SDLC Plan for Farmer Utility App

1. Planning:

Objective: Develop an app to support farmers with crop prices, financial management, and secure
transactions.
Stakeholders: Farmers, your family business, developers, and potential investors.
Scope:
- Provide real-time crop prices.
- Track payments, outstanding balances, and advances.
- Integrate AI for predicting crop prices.
- Use Razorpay for secure and transparent transactions.
- Offer scalability using cloud computing.

Resources:
- Development team (you and your teammate).
- Tools: Django, Node.js, cloud services.

Timeline: Estimate the time for each phase, from development to deployment.

2. Requirements Analysis:
Functional Requirements:
- User registration and login for farmers.
- Interface to view and update crop prices.
- Payment tracking for farmers (advances, outstanding balances).
- Notifications for price updates.
- AI-based price prediction model.

Non-Functional Requirements:
- Scalability: Cloud storage and processing.
- Performance: Quick response for real-time data.
- Reliability: Ensure availability during crucial seasons.

Technology Stack:
- Backend: Django with rest frameworks
- Frontend: React native for UI
- Database: Cloud-based database
- AI: Implement a basic ML model for crop price predictions.
-

3. Design:
System Architecture:
- Frontend: User-friendly interface to view crop prices and manage payments.

- Backend: Server-side handling of requests, AI prediction
- Database: Store user data, transaction records, and crop prices.
- API: Create APIs for the frontend to communicate with the backend.

User Interface Design:
- Simple, intuitive screens for farmers to view crop prices and payment details.
- Notifications for price updates or balance reminders.

4. Implementation:Development Phases:

- Frontend: Using React Native for App Frontend for the interface. Create forms for farmers to input
and view data.
- Backend: Develop with Django for the business logic, integrate Django rest framework for handling API approach
requests, and create routes for adding/viewing crop prices.
- AI Integration: Implement machine learning algorithms for price prediction using Python libraries
like scikit-learn.
- Payment Integration: Create smart contracts for payment handled by Razorpay
- Cloud Integration: Use AWS to Host Backend

Version Control: Use GitHub for version tracking.

5. Testing:

Unit Testing: Test individual components (crop price updates, payment calculations, etc.).
Integration Testing: Ensure that the frontend, backend, AI, together smoothly.
Security Testing: Ensure that all transactions are secure and encrypted.
User Acceptance Testing (UAT): Conduct testing with a small group of farmers and your family
business to get feedback on functionality and usability.
Performance Testing: Test for load handling, especially during harvest seasons when traffic might
spike.

6. Deployment:

Preparation: Finalize the app, ensuring that it's ready for deployment on cloud platforms.
Deployment Platform: Use AWS for deploying the app.
Database Migration: Ensure smooth migration to a live database.

Domain & Hosting: Set up a custom domain (if necessary) and host the app on a reliable service.
Monitoring: Use monitoring tools to ensure app stability (e.g., New Relic, AWS CloudWatch).

7. Maintenance:
Ongoing Support: Regular updates based on farmer feedback, resolving bugs, adding features.
Scaling: Adjust server capacity or database storage based on usage.

AI Model Updates: Retrain AI models periodically for improved crop price predictions.