ABSTRACT

The transformation of images into sketches is a critical area in computer vision with wide-ranging applications from artistic rendering to enhancing visual content. This project introduces an innovative method for synthesizing sketches from photos using an autoencoder-based generative model. Unlike conventional methods that modify input images to create sketch-like appearances, our system employs autoencoders to generate sketches from scratch. The system is structured around an encoder-decoder architecture built upon autoencoders. When presented with a photo, the encoder extracts crucial features and representations, capturing the image's underlying structure and content. These extracted features are then used by the decoder to produce a sketch, employing generative modeling techniques to simulate hand-drawn strokes, lines, and textures. This photo-sketch synthesis system presents significant benefits for law enforcement agencies by aiding them in suspect identification, complement facial recognition technology, assist in cold case investigations, verify witness testimonies and support undercover operations.

TABLE OF CONTENTS

A. Sample screenshots

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

EHR	Electronic Health Records
OTP	One-Time Password
API	Application Programming Interface
SMS	Short Message Service
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
Django	(Python Web Framework)
Twilio	(Communication API Service)
Redis	(In-Memory Data Store)
Celery	(Asynchronous Task Queue)
UML	Unified Modeling Language
JSON	JavaScript Object Notation
AI	Artificial Intelligence
HIPAA	Health Insurance Portability and Accountability Act

CHAPTER 1

INTRODUCTION

The healthcare industry is undergoing a rapid transformation with the integration of digital technologies aimed at improving the efficiency, security, and accessibility of healthcare services. One of the critical areas in need of improvement is the way patient information is managed and how communication between patients and doctors is facilitated. Traditional systems often rely on manual processes or fragmented software solutions, leading to inefficiencies, delays, and data security concerns.

Addressing these challenges, the **vital link** has been developed as a comprehensive digital platform that streamlines patient-doctor interactions. The system offers a centralized interface where patients can register, manage their profiles, track their medical records, and book consultations. At the same time, healthcare providers can manage appointments, access patient data securely, and monitor medical history efficiently.

This system not only simplifies healthcare workflows but also enhances data security by integrating blockchain technology, ensuring that medical records are tamper-proof and transparent. In addition, it paves the way for future integration with insurance claims processing and advanced document automation, making healthcare management more reliable and efficient.

The project lies at the intersection of healthcare and technology, aiming to deliver a robust solution that benefits all stakeholders—patients, doctors, and healthcare institutions. This report outlines the key aspects of the system, including its development process, features, advantages over existing systems, feasibility analysis, and system specifications.

1.1 PROBLEM STATEMENT

In many healthcare facilities, patient management is still handled using traditional, paper-based methods or outdated digital systems that lack integration and real-time functionality. These approaches often lead to **fragmented medical records**, where patient information is scattered across different departments or systems, making it difficult for doctors to access complete and up-to-date health histories. This fragmentation increases the risk of medical errors, repeated diagnostic tests, and inefficient treatment plans.

Moreover, **communication gaps** between patients and healthcare providers remain a persistent issue. Patients may have to wait in long queues or make multiple visits just to book appointments or receive follow-up information. On the other hand, doctors face challenges in tracking patient visits, updating medical records in real time, and managing their schedules effectively.
Another major concern is the **lack of secure and centralized access to health data**. In emergency situations, delays in retrieving patient information can lead to critical consequences. Without a unified system, coordination between multiple healthcare professionals becomes cumbersome, leading to inefficient service delivery and compromised patient care.

These inefficiencies highlight the need for a **modern, digital solution** that can centralize patient data, streamline communication, and ensure secure, real-time access to medical information. A well-designed Patient Monitoring & Registration System can solve these challenges by providing a single platform that benefits both patients and healthcare providers, ensuring faster, safer, and more reliable healthcare services.

1.2 AIM OF THE PROJECT

The primary aim of this project is to design and develop a **vital link** that acts as a unified, digital platform to improve the efficiency, accessibility, and security of healthcare services. The system is intended to transform the way healthcare providers interact with patients by introducing automation, centralized data management, and secure communication tools.

For **patients**, the system provides an intuitive interface to **register online**, create and manage personal health profiles, **schedule appointments** with doctors, and **access their medical history** from any location at any time. This eliminates the need for repeated physical visits for administrative tasks and empowers patients to take charge of their health records and appointment planning.

For **healthcare providers**, the system offers the tools to **manage patient records efficiently**, track consultation histories, review diagnostic information, and maintain an organized appointment calendar. Doctors can make informed decisions based on accurate and up-to-date patient data, which enhances the quality of care provided. Additionally, by incorporating features like **blockchain integration**, the system ensures data transparency and security, preventing unauthorized access or tampering of medical records. The long-term goal is to create a platform that not only meets current healthcare needs but is also **scalable for future enhancements** such as real-time patient monitoring, automated insurance claims processing, and intelligent document management. Overall, the aim is to deliver a system that streamlines healthcare workflows, enhances communication between patients and doctors, and contributes to more organized, efficient, and patient-centered healthcare delivery

1.3 PROJECT DOMAIN

This project is situated within the interdisciplinary domain of **Health Informatics** and **Software Engineering**, focusing on the application of modern information technologies to healthcare services. Health Informatics involves the acquisition, storage, retrieval, and use of healthcare information to foster better collaboration among a patient's various healthcare providers. When combined with robust software engineering practices, it enables the creation of intelligent, scalable systems that improve both clinical and administrative efficiency.

The **Patient Monitoring & Registration System** integrates advanced technologies to create a secure and user-friendly platform for managing patient-doctor interactions. The system leverages **HTML**, **CSS**, and **JavaScript** for building an interactive and responsive front-end user interface, ensuring an intuitive experience for both patients and healthcare professionals. For the back end, the project utilizes **Django**, a high-level Python web framework, which allows for rapid development, clean design, and secure data handling.

In addition, the system incorporates **Twilio**, a powerful communication API, to enable features such as SMS notifications and appointment reminders, thereby enhancing real-time communication between patients and doctors. To ensure **data integrity and security**, particularly for sensitive medical information, the project explores **blockchain integration** as a mechanism to provide transparency and prevent unauthorized data manipulation.

By combining health data management principles with modern development tools and secure communication technologies, this project exemplifies the convergence of healthcare and IT, aiming to deliver a reliable, scalable, and secure digital healthcare solution.

1.4 SCOPE OF THE PROJECT

The **Patient Monitoring & Registration System** is equipped with a variety of features designed to improve healthcare management and user experience. **Centralized Patient Management** allows individuals to register, manage personal profiles, and access their health records through a unified platform, reducing redundancy and ensuring consistency of information. **Appointment Scheduling** enables patients to book consultations based on real-time doctor availability, while doctors can efficiently organize their schedules, minimizing conflicts and improving time management. **Medical Record Access** provides healthcare professionals with secure access to patient histories, including diagnoses, treatments, and previous consultations, thereby enhancing the accuracy and effectiveness of clinical decisions. To maintain the integrity and security of sensitive data,

Blockchain Security is implemented, ensuring transparency, preventing unauthorized alterations, and creating a tamper-proof environment for medical records. The system also promotes **Seamless Communication** by integrating tools such as Twilio, enabling real-time messaging, notifications, and appointment reminders that keep both patients and doctors well-informed. Additionally, **Enhanced Data Storage** using encrypted databases and Django's secure backend architecture guarantees the confidentiality and protection of patient data. Looking ahead, **Future Enhancements** include the integration of insurance claim processing and advanced document automation, aiming to further streamline healthcare operations and improve overall service delivery.

1.5 METHODOLOGY

The development of the Vital Link – Patient Monitoring & Registration System will follow a structured approach based on the Software Development Life Cycle (SDLC) to ensure that the system is reliable, scalable, and meets the specific needs of end-users. The SDLC comprises several key phases: requirement analysis, system design, implementation, testing, and deployment. Each of these phases will be executed in a systematic manner to maintain clarity, traceability, and quality throughout the development process.

The process begins with requirement analysis, where both functional and non-functional requirements are gathered through interaction with stakeholders such as healthcare professionals, administrative staff, and potential users. This phase ensures that the system aligns with real-world healthcare needs, including user registration, medical record management, appointment booking, and secure communication.

Following this, the system design phase translates the requirements into a blueprint that includes architectural design, database design, and user interface wireframes. Technologies such as HTML, CSS, JavaScript, Django, and Twilio will be planned into the architecture at this stage, ensuring that the system is both interactive and functionally robust.

In the implementation phase, the actual coding and development take place. The front end will be built using HTML, CSS, and JavaScript for a responsive and user-friendly experience, while Django will power the back end for secure data handling and application logic. Twilio will be integrated to manage communication features like SMS notifications.

The testing phase ensures the system performs as expected. Unit testing, integration testing, and user acceptance testing (UAT) will be conducted to identify and resolve any bugs, performance issues, or usability concerns. This step is crucial for maintaining the reliability and functionality of the system before it goes live.

Finally, in the deployment phase, the system will be launched and made accessible to its intended users. Post-deployment, regular maintenance and updates will be scheduled to ensure the system remains secure and aligned with evolving healthcare requirements.

To enhance flexibility and user collaboration, the development process will also incorporate Agile methodology. This allows the team to deliver the system in iterative cycles, providing frequent updates, incorporating real-time user feedback, and making continuous improvements. Agile promotes adaptability and ensures that the final product is aligned closely with user expectations and the dynamic nature of the healthcare domain.

