B.M.S. COLLEGE OF ENGINEERING BENGALURU

Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

"AI POWERED DSS FOR FOREST RIGHT ACT"

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

NAME OF THE CANDIDATES

Palepu Vishal (1BM23CS224)

Pathamekala Yogesh Reddy (1BM23CS228)

Pragathi Y raj (1BM24CS415)

Resham Santosh Naik (1BM24CS417)

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2025-26

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **Palepu Vishal, P. Yogesh Reddy, Pragathi Y Raj, Resham Santosh Naik** students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "**AI powered DSS for Forest Right Act**" has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester August 2025- December 2025. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

Palepu Vishal (1BM23CS224)

Pathamekala Yogesh Reddy (1BM23CS228)

Pragathi Y raj (1BM24CS415)

Resham Santosh Naik (1BM24CS417)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled "**AI powered DSS for Forest Right Act**" has been carried out by **Vishal(1BM23CS224), Yogesh(1BM23CS228), Pragathi(1BM24CS415), Resham(11BM23CS417)** during the academic year 2025-2026.

Signature of the Faculty Incharge:

Dr. Adarsha Sagar HV
Assistant Professor

Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

Table of Contents

CHAPTER 1

Problem Statement

The Scheduled Tribes and Other Traditional Forest Dwellers (Recognition of Forest Rights) Act, 2006 (FRA) is a landmark legislation in India aimed at recognizing the rights of forest-dwelling communities to land and resources. However, over a decade and a half since its enactment, the implementation of the FRA has been uneven, slow, and fraught with challenges.
This leads to the continued marginalization of the intended beneficiaries and prevents the Act from achieving its full potential for both social justice and forest conservation.
There is a critical gap in the availability of a unified, transparent, and data-driven system to monitor the implementation of the FRA across its entire lifecycle—from claim filing and processing to title distribution and post-rights governance.
Current monitoring is often siloed, manual, and non-spatial, making it difficult for various stakeholders (government agencies, NGOs, communities themselves) to get a holistic, real-time view of the status, bottlenecks, and geographical disparities in FRA implementation.
To address this, we propose the development of an AI-powered FRA Atlas and a Web GIS-based Decision Support System (DSS). This integrated platform will leverage geospatial technology and artificial intelligence to create a "single source of truth" for FRA data, enabling:

- **Visualization:** Interactive maps displaying the spatial distribution of claims (filed, approved, rejected, pending) at various administrative levels.
- **Monitoring:** Real-time dashboards tracking key performance indicators (KPIs) like processing time, approval/rejection rates, and area recognized.
- **Analysis:** AI/ML models to identify patterns, predict bottlenecks, and flag potential areas of conflict or procedural delay.
- **Transparency & Accountability:** Providing all stakeholders with easy access to anonymized, actionable data.
- **Evidence-Based Decision Making:** Assisting government officials in resource allocation, policy intervention, and targeted support.

Specific Challenges in the Focus States:

- **Madhya Pradesh:** High volume of claims with significant variations in implementation across districts; issues of rejected claims and potential for conflict.
- **Odisha:** Presence of particularly vulnerable tribal groups (PVTGs); complexities of Community Forest Resource (CFR) rights recognition.
- **Telangana:** Issues related to the interface between FRA and other land-related schemes; need for integration with satellite data for evidence-based claim verification.
- **Tripura:** High density of tribal population and successful individual titles, but challenges in realizing the full potential of CFR rights and managing forest resources post-recognition.

CHAPTER 2

Software Requirement Specification

1. Introduction

1.1 Purpose of this Document

This document provides a detailed description of the requirements for the **AI-Powered FRA Atlas and Web GIS-based Decision Support System (DSS)**. It is intended to serve as a formal agreement between the developers and the stakeholders, including the Ministry of Tribal Affairs and State Tribal Welfare Departments, to ensure a unified understanding of the system's objectives, capabilities, and constraints.

1.2 Scope of this Document

The system to be developed is an integrated software platform designed to revolutionize the monitoring and implementation of the Forest Rights Act (FRA), 2006. Its core purpose is to digitize legacy FRA records, create a centralized visual FRA Atlas using AI and satellite data, map community assets, and provide a DSS for targeted welfare scheme planning for FRA beneficiaries.

1.3 Overview

The AI-Powered FRA Atlas is designed to bring transparency and efficiency to FRA implementation. The product will include:

- **WebGIS Portal:** For citizens and NGOs to view FRA data and maps.
- **AI Processing Engine:** For digitizing documents and analyzing satellite imagery.
- **Decision Support System (DSS):** For officials to plan and layer development schemes.
- **Centralized Geodatabase:** Stores all FRA records, spatial data, and asset maps.

2. General Description

2.1 General Functions

- Digitize legacy FRA records using AI (OCR/NER).
- Create a visual FRA Atlas with interactive maps.
- Automatically map community assets (land, water, forests) from satellite data.
- Integrate a DSS to recommend Central Sector Schemes (CSS) for FRA beneficiaries.
- Generate real-time reports on FRA implementation progress.

2.2 User Characteristics

User Class	Expertise	Primary Tasks
System Administrator	High IT proficiency	User management, system maintenance, data pipeline oversight
State/District Official	Moderate IT proficiency	FRA progress monitoring, report generation, DSS utilization
Field Officer	Basic IT proficiency	Spatial claim verification, status updates
Citizen/NGO	Basic IT proficiency	Public dashboard access, claim status tracking

3. Functional Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- WebGIS Portal: Responsive design compatible with Chrome, Firefox, Safari, Edge
- Admin Dashboard: Role-based access control interface
- Mobile Interface: Responsive design for tablet and mobile access
- Public Dashboard: Read-only interface for citizens and NGOs

3.1.2 Hardware Interfaces

- Support for standard government hardware specifications
- Compatibility with existing printers and scanning devices
- Cloud-server deployment capability

3.1.3 Software Interfaces

- GeoServer/PostGIS for spatial data serving
- Python/Django/Flask for backend services
- React/Leaflet for frontend mapping
- External APIs: Satellite imagery, PM Gati Shakti, authentication services

3.2 Functional Requirements

3.2.1 Data Digitization Module

ID	Requirement
FR-DM-01	System shall accept bulk upload of scanned FRA documents (PDF, JPEG, PNG)
FR-DM-02	OCR engine shall extract text with minimum 95% accuracy
FR-DM-03	NER model shall identify: Patta Holder Name, Father's Name, Village, Area
FR-DM-04	System shall validate extracted data against predefined rules
FR-DM-05	System shall generate data quality reports

3.2.2 Spatial Data Management

ID	Requirement
FR-SD-01	System shall accept shapefile, GeoJSON, KML formats
FR-SD-02	Automatic geometry validation and projection correction
FR-SD-03	Spatial data linkage with corresponding FRA records
FR-SD-04	Version control for spatial data updates

3.2.3 AI Asset Mapping

ID	Requirement
FR-AI-01	System shall process multispectral satellite imagery
FR-AI-02	CNN model shall classify: Agricultural Land, Forest Cover, Water Bodies
FR-AI-03	Minimum 85% classification accuracy for all land use classes
FR-AI-04	Generate GeoJSON vector layers of detected assets

3.2.4 WebGIS Portal

ID	Requirement
FR-WG-01	Interactive map with multiple base layers (Satellite, Topographic)
FR-WG-02	Toggleable data layers (IFR, CFR, Land Use, Water Index)
FR-WG-03	Advanced filtering by State/District/Block/Village/Tribal Group
FR-WG-04	Real-time FRA progress dashboard
FR-WG-05	Public access to anonymized data

3.2.5 Decision Support System

ID	Requirement
FR-DS-01	Geographic area selection (village/block/district level)
FR-DS-02	CSS eligibility rule engine with configurable parameters
FR-DS-03	AI-based intervention prioritization
FR-DS-04	PDF report generation with recommendations
FR-DS-05	Scheme impact simulation

4. Interface Requirements

4.1 Software Interfaces
- Centralized Geodatabase (PostgreSQL/PostGIS) for all spatial and textual data.
- AI/ML Model Servers for document processing and satellite image analysis.
- WebGIS Portal (Leaflet/OpenLayers) for map visualization.

4.2 Communication Interfaces
- Secure HTTPS communication for all data transfer.
- API integration with external systems (PM Gati Shakti, Satellite Imagery APIs).
- Role-based access control for secure data access.

5. Performance Requirements
- Map layers in the WebGIS portal should load within **5 seconds**.
- The system must support **5,000+ concurrent users** (officials and public).
- AI model inference (e.g., land classification) should complete within **3 minutes** per village.
- Database queries for reports should respond within **3 seconds**.

6. Non-Functional Attributes
- **Security:** End-to-end encryption and strict role-based access control.
- **Reliability:** 99.5% uptime with automated backup and disaster recovery.
- **Scalability:** Modular architecture to scale from 4 states to nationwide deployment.
- **Usability:** Intuitive interface available in English and Hindi for diverse users.

