Biofuel Web Application: Project Execution Blueprint

1. Project Plan & Milestones

1.1 High-Level Roadmap

Phase	Timeline	Deliverables
Phase 1: Discovery & Planning	Week 1-2	Finalized SRS, Technology Stack, Initial Designs
Phase 2: MVP Development	Week 3-6	Core Modules (Authentication, Plants, Vehicles, Vendors, Supervisors)
Phase 3: Full Feature Development	Week 7-12	Offline Support, Receipts, Weight Management, Multi-plant Support, Admin Panels
Phase 4: QA & UAT	Week 13-15	Full QA Cycle, UAT by Stakeholders
Phase 5: Deployment & Monitoring	Week 16	Live Deployment, Alerting, Analytics, Documentation

1.2 Team Roles & Responsibilities

Role	Responsibilities
Product Manager	Stakeholder coordination, feature prioritization, roadmap management
Frontend Developer	Build responsive UI, implement operator/admin views, offline capabilities
Backend Developer	Develop APIs, database schema, auth logic, integrate services
DevOps Engineer	CI/CD, infrastructure setup, monitoring, backup strategies
QA Engineer	Write and execute test cases, bug tracking, regression testing
UI/UX Designer	Wireframes, visual design, user journey mapping

1.3 Risk Assessment & Mitigation

Risk	Probability	Impact	Mitigation
Data loss during offline sync	Medium	High	IndexedDB fallback, sync queue, version checks
Multi-plant data collision	Low	Medium	Multi-tenant DB model, scoped queries
Untrained operators misusing system	High	Medium	Guided flows, training material, role restrictions

1.4 Assumptions, Dependencies & Constraints

- Assumes network intermittency is common in plant locations.
- Assumes each plant has a designated supervisor.
- Dependencies: MongoDB, Redis, Node.js, React, secure hosting (e.g., AWS, Azure).
- Constraints: Limited screen size for operator terminals, storage limits in local cache.

---

2. Design and Technical Documentation

2.1 System Architecture Overview

```
[ Operator Web App ]
        |
        v
[ React (PWA) with Offline Support via IndexedDB ]
        |
        v
[ Backend API Gateway (Node.js + Express) ]
        |
        +--> [ Auth Service (JWT) ]
        +--> [ Business Logic Service ]
        +--> [ Sync Engine for Offline Data ]
        +--> [ MongoDB with Mongoose ORM ]
        +--> [ Redis for Cache and Queues ]

[ Admin Dashboard, Supervisor View, Vendor Portal ]
        |
        v
[ Role-based Access Control Layer ]
```

2.2 Technology Stack

- Frontend: React + TypeScript (PWA)
- Backend: Node.js + Express
- Database: MongoDB
- Caching: Redis
- Offline Storage: IndexedDB
- DevOps: GitHub Actions, Docker, NGINX, AWS (EC2, RDS, S3)
- Security: JWT + Role-based access control + HTTPS + Data encryption at rest

2.3 Scalability & Robustness

- Horizontal scaling via load balancers (AWS ALB)
- Caching of vendor/vehicle data for fast access
- Retry queues for offline-first write-syncs
- Partitioning by plant for isolated workloads

2.4 Security Practices

- All API calls behind HTTPS
- Data encrypted at rest using MongoDB encryption
- Rate limiting & throttling for API abuse
- Auditing logs for admin/supervisor activities
- Field-level access control on sensitive data

2.5 UI/UX Design Approach

- Role-specific dashboards (minimalism for operators, detailed views for admin)
- Use of cards/lists for key data views
- Accessible forms with validations
- Status indicators for sync/offline states

2.6 CI/CD Plan

- GitHub Actions for automated tests, linting, builds
- Docker containers for staging and production
- Blue-green deployment strategy
- Rollback automation on failed health checks

2.7 Monitoring & Alerting

- Prometheus + Grafana for metrics
- ELK Stack (Elastic + Logstash + Kibana) for logs
- UptimeRobot for endpoint uptime
- PagerDuty for incident alerts

---

3. Detailed User Stories and Use Cases

3.1 Operator

- Story: "As an operator, I want to record incoming/outgoing material, so that I can track vendor loads."
- Acceptance:
- Can enter weight, vehicle, vendor.
- View only past 24-hour records.

- Can operate offline and auto-sync later.

- Story: "As an operator, I want to search vehicles by ID or vendor, so I can record data quickly."

- Acceptance:
- Autocomplete for vehicle/vendor fields.

3.2 Supervisor

- Story: "As a supervisor, I want to view data across my plant, so I can ensure accuracy."
- Acceptance:
- Plant-wide visibility.

• Can add operators or edit vendor/vehicle details.

3.3 Admin

• Story: "As an admin, I want to manage all plants, supervisors, and vendors, so that I have centralized control."
• Acceptance:
• CRUD operations on all entities.
• Can generate analytics/report downloads.

3.4 Use Case Table

User	Action	Outcome
Operator	Log incoming vehicle	Entry stored, receipt issued
Supervisor	View all day logs	Accurate summary & report
Admin	Add a new vendor and assign vehicles	Vendor can operate immediately

---

4. Test Plan and Sample Test Cases

4.1 Test Strategy

• Unit Tests: Functions, validation, auth logic (Jest)
• Integration Tests: API with DB (Supertest)
• End-to-End: Cypress for UI flows
• Performance: Apache JMeter for stress
• Security: OWASP ZAP automated scan
• UAT: Manual tests with plant operators
• Regression: Post-release sanity scripts

4.2 Test Case Table

ID	Objective	Steps	Expected	Priority	Type
TC001	Log vehicle weight	Login > Enter details > Submit	Receipt generated, data saved	High	Positive
TC002	Offline entry queue	Kill internet > Log entry > Reconnect	Data syncs successfully	High	Edge
TC003	View as supervisor	Login > Select plant	Data scoped to plant	Medium	Positive
TC004	Multi-vendor vehicle	Add 2 vendors to vehicle > Assign load	History maps correctly	High	Functional
TC005	Weight variance check	Enter -70kg delta	Show warning/ error	Medium	Negative

---

5. Additional Notes

- Vehicles can be unregistered or have temp IDs.
- There may be in-house tractors rented out — system must track usage logs.
- Supervisor approval workflows may be added in future.
- IndexedDB sync logic should include timestamps, conflict resolution.
- Use Notion for documentation, Figma for wireframes, Postman for API validation.