CHAPTER-1

Introduction

This chapter discusses the origin of the problem, the problem description, basic definitions, and applications.

1.1 Origin of the Problem:

It is common for patients to face difficulties in finding a suitable doctor for their medical needs, and traditional healthcare systems may involve long waiting times and limited access to medical care. The origin of this project will be to address these issues and provide an efficient and convenient way for patients to access medical care online or offline, while also improving the overall patient experience.

The lack of a centralized system for managing patient care. Currently, there are multiple systems in place for different aspects of patient care, such as booking appointments, managing lab tests, and prescribing medications. This leads to a fragmented experience for patients and makes it difficult for healthcare providers to share information and collaborate on patient care. For example, patients may have to book appointments through one system, manage their lab tests through another system, and get their medications from yet another system. This can be confusing and time-consuming for patients, and it can also lead to errors and delays in care.

Healthcare providers also face challenges due to the lack of a centralized system. For example, doctors may have to access multiple systems to get a complete picture of a patient's health history, and they may have to manually enter information into multiple systems, which can be time-consuming and error-prone.

A centralized system for managing patient care would address these challenges by providing a single platform for all aspects of patient care such as booking slots for the consultation by selecting a profile from the existing applications, then user can select a doctor based on symptoms or specialist of choice , after slot time selection and payment, consultation gets confirmed and after the consultation patient can book medicines and lab tests online .This will make it easier for patients to manage their care, and it would make it easier for healthcare providers and patients to share information and collaborate on patient care. These features could further improve the quality of patient care.

1.2 Basic definitions and Background:

REACT JS:

React.js plays a pivotal role in shaping the user interface (UI) of a web application featuring intricate healthcare functionalities like booking consultations, ordering lab tests, purchasing medicines online, and selecting doctors based on symptoms or specializations, with consultation options spanning audio, video, or text. Employing a component-based architecture, React.js facilitates the creation of modular UI components for each distinct feature, ensuring code modularity and reusability. This approach streamlines development, making it easier to manage complex interactions within the application.

The dynamic user interfaces delivered by React.js are instrumental in providing a responsive and seamless experience for users engaging in healthcare-related tasks, such as filling out consultation forms or reviewing lab results. The library's virtual DOM optimizes updates, enhancing performance and contributing to a smoother user experience. For features involving real-time communication, React.js seamlessly integrates with technologies supporting audio, video, and text consultations, offering flexibility and versatility in user interaction.

React.js excels in state management, efficiently handling the dynamic data associated with healthcare applications. This is vital for functionalities like real-time doctor availability updates or tracking the progress of lab test orders. Additionally, the library easily integrates with backend services and APIs, enabling the web application to communicate with servers for tasks like appointment booking and processing online medicine orders. This backend integration ensures the seamless flow of data, supporting the accuracy and timeliness of information presented to users.

NEXT JS :

Next.js plays a crucial role in crafting the user interface (UI) of a web application featuring intricate healthcare functionalities like booking consultations, lab tests, online medicine orders, and doctor selection based on symptoms or specializations, with consultation options spanning audio, video, or text. Leveraging Next.js, a React framework, facilitates server-side rendering and provides a robust foundation for

building dynamic and responsive UIs. The framework's file-based routing simplifies the organization of pages and components, contributing to a more modular and scalable codebase.

One of Next.js' significant advantages is its ability to optimize performance through server-side rendering, ensuring faster page loads and improved user experiences. This is particularly crucial in healthcare applications where users engage in complex interactions, such as scheduling consultations or reviewing medical information. Next.js also supports static site generation, allowing for the pre-rendering of pages at build time, which is advantageous for content that doesn't frequently change, such as general information about consultations or lab tests.

The dynamic routing capabilities of Next.js are beneficial for features like selecting doctors based on symptoms or specializations. It enables the creation of dynamic pages that adapt to user input, providing a personalized and interactive experience. Additionally, Next.js seamlessly integrates with APIs and backend services, facilitating the communication necessary for functionalities like booking appointments or processing online medicine orders. This ensures the consistency and accuracy of information presented to users.

For features involving real-time communication, such as audio or video consultations, Next.js can integrate with third-party libraries or services designed for these functionalities. This ensures a seamless and responsive experience during consultations, meeting the diverse needs of users who may prefer different communication modes.

MONGO DB:

MongoDB is employed in creating schemas for the database in the context of healthcare applications developed using React Native for mobile and React.js for web. The flexible and schema-less nature of MongoDB, a NoSQL database, is particularly advantageous in modeling the diverse and evolving data structures associated with features like booking consultations, lab tests, online medicine ordering, and selecting doctors based on symptoms or specializations. MongoDB's ability to handle nested and complex data structures allows for the storage of information such as patient details, appointment

schedules, lab test results, and doctor profiles in a way that aligns with the dynamic requirements of healthcare applications.

The connection between MongoDB and the frontend applications is established through the use of drivers or libraries that facilitate interaction with the database. In the case of React Native and React.js, libraries like Mongoose for Node.js or the official MongoDB driver for JavaScript can be utilized. These libraries provide an interface for the frontend applications to perform operations such as querying, inserting, updating, and deleting data in the MongoDB database.

In the context of a healthcare application's frontend, MongoDB is accessed through API endpoints. These endpoints are created on the backend, which could be developed using Node.js, Express.js, or another server-side technology. The frontend applications, developed using React Native and React.js, make HTTP requests to these API endpoints to retrieve or manipulate data in the MongoDB database. For example, when a user books a consultation or orders medicine, the frontend sends a request to the backend, which in turn interacts with MongoDB to store or retrieve the relevant data.

The integration of MongoDB with the frontend applications is crucial for maintaining the consistency of information displayed to users across platforms. The dynamic nature of healthcare data, which includes appointment schedules, doctor availability, and patient records, is effectively managed in MongoDB, ensuring that the frontend applications can retrieve and display up-to-date information to users. The use of MongoDB in this context allows for scalability and adaptability, accommodating the evolving requirements of healthcare applications that feature diverse functionalities, including audio, video, or text consultations. In summary, MongoDB serves as a robust and flexible backend database solution, seamlessly connecting with React Native and React.js frontend applications to provide a reliable foundation for healthcare features such as booking consultations, lab tests, and online medicine ordering.

REACT NATIVE :

React Native serves as an integral tool in crafting the user interface (UI) for a mobile application with diverse healthcare features, including booking consultations, lab tests, and online medicine services, as well as the ability to select doctors based on symptoms or specializations with consultation options ranging from audio and video to text. At

the heart of React Native's efficacy is its component-based architecture, allowing developers to create modular and reusable UI components that correspond to distinct functionalities within the application. Each feature, such as consultation booking or medicine ordering, is encapsulated in a component, fostering code reusability, scalability, and maintainability.

The framework's capability to deliver a responsive and dynamic user interface is crucial for healthcare applications where users engage in complex interactions, such as submitting consultation requests or reviewing lab results. React Native achieves this through its virtual DOM, optimizing updates for better performance and ensuring a smooth user experience. For functionalities like audio or video consultations, React Native seamlessly integrates with third-party libraries, facilitating the inclusion of real-time communication features. This integration enables users to consult with healthcare professionals using their preferred mode—audio, video, or text—enhancing the versatility of the application.

React Native's state management mechanisms play a pivotal role in handling the dynamic nature of healthcare data. Managing states efficiently allows for real-time updates on doctor availability or tracking the status of lab test orders. Additionally, the framework facilitates the integration of backend services and APIs, enabling the mobile application to interact with servers for tasks such as booking appointments or processing online medicine orders. This backend integration is crucial for maintaining the seamless flow of data and ensuring the accuracy of information presented to users.

1.3 Problem Statement:

The project aims to develop a comprehensive healthcare management system through the development of a Web application and a Mobile Application. The system will streamline and improve the healthcare experience for users by providing various features such as appointment scheduling by selecting a doctor based on symptoms or specialists of users choice and after the successful completion of appointment doctor generates prescription that includes list of medicines and lab test, patients can use laboratory module and pharmacy module for booking medicines and lab tests.

The goal is to enhance the efficiency of healthcare delivery and make it more accessible and convenient for patients.

1.4 Applications:

A healthcare management system can have a wide range of applications across different healthcare settings, including hospitals, clinics, private practices, and nursing homes. Here are some of the potential applications of a healthcare management system:

Patient management: A healthcare management system can be used to manage patient records, including medical history, demographics, test results, and medications. It can also be used to schedule appointments, track patient visits, and manage billing and insurance information.

