Kisan Yantra: Detailed Web Application Development Plan (Django)

Goal: To build a scalable and secure agricultural equipment rental platform utilizing Django, Django REST Framework (DRF), PostGIS, and integrated with the WhatsApp Business API and Firebase Firestore for conversational state management.

Phase 0: Foundation & Architecture Setup (2 Weeks)

This phase establishes the core environment, database structure, and multi-service connectivity required for the complex components.

0.1 Project Initialization

- **A. Core Project Setup:** Initialize the Django project, create core apps (users, inventory, bookings).
- **B. Environment:** Configure Docker containers for the application (Django, PostgreSQL/PostGIS) and the task queue (Celery, Redis).
- **C. Dependencies:** Install core packages: Django, djangorestframework, psycopg2, python-dotenv, python-firebase-admin, django-celery-beat.

0.2 Database & Geospatial Setup

- **A. Primary Database (PostgreSQL/PostGIS):** Set up PostgreSQL and enable the **PostGIS** extension to prepare for geospatial queries and geo-fencing (Phase 3). Configure Django settings.py to use django.contrib.gis.db.backends.postgis.
- **B. Firestore Integration:** Initialize the Firebase Admin SDK within the Django project. Create a dedicated module (whatsapp_state_manager.py) for all Firestore CRUD operations, ensuring security rules are in place. **Firestore will be the single source of truth for all active WhatsApp session data.**

0.3 Initial Models

- Define the base models in the Django ORM (PostgreSQL):
  - User: (Inherit from AbstractUser) Roles: Farmer, Owner, Admin.
  - EquipmentCategory.
  - OwnerProfile.

Phase 1: Core Web Platform & Inventory Management (3 Weeks)

This phase focuses on the primary web interface, API structure, and CRUD operations for

equipment owners/admins.

1.1 Authentication & User Profiles

- **A. Custom Auth:** Implement Django Custom User model and authentication (login/registration).
- **B. DRF API Endpoints:** Set up Django REST Framework (DRF) for token-based authentication (e.g., Simple JWT).
- **C. Role-Based Access Control (RBAC):** Implement permissions to restrict Owners to their own inventory and Admins to everything.

1.2 Inventory Management Module

- **A. Equipment Model:** Define the Equipment model with essential fields: name, category, rate_per_day, owner (ForeignKey), and **GeoDjango PointField** for location.
- **B. CRUD API:** Develop DRF views and serializers for Owners to create, read, update, and delete their equipment listings.
- **C. Admin Interface:** Customize the Django Admin site to allow Admins to easily audit users and equipment.

Phase 2: Real-Time State & Conversational Core (4 Weeks)

This is the most critical integration phase, linking the WhatsApp channel to the Django business logic via the Gemini LLM.

2.1 WhatsApp Webhook Setup

- **A. Webhook Endpoint:** Create a public-facing DRF endpoint (/api/whatsapp/webhook/) to receive Meta Cloud API POST requests. This endpoint must handle the verification challenge (GET request) and incoming message events (POST requests).
- **B. Data Validation:** Implement checks to validate the signature and format of incoming messages from WhatsApp.

2.2 LLM & State Management

- **A. Asynchronous LLM Processing:** Use **Celery** to handle all interactions with the LLM API to prevent webhook timeouts. When a message is received:
  1. The Webhook view immediately sends the message payload to a Celery task (process_whatsapp_message).
  2. The view returns a 200 OK instantly.
- **B. Firestore Conversation State:** Inside the Celery task:
  1. Retrieve the user's current session state from Firestore (e.g., artifacts/{appId}/public/data/whatsapp_sessions/{user_id}).
  2. Pass the conversation history and the new message to the **Gemini API** with a System

Instruction tailored for farm equipment rental and information gathering (e.g., "Act as a helpful rental agent for Kisan Yantra...").
3. Receive the LLM response, which contains the next conversational step or a command (e.g., "BOOKING:Tractor:3_Days").

- **C. Outbound Messaging:** Based on the LLM output, use the Meta API to send the reply message back to the user.

2.3 Command Processing Bridge

- If the LLM response contains a structured command (e.g., BOOKING), the Celery task triggers a Django Service Layer function (handle_booking_request) to interact with the core PostgreSQL database.
- The function updates the PostgreSQL database (creating a draft booking) and updates the status in the Firestore session for the LLM to know the next step.

Phase 3: Geo-Logistics & Transaction Engine (4 Weeks)

Implementing the complex booking and geo-location features.

3.1 Geospatial Search & Proximity

- **A. Geo-Query Implementation:** Develop DRF endpoints to allow users (or the AI Agent) to search for equipment using proximity queries (e.g., "tractors within 25 km of this GPS point") leveraging **PostGIS spatial lookups** on the PointField.
- **B. Geo-fencing Logic:** Implement business logic using PostGIS functions to determine if the equipment is returned within a specified, pre-defined geofence boundary (e.g., the owner's storage area).

3.2 Booking & Reservation Logic

- **A. Booking Model:** Create the Booking model (ForeignKey to Equipment, Farmer, Owner, start_date, end_date, total_cost, status).
- **B. Availability Check:** Implement atomic transactions to ensure equipment is not double-booked across the web portal and the WhatsApp channel.
- **C. Pricing Engine:** Implement logic for calculating rent, taxes, and potential late-return fees based on geo-fencing violation data.

3.3 Payments Integration

- **A. Payment Gateway:** Integrate a regional payment gateway popular in India (e.g., Razorpay or PayU) using Django webhooks to handle transaction completion and updates.
- **B. Booking Status Updates:** The payment webhook updates the PostgreSQL Booking status, which can then trigger an asynchronous notification (via Celery) back to the user

on WhatsApp or the Web Portal.

Phase 4: Testing, CI/CD, and Deployment (2 Weeks)

Preparing the entire system for production use and scaling.

4.1 Testing

- **A. Unit Tests:** Write comprehensive unit tests for all models, serializers, and utility functions (especially geo-queries and pricing).
- **B. Integration Tests:** Crucial tests for the WhatsApp flow: WhatsApp Message -> Webhook -> Celery -> LLM API -> Firestore State Update -> Database Booking -> Outbound WhatsApp Reply.

4.2 CI/CD and Logging

- **A. CI/CD Pipeline:** Implement Continuous Integration/Continuous Deployment (CI/CD) using a platform like GitHub Actions or GitLab CI to automate testing and deployment to a cloud host (e.g., GCP, AWS, or DigitalOcean).
- **B. Monitoring:** Set up logging (e.g., Sentry) and performance monitoring to track API latency, Celery queue depth, and webhook success rates.

4.3 Final Deployment

- **A. Production Server:** Deploy the Django application behind a proxy server (Nginx/Gunicorn).
- **B. HTTPS/SSL: Crucially, ensure HTTPS is enabled**, as it is a mandatory requirement for Meta's Webhook configuration.

Summary of Key Technologies:

Component	Technology	Rationale
Backend Core	Django, DRF	Rapid development, ORM, security.
Primary DB	PostgreSQL, PostGIS	Robust, transactional, essential for geospatial features.
Real-Time State	Firebase Firestore	Handles live, lightweight conversational history and chat state.
Asynchronous Tasks	Celery, Redis	Prevents webhook timeouts during LLM calls (Source

		1.1).
AI Integration	Gemini API (via Celery)	Conversational intelligence and intent recognition.
Geo-Services	GeoDjango, PostGIS	Standard for geo-fencing and proximity search (Source 3.4).