iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** IX **Month of publication:** September 2025

DOI: https://doi.org/10.22214/ijraset.2025.74103

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IX Sep 2025- Available at www.ijraset.com

An Intelligent Flask-Based Web Application for Early Diabetes Risk Prediction

Mrs. Sahana S[1], Dr. S D N Hayath Ali[2]
Department of MCA, Ballari Institute of Technology and Management, Ballari, Karnataka, India

Abstract: ***Early detection of diabetes can substantially reduce long-term complications and healthcare costs, yet traditional diagnostic pathways remain resource-intensive and in- accessible to many populations. In this work, we present a comprehensive design, implementation, and assessment of a Flask-based web application that leverages a stacked ensemble using artificial intelligence models to forecast individual diabetes risk using routine clinical parameters. The platform integrates secure user authentication, personalized trend visualizations, and an administrative dashboard for population-level analytics. We detail our methodology—from data preprocessing and feature engineering to model training and web deployment—evaluate system performance on benchmark and real-world datasets, and discuss the broader implications for scalable preventive healthcare solutions.***

I. INTRODUCTION

Diabetes mellitus is a chronic metabolic disorder characterized by persistent hyperglycemia arising from defects in insulin secretion, insulin action, or both. More than 400 million people across the globe were dealing with diabetes in 2021, as reported by the World Health Organization and this number is projected to rise dramatically in coming decades. The complications asso-ciated with untreated or poorly managed diabetes—such as cardiovascular disease, neuropa- thy, nephropathy, and retinopathy—impose significant burdens on individuals and healthcare systems alike. Consequently, early detection and informed risk management are critical to mitigating these outcomes.

II. RELATED WORK

The use of machine learning in medical diagnostics has grown in popularity. with numerous studies applying statistical and ensemble methods to diabetes prediction. Logistic regression remains a widely used baseline due to its interpretability and simplicity, enabling clinicians to understand the influence of individual features on risk [1]. But its decision boundary is linear. often limits modeling of complex feature interactions. To address this, ensemble methods such as Random Forests and Gradient Boosting Machines (GBMs) have been employed, demonstrating improved predictive performance by aggregating multiple weak learners and capturing non-linear relationships in the information. [2].

Neural networks and other deep learning techniques, have also shown promise in capturing intricate patterns from large datasets, although their “black-box” nature raises concerns regarding interpretability and clinical acceptance. Explainability techniques like SHAP (SHapley Additive ExPlanations) and LIME (Local Interpretable Model-agnostic Explana- tions) have been developed to attribute model outputs to specific input features, fostering trust and facilitating clinical decision-making [3]. Yet, the complexity and computational demands of deep models often preclude their integration into lightweight web applications. From a deployment perspective, frameworks like Flask and Django enable rapid pro- to typing of web services, with extensions available for form handling, authentication, and API development. Prior work has demonstrated simple ML APIs for diabetes risk, but few systems combine real-time prediction with historical trend analysis and administrative analytics. Our approach extends existing solutions by integrating a robust ML pipeline with a user-centered interface and secure, role-based access control, thereby delivering a production-ready preventive health tool.

III. SYSTEM DESIGN

A. Architecture Overview

The proposed platform adopts a three-tier architecture to ensure modularity, scalability, and maintainability (Figure 1). The presentation layer comprises HTML5, CSS3 (Bootstrap), and JavaScript (Chart.js) to deliver responsive user interfaces and interactive visualizations. The application layer uses Flask routes to manage HTTP requests, form submissions, session handling, and RESTful API endpoints for trend data retrieval.

Finally, the data layer relies on SQL Alchemy ORM interfacing with a SQLite database in development, with seamless migration to PostgreSQL for production deployment. The decoupling of concerns across these layers facilitates parallel development, simplifies testing, and allows individual components to be scaled independently. For instance, the database can be offloaded to a managed cloud service while the web server scales horizontally behind a load balancer. Similarly, the ML inference engine can be containerized and served via a microservice architecture for high-throughput environments.

