Project Design Phase-II
Technology Stack (Architecture & Stack)

Date	20 July 2025
Team ID	LTVIP2025TMID41443
Project Name	Transfer Learning-Based Classification of Poultry Diseases for Enhanced Health Management
Maximum Marks	4 Marks

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2

Example: Order processing during pandemics for offline mode

Order processing during pandemics (Offline Mode)

Local data storage

Store user interactions such as orders and updates securely on the device.

Deferred syncing

Automatically sync data with the server

Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	Web and mobile-based UI for farmers to upload poultry images	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	Backend logic for image processing and disease classification	Python
3.	Application Logic-2	Disease prediction using trained ML model	TensorFlow / Keras with Transfer Learning
4.	Application Logic-3	Treatment suggestion based on predicted disease	Python logic + JSON-based mapping
5.	Database	Stores user accounts and image metadata	SQLite / MySQL
6.	Cloud Database	Stores remote data for analytics (optional)	IBM Cloudant / Firebase Realtime DB
7.	File Storage	Stores uploaded poultry images locally or in the cloud	Local Filesystem / IBM Cloud Object Store
8.	External API-1	Weather conditions for disease correlation (optional)	IBM Weather API, etc.

9.	External API-2	Location-based vet services (optional)	Aadhar API, etc.
10.	Machine Learning Model	Classifies poultry images into disease categories	MobileNet / ResNet Transfer Learning Model
11.	Infrastructure (Server / Cloud)	Hosts the app and ML model backend	Local Server / Render / Heroku / IBM Cloud

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Frontend, backend, and ML libraries used are open-source	Flask, TensorFlow, Bootstrap, SQLite
2.	Security Implementations	Password hashing, file validation, and API protection	SHA-256, HTTPS, Flask-CORS, JWT (optional)
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	3-tier architecture + Docker-ready
4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Render, Heroku, IBM Cloud Foundry
5.	Performance	Caching predictions, optimized image size, fast API responses	Flask Cache, CDN (optional), SQLite Indexing