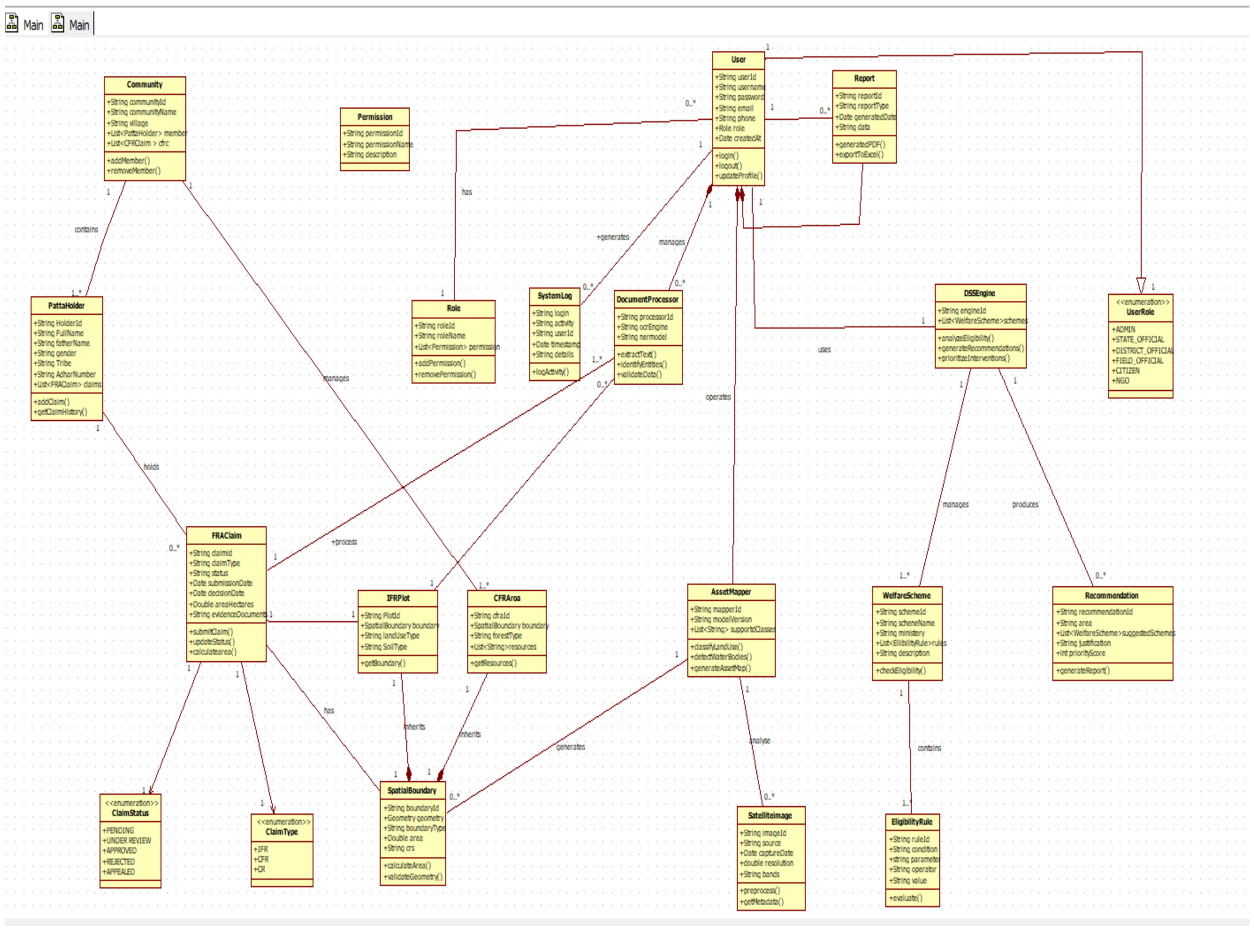
8. Preliminary Schedule and Budget

8.1 Development Schedule
- **Phase 1 (Design & Data Protocol): 2 months**
- **Phase 2 (Core Development & AI Engine): 6 months**
- **Phase 3 (Pilot & Integration - 2 States): 4 months**
- **Phase 4 (Scale-Up & Handover): 6 months**
  Total Duration: 18 months

8.2 Budget
The estimated budget for the development, testing, and deployment of the AI-Powered FRA Atlas and DSS is **$1,200,000**. This includes costs for AI model development, cloud infrastructure, training, and one year of support.

CHAPTER 3

Class Modeling



Community

- **Responsibility:** Model a village/community unit that groups PattaHolders and acts as a scope for claims and boundaries.
- **Key attributes:** communityId, communityName, village, members (list of PattaHolder), LST/CR fields shown in diagram.
- **Key methods:** addMember(), removeMember().
- **Relationships:** *Contains* PattaHolder (1..*).

PattaHolder

- **Responsibility:** Person or entity who holds a land title ("patta") and files claims.
- **Key attributes:** holderId, fullName, fatherName, gender, age, tribe, adharNumber, claims (list of FRAClaim).
- **Key methods:** addClaim(), getClaimHistory().
- **Relationships:** *Belongs to* Community; *Holds* FRAClaim.
- **Advanced features:** identity verification (OTP / e-KYC), linking to national ID registry, audit trail on profile edits, role-based limited access to sensitive PII.

FRAClaim

- **Responsibility:** Represents a formal land claim (submission) under the FRA process.
- **Key attributes:** claimId, claimType (enum), claimStatus (enum), submissionDate, decisionDate, doubleArcHolders?, evidenceDocuments (attachments), area fields.
- **Key methods:** submitClaim(), updateStatus(), calculateArea().
- **Relationships:** *Held* by PattaHolder; references SpatialBoundary (or FRPlot) and may be processed by DocumentProcessor and evaluated by DSS_Engine.
- **Advanced features:** immutable claim history (append-only), transactional submit with pessimistic locking to avoid duplicates, digital signatures for submitted docs, PDF generation and notarization stamps, automated completeness checks.

ClaimStatus (enum)

- **Values:** PENDING, UNDER_REVIEW, APPROVED, REJECTED, APPEALED.
- **Use:** drive workflow and UI state transitions.

ClaimType (enum)

- **Values shown:** IFR, CFR, other types (diagram shows IFR, CFR, maybe others).
- **Use:** selects validation and processing rules (different workflows and evidence requirements).

SpatialBoundary

- **Responsibility:** Generic geospatial representation used for plots/areas.
- **Key attributes:** boundaryId, geometry (GeoJSON or WKT), boundaryType, area.
- **Key methods:** calculateArea(), validateGeometry().
- **Relationships:** Parent for IFRPlot, CFRArea, FRPlot (diagram shows inheritance).
- **Advanced features:** store as PostGIS geometry, topology validation, geometry versioning, support for multi-part polygons, spatial indexing for fast queries.

IFRPlot / CFRArea / FRPlot (inherited spatial types)

- **Responsibility:** Specific boundary types for different claim kinds (inherit spatial properties).
- **Key attributes:** plotId, landuseType, soilType, resources etc.
- **Key methods:** getBoundary(), getResources().
- **Advanced features:** link to cadastral data, overlay with satellite-derived land-cover, change detection history.

Permission

- **Responsibility:** fine-grained permission entity (e.g., permissionId, permissionName, description).
- **Use:** building block for RBAC.

Role

- **Responsibility:** role-based access control grouping of Permissions.
- **Key attributes/methods:** roleId, roleName, list of Permission, addPermission(), removePermission().
- **Relationships:** assigned to User; system uses roles to authorize actions.
- **Advanced features:** hierarchical roles, time-limited role assignments, separation-of-duty constraints, dynamic roles (contextual: e.g., per-community admin).

User

- **Responsibility:** system user (admin, official, field staff, NGO, citizen).
- **Key attributes:** userId, username, password (store hashed), email, phone, role, dateCreated.
- **Key methods:** login(), logout(), updateProfile().
- **Relationships:** *Manages/generates* many system processes; UserRole enum in diagram enumerates types (ADMIN, STATE_OFFICIAL, DISTRICT_OFFICIAL, FIELD_OFFICIAL, CITIZEN, NGO).
- **Advanced features:** MFA, SSO integration, session management, token revocation, audit of privileged actions, account lockout after failed attempts.

UserRole (enumeration)

- **Values:** ADMIN, STATE_OFFICIAL, DISTRICT_OFFICIAL, FIELD_OFFICIAL, CITIZEN, NGO
- **Use:** quick classification for UI and coarse-grained authorization.

Report

- **Responsibility:** encapsulate generated reports (claim lists, recommendations, asset maps).
- **Key attributes:** reportId, reportType, dateGenerated, stringData / binaryData.
- **Key methods:** generatePDF(), exportToExcel().
- **Relationships:** generated by User, DSS_Engine, Recommendation.
- **Advanced features:** templating (HTML → PDF), scheduled report generation, role-based visibility, searchable metadata, signed reports.

SystemLog

- **Responsibility:** audit & activity logging.
- **Key attributes:** logId, login, activity, userId, timestamp, details.
- **Key methods:** addEntry(), query().
- **Relationships:** logs actions by User, processing by DocumentProcessor, and decisions from DSS_Engine.

DocumentProcessor

- **Responsibility:** process incoming documents & evidence (OCR, validate, extract metadata).
- **Key attributes:** processorId, engine, model (if ML-based).
- **Key methods:** extractText(), identifyEntities(), validateDocument().
- **Relationships:** used by User during claim submission; outputs feed to SystemLog and DSS_Engine.
- **Advanced features:** OCR pipeline, ML models for document classification, image quality checks, automated redaction of PII, document versioning, asynchronous processing queue (worker pool).

DSS_Engine (Decision Support System)

- **Responsibility:** core decision logic — analyze eligibility, generate recommendations, prioritize interventions.
- **Key methods:** analyzeEligibility(), generateRecommendations(), prioritizeInterventions().
- **Relationships:** *Consumes* FRAClaim, WelfareScheme, AssetMapper outputs, SatelliteImage data; produces Recommendation objects and reports.
- **Advanced features:** modular rule-based engine + ML scoring hybrid, explainability (why a recommendation was made), pluggable models, thresholds configurable, simulated scenario testing, feedback loop to retrain models from appeal outcomes.

WelfareScheme

- **Responsibility:** represents government schemes available for claimants.
- **Key attributes:** schemeId, schemeName, ministry, eligibilityCriteria, description.
- **Key methods:** checkEligibility().
- **Relationships:** contains EligibilityRules; used by DSS_Engine to recommend applicable schemes.
- **Advanced features:** versioned rules, dynamic rule engine (Drools or custom), scheduled activation/expiry, integration with external scheme registries.

EligibilityRule

- **Responsibility:** discrete rule used to evaluate scheme eligibility (ruleId, condition, parameter, operator).
- **Key method:** evaluate().
- **Relationships:** contained by WelfareScheme.
- **Advanced features:** rules as data (stored and editable through UI), safe sandboxed expression evaluation, audit of rule changes.

Recommendation

- **Responsibility:** output of DSS (what schemes to apply, priority actions).
- **Key attributes:** recommendationId, recommendationText, area, welfareScheme, priorityScore.
- **Key methods:** generateReport().
- **Relationships:** produced by DSS_Engine, consumed by User and Report.
- **Advanced features:** store reasoning or provenance, attach confidence scores, support rebuttal/appeal workflows.

AssetMapper

- **Responsibility:** identify and map assets (buildings, water bodies, vegetation) from imagery / spatial data.
- **Key attributes:** mapperId, modelVersion, supportClasses.
- **Key methods:** detectFeatures(), generateAssetMap().
- **Relationships:** uses SatelliteImage, produces asset layers used by DSS_Engine/Report.
- **Advanced features:** tiling & vector tile export, integration with Mapbox/Leaflet, feature change detection over time, team annotation tools.

SatelliteImage

- **Responsibility:** store metadata & pointers to imagery used for analysis.
- **Key attributes:** imageId, source, captureDate, resolution, bands.
- **Key methods:** preprocess(), getMetadata().
- **Relationships:** consumed by AssetMapper and DSS_Engine.

EligibilityRule (again)

- (Covered above; diagram also shows it contained in WelfareScheme and linked to evaluation functions.)

Additional cross-cutting components & flows

- **Document → Claim Flow (typical):**
    1. User (PattaHolder) creates FRAClaim and attaches SpatialBoundary + documents.
    2. DocumentProcessor validates docs (OCR, metadata extraction).
    3. AssetMapper and SatelliteImage analysis enrich the claim with remote-sensing evidence.
    4. DSS_Engine runs WelfareScheme EligibilityRules → produces Recommendation.
    5. Report is generated; SystemLog records all actions. Role / Permission govern who can change the claim status.
- **Audit & Governance:** every state change on FRAClaim, WelfareScheme rules, Recommendation should be logged in SystemLog and be attributable to User.