Electronic health records (EHRs): An EHR system can be integrated with a healthcare management system to provide a comprehensive view of a patient's health history. This can help healthcare providers make more informed decisions about diagnosis, treatment, and medication management.

Telemedicine: With the rise of telemedicine, a healthcare management system can be used to facilitate remote consultations between healthcare providers and patients. This can be especially useful for patients in rural areas or those who have difficulty traveling to appointments.

Inventory management: A healthcare management system can be used to manage inventory of medical supplies and equipment, ensuring that there is always adequate stock on hand and minimizing waste.

Analytics and reporting: A healthcare management system can provide valuable insights into patient care and operational performance. It can be used to generate reports on patient outcomes, quality measures, and financial performance, helping healthcare providers identify areas for improvement.

CHAPTER-2

Review of Literature

2.1 LITERATURE REVIEW:

The purpose of this literature review is to explore the current state of the art in healthcare management systems and identify the key challenges and opportunities facing the industry.

2.2 SUMMARY OF LITERATURE STUDY:

In recent years, there has been a growing interest in the use of data and analytics to improve healthcare management. Healthcare organizations are increasingly collecting and analyzing data from a variety of sources, such as electronic health records, claims data, and patient surveys. This data can be used to improve decision-making in areas such as resource allocation, clinical care, and patient satisfaction.

Some of the key trends in healthcare management research include:

The shift to value-based care: This is a shift away from paying for the volume of healthcare services provided to paying for the quality and outcomes of care. This is leading to a greater focus on preventive care, population health, and patient-centered care.

The rise of big data and analytics: Healthcare organizations are increasingly using data and analytics to improve decision-making in all areas of the organization.

The growing importance of patient engagement: Patients are becoming more involved in their own healthcare, and healthcare organizations are increasingly recognizing the importance of engaging patients in their care. The need for interprofessional collaboration: Healthcare is becoming increasingly complex, and there is a growing need for healthcare professionals to collaborate with each other to provide the best possible care to patients.

FOR WEB APPLICATION:

Table.2.1: Comparison of Web applications

DESCRIPTION	ZOCDOC	HEALTH GRADES	MAYO CLINIC	BOOK MY DOCTOR	IMEDS
Recommendation of doctors based on symptoms.	✓	×	×	×	✓
Medical packages are available for Patient.	×	✓	×	✓	✓
Virtual Consultations.	✓	×	×	✓	✓
Medical packages are available for Patients.	×	✓	×	✓	✓
Online lab test booking.	✓	✓	✓	×	✓
Record Consultation room.	×	×	×	✓	✓
Addition of multiple profiles for family members are available.	✓	×	✓	✓	✓

These are the features of the health care applications that were in use till date. In order to build a efficient health care application, we will be including additional features in our application such as booking appointments online by selecting doctors based on selected symptom or specialist of their choice. Booking appointment for themselves or any other relations. Mode of appointment can be customized according to user choice like text, audio and video. Consultation recording can be done. Health check up packages can be included for appointments. Insurance claim assistance is also available. Doctor profiles can be clearly reviewed for ratings before booking an appointment. Notifications and remainders are available and can be customized. Rescheduling and cancellation of appointments are available. Book medicines and lab tests online. And the whole application is also available for mobile interface.

FOR MOBILE APPLICATION:

Table.2.2.1: Comparison of Mobile applications1

DESCRIPTION	PRACTO	MY FAMILY DOCTOR	mFINE	IMEDS
Booking Appointments	✓	✓	✓	✓
Digital prescription will be accessed by doctors ,patients, lab operators, pharmacy	×	×	×	✓
Will get remainders for both patients and doctors and pharmacy and lab operators	×	×	×	✓
Medical packages are available for Patients	×	×	×	✓
Online pharmacy	×	×	×	✓

DESCRIPTION	PRACTO	MY FAMILY DOCTOR	mFINE	IMEDS
Video call, phone call, chat conversation are available	×	×	✓	✓
Addition of multiple profiles for family members are available	×	×	✓	✓
Will recommend the doctors based on symptoms	×	✓	✓	✓
Record the consultation room	×	×	×	✓
Online lab test booking	×	×	✓	✓
Remainders for taking medicines is present	×	×	✓	✓

Table.2.2.2: Comparison of Mobile applications2

PRACTO APP

- The Practo app [1] is a digital health platform designed to help patients with colorectal diseases.

- A search of the literature found limited research specifically related to the Practo app.
- However, there are studies on similar digital health platforms that provide insight into the potential benefits of such apps.
- For example, a study by Lai et al. (2020) found that a mobile health platform for patients with inflammatory bowel disease (IBD) led to improvements in disease management and patient satisfaction.
- Another study by Faleiro et al. (2020) found that a digital health platform for patients with chronic gastrointestinal disorders improved patient engagement and symptom monitoring.
- These findings suggest that the Practo app may also be beneficial for patients with colorectal diseases.

MY FAMILY DOCTOR APP:

- The My Family Doctor app is a telemedicine platform that allows patients to connect with doctors remotely.
- There is a growing body of research on telemedicine, which provides insights into the potential benefits and limitations of such platforms.
- For example, a systematic review by Flodgren et al. (2015) found that telemedicine can lead to improvements in patient outcomes, such as reduced hospital admissions and improved clinical parameters. Another study by Whitten et al. (2018) found that telemedicine can improve patient satisfaction and reduce healthcare costs.
- However, there are also limitations to telemedicine, such as concerns about patient privacy and the potential for misdiagnosis.
- Overall, the literature suggests that the My Family Doctor app has the potential to improve access to healthcare and patient outcomes, but further research is needed to fully evaluate its effectiveness and safety.

MFINE APP

- mfine's[3] online doctor consultation service allows you to connect with top-rated doctors from over 30 specialities, including general medicine, pediatrics, gynecology, dermatology, and more.
- Consultations can be done via chat, audio, or video call, and typically last for 15-20 minutes.
- mfine also offers a variety of health packages that include multiple doctor consultations and lab tests at a discounted price.
- In addition to online doctor consultations, mfine also offers a wide range of lab tests and health checks at home.
- mfine partners with NABL-accredited labs to offer over 1000 different tests and check-ups. The patients can book a test or check-up at your preferred time and date, and a lab technician will come to patient’s home to collect the samples.
- Reports will be uploaded to the app within 24-48 hours.
- mfine also offers a convenient way to order medicines online.
- Patients can simply upload the prescription to the app or select the medicines you need from the mfine pharmacy.
- mfine offers discounts on medicines and prescription refills, and delivers the medicines to the doorstep within 24-48 hours.

CHAPTER-3

Proposed Method

3.1 DESIGN METHODOLOGY

The Healthcare Management system is a group of two applications, Web application and mobile application. The web application will be installed on could and mobile application will be installed on smart phones with touch screen. The web application shall be able to process at least 100 requests per second. The web application shall not consume more than 40% of memory. The mobile application shall be supported on Android and iOS devices.

3.1.1 PRODUCT FUNCTIONS

The Healthcare Management System shall have following modules:

- Web application
- Mobile application
- Backend for the application

3.1.1.1 Web application:

The web application is the main application and shall provide APIs to be consumed by mobile applications. The web application shall also provide all features of mobile application to be accessed in any browser in a laptop or a computer. The web application shall be developed in Model View Controller architecture wherein there is a clear separation in presentation layer, business layer and data layer. The application shall be scalable to add more features in future based on requirement.

3.1.1.2 Mobile Application:

The Mobile application shall be available in Google play store or Apple store for downloading and installation. The app shall differentiate users based on login credentials (registered phone number). Same app shall be used by doctors, patients and lab operators. The app shall be user friendly, easy to navigate and supported on varied screen sizes. The app shall consume APIs provided by Web application. The app shall

support multiple users (family members) under one registered phone number and email Id.

3.1.1.3 Backend for the application:

Database shall be used for storing user data, transactional data, reports and case studies. The data stored in database shall be used for report generation.

3.1.1.3.1 User Characteristics

There are five types of users differentiated based on registered mobile number.

Doctor: The doctor user shall have access to appointment list, calendar, prescriptions and lab reports etc

User: User shall have access to schedule of appointments, scheduling lab tests, online consultations, consultation room, prescriptions, lab reports, reminders etc.

Lab admin: Lab admin shall have very limited access to appointments, uploading lab reports only.

