Model training/inference pipeline; job queue (e.g., Celery, RabbitMQ); integration with GPU nodes for heavy computation.
     - **Model Registry:** Stores versioned ML models and metadata (training date, dataset snapshot, performance metrics).
     - **Visualization Service:** Preprocesses 3D models and generates map tiles (vector tiles or cached rasters) on demand.
     - **Reporting Service:** Generates PDF/Word reports via templating engine (e.g., Jinja2 for Python or Docx for Node).
     - **Notification Service:** Sends in-app and email notifications (via WebSockets or a pub/sub mechanism).
   - **Databases & Storage:**
     - **PostgreSQL + PostGIS:** Core spatial database for vector features, metadata, project definitions.
     - **Object Storage (S3-compatible):** Stores rasters (satellite, prospectivity maps), 3D model files, backups.
     - **NoSQL (Elasticsearch or MongoDB):** Index for fast search of datasets, comments, logs.
     - **Redis:** Caching layer for session management and frequent read queries (legend data, map tile indices).
3. **Infrastructure & Operations**
   - **Container Orchestration:** Kubernetes (EKS, AKS, or GKE) or managed container service.
   - **CI/CD Pipeline:**
     - Git repository with branch protection.
     - Automated builds (Docker image creation), SAST/DAST scans, unit/integration tests.
     - Deployment to staging; manual approval for production.
   - **Monitoring & Logging:**
     - Prometheus + Grafana for metrics (CPU, memory, response times).
     - ELK Stack or cloud logging for centralized log ingestion and search.

    - Alerts via PagerDuty or Slack for critical incidents.
  - **CDN & WAF:**
    - CloudFront or Azure CDN for static assets (JS/CSS bundles, map tiles).
    - WAF in front to filter malicious traffic (SQLi, XSS, etc.).

---

7. Data Requirements

1. **Baseline Datasets (per country)**
   - **Geological Maps:** Vector polygons of lithology (1:250,000 or 1:100,000 scale), structural features, known mineral occurrences.
   - **Geophysical Grids:** Magnetic intensity, gravity anomaly grids (raster or XYZ).
   - **Geochemical Samples:** Historical soil/rock/drill-core assay tables with location coordinates and element concentrations.
   - **Topography & DEM:** Digital Elevation Model (30 m or finer) for terrain modeling and hydrological analysis.
   - **Satellite Imagery:** Sentinel-2 multispectral (10 m) and/or Landsat 8 (30 m) tiles for identified date ranges.
2. **User-Provided Data**
   - Drillhole logs: XLS/CSV with collar coordinates, downhole intervals, lithology codes, assays.
   - Custom survey data (e.g., resistivity, EM) in standard geoscience formats.
   - Environmental baseline data: water table measurements, biodiversity survey points (for risk modeling).
3. **Metadata & Data Quality**
   - Enforce a metadata schema for each upload: data source, sampling date, data type, method, data owner.
   - Perform automated QA/QC checks on sample tables (e.g., check for outliers via z-score, duplicate sample IDs).

---

8. User Roles & Permissions

Role	Permissions
Administrator	Create/edit/delete projects; manage users/roles; view all data; configure system settings (e.g., add new AI models, adjust resource quotas); view audit logs.
Geologist	Create and manage assigned projects; upload geological & geochemical data; run AI analysis; view stacks of datasets; interpret 2D/3D models; annotate maps; access standard reports; comment/annotate features; cannot modify system settings.

Role	Permissions
Geophysicist	Import geophysical survey grids; run AI models on geophysics data; view/interpret 2D/3D scenes; collaborate with geologists; generate geophysical anomaly reports.
Drilling Manager	Plan drilling: select AI-identified targets; create drilling schedules; update drillholes (collar coords, depths, assays); view drilling dashboards; export drillhole data to third-party modeling software.
QA/QC Officer	Access raw data uploads (assays, drill logs); flag/reject records failing QA; generate QA/QC reports; view data lineage; cannot run AI models or modify project settings; approve data for AI training.
Environmental Officer	View project extents and overlap with protected areas; upload environmental data; run environmental risk overlay; generate environmental compliance reports; comment on targets from an environmental perspective; cannot edit core geological data.
Executive/Manager	View high-level dashboards & KPIs across all projects; generate summary reports; compare project portfolios; adjust project budgets (if linked to financial module); invite new users to projects; cannot modify data or run low-level AI tasks.
Read-Only Viewer	View public or shared project layers and maps; cannot upload data or run AI models; used for external partners (e.g., government, investors) to see read-only results.

---

9. User Interface & UX Requirements

1. **Homepage & Global Navigation**
   - Clear top-bar with GeoVision AI Miner logo, global search, notifications icon, user profile menu.
   - Sidebar menu with icons and text for: Dashboard, Projects, Data Library, Maps, AI Analysis, Reports, Admin (conditionally rendered if user role includes “Administrator”).
   - Breadcrumb navigation to show hierarchy (e.g., Home → Projects → Project A → AI Analysis).
2. **Project Creation Wizard**
   - Step 1: Project name, description, commodity focus, partner organizations.
   - Step 2: Define geographic extent by drawing polygon on map or uploading GeoJSON/KML.
   - Step 3: Select baseline datasets to include (country geology, DEM, satellite).
   - Step 4: Assign team members and roles.
   - Confirmation page summarizing choices; “Create Project” button.
3. **Data Library Interface**
   - Tabular listing of datasets with columns: Name, Type (Geology, Geophysics, Geochemistry, Satellite), Date Added, Owner, Visibility (Public/Private), Actions (Preview, Download, Delete).