Suggested advanced technical features (architecture & ops)

1. **Geospatial stack:** PostGIS / GeoPackage for geometries; vector tiles for map UI; spatial indexes for fast queries.
2. **Document & image pipeline:** S3 (or equivalent) for storage + background workers for OCR and ML inference (Celery/Kafka + workers).
3. **Hybrid Decision Engine:** rule engine for deterministic policy rules + ML model for prioritization and anomaly detection. Store explainability traces with recommendations.
4. **Security & Privacy:** hashed passwords, encrypted PII at rest, role-based access control, audit trails, GDPR/Local privacy compliance.
5. **Scalability:** microservices for heavy components (AssetMapper, DocumentProcessor, DSS), API gateway, horizontal scaling for worker tiers.
6. **Monitoring & Observability:** metrics (Prometheus), logs (ELK), alerting on failures (OCR failures, geometry validation errors).
7. **Data quality & governance:** geometry validation, duplicate detection (spatial & attribute), provenance tracking of satellite imagery and model versions.
8. **User workflows & UX:** stepwise claim submission with client-side geometry drawing/editing, live area calculator, mobile-first offline-capable forms for field officials.

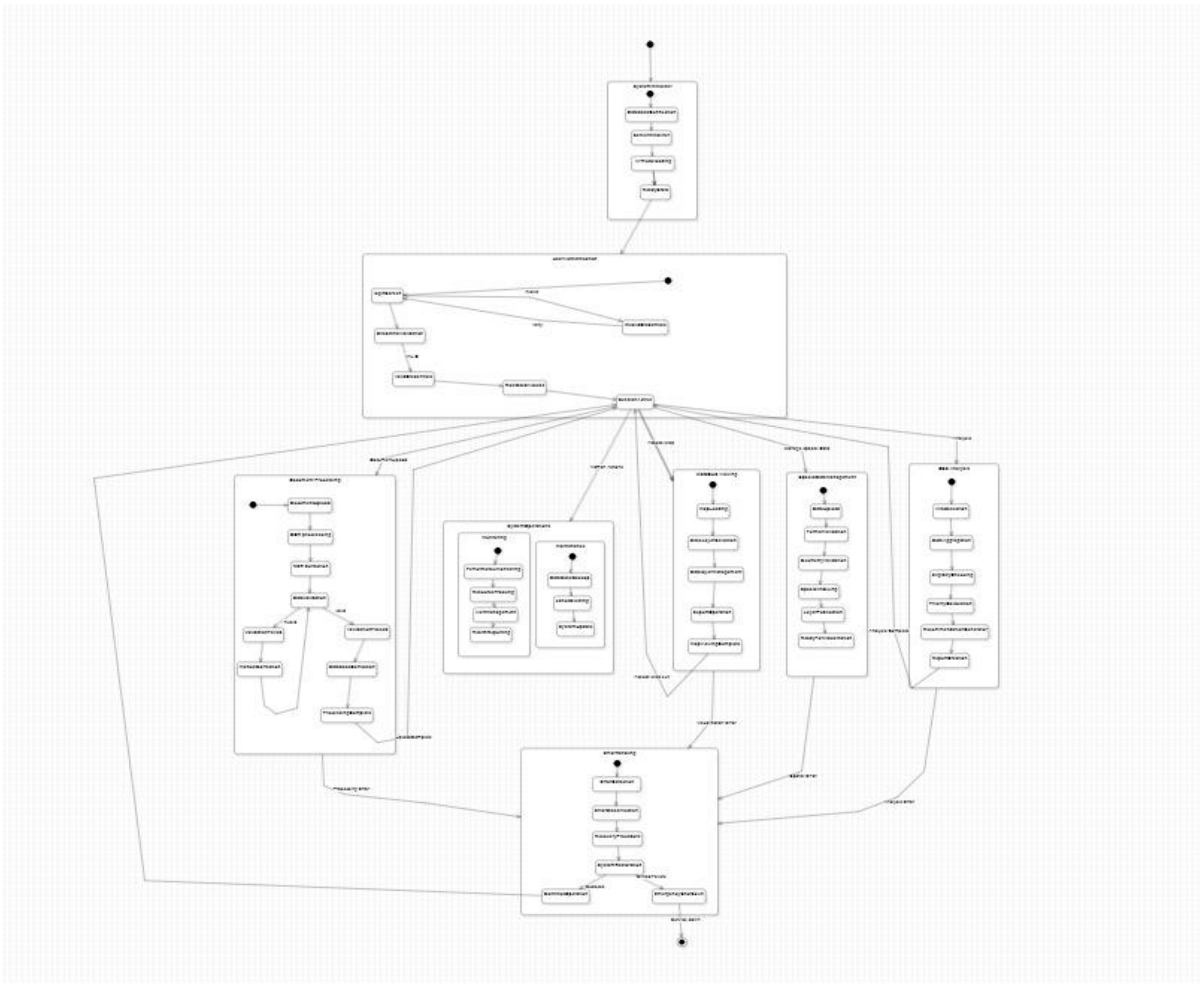
9. **Testing & CI/CD:** unit tests for evaluation/rule code, integration tests for document pipeline, model validation harness.
10. **Explainability & Appeals:** store justification and evidence for DSS decisions so users can appeal; track appeal resolution in FRAClaim.

Prioritized list of concrete additions (start here)

1. Add **audit trail** fields to FRAClaim (createdBy, createdAt, version, changeLog).
2. Persist SpatialBoundary as GeoJSON + PostGIS geometry, add validateGeometry() server-side.
3. Implement DocumentProcessor as an asynchronous worker; keep lightweight sync checks in the request path.
4. Implement DSS_Engine in two parts: rule-evaluator (configurable by admins) + ML scorer (returns priorityScore).
5. Add Report templating that includes embedded geometry snapshots and recommendation provenance.

CHAPTER 4

State Modeling



Global notes / assumptions

- Primary entity: FRAClaim. States and events below are focused on the claim lifecycle, because that is the central flow connecting PattaHolder, DocumentProcessor, SpatialBoundary, DSS, Report, and Users.
- Supporting components (DocumentProcessor, AssetMapper, DSS) have smaller state-machines that interact with the FRAClaim machine (via events and callbacks).
- Events can be triggered by users (PattaHolder / Field Official), system processes (OCR complete, model finished), or scheduled processes (timeout, SLA).
- All transitions should be logged (SystemLog) and include actor, timestamp, reason, metadata (evidence references), to enable audit & appeals.

1) FRAClaim state machine (recommended canonical states & events)

1. **Draft**
    - **Relevance:** Claim is being prepared by the PattaHolder or field officer. Not visible to reviewers. Allows iterative edits and attachments.
    - **Typical duration:** short; frontend autosave.
2. **Submitted**
    - **Relevance:** Claim form + minimal required documents uploaded and the claimant pressed Submit. This triggers initial validation and document ingestion.
    - **Visibility:** visible to intake/admin queue.

3. **Intake Validation**

   - **Relevance:** Automated checks run (document presence, basic OCR checks, geometry validity, duplicates). It's a sync/short async composite state before formal review.

   - **Purpose:** quickly reject incomplete submissions and queue complete ones for processing.

4. **Processing**

   - **Relevance:** Heavy processing happens: DocumentProcessor OCR & classification, AssetMapper analysis, spatial checks, and DSS eligibility analysis.

   - **Note:** This represents an *orchestration* of multiple sub-jobs (could be modeled as parallel sub-states).

5. **Under Review**

   - **Relevance:** Human officials review the outputs, evidence, and system recommendations; make decisions. This is the manual decision state.

   - **Actors:** field official / district official / committee.

6. **Pending Evidence**

   - **Relevance:** Reviewer requested additional evidence (e.g., better document scans or clarifications). Claim waits for claimant to upload more docs.

   - **Purpose:** avoids hard reject for minor missing items; provides appeal-friendly flow.

7. **Approved**

   - **Relevance**: Claim accepted; leads to downstream processes (report generation, register update, issuance of title).
   - **Actions:** generate official certs, update registry, notify claimant, begin benefit allocation.

8. **Rejected**

   - **Relevance:** Claim denied. Must include reason, evidence reference, and appeal instructions.
   - **Actions:** record rejection reason; allow appeal submission.

9. **Appealed**

   - **Relevance:** Claimant appealed a rejection; triggers secondary review/tribunal workflow.
   - **Actions:** escalate to higher-level official, freeze some actions, track appeal outcomes**.**

10. **Closed**

    - **Relevance:** Final end-state after approval actions complete or after appeals exhausted. Read-only. Records remain for audit.
    - **Actions:** archive evidence, export to reports, finalize logs.

11. **Withdrawn**

   - **Relevance:** Claimant withdrew voluntarily. Stops further processing. Archived with reason.

Events (that drive transitions) — explanation & relevance

- **save_draft:** triggered by claimant saves progress → Draft**.**
- **submit_claim:** user presses submit → Draft → Submitted.
- **basic_validation_passed:** system checks passed (required docs present, minimum metadata) → Submitted → Intake Validation → proceed to Processing.
- **basic_validation_failed:** required fields or docs missing → Submitted → Pending Evidence (or back to Draft).
- **enqueue_processing:** submitted claim is queued for heavy processes → Intake Validation → Processing.
- **doc_processing_completed:** DocumentProcessor finished OCR & classification → signals Processing (used to progress when all sub-jobs done).
- **asset_mapping_completed:** AssetMapper finishes imagery analysis → Processing (used similarly).
- **dss_analysis_completed:** DSS generated recommendations → Processing (all results aggregated).
- **processing_failed:** retry or mark for manual intervention → Processing → Pending Evidence or Under Review with "needs manual check".

- **request_additional_evidence:** reviewer asks for docs → Under Review → Pending Evidence.
- **evidence_uploaded:** claimant uploads requested docs → Pending Evidence → Under Review or Processing (if more work required).
- **manual_approve:** reviewer approves → Under Review → Approved.
- **manual_reject:** reviewer rejects → Under Review → Rejected.
- **appeal_filed:** claimant appeals a rejection → Rejected → Appealed.
- **appeal_resolved_approve:** appeal decided in favor → Appealed → Approved.
- **appeal_resolved_reject:** appeal denied → Appealed → Closed (or Rejected with final flag).
- **withdraw_claim:** claimant withdraws → any active state → Withdrawn.
- **finalize_actions_completed:** post-approval tasks done (register update, report) → Approved → Closed.
- **timeout_no_response:** SLA breach (e.g., claimant doesn't respond to evidence request) → Pending Evidence → Closed or Rejected (policy-defined).
- **admin_override:** authorized admin force-transition (e.g., emergency approval) → jumps to target state, must be audited**.**

Example transitions (compact)

- Draft → *(submit_claim)* → Submitted → *(basic_validation_passed)* → Intake Validation → *(enqueue_processing)* → Processing → *(all sub-jobs complete)* → Under Review → *(manual_approve)* → Approved → *(finalize_actions_completed)* → Closed.
- Submitted → *(basic_validation_failed)* → Pending Evidence → *(evidence_uploaded)* → Processing.
- Under Review → *(manual_reject)* → Rejected → *(appeal_filed)* → Appealed → *(appeal_resolved_approve)* → Approved.

