IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

SasyaVedha: A Soil-Aware and Climate-Resilient Agronomic Decision Support System for Indian Farms

Nishanth U[1], Lalith N[2], Punith Y D[3]

Student, Dept. of ISE (Information Science and Engineering), AMC Engineering College,

AMC Campus Bannerghatta Rd, Kalkere, Bengaluru – 560083, Karnataka, India[1,2,3]

Abstract: Indian farmers routinely have to make five separate decisions in a single season — what disease is on the leaf, how much yield to expect, when to irrigate, what to feed the soil, and how a coming drought or flood will change all three — and most digital tools answer only one of these at a time. This paper presents SasyaVedha, a web-based agronomic decision support system that brings crop disease identification, yield prediction, irrigation scheduling, fertilizer advisory, and climate-event impact simulation into a single Flask-and-React platform, alongside an enterprise mode for multi-farm operations with role-based access control and an append-only audit log. Disease identification uses a MobileNetV2 transfer-learning CNN trained on the PlantVillage dataset, with a deliberate fallback to a curated disease database when trained weights are unavailable so the feature degrades gracefully rather than failing outright. Yield and irrigation are each modelled with a scikit-learn ensemble — Random Forest for yield, Extra Trees for daily net irrigation need — built inside a Column Transformer pipeline that one-hot encodes crop and soil type and passes four live weather features through unchanged, with weather drawn at request time from the Open-Meteo forecast API. Fertilizer guidance and climate-event projection are implemented as heuristic engines grounded in ICAR-style NPK target ranges and per-state, per-crop vulnerability tables rather than as learned models, a choice made specifically to keep their recommendations traceable to known agronomic thresholds. The system is backed by a file-based SQLite schema covering users, sessions, farms, farm assignments, and an activity log, and the frontend is localized into six Indian languages. We describe the architecture, the methodology behind each of the seven functional modules, the role-based security model, and the engineering trade-offs made along the way, including the current reliance on a bootstrap-synthesized training set for the regression models pending real farm records. We close with the limitations this implies, and the roadmap items identified during development.

Keywords: Agronomic Decision Support; MobileNetV2; Random Forest; Extra Trees; Climate Impact Simulation; Role-Based Access Control; Open-Meteo; Precision Agriculture

I. INTRODUCTION

A farmer managing even a modest plot has to hold together several unrelated bodies of knowledge at once: plant pathology to recognize disease symptoms, agronomy to judge soil nutrient balance, hydrology to time irrigation against rainfall, and increasingly, an informal sense of how a changing climate is shifting all of the above. Extension officers traditionally bridged this gap, but their reach does not scale to the number of smallholders who need advice, particularly in remote regions. Mobile and web applications have tried to fill the space left behind, and a large literature now exists on applying deep learning to leaf images for disease detection and ensemble regression to structured soil and weather data for yield forecasting. What is less common is a single platform that treats these as facets of one ongoing decision rather than as separate products that a farmer must open one after another.

SasyaVedha was built to close that gap for an Indian farming context specifically. Rather than exposing abstract climate-scenario sliders, the system asks a farmer about something immediately recognizable — their own state, their irrigation source, and a climate event they have personally lived through, such as a late monsoon or a heatwave — and projects how that event would change yield, irrigation demand, and crop suitability on their own soil. The same design intent carries through the rest of the platform: the fertilizer module reports doses in kilograms per hectare and cost in rupees against known ICAR target ranges rather than an opaque score, and the irrigation module expresses its recommendation as a weekly schedule rather than a single number.

The platform is deliberately engineered to keep working when its more advanced components are not available. If the trained MobileNetV2 weights are missing from the backend, the disease-detection endpoint returns a curated disease-

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

database answer instead of an error, which keeps the rest of the application usable in development and in the event of a production model outage. This paper documents the system end to end: its architecture, the methodology behind its seven functional modules, its role-based access control and audit logging, the technology choices made at each layer, and the limitations we are aware of, most notably that the yield and irrigation regressors are currently trained on a bootstrap-synthesized dataset pending real farm records.