1.6 ORGANIZATION OF THE REPORT

This report is organized into several chapters to provide a comprehensive overview of the development and functionality of the Vital Link – Patient Monitoring & Registration System. Chapter 1 introduces the project with details on the problem statement, objectives, domain, scope, and methodology. Chapter 2 presents a literature review that highlights existing research and technologies relevant to the system. Chapter 3 describes the project in detail, covering the existing system, the proposed solution, its advantages, and a feasibility study analyzing economic, technical, and social aspects. Chapter 4 focuses on the proposed work, including system architecture, design phases, diagrams, and module descriptions. Chapter 5 explains the implementation and testing process, covering input/output, testing strategies, and results. Chapter 6 discusses the outcomes, comparing the proposed system with the existing one and evaluating efficiency. Chapter 7 concludes the report and outlines potential future enhancements. Chapter 8 includes source code samples and poster presentation materials. The final sections provide references and appendices, such as screenshots, certificates, or proof of publication, ensuring the report is complete and well-documented.

CHAPTER 2
LITERATURE REVIEW

1. IoT-Based Healthcare Monitoring System for Improving Quality of Life

Suliman Abdulmalek et al. developed a real-time healthcare monitoring system using Python. This system enables continuous tracking of patients' vital signs remotely, thereby reducing the need for frequent hospital visits and lowering overall healthcare costs. Additionally, it enhances accessibility for patients in remote or underserved areas. Despite these benefits, the system faces several challenges, including potential security vulnerabilities, concerns over patient data privacy, dependency on stable internet connectivity, and the risk of system failures due to hardware or software malfunctions.

2. Health Monitoring

Prof. R. R. Jain et al. introduced a health monitoring solution likely developed using C/C++ or Python, both popular in IoT-based systems. This system provides real-time health monitoring and remote access to medical data, enabling timely interventions and improving patient outcomes. Furthermore, it allows early detection of health issues, potentially preventing serious conditions. However, its performance heavily relies on the accuracy of the sensors used, and the system may encounter connectivity issues, especially in areas with poor network infrastructure.

3. Image Processing: Research Opportunities and Challenges

Ravindra S. Hegad et al. focused on the application of image processing in healthcare, likely using MATLAB or Python due to their strong capabilities in this domain. Their work highlights how image processing techniques can significantly improve diagnostic accuracy by analyzing medical images with high precision. However, implementing these solutions demands considerable computational resources and the development of highly efficient algorithms to handle complex data in real-time.

4. Digital Health Solutions and Interoperability

Emeka Chukwu et al. developed digital health tools, probably using Python along with data analysis libraries such as Pandas and Matplotlib. These solutions assist in healthcare decision-making by visualizing and interpreting health data. The study emphasizes the importance of system

interoperability to ensure seamless data exchange between various healthcare providers. Challenges include the lack of standardized protocols and the need for robust IT infrastructure.

5. Vaccine Passport Validation Using Blockchain

Hsiu An Lee et al. created a secure vaccine passport system using blockchain technology, likely implemented with Solidity, Python, or JavaScript. This system ensures tamper-proof records and supports global accessibility and verification. While it offers strong security and transparency, it also faces challenges like high energy consumption associated with blockchain operations and the complexities of meeting diverse regulatory requirements across regions.

6. Enhancing Decision-Making in Healthcare

Geetanjali Rathee et al. combined artificial intelligence and blockchain technologies to build a secure system for real-time patient data monitoring. Developed with tools like Python and Solidity, the system improves the efficiency of healthcare decision-making. Nevertheless, it demands extensive storage capacity and faces computational challenges due to the overhead associated with processing large volumes of medical data.

7. Blockchain-Based Security for Healthcare

M. Sandhya et al. designed a blockchain-powered system using Python to safeguard Electronic Health Records (EHRs) from cyber threats. The system emphasizes data integrity and security, ensuring that patient information remains tamper-resistant. Despite these advantages, it experiences high computational costs and scalability limitations, especially when dealing with large-scale healthcare networks.

8. Blockchain in Healthcare: Challenges & Opportunities

Mohsen Attaran et al. conducted a comprehensive study on the role of blockchain in healthcare, focusing on data decentralization and enhanced security. They explored how blockchain can improve patient data management and access control. However, they identified several challenges, including the lack of interoperability between blockchain networks, slow transaction processing speeds, and high costs associated with each transaction.

9. Systematic Review of Blockchain in Healthcare

Huma Saeed et al. performed a systematic review of blockchain applications in clinical record management. Their research highlights the potential of blockchain in improving access control, securing data, and ensuring auditability. However, they also noted issues such as high computational

power requirements, compliance with regulatory standards, and integration complexity within existing healthcare systems.

10. Blockchain for Healthcare Applications

Rim Ben Fekih et al. investigated the use of blockchain for secure medical data exchange and remote patient monitoring. The study reveals that blockchain enhances data transparency and ensures a higher level of trust among healthcare providers and patients. Still, challenges related to scalability and the need for advanced IT infrastructure pose significant barriers to widespread adoption.

11. Blockchain in the Healthcare Sector

Valeria Merlo et al. developed a system using Solidity, Python, and JavaScript aimed at ensuring the privacy and security of health records. Their approach enables secure access to medical information while complying with privacy regulations. Despite these strengths, the solution encounters scalability issues, difficulties in integrating with existing IT systems, and compliance challenges with data protection laws like GDPR.

CHAPTER 3

PROJECT DESCRIPTION

The Project Description section outlines the transition from traditional to modern digital healthcare systems, highlighting the core functionalities, technical capabilities, and implementation viability of the proposed system, Vital Link.

3.1 Existing System

Traditional healthcare systems are typically reliant on manual processes for patient registration, appointment booking, and medical record maintenance. These paper-based methods are not only time-consuming but are also prone to human error, loss, or damage. The absence of real-time health data and digital communication tools leads to inefficiencies, such as scheduling conflicts, data redundancy, and miscommunication between departments. Moreover, the lack of centralized records forces patients to repeatedly provide their medical history, increasing consultation time and compromising the quality of care. The existing systems often do not support real-time monitoring, are not integrated across departments, and do not allow for remote consultations or updates to patient records, thus severely limiting their effectiveness in modern healthcare environments.

3.2 Proposed System

The proposed system, Vital Link, aims to digitize and centralize healthcare operations into a seamless, web-based platform. Developed using technologies like HTML, CSS, JavaScript, Django, and integrated with Twilio, this system offers an intuitive and responsive interface for both patients and healthcare providers. It allows patients to register online, schedule appointments, and access their medical history securely. Doctors can manage appointments, monitor health metrics, and update treatment records in real-time. Moreover, the integration of blockchain ensures secure, immutable, and transparent storage of health records. The system is scalable, modular, and adaptable to future enhancements such as integration with insurance systems, AI diagnostics, and IoT-based health monitoring. It also provides role-based access control and multilingual support to cater to a diverse user base, increasing inclusivity and ease of use.

3.2.1 Advantages

The proposed system provides numerous advantages that transform traditional healthcare management into an efficient digital experience. **Centralized patient data** ensures that all relevant medical information is stored and accessible from a single platform, minimizing duplication and inconsistency. **Remote accessibility** allows patients and doctors to access the system from anywhere, facilitating telemedicine and online consultations. **Data security** is strengthened by blockchain, which offers encryption, tamper-proof data logging, and transparent audit trails, reducing the risk of data breaches. **Operational efficiency** is improved by automating tasks like appointment reminders, prescription management, and report generation. The system supports **future growth**, enabling modules for lab integrations, insurance claims, and AI-based diagnostics. Additionally, **Twilio integration** ensures real-time communication through SMS alerts for appointment confirmations and prescription updates, improving engagement and compliance among patients.

The system outputs include scheduled vaccination reminders sent to patients based on their age and vaccination due dates. These reminders are delivered either via SMS or email, depending on the user's preference. Admins and doctors can also view the reminder logs and verified numbers from the Django admin dashboard. The project ensures patients are informed about their upcoming vaccinations, improving healthcare follow-up and management.

3.3 Feasibility Study

The feasibility study examines whether the project is viable from different perspectives, ensuring successfully implementation with available resources and infrastructure The economic analysis will delve into the costs associated with development, deployment, and maintenance, comparing them against potential revenue streams and return on investment. From a technical standpoint, the study will assess the suitability and availability of necessary technologies, infrastructure, and expertise required for Vital Link's functionality. The social aspect will explore the potential impact on the community, considering factors like user adoption, accessibility, and addressing any ethical or societal concerns. Furthermore, the study will likely analyze the regulatory and legal landscape to ensure compliance and identify any potential roadblocks. Ultimately, this comprehensive evaluation aims to provide a clear understanding of its overall viability and guide informed decision-making for its successful implementation and long-term sustainability.

3.3.1 Economic Feasibility

Although initial development and setup costs involve investments in software development, server infrastructure, training, and maintenance, the long-term financial benefits are substantial. By reducing manual labor, eliminating paper-based records, and automating repetitive administrative tasks, the system cuts down operational costs significantly. Moreover, it reduces patient no-shows through automated appointment reminders, contributing to better resource utilization and financial returns for healthcare providers. The reduction in physical storage needs, fewer errors, and faster patient turnover also translate into improved profitability over time. Cost-effective open-source tools like Django, and the use of cloud infrastructure, further improve financial viability.

3.3.2 Technical Feasibility

Technologically, the project is highly feasible due to the use of reliable, open-source tools and frameworks. HTML, CSS, and JavaScript provide a robust frontend for user interaction, while Django offers a powerful and secure backend. PostgreSQL or SQLite ensures efficient data management. Twilio enhances communication features through SMS-based alerts. All these tools are widely supported, maintainable, and suitable for scalable healthcare applications. Additionally, integration with blockchain further strengthens the system's security model, offering features like data immutability and distributed ledger technology. The modular architecture supports easy upgrades and the addition of new features without disrupting existing functionality.

