Hariyali (BHOOMI) - Detailed Working Module Report

1. Overview

Hariyali (BHOOMI) is a Flask-based web application that provides:

- Crop recommendation based on soil nutrients, weather, and pH
- Fertilizer recommendation based on target crop and N-P-K gaps
- Plant disease detection from leaf images using a CNN (ResNet9)

Deployed components include a Flask server, scikit-learn model (RandomForest), and a PyTorch model for disease classification.

2. High-Level Architecture

```
flowchart LR
  subgraph Client
    A[Browser UI]
  end

  subgraph Server[Flask App]
    B1[Routes: /, /crop-recommend, /fertilizer, /disease]
    B2[Controllers: crop-predict, fertilizer-predict, disease-predict]
    B3[Services: weather_fetch, predict_image]
    B4[Models: RandomForest.pkl, ResNet9.pth]
    B5[Data: Data/fertilizer.csv]
    B6[Templates & Static]
  end

  A <--> B1
  B1 --> B2
  B2 --> B3
  B2 --> B4
  B2 --> B5
  B1 --> B6
```

3. Modules and Responsibilities

- UI Layer (templates, static)
  - `templates/index.html` : Landing page and navigation
  - `templates/crop.html` : Crop input form (N, P, K, pH, rainfall, city)
  - `templates/fertilizer.html` : Fertilizer input form (crop, N, P, K)
  - `templates/disease.html` : Image upload form
  - `templates/*-result.html` : Display prediction outputs
  - Static assets: Bootstrap CSS, images, client JS (e.g., `static/scripts/cities.js`)
- Flask App (`app/app.py`)
  - Routing and controllers
  - Loads models: `models/RandomForest.pkl` , `models/plant_disease_model.pth`
  - Services: `weather_fetch` , `predict_image`

    - Integrates domain dictionaries: `utils/fertilizer.py` , `utils/disease.py`

- Utils
    - `utils/model.py` : ResNet9 architecture
    - `utils/fertilizer.py` : `fertilizer_dic` mapping N/P/K status to advice
    - `utils/disease.py` : `disease_dic` mapping predicted classes to guidance
- Data
    - `app/Data/fertilizer.csv` : Crop-wise recommended N, P, K values
    - `Data-processed/crop_recommendation.csv` : Dataset used during training

4. Detailed Flows

4.1 Crop Recommendation Flow

```
sequenceDiagram
  participant U as User (Browser)
  participant F as Flask Controller (/crop-predict)
  participant W as weather_fetch()
  participant M as RandomForest.pkl

  U->>F: POST N,P,K,pH,rainfall,city
  F->>W: Get temperature, humidity (OpenWeatherMap)
  W-->>F: temperature, humidity (fallback defaults on error)
  F->>M: Predict crop with [N,P,K,temp,humidity,pH,rainfall]
  M-->>F: crop_name
  F-->>U: Render crop-result.html with prediction
```

Input features: N, P, K, temperature, humidity, pH, rainfall. Model: scikit-learn RandomForest.

4.2 Fertilizer Recommendation Flow

```
sequenceDiagram
  participant U as User (Browser)
  participant F as Flask Controller (/fertilizer-predict)
  participant D as Data/fertilizer.csv
  participant L as fertilizer_dic

  U->>F: POST crop, N, P, K
  F->>D: Lookup recommended N,P,K for crop
  D-->>F: N_req, P_req, K_req
  F->>F: Compute deltas, choose max deficit/excess key
  F->>L: Map key to advisory text
  L-->>F: Recommendation (HTML safe)
  F-->>U: Render fertilizer-result.html
```

Logic: identify the nutrient with greatest absolute deviation and pick from `NHigh/Nlow/PHigh/Plow/KHigh/Klow` .

4.3 Disease Detection Flow

```
sequenceDiagram
  participant U as User (Browser)
  participant F as Flask Controller (/disease-predict)
  participant P as predict_image()
  participant C as ResNet9 (PyTorch)
  participant DD as disease_dic

  U->>F: POST image file
  F->>P: Read bytes and preprocess (Resize, ToTensor)
  P->>C: Forward pass
  C-->>P: class index
  P-->>F: class label
  F->>DD: Map class to description
  DD-->>F: Advisory content (HTML safe)
  F-->>U: Render disease-result.html
```

Preprocessing: Resize(256), ToTensor, batch dimension. Inference runs on CPU.