Pharmacy Operator: Pharmacy operator shall be able to generate bills, add stock, update stock etc.

System User: The system user shall have complete access to all modules and the user can configure system level settings and module level settings.

Front desk Operator: The front desk operator shall have access to hospital management.

3.1.1.3.2 Constraints

- The application shall be easy to use, navigate and adoptable to users.
- The application shall be scalable to add new functionalities in future.
- The application shall be reliable and free of errors.

3.1.1.3.3 Assumptions and Dependencies

The solution implemented will be built using :

- Reactjs for front-end
- React Native for Mobile Application
- Nextjs and Nodejs for Backend
- MongoDB for database support.
- The UX design shall be completed by Avantel Ltd.
- The solution will cater for long – term storage requirements
- One or more super user will be assigned by each functional area

- System Admin has access to all modules . Front desk operator operates under System
- Admin

3.2 SYSTEM ARCHITECTURE DIAGRAM

The Architecture diagram of our work is displayed in figure below:

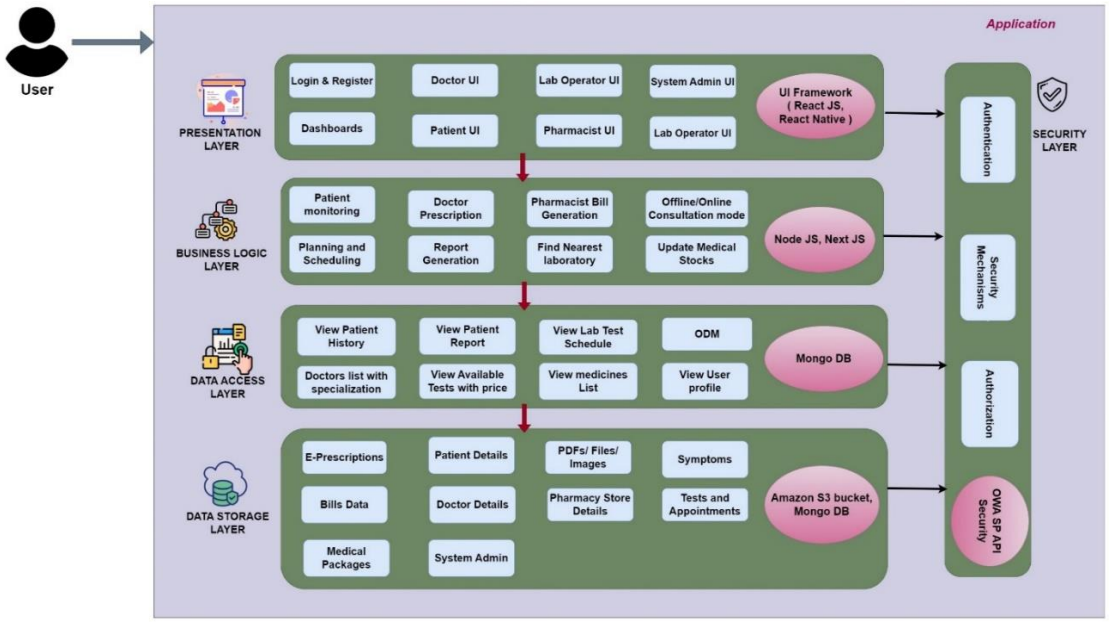

Fig.3.2: System Architecture Diagram

3.2.1 DATA ACCESS LAYER :

3.2.1.1 Overview

The Data Access Layer (DAL) in the Health Care Management System serves as the crucial bridge between the application's business logic layer and the database. Its primary function is to manage the retrieval, insertion, updating, and deletion of data in the underlying MongoDB database. By abstracting these operations into a separate layer, the system ensures that data interactions are handled efficiently, securely, and are easily maintainable.

3.2.1.2 Enhanced Data Management Capabilities

The DAL is engineered to provide advanced data management capabilities that are crucial for handling the complex and voluminous data inherent in healthcare systems. By leveraging MongoDB's flexible data model, the DAL allows for the dynamic addition and modification of schema without downtime or performance degradation, which is essential in the evolving field of healthcare where new data elements and structures may need to be accommodated quickly as treatment protocols and compliance requirements change.

3.2.1.3 Optimization and Performance

Efficiency in data retrieval and manipulation is a critical aspect of the DAL. It utilizes MongoDB's powerful indexing features to speed up the search operations, particularly for frequently accessed data such as patient records and appointment schedules. Indexes are strategically created on fields such as patient IDs, appointment dates, and doctor IDs to facilitate quick lookups and efficient query performance. This optimization ensures that the system can handle high volumes of concurrent requests from multiple users without a significant drop in performance, making it suitable for large healthcare facilities with substantial user traffic.

3.2.1.4 Data Consistency and Isolation

The DAL incorporates transactional support provided by MongoDB, which is especially important when dealing with operations that involve multiple updates across various documents or collections. For instance, updating a patient's medical record while simultaneously logging an appointment ensures that either both operations succeed or neither does, maintaining data integrity. This transactional support, with features such as atomicity and isolation, is crucial in a healthcare environment where data accuracy and consistency are paramount.

3.2.1.5 Security Integration

From a security standpoint, the DAL implements robust mechanisms to ensure data privacy and security in compliance with healthcare regulations, such as HIPAA in the United States. Access to the database is controlled through rigorous authentication and authorization checks, ensuring that only authorized personnel have access to sensitive

medical data. Data encryption both at rest and in transit is applied to protect personal and medical information from unauthorized access and potential data breaches.

3.2.1.6 Scalability and Flexibility

The design of the DAL is inherently scalable, thanks to MongoDB's distributed architecture. It supports horizontal scaling through sharding, allowing the database to handle more data by distributing it across multiple servers as the load increases. This scalability is crucial for the HCMS as it expands to accommodate more patients, doctors, and services. The DAL's flexibility and scalability ensure that the system can grow with minimal changes to the underlying architecture, thereby protecting investment and reducing future development costs.

3.2.1.7 Collection : activeappointments

This collection holds the active and upcoming appointments in the healthcare system. It captures detailed information about each appointment, including doctor and patient specifics, appointment details, and payment status. This collection is crucial for tracking the status of consultations and ensuring smooth operational workflows within the healthcare facility.

Fields:

_id: This field is a unique identifier for each appointment document stored in the MongoDB database. It is automatically generated by MongoDB and is crucial for indexing and retrieving appointment data efficiently.

Example: 65b20de9f5d7e794c761d89f

doctorId: Stores the unique identifier of the doctor associated with the appointment. This ID links the appointment to specific doctor details in the doctors collection.

Example: "658131f7b1788a10679cdfa8"

doctorName: Contains the full name of the doctor involved in the appointment, providing easy reference without needing to query the doctor's collection.

Example: "rizwanullah mohammad"

doctorImageurl: Holds the URL to the doctor's profile image, typically stored in an AWS S3 bucket. This is used in the user interface to display the doctor's image alongside appointment details.

Example:"https://imedslife-image-upload-bucket.s3.ap-south1.amazonaws.com/rizw…"

doctorSpeciality: An array containing the specialties of the doctor, which helps in categorizing appointments and assisting patients in finding the right specialist.

Example: Array

patientId: The unique identifier of the patient who has booked the appointment. This ID is used to link the appointment to the patient's detailed records in the patients collection.

Example: "65b164a6f6b6c0f83ab39d91"

patientdob: Stores the date of birth of the patient, which is crucial for medical records and age verification.

Example: "2003-07-19T18:30:00.000Z"

patientGender: The gender of the patient as specified in their profile, important for personalized care and statistical analysis.

Example: "male"

patientImageurl: Contains the URL to the patient's profile image, aiding in visual identification in the user interface.

Example:"https://imedslife-image-upload-bucket.s3.ap-south-1.amazonaws.com/rizw…"

paymentSuccessful: A boolean value indicating whether the payment for the appointment was successful, essential for financial management and record-keeping.

Example: true

appointmentTime: The specific time at which the appointment is scheduled, critical for both doctor and patient schedules.

Example: "14:20"

appointmentDate: The date on which the appointment is set to occur, used for scheduling and historical tracking.

Example: 2024-01-25T00:00:00.000+00:00

appointmentFor: The name of the person for whom the appointment is booked, which could be the patient or a dependent.

Example: "rizwan"

consultationType: Indicates the mode of consultation (e.g., in-person, video), important for preparing the necessary arrangements.