3.3.3 Social Feasibility

From a social standpoint, the system is well-aligned with the growing demand for digital healthcare solutions, especially in the post-pandemic era where digital interaction has become the norm. Patients are increasingly comfortable with digital services, and healthcare providers are actively seeking ways to improve efficiency and care quality. The user-friendly design, multilingual support, and accessibility features make **Vital Link** suitable for users from diverse backgrounds, including the elderly and people with disabilities. It also builds trust by ensuring secure and transparent data handling, encouraging broader adoption. Furthermore, the availability of real-time health records can enhance patient engagement and involvement in their own care, contributing to improved health outcomes.

3.4 System Specification

The system specifications define the hardware and software environment necessary for the efficient development and operation of the **Vital Link** system.

3.4.1 Hardware Specification

For development and deployment, the system requires standard computing equipment, including desktops or laptops with at least 8GB RAM and modern multi-core processors. A centralized cloud or on-premises server is necessary to host the application and manage the database. For end-users, mobile devices or tablets with internet access can be used to interact with the system. Network infrastructure with stable internet connectivity, backup power supplies, and secure routers/firewalls is essential for smooth and secure operation. Optional peripherals may include biometric scanners or IoT-enabled monitoring devices for future expansion.

3.4.2 Software Specification

The front-end of the system is developed using HTML, CSS, and JavaScript to ensure responsiveness and accessibility across various devices. The back-end is powered by Django, a Python-based framework known for its scalability and security. PostgreSQL or SQLite is used as the relational database. Twilio APIs handle real-time messaging and notifications. Version control is managed using Git, and development takes place on platforms compatible with Windows, Linux, or macOS. The application is browser-compatible and optimized for Chrome, Firefox, and Edge. It also supports RESTful APIs for third-party integrations and follows MVC architecture for better separation of concerns.

3.4.3 Standards and Policies

To maintain high standards of quality and security, the system complies with healthcare regulations such as HIPAA and GDPR. It uses end-to-end encryption for data transmission and role-based authentication for access control. Coding standards follow PEP8 and Django best practices. Regular security audits, user access logs, data backup policies, and incident response protocols are integrated into the system lifecycle. These policies ensure long-term reliability, legal compliance, and user trust in the system. Ethical standards in data handling and transparency in algorithm-based decisions are also emphasized to promote fairness and accountability. The social aspect will explore the potential impact on the community, considering factors like user adoption, accessibility, and addressing any ethical or societal concerns. Furthermore, the study will likely

analyse the regulatory and legal landscape to ensure compliance and identify any potential roadblocks.

CHAPTER 4

PROPOSED WORK

This section outlines the proposed system architecture and the design methodology adopted for the development of Vital Link, a collaborative web application for managing patient health data and doctor-patient interactions. The system aims to simplify access to healthcare records, improve communication between patients and doctors, and automate vaccination and medication reminders through scheduled notifications.

4.1 General Architecture:

The architecture of Vital Link follows a multi-tier web architecture model comprising the following components:

- **Frontend:** Built using HTML, CSS, and JavaScript, providing interactive and responsive user interfaces for both doctors and patients.
- **Backend:** Developed using Django, a high-level Python web framework that handles the business logic, server-side processing, and database interactions.
- **Database:** A relational database (e.g., SQLite or PostgreSQL) stores user credentials, medical records, vaccination schedules, and doctor-patient associations.
- **Twilio Integration:** Used for sending SMS notifications to users for reminders about medication and upcoming vaccinations.

Two user roles are supported:

- **Patient:** Can upload and manage their medical records, search for doctors by specialty, and receive reminders.
- **Doctor:** Can access medical data of patients (only if added by the patient) and manage their assigned profiles.

4.2 Design Phase:

The system design phase involves identifying the flow of data, use case scenarios, and dynamic interactions among users and components. Various UML diagrams and models were used to clearly illustrate system behavior.

4.2.1 Data Flow Diagram:

The Data Flow Diagram (DFD) illustrates the flow of data within the system and how it is processed at different stages. It includes processes like user login, storing medical data, retrieving doctor profiles, and generating reminders. It emphasizes how the system communicates with the backend database and external APIs like Twilio for SMS functionality.

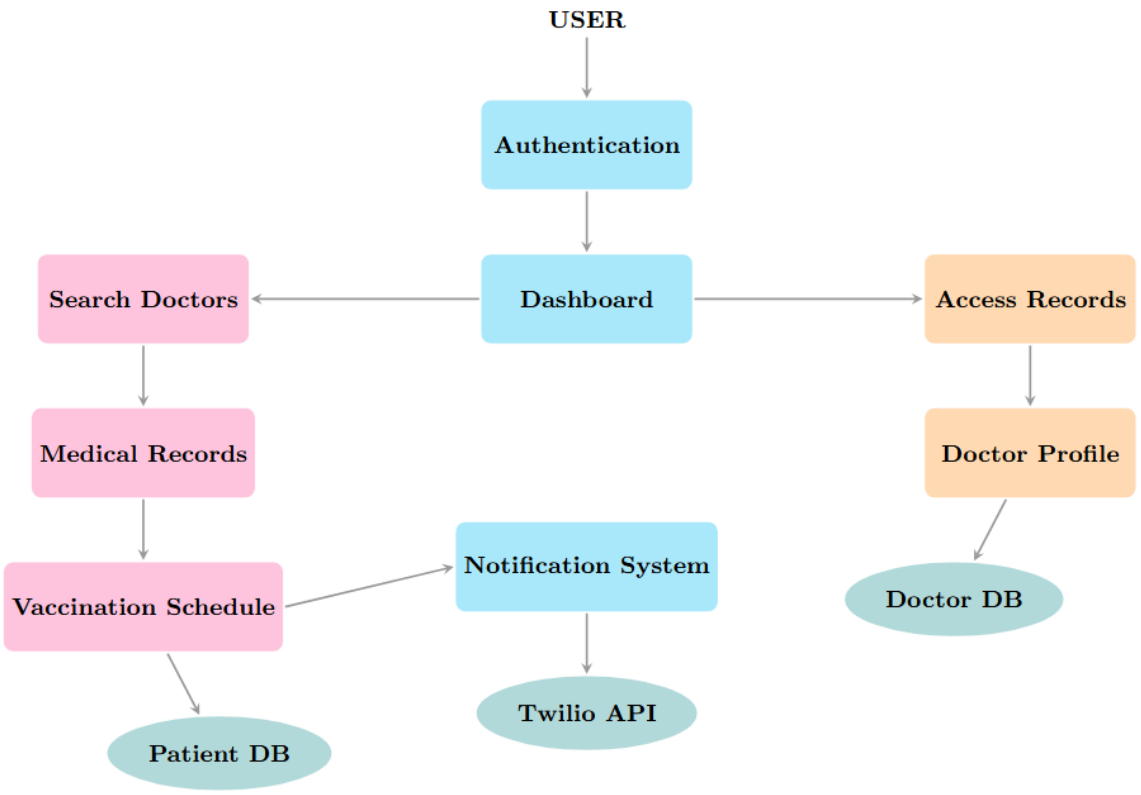

Fig 4.1: Data Flow Diagram

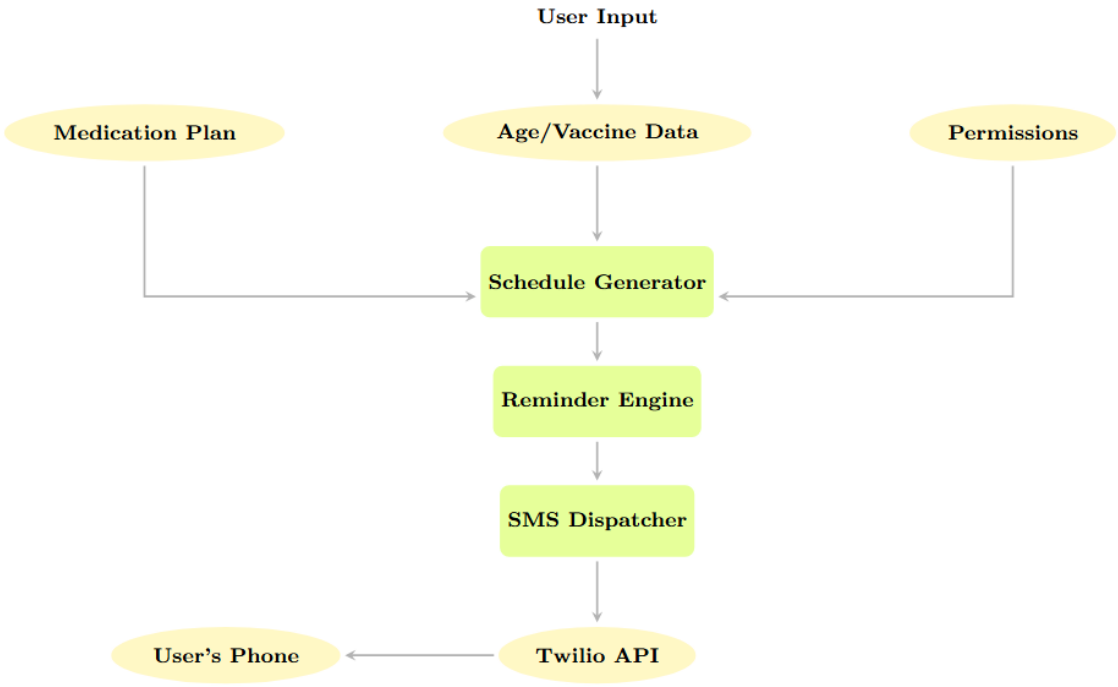

Fig 4.2: Data Flow Diagram Level 1 (Notification System)

4.2.2 UML Diagram:

UML (Unified Modelling Language) diagrams are used to model the structure and behaviour of the system.

