# AgriTech Project - Complete Django Task Master Plan

## Phase 1: Project Setup & Research (Weeks 1-6)

### Week 1: Environment Setup & Tool Installation

#### Day 1-2: Development Environment

- [ ] Install Python 3.9+ (use Anaconda for easier package management)
- [ ] Install Visual Studio Code or PyCharm IDE
- [ ] Set up Git and create GitHub repository
- [ ] Install Node.js and npm for frontend development
- [ ] Create virtual environment: `python -m venv agritech_env`
- [ ] Activate environment: `source agritech_env/bin/activate` (Linux/Mac) or `agritech_env\Scripts\activate` (Windows)

#### Day 3-4: Django Backend Setup

- [ ] Install Django and essentials: `pip install django djangorestframework django-cors-headers`
- [ ] Install ML libraries: `pip install scikit-learn pandas numpy matplotlib seaborn plotly`
- [ ] Install data processing: `pip install requests beautifulsoup4 selenium openpyxl`
- [ ] Install database: `pip install psycopg2-binary`
- [ ] Install additional Django packages: `pip install django-extensions django-debug-toolbar`
- [ ] Create Django project: `django-admin startproject agritech_backend`
- [ ] Create requirements.txt file: `pip freeze > requirements.txt`

#### Day 5-6: Frontend Setup (React for better real-time features)

- [ ] Create React app: `npx create-react-app agritech-frontend`
- [ ] Install UI libraries: `npm install @mui/material @emotion/react @emotion/styled @mui/icons-material`
- [ ] Install charts: `npm install chart.js react-chartjs-2 plotly.js react-plotly.js`
- [ ] Install maps: `npm install leaflet react-leaflet`
- [ ] Install HTTP client: `npm install axios`
- [ ] Install real-time: `npm install socket.io-client`

#### Day 7: Project Structure & Documentation Setup

- [ ] Create project folder structure: `docs/`, `data/`, `models/`, `notebooks/`
- [ ] Set up Django project structure with apps
- [ ] Create project README.md file
- [ ] Set up folder structure for documentation
- [ ] Create initial project architecture diagram (use draw.io)
- [ ] Write project description and objectives

## Week 2: Research Paper Collection & Analysis

### Day 1-2: Literature Search

- [ ] Search Google Scholar for "crop prediction machine learning" (collect 15-20 papers)
- [ ] Search for "agricultural weather forecasting" papers (collect 10-15 papers)
- [ ] Find "soil health monitoring IoT" research (collect 8-10 papers)
- [ ] Look for "pest disease detection deep learning" studies (collect 12-15 papers)
- [ ] Download papers from ResearchGate, IEEE Xplore, ScienceDirect

### Day 3-4: Paper Organization & Analysis

- [ ] Create folders: `docs/research/crop_prediction`, `weather`, `soil`, `pest_disease`
- [ ] Read abstracts and create summary table in Excel/Google Sheets
- [ ] Rate papers by relevance (1-5 scale)
- [ ] Extract key datasets mentioned in papers
- [ ] Note down algorithms and models used (Random Forest, XGBoost, CNN, etc.)

### Day 5-6: Key Insights Documentation

- [ ] Create document: "Research Insights Summary.md"
- [ ] List best performing ML algorithms for each feature
- [ ] Document data requirements for each model
- [ ] Identify gaps in current research that you can fill
- [ ] Note innovative approaches that can be implemented

### Day 7: Competitive Analysis

- [ ] Research existing AgriTech platforms (FarmLogs, Granular, Climate FieldView, CropIn)
- [ ] List their features in comparison table
- [ ] Identify unique features you can add
- [ ] Document their limitations and your solutions
- [ ] Plan your competitive advantages

## Week 3: Data Source Identification & API Setup

### Day 1-2: Weather Data Sources

- [ ] Sign up for OpenWeatherMap API (free tier: 1000 calls/day)
- [ ] Get NASA POWER API access (free, no key needed)
- [ ] Register for WeatherAPI.com (free tier: 1M calls/month)
- [ ] Test basic API calls and understand response formats
- [ ] Create Python scripts to fetch weather data
- [ ] Document API rate limits and costs