2) Supporting components: state machines & events

These are important because FRAClaim's Processing depends on them. Model them as sub-state machines or separate services that emit events back to the main claim state machine.

DocumentProcessor (per-claim-document)

States:

- Uploaded (user uploaded doc)
- Queued
- Processing (OCR, classification)
- Processed (text extracted, entities identified)
- Rejected (unreadable or invalid)

Events:

- doc_uploaded → Uploaded → queue_doc
- doc_queued → Queued
- doc_processing_started
- doc_processing_succeeded → Processed (includes extracted metadata)
- doc_processing_failed → Rejected / ManualReviewNeeded (with failure reason)
- doc_manual_review_passed → Processed

Relevance: allows asynchronous heavy jobs; DocumentProcessor returns evidence and metadata used by DSS and reviewers.

AssetMapper (per claim)

States:

- NotRequested
- Queued
- Processing
- Completed

Events: request_asset_map, mapping_complete, mapping_failed, retry_mapping.

Relevance: runs model to detect assets, change detection used in decisioning.

DSS_Engine (per claim analysis job)

States:

- Idle
- Running
- Succeeded
- Failed

Events: run_analysis, analysis_succeeded, analysis_failed.

Relevance: produces recommendations and priorityScore used by reviewers.

3) Advanced features to include (production-ready ideas)

1. **Parallel & composite states**
    - Model Processing as a composite where DocumentProcessor, AssetMapper and DSS run in parallel. Claim moves to Under Review only when *all* required sub-jobs succeed, or to Pending Evidence / ManualCheck if any fail.

2. **Event-driven architecture**

   - Implement each event as a message on a durable message bus (Kafka / RabbitMQ). Services subscribe and publish events (doc_processing_completed, dss_analysis_completed), enabling resilient, auditable workflows.

3. **SLA & timers**

   - Configure timers on states like Pending Evidence (e.g., 14 days). If evidence_uploaded not seen before timeout, auto-escalate (timeout_no_response), send reminders, or auto-close per policy.

4. **Idempotency & retries**

   - Make processing events idempotent (include event_id, claim_version); retry failed jobs with exponential backoff; track retry counts and escalate after N failures**.**

5. **Compensation / Saga patterns**

   - Multi-step side-effects (e.g., payment issuance after approval) require compensation on failures. Use Saga orchestration to roll back or compensate if downstream steps fail.

6. **State versioning & optimistic locking**

   - Add claim.version for concurrency control to avoid race conditions when multiple actors change state. Reject conflicting transitions with clear error to client.

7. **Role-based transitions**

   - Restrict events by role/permission. E.g., only DISTRICT_OFFICIAL or higher can trigger manual_approve for certain claim types.

8. **Provenance & explainability**

   - For each transition (especially Approve/Reject), store the evidence set and DSS traces (rules triggered, scores, model version) to help appeals.

9. **Visual timeline & notifications**

   - Create timeline UI showing state history, actors, attachments. Attach notifications (SMS/email/in-app) for key events (submit, request_evidence, approved, rejected, appeal_result).

10. **Audit & tamper-evidence**

    - Hash chains linking state transitions so tampering is detectible. Store logs in append-only store or ledger (if required by policy).

11. **Simulation & test harness**

    - Provide a "dry-run" mode for DSS_Engine and for state transitions to test policies and rule changes without affecting production.

12. **Batch operations & bulk decisions**

    - Admins sometimes need to approve many claims of same type/region; allow bulk transitions but ensure each claim retains its own audit trail.

13. **Partial approvals / multi-issue claims**

    - Claims may contain multiple parcels or multiple entitlement items. Support partial approvals where sub-claims are individually state-managed.

14. Escalation & SLA dashboards

- Expose dashboards listing claims approaching SLA breach by state, so managers can reassign resources.

15. Safe manual override & lock

- Allow override, but require reasons and two-person rule for sensitive cases.

16. Accessibility & offline

- Allow field officers to work offline and queue events for later sync; system resolves conflicts using state versioning on sync.

4) Practical representation: a compact state/event table (for implementation)

Current State	**Event**	**New State**	**Notes**
Draft	submit_claim	Submitted	Validate minimal fields
Submitted	basic_validation_passed	Intake Validation	Auto-checks ok
Submitted	basic_validation_failed	Pending Evidence	Ask claimant for docs
Intake Validation	enqueue_processing	Processing	Kick off sub-jobs

Current State	Event	New State	Notes
Processing	all_subjobs_succeeded	Under Review	Human reviewer picks up
Processing	subjob_failed	Pending Evidence / ManualReviewNeeded	Depends on failure type
Under Review	manual_approve	Approved	Generate outputs
Under Review	request_additional_evidence	Pending Evidence	Document / geo fix needed
Pending Evidence	evidence_uploaded	Processing / Under Review	Re-trigger processing or direct review
Under Review	manual_reject	Rejected	Save reason, invite appeal
Rejected	appeal_filed	Appealed	Escalate
Appealed	appeal_resolved_approve	Approved	
Approved	finalize_actions_completed	Closed	Registry updated
any active	withdraw_claim	Withdrawn	By claimant

5) Implementation tips & technologies

- **State machine library:** use durable state machine libraries that persist state (e.g., Durable Functions, Temporal, state-machine libraries with DB persistence, or custom with DB + events). Temporal or Cadence are excellent for complex long-running workflows.

- **Event bus**: Kafka/RabbitMQ for reliability.

- **Storage**: Postgres (with JSONB) / PostGIS for geometry, plus object store (S3) for docs/images.

- **Audit:** Elasticsearch/ELK for searchable logs; append-only ledger if required.

- **UI:** timeline view for each claim; visual state badges; action buttons driven by allowed transitions for the logged-in role.

- **Testing:** unit test state machines with deterministic event sequences; integration tests for retries & timeouts.

CHAPTER 5

Interaction Modeling

Use-Case Diagram

1) Actors — relevance & responsibilities

1. **External System**

    - **Relevance:** third-party systems that integrate with the FRA platform (satellite imagery providers, national ID APIs, scheme registries).

    - **Responsibilities:** supply satellite imagery, authenticate users via API, push scheme data or other reference data. Enables automation and up-to-date external data.

2. **Citizen / NGO**

    - **Relevance:** primary public stakeholders who view atlas maps, check claim status, and access public dashboards. NGOs may assist claimants.

    - **Responsibilities:** view public information, track claim progress, optionally lodge claims or provide supporting evidence.

3. **Field Officer**

    - **Relevance:** operational user working in the field to verify on-ground data, upload scanned documents, and update claim statuses.

- **Responsibilities:** verify spatial claims (compare satellite/asset maps with field observations), upload scans/photos, mark claim status transitions (e.g., verified, requires evidence).

4. **State / District Official**
    - **Relevance:** decision makers and regional administrators who run analyses and generate official reports.
    - **Responsibilities:** run Decision Support System (DSS) analysis, generate FRA progress reports, download scheme recommendation reports, access dashboards for oversight.
5. **System Administrator**
    - **Relevance:** platform operators responsible for maintenance and configuration.
    - **Responsibilities:** manage users and roles, maintain the database, configure and update AI/ML models, monitor system performance and health.

2) Use Cases — relevance & details

Below each use case label from the diagram with its purpose and who uses it.

1. **Provide Satellite Imagery (External System)**
    - **Purpose:** supply imagery layers used for asset mapping and verification.
    - **Why:** critical for AssetMapper and spatial validation; enables remote evidence and change detection.

2. **Authenticate Users (API) (External System)**
    - **Purpose:** external auth provider (SSO) or centralized identity check.
    - **Why:** secure login, role resolution, auditability.
3. **Supply Scheme Data via API (External System)**
    - **Purpose:** bring in up-to-date welfare scheme metadata/eligibility criteria.
    - **Why:** DSS needs current schemes and rules to produce recommendations.
4. **View FRA Atlas Maps (Citizen / NGO)**
    - **Purpose:** browse public map layers and basic parcel information.
    - **Why:** transparency; enables community monitoring.
5. **Check Claim Status (Citizen / NGO)**
    - **Purpose:** let claimants/NGOs track progress.
    - **Why:** reduces friction and call/visit load on field offices.
6. **Access Public Dashboard Data (Citizen / NGO)**
    - **Purpose:** view aggregated metrics (claims by region, approvals, etc.).
    - **Why:** public accountability and planning insights.

7. **Upload Scanned FRA Documents (Field Officer / Citizen)**
   - **Purpose:** submit scanned documents (IDs, application forms, evidence).
   - **Why:** forms the primary evidence base for claims and OCR pipelines.
8. **Verify Spatial Claims (Field Officer)**
   - **Purpose:** compare claimed boundaries to satellite/asset maps and confirm/mark discrepancies.
   - **Why:** ensures on-ground verification for fairness and correctness.
9. **Update Claim Status (Field Officer)**
   - **Purpose:** move claims through workflow stages (verified, pending, rejected, approved).
   - **Why:** progresses claims to decision makers.
10. **Generate FRA Progress Reports (State/District Official)**
    - **Purpose:** produce periodic reports for program tracking.
    - **Why:** monitoring & reporting to senior authorities.
11. **Run Decision Support Analysis (State/District Official)**
    - **Purpose:** run DSS to determine eligibility, prioritize interventions, and generate recommendations.

  - **Why:** speeds up consistent decisions and policy alignment.

12. **Download Scheme Recommendation Reports (State/District Official)**

    - **Purpose:** get structured outputs recommending welfare schemes for approved claims.

    - **Why:** operationalizes DSS outputs for benefits delivery.

13. **Access Dashboard (State/District Official)** *(included by Generate FRA Progress Reports in diagram)*

    - **Purpose:** central console for regional managers.

    - **Why:** merge real-time stats and DSS outputs.

14. **Manage Users (System Administrator)**

    - **Purpose:** create/modify roles and user access.

    - **Why:** ensures correct RBAC and least privilege.

15. **Maintain Database (System Administrator)**

    - **Purpose:** backups, schema migrations, data integrity tasks.

    - **Why:** critical for availability and compliance.

16. **Configure AI Models (System Administrator)**

   - **Purpose:** deploy/revert ML models used by AssetMapper or OCR pipelines.
   - **Why:** ensures model governance and reproducibility of results.

17. **Monitor System Performance (System Administrator)**

   - **Purpose:** observe system health, performance metrics, and troubleshoot.
   - **Why:** SRE/ops tasks for uptime and SLA adherence.

3) Core Use Case — Sequence Diagram (textual flow)

I'll describe a sequence diagram for the **"Verify Spatial Claims / Upload Scanned Documents / Run DSS Analysis"** workflow that ties many actors and use cases together.

