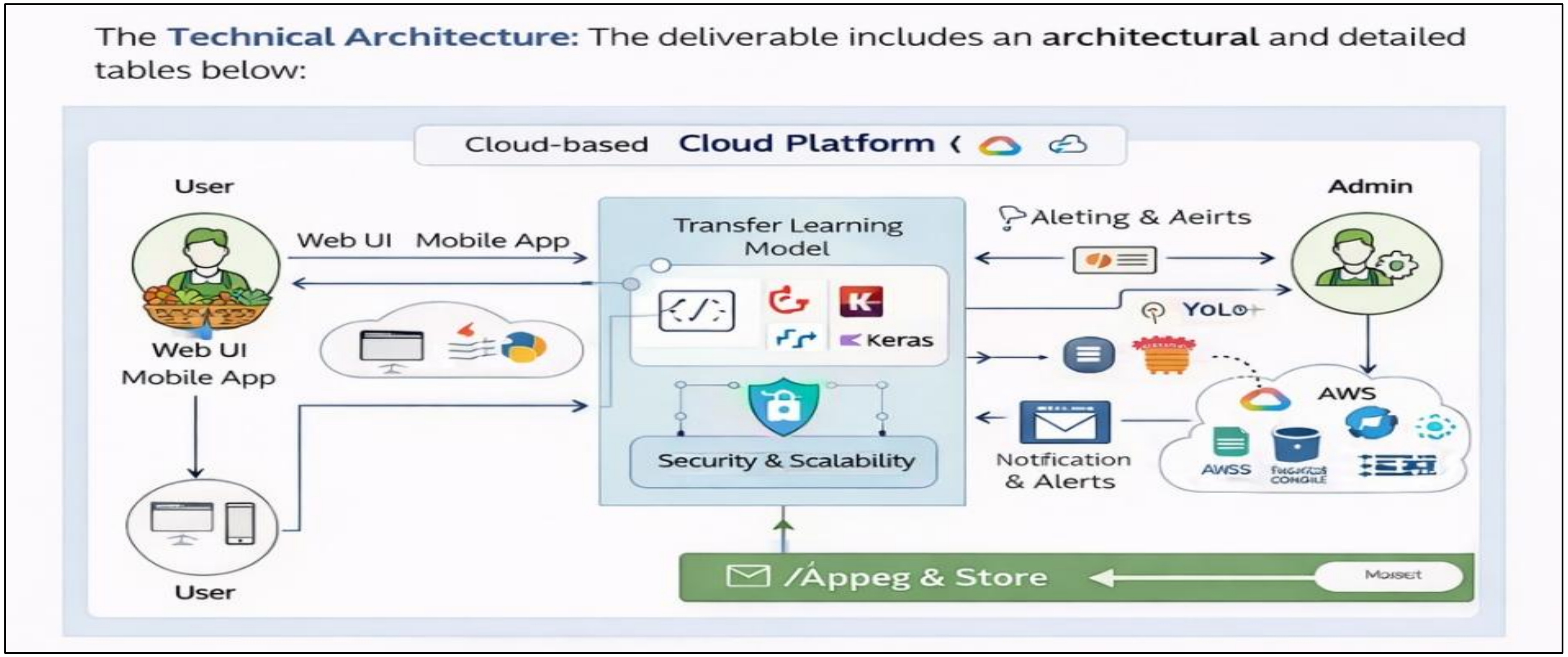
Project Design Phase-II
Technology Stack (Architecture & Stack)

Date	19 February 2026
Team ID	LTVIP2026TMIDS77295
Project Name	Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

Technical Architecture:

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

Example: “Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables”.



Project Flow

1. User Access and Navigation

1. The user opens a browser and visits the application URL (e.g., http://127.0.0.1:5000/).
2. The Home page loads, showing the project title, brief description, and navigation menu (Home, About, Predict, Contact).
3. From the navbar, the user can move to:
   - About – to read about the model and metrics.
   - Predict – to perform fruit/vegetable freshness prediction.
   - Contact – to view team and contact details.

---

2. Dataset and Model Preparation (Offline Flow)

1. Images of fresh and rotten fruits/vegetables are collected, cleaned, and organized into labeled folders (Fresh, Rotten). Similar workflows are used in fruit quality projects.
2. The dataset is split into training and testing sets and loaded using data generators.
3. A MobileNetV2-based model is built using TensorFlow/Keras, with a custom classification head for Fresh vs Rotten.
4. The model is trained, evaluated (accuracy, confusion matrix, F1-score), and the best weights are saved to a file such as model/healthy_vs_rotten.h5.
5. This trained model file is then used by the Flask app during runtime for predictions, so training does not happen every time the app runs.

---

3. Prediction Flow (End-to-End Runtime)

1. The user clicks on the Predict menu item to open the prediction page.
2. On the Predict page, the user:
   - Clicks “Choose File” / “Browse”.

- Selects an image of a fruit or vegetable from their device.
- Clicks the Predict / Upload button.