### Day 3-4: Agricultural Data Sources

- [ ] Explore FAO Statistics database (FAOSTAT) - download CSV files
- [ ] Access India's Agriculture Statistics at a Glance reports (PDF downloads)
- [ ] Download historical crop yield data by state from government portals
- [ ] Get soil data from SoilGrids (ISRIC) - global soil database
- [ ] Access satellite data from Google Earth Engine (requires signup)
- [ ] Create data source inventory document with update frequencies

#### Day 5-6: Market & Economic Data

- [ ] Find commodity price APIs (Alpha Vantage, Quandl)
- [ ] Access Indian market prices from agmarknet.gov.in
- [ ] Collect fertilizer price data from government sources
- [ ] Get seed price information from agricultural departments
- [ ] Document update frequencies and data formats

#### Day 7: Data Quality Assessment

- [ ] Check data completeness (missing values analysis using pandas)
- [ ] Verify data accuracy by cross-referencing multiple sources
- [ ] Identify data cleaning requirements
- [ ] Create data quality report with visualizations
- [ ] Plan data preprocessing steps and create cleaning scripts

### Week 4: Django Project Architecture & Database Setup

#### Day 1-2: Django Project Structure

- [ ] Create Django apps: `python manage.py startapp crops`, `weather`, `soil`, `predictions`, `users`, `dashboard`
- [ ] Set up Django settings for development and production
- [ ] Configure PostgreSQL database connection
- [ ] Set up Django REST Framework in settings
- [ ] Configure CORS settings for React frontend

#### Day 3-4: Database Models Creation

- [ ] Design database schema (tables for crops, weather, soil, users, predictions)
- [ ] Create Entity Relationship Diagrams (use dbdiagram.io)
- [ ] Write Django model classes for each entity:
  - User profiles (extend Django User model)
  - Crop information (name, season, water requirements, etc.)
  - Weather data (temperature, rainfall, humidity, etc.)
  - Soil data (pH, NPK levels, organic content, etc.)

- Prediction results (crop recommendations, yield predictions, etc.)

- [ ] Define model relationships (ForeignKey, ManyToMany)
- [ ] Create model string representations and admin configurations

**Day 5-6: Database Migration & Admin Setup**

- [ ] Generate initial migrations: `python manage.py makemigrations`
- [ ] Apply migrations: `python manage.py migrate`
- [ ] Create superuser: `python manage.py createsuperuser`
- [ ] Set up Django admin interface for all models
- [ ] Customize admin interface with list displays and filters
- [ ] Test CRUD operations through admin interface

**Day 7: API Endpoints Planning**

- [ ] Plan RESTful API structure for all features
- [ ] Create API documentation template
- [ ] Set up Django REST Framework serializers for all models
- [ ] Create basic ViewSets for CRUD operations
- [ ] Test basic API endpoints using Django's test client

## Week 5: Data Collection & Initial Import

**Day 1-3: Weather Data Collection**

- [ ] Write Python scripts to collect historical weather data (last 10 years)
- [ ] Get data for major agricultural regions in India (Punjab, Haryana, UP, etc.)
- [ ] Clean weather data (handle missing values, outliers using pandas)
- [ ] Create weather data aggregation functions (daily, weekly, monthly averages)
- [ ] Create Django management command to import weather data
- [ ] Store processed weather data in PostgreSQL database

**Day 4-6: Crop & Agricultural Data Processing**

- [ ] Download crop production data from multiple government sources
- [ ] Clean and standardize crop names and classifications
- [ ] Process soil data (pH, NPK, organic content) from various sources
- [ ] Create location-based soil profiles for different regions
- [ ] Merge different datasets using common identifiers (district, state)
- [ ] Create Django management commands for data import

**Day 7: Data Validation & Quality Check**

- [ ] Run comprehensive data quality checks on all collected data
- [ ] Create data profiling reports using pandas and matplotlib

- [ ] Fix data inconsistencies and errors
- [ ] Generate summary statistics and visualizations
- [ ] Document data preprocessing steps and create data dictionary

### Week 6: Initial Machine Learning Models

#### Day 1-2: Crop Recommendation Model

- [ ] Prepare training data (soil parameters + weather + historical crop success rates)
- [ ] Split data into train/validation/test sets (70/15/15)
- [ ] Try different algorithms: Random Forest, XGBoost, SVM, Logistic Regression
- [ ] Implement cross-validation for model selection
- [ ] Compare model performances using accuracy, precision, recall, F1-score