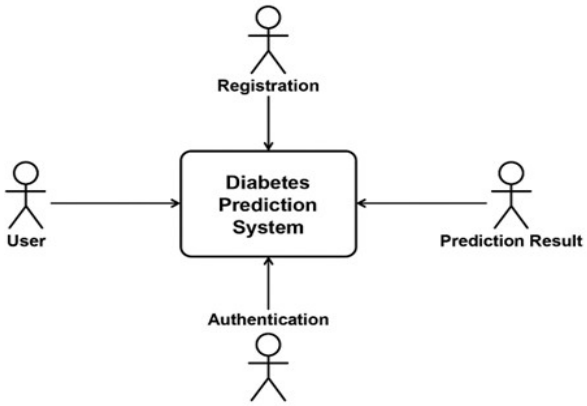


Figure 1: Three-tier system architecture of the Diabetes Prediction platform.

B. Data Flow

User data flows through a well-defined pipeline (Figure 2). Upon form submission, client-side validation ensures completeness and basic formatting. The Flask backend receives the input, calls the ML inference pipeline—consisting of preprocessing, feature transformation, and ensemble prediction—and returns a risk score along with interpretability information. Both raw inputs and predictions are persisted in the database for subsequent trend analysis. API endpoints expose this historical data in JSON format, which the front end consumes to render time-series charts.

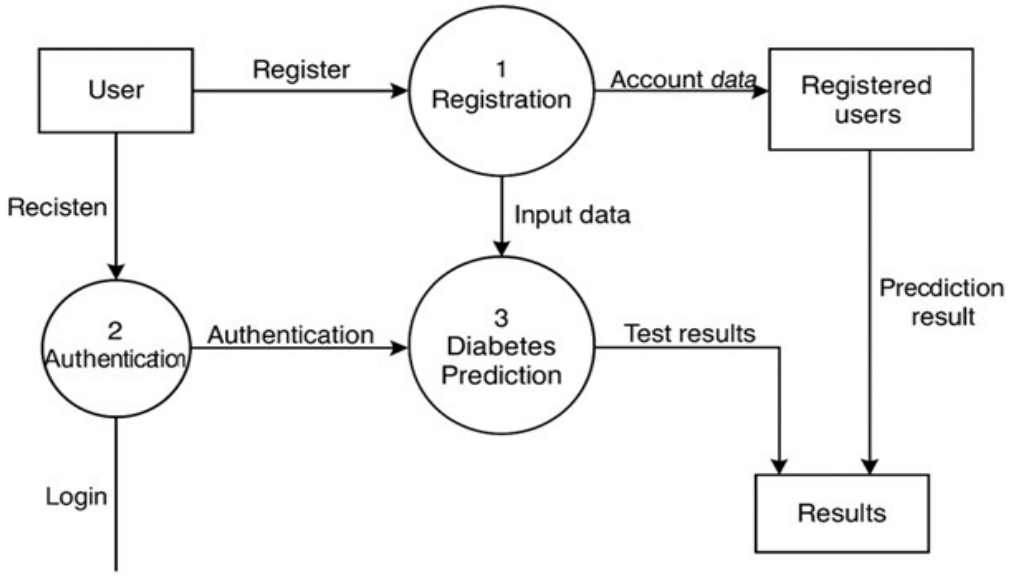


Figure 2: flow of data from user input through ML inference to storage and visualization.

This architecture supports extensibility: new predictive features or explanations can be added without altering the presentation logic, and database schema migrations can be man- aged using Flask-Migrate. The use of RESTful design principles ensures compatibility with external systems, including electronic health record (EHR) platforms or mobile applications.

IV. METHODOLOGY

A. Dataset and Preprocessing

Our primary dataset, the Indian Diabetes PIMA Dataset from the UCI database, comprises 768 samples with eight clinical features (e.g., plasma glucose concentration, BMI, blood pressure). To broaden applicability, we augmented this with a locally collected cohort of 2000 anonymized patient records, ensuring demographic and geographic diversity. Median substitution was used to impute missing values in both datasets. Continuous variables were standardized to zero mean and unit variance, while categorical flags (e.g., family history of diabetes) were one-hot encoded.