5. Data Models and Files

- RandomForest crop model: `app/models/RandomForest.pkl`
  - Inputs: [N, P, K, temperature, humidity, pH, rainfall]
  - Output: crop label (string)
- Disease model: `app/models/plant_disease_model.pth`
  - Architecture: ResNet9, 3-channel input, 38 classes
  - Classes defined in `app/app.py` `disease_classes`
- Fertilizer reference: `app/Data/fertilizer.csv`
  - Columns: Crop, N, P, K

6. Configuration and Deployment

- Config
  - `app/config.py` : `weather_api_key` for OpenWeatherMap. In production, use environment variables.
- Local run
  - `python app/app.py` (default port 8080), templates served from `app/templates`
- Production
  - `app/Procfile` : `web: gunicorn app:app --log-level debug`
  - Ensure models and data paths are correct relative to `app/`

7. Error Handling and Defaults

- Weather API failures fall back to default temperature/humidity (25.0, 60.0)
- Disease upload validates presence of `file` ; non-image exceptions render input page

8. Security and Privacy Notes

- Do not commit real API keys; use environment variables and secrets management
- Validate and sanitize file uploads; restrict file size and type

9. Future Enhancements

- Add unit tests for services and controllers
- Replace blocking external API call with cached or async approach
- Improve preprocessing (normalization) for disease model
- Add localization and accessibility improvements in UI