#### Day 3-4: Model Optimization & Feature Engineering

- [ ] Perform hyperparameter tuning using GridSearchCV
- [ ] Feature selection and importance analysis
- [ ] Handle class imbalance if present (SMOTE, class weights)
- [ ] Create new features from existing data (weather aggregations, soil ratios)
- [ ] Save best models using joblib or pickle

#### Day 5-6: Weather Impact Analysis Model

- [ ] Build weather-crop compatibility scoring system
- [ ] Create weather risk assessment models
- [ ] Implement seasonal pattern analysis using time series
- [ ] Test model predictions on holdout data
- [ ] Create model performance reports with visualizations

#### Day 7: Model Integration with Django

- [ ] Create Django services to load and use ML models
- [ ] Create API endpoints for model predictions
- [ ] Test API responses with sample data
- [ ] Implement model caching for better performance
- [ ] Document model inputs, outputs, and usage

## Phase 2: Core Development (Weeks 7-14)

### Week 7: Advanced ML Model Development

#### Day 1-2: Yield Prediction Model

- [ ] Collect and prepare historical yield data for major crops
- [ ] Create comprehensive feature engineering pipeline:

- Weather aggregations (growing season averages, extremes)
- Soil health indicators
- Historical yield trends
- Market price influences

- [ ] Implement time series forecasting models (ARIMA, Prophet, LSTM)
- [ ] Try ensemble methods (Random Forest + XGBoost + Linear Regression)
- [ ] Validate models using time-based splits (not random)

**Day 3-4: Soil Health Analysis System**

- [ ] Build soil fertility classification model (High/Medium/Low fertility)
- [ ] Create nutrient deficiency prediction system (NPK deficiencies)
- [ ] Implement soil pH optimization recommendations
- [ ] Add soil erosion risk assessment using terrain and weather data
- [ ] Create soil improvement recommendation engine
- [ ] Test models with real soil test data from agricultural labs

**Day 5-6: Pest & Disease Prediction System**

- [ ] Collect pest occurrence data correlated with weather conditions
- [ ] Build disease outbreak probability models using weather patterns
- [ ] Create pest lifecycle prediction models based on temperature and humidity
- [ ] Implement basic image-based disease detection using CNN
- [ ] Create integrated pest management (IPM) recommendation system
- [ ] Validate predictions against agricultural extension data

**Day 7: Model Performance Optimization**

- [ ] Optimize all models for speed and memory usage
- [ ] Implement model caching and batch predictions
- [ ] Create model monitoring and performance logging
- [ ] Set up automated model evaluation and alerts
- [ ] Document all model architectures and parameters
- [ ] Create model comparison dashboard

## Week 8: Django REST API Development

**Day 1-2: Core API Endpoints**

- [ ] Create comprehensive crop recommendation API with filtering
- [ ] Build weather data retrieval endpoints with location-based queries
- [ ] Implement soil analysis APIs with visualization data
- [ ] Add yield prediction endpoints with confidence intervals
- [ ] Create user authentication and profile management APIs

### Day 3-4: Advanced Features APIs

- [ ] Build market price integration and forecasting APIs
- [ ] Create irrigation scheduling endpoints based on soil moisture and weather
- [ ] Add pest alert system APIs with severity levels
- [ ] Implement recommendation history and tracking
- [ ] Create farm management APIs (field mapping, crop rotation planning)

### Day 5-6: Data Pipeline & Management APIs

- [ ] Build automated data update endpoints with scheduling
- [ ] Create data validation and quality check APIs
- [ ] Add bulk data import/export functionality
- [ ] Implement data backup and restore APIs
- [ ] Add system health monitoring and status endpoints

### Day 7: API Testing & Documentation

- [ ] Write comprehensive API tests using Django's test framework
- [ ] Create detailed API documentation using Django REST Framework's built-in docs
- [ ] Test API rate limiting and security measures
- [ ] Implement proper error handling and meaningful error messages
- [ ] Create API usage examples and tutorials