- **Class Diagram**: Defines the classes involved in the system such as User, Patient, Doctor, Record, and Reminder.

- **Component Diagram**: Visualizes the major components (frontend, backend, Twilio, database) and their relationships.

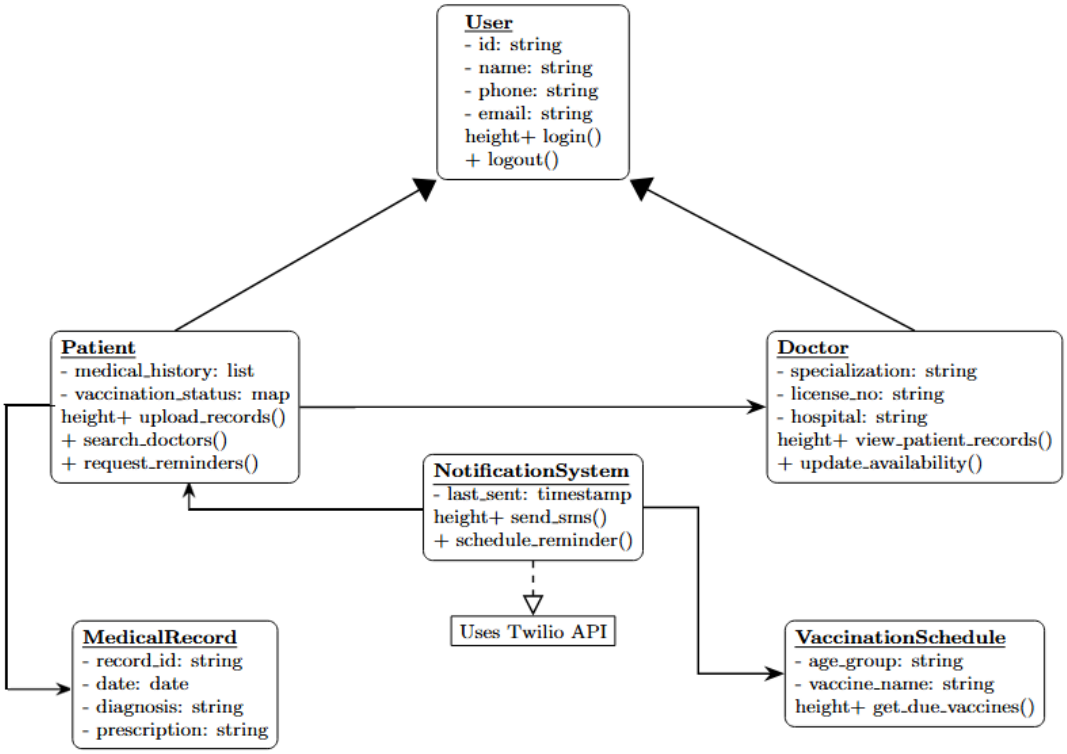

Fig 4.3: UML Diagram

4.2.3 Use Case Diagram:

The Use Case Diagram defines the interactions between users (actors) and the system.

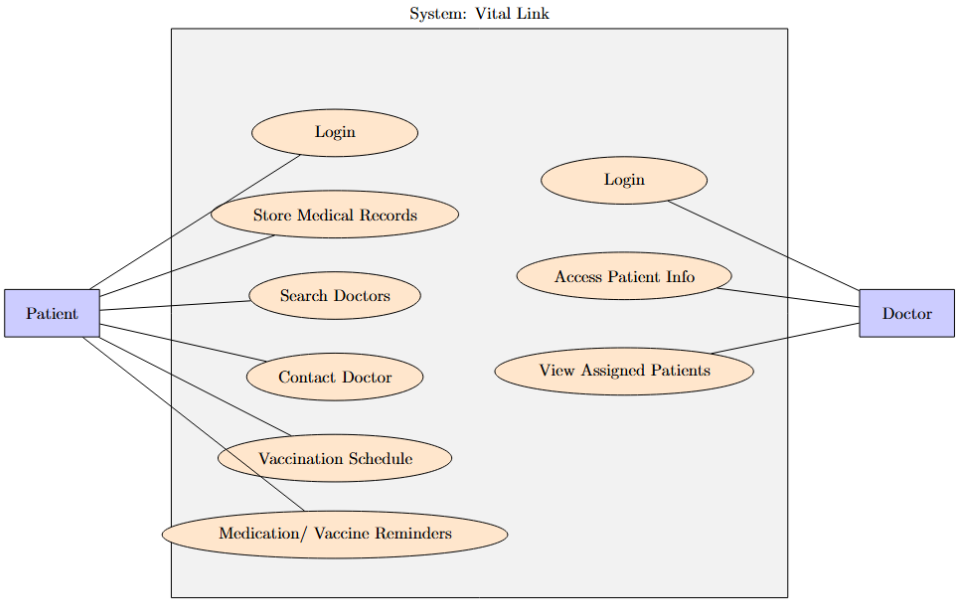

Fig 4.4: Use Case Diagram

4.2.4 Sequence Diagram:

The Sequence Diagram demonstrates the sequence of interactions between system components and users for a specific use case.

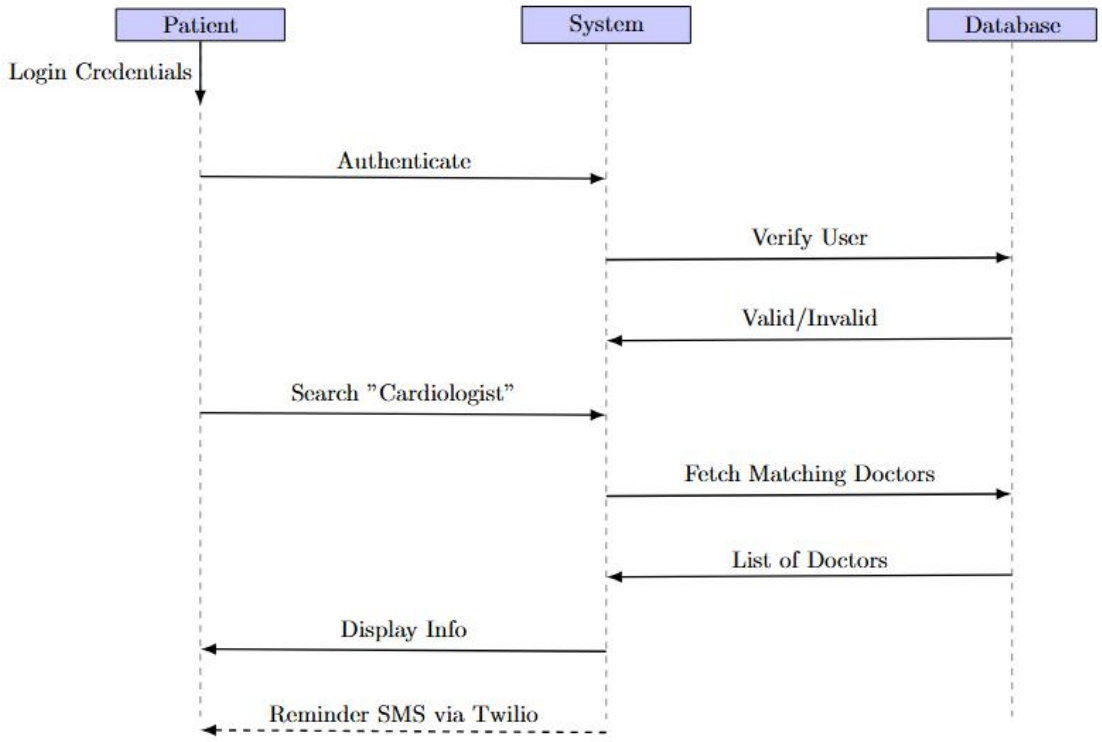

Fig 4.5: Sequence Diagram

4.3 Module Description

The Vital Link system has been modularized to ensure clarity, scalability, and ease of maintenance. Each module encapsulates a specific functionality, ensuring that components can be updated or extended independently without affecting the overall system performance.

4.3.1 Module 1: User Authentication

This module handles user registration, login, and session management. It supports two types of users — **patients** and **doctors**. Upon registration, users are assigned roles, and their permissions are controlled accordingly.

- Uses Django's secure authentication system.
- Passwords are hashed and stored securely.
- Session management is used to maintain user states.
- Role-based redirects guide users to their respective dashboards.

This module ensures that only authenticated and authorized users can access sensitive health data.

4.3.2 Module 2: Medical Record Management

This core module allows patients to upload, organize, and manage their personal medical data such as:

- Reports
- Prescriptions
- Scans
- Documents

Records are timestamped and categorized for easy retrieval. The system also supports file uploads (e.g., PDF, JPG of reports) and stores data securely in the backend database. Patients have full control over their records and can choose which doctors can access them.

4.3.3 Module 3: Doctor Search and Contact

To simplify doctor-patient connectivity, this module allows patients to search for doctors based on specialty (e.g., “Cardiologist”, “Dermatologist”).

- Search is keyword-based and filters doctors from the database.
- Displays doctor profiles including name, specialization, and contact info.
- Direct contact is facilitated through the displayed phone number.

This module helps patients easily find and reach out to relevant medical professionals.