The remainder of this paper is organized as follows. Section II summarizes related work in vision- and tabular data-based agronomic prediction. Section III describes the overall system architecture. Section IV details the methodology of each functional module. Section V covers the security and access control model. Section VI describes the technology stack and representative API usage. Section VII discusses the current limitations of the system, and Section VIII concludes with the development of roadmap.

II. RELATED WORK

Deep learning for plant disease recognition from leaf imagery was popularized by Mohanty et al., who showed that a convolutional network trained on the Plant Village dataset could classify 26 diseases across 14 crop species with high accuracy directly from RGB photographs, without hand-engineered features [1]. SasyaVedha's disease module builds on the same dataset but substitutes a MobileNetV2 backbone [2], whose depth wise-separable convolutions make it practical to run inference without specialized server hardware, which matters for a system intended to run on modest, low-cost infrastructure in rural deployments.

For structured, tabular agricultural data, ensemble tree methods remain a strong and interpretable baseline. Random Forests [3] average many decorrelated decision trees to reduce variance without requiring the large, labelled datasets that deep networks typically need, which is why SasyaVedha uses a Random Forest for yield prediction. Extremely Randomized Trees (Extra Trees) [4] extend this idea by additionally randomizing the split thresholds during tree construction, trading a small amount of bias for lower variance and faster training; this property makes Extra Trees well suited to the irrigation module, where predictions are requested interactively and need to be both fast and stable.

Fertilizer recommendation and climate-impact projection are less often approached with machine learning at all, because the inputs that matter most — target nutrient ranges for a given crop, or how a particular state's agro-climatic zone responds to a drought — are already codified in agronomic reference literature such as the guidance published by the Indian Council of Agricultural Research. SasyaVedha treats these two modules as heuristic engines rather than learned models for exactly this reason: a transparent, rule-based recommendation that a farmer or agronomist can trace back to a known threshold is, for this use case, more useful than a black-box prediction with marginally higher accuracy.

III. SYSTEM ARCHITECTURE

SasyaVedha is organized as a conventional three-tier web application, shown in Fig. 1. A React 18 frontend communicates exclusively through an Axios client that attaches the user's bearer session token to every request; during development, the React dev server proxies all /api calls to the Flask backend on port 5000, which avoids any need for CORS configuration. The backend itself is a Flask application composed of two blueprints: auth_bp, which handles signup, login, logout, and session verification, and farm_bp, which exposes the agronomic prediction endpoints together with enterprise farm management and activity tracking. Authentication is session-token based rather than JWT-based: a token issued at login is checked against a sessions table on every protected request, and two decorators — login_required and roles_required — compose to enforce both authentication and role gating at the route level.

Underneath the API sits a model and service layer containing the MobileNetV2 disease classifier, the Random Forest and Extra Trees regression pipelines, and the two heuristic engines for fertilizer and climate simulation. This layer calls the Open-Meteo forecast API at request time for live temperature, humidity, rainfall, and wind data; no on-site hardware sensors are required anywhere in the system. The persistence layer is a single file-based SQLite database, seeded automatically on first startup, with a schema of five tables: users, sessions, farms, farm_assignments, and activity_log. The authors chose SQLite specifically for its zero-configuration startup during development and coursework evaluation, with the schema kept intentionally simple to ease a later migration to PostgreSQL for production use. The relationships between these five tables are shown in Fig. 2.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