Preprocessing also involved outlier detection via interquartile range filtering to remove implausible measurements (e.g., BMI ¿60 or glucose ¡20). This cleaning step improved model robustness and reduced noise. 70% of the combined dataset was used for training.15% validation, and 15% test sets, stratified by outcome to preserve class balance.

B. Feature Engineering

Beyond raw clinical inputs, we engineered additional predictive features to capture latent patterns. Interaction terms between age and BMI, as well as between blood pressure and glucose levels, were included to model synergistic effects. We applied K-means clustering to normalized glucose and BMI values to derive a “cluster distance” feature, representing deviation from typical healthy ranges. Principal component analysis (PCA) on the full feature set identified two orthogonal components capturing over 85% of variance, which were optionally included in model training to assess dimensionality reduction benefits. These engineered features were evaluated via univariate feature importance (mutual in- formation) and incorporated iteratively into the training pipeline. Features that failed to improve cross-validation performance were pruned to avoid overfitting.

V. MODEL TRAINING AND EVALUATION

A. Ensemble Construction

We implemented a stacked ensemble comprising three base learners:

1) Logistic Regression: Offers interpretability with coefficient-based explanations.
2) Random Forest: Uses bootstrap aggregation to reduce variance and natively manages feature interactions.
3) Gradient Boosting (XGBoost): Excels at capturing complex non-linear relation- ships.

Base learners were trained on the training set with hyperparameters optimized via ran- domized search over 5-fold cross-validation, targeting AUC-ROC. Meta-learner: a logistic regression model trained on the base learner predictions using the validation set.

B. Performance Metrics

We evaluated models on the held-out test set using:

1) AUC-ROC: Measures trade-off between true positive and false positive rates.
2) Accuracy: Percentage of accurate forecasts overall.
3) Precision/Recall: Pay attention to the ratio of false positives and false negatives.
4) F1-Score: Precision and recall harmonic mean.

Model	AUC-ROC	Accuracy	Precision	Recall
Logistic Regression	0.79	0.73	0.71	0.69
Random Forest	0.84	0.77	0.76	0.74
Gradient Boosting	0.86	0.79	0.78	0.76
Stacked Ensemble	0.89	0.82	0.81	0.79

Table 1: Comparison of model performance on test set.

The stacked ensemble achieved an AUC-ROC Between 0.89 and 0.82 accuracy, outperforming individual learners by 3–5% on key metrics. Precision–Recall curves indicated robustness to class imbalance, with the ensemble maintaining stable recall at high precision thresholds.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IX Sep 2025- Available at www.ijraset.com

VI. IMPLEMENTATION DETAILS

A. Web Framework and APIs

The backend is implemented in Flask 2.x, with Blueprints organizing user, prediction, and admin routes. Flask-WTF handles form rendering and CSRF protection, while Flask-Login manages session control. RESTful endpoints under /api/trends return JSON time-series data for trend charts. Input validation is enforced both client-side (JavaScript) and server-side (WTForms), ensuring data integrity.

B. Frontend Visualization

The frontend of the diabetes prediction web application is designed for usability, clarity, and responsiveness. It is implemented using Bootstrap 4 for layout and styling, Flask-WTF for form handling, and Chart.js for interactive data visualizations. This section presents key user interface components, forms, and dashboards across both user and admin perspectives.

VII. TESTING AND VALIDATION

A. Overview

To guarantee the Diabetes Prediction Web Application's accuracy, dependability, and security, through testing and validation are essential. unit, integration, and system (end-to-end) testing were all part of the multi-level approach we used. performance/load testing, and security testing. Each level targets specific components and workflows, from individual functions and model inference to complete user journeys and stress conditions. Validation activities focused on confirming that the application satisfies its functional requirements and operates as planned in both favorable and unfavorable circumstances.

