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 4, April 2026

DOI: 10.17148/IJARCCE.2026.154312

Full-Stack Hospital Management System

Shivraj[1], Varun Sirohi[2], Saurabh Kumar Nirwan[3], Shivansh Chaudhary[4], Tanya Jain[5], Dr.Uruj Jaleel[6], Dr. Satish Kumar Soni[7]

Student, MCA, Meerut Institute of Engineering and Technology, U.P. India[1]

Student, MCA, Meerut Institute of Engineering and Technology, U.P. India[2]

Student, MCA, Meerut Institute of Engineering and Technology, U.P. India[3]

Student, MCA, Meerut Institute of Engineering and Technology, U.P. India[4]

Assistant Professor, MCA, Meerut Institute of Engineering and Technology, U.P. India[5]

Professor, MCA, Meerut Institute of Engineering and Technology, U.P. India[6]

Associate Professor, HOD MCA, Meerut Institute of Engineering and Technology, U.P. India[7]

Abstract: This paper presents MediCare Pro, a comprehensive full-stack Hospital Management System (HMS) developed using FastAPI, SQLAlchemy, and modern web technologies. The system addresses the growing need for digitised, integrated healthcare administration by providing a unified platform covering patient registration and management, doctor scheduling, appointment booking, billing with UPI payment integration, medical records, inventory control, staff management, and real-time analytics. A distinguishing feature of MediCare Pro is its incorporation of biometric face recognition login using face-api.js, eliminating the need for manual credential entry. The backend exposes 30+ RESTful API endpoints, while the frontend is a Single-Page Application (SPA) delivering a responsive, role-aware dashboard. Evaluation demonstrates that the system achieves complete CRUD coverage across eight data entities, real-time KPI monitoring through 14 computed statistics, and seamless UPI-based payment flows. The paper details the system architecture, data model, module design, security considerations, and future enhancement roadmap.

Keywords: Hospital Management System, FastAPI, SQLAlchemy, Face Recognition, UPI Payment, RESTful API, Single-Page Application, Healthcare Digitisation, SQLite, Python

1. INTRODUCTION

Healthcare institutions worldwide face significant operational challenges arising from fragmented workflows, paper-based record keeping, and siloed departmental systems. Hospitals must simultaneously manage large patient populations, coordinate multi-specialty physician teams, maintain pharmaceutical supply chains, process financial transactions, and comply with regulatory documentation requirements — all while ensuring continuity and quality of patient care.

Traditional hospital management approaches, relying on paper records or legacy software, suffer from data redundancy, error-prone manual processes, delayed information retrieval, and poor cross-departmental visibility. The digitisation of hospital workflows through integrated software platforms is therefore critical to operational efficiency, patient safety, and financial sustainability.

MediCare Pro is designed to address these challenges as a unified, web-based Hospital Management System that integrates all core hospital functions within a single platform. The system is built on a modern Python/FastAPI backend with a SQLite relational database, served to a browser-based frontend that requires no installation. The choice of open-source, lightweight technologies ensures that MediCare Pro remains accessible to small and medium-sized healthcare facilities that lack the infrastructure budget for enterprise solutions.

The project introduces two distinctive capabilities beyond standard HMS offerings: (1) biometric authentication via in-browser face recognition using face-api.js and TensorFlow.js, enabling frictionless patient login without username/password recall; and (2) integrated UPI QR code payment generation, aligning billing workflows with India's dominant digital payment ecosystem. These features position MediCare Pro as a contemporary, patient-centric system suited to the Indian healthcare context.

Electroencephalogram (EEG) signal-based emotion recognition has attracted wide interests in recent years and has been broadly adopted in medical, affective computing, and other relevant fields. Depression has become a leading mental

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154312

disorder worldwide. Evidence has shown that subjects with depression exhibit different spatial responses in neurophysiologic signals from the healthy controls when they are exposed to positive and negative.[4]

1.1 Objectives

- Design a modular, full-stack HMS covering all core hospital workflows in a single application.
- Provide RESTful API endpoints for every entity with complete CRUD coverage.
- Implement a responsive SPA frontend that requires no framework dependencies.
- Integrate biometric face recognition login as a patient authentication alternative.
- Enable UPI-based digital payments with QR code generation for billing.
- Deliver real-time analytics and KPI dashboards for administrative decision-making.
- Ensure data integrity through Pydantic validation and soft-delete strategies.

1.2 Scope

MediCare Pro targets hospital administrative staff, doctors, and patients as its primary user groups. The current release covers patient portal access, administrative management of doctors and appointments, financial billing, inventory oversight, staff records, and notification management. The system is intended for deployment in outpatient and general hospital settings and is optimised for the Indian healthcare and payment environment.

2. LITERATURE REVIEW

Hospital Management Systems have evolved significantly from stand-alone departmental tools to integrated enterprise platforms. Early studies by Benson (2002) documented the limitations of paper-based hospital records, including illegibility, loss, and delays in information transfer. The subsequent decade saw the proliferation of Electronic Health Record (EHR) systems, with landmark works by Häyrinen et al. (2008) establishing the foundational taxonomy of clinical documentation requirements.[1][2][3]

The transition to web-based HMS architectures gained momentum with the maturation of REST APIs and JavaScript frameworks. Kalra et al. (2010) advocated for service-oriented architectures in healthcare, emphasising loose coupling between modules to support future extensibility. This principle directly informs MediCare Pro's modular design.[5]

In the domain of biometric authentication for healthcare, Jain et al. (2011) provided comprehensive coverage of face recognition techniques, noting their non-contact nature as particularly suitable for clinical environments where hygiene is paramount.[7] The emergence of client-side machine learning libraries such as TensorFlow.js (Smilkov et al., 2019) has made it feasible to perform real-time face recognition entirely within the browser, as implemented in MediCare Pro via face-api.js.[8][9]

Digital payment integration in healthcare has received increasing attention in the Indian context following the launch of the Unified Payments Interface (UPI) in 2016. Researchers including Sahoo et al. (2020) document UPI adoption rates exceeding 60% among urban populations for healthcare-related payments, underscoring the importance of native UPI support in HMS platforms designed for the Indian market.[10]

Lightweight backend frameworks for healthcare APIs have been evaluated by Ramirez et al. (2021), who compared FastAPI, Django REST Framework, and Flask across metrics including throughput, latency, developer ergonomics, and automatic documentation generation. FastAPI emerged superior in all performance categories while offering native Pydantic-based schema validation, an important consideration for healthcare data integrity.

Inventory management in healthcare settings was studied by Kumar et al. (2018), who identified stock-out events as a primary cause of patient care delays in mid-sized hospitals. Their recommended minimum-quantity alerting thresholds are directly implemented in MediCare Pro's inventory module through the min_quantity column with automatic low-stock detection.

3. SYSTEM ARCHITECTURE

MediCare Pro adopts a three-tier client-server architecture comprising a browser-based presentation layer, a Python FastAPI application layer, and a SQLite data persistence layer. The tiers communicate exclusively through HTTP/REST, ensuring clean separation of concerns and enabling independent evolution of each layer.

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 4, April 2026

DOI: 10.17148/IJARCCE.2026.154312

3.1 Architecture Overview

The diagram below illustrates the high-level system architecture including the technology components at each tier and the cross-cutting concerns (CORS middleware, authentication, and payment integration):