Participants (left→right): FieldOfficer (actor) —> WebUI/API —> DocumentProcessorService —> AssetMapperService —> DSSService —> ExternalSystems (sat imagery / scheme API) —> Database

Sequence steps:

1. **FieldOfficer** uses **WebUI** to open a claim → WebUI queries **Database** for claim and spatial data.

2. FieldOfficer clicks **Upload Scanned FRA Documents** → WebUI uploads files to storage and sends document_uploaded event to **DocumentProcessorService**.

3. **DocumentProcessorService**: preprocesses images, runs OCR, extracts entities (IDs, names), and stores extracted metadata in Database. On success, it emits doc_processing_completed event.

4. WebUI or an orchestration service receives doc_processing_completed and triggers **AssetMapperService** (if spatial analyses required) and attaches document metadata to the claim.

5. AssetMapper requests **ExternalSystems** for latest satellite imagery as needed (tile/cutout) → ExternalSystem returns imagery assets.

6. **AssetMapperService** runs detection models (identify buildings, water bodies, forest), generates asset layers, stores results in Database, emits asset_mapping_completed.

7. Orchestration initiates **DSSService**: it aggregates claim data, document metadata, asset maps, and polls **ExternalSystems** for current scheme data (via Supply Scheme Data API).

8. **DSSService** runs eligibility rules and scoring models → produces **Recommendation** objects and recommendation_generated event; results saved to Database.

9. **State/District Official** is notified (or pulls) and can **Run Decision Support Analysis** or **Download Scheme Recommendation Reports**; WebUI shows report and allows download.

10. If FieldOfficer had flagged inconsistencies during verification, the system marks Pending Evidence and sends notification to the claimant (Citizen/NGO) to Upload Scanned Documents again.

11. All actions (uploads, status changes, analysis runs) are recorded in **SystemLog** for audit.

4) Explanation of the diagram as a whole

- The diagram is a **system-level use case view** that shows who interacts with the FRA platform and what high-level goals they accomplish.

- Actors on the left are real users or systems; ovals are use cases. The diagram focuses on **data ingestion (documents, imagery, scheme data), processing (OCR, asset mapping, DSS), verification (field officer), decision making (officials), and operations (admin)**.

- Relationships shown (<<include>>, <<extend>>) indicate dependencies: e.g., **Generate FRA Progress Reports** may *include* accessing Dashboard data; **Verify Spatial Claims** *extends* or uses **Upload Scanned FRA Documents** when evidence is required.

5) Advanced features to include (design & non-functional)

1. **Event-driven orchestration**: publish domain events (document_uploaded, asset_mapping_completed, recommendation_generated) to a durable bus so microservices decouple and can retry/persist progress.

2. **Asynchronous processing with progress UI**: long jobs (OCR, mapping, DSS) show progress and estimated completion.

3. **Role-based access control (RBAC)** + session MFA for sensitive actions (approve/reject/override).

4. **Geo-provenance & versioning**: store which satellite image (source, date, model version) influenced each decision; essential for appeals.

5. **Explainability + audit trail**: every DSS recommendation should include rule traces, confidence score, and model version. Store these together with SystemLog entries.

6. **SLA & timers**: timeouts for claimant responses (e.g., if additional evidence not provided in X days), with automatic escalation.

7. **Simulation & dry-run mode for DSS**: let officials run “what-if” analyses when they change eligibility rules (no state changes).

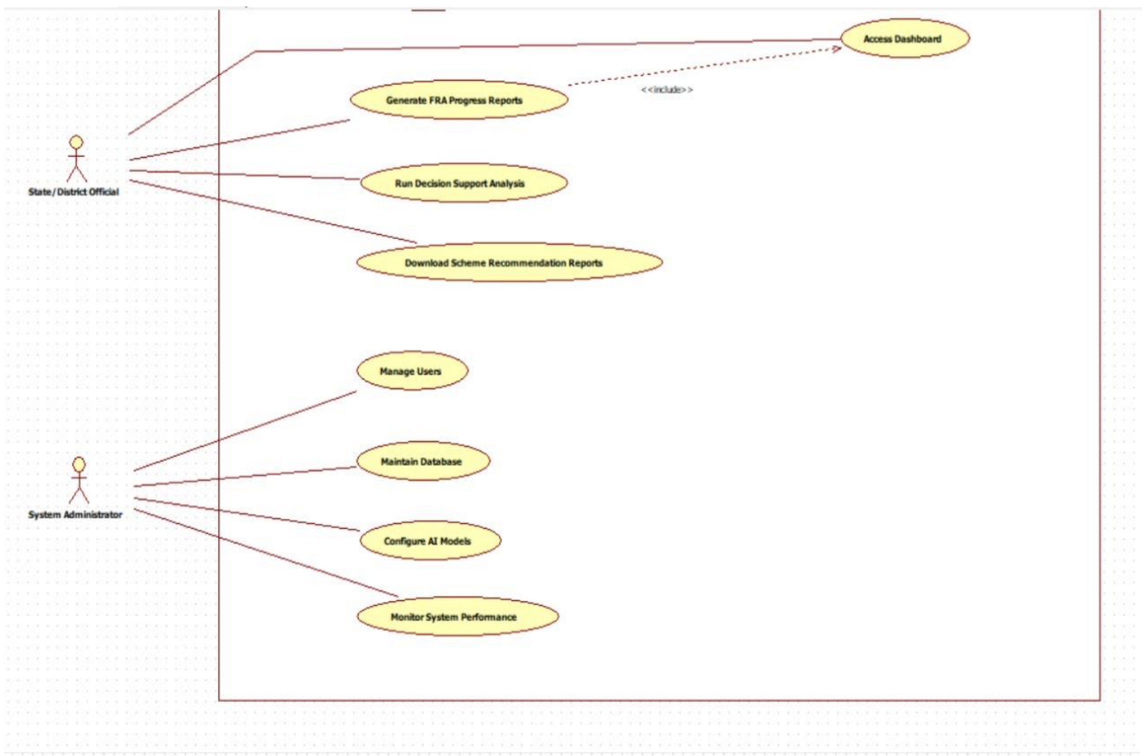
8. **Offline mobile client for Field Officers**: allow capture and queueing of documents and geo-data for later sync; handle conflict resolution with optimistic locking.

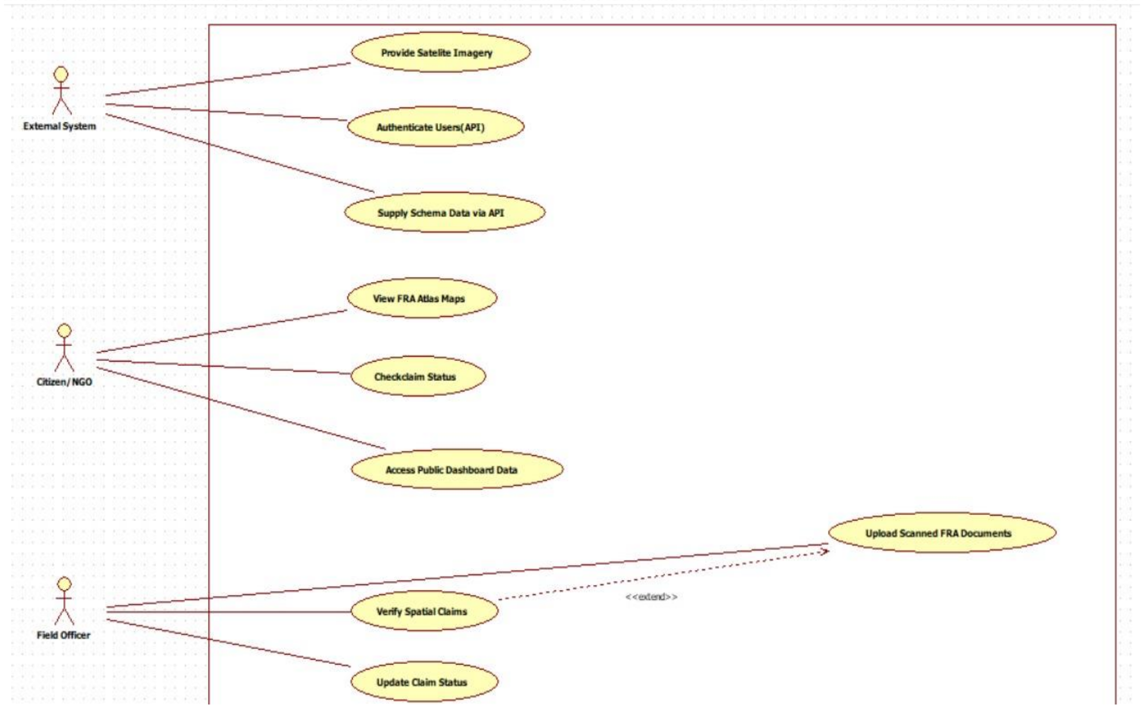
9. **Bulk processing & batch approvals**: admins can approve a set of low-risk claims with per-claim audit.

10. **Secure storage & PII controls**: encrypt documents at rest, mask PII in UI for certain roles, and implement data retention policies.

