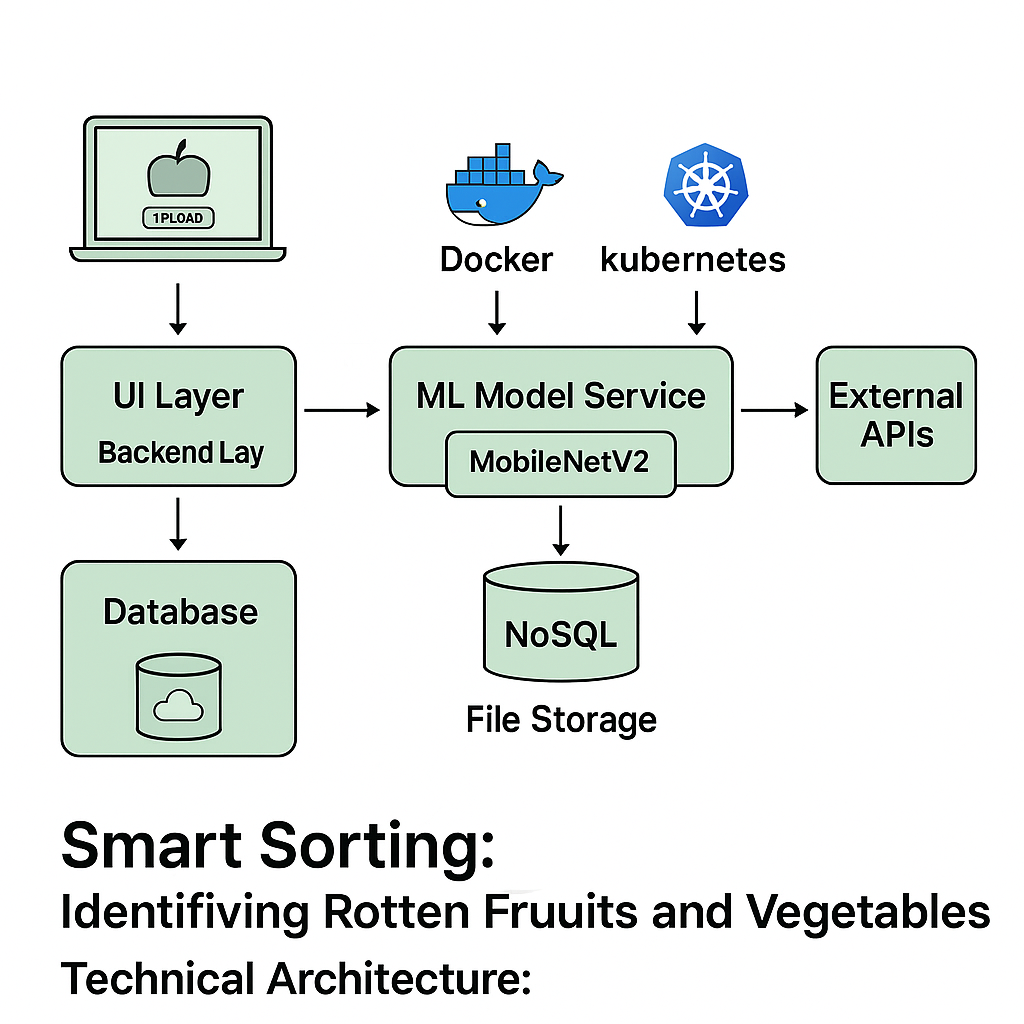
**Project Design Phase-II**

**Technology Stack (Architecture & Stack)**

|  |  |
| --- | --- |
| Date | 27 June 2025 |
| Team ID | LTVIP2025TMID35598 |
| Project Name | Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables |
| Maximum Marks | 4 Marks |

**Technical Architecture:**

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



**Guidelines:**

* UI Layer: Web app to upload images.
* Backend Layer: Python Flask API for prediction.
* ML Model Service: TensorFlow/Keras REST API serving pre-trained transfer learning model (e.g., MobileNetV2).
* Storage: Cloud storage (AWS S3, Azure Blob) for images.
* Database: NoSQL (MongoDB) to store predictions and logs.
* Scalability: Containerized deployment (Docker) on Kubernetes/Cloud Foundry.
* Security: HTTPS, authentication, access controls**.**

**Table-1 : Components & Technologies:**

| **S.No** | **Component** | **Description** | **Technology** |
| --- | --- | --- | --- |
| 1. | User Interface | Web UI for image upload, results display | HTML, CSS, JavaScript, React.js |
| 2. | Application Logic-1 | Backend API for handling requests, prediction calls | Python (Flask / Django REST Framework) |
| 3. | Application Logic-2 | Image preprocessing and transformation pipeline | OpenCV, Pillow |
| 4. | Application Logic-3 | Transfer learning inference | TensorFlow / Keras Model Serving |
| 5. | Database | Store prediction logs, user data | MongoDB |
| 6. | Cloud Database | Managed database service | MongoDB Atlas / Firebase Firestore |
| 7. | File Storage | Store uploaded images | AWS S3 / Azure Blob Storage |
| 8. | External API-1 | Optional: Weather API to link spoilage probability (future) | OpenWeather API |
| 9. | External API-2 | Optional: Notifications API (Email/SMS) | Twilio / SendGrid API |
| 10. | Machine Learning Model | Predict rotten vs fresh produce using transfer learning | MobileNetV2 trained on custom dataset |
| 11. | Infrastructure | Application hosting and scaling | Docker, Kubernetes, AWS EC2 / Azure App Service |

Table-2: Application Characteristics:

| **S.No** | **Characteristics** | **Description** | **Technology/Approach** |
| --- | --- | --- | --- |
| 1. | Open-Source Frameworks | Backend, ML, and frontend frameworks | Flask, TensorFlow, React.js |
| 2. | Security Implementations | Data encryption, HTTPS, authentication, access control | SSL/TLS, JWT Authentication, IAM Policies |
| 3. | Scalable Architecture | Containerized microservices, independent scaling of backend and ML model | Docker, Kubernetes, REST APIs |
| 4. | Availability | High availability via load balancer, redundant instances | AWS Load Balancer, Auto-Scaling Groups |
| 5. | Performance | Optimized prediction pipeline, caching, preloaded model, CDN for static assets | Redis Caching, CloudFront CDN, TensorFlow Model Server |