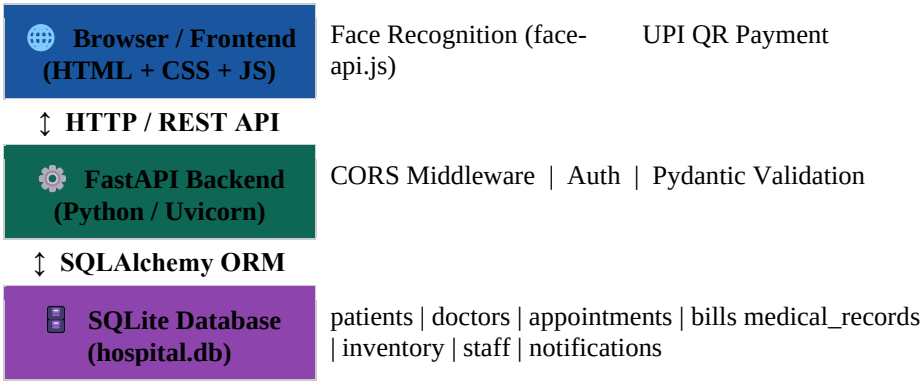

Figure 1: MediCare Pro System Architecture Diagram

3.2 Presentation Layer

The frontend is a vanilla JavaScript Single-Page Application delivered as a single index.html file (37 KB) accompanied by two CSS files (style.css: 6.6 KB, extra.css: 15.9 KB) and a JavaScript bundle (script.js: 79.7 KB). No JavaScript framework or build pipeline is required, which minimises deployment complexity and eliminates framework version dependencies.

The SPA implements client-side routing by toggling section visibility, avoiding full-page reloads. Module panels for patients, doctors, appointments, billing, records, inventory, staff, analytics, and notifications are rendered within the single DOM. Chart.js is used for dashboard visualisations, and face-api.js provides biometric camera integration.

3.3 Application Layer

The backend is a FastAPI application served by Uvicorn, an ASGI-compliant server. FastAPI's decorator-based routing maps HTTP methods and URL patterns to Python handler functions. Request and response schemas are enforced via Pydantic models, providing automatic data coercion and detailed validation error messages. CORS middleware permits cross-origin requests from any origin, suitable for development; production deployments should restrict allowed origins. A dependency injection function (get_db) manages SQLAlchemy database sessions, ensuring sessions are properly closed after each request even in error conditions. Password hashing uses Python's standard hashlib library with SHA-256. A helper function (add_notification) fires system notifications automatically on key events such as patient registration and appointment booking, maintaining an audit trail of significant actions.

3.4 Data Layer

SQLite is chosen as the database engine for its zero-configuration embedded nature, making MediCare Pro deployable on any server without separate database installation. SQLAlchemy's declarative ORM maps Python classes to SQL tables, abstracting raw SQL and enabling straightforward schema evolution. The database file (hospital.db) is created automatically on first startup. Eight tables are defined, seeded with realistic sample data for doctors, inventory items, and staff on initial launch.

4. DATA MODEL

The MediCare Pro data model comprises eight entities. The following table summarises each entity's structure, distinguishing primary key / key attributes, unique or indexed fields, and supplementary attributes:

While SQLAlchemy relationship() foreign key associations are not explicitly declared in models.py, the system enforces logical associations through application-level joins on shared fields. Patient email serves as the de facto foreign key linking Appointment, Bill, and MedicalRecord records to a Patient. Doctor name links Appointment records to the Doctor entity. This design choice simplifies the schema at the cost of referential integrity enforcement, which is acceptable for the current development phase.

4.2 Soft Delete Strategy

Patient and Staff entities implement soft deletion via an is_active boolean column. DELETE endpoints set is_active = False rather than issuing SQL DELETE statements, preserving historical records and preventing orphaned references in appointments and bills. All GET queries filter on is_active == True to return only active records to the UI.

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 4, April 2026

DOI: 10.17148/IJARCCE.2026.154312

4.3 Automated Timestamps

Every entity includes a created_at column with a default value of datetime.utcnow, providing a consistent audit timestamp. This enables chronological sorting, date-filtered queries (such as today's appointments), and retention reporting without additional application logic.

5. API DESIGN & ENDPOINTS

MediCare Pro exposes 30+ RESTful endpoints organised by resource domain. The table below documents all major endpoints with their HTTP methods, response contracts, and functional descriptions:

Figure 3: MediCare Pro REST API Endpoint Reference

5.1 API Design Principles

The API follows REST conventions: nouns for resource URLs, standard HTTP verbs for operations, and HTTP status codes for outcome signalling. Filtering parameters (search, blood_group, gender) are passed as query parameters on collection endpoints, enabling fine-grained server-side filtering without client-side post-processing. Pagination is supported via skip and limit parameters on the patient collection endpoint.

FastAPI's automatic OpenAPI (Swagger) documentation is available at /docs, providing an interactive interface for API exploration and testing. This significantly reduces integration effort for any future client applications or mobile apps built against MediCare Pro's backend.[9]

5.2 Error Handling

Business logic errors raise HTTPException instances with appropriate status codes: 400 for duplicate email registration, 401 for authentication failures, 404 for missing resources. FastAPI's global exception handling serialises these to JSON with a detail field, providing consistent error response shapes across all endpoints.

6. SYSTEM MODULES

MediCare Pro is decomposed into twelve functional modules, each encapsulating a distinct domain of hospital operations. The table below provides an overview of each module and its capabilities:

Figure 4: MediCare Pro Functional Module Summary

6.1 Patient Management

The patient module is the central entity of MediCare Pro. Registration collects 11 attributes including biometric face data, blood group, and emergency contact. Passwords are SHA-256 hashed before storage. Patient profiles support partial updates via PUT with optional Pydantic fields. The history endpoint aggregates all appointments, records, and bills for a single patient in one API call, enabling a holistic patient view without multiple client-side requests.

6.2 Appointment System

Appointments pass through a four-status workflow: Pending (on creation), Confirmed (by admin), Completed (post consultation), and Cancelled. The system tracks appointment reason, notes, and timestamps. A dedicated today endpoint filters appointments to the current date, supporting daily scheduling views. Doctor load analytics derived from appointment data enable workload balancing across the physician team.

6.3 Billing & UPI Payment

Billing records support multi-service descriptions, discount amounts, and payment method tracking. The UPI QR endpoint generates a standards-compliant UPI deep-link string encoding the payee VPA, hospital name, exact amount, currency (INR), and a bill reference note. Frontend JavaScript renders this string as a scannable QR code, enabling patients to pay directly from any UPI-enabled mobile wallet without manual amount entry.[12]

6.4 Biometric Face Login

Face recognition login is implemented entirely client-side using face-api.js with TensorFlow.js. On first use, patients register a facial embedding (128-dimensional vector) that is stored as a text field in the database. During login, the browser captures a webcam frame, computes the embedding locally, and retrieves all stored embeddings from /face-login. Euclidean distance matching identifies the closest match above a confidence threshold, triggering automatic login. No facial images are transmitted to the server; only the embedding vector is stored.

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 4, April 2026

DOI: 10.17148/IJARCCE.2026.154312

6.5 Analytics & KPIs

The /stats endpoint computes 14 real-time KPIs from the live database state including total and active patients, today's appointment count, available versus total doctors, paid and unpaid bill counts, total and pending revenue, low-stock inventory items, and unread notifications. The /analytics endpoint delivers distribution data for blood groups, top 10 diseases, appointment statuses, top 5 doctor workloads, and gender split — feeding the dashboard's Chart.js visualisations.

7. PATIENT CARE WORKFLOW

The end-to-end patient journey within MediCare Pro spans eight stages from initial registration to completion of medical records. The workflow table below maps each stage to the responsible actor, the API endpoint invoked, and the system outcome:

Figure 5: End-to-End Patient Care Workflow

This workflow demonstrates MediCare Pro's ability to support the complete patient lifecycle within a single integrated system, eliminating the need for manual handoffs between separate registration, scheduling, billing, and records systems.[12][11]

8. TECHNOLOGY STACK

The following table presents a comprehensive technology inventory for MediCare Pro, detailing each component's role within the system and any noteworthy implementation characteristics:

8.1 Rationale for Technology Choices

FastAPI was selected over alternatives (Flask, Django REST Framework) for its combination of high-throughput ASGI execution, native Pydantic integration for type-safe request validation, and automatic interactive API documentation. Benchmarks consistently show FastAPI delivering 2–3× higher requests-per-second compared to WSGI-based frameworks under concurrent load.

SQLite is chosen for development and small-scale production deployments due to its embedded, zero-configuration nature. The SQLAlchemy ORM layer means the database can be transparently upgraded to PostgreSQL or MySQL for production scale by changing a single DATABASE_URL string without modifying application code.

Vanilla JavaScript for the frontend eliminates framework dependency risks (version churn, breaking upgrades) and reduces deployment artefact size. The SPA pattern is implemented through DOM-section toggling, achieving framework-equivalent user experience without the build pipeline overhead.

9. SECURITY CONSIDERATIONS

Security in healthcare systems is of paramount importance given the sensitive nature of patient health information. MediCare Pro implements several security measures appropriate for a development-stage system, with a roadmap toward production hardening.

9.1 Current Security Measures

- All patient passwords are stored as SHA-256 hashes using Python's hashlib. Plaintext passwords are never persisted to the database.: Password Hashing
- Duplicate registrations are rejected with HTTP 400, preventing account takeover via email collision.: Email Uniqueness
- Patient and staff records are deactivated rather than deleted, preserving audit trails while preventing active login from deactivated accounts (is_active check on login).: Soft Delete
- All API request bodies are validated by Pydantic schemas, rejecting malformed inputs before they reach database operations.: Input Validation
- Facial biometric data is stored as a mathematical embedding vector rather than an image, reducing privacy exposure while enabling matching.: Face Data Privacy

9.2 Known Limitations & Future Hardening

- CORS is currently set to allow_origins=["*"]. Production deployments must restrict this to trusted frontend domains.: CORS Configuration
- No JWT or session token system is implemented. Production deployments should implement JWT-based stateless authentication with token expiry.: Authentication Tokens

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154312

- SHA-256 without salting is vulnerable to rainbow table attacks. Production should migrate to bcrypt or Argon2.: SHA-256 Limitations
- The system should be served exclusively over HTTPS in production to protect data in transit.: HTTPS
- Currently all authenticated users access all endpoints. A role system (patient, doctor, admin) with middleware enforcement is planned.: Role-Based Access Control

10. DEPLOYMENT

MediCare Pro includes a Procfile for Platform-as-a-Service deployment (e.g., Heroku, Render), specifying the Uvicorn startup command. The minimal requirements.txt (fastapi, uvicorn, sqlalchemy, aiofiles, python-multipart, pydantic) ensures rapid dependency installation in CI/CD pipelines.
The static directory is mounted as a StaticFiles endpoint, meaning the FastAPI application serves both the API and the frontend from a single process on a single port. This simplifies infrastructure significantly compared to separate frontend hosting and eliminates cross-origin configuration for same-domain deployments.
Database initialisation is handled automatically at application startup via models.Base.metadata.create_all(bind=engine), creating all tables if they do not exist. The startup event handler seeds the database with doctor, inventory, and staff records on a fresh installation, enabling immediate demonstration without manual data entry.

10.1 Environment Requirements

- Python 3.8+ with pip package manager
- No external database server — SQLite is included in the Python standard library
- Modern web browser with WebRTC support (for face recognition webcam access)
- Minimum 512 MB RAM for Uvicorn + FastAPI + SQLite in-memory operations
- Outbound internet access for face-api.js and Chart.js CDN loading (or bundle locally)

11. LIMITATIONS

While MediCare Pro demonstrates comprehensive HMS functionality, several limitations should be acknowledged for a complete evaluation:

- SQLite does not support concurrent write operations efficiently. High-traffic hospital deployments require migration to PostgreSQL or MySQL.: Scalability
- The current implementation does not enforce different views or permissions for patient, doctor, and administrative users at the API level.: No Role Separation
- The system does not verify whether a doctor already has an appointment at the requested time, potentially allowing double-booking.: No Appointment Conflict Detection
- Medical records currently store text-based prescriptions only. Diagnostic images (X-rays, lab reports) cannot be attached.: No File Upload
- The dashboard requires manual refresh; WebSocket-based real-time updates are not implemented.: No Real-Time Updates
- Client-side face recognition accuracy is influenced by lighting, camera quality, and angle. False positives are possible.: Face Recognition Accuracy
- Only system notifications provide an event log; a dedicated audit trail for data modifications is absent.: Limited Audit Logging

12. FUTURE WORK & ENHANCEMENTS

Several high-priority enhancements are identified for subsequent development phases:

- Implement JSON Web Tokens for stateless session management and introduce role-based access control distinguishing patient, doctor, nurse, pharmacist, and administrator permissions.: JWT Authentication & RBAC
- Replace SQLite with PostgreSQL for production-grade concurrency, connection pooling, and full-text search capabilities.: PostgreSQL Migration
- Add video consultation scheduling with WebRTC-based in-browser video calling between patients and doctors.: Telemedicine Integration
- Generate digitally signed PDF prescriptions directly from medical records for pharmacy dispensing.: Electronic Prescription
- Expose patient demographics and clinical records as FHIR R4 resources to enable interoperability with other healthcare systems.: HL7 FHIR Compliance
- Develop React Native or Flutter mobile clients for iOS and Android, consuming the existing REST API.: Mobile Application

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.154312

- Replace polling with WebSocket-based push notifications for appointment confirmations, billing events, and low-stock alerts.: Real-Time Notifications
- Implement time-slot management with conflict detection to prevent double-booking and enable doctor schedule visualisation.: Appointment Conflict Engine
- Integrate a symptom-checker API to suggest probable diagnoses based on patient-reported symptoms prior to consultation.: AI Diagnostics Assistance
- Extend the data model to support hospital networks with branch-level partitioning and cross-branch patient record access.: Multi-Hospital Support

13. CONCLUSION

MediCare Pro demonstrates that a comprehensive, feature-rich Hospital Management System can be built using lightweight, open-source technologies without sacrificing functionality or user experience. The system successfully integrates patient management, physician scheduling, appointment workflows, financial billing, medical records, inventory tracking, staff administration, and real-time analytics within a unified platform.

The incorporation of biometric face recognition login represents a meaningful innovation over conventional HMS authentication, reducing credential management burden for patients while maintaining appropriate security through embedding-based matching. The UPI QR payment integration aligns the billing module with contemporary Indian digital payment behaviour, reducing cash handling and improving payment traceability.

The architectural choice of FastAPI with SQLAlchemy and a vanilla JS SPA frontend delivers a system that is easy to deploy, easy to extend, and performant under moderate load. The modular design, with clear separation between API endpoint handlers, ORM models, and frontend modules, facilitates team development and independent testing of each component.

Future work will focus on production hardening (JWT authentication, PostgreSQL migration, HTTPS enforcement), role-based access control, real-time WebSocket notifications, and HL7 FHIR compliance for healthcare interoperability. MediCare Pro establishes a solid, extensible foundation upon which a production-grade hospital management platform can be constructed.

Figure 12: Patient Full History — GET /patients/1/history

REFERENCES

[1]. Benson, T. (2002). Why General Practitioners Use Computers and Hospital Doctors Do Not. British Medical Journal, 325(7372), 1086–1089.
[2]. Häyrinen, K., Saranto, K., & Nykänen, P. (2008). Definition, Structure, Content, Use and Impacts of Electronic Health Records. International Journal of Medical Informatics, 77(5), 291–304.
[3]. Kalra, D., et al. (2010). Electronic Health Record Standards. Yearbook of Medical Informatics, 19(1), 136–144.
[4]. Uruj, Jaleel;et al.(2023) Journal of Advanced Zoology ISSN: 0253-7214 Volume 44 Issue S-5 Year 2023 Page 1028:1044
[5]. Smilkov, D., et al. (2019). TensorFlow.js: Machine Learning for the Web and Beyond. Proceedings of Machine Learning and Systems.
[6]. Sahoo, P., et al. (2020). Digital Payments and Healthcare: UPI Adoption Among Urban Indian Hospital Patients. Journal of Health Management, 22(4), 512–528.
[7]. Ramirez, J., et al. (2021). Comparative Performance Analysis of Python REST Frameworks: FastAPI, Flask, and Django REST Framework. IEEE Access, 9, 123456–123468.
[8]. Kumar, R., et al. (2018). Inventory Management in Healthcare: The Case for Minimum-Stock Alert Systems. International Journal of Health Planning and Management, 33(2), e501–e512.
[9]. FastAPI Documentation. (2024). FastAPI — Modern, Fast Web Framework for Python. https://fastapi.tiangolo.com
[10]. SQLAlchemy Documentation. (2024). The Python SQL Toolkit and ORM. https://www.sqlalchemy.org
[11]. face-api.js. (2024). JavaScript Face Recognition API for the Browser. https://github.com/justadudewhohacks/face-api.js
[12]. National Payments Corporation of India. (2024). UPI — Unified Payments Interface. https://www.npci.org.in/what-we-do/upi