   - Filter controls: by type, date range, owner, keyword search.
   - "Upload Data" button opens a multi-step form: select file(s), auto-detect format, preview metadata, confirm upload.
4. **Map & Visualization Module**
   - Central map canvas occupying 70 % of width; collapsible layer panel on left (toggleable).
   - Layer panel shows hierarchical categories (e.g., "Basemaps" → "Satellite"; "User Data" → "Project A → Drillholes"). Each layer has a checkbox, opacity slider, legend icon.
   - Top toolbar with tools: Zoom Extents, Pan, Draw Polygon, Measure, Identify (click to see feature attributes).
   - Bottom status bar showing mouse coordinates, scale, selected layer info.
   - Right-click context menu on the map: "Center Here," "Add Marker," "Start Cross-Section."
5. **3D Scene Interface**
   - Toggle switch ("2D Map" vs. "3D Model") in map header.
   - 3D scene displays terrain with draped raster layers; drillholes rendered as vertical cylinders with color bars.
   - Floating control panel:
     - Layer visibility (e.g., toggle "Drillholes," "Orebody Isosurfaces," "Lithology Surfaces").
     - Transparency sliders.
     - Cross-section drawer: click start and end points; 3D viewer automatically generates a vertical slice plane.
   - Camera controls: Orbit, Zoom, Pan (mouse or touch gestures).
6. **AI Analysis Workflow**
   - Within a project, "AI Analysis" tab presents:
     - "Select Model" dropdown (e.g., "Copper Prospectivity – Zambia Model," "Gold Prospectivity – SA Model").
     - "Define Area of Interest" panel: draw or import polygon.
     - "Feature Selection" optional panel: choose which layers to include (e.g., include magnetics, exclude geochemical data). Defaults to all.
     - "Run Analysis" button. Upon clicking:
       1. Validate inputs (AOI size < threshold, required layers exist).
       2. Submit job to AI queue.
       3. Display "Job Queued" notification with estimated wait time.
     - Once complete: show thumbnail of prospectivity map; buttons for "View in 2D Map," "Download GeoTIFF," "Generate Report."
7. **Reporting & Export**
   - "Reports" tab lists available report templates. Each row has: Template Name, Description, Last Modified, "Generate" button.
   - Clicking "Generate" opens a dialog: select project, specify AOI (optional), add custom notes.
   - After generation, show progress indicator; when done, list the report under "My Reports" with action buttons: View (in-app PDF viewer), Download, Share Link (expiring URL).

- “Export Data” button on Data Library allows CSV/GeoJSON/Shapefile export; also “Export Model” on AI results.

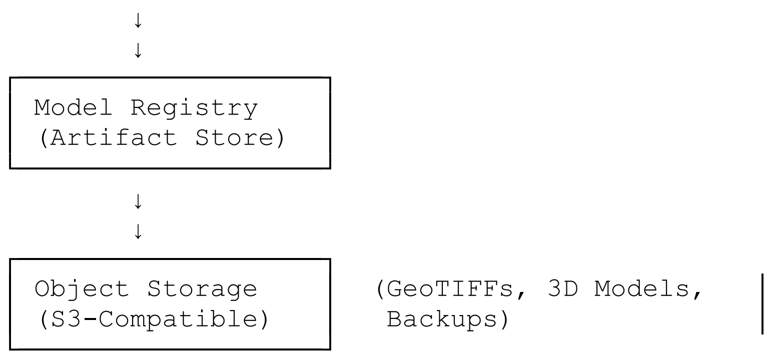
8. **Administration & Settings**
   - “Admin” menu with submenus:
     - **User Management:** list of all users; columns: Name, Email, Role(s), Last Active, Actions (Edit, Deactivate). “Invite User” button opens form to send invitation.
     - **Role & Permission Definitions:** view/edit who can do what; optionally create custom roles.
     - **System Settings:** set MFA requirements, password policy, idle session timeout, email templates for notifications.
     - **Audit Logs:** searchable, filterable table of events (date/time, user, action, resource). Can export logs for compliance.
     - **Model Management:** upload new AI model, view existing model versions, retire old models.

---

10. Technical Architecture & Integration

10.1. High-Level Diagram

```
┌────────────────┐            ┌─────────────────┐  ┌────────────────────┐
│  Web Browser   │ ⟷ (HTTPS) → │  API Gateway    │ ⟷ │ Auth Service       │
│ (React/TS UI)  │              │ (Rate Limit, WAF)│      │ (OAuth2, JWT)
│                │              └─────────────────┘
└────────────────┘
└──────────────────┘
        ↓                                            ↓
        ↓                                            ↓
┌──────────────────┐    ⟷     ┌───────────────────┐    ⟷
┌──────────────────┐
│ Map Service      │          │  Data Service     │       │ Notification
│                  │          │                   │       │
│(Tile Rendering,  │          │ (PostGIS, ES)     │       │ Service (Email,
│                  │          │                   │       │
│ 2D/3D Precompute)│          │                   │       │  In-App)
│                  │          └───────────────────┘
└──────────────────┘
└──────────────────┘
        ↓                                           ↑
        ↓                                           │
┌───────────────────┐    ⟷     ┌────────────────────┐    ⟷
┌───────────────────┐
│ AI/Analytics      │          │ Reporting Service  │        │  Caching
(Redis)  │
│ (Model Inference, │          │ (PDF/DocX Engine)  │        │
│                   │          │                    │
│  Training)        │          │                   │
└───────────────────┘          └───────────────────┘
└───────────────────┘
```