Example: "video"

patientSymptoms: An array that lists symptoms reported by the patient, aiding in preliminary assessments and readiness for the consultation.

Example: Array

relation: Describes the relationship of the person attending the appointment with the patient (e.g., self, spouse).

Example: "myself"

razorpay_order_id: An identifier for the payment order created with Razorpay, used for tracking and reconciliation of payments.

Example: "order_NSlNRtu9dO674y"

razorpay_payment_id: The payment ID provided by Razorpay, crucial for confirming payments and handling refunds if necessary.

Example: "pay_NSlNgUOWEvpm57"

razorpay_signature: A digital signature generated by Razorpay to ensure the integrity and authenticity of the payment transaction.

Example:

"d9127744ab9bb92452f3856f5571579790dc0bae2668ae591ab09f3eef3e82e0"

appointmentCompleted: A boolean indicating whether the appointment has been completed, used for updating records and follow-up actions.

Example: false

createdAt: The timestamp when the appointment record was initially created, important for auditing and tracking the creation of records.

Example: 2024-01-25T07:29:45.858+00:00

updatedAt: The timestamp of the last update made to the appointment record, used for tracking changes and maintaining record accuracy.

Example: 2024-01-25T07:29:45.858+00:00

3.2.1.8 Collection: doctorappointmentslots

This collection is crucial for managing the scheduling and availability of doctors within the Health Care Management System. It keeps track of all time slots booked and available for each doctor, ensuring that appointments are scheduled without conflicts and optimizing the use of doctor's time. This collection aids in the dynamic scheduling interface that patients interact with, offering real-time updates on available slots for efficient appointment setting.

Fields:

_id: This field serves as a unique identifier for each document in the collection, automatically generated by MongoDB. It is essential for indexing and efficient retrieval of appointment slot information.

Example: 658140cebac8ab36b2c9fba3

doctorId: Stores the unique identifier of the doctor to whom the slots pertain. This ID is crucial for linking the time slots to specific doctors in the system.

Example: "658131f7b1788a10679cdfa8"

date: Captures the specific date for which the appointment slots are being managed. This field is used to organize and retrieve availability based on daily schedules.

Example: "2023-12-19"

bookedSlots: An array that lists all the time slots that have already been booked for the specified date. This field is dynamic and updated in real-time as patients book or cancel appointments. It helps prevent double bookings and assists in visualizing a doctor's daily schedule.

Example: Array

createdAt: The timestamp when the document was created. This field is important for logging when the appointment slots were initially set up in the system.

Example: 2023-12-19T07:05:50.262+00:00

updatedAt: Reflects the last time any modifications were made to the document, such as adding or removing appointment slots. This ensures that the scheduling data is current and reflects any changes promptly.

Example: 2023-12-19T07:46:21.953+00:00

__v: A version key set by MongoDB that tracks the revision of the document. This internal field is used by MongoDB to handle document versioning and is automatically managed.

Example: Value not shown, but it increments on each update.

3.2.1.9 Collection: doctors

The doctors collection is integral to the Health Care Management System, storing comprehensive profiles of each doctor registered within the system. This collection is essential for allowing patients to search for and select doctors based on various criteria such as specialty, experience, and qualifications, thereby enhancing the patient experience and facilitating better healthcare outcomes.

Fields:

_id: Serves as the primary unique identifier for each doctor document in the database, automatically generated by MongoDB. This ID is crucial for indexing and linking the doctors with appointments and other relevant collections.

Example: 65b3ac03cba53573c4f23885

doctorId: A unique identifier assigned to the doctor within the healthcare system, used for internal tracking and linking across different system functionalities.

Example: "1ae0bd97-62ff-44b7-b3d0-49714ab6aa5e"

publicDoctorId: An additional unique identifier that may be used for more public-facing features, such as when interfacing with APIs that require a less sensitive identifier.

Example: "3a4c46d7-bc62-4574-bd37-10c4a0d17e6c"

firstname: Stores the first name of the doctor, used across the system to personalize communications and display information in user interfaces.

Example: "kalyanchakravarti"

lastname: Contains the doctor's last name, important for identification and documentation purposes.

Example: "yelavarti"

email: The professional email address of the doctor, used for system logins, communications, and notifications.

Example: "klyn.518@gmail.com"

emailVerified: A boolean indicating whether the doctor's email address has been verified, which is crucial for ensuring that communications reach legitimate addresses.

Example: false

phno: The doctor's phone number, used for contact and verification purposes.

Example: "9985111518"

phnoVerified: Indicates whether the phone number has been verified, ensuring that the contact information is current and accurate.

Example: false

dob: Date of birth of the doctor, used for verification and possibly for internal records concerning age-related policies.

Example: "1984-05-14T00:00:00.000Z"

gender: The gender of the doctor, which may be relevant for patient preferences and for statistical reporting.

Example: "male"

otp: Typically used for one-time password functionalities during authentication processes, important for security protocols within the system.

Example: null

education: An array containing information about the doctor's educational background, qualifications, and any certifications, crucial for validating credentials and expertise to patients.

Example: Array

about: A brief description about the doctor, providing patients with insights into the doctor's expertise, specialization, and personal approach to healthcare.

Example: "description about the doctor"

photo: URL to the doctor's professional photograph, used in the user interface to personalize doctor profiles.

Example:"https://imedslife-image-upload-bucket.s3.ap-south-1.amazonaws.com/rizw…"

certificate: Link to digital copies of the doctor's certificates, providing proof of qualifications and allowing for easy verification by the system or patients.

Example:https://imedslife-image-upload-bucket.s3.ap-south-1.amazonaws.com/RIZW…"

speciality: An array that lists the doctor's specialties, key for matching doctor expertise with patient needs.

Example: Array

practicename: The name of the practice or clinic where the doctor works, used for logistical and contact purposes.

Example: "kalyan"

yearsOfExp: The number of years the doctor has been practicing, important for assessing experience level.

Example: 12

consultationFee: The standard fee charged by the doctor for a consultation, relevant for patients during the selection process.

Example: "500"

languageKnown: An array of languages the doctor is fluent in, essential for patient-doctor communication.

Example: Array

registrationNumber: A unique number that officially registers the doctor with a medical board or similar regulatory body.

Example: "121"

currentAddress: The current residential address of the doctor, occasionally necessary for contact by the administration.

Example: "d-no 11-217 opp nandini hospital"

currentWorkAddress: The address of the doctor's current workplace, used for directing patients and for logistical arrangements.

Example: "d-no 11-217 opp nandini hospital"

newUser: Indicates whether the doctor is new to the system, useful for triggering onboarding processes and initial setups.

Example: false

3.2.1.10 Collection: doctortimeslots

This collection manages the scheduling times available for doctors, tracking available and booked time slots throughout each day. This is crucial for appointment scheduling, helping to prevent conflicts and ensuring efficient use of each doctor's time.

Fields:

_id: This field is the primary identifier for each document within the collection, uniquely identifying each time slot entry.

Example: Automatically generated by MongoDB, e.g., 658140cebac8ab36b2c9fba3

doctorId: Stores the unique identifier of the doctor to whom the time slots belong, linking the time slots to specific doctors in the system.

Example: "658131f7b1788a10679cdfa8"

date: Captures the date for which these time slots are applicable, organizing the slots into daily schedules.

Example: "2023-12-19"

bookedSlots: An array detailing all the times that have been booked for the specified date, used to display availability to patients and manage scheduling dynamically.

Example: Array

createdAt: The timestamp indicating when this time slot data was initially created in the database.

Example: 2023-12-19T07:05:50.262+00:00

updatedAt: The timestamp of the last update made to this document, ensuring that changes to the schedule are tracked and reflected in real time.

Example: 2023-12-19T07:46:21.953+00:00

patientdob: Date of birth of the patient, important for verifying the patient's age at the time of consultation.

Example: "2023-12-12T18:30:00.000Z"

patientGender: Gender of the patient as recorded in their profile, useful for demographic analysis and personalized care.

Example: "male"

patientImageurl: URL to the patient's profile image, used in the user interface to personalize the health records.

Example:"https://imedslife-image-upload-bucket.s3.ap-south-1.amazonaws.com/rizw…"

paymentSuccessful: Indicates whether the payment for the appointment was successfully processed, vital for financial tracking and auditing.

Example: true

appointmentTime: The time at which the appointment occurred, critical for scheduling accuracy and record-keeping.

Example: "14:40"

appointmentDate: The date on which the appointment took place, used for historical tracking and medical record management.