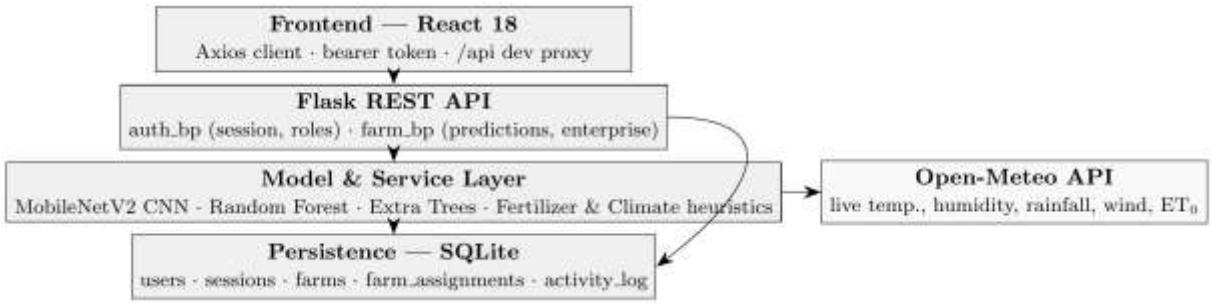


Fig. 1. SasyaVedha system architecture: the React frontend talks only to the Flask API, which calls the model/service layer and the Open-Meteo API, and persists state to a single SQLite database shared by authentication, farm, and audit tables.

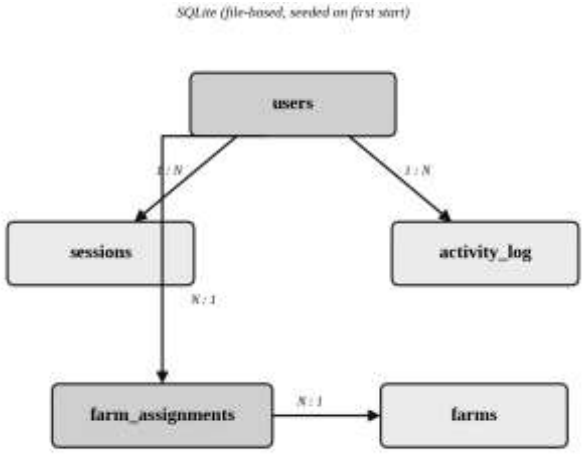


Fig. 2. SQLite persistence schema: sessions and activity_log belong to a user, while farm_assignments links users to farms.

The system runs in two modes. A Standard mode presents a single-farm view of the five prediction modules for an individual user, while an Enterprise mode adds multi-farm management, team assignment, and an audit-log overview for organizations managing several farms through one account. Table I summarizes the seven functional modules and the technique behind each.

TABLE I SYSTEM MODULES AND UNDERLYING APPROACH

Module	Approach	Purpose
Disease Detection	MobileNetV2 transfer learning on PlantVillage	Leaf-image classification with curated fallback guidance
Yield Prediction	Random Forest regression	Forecast yield per hectare from crop, soil, and live weather
Smart Irrigation	Extra Trees regression	Daily irrigation need from crop, soil, and weather balance
Fertilizer Advisory	Heuristic NPK/pH model	Soil-test-driven doses against ICAR target ranges
Climate Simulation	Heuristic state/crop/event model	Project climate-event impact on a farm to 2070
Activity Tracking	Append-only audit log	Cross-farm user actions and per-user timelines
Advanced Analytics	Aggregation over farm records	KPIs, charts, and auto-derived insights

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

IV. METHODOLOGY

A. Disease Detection

The disease module accepts a crop name and a base64-encoded leaf image and returns a predicted disease, a confidence score, severity, and separate fields for description, general measures, chemical treatment, and recommended products, each tagged with its type and intended use. Inference is performed by a MobileNetV2 CNN fine-tuned on the PlantVillage dataset, loaded on demand from backend/models/plant_disease_cnn.h5. When this weights file is absent, or inference raises an error, the endpoint does not fail the request; it instead falls back to a curated entry from a disease-guidance database bundled with the backend. This was a deliberate engineering decision rather than an oversight: it allows the application to be developed and demonstrated on machines without the TensorFlow model file installed, and it means a production outage of the model infrastructure degrades the feature rather than removing it. Every prediction, whether model-driven or fallback-driven, is written in the activity log. Fig. 3 summarizes this flow, including the fallback branch.