## Week 9: Real-time Data Integration

### Day 1-2: Live Weather Data System

- [ ] Set up automatic weather data updates every 15 minutes using Celery
- [ ] Create weather alert system for extreme conditions (heat waves, storms, etc.)
- [ ] Implement weather data caching using Redis for better performance
- [ ] Add weather forecast integration with accuracy tracking
- [ ] Test real-time weather API reliability and failover systems

### Day 3-4: Market Data Integration

- [ ] Connect to live commodity price feeds (government mandi prices)
- [ ] Create price change alert system with percentage thresholds
- [ ] Implement market trend analysis and pattern recognition
- [ ] Add price prediction models using time series forecasting
- [ ] Test market data accuracy and timeliness

### Day 5-6: IoT Sensor Data Preparation

- [ ] Design IoT sensor data ingestion system architecture

- [ ] Create mock sensor data generators for testing
- [ ] Implement sensor data validation and outlier detection
- [ ] Add sensor status monitoring and offline handling
- [ ] Plan for future integration with actual IoT devices

#### Day 7: WebSocket Implementation with Django Channels

- [ ] Install and configure Django Channels for WebSocket support
- [ ] Set up WebSocket consumers for real-time data broadcasting
- [ ] Create separate channels for different data types (weather, prices, alerts)
- [ ] Test real-time data broadcasting to multiple clients
- [ ] Implement connection management and error handling

### Week 10: React Frontend Foundation

#### Day 1-2: React Application Setup

- [ ] Create main application layout with responsive navigation
- [ ] Set up routing using React Router for different pages
- [ ] Create authentication components (login, register, password reset)
- [ ] Build responsive navigation menu with user account management
- [ ] Add global loading states and error handling components

#### Day 3-4: Dashboard Layout & Components

- [ ] Create main dashboard layout with grid system
- [ ] Build weather information cards with current and forecast data
- [ ] Add crop recommendation display with interactive cards
- [ ] Create quick stats overview (total farms, predictions made, etc.)
- [ ] Implement responsive design for mobile and tablet views

#### Day 5-6: Form Components & Input Handling

- [ ] Build comprehensive crop selection forms with autocomplete
- [ ] Create location input components with map integration
- [ ] Add soil parameter input forms with validation
- [ ] Build date range selectors for seasonal planning
- [ ] Implement form validation with user-friendly error messages

#### Day 7: State Management Setup

- [ ] Set up Redux Toolkit for complex state management
- [ ] Create actions and reducers for all major features
- [ ] Implement API call management with loading and error states
- [ ] Add user authentication state management

- [ ] Test state updates and persistence across components

## Week 11: Data Visualization Development

### Day 1-2: Chart Components with Chart.js

- [ ] Create weather trend charts (temperature, rainfall, humidity over time)
- [ ] Build yield prediction visualizations with confidence intervals
- [ ] Add soil parameter radar charts for multi-dimensional analysis
- [ ] Create crop comparison graphs (yield, profit, risk analysis)
- [ ] Implement interactive chart features (zoom, pan, tooltips)

### Day 3-4: Advanced Visualizations with Plotly

- [ ] Create 3D surface plots for soil parameter analysis
- [ ] Build time series visualizations with seasonal decomposition
- [ ] Add heatmaps for regional crop suitability
- [ ] Create gauge charts for risk assessments and scores
- [ ] Implement comparative analysis charts for different scenarios

### Day 5-6: Map Integration with Leaflet

- [ ] Set up interactive maps for farm location selection
- [ ] Add weather overlay maps with color-coded regions
- [ ] Create crop distribution visualizations on maps
- [ ] Implement location search and GPS coordinates
- [ ] Add custom map markers for farms, weather stations, markets

### Day 7: Real-time Chart Updates

- [ ] Connect charts to WebSocket data using Socket.io client
- [ ] Implement automatic chart refreshing without page reload
- [ ] Add smooth animations for data changes
- [ ] Test chart performance with continuous live data
- [ ] Add chart export functionality (PNG, PDF, CSV)

## Week 12: User Interface Enhancement

### Day 1-2: User Experience Improvements

- [ ] Add intuitive icons and agricultural imagery
- [ ] Create helpful tooltips and guided tours for new users
- [ ] Implement breadcrumb navigation for complex workflows
- [ ] Add comprehensive search functionality across all data
- [ ] Create user onboarding flow with tutorials