Example: 2023-12-21T00:00:00.000+00:00

appointmentFor: Specifies the person for whom the appointment was scheduled, particularly useful in family practice settings.

Example: "rizwanullah"

consultationType: The mode of the consultation (e.g., in-person, call, video), providing insights into the logistics of the appointment.

Example: "call"

patientSymptoms: An array that captures the symptoms reported by the patient, aiding in medical diagnosis and treatment planning.

Example: Array

relation: Describes the relationship of the individual attending the appointment to the patient, if different from the patient.

Example: "myself"

razorpay_order_id, razorpay_payment_id, razorpay_signature: These fields relate to the payment processing details, ensuring that financial transactions are securely recorded and traceable.

Examples:"order_NEteaN4B2hARd9","pay_NEtejahDukDD7R",

"c45e1870fca3fa306c93c769771ad3b139d812d68dac92aff1947fa90f5b2386"

appointmentCompleted: A boolean status that confirms the completion of the appointment, crucial for updating patient records and follow-up scheduling.

Example: true

createdAt, updatedAt: Timestamps reflecting when the appointment record was created and last updated, providing a timeline of medical interactions for compliance and quality control.

Examples: 2023-12-21T06:30:20.234+00:00

3.2.1.11 Collection: patientfamilydetailsschemas

This collection manages the detailed family records associated with each patient, allowing for shared management of health records within a family.

Fields:

_id: Unique identifier for each document, essential for database operations.

Example: Automatically generated by MongoDB.

patientId: Links family details to a specific patient, central for managing family health records.

Example: "65b164a6f6b6c0f83ab39d91"

familyMembers: An array containing identifiers and details of family members related to the patient, crucial for familial disease tracking and appointments.

Example: Array

createdAt, updatedAt: Timestamps for creating and updating family details.

Examples: 2024-01-24T19:27:34.625+00:00, 2024-01-26T14:03:09.286+00:00

3.2.1.12 Collection: patients

Stores comprehensive data about each patient, supporting all facets of healthcare management from registration through treatment.

Fields:

_id: Primary unique identifier for each patient document.

Example: 65b164a6f6b6c0f83ab39d91

patientId: A unique identifier used internally to link the patient across various system functions.

Example: "a07e11e2-7942-4bb1-914b-b90231fe53c1"

name: Patient's full name, used throughout the system for identification.

Example: "rizwan"

dob: Date of birth, important for medical treatment and age verification.

Example: "2003-07-19T18:30:00.000Z"

email: Contact email for communications and system notifications.

Example: "rizwanrockzz13@gmail.com"

phno: Phone number for contact and verification purposes.

Example: "7416263730"

imageurl: URL to the patient's profile image for identification in the user interface.

Example:"https://imedslife-image-upload-bucket.s3.ap-south-1.amazonaws.com/rizw…"

gender: Patient's gender, relevant for personalizing care and treatments.

Example: "male"

height, weight: Physical characteristics used for medical assessments.

Examples: 100, 100

address: Home address, important for billing and emergency contact.

Example: ""

newuser: Indicates if this is a newly registered patient.

Example: false

familyMembers: Details of the patient's family members, essential for family healthcare management.

Example: Array.

3.2.1.13 Collection: prescriptions

Maintains records of all medical prescriptions issued to patients, critical for medication management and compliance.

Fields:

_id: Unique identifier for each prescription record.

Example: 654e826f6fc9c4723e71199d

consultationId: Links the prescription to a specific medical consultation.

Example: "653914bfe903757ba00ba9ba"medicines,

labTests: Lists prescribed medicines and required lab tests, essential for treatment and follow-up.

Examples: Array, Array

revisitAfterDays: Indicates when a follow-up is needed, ensuring continuous care.

Example: 3

revisitAfter: Boolean indicating if a follow-up consultation is necessary.

Example: true

remarks: Doctor's notes providing additional instructions or dietary recommendations.

Example: "eat only green diet.no whites."

Collection: roles

Manages user roles within the system, defining access and permissions crucial for system security and functionality.

Fields:

_id: Unique identifier for each role document.

Example: 65b164a6f6b6c0f83ab39d93

phno: Phone number associated with a specific role, used for verification.

Example: "7416263730"

role: Defines the user's role within the system (e.g., patient, doctor, admin).

Example: "patient"

uniqueToken: A token used for session management or role-specific actions.

Example: ""

3.2.1.14 Collection: specialists

Stores information about medical specialists, facilitating referrals and specialist searches within the system.

Fields:

_id: Primary identifier for each specialist entry.

Example: Automatically generated by MongoDB.

title: The title or name of the specialty.

Example: "most selected issues"

listOfTitles: An array of sub-specialties or related medical fields.

Example: Array

3.2.1.15 Collection: patientfamilydetailsschemas

This collection manages the detailed family records associated with each patient, allowing for shared management of health records within a family.

Fields:

_id: Unique identifier for each document, essential for database operations.

Example: Automatically generated by MongoDB.

patientId: Links family details to a specific patient, central for managing family health records.

Example: "65b164a6f6b6c0f83ab39d91"

familyMembers: An array containing identifiers and details of family members related to the patient, crucial for familial disease tracking and appointments.

Example: Array

createdAt, updatedAt: Timestamps for creating and updating family details.

Examples: 2024-01-24T19:27:34.625+00:00, 2024-01-26T14:03:09.286+00:00

3.2.1.16 Collection: prescriptions

Maintains records of all medical prescriptions issued to patients, critical for medication management and compliance.

Fields:

_id: Unique identifier for each prescription record.

Example: 654e826f6fc9c4723e71199d

consultationId: Links the prescription to a specific medical consultation.

Example: "653914bfe903757ba00ba9ba"

medicines, labTests: Lists prescribed medicines and required lab tests, essential for treatment and follow-up.

Examples: Array, Array

revisitAfterDays: Indicates when a follow-up is needed, ensuring continuous care.

Example: 3

revisitAfter: Boolean indicating if a follow-up consultation is necessary.

Example: true

remarks: Doctor’s notes providing additional instructions or dietary recommendations.

Example: "eat only green diet.no whites."

3.2.1.17 Collection: roles

Manages user roles within the system, defining access and permissions crucial for system security and functionality.

Fields:

_id: Unique identifier for each role document.

Example: 65b164a6f6b6c0f83ab39d93

phno: Phone number associated with a specific role, used for verification.

Example: "7416263730"

role: Defines the user's role within the system (e.g., patient, doctor, admin).

Example: "patient"

uniqueToken: A token used for session management or role-specific actions.

Example: ""

3.2.1.18 Collection: specialists

Stores information about medical specialists, facilitating referrals and specialist searches within the system.

Fields:

_id: Primary identifier for each specialist entry.

Example: Automatically generated by MongoDB.

title: The title or name of the specialty.

Example: "most selected issues"

listOfTitles: An array of sub-specialties or related medical fields.

Example: Array

3.2.1.19 Collection: symptoms

Manages a list of common symptoms to aid in preliminary diagnoses and patient triage.

Fields:

id: Unique identifier for each symptom entry.

Example: 649bfa4d1990a201a4ba6151

title: The main symptom or symptom category.

Example: "most selected issues"

listOfTitles: An array detailing related symptoms or further classifications.

Example: Array

3.2.1.20 Collection: patientfamilydetailsschemas

This collection manages the detailed family records associated with each patient, allowing for shared management of health records within a family.

Fields:

_id: Unique identifier for each document, essential for database operations.

Example: Automatically generated by MongoDB.

patientId: Links family details to a specific patient, central for managing family health records.

Example: "65b164a6f6b6c0f83ab39d91"

familyMembers: An array containing identifiers and details of family members related to the patient, crucial for familial disease tracking and appointments.

Example: Array

createdAt, updatedAt: Timestamps for creating and updating family details.

Examples: 2024-01-24T19:27:34.625+00:00, 2024-01-26T14:03:09.286+00:00

3.2.1.21 Collection: patients

Stores comprehensive data about each patient, supporting all facets of healthcare management from registration through treatment.

Fields:

_id: Primary unique identifier for each patient document.

Example: 65b164a6f6b6c0f83ab39d91

patientId: A unique identifier used internally to link the patient across various system functions.

Example: "a07e11e2-7942-4bb1-914b-b90231fe53c1"

name: Patient's full name, used throughout the system for identification.

Example: "rizwan"

dob: Date of birth, important for medical treatment and age verification.