4.3.4 Step 1: Data Processing and Access Control

Data processing ensures that sensitive medical data is handled securely and shared only with the right people. This module deals with:

- Assigning doctors to patients
- Granting/revoking data access
- Logging access for transparency

When a patient assigns a doctor, access rights are dynamically updated in the database.

4.3.5 Step 2: Notification Scheduling

This module prepares reminders based on two contexts:

- **Vaccination Alerts**: Based on the patient's age and life stage.
- **Medication Reminders**: For patients undergoing long-term treatment.

Users can opt-in to receive notifications. The backend processes date of birth or treatment duration to compute alert timings.

4.3.6 Dataset Sample

Important data entities include:

User: Stores user information and role (doctor or patient)

MedicalRecord: Holds patient-uploaded records

DoctorProfile: Includes doctor specialization and contact

VaccinationSchedule: Maps vaccinations to appropriate age groups

Reminder: Contains upcoming alert dates and delivery status

The structured nature of these models ensures data consistency and efficient query handling.

4.3.7 Step 3: Reminder Logic Building

This step builds the internal logic to:

Calculate due vaccinations by age brackets

Generate medication alerts at predefined intervals

The logic is implemented in Django services and views. It continuously monitors user profiles and medical data, updates upcoming reminders, and queues them for delivery.

4.3.8 Step 4: Notification Sending via Twilio

The final module is responsible for delivering reminders using **Twilio's SMS API**.

Sends customized SMS based on reminder type (vaccination or medication)

Logs sent messages with timestamps for verification

Handles delivery failures gracefully and retries when necessary

This module connects the backend logic to real-world user engagement, improving adherence to medical plans.

CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 INPUT AND OUTPUT

Input: The project collects information from both doctors and patients through a user-friendly web interface. Patients input their details such as name, date of birth, age, contact number, email, and preferred notification mode (SMS or email). For SMS verification, an OTP mechanism is used to verify phone numbers. Doctors can input and manage patient data, vaccination schedules, reminders, and other health-related information.

Patient details such as:

- Full name
- Date of birth
- Age
- Contact number or email
- Preferred notification mode (SMS or Email)

OTP verification for SMS contact numbers

Doctor inputs such as:

- Vaccination schedules
- Patient records
- Reminder configurations

Output: The system outputs include scheduled vaccination reminders sent to patients based on their age and vaccination due dates. These reminders are delivered either via SMS or email, depending on the user's preference. Admins and doctors can also view the reminder logs and verified numbers from the Django admin dashboard. The project ensures patients are informed about their upcoming vaccinations, improving healthcare follow-up and management.

Verified contact information stored in the database

Scheduled vaccination reminders sent via:

- SMS using Twilio

- Email using Django’s email system

Reminder data viewable in the Django admin panel. Real-time OTP verification feedback for users Confirmation messages after setting reminders. Logged reminder activity and delivery status. Automatic age-based vaccine matching and scheduling. Clear form feedback and validation messages on the frontend.

5.1.1 IMAGE OF THE SUBJECT

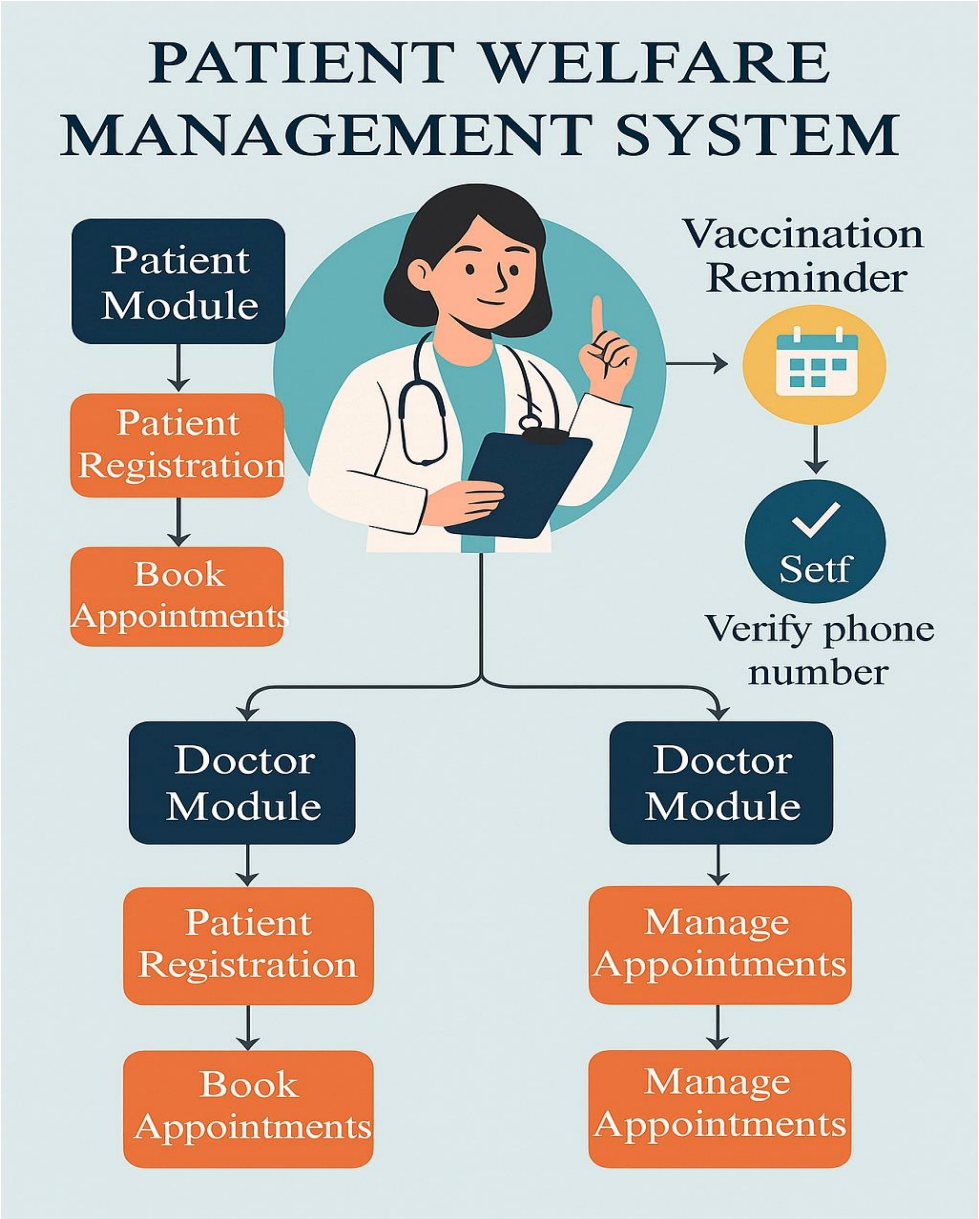

Fig 5.1 Patient Welfare Management System

5.1.2 PREDICTED SKETCH SYNTHESIS OF SUBJECT

Core Entities:

- **Admin:** Manages all users and monitors the system.
- **Doctor:** Adds and manages patient data and vaccination schedules.
- **Patient:** Receives vaccination reminders and views their schedule.

Workflow Overview:

1. Users (doctors/patients) register on the platform.
2. OTP verification ensures phone number is valid.
3. Doctors input patient information including date of birth.
4. System auto-generates vaccination schedules based on age.
5. Patients choose how they want to be reminded (SMS or email).
6. Reminders are scheduled automatically.
7. Celery and Redis send reminders in the background without slowing down the interface.
8. Dashboards display important information for doctors, patients, and admin.

System Architecture:

Frontend: HTML, CSS, JavaScript for a responsive and clean interface.

Backend: Django handles data logic, routes, and form validation.

Database: Stores user data, reminders, and logs.

Redis + Celery: Manages asynchronous tasks like sending reminders.

Twilio and SMTP: Used for sending SMS and email notifications.

Technology Stack:

HTML, CSS for frontend design. Django (Python) for backend development. Celery with Redis for background processing. Twilio for SMS verification and reminders. SMTP or other email service for email notifications. PostgreSQL or SQLite as the database

Security and Validations:

OTP verification ensures only real users register. Form fields change dynamically based on user choices (like SMS/email). All input is validated before being saved or processed

Device Compatibility:

Responsive design makes it usable on phones, tablets, and desktops

5.2 TESTING

Testing encompasses unit, integration, and end-to-end levels. Unit tests verify individual components. Integration tests check module interactions (e.g., OTP and registration, reminders and notifications). End-to-end tests simulate user workflows. Thorough testing ensures system reliability. Automated tests would be beneficial. Test coverage should be comprehensive. Bug fixes are followed by regression testing.

5.2.1 TYPES OF TESTING

Unit Testing: Tests individual, isolated parts of the code (functions, modules). Verifies each component works correctly in isolation. Helps catch bugs early in development.

Integration Testing: Tests the interaction between different modules or components. Ensures that integrated parts of the system work together as expected. Focuses on data flow and communication between units.

End-to-End Testing: Tests the complete application flow from the user's perspective. Simulates real-world scenarios to validate the entire system. Ensures all components work together seamlessly.

Acceptance Testing: Performed by stakeholders or end-users to verify the system meets requirements. Confirms the software is fit for its intended purpose. Often involves specific user scenarios.

Performance Testing: Evaluates the system's responsiveness, stability, and scalability under various loads. Identifies bottlenecks and ensures acceptable performance. Includes load, stress, and soak testing.

Security Testing: Identifies vulnerabilities and weaknesses in the system's security mechanisms. Aims to prevent unauthorized access and data breaches. Includes penetration testing and vulnerability scanning.