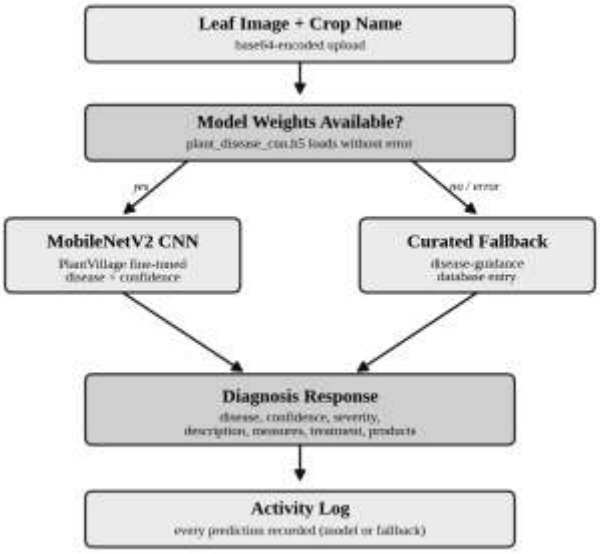


Fig. 3. Disease detection flow: MobileNetV2 inference with automatic fallback to the curated disease database.

B. Yield Prediction

Given crop, soil type, farm location, and area in hectares, this module returns predicted yield per hectare, total yield for the supplied area, a district-level average for comparison, the percentage deviation from that average, a grouped feature-importance breakdown, and a six-month trend projection. The model is a scikit-learn RandomForestRegressor wrapped in a ColumnTransformer pipeline: crop and soil type are one-hot encoded, while four live weather features — temperature, humidity, forecast rainfall, and reference evapotranspiration — are passed through unchanged. Weather values are fetched from Open-Meteo at request time rather than drawn from a static table, so two otherwise identical requests made a day apart can return different yield forecasts. Soil type is restricted to seven classes — Loamy, Clay, Sandy, Silty, Peaty, Chalky, and Black — and training data applies per-soil yield multipliers consistent with agronomic literature, for instance lowering yield for water-loving crops grown on sandy soil and raising it on black or peaty soil. Because one-hot encoding a seven-category feature would otherwise scatter its importance across seven separate bars, the feature-importance output re-aggregates the per-soil weights into a single “Soil type” entry before it reaches the chart. If a request omits soil type, the pipeline defaults to Loamy to preserve backward compatibility with earlier clients.

C. Smart Irrigation

The irrigation module takes the same crop, soil type, and location inputs together with current soil moisture, temperature, humidity, and forecast rainfall, and targets daily net irrigation need in millimeters using an ExtraTreesRegressor built on the identical soil-type-aware feature pipeline used for yield. From that daily figure the endpoint derives an urgency classification (high, medium, or low), the water volume required in liters for the given field area, a recommended time of day, a full weekly schedule, an evapotranspiration estimate, and a short water-saving tip. The weekly figure is computed as daily need multiplied by seven, minus forecast rainfall scaled by 0.45 to account for runoff and imperfect capture; the schedule marks an individual day for irrigation whenever forecast rainfall for that day falls below 60% of the daily need and the daily need itself exceeds 1.2 mm, which keeps the system from recommending irrigation on days where rain alone is expected to be sufficient.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

Two further details make the module easier to trust and to check. Because the weather features pass through the same pipeline used for yield, a change in the forecast can change the recommendation for the same field from one day to the next. The weekly arithmetic is also simple enough to verify by hand: for an illustrative daily need of 4 mm and 10 mm of forecast rainfall across the week, the weekly requirement is $7 \times 4 - 0.45 \times 10 = 23.5$ mm. The per-day rule works the same way. With that 4 mm daily need, the 60% threshold is 2.4 mm, so a day forecast to receive 1 mm of rain is marked for irrigation, whereas a day forecast to receive 5 mm is skipped. Converting a daily depth into a water volume rests on the unit identity that 1 mm over one hectare equals 10,000 litres. As with the yield model, the Extra Trees regressor is trained on the bootstrap-synthesized baseline discussed in Section VII, so the urgency class, schedule, and volume should be read as a demonstration of the method until real field records are available. Fig. 4 shows the shared pipeline behind the yield and irrigation models.

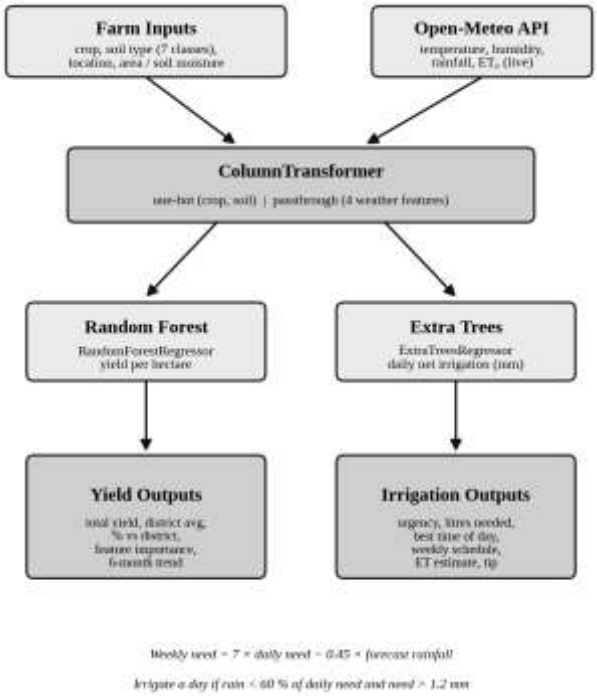


Fig. 4. Yield and irrigation prediction pipeline: one-hot encoded farm inputs plus live Open-Meteo weather feed a Random Forest and an Extra Trees regressor.

D. Fertilizer Advisory

This module accepts measured nitrogen, phosphorus, and potassium in kilograms per hectare, soil pH, crop, and area, and compares the submitted values against ICAR-style target NPK ranges for that crop and against a healthy pH band of 6.0 to 7.5. For each nutrient it returns a status of low, medium, or high together with a one-line note, and where a nutrient is deficient it recommends a product from a small built-in fertilizer database — Urea at 46% nitrogen, DAP at 46% P2O5, and Muriate of Potash at 60% K2O — with the exact kilogram-per-hectare dose needed to close the measured gap, a priority level, and an estimated cost in Indian rupees. Soils outside the 5.5 to 7.5 pH band additionally receive a lime (for acidic soil) or gypsum (for alkaline soil) correction scaled to area, and the response includes organic alternatives such as farmyard manure, vermicompost, neem-coated urea, and rock phosphate, alongside a three-stage application schedule: a basal dose at sowing, a first top-dressing at 30 days, and a second at 45 to 60 days. If every nutrient is already within its target range, the module returns a "None needed" recommendation rather than an empty list, and the total cost across all recommended products and the given area is summed into a single figure.

E. Climate Change Impact Simulation

Rather than exposing a farmer to an abstract IPCC emissions scenario, this module asks for inputs a farmer already understands: their Indian state and district, season, crop, soil type, irrigation source (rainfed, tube well, canal, rainfed with borewell, drip, or sprinkler), planting month, a time horizon, and a climate event drawn from a normal year, drought, heatwave, flood, unseasonal rain, late monsoon, or cold wave. A heuristic engine combines a per-state base temperature and monsoon-dependence value, a per-crop vulnerability table for each event type, a soil water-holding coefficient, and an irrigation-source buffering coefficient to project yield, irrigation need, and crop suitability year by year from 2025 out

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

to the chosen horizon, which can extend as far as 2070. The output includes three impact cards for yield, irrigation, and suitability with percentage deltas against today, a year-by-year line chart to the horizon, a bar chart comparing every climate event side by side for the same farm and crop with the selected event highlighted, and adaptation recommendations that vary with both the event and the irrigation source already in use. The deliberate design point here is that the same drought event produces very different projected outcomes for a rainfed smallholder than for a drip-irrigated farm on identical soil, which is the comparison a farmer actually wants to see. Fig. 5 illustrates how these inputs flow through the engine.

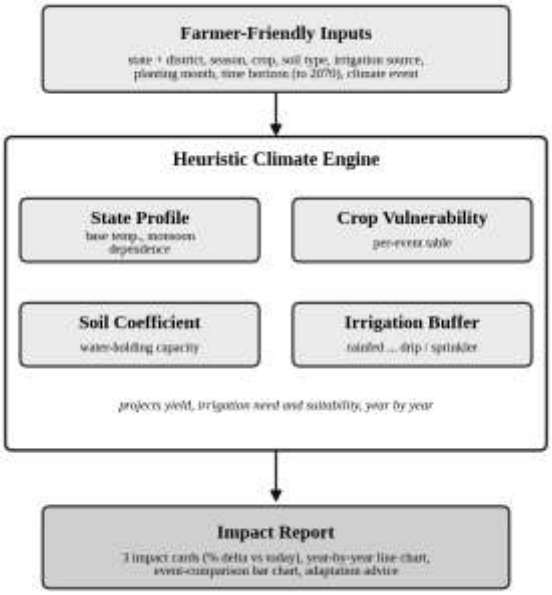


Fig. 5. Climate impact simulation: farmer-level inputs pass through four coefficient tables to produce the impact report.

F. Activity Tracking and Audit Log

Every authentication, assignment, and prediction route writes an append-only activity_log table through a shared log_activity helper that silently swallows its own failures, so a logging error never blocks a user-facing request. A separate farm_assignments table of records which users are linked to which farms or individual fields, subject to owner or admin authorization. From these two tables, the system derives a cross-tabular tracking overview joining users, farms, assignments, and per-user activity counts and last-active timestamps; a per-user activity timeline; and assignments list filterable by farm or user. Owners and admins can create or delete assignments; owners, admins, and managers can read the tracking overview, though managers see only the farms and users tied to their own assignments. The overview endpoint deliberately embeds shallow copies of each user and farm record into its assignment rows rather than references, a defensive choice that avoids the JSON encoder tripping over a circular reference once real assignment data exists in the database.

G. Advanced Analytics

This module takes no new input, consuming only the farm and enterprise-overview data already on file, and derives four key indicator cards (total farms, total fields, cultivated area, and number of distinct crops), a bar chart comparing farm areas, a donut chart of crop distribution, a farm breakdown table with crop tags and status pills, and a handful of plain-English insights such as the largest farm, the most commonly grown crop, and which farms need maintenance attention. It replaced an earlier version of the page that stacked four chart types including a radar visualization over placeholder data; the redesign favors readability over visual density specifically because the primary audience is non-technical farm managers rather than data analysts.

V. SECURITY AND ACCESS CONTROL

Authentication uses bearer session tokens rather than stateless JWTs, which means a logout can immediately invalidate a session rather than waiting for token expiry. Four roles are defined: user (the default for all new sign-ups), manager, owner, and admin. Owners manage non-privileged team accounts within their own organization; administrators may assign any role; and only owners and admins can reach Enterprise Settings or write farm assignments. Role enforcement

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

is implemented through two composable decorators, login_required and roles_required, rather than scattered manual checks, which keeps the authorization logic auditable at each route definition. A local administrator account is seeded automatically on first backend startup (admin@sasyavedha.com) with a default password that can be overridden by setting the SASYAVEDHA_ADMIN_PASSWORD environment variable before the first run, which is the only environment-level configuration the backend requires. Fig. 6 depicts the request check and the role hierarchy.

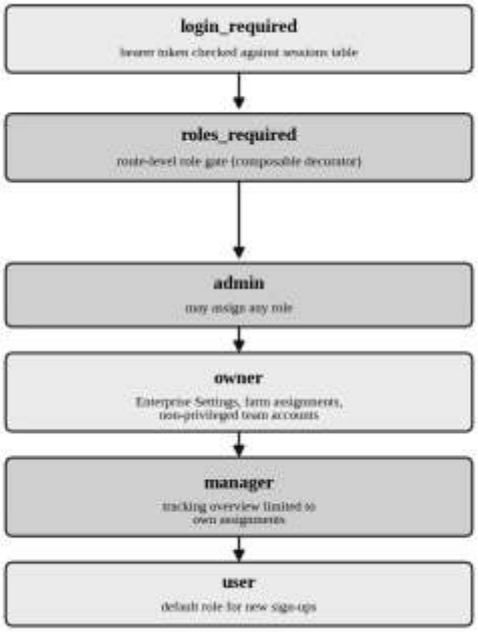


Fig. 6. Access-control flow: login_required and roles_required decorators gate requests by the four user roles.

VI. IMPLEMENTATION AND TECHNOLOGY STACK

Table II summarizes the technology choices at each layer. The frontend is built in React 18 with Recharts for data visualization and react-icons for iconography; the backend is a Python Flask application with Flask-CORS; machine learning components use scikit-learn for the Random Forest and Extra Trees pipelines and TensorFlow for the MobileNetV2 CNN; weather is sourced from Open-Meteo, which requires no API key and geocodes location strings server-side; and persistence is a single seeded SQLite file. The interface is localized into six Indian languages: English, Hindi, Marathi, Tamil, Telugu, and Kannada, covering both application chrome and crop names.

TABLE II TECHNOLOGY STACK BY LAYER

Layer	Technology
Frontend	React 18, Recharts, Axios, react-icons
Backend	Python Flask, Flask-CORS
Machine Learning	scikit-learn (Random Forest, Extra Trees), TensorFlow (MobileNetV2 CNN)
Weather Data	Open-Meteo forecast API (no key required)
Database	SQLite, file-based, seeded on first start
Authentication	Bearer session tokens, role-based decorators
Internationalization	English, Hindi, Marathi, Tamil, Telugu, Kannada

A representative call to the yield endpoint illustrates the shape of the API:

```
POST /api/predict-yield
{"crop":"Rice","soil_type":"Black",
 "location":"Bengaluru","area":2}
```

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

```
-> {"predicted_yield_per_ha":4.8,
   "total_yield":9.6,
   "district_average":4.0,
   "vs_district_pct":20.0, ...}
```

Setting the system up locally requires only Python 3.9 or later and Node.js 18 or later: the backend is started with pip install -r requirements.txt followed by python app.py, serving on port 5000, and the frontend with npm install followed by npm start, serving on port 3000 and proxying /api calls to the backend automatically. No database server, message queue, or external credential beyond the optional admin password override is needed to run the full system from end to end.

VII. DISCUSSION AND LIMITATIONS

Several of SasyaVedha's modules are intentionally heuristic rather than learned, and we think this is a strength rather than a shortcoming for the fertilizer and climate modules specifically, since a farmer or examiner can trace every recommended dose back to a stated NPK range or vulnerability coefficient. The same is not true of the yield and irrigation regressors: both are currently trained on a bootstrap baseline synthesized per crop from normal distributions around plausible agronomic ranges, with soil-type modifiers applied at training time, rather than on labelled records from real farms. The pipeline implementation does not need to change to accept real data — the ColumnTransformer structure is independent of where the training rows come from — but the predictions reported today should be read as a working demonstration of the method rather than as field-validated numbers and replacing the bootstrap set with real farm-level yield records is the single highest-priority item before any production deployment.

The disease-detection module carries a similar caveat in the opposite direction: it is the one component fully trained on a real, widely used dataset (PlantVillage), but the trained weights file is an optional artifact that may not be present on every deployment, in which case the system silently serves curated fallback guidance instead of a live model prediction. This is good for availability but means a user cannot always tell from the response alone, whether a given diagnosis came from the CNN or from the fallback database.

A separate limitation sits outside the machine learning components entirely: several enterprise-facing screens — Farm, Team, Inventory, Settings, and Report Generation, along with the Climate Simulation screen's scenario controls and the Export Report and Advanced Analytics summaries listed as frontend-only — currently persist their working data in browser storage rather than through a server endpoint. This keeps those features usable without backend changes, but it also means their demo data disappears if the browser storage is cleared, and it rules out any cross-device continuity for those specific screens until they are backed by the same SQLite-backed API the rest of the system uses.

Finally, the system has no hardware sensor layer at all: every environmental reading comes from the Open-Meteo forecast API, so its accuracy is bounded by how well that forecast matches conditions on a specific, possibly microclimate-affected, field.

VIII. CONCLUSION AND FUTURE WORK

SasyaVedha integrates five agronomic prediction modules and two management modules into a single Flask-and-React platform, built around role-based access control and an append-only audit trail, and localized into six Indian languages. The architecture keeps each predictive component (a CNN, two tree ensembles, and two heuristic engines) behind a common REST interface, which made it straightforward to add the Enterprise mode, multi-farm management, and activity tracking on top of the same five prediction endpoints originally built for a single-farm user. Several concrete next steps were identified during development rather than invented after the fact: a cross-module climate-event action plan that would combine a projected climate event with the relevant fertilizer, irrigation, and disease-scan advice for the same farm into one coherent output; a per-field yield-history chart built directly from entries already sitting in the activity log; completing CRUD on the enterprise farm resource with a farm-deletion endpoint; inventory restock alerts tied to scheduled fertilizer and pesticide applications; soil-test report upload with OCR so the fertilizer form can be pre-filled from a lab report photograph; Progressive Web App packaging so field scouting notes can be recorded offline in low-connectivity areas and synced later; and delivery of weekly farm summaries to field managers over WhatsApp. Alongside these, replacing the bootstrap synthetic training set behind the yield and irrigation models with real, labelled farm records remains the most important step toward a production-ready system.

 This work is licensed under a Creative Commons Attribution 4.0 International License



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.1510127

REFERENCES

[1] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection," *Frontiers in Plant Science*, vol. 7, art. 1419, 2016.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[3] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely Randomized Trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[5] Open-Meteo, "Open-Meteo Weather Forecast API," https://open-meteo.com/ (accessed 2026).

[6] Indian Council of Agricultural Research (ICAR), *General Crop Estimation Surveys and Fertilizer Recommendations for Field Crops*, ICAR Technical Bulletin Series.

[7] K. P. Ferentinos, "Deep Learning Models for Plant Disease Detection and Diagnosis," *Computers and Electronics in Agriculture*, vol. 145, pp. 311–318, 2018, doi: 10.1016/j.compag.2018.01.009.

[8] E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, "A Comparative Study of Fine-Tuning Deep Learning Models for Plant Disease Identification," *Computers and Electronics in Agriculture*, vol. 161, pp. 272–279, 2019, doi: 10.1016/j.compag.2018.03.032.

[9] S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, and D. Stefanovic, "Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification," *Computational Intelligence and Neuroscience*, vol. 2016, art. 3289801, 2016, doi: 10.1155/2016/3289801.

[10] A. Fuentes, S. Yoon, S. C. Kim, and D. S. Park, "A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition," *Sensors*, vol. 17, no. 9, art. 2022, 2017, doi: 10.3390/s17092022.

[11] A. Johannes, A. Picon, A. Alvarez-Gila, J. Echazarra, S. Rodriguez-Vaamonde, A. Díez Navajas, and A. Ortiz-Barredo, "Automatic Plant Disease Diagnosis Using Mobile Capture Devices, Applied on a Wheat Use Case," *Computers and Electronics in Agriculture*, vol. 138, pp. 200–209, 2017, doi: 10.1016/j.compag.2017.04.013.

[12] G. Wang, Y. Sun, and J. Wang, "Automatic Image-Based Plant Disease Severity Estimation Using Deep Learning," *Computational Intelligence and Neuroscience*, vol. 2017, art. 2917536, 2017, doi: 10.1155/2017/2917536.

[13] T. van Klompenburg, A. Kassahun, and C. Catal, "Crop Yield Prediction Using Machine Learning: A Systematic Literature Review," *Computers and Electronics in Agriculture*, vol. 177, art. 105709, 2020, doi: 10.1016/j.compag.2020.105709.

 This work is licensed under a Creative Commons Attribution 4.0 International License