### Day 3-4: Mobile Optimization

- [ ] Test and optimize for mobile responsiveness
- [ ] Optimize touch interactions and gesture support
- [ ] Add mobile-specific UI elements (collapsible panels)
- [ ] Test on various screen sizes and devices
- [ ] Implement Progressive Web App (PWA) features

### Day 5-6: Accessibility & Performance

- [ ] Add ARIA labels and screen reader support
- [ ] Optimize image loading with lazy loading and compression
- [ ] Implement code splitting for faster initial load times
- [ ] Add keyboard navigation support throughout the app
- [ ] Test with accessibility tools and screen readers

### Day 7: User Testing & Feedback Collection

- [ ] Conduct usability testing with agriculture students
- [ ] Collect feedback on navigation and feature accessibility
- [ ] Identify confusing areas and pain points
- [ ] Document improvement suggestions with priority levels
- [ ] Plan UI/UX iterations based on feedback

## Week 13: Advanced Features Implementation

### Day 1-2: Notification System

- [ ] Create comprehensive in-app notification system
- [ ] Add email notifications for critical alerts (weather warnings)
- [ ] Implement SMS alerts for time-sensitive warnings
- [ ] Create notification preferences and customization
- [ ] Test notification delivery across different channels

### Day 3-4: Reporting & Analytics

- [ ] Build detailed farm performance reports with charts
- [ ] Create seasonal analysis reports with year-over-year comparisons
- [ ] Add PDF export functionality using libraries like ReportLab
- [ ] Implement data export in multiple formats (CSV, Excel, JSON)
- [ ] Create shareable report links with access controls

### Day 5-6: Multi-language Support

- [ ] Set up Django internationalization (i18n) framework
- [ ] Add Hindi language support for Indian farmers

- [ ] Create language switching functionality in frontend
- [ ] Translate key agricultural terms and technical jargon
- [ ] Test UI layout with different text lengths and scripts

**Day 7: Integration Testing**

- [ ] Test all features working together seamlessly
- [ ] Verify data consistency across all modules
- [ ] Test complete user workflows from registration to recommendations
- [ ] Check for performance bottlenecks under load
- [ ] Document and prioritize any issues found

## Week 14: Advanced ML Features

**Day 1-2: Computer Vision for Disease Detection**

- [ ] Collect and organize plant disease image datasets (PlantVillage, etc.)
- [ ] Implement CNN model using TensorFlow/PyTorch for image classification
- [ ] Create image preprocessing pipeline (resize, normalize, augment)
- [ ] Build Django API endpoint for image uploads and processing
- [ ] Test disease detection accuracy with validation dataset

**Day 3-4: Natural Language Processing Features**

- [ ] Implement chatbot for answering farmer queries using basic NLP
- [ ] Create FAQ system with intelligent search capabilities
- [ ] Add text analysis for processing user feedback and reviews
- [ ] Build recommendation explanation system in simple language
- [ ] Test NLP features with sample agricultural queries

**Day 5-6: Time Series Forecasting Enhancement**

- [ ] Implement advanced weather prediction models (LSTM, Prophet)
- [ ] Create market price forecasting with multiple algorithms
- [ ] Add seasonal pattern analysis and trend detection
- [ ] Build interactive forecasting visualizations
- [ ] Validate forecasting accuracy against historical data

**Day 7: Model Deployment & Monitoring**

- [ ] Optimize all models for production deployment
- [ ] Implement model versioning and A/B testing framework
- [ ] Create automated model performance monitoring
- [ ] Add model drift detection and retraining triggers
- [ ] Document model deployment and maintenance procedures

# Phase 3: Testing & Deployment (Weeks 15-20)

## Week 15: Comprehensive Testing

### Day 1-2: Unit Testing

- [ ] Write unit tests for all Django models and functions
- [ ] Create tests for ML model components and predictions
- [ ] Add tests for data processing and cleaning functions
- [ ] Test all API endpoints with various input scenarios
- [ ] Achieve >85% test coverage using coverage.py