10.2. Key Components & Integration Points

- **API Gateway:**
  - Central entry point for all HTTPS requests.
  - Enforces rate limiting (e.g., 100 requests/min per IP), WAF rules, and routes authenticated requests to appropriate microservices.
- **Auth Service (OAuth2/JWT):**
  - Manages user accounts, password resets, MFA enrollment, and token issuance/validation.
  - Provides endpoints:
    - **POST** `/auth/register` (triggers email verification).
    - **POST** `/auth/login` (returns JWT).
    - **POST** `/auth/mfa/verify` (TOTP code).
    - **GET** `/auth/user` (returns user profile and roles).
- **Data Service:**
  - Central PostgreSQL/PostGIS for spatial tables and project metadata.
  - Elasticsearch for metadata indexing (fast search/filter).
  - CRUD endpoints:
    - **POST** `/api/projects` (create new project).
    - **GET** `/api/projects/{projectId}.`
    - **POST** `/api/data/upload` (file upload, format validation, store in object storage, ingest metadata into PostGIS).
    - **GET** `/api/data/{dataId}/preview` (returns thumbnail or summary).
    - **DELETE** `/api/data/{dataId}` (soft delete with retention policy).
- **Map Service:**
  - Generates vector and raster tiles on demand or pre-caches tiles for frequently used basemaps.
  - Renders 3D model slices to WebGL-compatible formats (e.g., glTF or binary buffers).
  - Serves tile endpoints:
    - **GET** `/tiles/2d/{z}/{x}/{y}.png` (PNG tile).
    - **GET** `/tiles/3d/{modelId}` (3D model stream).
- **AI/Analytics Service:**
  - Consumes data from Data Service and Object Storage.

  - Uses job queue (e.g., Celery + RabbitMQ or AWS SQS) to schedule model training/inference.
  - Model training pipeline:
    1. Extract labeled training data from PostGIS.
    2. Preprocess into feature matrices (numpy arrays, raster patches).
    3. Train models (scikit-learn, XGBoost, or TensorFlow).
    4. Save model artifact to Model Registry.
  - Inference pipeline:
    1. User selects AOI → generate raster tile grid.
    2. Load latest model from Model Registry.
    3. Run inference on grid blocks in parallel (using GPU if available).
    4. Stitch block predictions into a single GeoTIFF (using rasterio) with appropriate CRS (EPSG:4326).
    5. Upload the resulting GeoTIFF to object storage under `/projects/{projectId}/ai_runs/{runId}/prospectivity.tif`.
    6. Update the `ai_runs` table with status and `result_url`.
    7. On failure, update status to `failed` and log the error.
- **Reporting Service:**
  - Uses a headless browser or templating engine to generate polished PDFs (e.g., via wkhtmltopdf or Puppeteer).
  - Pulls maps (via Map Service tile snapshots), AI summaries, user notes, and exports as a branded report.
- **Notification Service:**
  - Listens for events (e.g., AI job complete, comment added).
  - Sends emails (SMTP or third-party provider like SendGrid) and in-app push notifications (via WebSockets or a pub/sub mechanism).

---

11. Data & Integration Details

11.1. Spatial Data Schema (PostGIS)

1. **Table: projects**
   - `project_id` (UUID, primary key)
   - `name` (text)
   - `description` (text)
   - `commodity` (enum: copper, gold, lithium, etc.)
   - `owner_id` (UUID, references users.user_id)
   - `created_at` (timestamp)
   - `updated_at` (timestamp)
   - `geom_extent` (geometry, Polygon, SRID=4326)
2. **Table: datasets**
   - `dataset_id` (UUID)
   - `project_id` (UUID, references projects)

- `name` (text)
- `data_type` (enum: geology, geophysics, geochemistry, satellite)
- `file_url` (text) – pointer to Object Storage
- `schema` (JSON) – column definitions (if tabular) or other metadata
- `visible_to` (enum: public, project_members)
- `upload_date` (timestamp)

3. **Table: drillholes**
    - `drillhole_id` (UUID)
    - `project_id` (UUID)
    - `collar_geom` (geometry, Point)
    - `elevation` (numeric)
    - `drilling_date` (date)
    - `driller` (text)
    - `status` (enum: planned, in_progress, completed)
4. **Table: drill_intervals**
    - `interval_id` (UUID)
    - `drillhole_id` (UUID, references drillholes)
    - `from_depth` (numeric)
    - `to_depth` (numeric)
    - `lithology` (text)
    - `azm` (numeric)
    - `dip` (numeric)
5. **Table: assays**
    - `assay_id` (UUID)
    - `interval_id` (UUID, references drill_intervals)
    - `element` (text)
    - `value` (numeric)
    - `unit` (text: ppm, ppb, % etc.)
    - `qc_flag` (enum: pass, fail, review)
6. **Table: ai_models**
    - `model_id` (UUID)
    - `name` (text)
    - `commodity` (text)
    - `version` (text)
    - `training_date` (timestamp)
    - `performance_metrics` (JSON: e.g., {“precision”: 0.82, “recall”: 0.74})
    - `artifact_url` (text)
7. **Table: ai_runs**
    - `run_id` (UUID)
    - `project_id` (UUID)
    - `model_id` (UUID, references ai_models)
    - `area_geom` (geometry, Polygon)
    - `submission_time` (timestamp)
    - `completion_time` (timestamp)
    - `status` (enum: queued, running, completed, failed)

  - `result_url` (text, GeoTIFF link)

8. **Table: users**
   - `user_id` (UUID)
   - `email` (text, unique)
   - `password_hash` (text)
   - `roles` (array of enums)
   - `mfa_enabled` (boolean)
   - `created_at`, `updated_at` (timestamps)
9. **Table: comments**
   - `comment_id` (UUID)
   - `project_id` (UUID)
   - `user_id` (UUID, references users)
   - `comment_text` (text)
   - `geom` (geometry, optional point or polygon where comment applies)
   - `timestamp` (timestamp)

11.2. Object Storage Structure

- Bucket: `geovision-ai-miner-data`
  - `/country/{country_code}/baseline/geology/*.geojson`
  - `/country/{country_code}/baseline/geophysics/*.tif`
  - `/country/{country_code}/baseline/satellite/{year_month}/*.tif`
  - `/projects/{project_id}/datasets/{dataset_id}/{filename}`
  - `/projects/{project_id}/ai_runs/{run_id}/prospectivity.tif`
  - `/projects/{project_id}/3d_models/{model_file}`

---

12. Deployment & Scalability

1. **Environment Separation**
   - **Development:** deploy on a small Kubernetes cluster (2 nodes) with lower resource quotas. Developers use feature branches to test.
   - **Staging:** mirrors production size but uses test data; used for QA, UAT, security scans.
   - **Production:** multi-AZ Kubernetes cluster with at least 3 control nodes, 6 worker nodes (autoscaling enabled).
2. **Continuous Integration / Continuous Deployment**
   - **CI Pipeline** (on each pull request):
     1. Lint code (ESLint, Pylint).
     2. Run unit tests (Jest for front end, PyTest or Mocha for back end).
     3. Run static security scans (Snyk or Dependabot).
     4. Build Docker images and push to container registry with “snapshot” tag.
   - **CD Pipeline** (on merge to main branch):
     1. Deploy to staging for automated integration tests.
     2. Trigger smoke tests (basic health checks, critical UI workflows).

3. Await manual approval.
4. Deploy to production cluster with blue/green or canary deployment strategy (minimize downtime).
5. Run post-deployment health checks and performance tests.

3. **Autoscaling & Load Balancing**
   - Use a managed load balancer (e.g., AWS ALB) in front of web service pods.
   - Configure Horizontal Pod Autoscaler (HPA) for each microservice based on CPU and memory metrics (e.g., scale up AI service if CPU > 70 % for > 2 minutes).
   - Use a CDN (CloudFront, Azure CDN) to serve static assets (JS/CSS bundles, map tiles) for global performance.
4. **Monitoring & Alerting**
   - **Metrics:** CPU, RAM, disk I/O, network I/O per pod; response times; error rates.
   - **Logs:** All microservices send logs to a central ELK stack (Elasticsearch + Logstash + Kibana) or cloud logging service.
   - **Alerts:**
     - CPU/Memory > 80 % triggers an alert to DevOps team.
     - 5 % error rate on API endpoints in a 5-minute window triggers an alert to the on-call engineer.
     - AI job failures more than 3 in an hour trigger investigation.

---

13. Maintenance & Support

1. **Release Cadence**
   - **Weekly:** Minor bug fixes and security patches.
   - **Bi-monthly:** Small feature updates or UI improvements.
   - **Quarterly:** Major feature releases (e.g., new AI models, new country datasets, 3D enhancements).
2. **Support Tiers**
   - **Tier 1 (Helpdesk):** Initial triage for user queries, password resets, basic how-to questions. (SLA: response < 4 hours).
   - **Tier 2 (Technical):** Troubleshooting complex issues (data import errors, map loading issues, AI run failures). (SLA: response < 1 hour for critical).
   - **Tier 3 (Engineering):** Bug fixes, new feature requests, performance optimization. Prioritized in sprint planning.
3. **Documentation & Knowledge Base**
   - **User Documentation:**
     - "Getting Started" guide for each role (Geologist, Manager, etc.).
     - Step-by-step tutorials: "How to Upload Drillhole Data," "How to Run a Prospectivity Analysis," "How to Generate a Report."
     - FAQs for common issues (login problems, file format errors).
   - **Developer Documentation:**
     - API reference with examples (OpenAPI/Swagger).
     - Architecture diagrams, data schema docs, microservice responsibilities.

        - Onboarding guide for new developers (local dev environment setup, CI/CD workflow, coding standards).
    - **Training Materials:**
        - Recorded video tutorials for each module.
        - Slide decks for in-person workshops.
        - Quick-reference one-pagers (e.g., "Top 10 Tips for AI-Driven Prospectivity").
4. **Versioning & Changelog**
    - Tag each release with semantic versioning (MAJOR.MINOR.PATCH).
    - Maintain a `CHANGELOG.md` in the repo, listing new features, bug fixes, and security patches for each version.
    - Display current version and recent changes in the "About" page of the app so users know when new functionality arrives.

---

14. Acceptance Criteria & Milestones

Milestone	Acceptance Criteria
M1: Core MVP Deployment (End of Month 5)	- Users can register/login with MFA. - Project creation wizard is functional. - Data upload (geology, geochem, geophysics, satellite) works end-to-end (file → PostGIS/Object Storage). - 2D map displays a baseline country layer and uploaded user data. - AI model can be triggered on a small test dataset; prospectivity raster appears in user's project. - Basic role-based access control enforced. - Unit tests ≥ 80 % coverage; zero critical test failures.
M2: Beta Release (End of Month 6)	- 3D visualization of drillholes and geological surfaces is functional. - Full AI inference for at least two commodities (e.g., copper and gold) runs on real project data. - Reporting engine generates "Exploration Target Report" with map snapshots, project summary, and AI results. - Collaboration features (in-map comments, notifications) work with real-time updates. - Security audit completed: no OWASP Top 10 critical issues; MFA & encryption verified. - Pilot users (≥ 5) onboarded and able to run end-to-end workflows.
M3: Version 1.0 Launch (End of Month 12)	- All seven target countries have baseline datasets ingested (geology, DEM, satellite). - Multi-language support enabled (English + 1 additional language) based on pilot feedback. - Performance benchmarks met: 200 concurrent users, AI job completion SLA met (AOI ≤ 100 km² in < 10 minutes). - Reporting templates complete for all roles (technical, managerial, environmental). - Monitoring & alerting in place; recovery drills successfully executed. - At least 10 paying clients or signed partnership agreements across focus countries.
M4: Feature Expansion &	- Plugin framework documented and tested. - Mobile companion app for offline field data capture released (basic functionality). - Additional AI

Milestone	Acceptance Criteria
Growth (Months 13–24)	models for 2 more commodities (e.g., lithium, uranium) trained and deployed. - Integration with one major mine-planning software (e.g., Surpac or Leapfrog) via API. - User community forum launched; annual GeoVision AI Miner User Conference planned. - Achieve ISO 27001 readiness (audit passed).