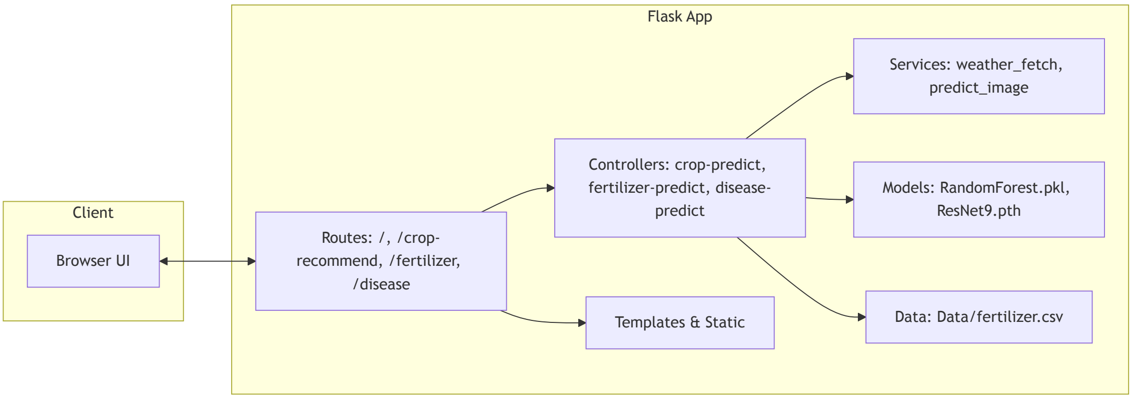
10. Appendix: Route Summary

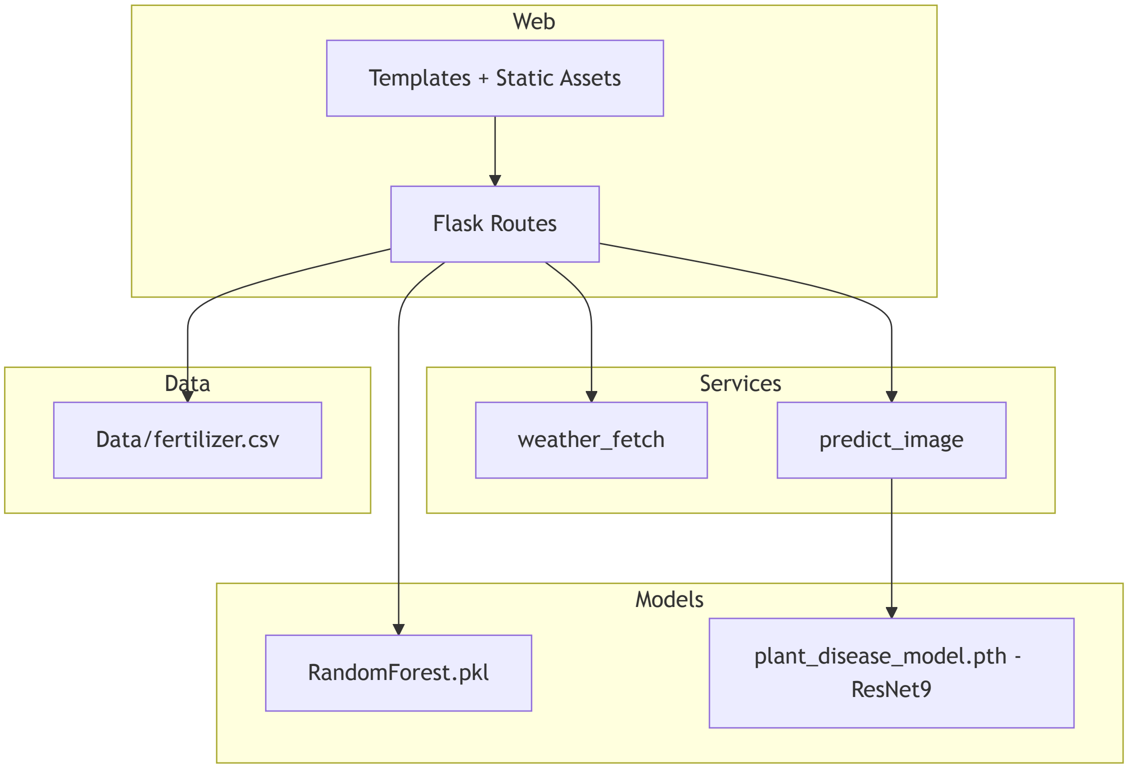
Route	Method	Purpose
`/`	GET	Home page
`/crop-recommend`	GET	Crop form
`/crop-predict`	POST	Predict crop
`/fertilizer`	GET	Fertilizer form
`/fertilizer-predict`	POST	Recommend fertilizer
`/disease`	GET	Disease form
`/disease-predict`	GET/POST	Predict disease

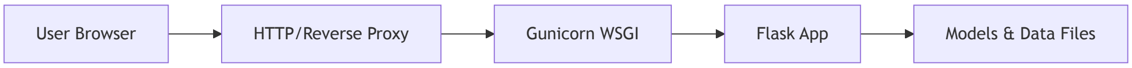
11. Printable Diagrams

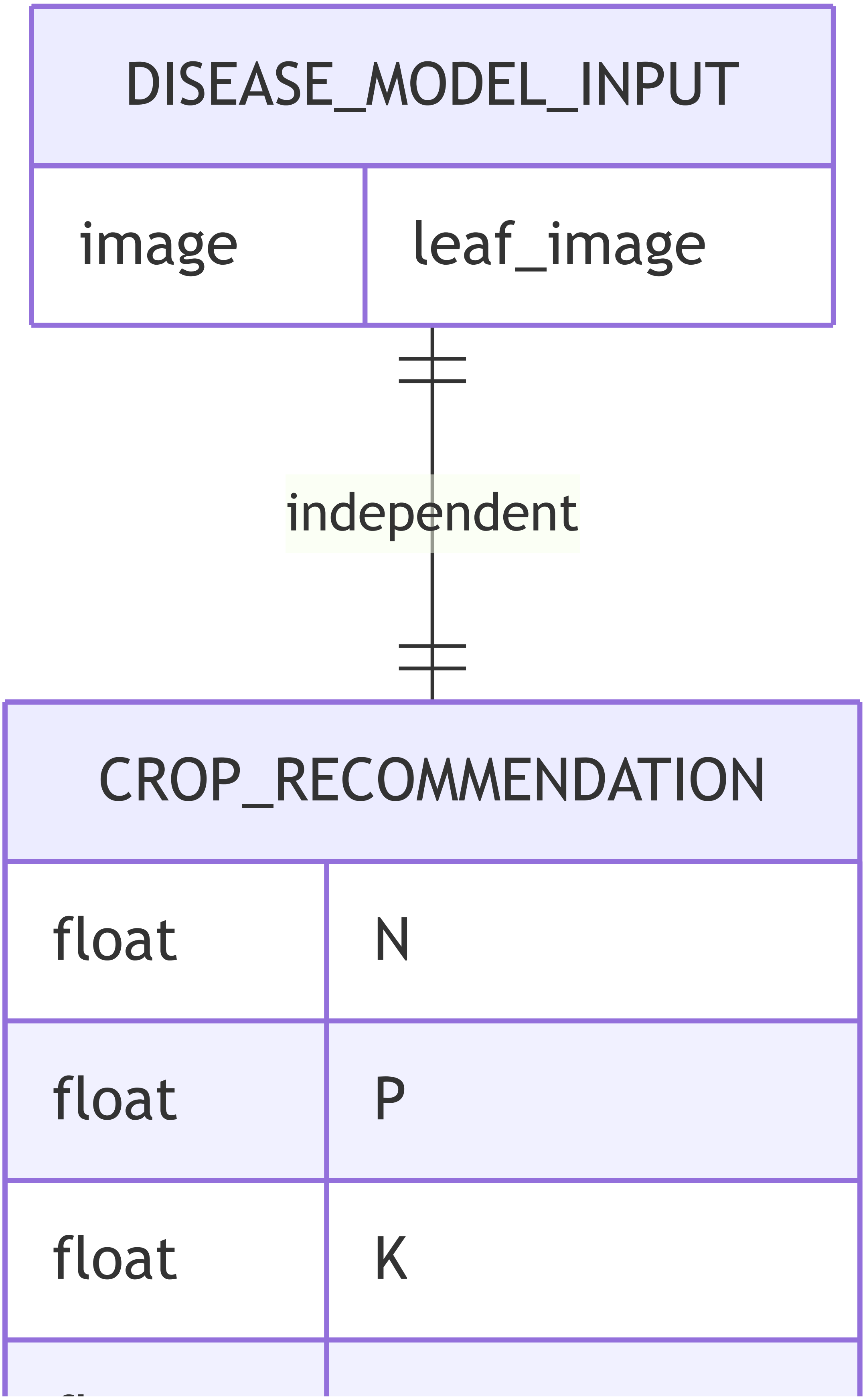
Rendered SVGs for printing/sharing:

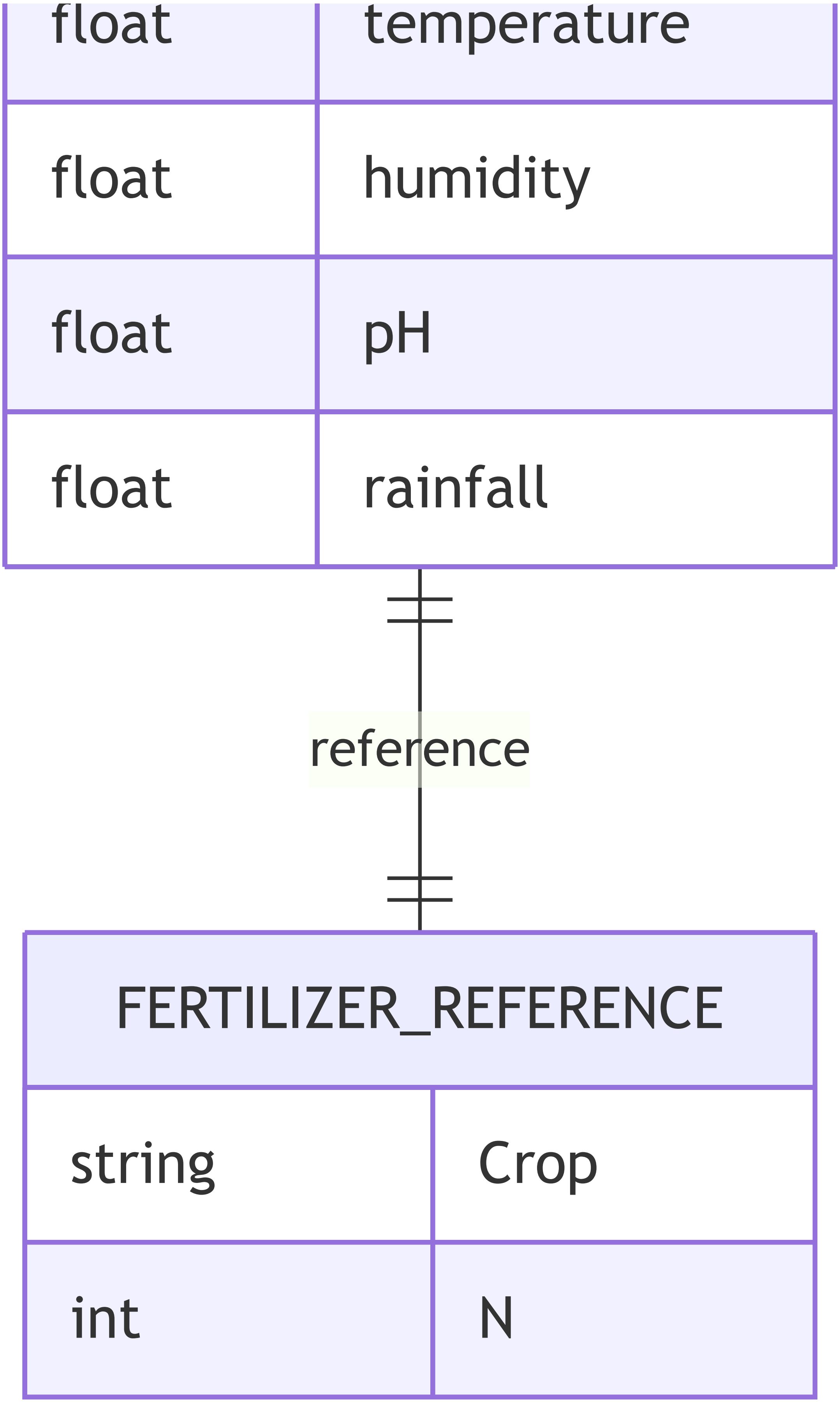
- Architecture: `docs/diagrams/architecture.svg`
- Component: `docs/diagrams/component.svg`
- Deployment: `docs/diagrams/deployment.svg`
- ERD: `docs/diagrams/erd.svg`
- Sequences:
  - Crop: `docs/diagrams/crop_seq.svg`
  - Fertilizer: `docs/diagrams/fert_seq.svg`
  - Disease: `docs/diagrams/disease_seq.svg`