Example: "2003-07-19T18:30:00.000Z"

email: Contact email for communications and system notifications.

Example: "rizwanrockzz13@gmail.com"

phno: Phone number for contact and verification purposes.

Example: "7416263730"

imageurl: URL to the patient's profile image for identification in the user interface.

Example:"https://imedslife-image-upload-bucket.s3.ap-south-1.amazonaws.com/rizw…"

gender: Patient's gender, relevant for personalizing care and treatments.

Example: "male"

height, weight: Physical characteristics used for medical assessments.

Examples: 100, 100

address: Home address, important for billing and emergency contact.

Example: ""

newuser: Indicates if this is a newly registered patient.

Example: false

familyMembers: Details of the patient's family members, essential for family healthcare management.

Example: Array

3.2.1.22 Collection: prescriptions

Maintains records of all medical prescriptions issued to patients, critical for medication management and compliance.

Fields:

_id: Unique identifier for each prescription record.

Example: 654e826f6fc9c4723e71199d

consultationId: Links the prescription to a specific medical consultation.

Example: "653914bfe903757ba00ba9ba"

medicines, labTests: Lists prescribed medicines and required lab tests, essential for treatment and follow-up.

Examples: Array, Array

revisitAfterDays: Indicates when a follow-up is needed, ensuring continuous care.

Example: 3

revisitAfter: Boolean indicating if a follow-up consultation is necessary.

Example: true

remarks: Doctor's notes providing additional instructions or dietary recommendations.

Example: "eat only green diet.no whites."

3.2.1.23 Collection: roles

Manages user roles within the system, defining access and permissions crucial for system security and functionality.

Fields:

_id: Unique identifier for each role document.

Example: 65b164a6f6b6c0f83ab39d93

phno: Phone number associated with a specific role, used for verification.

Example: "7416263730"

role: Defines the user's role within the system (e.g., patient, doctor, admin).

Example: "patient"

uniqueToken: A token used for session management or role-specific actions.

Example: ""

3.2.1.24 Collection: specialists

Stores information about medical specialists, facilitating referrals and specialist searches within the system.

Fields:

_id: Primary identifier for each specialist entry.

Example: Automatically generated by MongoDB.

title: The title or name of the specialty.

Example: "most selected issues"

listOfTitles: An array of sub-specialties or related medical fields.

Example: Array

3.2.1.25 Collection: symptoms

Manages a list of common symptoms to aid in preliminary diagnoses and patient triage.

Fields:

id: Unique identifier for each symptom entry.

Example: 649bfa4d1990a201a4ba6151

title: The main symptom or symptom category.

Example: "most selected issues"

listOfTitles: An array detailing related symptoms or further classifications.

Example: Array

3.2.1.26 Collection: symptomtitlelists

Organizes detailed lists of symptoms for comprehensive symptom tracking and categorization.

Fields:

id: Primary identifier for each detailed symptom list.

Example: Automatically generated by MongoDB.

title: Descriptive title of the symptom list, aiding in categorization.

Example: "symptomtitlelists"

listOfTitles: An array containing detailed symptom descriptions or related terms.

Example: Array

3.2.2 Data Storage Layer

3.2.2.1 Overview

The Data Storage Layer in the Health Care Management System is critical for managing non-relational data such as images, documents, and other binary files. This layer primarily utilizes Amazon Web Services (AWS) S3 (Simple Storage Service), a scalable object storage service that offers industry-leading durability, availability, and performance. S3 is particularly suited to healthcare settings due to its robust security features, compliance certifications, and flexible data management capabilities.

3.2.2.2 Integration with AWS S3

AWS S3 is integrated into the healthcare management system to store a variety of content, including medical images, digital copies of healthcare records, insurance documents, and other patient-related files. The choice of S3 is due to its high durability, ensuring that data is replicated across multiple physical locations to prevent loss. Additionally, S3 provides easy scalability, making it a cost-effective solution for healthcare providers who require increasing storage space without significant upfront investment.

3.2.2.3 File Upload Procedure

The procedure for uploading files to AWS S3 involves several steps, designed to ensure data security and integrity:

User Authentication and Authorization: Prior to any file upload, users must authenticate against the healthcare system's user management subsystem. Authorization is then checked to ensure the user has permission to upload files, thus securing access based on roles.

File Selection and Verification: Users select files for upload via a web or mobile interface. The system performs checks to validate file types and sizes against predefined standards to prevent the upload of unsupported or potentially malicious files.

Generating Pre-Signed URLs: The healthcare system generates pre-signed URLs using AWS SDK. These URLs provide a secure way for users to upload files directly to S3 without exposing sensitive AWS credentials. The pre-signed URL is temporary and expires after a specified time, enhancing security.

Secure File Transfer: Files are uploaded directly to the S3 bucket using the pre-signed URL. Data transfer is secured using SSL/TLS, encrypting data in transit to protect personal and sensitive information from interception.

Confirmation and Metadata Storage: Upon successful upload, AWS S3 sends a confirmation back to the healthcare system. The system then records the metadata associated with the file in the MongoDB database, including the file name, storage location, and any relevant tags for retrieval. This metadata is crucial for efficiently managing and retrieving stored files.

Access Control and Security Settings: S3 buckets are configured with appropriate access control policies to restrict who can view or modify the stored files. Policies are set to allow only authenticated and authorized users to access the files, adhering to healthcare compliance requirements such as HIPAA in the U.S.

3.2.2.4 Procedure

Step 1: Set Up Your AWS S3 Bucket

Create an S3 Bucket:

- Log into the AWS Management Console.
- Navigate to the S3 service and create a new bucket.
- Choose a unique name for your bucket and select the AWS Region closest to your users to minimize latency.
- Ensure that the bucket has the appropriate permissions set to restrict public access unless specifically required for your application.
- Configure Bucket Policies:

- Set up bucket policies that define who can access the bucket and what actions they can perform. For instance, you might want to allow only authenticated users to upload files.
- enable versioning to keep track of and recover previous versions of files.

Step 2: Set Up IAM Permissions

- Create IAM User or Role:

- If your application is running on an AWS service like EC2, you can attach an IAM role to the instance. Otherwise, create an IAM user.
- Ensure that this IAM user or role has the necessary permissions to put objects in the S3 bucket. Use the least privilege principle to provide only the required permissions.
- Attach Policy to IAM Role/User:

- Attach a policy that allows actions like s3:PutObject on your specific bucket. Here is an example policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::your-bucket-name/*"
    }
  ]
}
```

Step 3: Generate a Pre-Signed URL

- Set Up AWS SDK:
- Install AWS SDK for your application's technology stack (e.g., AWS SDK for Python (Boto3), JavaScript (AWS SDK for JavaScript in Node.js), etc.).
- Configure the SDK with your AWS credentials and region.
- Generate Pre-Signed URL:Use the AWS SDK to generate a pre-signed URL for uploading files. Here is an example using Python (Boto3):

```
import boto3
from botocore.exceptions import NoCredentialsError

def create_presigned_url(bucket_name, object_name, expiration=3600):
    """Generate a presigned URL to share an S3 object"""
    s3_client = boto3.client('s3')
    try:
        response = s3_client.generate_presigned_url('put_object',
                                                    Params={'Bucket': bucket_name,
                                                            'Key': object_name},
                                                    ExpiresIn=expiration)
    except NoCredentialsError:
        print("Credentials not available")
        return None
    return response
```

- This function creates a URL that allows an HTTP PUT operation for the specified object_name in bucket_name, valid for expiration seconds.

Step 4: Upload File Using the Pre-Signed URL

- Upload from Client-Side:
- Use the pre-signed URL to upload a file directly from the client side, such as a web browser or mobile app. This can be done using standard HTTP requests. For example, in JavaScript:

```
const uploadFile = async (file, presignedUrl) => {
  const response = await fetch(presignedUrl, {
    method: 'PUT',
    body: file,
    headers: {
      'Content-Type': 'file.type'
    }
  });
  return response.status;
}
```

Step 5: Verification and Error Handling

- Verify Upload Success:
- After the upload, check the response status from the HTTP request. A successful upload returns a 200 OK status.
- Handle Errors:
- Implement error handling to manage cases where the upload fails due to network issues, expired URLs, or permission errors.

Step 6: Cleanup and Security Considerations

- Security Best Practices:
- Regularly rotate AWS credentials and review IAM permissions.
- Use HTTPS to encrypt the data transmitted to and from S3.
- Logging and Monitoring:
- Enable logging and monitoring on your S3 bucket to track access and changes, helping to ensure compliance and security.

3.2.2.5 Data Retrieval

Retrieving files from S3 is streamlined through the use of the stored metadata in MongoDB. When a file is needed, the system queries MongoDB to obtain the file's location and other access parameters. It then generates a secure, temporary URL that can be used to download the file directly from S3.

3.2.2.6 Backup and Disaster Recovery

AWS S3's versioning and cross-region replication features are employed to further enhance data safety and availability. Versioning allows for the recovery of overwritten or deleted files, while replication ensures that data is backed up in geographically distinct regions to provide disaster recovery capabilities.

3.2.2.7 Compliance and Security

AWS S3 complies with major regulatory frameworks, which is essential for storing medical data. Regular audits, encryption at rest using AWS KMS (Key Management Service), and detailed access logs further support compliance and help maintain a high standard of data security.

3.3 High Level Architecture :

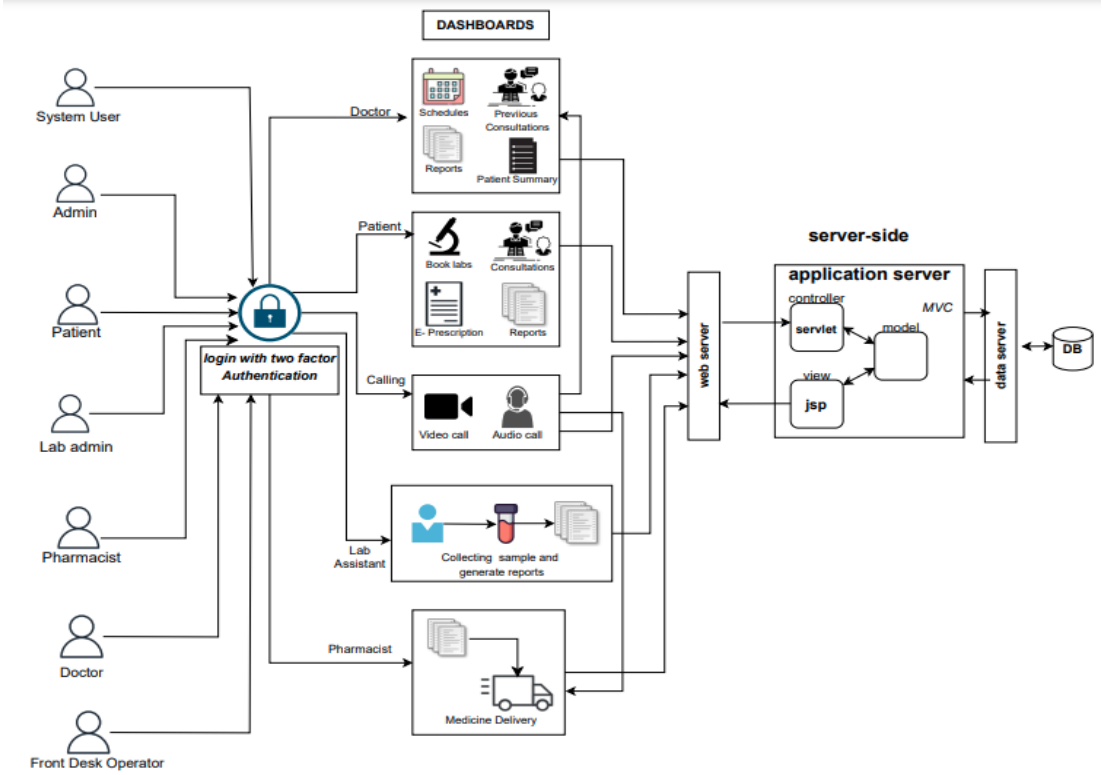

Fig.3.3: Architecture of the Overall Proposed System

Block diagrams are used to show the different parts of a system and how they interact with each other. The blocks in the diagram represent different components of the healthcare system, such as patients, doctors, pharmacists, and lab admins. The arrows between the blocks show how data and information flows between the different components.

Here is a brief explanation of each block in the diagram:

Patient: This block represents the patients who use the healthcare system. Patients interact with the healthcare system in a variety of ways, such as by scheduling appointments, visiting doctors, and filling prescriptions.

Doctor: This block represents the doctors who provide care to patients. Doctors order tests, diagnose illnesses, and prescribe medications.

Pharmacist: This block represents the pharmacists who dispense medications to patients. Pharmacists also provide counseling on how to take medications safely and effectively.

Lab admin: This block represents the laboratory administrators who oversee the testing process. Lab admins also ensure that test results are accurate and timely.

Front desk operator: This block represents the front desk operators who greet patients and help them schedule appointments. Front desk operators also handle billing and insurance matters.

Server: This block represents the computer servers that store and process data for the healthcare system. Servers store patient information, test results, and other important data.

Database: This block represents the database that stores patient information, test results, and other important data. The database is used to generate reports and track patient progress.

Application server: This block represents the application server that runs the healthcare system's software applications. The application server allows users to access the healthcare system's features and functionality.

Servlet: This block represents the servlet that handles requests from users and generates responses. Servlets are used to implement the healthcare system's business logic.

JSP: This block represents the JSP (JavaServer Pages) that generate dynamic HTML pages. JSPs are used to display data to users and collect input from users.

MVC: This block represents the Model-View-Controller (MVC) architecture that is used to develop the healthcare system. The MVC architecture separates the healthcare system's data model, presentation view, and control logic.

Consultation: This block represents the consulate that handles patient appointments. The consulate schedules appointments, confirms appointments, and sends reminders to patients.

Patient Summary: This block represents the patient summary that provides a summary of a patient's medical history. The patient summary is used by doctors to make informed decisions about patient care.

Dashboards: This block represents the dashboards that provide real-time data about the healthcare system. Dashboards are used by healthcare administrators to track key performance indicators (KPIs) and identify areas for improvement.

The arrows between the blocks in the diagram show how data and information flows between the different components of the healthcare system. For example, the arrow

from the Patient block to the Doctor block shows that patients provide doctors with information about their medical history. The arrow from the Doctor block to the Laboratory block shows that doctors order tests from the laboratory. The arrow from the Laboratory block to the Doctor block shows that the laboratory sends test results to doctors.

The arrows between the blocks also show how the different components of the healthcare system interact with each other. For example, the arrow from the Doctor block to the Pharmacist block shows that doctors prescribe medications to patients. The pharmacist then dispenses the medications to the patients.The block diagram provides a high-level overview of how a healthcare system works. The different components of the healthcare system interact with each other to provide care to patients. The diagram also shows how data and information flows between the different components of the healthcare system. the diagram may include arrows from the Laboratory block back to the Doctor block, signifying the flow of lab results which doctors use to diagnose conditions and monitor the effectiveness of treatments. This feedback loop is vital for the ongoing assessment of a patient's health and adjustment of their treatment plan as necessary.