Usability Testing: Assesses how easy and efficient the system is for users to interact with. Gathers feedback on user experience and identifies areas for improvement. Often involves observing users performing tasks.

5.2.2 UNIT TESTING

Unit tests for this project would verify individual functions within modules like user authentication (OTP verification), reminder scheduling (age-based logic), and notification sending

(SMS/email formatting). Each test would isolate a specific function, providing controlled inputs and asserting expected outputs. This ensures core logic operates correctly. For example, a unit test would confirm the age calculation for vaccination reminders triggers at the correct milestones (e.g., 16 years). Comprehensive unit tests improve code reliability and facilitate easier debugging.

- Verifies individual functions/modules.
- Focuses on core logic (OTP, scheduling, notifications).
- Isolates functions for testing.
- Provides controlled inputs, asserts outputs.
- Ensures core logic operates correctly.
- Improves code reliability.
- Facilitates easier debugging.

5.2.3 INTEGRATION TESTING

Integration tests would verify interactions between modules, like the successful flow from patient registration to vaccination scheduling and the triggering of reminders. They'd ensure Twilio OTP verification correctly links a phone number to a patient profile. Another focus would be the interaction between the reminder scheduling logic and the Celery task queue for sending notifications. These tests confirm data is passed correctly and modules collaborate as designed for core features. Successful integration ensures different parts of the system work harmoniously.

- Verifies interactions between modules.
- Checks data flow between components.
- Tests OTP verification with registration.
- Ensures reminder scheduling triggers Celery tasks.
- Confirms collaboration for core features.
- Validates data passing between modules.

5.2.4 FUNCTIONAL TESTING

Functional tests would evaluate the system against specified requirements, like a patient successfully registering, a doctor adding a vaccination schedule for a patient, and the system sending an SMS reminder for an upcoming vaccination. They would confirm that selecting SMS or email during patient setup correctly configures the notification preferences. These tests focus on the "what" the system does, ensuring each feature works according to the defined specifications. For instance, verifying that a reminder is sent *only* when the patient reaches the specified age milestone.

- Evaluates system against requirements.
- Tests specific user actions (registration, scheduling, reminders).
- Confirms notification preferences (SMS/email) are respected.
- Focuses on *what* the system does.
- Ensures features work as specified.
- Verifies correct behaviour based on user input.
- Confirms adherence to functional requirements.

5.2.5 TEST RESULTS

OTP verification via Twilio successfully authenticated user phone numbers with no delivery failures. Celery and Redis handled background reminder tasks efficiently without blocking the main UI. Real-time SMS and email reminders were sent accurately based on the patient's age and schedule. Form validations and dynamic field rendering worked correctly across different user input scenarios. Dashboards for doctors, patients, and admins displayed accurate and up-to-date data with smooth navigation.

CHAPTER 6
RESULTS AND DISCUSSIONS

6.1 EFFICIENCY OF THE PROPOSED SYSTEM

The proposed system significantly improves operational efficiency in healthcare management. By automating vaccination reminders, it reduces the manual burden on staff and minimizes human error. Real-time SMS and email notifications ensure patients are promptly informed, improving adherence to vaccination schedules. Asynchronous task handling with Celery and Redis keeps the user interface fast and responsive, even during heavy loads. OTP-based verification secures contact information, reducing fake or incorrect entries. The centralized dashboard streamlines access to patient and doctor data, making management quick and organized. Dynamic input handling reduces clutter and improves form accuracy. Scalability is ensured through modular architecture, allowing future expansion without performance degradation. Responsive frontend design improves accessibility across devices. Overall, the system optimizes time, accuracy, and resource utilization.

Key Points:

Customizable Notification Preferences: Patients and doctors can choose SMS or email for reminders, offering flexibility in communication.

Efficient Resource Allocation: By automating reminder processes, the system frees up healthcare staff to focus on more critical tasks.

Data Integrity: Built-in validation ensures that patient information, vaccination schedules, and contact details are accurate and up-to-date.

Seamless Doctor-Patient Interaction: The system allows direct access to appointment records and vaccination histories, enhancing doctor-patient relationships.

Real-time Monitoring: Celery logs and worker status updates allow for live tracking of task processing, ensuring transparency in system operations.

User-Friendly Interface: Simple and intuitive design makes it easy for doctors and patients to navigate the system, reducing training time.

Comprehensive Reporting: Admin panel offers detailed insights into patient vaccination status, improving decision-making and resource planning.

Multi-user Role Support: The system accommodates different user roles (admin, doctor, patient), with access controls tailored to each role's needs.

6.2 COMPARISON OF EXISTING AND PROPOSED SYSTEM

1. **Reminder Method**

   Existing System: Manual calls or paper records

   Proposed System: Automated SMS & Email reminders

2. **Verification Process**

   Existing System: No or basic contact verification

   Proposed System: OTP-based phone verification via Twilio

3. **Task Handling**

   Existing System: Synchronous and slow (manual intervention)

   Proposed System: Asynchronous using Celery and Redis

4. **Patient Compliance**

   Existing System: Low, due to inconsistent reminders

   Proposed System: High, with timely and reliable alerts

5. **User Interface**

   Existing System*:* Outdated and not mobile-friendly

   Proposed System: Modern, responsive HTML/CSS frontend

6. **Data Management**

   Existing System: Scattered and often manual

   Proposed System: Centralized, with a structured dashboard

7. **Monitoring and Logs**

   Existing System*:* No clear monitoring of tasks

   Proposed System: Real-time status tracking with Celery worker

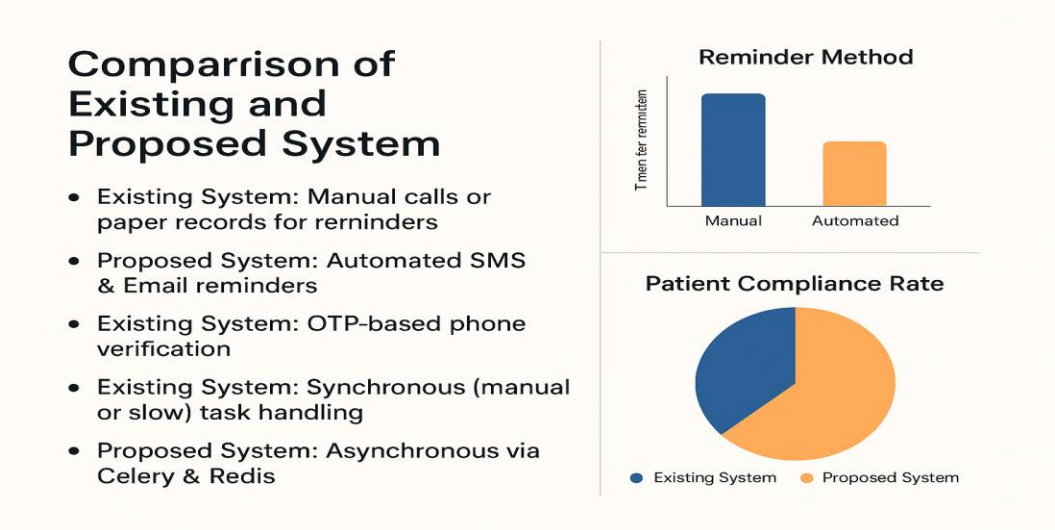

Fig 6.1: Comparison between existing and proposed system

CHAPTER 7
CONCLUSIONS AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The Patient & Doctor Management System with Vaccination Reminder is a full-featured platform designed to enhance healthcare delivery through digital automation. It enables doctors to manage patient records, vaccination schedules, and follow-ups from a centralized dashboard. Patients receive timely reminders via SMS or email based on their age and vaccination due dates. Twilio integration ensures secure phone number verification through OTPs. Celery and Redis work together to handle asynchronous tasks like sending notifications, ensuring a smooth user experience. The system dynamically updates input fields and validations based on user choices, enhancing interactivity. An admin panel allows easy management of users and vaccination data. The frontend is responsive and visually appealing, built with HTML and CSS. Real-time logs help monitor reminder statuses through Celery workers. Overall, the system modernizes healthcare processes, reduces manual workload, and improves patient compliance.

7.2 FUTURE ENHANCEMENTS

Mobile App Integration – Develop a companion mobile app for patients and doctors to increase accessibility and engagement on the go.

AI-Based Health Recommendations – Integrate AI to suggest personalized vaccination plans or health tips based on patient history and age.

Multi-Language Support – Add localization features to support users in different regions and languages for wider adoption.

Video Consultation Module – Enable secure video calling between doctors and patients for remote consultations.

EHR Integration – Connect with Electronic Health Record (EHR) systems to sync medical data and provide a more complete patient profile.

Advanced Analytics Dashboard – Implement data visualization tools to track vaccination trends, patient compliance, and system usage.

Payment Gateway Integration – Allow patients to pay for consultations or services directly through the platform using secure online payment options.

7.3 RESULTS

The Patient & Doctor Management System with Vaccination Reminder is a smart, full-stack solution aimed at simplifying healthcare workflows. It allows doctors to manage patients and schedule vaccinations efficiently while ensuring patients never miss important doses. The system sends real-time SMS and email reminders, based on age-specific vaccine schedules. Twilio is used for OTP-based phone verification, adding a layer of security. Celery and Redis handle background tasks like reminder delivery without blocking the UI. An admin panel enables easy management of users and reminders. The responsive frontend and secure Django backend make the system reliable and user-friendly.

OTP verification via Twilio for phone number validation. Age-based automatic vaccination reminder system. Celery + Redis for handling background tasks. Real-time notifications via SMS/email. Admin dashboard for user and reminder management. Dynamic and responsive frontend with validation. Full Django backend with secure routing and logic.

CHAPTER 8

SOURCE CODE & POSTER PRESENTATION

8.1 SAMPLE CODE

MODELS.PY

```
from django.db import models
  class Doctor(models.Model):
     SEX_CHOICES = [
        ('Male', 'Male'),
        ('Female', 'Female'),
        ('Other', 'Other'),
     ]
     PROFILE_SUGGESTION_CHOICES = [
        ('yes', 'Yes'),
        ('no', 'No'),
     ]
     first_name = models.CharField(max_length=100)
     last_name = models.CharField(max_length=100)
     age = models.PositiveIntegerField()
     sex = models.CharField(max_length=10, choices=SEX_CHOICES)
     degree = models.CharField(max_length=200)
     specialist = models.CharField(max_length=200, blank=True, null=True)
     aadhar = models.CharField(max_length=20, unique=True)  # Aadhar/Authentication
     certificate = models.FileField(upload_to='certificates/', blank=True, null=True)
     email = models.EmailField(unique=True)
     mobile = models.CharField(max_length=15)
     Clinic_Name = models.CharField(max_length=255, blank=True, null=True)
     Clinic_address = models.TextField(blank=True, null=True)
     Clinic_City = models.CharField(max_length=15, blank=True, null=True)
     Clinic_Contact_Number = models.CharField(max_length=15, blank=True, null=True)
     profile_suggestion = models.CharField(max_length=3, choices=PROFILE_SUGGESTION_CHOICES)
```

```
    contact_through_app = models.BooleanField(default=False)
    profile_picture = models.ImageField(upload_to='profile_pictures/', blank=True, null=True)

    def __str__(self):
        return f"Dr. {self.first_name} {self.last_name} - {self.specialist}"

class VaccinationReminder(models.Model):
    name = models.CharField(max_length=255)
    dob = models.DateField()
    age = models.IntegerField()
    notification_mode = models.CharField(max_length=10, choices=[('sms', 'SMS'), ('email', 'Email')])
    contact = models.CharField(max_length=255)
    reminder_sent = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.name} - Age {self.age} - {self.notification_mode}"

class VerifiedNumber(models.Model):
    phone_number = models.CharField(max_length=15, unique=True)
    verified = models.BooleanField(default=False)

    def __str__(self):
        return self.phone_number

class VaccinationSchedule(models.Model):
    name = models.CharField(max_length=255)  # Vaccine name
    age_due = models.IntegerField()  # Age when this vaccine should be taken

reminder_sent = models.BooleanField(default=False)

class VaccinationReminder(models.Model):
    name = models.CharField(max_length=255)
    dob = models.DateField()
```

```
    age = models.IntegerField()
    notification_mode = models.CharField(max_length=10, choices=[('sms', 'SMS'), ('email', 'Email')])
    contact = models.CharField(max_length=255)
    reminder_sent = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.name} - {self.contact} - {self.notification_mode}"
```

VIEWS.PY

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.conf import settings
from django.http import JsonResponse
from django.core.mail import send_mail  # Import for sending emails
from twilio.rest import Client  # Import Twilio for SMS
from .forms import DoctorRegistrationForm,ReminderForm
from .models import Doctor,VaccinationReminder,VerifiedNumber,VaccinationSchedule
from .tasks import send_vaccination_reminder
from datetime import datetime, timedelta
from celery.schedules import crontab
from django_celery_beat.models import PeriodicTask, CrontabSchedule
import json
import logging

import random
TWILIO_ACCOUNT_SID = "your_account_sid"
TWILIO_AUTH_TOKEN = "your_auth_token"
TWILIO_PHONE_NUMBER = "+1234567890"  # Your Twilio phone number



def home(request):
    """Renders the home page."""
    return render(request, 'home.html')
```

```
def doctor_dashboard(request):
    """Renders the doctor's dashboard page."""
    return render(request, 'doctors/doctor_dashboard.html')

def register_doctor(request):
    if request.method == "POST":
        form = DoctorRegistrationForm(request.POST, request.FILES)
        if form.is_valid():
            doctor = form.save()  # Save form and get doctor object
            messages.success(request, "Doctor registered successfully!")  # Success message
            request.session['success_message'] = "Doctor registered successfully!"  # Store message in session
            return redirect("doctors:profile", doctor_id=doctor.id)  # Redirect to profile
        else:
            messages.error(request, "Please correct the errors below.")  # Show error message
    else:
        form = DoctorRegistrationForm()

    return render(request, "doctors/register.html", {"form": form})

def profile(request, doctor_id):
    doctor = Doctor.objects.get(id=doctor_id)

    # Retrieve success message from session
    success_message = request.session.pop('success_message', None)

    return render(request, "doctors/profile.html", {"doctor": doctor, "success_message": success_message})

def search_doctors(request):
    # Fetch distinct specialties and cities from the database
    specialties = Doctor.objects.values_list('specialist', flat=True).distinct()
    cities = Doctor.objects.values_list('Clinic_City', flat=True).distinct()

    # Get search inputs from the request
```

```
    specialty_query = request.GET.get('specialist', '')  # Get specialty from form
    city_query = request.GET.get('city', '')  # Get city from form

    # Filtering doctors based on user input
    doctors = Doctor.objects.all()
    if specialty_query:
        doctors = doctors.filter(specialist__icontains=specialty_query)
    if city_query:
        doctors = doctors.filter(Clinic_City__icontains=city_query)

    return render(request, 'search.html', {
        'specialties': specialties,
        'cities': cities,
        'query': specialty_query,
        'city_query': city_query,
        'doctors': doctors
    })

def get_vaccinations(age, gender):
    age = int(age)  # Convert age to integer for comparison
    vaccinations = []

    # Vaccination data categorized by age groups
VACCINATION_SCHEDULE = [
        # Infants & Children
         {"age": 0, "name": "BCG, Hepatitis B (1st dose)", "notes": "Protects against tuberculosis and
hepatitis B"},
        {"age": 1, "name": "MMR (1st dose), Varicella (1st dose), Hepatitis A (1st dose)",
     "notes": "Protects against measles, mumps, rubella, chickenpox, and hepatitis A"},

        {"age": 1.5, "name": "DTP Booster (1st), Hib Booster, IPV Booster, Influenza (annual)",
     "notes": "Boosts immunity against diphtheria, tetanus, pertussis, polio, and flu"},
```

```
        {"age": 2, "name": "Typhoid (1st dose), Influenza (annual)",
     "notes": "Prevents typhoid and flu"},

        {"age": 3, "name": "MMR (2nd dose), Varicella (2nd dose), Hepatitis A (2nd dose), Influenza (annual)",
     "notes": "Provides long-term immunity against MMR, chickenpox, hepatitis A, and flu"},

        {"age": 4, "name": "DTP Booster (2nd), IPV Booster, PCV Booster, Influenza (annual)",
     "notes": "Further strengthens immunity against diphtheria, tetanus, pertussis, polio, pneumonia, and flu"},

        {"age": 5, "name": "Typhoid Booster, Meningococcal Vaccine, Influenza (annual)",
     "notes": "Continued protection against typhoid, meningitis, and flu"},

        # Adolescents
          {"age": 10, "name": "HPV Vaccine (Females only)", "notes": "Prevents cervical cancer", "gender": "female"},
        {"age": 12, "name": "Tdap Booster", "notes": "Protects against diphtheria, tetanus, pertussis"},
          {"age": 16, "name": "Meningococcal (MCV), Hepatitis A (2nd dose)", "notes": "Prevents bacterial meningitis"},

        # Adults
        {"age": 40, "name": "Shingles (Herpes Zoster)", "notes": "Prevents painful rash"},
        {"age": 50, "name": "Pneumococcal Vaccine", "notes": "Prevents pneumonia"},
        {"age": 60, "name": "COVID-19 Booster, Tdap Booster", "notes": "Immunity maintenance"},
    ]

# View for vaccination form page
def vacc_form(request):
    return render(request, "vacc.html")

# View for vaccination results page
def vacc_result(request):
```

```
    if request.method == "POST":
        age = int(request.POST.get("age"))
        phone_number = request.POST.get("phone_number", "")
        email = request.POST.get("email", "")

        # Missed vaccinations (ages less than the entered age)
        missed_vaccinations = [v for v in VACCINATION_SCHEDULE if v["age"] < age]

        # Upcoming vaccinations (ages greater than or equal to the entered age)
        upcoming_vaccinations = sorted(
            [v for v in VACCINATION_SCHEDULE if v["age"] >= age],
            key=lambda x: x["age"]
        )

        # Send SMS or Email Reminder
        if phone_number:
            sms_message = f"Reminder: Your upcoming vaccinations: {', '.join([v['name'] for v in upcoming_vaccinations])}."
            send_sms_reminder(phone_number, sms_message)

        if email:
            email_subject = "Vaccination Reminder"
            email_message = f"Dear user, \n\nYour upcoming vaccinations:\n" + "\n".join([f"- {v['name']}: {v['notes']}" for v in upcoming_vaccinations])
            send_email_reminder(email, email_subject, email_message)

        return render(request, "vacre.html", {
            "missed_vaccinations": missed_vaccinations,
            "upcoming_vaccinations": upcoming_vaccinations
        })
    else:
        return render(request, "vacc.html")
```

```
# Save Vaccination Reminder

# Send Reminder Notification (Decides whether to send via SMS or Email)
def send_reminder_notification(reminder):
    message = f"Hello {reminder.name}, your vaccination is scheduled for {reminder.next_vaccination_date}."