### Day 3-4: Integration Testing

- [ ] Test frontend-backend integration with real API calls
- [ ] Verify database operations and data consistency
- [ ] Test third-party API integrations with error handling
- [ ] Check real-time data flow and WebSocket connections
- [ ] Test complete user authentication and authorization flow

### Day 5-6: Performance Testing

- [ ] Load test API endpoints using tools like Apache Bench or Locust
- [ ] Test database query performance with large datasets
- [ ] Check frontend rendering speed and responsiveness
- [ ] Test concurrent user scenarios and resource usage
- [ ] Identify and optimize performance bottlenecks

### Day 7: Security Testing

- [ ] Test API authentication and authorization edge cases
- [ ] Check for common vulnerabilities (SQL injection, XSS, CSRF)
- [ ] Test data validation and input sanitization
- [ ] Verify HTTPS implementation and SSL certificates
- [ ] Test user data privacy and access controls

## Week 16: User Acceptance Testing

### Day 1-2: Test Environment Setup

- [ ] Create production-like staging environment
- [ ] Set up realistic test data covering various scenarios
- [ ] Create test user accounts with different permission levels
- [ ] Document comprehensive test procedures and scenarios
- [ ] Set up feedback collection system and bug tracking

### Day 3-4: User Testing Sessions

- [ ] Conduct testing sessions with agriculture students
- [ ] Get feedback from farmers or agricultural cooperatives
- [ ] Gather input from agricultural experts and extension officers
- [ ] Document usability issues and feature requests
- [ ] Collect suggestions for improvement and additional features

### Day 5-6: Bug Fixes & Critical Improvements

- [ ] Fix all critical bugs identified during testing
- [ ] Implement high-priority usability improvements
- [ ] Update user interface based on feedback
- [ ] Optimize performance issues identified during testing
- [ ] Re-test all fixed issues to ensure they work correctly

### Day 7: Final Testing Round

- [ ] Conduct final comprehensive testing of all features
- [ ] Verify data accuracy and model predictions
- [ ] Test on different browsers, devices, and screen sizes
- [ ] Check all integrations and real-time features
- [ ] Get formal sign-off on testing completion

## Week 17: Production Deployment Preparation

### Day 1-2: Cloud Infrastructure Setup

- [ ] Choose and set up cloud provider (AWS/Google Cloud/DigitalOcean)
- [ ] Configure production servers with proper specifications
- [ ] Set up PostgreSQL database with backup configurations
- [ ] Configure domain name and SSL certificates
- [ ] Set up CDN for static files and media

### Day 3-4: CI/CD Pipeline Setup

- [ ] Create GitHub Actions workflow for automatic deployment
- [ ] Write deployment scripts for both backend and frontend
- [ ] Configure environment variables and secrets management
- [ ] Set up database migration scripts for production
- [ ] Test entire deployment pipeline in staging environment

### Day 5-6: Security & Monitoring Configuration

- [ ] Configure firewalls and security groups
- [ ] Set up automated database backups with retention policies

- [ ] Implement comprehensive logging and monitoring (Sentry, LogRocket)
- [ ] Configure SSL/TLS certificates with auto-renewal
- [ ] Test disaster recovery procedures and backup restoration

**Day 7: Performance & Optimization**

- [ ] Optimize database queries with proper indexing
- [ ] Configure Redis caching for frequently accessed data
- [ ] Compress and optimize static files (CSS, JS, images)
- [ ] Set up content delivery network (CDN) for global access
- [ ] Test production environment performance under load

## Week 18: Production Deployment

**Day 1-2: Initial Deployment**

- [ ] Deploy Django backend to production servers
- [ ] Deploy React frontend with proper build optimization
- [ ] Configure production database with initial data
- [ ] Set up domain DNS and SSL certificates
- [ ] Test basic functionality in production environment

**Day 3-4: Data Migration & Integration Setup**

- [ ] Import all production data safely with validation
- [ ] Configure API keys and third-party service credentials
- [ ] Set up scheduled jobs for automatic data updates (Celery + Redis)
- [ ] Test all external integrations (weather APIs, market data)
- [ ] Verify data accuracy and real-time updates