3. The browser sends a POST request to the /predict route of the Flask server with the image file attached.
4. In the Flask backend:
   - The server checks if a file has been uploaded and whether its extension is allowed (e.g., .jpg, .png).
   - The image is saved temporarily in the uploads/ directory.
   - The saved image path is passed to a prediction function (e.g., in predict.py).
5. In the prediction function:
   - The image is opened and preprocessed (resize to target size such as 224×224, convert to RGB, normalize pixels) to match the MobileNetV2 input format.
   - The preprocessed image is fed into the loaded model (healthy_vs_rotten.h5).
   - The model outputs a probability score; this is converted into:
     - Predicted class: Fresh or Rotten.
     - Confidence percentage (e.g., 92.3%).
6. The prediction result (label and confidence) and the image path are returned to Flask and passed to the Result template.
7. The Result page is rendered, showing:
   - The uploaded image.
   - The predicted label (Fresh/Rotten).
   - The confidence score.
   - Optional message (e.g., “This fruit is likely fresh with high confidence”).

---

4. About and Contact Flow

1. When the user clicks About, Flask serves a page that explains:
    - Project idea and purpose.
    - Technologies used (Flask, TensorFlow, MobileNetV2, etc.).
    - Model performance metrics (accuracy, dataset size, classes).
2. When the user clicks Contact, the Contact page shows:
    - Team member names and roles.
    - Project/college identifiers or contact information.

---

5. Error Handling Flow

1. If the user submits the Predict form without selecting an image, Flask detects the missing file and:
    - Does not call the model.
    - Returns the Predict page again with a clear error message (e.g., "Please upload an image").
2. If the user uploads an invalid file type:
    - The backend checks the extension and rejects the file.
    - A friendly error message is shown instead of a server crash.

---

6. Optional Future Flow (Extension)

You can also mention how the flow would extend in future:

- Store each prediction (image path, result, timestamp) in a database for history and analytics.
- Add user login so only authorized users can access prediction or admin pages.
- Deploy the Flask app to a cloud platform so external systems or mobile apps can call the prediction API. Similar deployments are shown in other Flask + MobileNetV2 applications.

Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	Web UI where user opens Home, About, Predict, Contact and uploads fruit/vegetable images.	HTML5, CSS3, JavaScript, Jinja2 templates (Flask).
2.	Application Logic-1	Web application backend handling routing (/, /about, /predict, /contact) and form handling.	Python 3, Flask web framework.
3.	Application Logic-2	Image upload logic: validation, saving file to uploads/, building URL for result page.	Python (Flask request, send_from_directory, os)
4.	Application Logic-3	Prediction logic: calling predict_image(), preprocessing image, passing to model, mapping to label.	Python (TensorFlow/Keras, NumPy, tensorflow.keras.image).
5.	Database	(Optional/Planned) Store prediction history, image path, timestamp, label and confidence.	SQLite or MySQL (simple relational schema).
6.	Cloud Database	If deployed to cloud and history needed, same schema hosted as managed DB.	MySQL on cloud / PostgreSQL on Render/Heroku.
7.	File Storage	Store uploaded test images and trained model file.	Local filesystem: /uploads/, /model/healthy_vs_rotten.h5
8.	External API-1	(Planned) Email/contact integration for feedback or alerts.	SMTP email API / Gmail SMTP.
9.	External API-2	(Optional future) Cloud storage or monitoring API if app is deployed at scale.	AWS S3 / GCP Storage / logging API.
10.	Machine Learning Model	Binary classifier to detect Fresh vs Rotten using transfer learning.	MobileNetV2 base (ImageNet weights) + custom dense head (Keras).
11.	Infrastructure (Server / Cloud)	Deployment environment hosting Flask app, model, and static files.	L Local machine (development) and optional cloud VM / PaaS (e.g., Render/Heroku).

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Core stack is built completely on open-source libraries and tools.	Flask, TensorFlow/Keras, NumPy, Bootstrap/FontAwesome, Python 3.
2.	Security Implementations	Basic security: server-side validation of uploads, limit to image types, hide model path, no direct directory listing.	Flask send_from_directory, file type checks, app-level access control, HTTPS when deployed.
3.	Scalable Architecture	Three-layer style: UI (templates) → Flask API layer → ML model layer; can be separated into micro-services later if needed.	3-tier web architecture on Python/WSGI; model callable as separate service in future.
4.	Availability	In dev: single Flask instance; in production can run behind a WSGI server with multiple workers and health checks.	Gunicorn/uWSGI, Nginx reverse proxy (when deployed).
5.	Performance	Pre-loaded model in memory; images resized to 224×224; lightweight MobileNetV2 for fast inference (< few seconds per image).	TensorFlow/Keras optimization, caching model at startup, using GPU/CPU acceleration if available.
6.	Usability	Simple navigation (Home, About, Predict, Contact), responsive layout, dark/light theme toggle.	CSS, responsive design, minimal clicks to predict.
7.	Maintainability	Clear separation into scripts (split_dataset.py, train.py, predict.py, app.py) and templates/static assets.	Modular Python packages, template inheritance in Flask.
8.	Portability	Runs on any OS with Python; can be containerized and deployed on different clouds.	Python virtual env, Docker (optional).