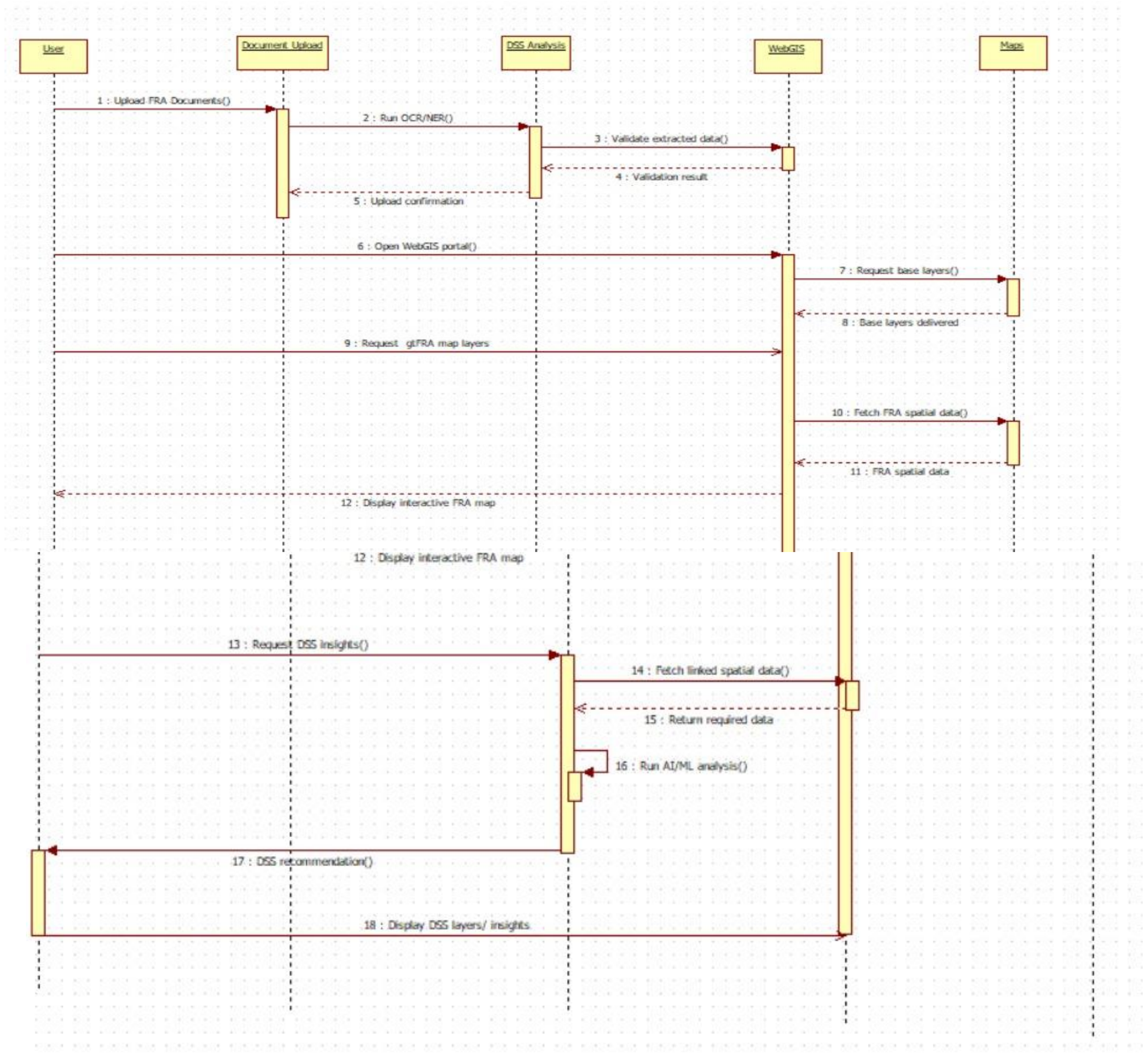
6) Deliverables I can produce next (pick one)

- A **Use Case Description Table** (for each use case: goal, actor, preconditions, basic flow, postconditions, exceptions).
- A **PlantUML** version of this use case diagram (text you can paste into PlantUML).
- A **Sequence Diagram** in PlantUML for the core workflow described above.
- A **CSV** of use cases mapped to actors and priority.


State/District Official
Access Dashboard
Generate FRA Progress Reports
<<include>>
Run Decision Support Analysis
Download Scheme Recommendation Reports
System Administrator
Manage Users
Maintain Database
Configure AI Models
Monitor System Performance


External System
Provide Satelite Imagery
Authenticate Users(API)
Supply Schema Data via API
Citizen/NGO
View FRA Atlas Maps
Checkclaim Status
Access Public Dashboard Data
Field Officer
Upload Scanned FRA Documents
Verify Spatial Claims
<<extend>>
Update Claim Status

Sequence Diagram


User
Document Upload
DSS Analysis
WebGIS
Maps
1 : Upload FRA Documents()
2 : Run OCR/NER()
3 : Validate extracted data()
4 : Validation result
5 : Upload confirmation
6 : Open WebGIS portal()
7 : Request base layers()
8 : Base layers delivered
9 : Request gtFRA map layers
10 : Fetch FRA spatial data()
11 : FRA spatial data
12 : Display interactive FRA map
12 : Display interactive FRA map
13 : Request DSS insights()
14 : Fetch linked spatial data()
15 : Return required data
16 : Run AI/ML analysis()
17 : DSS recommendation()
18 : Display DSS layers/ insights

Actors — relevance & responsibilities

1. **User**
    - Any system user (claimant, field officer, official) who initiates flows: uploads documents, opens the map, requests DSS insights**.**
    - Responsible for starting interactive processes and viewing results.
2. **Document Upload (service / subsystem)**
    - Handles file ingestion, storage, pre-processing and enqueuing of documents for OCR/NER and validation.
    - Responsibility: provide durable storage and reliable event emit for downstream processors.
3. **DSS Analysis (Decision Support Service)**
    - Aggregates data (documents, spatial layers, scheme rules), runs eligibility rules and ML models, and produces recommendations and scores.
    - Responsibility: compute recommendations and return structured insights.
4. **WebGIS (map portal / front-end map server)**
    - Orchestrates user map requests, fetches base and overlay layers, and renders interactive maps in the browser.

  - Responsibility: coordinate spatial requests, apply symbology/filters, and present DSS layers.

5. **Maps (tile server / spatial data store / external map provider)**

  - The backend that stores/serves base tiles, GFRA layers, FRA spatial datasets (PostGIS, tile server, WMS/WMTS).

  - Responsibility: serve spatial assets reliably and quickly.

Use cases (from sequence) — short description & why they matter

- Upload FRA Documents — user uploads scanned documents as evidence. Essential to start automated extraction and validation.

- Run OCR / NER — extracts text and entities (names, IDs, dates) from the uploaded images/PDFs so structured metadata can be attached to claims.

- Validate extracted data — automated validation rules (required fields, formats, duplicates). Prevents bad data from entering analysis.

- Open WebGIS portal — user opens the mapping UI to inspect claims and overlays.

- Request base layers / base layers delivered — WebGIS pulls base map tiles (satellite, streets) so context is available.

- Request GFRA map layers / fetch FRA spatial data — WebGIS requests domain layers (parcels, registered claims) from Maps backend.

- Display interactive FRA map — final rendering step so user can pan/zoom and inspect features.

- Request DSS insights / Fetch linked spatial data — the WebGIS or user calls DSS; DSS fetches spatial and document metadata needed for analysis.

- Run AI/ML analysis — DSS runs scoring or ML models on aggregated inputs.

- DSS recommendation — DSS returns recommendations which are displayed as additional map layers or sidebar insights.

Step-by-step sequence (maps to your numbered messages)

1. **Upload FRA Documents() — User → Document Upload**

    - Action: user uploads scanned docs (IDs, forms, photos). The Document Upload service stores files (object store) and emits an ingestion event.

2. **Run OCR/NER() — Document Upload → DSS Analysis**

    - Document Upload enqueues a job or forwards the file metadata to the OCR/NER pipeline (may be inside DSS Analysis or a separate DocumentProcessor service).

3. **Validate extracted data() — DSS Analysis → WebGIS (validation service)**

    - Extracted text/entities are validated against business rules (format checks, required fields, duplicates). The validation service returns pass/fail.

4. **Validation result ← WebGIS**
    - Result is propagated back (so the portal or user is notified). If pass → continues; if fail → request re-upload or manual review.
5. **Upload confirmation ← Document Upload → User**
    - Upload + preliminary processing acknowledged to the user (UI displays success or next steps).
6. **Open WebGIS portal() — User → WebGIS**
    - The user opens the mapping portal to visualize claims and spatial context.
7. **Request base layers() — WebGIS → Maps**
    - WebGIS requests base map tiles (satellite / street / elevation).
8. **Base layers delivered ← Maps → WebGIS**
    - Tile server responds with map tiles.
9. **Request gFRA map layers — WebGIS → Maps**
    - Request domain-specific GFRA overlays (parcels, registered claims, asset layers).
10. **Fetch FRA spatial data() — WebGIS → Maps**

- WebGIS fetches FRA datasets (geojson, vector tiles, or WMS responses).

11. FRA spatial data ← Maps → WebGIS

- Maps returns the FRA layers (vector tiles/GeoJSON).

12. Display interactive FRA map ← WebGIS → User

- WebGIS renders layers and interactive widgets (popups, legend).

13. Request DSS insights() — User/WebGIS → DSS Analysis

- User or portal requests analytic overlays or eligibility recommendations for features on the current map extent.

14. Fetch linked spatial data() — DSS Analysis → WebGIS/Maps

- DSS pulls spatial features and linked metadata needed to compute recommendations.

15. Return required data ← Maps/WebGIS → DSS Analysis

- Returns the feature geometries, attributes, doc metadata, time-series if needed.

16. Run AI/ML analysis() — DSS Analysis (internal)

- DSS runs rule engine + ML model(s) to score and produce recommendations.

17. DSS recommendation() ← DSS Analysis → User/WebGIS

- Results are sent back; could be GeoJSON layers with confidence, rule traces, or tabular reports.

18. Display DSS layers/insights ← WebGIS → User

- WebGIS overlays DSS outputs (heatmaps, recommended schemes, priority scores) and provides tools for export/download.

Explanation of the diagram as an integrated flow

- The diagram is asynchronous and event-driven in nature: document upload triggers OCR/NER and validation, while map viewing and DSS requests are synchronous UX flows that fetch precomputed or on-demand analysis results.
- The WebGIS acts as the UI orchestrator: it triggers spatial fetches and pushes requests to DSS for analytics.
- DSS depends on both document metadata (from OCR/NER) and spatial layers (from Maps) — it is therefore a *data aggregator and compute* node.
- The user sees combined outputs: validated document metadata + spatial overlays + DSS recommendations.

Advanced production features to include (recommended)

A. Reliability & Scalability

- Asynchronous job queue for OCR/NER (e.g., Kafka + worker pool, Celery/RabbitMQ) — avoid blocking uploads.
- Idempotent event handlers (include event_id and claim_version) so retries are safe.
- Autoscaling tile servers / vector tile generation to handle map load.

B. UX & Observability

- Progress & notifications: show upload/processing progress, send email/SMS/in-app notifications when processing completes or needs user action.
- Map caching & prefetch: cache frequently used tiles and pre-warm tiles for regions with high activity.
- Monitoring & tracing: OpenTelemetry traces across DocumentProcessor → DSS → Maps so you can debug long flows.

C. Data Quality & Governance

- Schema & geometry validation hooks (CRS checks, topology checks). Reject or quarantine bad inputs.
- Provenance & model versioning: for each DSS recommendation store model version, rules triggered, input snapshots, and confidence scores (helps appeals & audit).

D. Security & Access Control

- RBAC + MFA — sensitive operations (approve/reject, admin override) require stronger authentication and specific roles.
- PII controls: encryption at rest for document store, mask PII for lower-privilege roles, data retention policies.

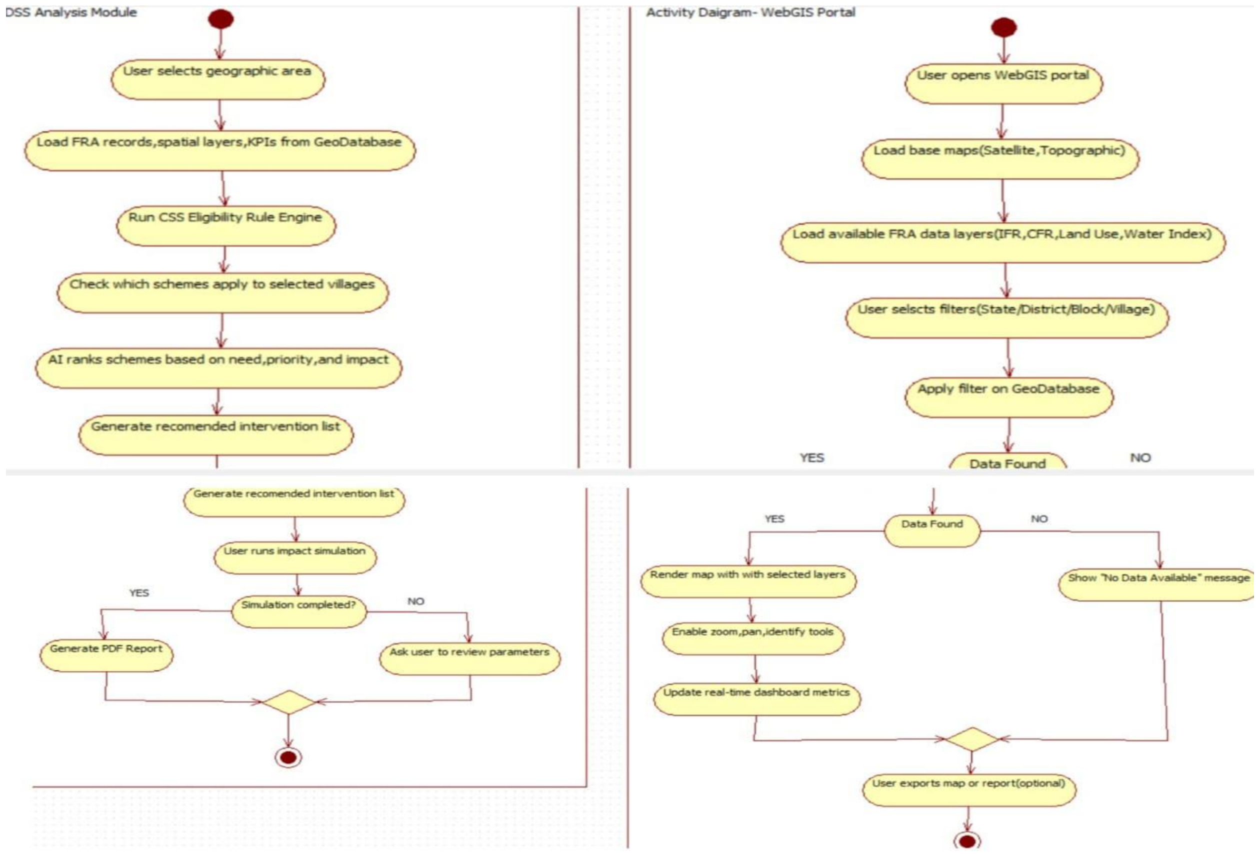
E. Advanced Analytical features

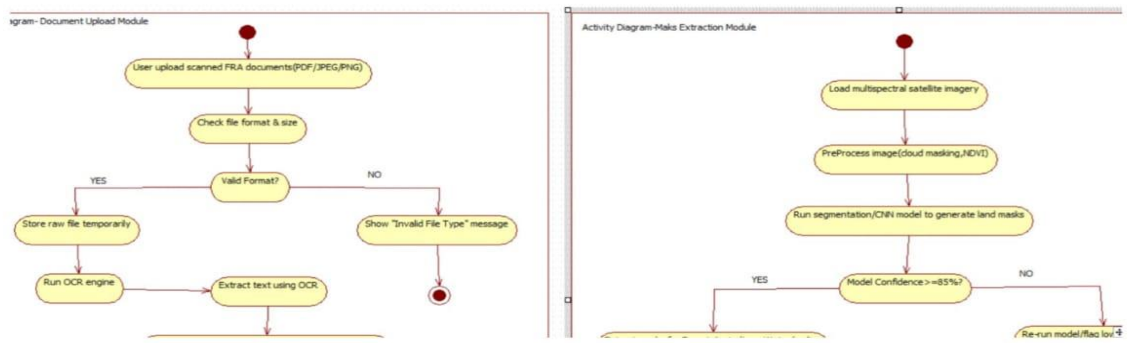
- Hybrid rule-engine + ML: deterministic eligibility rules + ML prioritization (score + explainability).
- Explainability traces: for each recommended action include which rules fired, which features influenced scores, and top contributing evidence items.
- Dry-run / simulation mode for DSS so officials can test rule changes without affecting live state.

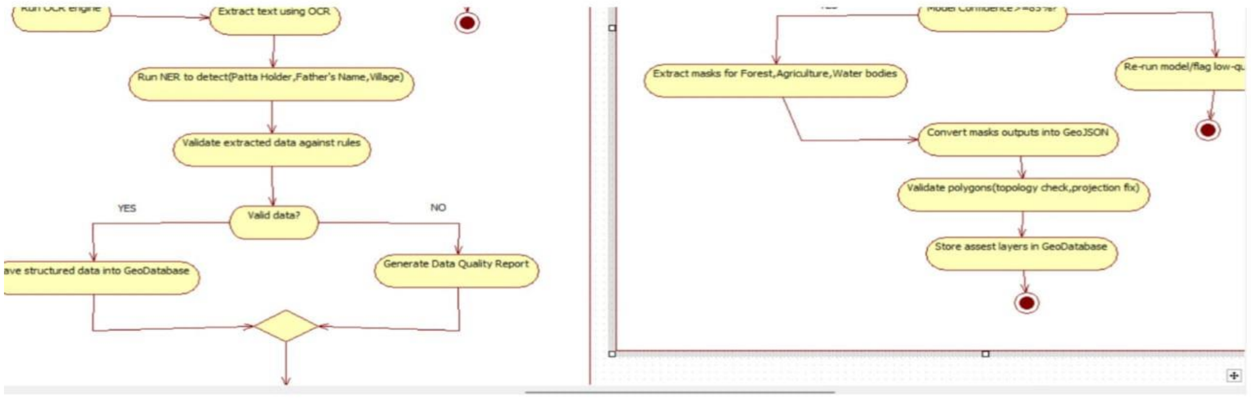
F. Operational Resilience

- SLA timers & escalation: if claimant doesn't respond to request additional docs within X days, escalate or auto-close per policy.
- Compensating transactions / Sagas for multi-step side-effects (issuing title, ledger writes) so failures can be compensated.
- Offline-capable mobile app for field officers to capture geometry/photos and queue uploads; sync on connectivity with conflict resolution via version numbers.

Activity diagram


DSS Analysis Module
User selects geographic area
Load FRA records,spatial layers,KPIs from GeoDatabase
Run CSS Eligibility Rule Engine
Check which schemes apply to selected villages
AI ranks schemes based on need,priority,and impact
Generate recomended intervention list
User runs impact simulation
Simulation completed?
YES
NO
Generate PDF Report
Ask user to review parameters
Activity Daigram- WebGIS Portal
User opens WebGIS portal
Load base maps(Satellite,Topographic)
Load available FRA data layers(IFR,CFR,Land Use,Water Index)
User selscts filters(State/District/Block/Village)
Apply filter on GeoDatabase
Data Found
YES
NO
Render map with with selected layers
Enable zoom,pan,identify tools
Update real-time dashboard metrics
Show "No Data Available" message
User exports map or report(optional)


agram- Document Upload Module
User upload scanned FRA documents(PDF/JPEG/PNG)
Check file format & size
Valid Format?
YES
NO
Store raw file temporarily
Show "Invalid File Type" message
Run OCR engine
Extract text using OCR
Activity Diagram-Maks Extraction Module
Load multispectral satellite imagery
PreProcess image(cloud masking,NDVI)
Run segmentation/CNN model to generate land masks
Model Confidence>=85%?
YES
NO
Re-run model/flag lo


Run OCR engine
Extract text using OCR
Run NER to detect(Patta Holder,Father's Name,Village)
Validate extracted data against rules
Valid data?
YES
NO
ave structured data into GeoDatabase
Generate Data Quality Report
Extract masks for Forest,Agriculture,Water bodies
Re-run model/flag low-qu
Convert masks outputs into GeoJSON
Validate polygons(topology check,projection fix)
Store assest layers in GeoDatabase

1. Activity Diagram – DSS Analysis Module

Swimlanes

- **User** → selects the geographic area and runs simulations.
- **System (DSS Engine)** → loads data, runs rule engine, ranks schemes.
- **AI Module** → prioritizes schemes and calculates impact.

Flow Explanation

1. **User selects geographic area**
   - Kickstarts the workflow.
2. **System loads FRA records, spatial layers, KPIs from GeoDatabase**
   - Automatic data retrieval.
3. **CSS Eligibility Rule Engine runs**
   - Matches villages with eligible schemes.

4. **Check which schemes apply to selected villages**
    - Filtering candidate schemes.
5. **AI ranks schemes based on need, priority, impact**
    - Uses ML/AI prioritization models.
6. **Generate recommended intervention list**
    - Output of analysis.

2. Activity Diagram – WebGIS Portal Module

Swimlanes

- **User** → opens portal, applies filters, exports maps.
- **GIS Engine** → loads maps, fetches layers, renders visuals.
- **GeoDatabase** → stores spatial information.

Flow Explanation

1. User opens the **WebGIS Portal**.
2. System loads base maps (satellite/topographic).
3. FRA layers are loaded (IFR, CFR, land use, water indices).
4. User applies filters (District/Block/Village).
5. System queries GeoDatabase.

3. Activity Diagram – Document Upload Module (OCR + NER)

Swimlanes

- **User** → uploads document.
- **File Validator** → checks format/size.
- **OCR Engine** → extracts text.
- **NER Module** → extracts fields (Name, Village).
- **GeoDatabase** → stores structured data.