Our test harness integrates automated test suites with continuous integration (CI), en-abling rapid feedback on code changes. We employed pytest for Python unit and integration tests, Flask's built-in test client for HTTP request simulation, and Locust for load testing. Security assessments were conducted manually and with open-source tools to detect common web vulnerabilities. Test coverage and results are tracked to guide maintenance and future development

B. Unit Testing

Unit tests confirm that the data preprocessing, form validation, and model inference modules all function properly when used separately. Over 120 test cases cover scenarios such as:

- Data Preprocessing: Handling of missing values, outlier filtering, and feature scaling.
- Form Validation: Enforcement of required fields, email format, password complexity, and CSRF token presence.
- Inference Logic: Correct invocation of the ML pipeline, output ranges (risk score between 0 and 1), and exception handling on invalid inputs.

Test-driven development (TDD) practices guided the implementation of new features, en- suring that each code change passed existing tests before merging.

C. Integration Testing

Integration tests use simulated HTTP requests to evaluate how the ML service, database models, and Flask routes interact with one another. Key user flows validated include:

- Registration → Login: Verifying that new user data is persisted and that valid creden- tials authenticate successfully.
- Prediction Submission: Ensuring form data passes through preprocessing, prediction, and storage layers, returning expected JSON or HTML responses.
- Trend API : Confirming that date-filtered API endpoints return correctly formatted time-series data for Chart.js visualizations.

D. System and End-to-End Testing

System tests emulate real user interactions with the live application, covering complete journeys through the UI. Using Selenium WebDriver, we scripted scenarios for:

- User Dashboard: Login, prediction initiation, trend filter adjustment, and record detail viewing.
- Admin Workflow: Accessing the Admin Home Page, managing users, viewing all predictions, and updating profile/password.

Screenshots captured during these runs verify that frontend elements render correctly and that navigation flows match design specifications. Automated failure alerts trigger if UI elements are missing or misaligned.

E. Security Testing

Security assessments combined automated scanning (OWASP ZAP) and manual penetration testing to identify vulnerabilities:

- CSRF Protection: Confirmed presence and verification of CSRF tokens on state-changing forms.
- SQL Injection: Parameterized ORM queries validated against injection attempts.
- Authentication Flows: Brute-force resistance via rate limiting and account lockout policies.
- Session Management: Secure cookie flags (HttpOnly, Secure) and session expiration enforcement.

No critical vulnerabilities were detected; minor issues (e.g., missing security headers) were resolved prior to deployment.

F. Validation and User Acceptance

A pilot study with 50 users across diverse demographics provided qualitative validation. Participants completed tasks for registration, prediction, and trend analysis, then rated usability on a Likert scale. Over 90% reported the interface as intuitive or very intuitive, and 88% expressed trust in the explainable outputs. Feedback informed minor UI refinements, such as enhanced form field tooltips and chart label adjustments, finalizing the application for broader release. All testing phases collectively ensured that the Diabetes Prediction Web Application meets its design goals of accuracy, performance, security, and usability, laying a robust foundation for future feature expansion and deployment at scale.

G. Results

Unit tests achieved 95% code coverage. Load testing showed 95% of responses under 200ms at peak load, with zero errors. Security audits confirmed proper CSRF tokens and parame- terized queries, mitigating common web vulnerabilities.