---

15. Prompts for AI-Powered Development Assistants

Below are a set of carefully crafted prompts you can use with AI coding assistants (e.g., Cursor, V0, Claude) to generate boilerplate code, components, and configurations for building GeoVision AI Miner. They cover front-end components, back-end endpoints, data pipelines, and DevOps scripts. Adapt as needed to the specific assistant's syntax conventions; these prompts are designed to be clear and precise.

15.1. Front-End (React / TypeScript / Mapbox GL / CesiumJS)

1. **Project Structure & Boilerplate**
2. `Prompt:`
3. `"Generate a new React project skeleton in TypeScript called 'geovision-ai-miner-client'. Set up folder structure with 'src/components', 'src/pages', 'src/services', and 'src/styles'. Include React Router v6 for navigation. Output package.json dependencies optimized for mapping (e.g., react, react-dom, react-router-dom, mapbox-gl, cesium, @mui/material). Do not include any data or API keys."`

   Purpose: Creates the basic React app with required dependencies and folder layout.

4. **Authentication Wrapper**
5. `Prompt:`
6. `"Write a React component 'AuthProvider' that wraps the app and uses Context API to store JWT token and user role. Include functions to login (POST to '/auth/login'), logout, and check if user is authenticated. Use axios for HTTP requests. Provide TypeScript interfaces for User and AuthContext."`

   Purpose: Boilerplate for managing authentication state across components.

7. **Login Page**
8. `Prompt:`
9. `"Create a React page component 'LoginPage.tsx' with Material UI form inputs for email and password, a submit button, and client-side validation. On submit, call AuthProvider.login(email, password). Show loading spinner while request is pending and display error messages from the server beneath the form fields."`

Purpose: A ready-to-use login form that integrates with the authentication context.

10. **Role-Based Route Guard**
11. `Prompt:`
12. `"Implement a 'ProtectedRoute' component in React that takes 'allowedRoles' as a prop (array of strings). It checks AuthContext for the current user's roles and, if unauthorized, redirects to '/unauthorized'. Otherwise, renders the child component. Use React Router v6."`

Purpose: Ensure pages/components are accessible only by specified user roles.

13. **Sidebar Navigation**
14. `Prompt:`
15. `"Generate a responsive sidebar component using Material UI's Drawer. It should list menu items: Dashboard, Projects, Data Library, Maps, AI Analysis, Reports, Admin (conditionally rendered if user role includes 'Administrator'). Each item navigates to a React Router route. Include collapse/expand functionality on small screens."`

Purpose: Prebuilt side menu that hides or shows options based on user role.

16. **2D Map Component (Mapbox GL)**
17. `Prompt:`
18. `"Write a React component called 'Map2D' that initializes a Mapbox GL map centering on Africa (lng: 20, lat: 0, zoom: 2). The component should accept props: 'layers' (array of layer definitions with id, source, type, paint), and 'onFeatureClick' (callback). On mount, it adds layers to the map and sets up a click handler to call 'onFeatureClick' with feature properties. Use TypeScript types for layer definitions."`

Purpose: Core 2D map that can load dynamic layers and handle user clicks.

19. **3D Visualization Component (CesiumJS)**
20. `Prompt:`
21. `"Create a React component 'Map3D' using CesiumJS. It should display a terrain provider (Cesium Ion default) and allow adding drillholes as cylinders. Accept a prop 'drillholes' which is an array of objects {id, longitude, latitude, depth, assayValue}. For each drillhole, create a colored cylinder in the scene, where color is determined by 'assayValue' (e.g., a gradient from low to high). Also include camera controls to orbit and zoom."`

Purpose: Initializes a Cesium scene and plots drillholes in 3D.

22. **Layer Panel & Legend UI**
23. `Prompt:`
24. `"Generate a React component 'LayerPanel' that displays a list of map layers (passed as props). Each layer row shows a checkbox (toggle visibility), layer name, and opacity slider (0–100 %). When the checkbox or slider changes, call provided callbacks`

`onToggleVisibility(layerId, visible) and onChangeOpacity(layerId, opacity). Also render a color legend icon next to each layer if 'hasLegend' is true."`

Purpose: UI for controlling map layer visibility and transparency.

25. **Data Upload Form**
26. `Prompt:`
27. `"Create a React component 'DataUploadForm.tsx' that allows users to select a file (shapefile, GeoTIFF, CSV) via an <input type='file'>. After file selection, show a preview of detected format (e.g., CSV table snippet or GeoTIFF metadata like width/height). Include fields for metadata: 'Dataset Name', 'Type' (dropdown), 'Description'. On submit, package file and metadata into FormData and POST to '/api/data/upload'. Display upload progress bar using axios 'onUploadProgress'."`

Purpose: Enables users to upload geospatial or tabular data with metadata.

28. **Project Dashboard**
29. `Prompt:`
30. `"Develop a React component 'ProjectDashboard' that fetches from '/api/projects/{projectId}/overview' and displays: project name, commodity, number of datasets, number of AI runs, and a small D3 or Recharts chart showing assay value distribution (histogram). Also list recent events (comments, data uploads) with timestamps. Use Material UI Card components."`

Purpose: Shows a quick overview of a project's status and recent activity.

15.2. Back-End (Python / FastAPI / PostgreSQL / Celery)

1. **Project Model & CRUD Endpoints**
2. `Prompt:`
3. `"Define a SQLAlchemy model 'Project' with fields: project_id (UUID, primary key), name (string), description (text), commodity (enum), owner_id (UUID), geom_extent (Geography(Polygon, 4326)), created_at (timestamp), updated_at (timestamp). Then create FastAPI endpoints under '/projects':`
4. `- POST '/projects' to create a project (validate name, commodity, and parse 'geom_extent' as GeoJSON).`
5. `- GET '/projects/{project_id}' to retrieve project details.`
6. `- PUT '/projects/{project_id}' to update name, description, and commodity.`
7. `- DELETE '/projects/{project_id}' to soft delete (set an 'is_deleted' flag).`
8. `Include Pydantic schemas for request/response validation."`

Purpose: Basic CRUD for projects with spatial extent stored in PostGIS.

9. **User Authentication & JWT**
10. Prompt:
11. "Generate FastAPI endpoints for authentication under '/auth':
12. - POST '/auth/register' that creates a new user, hashes password with bcrypt, and sends email verification token (mocked).
13. - POST '/auth/login' that verifies credentials and returns a JWT (using PyJWT) with claims: user_id, roles, exp.
14. - POST '/auth/refresh' that takes a refresh token cookie and issues a new JWT.
15. - Dependency 'get_current_user' that extracts and validates JWT from 'Authorization: Bearer' header and returns user info.
16. Use Pydantic models for request/response and show how to configure JWT secret and algorithm in environment variables."

Purpose: Endpoints for user registration, login, and JWT-based auth.

17. **Data Upload Endpoint**
18. Prompt:
19. "Write a FastAPI endpoint 'POST /api/data/upload' that accepts multipart/form-data: 'file' (UploadFile), 'datasetName' (string), 'dataType' (enum), 'description' (string), and 'projectId' (UUID).
20. - Save the uploaded file to a configured Object Storage bucket (e.g., using boto3 for AWS S3).
21. - On successful upload, insert a record into PostgreSQL 'datasets' table with metadata and file URL.
22. - If dataType is 'geochemistry' and file is CSV, parse first 10 rows to validate required columns (sample_id, easting, northing, element columns).
23. - Return JSON response with 'datasetId' and 'uploadTimestamp'.
24. Include error handling: reject unsupported file types and missing fields."

Purpose: Accepts and validates user data uploads; stores files and metadata.

25. **AI Job Submission & Worker (Celery)**
26. Prompt:
27. "Create a Celery task called 'run_prospectivity_model' that takes arguments: project_id (UUID), model_id (UUID), area_geojson (dict). It should:
28. 1. Query training features from PostGIS (e.g., use SQLAlchemy to retrieve relevant rasters and layers for the AOI).
29. 2. Load the trained model artifact from object storage (S3 path from ai_models table).
30. 3. Run inference on the grid: tile the AOI into 1 km^2 blocks, generate feature vectors, predict probability for each block.
31. 4. Stitch block predictions into a single GeoTIFF (use rasterio) with appropriate CRS (EPSG:4326).
32. 5. Upload the resulting GeoTIFF to object storage under '/projects/{project_id}/ai_runs/{run_id}/prospectivity.tif'.
33. 6. Update the 'ai_runs' table with status and 'result_url'.
34. 7. If any step fails, update status to 'failed' and log the error.
35. Provide code for Celery configuration (broker URL, result backend) and example Pydantic model for storing run metadata."

Purpose: Backend logic to run ML inference as an asynchronous job.

36. **Retrieve AI Results Endpoint**
37. Prompt:
38. "Implement a FastAPI endpoint 'GET /api/projects/{projectId}/ai-runs/{runId}' that returns metadata about the AI run: model name, submission time, completion time, status, result_url.
39. Also create 'GET /api/projects/{projectId}/ai-runs/{runId}/download' to redirect or stream the GeoTIFF file from object storage with proper headers (Content-Type: 'image/tiff', Content-Disposition: 'attachment; filename=prospectivity.tif').
40. Ensure only project members or admins can access these endpoints (use 'get_current_user' and check project membership in the dependency)."

Purpose: Lets front end poll for AI run status and download results.

41. **Spatial Query & Tile Generation Service**
42. Prompt:
43. "Create a Python microservice 'TileService' using FastAPI that provides endpoints:
44. - GET '/tiles/2d/{z}/{x}/{y}.png' which reads a cached tile from object storage or, if missing, generates a tile by cropping the appropriate raster (e.g., geological map, prospectivity) from PostGIS or GeoTIFF via rasterio. Use Python Imaging Library (PIL) to convert to PNG.
45. - GET '/tiles/3d/{modelId}' which streams a pre-generated 3D model file (glTF or Cesium 3D Tiles) from object storage.
46. Include appropriate caching headers (Cache-Control: max-age=3600) and set MIME types correctly."

Purpose: Serve map tiles and 3D model files on demand, with caching.

47. **Reporting Service – PDF Generation**
48. Prompt:
49. "Implement a FastAPI POST endpoint '/reports/generate' that accepts JSON body: { 'templateName': string, 'projectId': UUID, 'areaGeoJSON': dict (optional), 'notes': string }.
50. - Load the corresponding Jinja2 HTML template and populate it with: project metadata (name, commodity), a base map image (via a Map2D tile snapshot at center of AOI), AI prospectivity summary (embed a small raster thumbnail), and any user notes.
51. - Use WeasyPrint or wkhtmltopdf to convert the HTML to PDF.
52. - Save the PDF to object storage under '/projects/{projectId}/reports/{reportId}.pdf'.
53. - Return JSON with 'reportId', 'downloadUrl', and 'timestamp'.
54. Ensure the endpoint validates that 'templateName' exists and user is authorized to generate reports for that project."

Purpose: Automate creation of branded PDF reports with maps and AI results.

55. **Authentication & Role Management**
56. Prompt:

```
57.  "Write Python code to define a Pydantic model 'UserInDB' with fields:
   user_id (UUID), email (EmailStr), hashed_password (str), roles
   (List[str]), mfa_enabled (bool). Then implement two dependencies:
58.  - 'get_current_user' that decodes JWT, verifies signature, fetches
   user record from database, and returns UserInDB.
59.  - 'require_roles' that takes a list of roles and ensures current user
   has at least one; raise HTTPException 403 otherwise.
60.  Show how to use these dependencies in an example endpoint: 'GET
   /api/admin/users' which returns a list of all users, only accessible to
   users with role 'Administrator'."
```

Purpose: Centralized auth dependencies to secure endpoints by role.

61. **Database Migration Scripts**

```
62.  Prompt:
63.  "Generate Alembic migration scripts for the following schema changes:
64.  1. Create table 'projects' with fields as described (project_id as
   UUID, name, description, commodity, owner_id, geom_extent as geometry,
   created_at, updated_at).
65.  2. Create table 'datasets' with dataset_id, project_id (FK), name,
   data_type, file_url, schema (JSON), visible_to, upload_date.
66.  3. Create table 'drillholes' and 'drill_intervals' as described
   previously.
67.  4. Create table 'ai_models' and 'ai_runs'.
68.  5. Create table 'users' and 'comments'.
69.  For each migration, include both 'upgrade' and 'downgrade' functions.
   Use PostgreSQL GUID extension and PostGIS for geometry."
```

Purpose: Automate creation of database schema with migrations.

70. **CI/CD Pipeline Configuration (GitHub Actions)**

```
71.  Prompt:
72.  "Create a GitHub Actions workflow file '.github/workflows/ci-cd.yml'
   that:
73.  1. Triggers on push to 'main' and pull requests.
74.  2. Checks out code, sets up Python 3.9 and Node 16.
75.  3. Runs linting: 'npm run lint' for front end, 'flake8' for back end.
76.  4. Runs unit tests: 'npm test' for front end, 'pytest' for back end.
77.  5. If on 'main' branch and tests pass, build Docker images for
   'client' and 'server' services, tag with '${{ github.sha }}', and push
   to Docker Hub (authenticate via secrets).
78.  6. Deploy the new images to the staging Kubernetes cluster via
   'kubectl set image' commands.
79.  Include environment variables for DOCKER_USERNAME, DOCKER_PASSWORD,
   KUBE_CONFIG as encrypted secrets."
```

Purpose: Automate testing, building, and deploying microservices via CI/CD.

15.3. DevOps & Infrastructure IaC (Terraform / Kubernetes Manifests)

Product Development Specification (PDS) for GeoVision AI Miner

1. **Terraform VPC & Networking**
2. Prompt:
3. "Write Terraform code to provision a Virtual Private Cloud (VPC) on AWS with:
4. - Two public subnets (for load balancer) and two private subnets (for application pods) across two Availability Zones (e.g., us-east-1a, us-east-1b).
5. - Internet Gateway attached to the VPC.
6. - Public route tables for public subnets.
7. - NAT Gateway in each public subnet and private route tables for private subnets.
8. - Security Group for Kubernetes worker nodes that allows inbound from load balancer on ports 80 and 443, and SSH from a specific IP range.
9. Output VPC ID, subnet IDs, and security group IDs as terraform outputs."

   Purpose: Sets up the foundational network architecture required by the Kubernetes cluster.

10. **Kubernetes Deployment Manifests**
11. Prompt:
12. "Create Kubernetes manifests for the Auth Service and Data Service:
13. - Define a Deployment for 'auth-service' with 2 replicas, container image 'geovision-ai-miner/auth:latest', environment variables from Secrets (JWT_SECRET, DB_URL), liveness/readiness probes on '/health'.
14. - Expose via a Service of type ClusterIP on port 8000.
15. - Create an Ingress resource that maps 'api.geovision-ai-miner.com/auth/*' to the 'auth-service'. Use host-based routing.
16. - Repeat similarly for 'data-service', but using port 8001 and host 'api.geovision-ai-miner.com/data/*'.
17. Include CPU/memory resource requests and limits."

   Purpose: Deploy back-end microservices in Kubernetes with ingress rules.

18. **Persistent Volume Claims for PostgreSQL**
19. Prompt:
20. "Generate Kubernetes YAML for a PersistentVolume (PV) and a PersistentVolumeClaim (PVC) for PostgreSQL:
21. - PV with storageClassName 'gp2', capacity '50Gi', access mode 'ReadWriteOnce', and AWS EBS volume in 'us-east-1'.
22. - PVC named 'postgres-data-pvc' requesting '50Gi' with storageClassName 'gp2' and access mode 'ReadWriteOnce'.
23. - Deployment for 'postgres' using official Postgres image, mounting 'postgres-data-pvc' at '/var/lib/postgresql/data'. Set environment variables POSTGRES_USER, POSTGRES_PASSWORD from secrets.
24. - Expose Postgres as a ClusterIP service on port 5432."

   Purpose: Ensures Postgres data persists across pod restarts.

25. **Redis & RabbitMQ Deployment**
26. Prompt:
27. "Write Helm values or Kubernetes deployment YAML for:

28. - Redis: one master and one replica, using the official Bitnami Helm chart, with password set via Kubernetes Secret. Expose via ClusterIP.
29. - RabbitMQ: single replica cluster using the official RabbitMQ Helm chart, with default user and password set via Secret. Expose via ClusterIP on port 5672.
30. Provide example Helm install commands with custom values file."

Purpose: Sets up caching (Redis) and job queue broker (RabbitMQ) for Celery.