**Day 5-6: Final Production Configuration**

- [ ] Configure comprehensive monitoring and alerting systems
- [ ] Set up error tracking and performance monitoring
- [ ] Configure automated backup procedures and testing
- [ ] Test disaster recovery and rollback procedures
- [ ] Create detailed deployment and maintenance documentation

**Day 7: Go-Live Preparation & Testing**

- [ ] Conduct final production environment testing
- [ ] Test with a small group of beta users
- [ ] Monitor system performance and resource usage
- [ ] Address any last-minute critical issues
- [ ] Prepare launch announcement and user guides

## Week 19: Launch & Initial Monitoring

### Day 1-2: Official Launch

- [ ] Announce project launch on relevant platforms
- [ ] Monitor system performance and user activity closely
- [ ] Track user registrations, engagement, and usage patterns
- [ ] Respond promptly to user feedback and support requests
- [ ] Monitor and fix any immediate post-launch issues

### Day 3-4: User Support & Community Building

- [ ] Set up user support system (email, chat, documentation)
- [ ] Create comprehensive user documentation and video tutorials
- [ ] Respond to user queries and technical support requests
- [ ] Build community around the platform (social media, forums)
- [ ] Collect and analyze user feedback for future improvements

### Day 5-6: Performance Monitoring & Optimization

- [ ] Monitor system metrics, API performance, and database queries
- [ ] Track error rates, response times, and user satisfaction
- [ ] Monitor ML model performance and prediction accuracy
- [ ] Optimize any performance bottlenecks discovered
- [ ] Ensure system stability under increasing user load

### Day 7: Launch Week Analysis

- [ ] Analyze launch week performance and user adoption
- [ ] Document lessons learned and areas for improvement
- [ ] Plan immediate feature updates based on user feedback
- [ ] Update project documentation with post-launch insights
- [ ] Prepare detailed progress report for stakeholders

## Week 20: Documentation & Future Planning

### Day 1-2: Technical Documentation Completion

- [ ] Complete comprehensive API documentation with examples
- [ ] Document deployment procedures and system architecture
- [ ] Create troubleshooting guides for common issues
- [ ] Write detailed code documentation and comments
- [ ] Create system architecture diagrams and data flow charts

### Day 3-4: User Documentation & Training Materials

- [ ] Create detailed user manual with screenshots
- [ ] Make video tutorials for all major features
- [ ] Create FAQ section based on actual user questions
- [ ] Write help articles for common use cases
- [ ] Test documentation with new users for clarity

**Day 5-6: Project Portfolio Development**

- [ ] Create impressive project showcase presentation
- [ ] Write detailed technical project description for resume
- [ ] Document all technologies used and lessons learned
- [ ] Create demo videos showcasing key features
- [ ] Prepare project summary optimized for job interviews

**Day 7: Future Roadmap & Maintenance Planning**

- [ ] Plan next version features based on user feedback
- [ ] Identify technical debt and areas needing improvement
- [ ] Research new technologies and ML techniques to integrate
- [ ] Create maintenance schedule and update procedures
- [ ] Set up long-term project sustainability plan

## Tools & Technologies Summary

### Backend (Django)

- **Framework**: Django 4.x with Django REST Framework
- **Database**: PostgreSQL with Redis for caching
- **ML Libraries**: scikit-learn, TensorFlow, pandas, numpy
- **Task Queue**: Celery with Redis broker
- **Real-time**: Django Channels for WebSocket support

### Frontend (React)

- **Framework**: React 18 with TypeScript
- **UI Library**: Material-UI (MUI) for professional design
- **Charts**: Chart.js, Plotly.js for interactive visualizations
- **Maps**: Leaflet.js for location-based features
- **State Management**: Redux Toolkit

### DevOps & Deployment

- **Version Control**: Git with GitHub

- **CI/CD**: GitHub Actions
- **Cloud**: AWS/Google Cloud/DigitalOcean
- **Monitoring**: Sentry for error tracking
- **Documentation**: Swagger for API docs

## Success Metrics

- **Technical**: >90% uptime, <2s response time, >85% model accuracy
- **User**: Active user growth, positive feedback, feature adoption
- **Academic**: Research paper quality, innovation in AgriTech

This detailed plan gives you a day-by-day roadmap to build a professional, industry-ready AgriTech platform that will significantly enhance your resume and provide real value to the agricultural community.