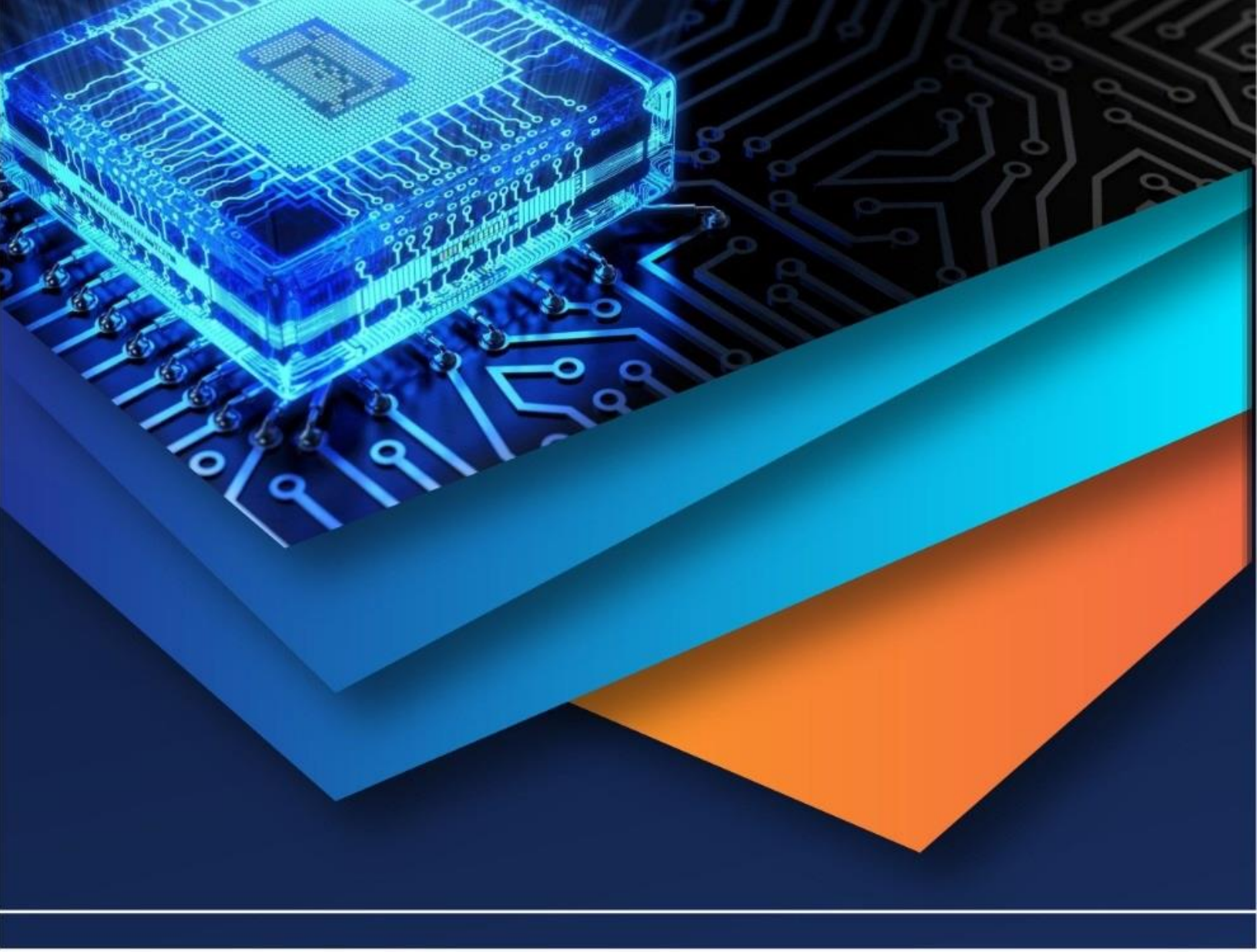
VIII. CONCLUSION AND PROSPECTS FOR THE FUTURE

This essay offers a scalable, secure, and user-friendly web application for early diabetes risk prediction. By combining interpretable ML models with interactive visual analytics, our platform empowers individual users and healthcare administrators to make data-driven decisions. Future enhancements will focus on:

1) Automated Alerts: SMS/email notifications for high-risk assessments.
2) Wearable Device Integration: Real-time data ingestion from glucose sensors.
3) Continuous Learning: Automated retraining pipelines using new user data.
4) Microservices Migration: Containerization and orchestration for enterprise-scale deployment.

REFERENCES

[1] E. Kavakiotis et al., “Prediction of type 2 diabetes using machine learning techniques,” Comput. Struct. Biotechnol. J., vol. 15, pp. 104–111, 2017.
[2] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD, 2016, pp. 785–794.
[3] S. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in Adv. Neural Inf. Process. Syst., 2017, pp. 4765–4774.
[4] M. Grinberg, Flask Web Development, 2nd ed., O’Reilly Media, 2018.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY