Product Requirements & Specification Document

Project Name

AgroTrack - Smart Agriculture Resource Management

Overview

AgroTrack is a full-stack application designed for efficient management of agricultural resources, crop cycles, and equipment. The platform supports complex entity relationships, advanced validation, role-based access, asynchronous report generation, and real-time monitoring via a React frontend. The solution is containerized with Docker and built using Maven.

---

1. Objectives

- Centralize management of agricultural resources, crop cycles, and equipment.
- Enable real-time monitoring and reporting.
- Ensure secure, role-based access for different user types.
- Support scalable deployment and maintainability.

---

2. Core Features

Feature	Description
Resource Management	CRUD for fields, crops, equipment, and consumables.
Crop Cycle Tracking	Plan, monitor, and record crop cycles and associated activities.
Equipment Scheduling	Assign, track, and maintain equipment usage and status.
Role-Based Access Control	Admin, Manager, Worker roles with granular permissions.
Real-Time Monitoring	Live dashboard for resource status and alerts (React frontend).
Async Report Generation	Generate and download reports without blocking UI.
Advanced Validation	Enforce business rules and data integrity at API and DB levels.

---

3. User Roles & Permissions

Role	Permissions
Admin	Full access: manage users, resources, cycles, equipment, and system settings.
Manager	Manage resources, crop cycles, equipment; view/generate reports; assign tasks.
Worker	View assigned tasks, update task status, limited resource/equipment view.

---

4. System Architecture

- **Backend:** Java, Spring Boot, Spring Data JPA, Spring Security
- **Frontend:** React (with real-time updates via WebSockets or polling)
- **Database:** PostgreSQL
- **Deployment:** Docker containers, orchestrated via Docker Compose
- **Build:** Maven

---

5. Entity Model (Simplified)

Entity	Key Fields & Relationships
User	id, name, email, role
Field	id, name, location, size, crops (1:N)
Crop	id, type, variety, field_id, cycle_id
CropCycle	id, crop_id, start_date, end_date, status, activities (1:N)
Equipment	id, type, status, assigned_to (User), maintenance_logs (1:N)
Activity	id, cycle_id, type, scheduled_date, status, assigned_to (User)
Report	id, type, parameters, status, download_url

---

6. Key Functional Requirements

6.1 Resource Management

- CRUD operations for all core entities.
- Validation: e.g., equipment cannot be double-booked; crop cycles must not overlap on the same field.

6.2 Crop Cycle & Activity Tracking

- Plan and monitor crop cycles.
- Assign and track activities (e.g., planting, irrigation, harvesting).

6.3 Equipment Management

- Schedule equipment usage.
- Track maintenance and status.

6.4 Role-Based Access

- Enforce permissions at API and UI levels.
- Secure endpoints using Spring Security.

6.5 Real-Time Monitoring

- Live dashboard: field status, equipment availability, alerts.
- Push updates via WebSockets or efficient polling.

6.6 Async Report Generation

- Users request reports; backend processes asynchronously.
- Notify user and provide download link when ready.

---

7. Non-Functional Requirements

Category	Specification
Performance	<1s response for standard queries; async for heavy reports
Security	JWT-based authentication; encrypted credentials; audit logging
Scalability	Containerized deployment; stateless backend
Maintainability	Modular codebase; clear API documentation (OpenAPI/Swagger)
Usability	Responsive, intuitive React UI; accessible on desktop and tablet
Reliability	Graceful error handling; retry logic for async tasks

---

8. API & Integration

- RESTful API endpoints for all entities.
- OpenAPI/Swagger documentation.
- WebSocket endpoint for real-time updates.

Sample Endpoint:

```
GET /api/fields/{id}/status
Authorization: Bearer <token>
Response: { "fieldId": 1, "status": "Active", "currentCrop": "Wheat" }
```

---

9. Validation & Business Rules

- Unique constraints (e.g., equipment assignment).
- Date validations (e.g., crop cycles).
- Role-based field visibility and editability.

---

10. Deployment & DevOps

Aspect	Specification
Containerization	Docker, Docker Compose
Build	Maven
Database	PostgreSQL (containerized)
CI/CD	Optional: GitHub Actions or similar

---

11. Milestones

Milestone	Description	Target Date
MVP Backend API	Core entities, validation, RBAC	T+4 weeks
Frontend Dashboard	Real-time monitoring, CRUD UI	T+6 weeks
Async Reporting	Background report generation, download	T+8 weeks
Dockerized Deployment	Full stack in containers	T+9 weeks

---

12. Acceptance Criteria

- All core features implemented and tested.
- Role-based access enforced.
- Real-time dashboard operational.
- Reports generated asynchronously.
- Deployed and running via Docker Compose.

---

13. Out of Scope

- Mobile app
- IoT device integration
- Advanced analytics/ML

End of Document