4. Activity Diagram – ML/AI Extraction Module (Satellite Imagery)

Swimlanes

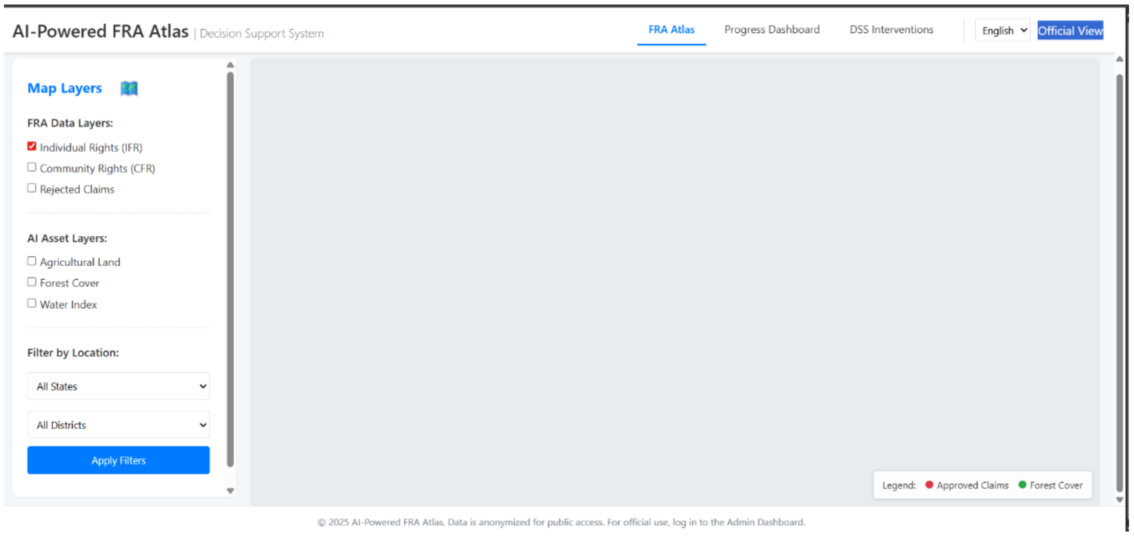
- **AI/ML Engine** → preprocesses imagery, runs segmentation.
- **Validation Module** → checks confidence value.
- **GIS Layer Processor** → converts outputs and stores assets.

Flow Explanation

1. Load multispectral satellite imagery.
2. Preprocess using cloud masking/NDVI.
3. Run segmentation/CNN model → land masks.

CHAPTER 6

UI Design with Screenshots



AI-Powered FRA Atlas | Decision Support System

FRA Atlas · Progress Dashboard · DSS Interventions · English · Official View

Map Layers

FRA Data Layers:

- [✓] Individual Rights (IFR)
- [ ] Community Rights (CFR)
- [ ] Rejected Claims

AI Asset Layers:

- [ ] Agricultural Land
- [ ] Forest Cover
- [ ] Water Index

Filter by Location:

All States

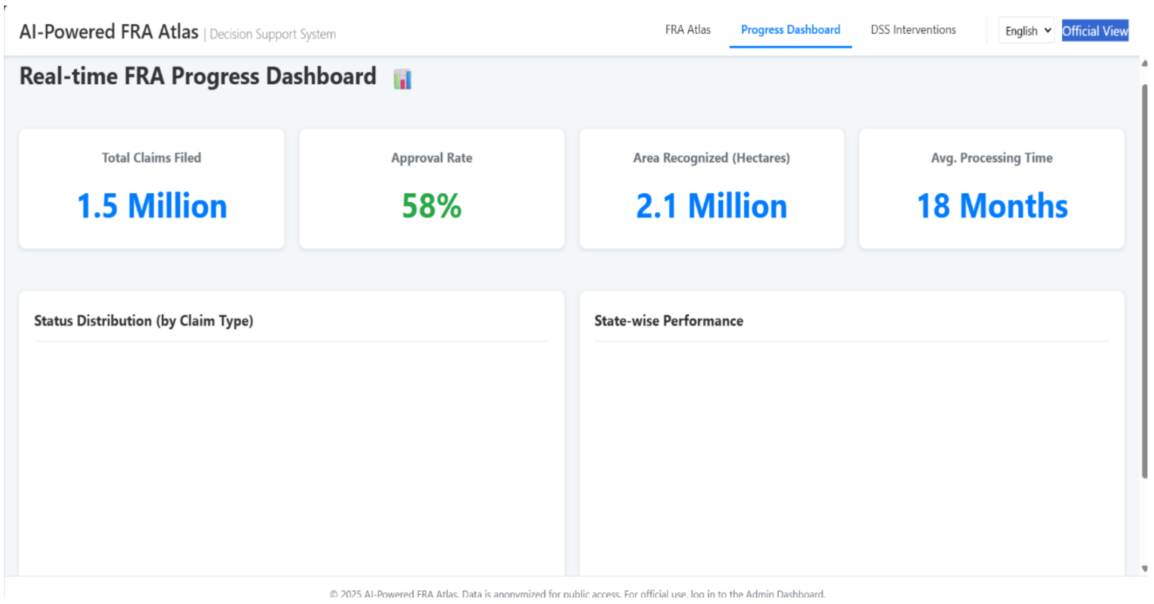
All Districts

Apply Filters

Legend: ● Approved Claims ● Forest Cover

© 2025 AI-Powered FRA Atlas. Data is anonymized for public access. For official use, log in to the Admin Dashboard.

Fig: Home Page



AI-Powered FRA Atlas | Decision Support System

FRA Atlas · Progress Dashboard · DSS Interventions · English · Official View

Real-time FRA Progress Dashboard

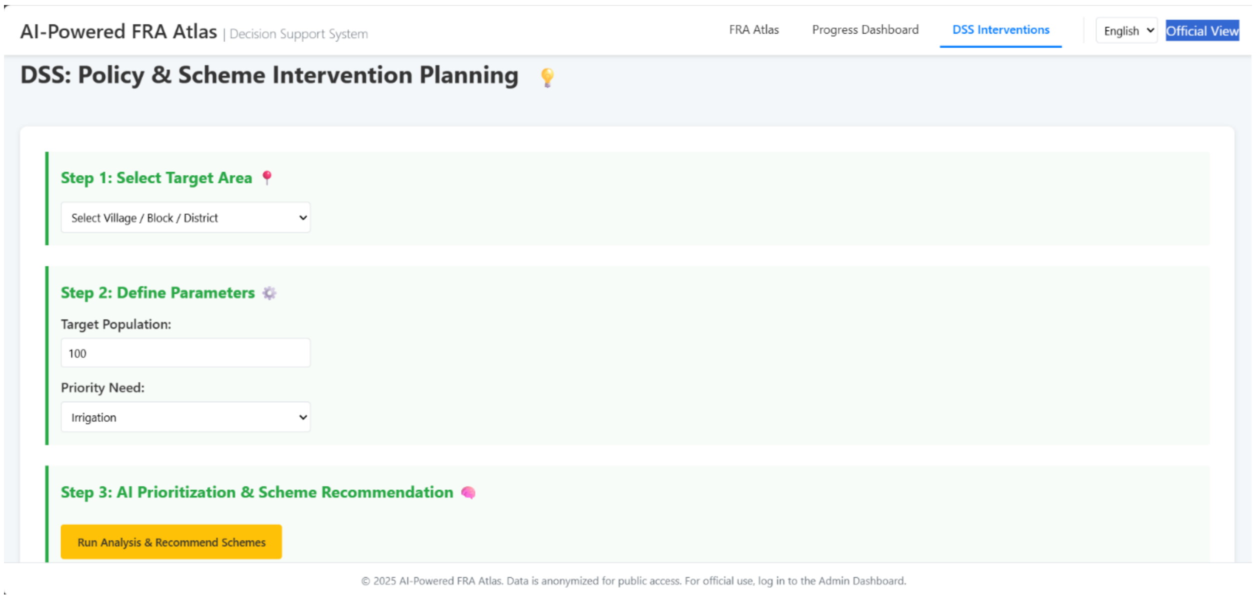
Total Claims Filed	Approval Rate	Area Recognized (Hectares)	Avg. Processing Time
1.5 Million	58%	2.1 Million	18 Months

Status Distribution (by Claim Type)

State-wise Performance

© 2025 AI-Powered FRA Atlas. Data is anonymized for public access. For official use, log in to the Admin Dashboard.

Fig: Progress dashboard



AI-Powered FRA Atlas | Decision Support System

FRA Atlas | Progress Dashboard | DSS Interventions | English | Official View

DSS: Policy & Scheme Intervention Planning 💡

Step 1: Select Target Area 📍

Select Village / Block / District

Step 2: Define Parameters ⚙

Target Population: 100

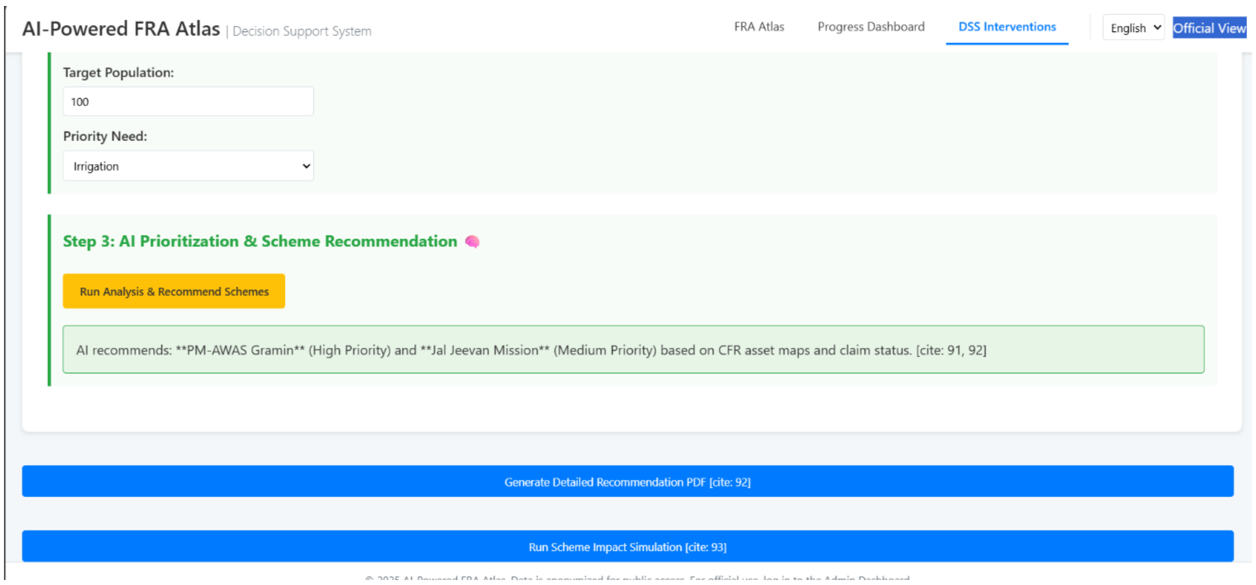
Priority Need: Irrigation

Step 3: AI Prioritization & Scheme Recommendation 🧠

Run Analysis & Recommend Schemes

© 2025 AI-Powered FRA Atlas. Data is anonymized for public access. For official use, log in to the Admin Dashboard.

Fig: DSS Interventions



AI-Powered FRA Atlas | Decision Support System

FRA Atlas | Progress Dashboard | DSS Interventions | English | Official View

Target Population: 100

Priority Need: Irrigation

Step 3: AI Prioritization & Scheme Recommendation 🧠

Run Analysis & Recommend Schemes

AI recommends: **PM-AWAS Gramin** (High Priority) and **Jal Jeevan Mission** (Medium Priority) based on CFR asset maps and claim status. [cite: 91, 92]

Generate Detailed Recommendation PDF [cite: 92]

Run Scheme Impact Simulation [cite: 93]

Fig: Generated Recommendations