Flask App
Client
Browser UI
Routes: /, /crop-recommend, /fertilizer, /disease
Controllers: crop-predict, fertilizer-predict, disease-predict
Services: weather_fetch, predict_image
Models: RandomForest.pkl, ResNet9.pth
Data: Data/fertilizer.csv
Templates & Static


Web
Templates + Static Assets
Flask Routes
Data
Data/fertilizer.csv
Services
weather_fetch
predict_image
Models
RandomForest.pkl
plant_disease_model.pth - ResNet9


User Browser
HTTP/Reverse Proxy
Gunicorn WSGI
Flask App
Models & Data Files

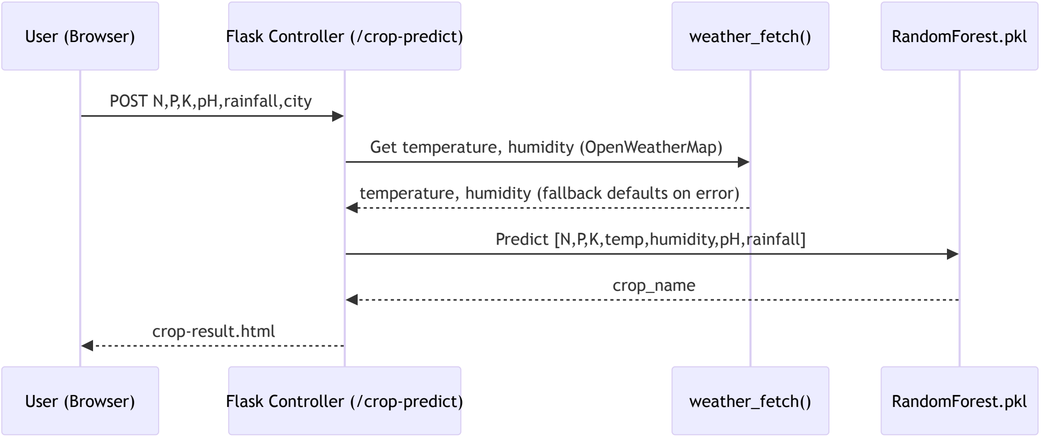

DISEASE_MODEL_INPUT
image
leaf_image
independent
CROP_RECOMMENDATION
float
N
float
P
float
K

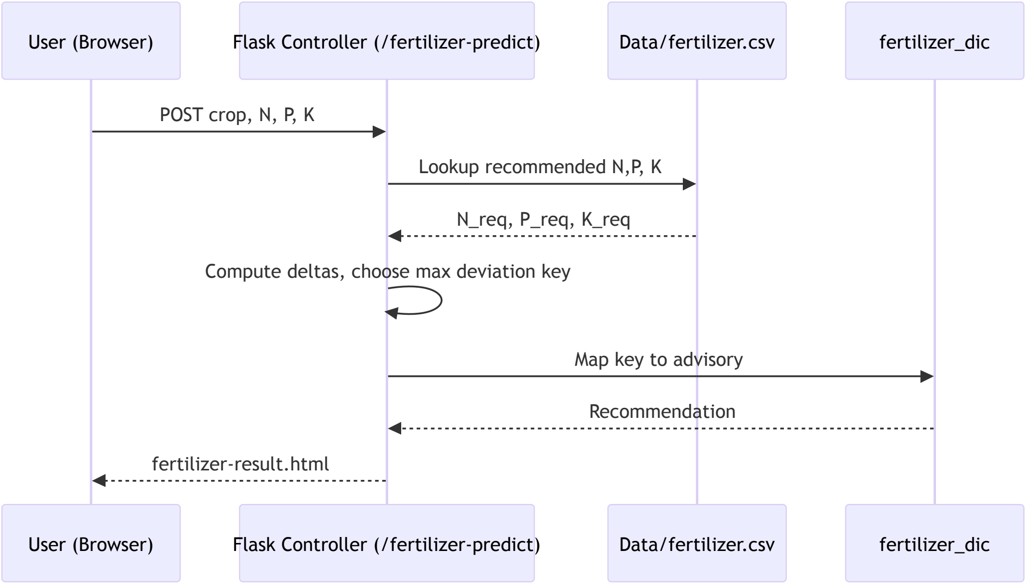


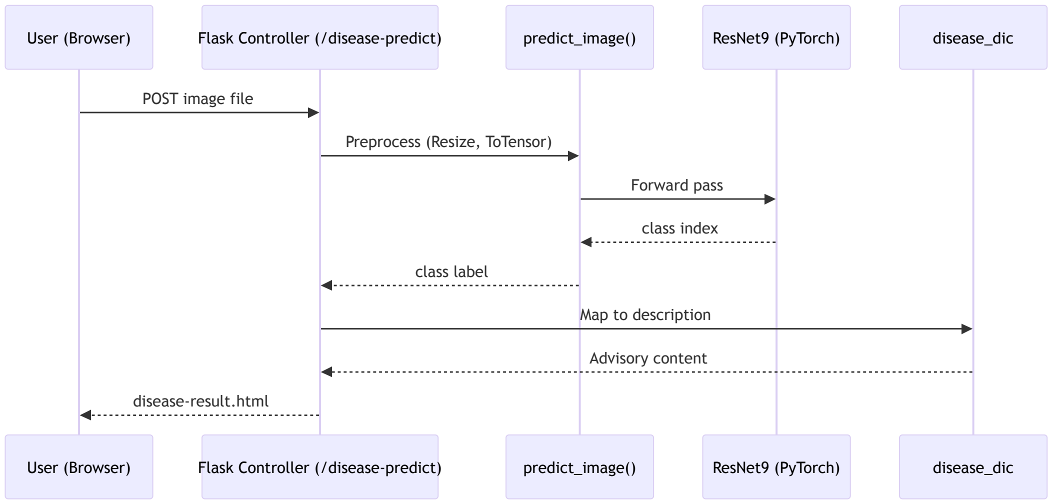
float	temperature
float	humidity
float	pH
float	rainfall

FERTILIZER_REFERENCE	
string	Crop
int	N

int	P
int	K


User (Browser)
Flask Controller (/crop-predict)
weather_fetch()
RandomForest.pkl
POST N,P,K,pH,rainfall,city
Get temperature, humidity (OpenWeatherMap)
temperature, humidity (fallback defaults on error)
Predict [N,P,K,temp,humidity,pH,rainfall]
crop_name
crop-result.html
User (Browser)
Flask Controller (/crop-predict)
weather_fetch()
RandomForest.pkl


User (Browser)
Flask Controller (/fertilizer-predict)
Data/fertilizer.csv
fertilizer_dic
POST crop, N, P, K
Lookup recommended N,P, K
N_req, P_req, K_req
Compute deltas, choose max deviation key
Map key to advisory
Recommendation
fertilizer-result.html
User (Browser)
Flask Controller (/fertilizer-predict)
Data/fertilizer.csv
fertilizer_dic


User (Browser)
Flask Controller (/disease-predict)
predict_image()
ResNet9 (PyTorch)
disease_dic
POST image file
Preprocess (Resize, ToTensor)
Forward pass
class index
class label
Map to description
Advisory content
disease-result.html
User (Browser)
Flask Controller (/disease-predict)
predict_image()
ResNet9 (PyTorch)
disease_dic