31. **Object Storage Configuration (S3 Bucket & Policies)**
32. Prompt:
33. "Using Terraform, create an AWS S3 bucket named 'geovision-ai-miner-data-{randomSuffix}'. Enable server-side encryption (AES256) and versioning. Attach a bucket policy that:
34. - Allows only the IAM role 'geovision-ai-miner-app-role' to PUT/GET objects.
35. - Blocks all public access.
36. - Logs all S3 access events to CloudTrail.
37. Provide Terraform IAM role and policy definitions for 'geovision-ai-miner-app-role' granting s3:GetObject and s3:PutObject on the bucket."

Purpose: Programmatically provision an encrypted, access-controlled S3 bucket.

38. **CI/CD Secrets Management**
39. Prompt:
40. "Generate a sample Kubernetes Secret manifest named 'app-secrets' containing:
41. - JWT_SECRET: base64-encoded value 'your_jwt_secret_here'.
42. - DB_URL: base64-encoded 'postgres://user:password@postgres-service:5432/geovision_ai_miner'.
43. - S3_ACCESS_KEY, S3_SECRET_KEY: placeholder base64 values.
44. - REDIS_PASSWORD: placeholder base64 value.
45. Also show how to reference this secret in the Deployment environment variables using 'valueFrom: secretKeyRef'."

Purpose: Illustrates how to manage sensitive environment variables in Kubernetes.

46. **Monitoring & Alerting Setup**
47. Prompt:
48. "Write a PrometheusConfig YAML that scrapes metrics from endpoints:
49. - 'auth-service:8000/metrics'
50. - 'data-service:8001/metrics'
51. - 'map-service:8002/metrics'
52. Then create alerting rules:
53. - Alert 'HighCPUUsage' if job:container_cpu_usage_seconds_total > 0.8 for 5 minutes.
54. - Alert 'PodRestartLoop' if kube_pod_container_status_restarts_total > 3 within 10 minutes.
55. Provide example Grafana dashboard JSON that shows a panel for CPU usage per service and a panel for number of active users over time."

Purpose: Configures real-time monitoring and sets up basic alerts.

56. **Database Migration CI Step**
57. `Prompt:`
58. `"Add a new step to the existing CI workflow that runs Alembic migrations:`
59. `- Name: 'Run DB Migrations'.`
60. `- Use official Python 3.9 image.`
61. `- Check out repo, install dependencies (requirements.txt), set environment variable 'DATABASE_URL' from GitHub secret.`
62. `- Run 'alembic upgrade head'.`
63. `- On failure, exit with non-zero code.`
64. `Provide the YAML snippet to insert into '.github/workflows/ci-cd.yml' after 'Run Tests' and before 'Build Docker Images' steps."`

    Purpose: Ensures database schema is up to date before deploying new code.

65. **Terraform for Kubernetes Cluster Provisioning**
66. `Prompt:`
67. `"Write Terraform code to provision an Amazon EKS cluster named 'geovision-ai-miner-eks' with:`
68. `- Kubernetes version 1.22.`
69. `- Two node groups (dev nodes: 2 instances t3.medium; prod nodes: 4 instances m5.large).`
70. `- IAM roles and policies for node groups.`
71. `- Outputs: cluster_endpoint, cluster_certificate_authority_data.`
72. `Use AWS EKS Terraform module and include necessary VPC/subnet IDs as variables."`

    Purpose: Automates creation of a managed Kubernetes cluster.

73. **Automated Backup & Restore Scripts**
74. `Prompt:`
75. `"Create a Bash script 'backup_postgres.sh' that:`
76. `- Runs 'pg_dump' to export the 'geovision_ai_miner' database to a local file named 'backup_$(date +%F_%H%M%S).sql'.`
77. `- Compresses the SQL file with gzip.`
78. `- Uploads the compressed file to S3 bucket 'geovision-ai-miner-backups' under '/backups/{date}/'.`
79. `- Logs success or failure to '/var/log/geovision-ai-miner/backup.log'.`
80. `Also create 'restore_postgres.sh' that downloads a specified backup from S3, unzips it, and uses 'psql' to restore into the database (prompting for confirmation if the database isn’t empty).`
81. `Include environment variable checks (e.g., AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, PGPASSWORD) and exit if missing."`

    Purpose: Automates regular database backups and safe restores.

---

16. Summary & Next Steps

Product Development Specification (PDS) for GeoVision AI Miner

1. **Review & Approve PDS:** Share this PDS with stakeholders (technical leads, pilot partners) for feedback and sign-off. Adjust any functional or non-functional requirements based on their priorities.
2. **Assign Development Tasks:** Break down the PDS into epics and user stories in your project management tool (e.g., Jira, Trello). Prioritize M1 tasks (core MVP) first.
3. **Kick Off Development Sprints:** Use the prompts above to seed boilerplate code in each sprint. For example, start with authentication and basic project CRUD; then integrate mapping components; then data upload, etc.
4. **Set Up DevOps Pipeline:** Use the DevOps prompts to provision infrastructure (Terraform), configure CI/CD, and deploy initial microservices to a staging environment.
5. **Begin Pilot Integrations:** Once core features are ready (M1), onboard pilot users in one focus country to test data ingest and AI workflows. Gather feedback and iterate.
6. **Plan Training & Workshops:** Prepare training materials alongside feature development so that when Beta is released (end of Month 6), you can immediately onboard pilot teams with minimal friction.
7. **Iterate & Expand:** Incorporate feedback, add new features (3D, advanced reports), onboard new partners, and steadily work toward Version 1.0 launch at Month 12.

By following this Product Development Specification and leveraging the provided AI assistant prompts, your team can rapidly bootstrap GeoVision AI Miner's codebase, infrastructure, and deployment processes—ensuring a robust, secure, and user-friendly platform that meets the needs of geoscientists and decision-makers across Africa. Good luck with building and launching GeoVision AI Miner!