    if reminder.notification_mode == "sms":
        send_sms_reminder(reminder.contact, message)
    elif reminder.notification_mode == "email":
        send_email_reminder(reminder.contact, "Vaccination Reminder", message)

# Send SMS Reminder using Twilio

logger = logging.getLogger(__name__)
def send_sms_reminder(phone_number, message):
    client = Client(settings.TWILIO_ACCOUNT_SID, settings.TWILIO_AUTH_TOKEN)

    try:
        twilio_message = client.messages.create(
            body=message,
            from_=settings.TWILIO_PHONE_NUMBER,
            to=phone_number,
        )
        logger.info(f"✅ Twilio Message Sent: {twilio_message.sid}")
        return True
    except Exception as e:
        logger.error(f"❌ Twilio Error: {e}")
        return False

# Send Email Reminder
def send_email_reminder(email, subject, message):
    send_mail(
```

```
        subject,
        message,
        settings.EMAIL_HOST_USER,
        [email],
        fail_silently=False,
    )

def schedule_reminder(request):
    if request.method == "POST":
        form = ReminderForm(request.POST)
        if form.is_valid():
            reminder = form.save()

            # Schedule the task dynamically
            schedule, _ = CrontabSchedule.objects.get_or_create(
                minute=reminder.scheduled_time.minute,
                hour=reminder.scheduled_time.hour,
                day_of_month=reminder.scheduled_time.day,
                month_of_year=reminder.scheduled_time.month,
                timezone="Asia/Kolkata"  # Change this to your timezone
            )

            PeriodicTask.objects.create(
                crontab=schedule,
                name=f"Vaccination Reminder for {reminder.contact}",
                task="send_vaccination_reminder",
                args=json.dumps([reminder.contact, reminder.message, reminder.notification_mode]),
                one_off=True  # Ensures it runs only once
            )

            return redirect("success_page")  # Redirect to a success page

    else:
```

```
        form = ReminderForm()

    return render(request, "schedule_reminder.html", {"form": form})
  # Ensure you have this model

def send_otp(request):
    if request.method == "POST":
        try:
            # Handle JSON request
            data = json.loads(request.body)
            phone = data.get("phone", "").strip()

            if not phone:
                return JsonResponse({"error": "📌 Phone number is required!"}, status=400)

            if not phone.startswith("+") or len(phone) < 10:
                return JsonResponse({"error": "📞 Invalid phone number! Must include country code (e.g., +1234567890)."}, status=400)

            otp = str(random.randint(100000, 999999))  # Generate 6-digit OTP
            request.session["otp"] = otp
            request.session["phone"] = phone

            # Twilio Client
            client = Client(settings.TWILIO_ACCOUNT_SID, settings.TWILIO_AUTH_TOKEN)

            try:
                message = client.messages.create(
                    body=f"🔐 Your OTP is {otp}",
                    from_=settings.TWILIO_PHONE_NUMBER,  # Ensure this is a verified Twilio number
                    to=phone,
                )
```

```
            print(f"✅ Twilio Message Sent: {message.sid}")  # Debugging
            return JsonResponse({"message": "📩 OTP sent successfully!"})
        except Exception as e:
            print(f"❌ Twilio Error: {e}")  # Debugging
            return JsonResponse({"error": f"Twilio Error: {str(e)}"}, status=500)

    except json.JSONDecodeError:
        return JsonResponse({"error": "⚠ Invalid JSON request!"}, status=400)

return JsonResponse({"error": "❌ Invalid request method!"}, status=405)


def verify_otp(request):
    if request.method == "POST":
        data = json.loads(request.body)
        entered_otp = data.get("otp")
        stored_otp = request.session.get("otp")
        phone = request.session.get("phone")

        if entered_otp == stored_otp:
            # Check if phone number already exists in VerifiedNumber
            verified_number, created = VerifiedNumber.objects.get_or_create(
                phone_number=phone,
                defaults={"verified": True}  # Set verified=True only if it's new
            )

            if not created:
                return JsonResponse({"message": "✅ Phone number already verified!"})

            return JsonResponse({"message": "✅ Phone number verified successfully!"})
```

```
        return JsonResponse({"error": "❌ Incorrect OTP. Try again!"})

    return JsonResponse({"error": "Invalid request"})

def save_vaccination_reminder(request):
    if request.method == "POST":
        name = request.POST["name"]
        dob = request.POST["dob"]
        age = int(request.POST["age"])
        notification_mode = request.POST["notification_mode"]
        contact = request.POST.get("contact", None)

        if notification_mode == "sms":
            verified = VerifiedNumber.objects.filter(phone_number=contact, verified=True).exists()
            if not verified:
                return JsonResponse({"error": "Phone number not verified!"})

        # Fetch the next vaccination from database based on age
        next_vaccine = VaccinationSchedule.objects.filter(age_due=age + 1).first()



        # Schedule reminder
        # (Logic to schedule reminder using Celery or another method)

        return JsonResponse({"message": "Reminder set successfully!"})

    return render(request, "rem.html")
```

SEND-OTP.HTML

```
<form method="post" action="{% url 'send_otp' %}">
    {% csrf_token %}
    <input type="text" name="phone" placeholder="Enter your phone number" required>
    <button type="submit">Send OTP</button>
</form>
```

VERIFY-OTP.HTML

```
<form method="post" action="{% url 'verify_otp' %}">
    {% csrf_token %}
    <input type="text" name="otp" placeholder="Enter OTP" required>
    <button type="submit">Verify</button>
</form>
```

SAMPLE HOME PAGE

```
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome to the Homepage</h1>

    <!-- Buttons to Navigate -->
    <a href="{% url 'doctors:register' %}">
        <button>Register as Doctor</button>
    </a>

    <a href="{% url 'doctors:search_doctors' %}">
        <button>Find a Doctor</button>
    </a>
    <a href="{% url 'doctors:vacc_form' %}">
        <button>Check Vaccination Schedule</button>
    </a>
</body>
  </html>
```

REFERENCE

[1] S. A. Abdulmalek, M. A. Khan, and F. Alsubaie, "IoT-Based Healthcare Monitoring System for Improving Quality of Life," IEEE Access, vol. 9, pp. 14521–14533, 2021.

[2] R. R. Jain, P. Mehta, and S. Shah, "Health Monitoring System Using IoT," International Journal of Scientific Research in Engineering and Management, vol. 5, no. 6, pp. 22–27, Jun. 2021.

[3] M. Attaran, "Blockchain Technology in Healthcare: Challenges and Opportunities," International Journal of Healthcare Management, vol. 14, no. 1, pp. 1–10, Jan. 2021.

[4] E. Chukwu and R. Garg, "Digital Health Solutions and Data Interoperability," Health Informatics Journal, vol. 26, no. 3, pp. 1842–1855, 2020.

[5] H. A. Lee, Y. J. Kim, and J. H. Park, "Blockchain-Based Vaccine Passport Validation," Journal of Medical Systems, vol. 46, no. 1, pp. 1–9, 2022.

[6] Geetanjali Rathee et al., "Blockchain and Artificial Intelligence Integration for Healthcare Decision-Making," Computers in Biology and Medicine, vol. 134, 104113, 2021.

[7] "Django: The Web Framework for Perfectionists with Deadlines," Django Project. [Online]. Available: https://www.djangoproject.com/

[8] "Twilio Docs – Messaging API," Twilio. [Online]. Available: https://www.twilio.com/docs/sms/send-messages

[9] "Introduction to Blockchain Technology," IBM Blockchain. [Online]. Available: https://www.ibm.com/topics/what-is-blockchain

[10] "HIPAA Compliance Guide," HIPAA Journal. [Online]. Available: https://www.hipaajournal.com/hipaa-compliance-guide/

[11] A. A. Shaikh, N. Jamil, and M. Y. Javed, "Secure and Scalable IoT Healthcare Platform Using Cloud," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12345–12354, Aug. 2021.

[12] A. Alzahrani and N. Bulusu, "BlockIoTIntelligence: A Blockchain-Enabled Intelligent IoT Architecture with Artificial Intelligence," *Future Generation Computer Systems*, vol. 102, pp. 409–420, Jan. 2020.

[13] V. K. Sehgal, S. P. Saini, and D. K. Sharma, "Design of an IoT-Based Smart Health Monitoring System," *Procedia Computer Science*, vol. 167, pp. 2470–2479, 2020.

[14] A. B. Youssef, F. Abidi, and M. A. Serhani, "An Intelligent Patient Monitoring System Using Machine Learning Algorithms," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 6, pp. 6583–6596, Jun. 2021.

[15] M. A. Khan and S. Salahuddin, "Data Privacy and Security in eHealth Systems Using Blockchain," *Journal of Network and Computer Applications*, vol. 190, 103115, 2021.

[16] R. S. Sandhu, P. Samarati, and V. S. Subrahmanian, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[17] "Using the Django Admin Site," Django Documentation. [Online]. Available: https://docs.djangoproject.com/en/stable/ref/contrib/admin/

[18] "Best Practices for Securing Patient Data in Web Applications," OWASP. [Online]. Available: https://owasp.org/www-project-top-ten/

[19] S. Bhardwaj, A. Kaushik, and S. Deep, "AI-Driven Healthcare Monitoring Systems," *Biomedical Signal Processing and Control*, vol. 69, 102894, 2021.

[20] R. K. Jain and A. Dey, "Smart Reminder System for Medication Adherence Using Twilio API," *International Journal of Computer Applications*, vol. 183, no. 14, pp. 25 30, Aug. 2021.