The block diagram also underscores the role of technology in enhancing the efficiency and accuracy of these interactions. For instance, the use of health information technology (HIT) can streamline the flow of information from the Doctor block to the Pharmacy and Laboratory blocks, reducing errors and improving the timeliness of care delivery.

The block diagram is not just a static representation of a healthcare system's structure but a dynamic map of its operations, showing how each component contributes to a cohesive healthcare delivery process. It serves as an educational tool for healthcare professionals to understand their role within a broader context and as a design tool for administrators to improve system efficiency and patient care outcomes. Through such diagrams, the complex nature of healthcare delivery is made more comprehensible, facilitating better communication, planning, and coordination across different components of the system.

3.4 DATA BASE DESIGNS :

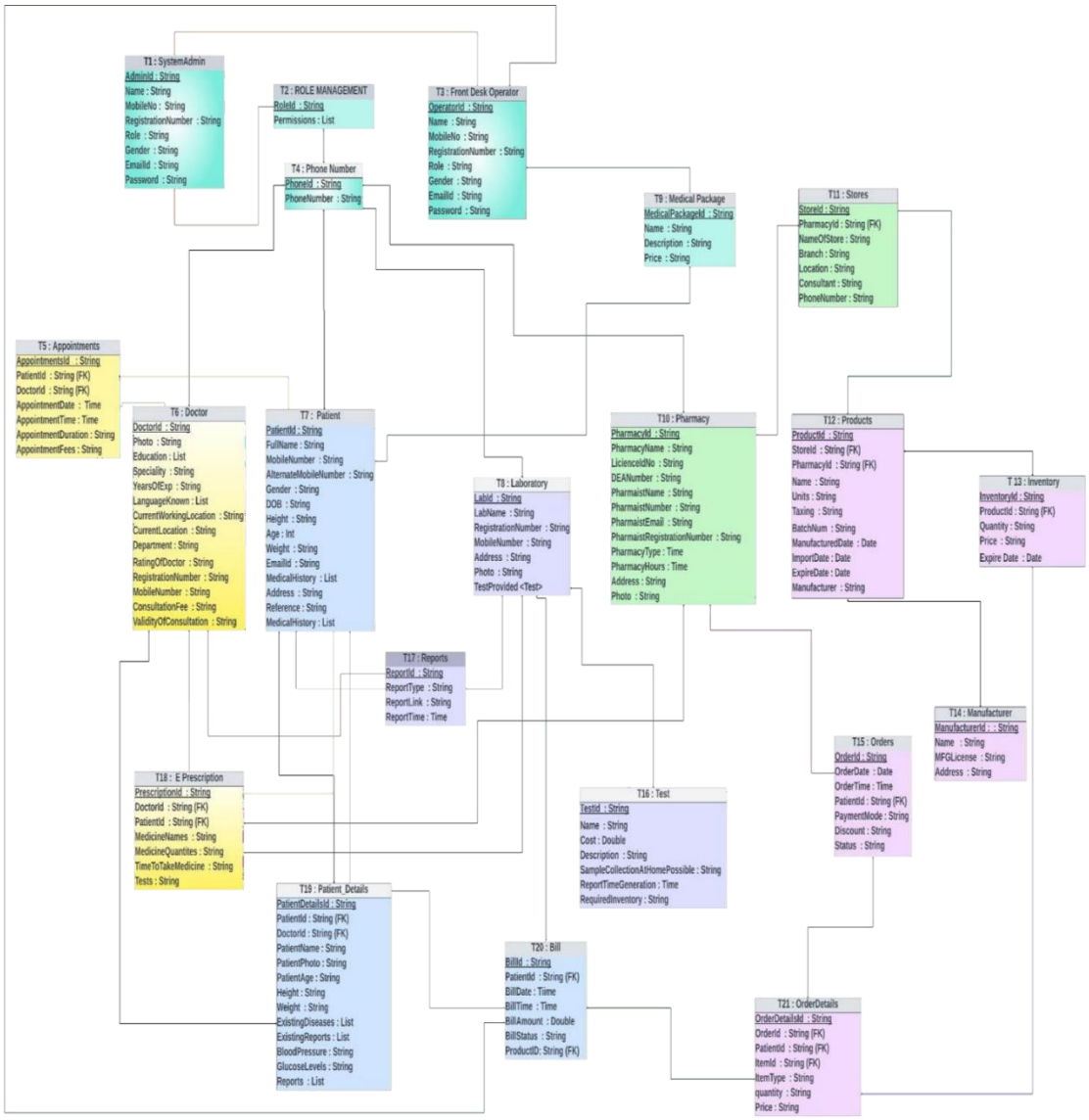

Figure 3.4 Data Base design

Database Design

- T1 : SystemAdmin : AdminId (PK) ,Name , MobileNo , RegistrationNumber , Role , Gender , EmailId, Password : This is the super user in the proposed system. Admin will have access to all the Modules.
- T2 : ROLE MANAGEMENT : RoleId (PK), Permissions : This table is used for permissions.
- T3 : Front Desk Operator : OperatorId (PK), Name , MobileNo , RegistrationNumber , Role , Gender, EmailId , Password : Bills and payments can also be accessed by the Front desk operator. The front desk operator will have

access to all and medical packages and discounts will be managed by front desk operator .

- T4 : Phone Number : PhoneId (PK), PhoneNumber : It will store all the phone numbers.and with accordinly give access to the particular user.
- T5 : Appointments : AppointmentsId (PK), PatientId (FK), DoctorId (FK) ,AppointmentDate, AppointmentTime , AppointmentDuration , AppointmentFees : The appointemnts booking will be stored in this table with reference to the patinetsid and doctorsID .
- T6 : Doctor : DoctorId (PK), Photo , Education ,Speciality , YearsOfExp ,LanguageKnown, CurrentWorkingLocation ,CurrentLocation , Department ,RatingOfDoctor RegistrationNumber , MobileNumber ,ConsultationFee ,ValidityOfConsultation : The individual doctor details will be stored here and Doctorid is the Foreign key. It has a relationship with E Precription table.
- T7 : Patient : PatientId , FullName , MobileNumber , AlternateMobileNumber , Gender , DOB , Height , Age : Int Weight , EmailId , MedicalHistory , Address , Reference , MedicalHistory .
- T8 : Laboratory : LabId , LabName , RegistrationNumber , MobileNumber , Address , Photo , TestProvided <TEST > : Apart from laboratory details, this table is connected to Test table.
- T9 : Medical Package : MedicalPackageId , Name , Description , Price : The medical packages which are dialy updated by the front desk operator will be stored in this and will be accessed by the patients and they can use the packages .
- T10 : Pharmacy : PharmacyId , PharmacyName , LicienceIdNo , DEANumber , PharmaistName PharmaistNumber , PharmaistEmail , PharmaistRegNumber , PharmacyType PharmacyHours ,Address , Photo : Maintains pharmacy operations. It has connections to the Store table.
- T11 : Stores : StoreId , PharmacyId , (FK) NameOfStore , Branch , Location , Consultant , PhoneNumber : The stores contains all the stores which are registred under the pharmacy table. The store table is connected to the products table.
- T12 : Products : ProductId , StoreId , (FK) PharmacyId , (FK) Name , Units , Taxing , BatchNum , ManufacturedDate , ImportDate , ExpireDate , Manufacturer : Contains all the Products like medicines etc., available in the stores. This table has relationship with inventory table.

- T13 : Inventory : InventoryId , ProductId , (FK) Quantity , Price , Expire Date : Comprises all medicines details. Like how many tables from a company and its details . This has relationship with ordered details table.
- T14 : Manufacturer : ManufacturerId : , Name , MFGLicense , Address : The manufacturer details of the medicines will be stored In this .
- T15 : Orders : OrderId , OrderDate , OrderTime ,PatientId , (FK) PaymentMode , Discount , Status , : Ther orders which are received from the patient will be stored in this .
- T16 : Test : TestId , Patient ID , (FK) Name , Cost : Double Description , SampleCollectionAtHomePossible , ReportTimeGeneration ,Required Inventory : The test table will contain all the tests provided by the laboratory.
- T17 : Reports : ReportId ,ReportType ,ReportLink,ReportTime : will be accessed by Doctors, lab operator, patient
- T18 : E Prescription : PrescriptionId (PK), DoctorId (FK) ,PatientId (FK), MedicineNames MedicineQuantites , TimeToTakeMedicine , Tests : Contains the prescription given by the doctors which will be accessible by the Patient, Pharmacy and Lab operator.
- T19 : Patient_Details : PatientDetailsId ,PatientId (FK), DoctorId (FK), PatientName , PatientPhoto , PatientAge , Height , Weight , ExistingDiseases, ExistingReports , BloodPressure, GlucoseLevels ,Reports : It has relationship with Doctor Table. By this the doctor can view the health conditions of the patients before consultation.
- T20 : Bill : BillId (PK), PatientId (FK), BillDate ,BillTime , BillAmount , BillStatus , ProductID, (FK) : Contains all the billing information. The bills belong to patient, pharmacy, laboratory and consultation Fees. It is connected to front desk operator table.
- T21 : OrderDetails : OrderDetailsId (PK), OrderId (FK) ,PatientId (FK), ItemId (FK) ,ItemType , quantity,Price : This contains all the orders received from the patient.
- Patient Module : Registration, Online consultation, Follow-up consultation, Schedule lab test, Previous consultations, Profile management

- Doctor Module : Registration, Today’s consultations, Previous consultations, Calendar, View Scheduled Appointments ,Receive Notifications, View lab reports, Prescribe lab tests, Prepare Prescription
- Pharmacy Module : Generate invoice, Payments, Stock availability etc.
- Lab operator Module : Today’s appointments, Previous appointments, Upload reports
- System Admin Module : Access to all the Modules
- Front desk Operator Module : Controlled by system admin and access accordingly, Managing other information like payments, medical packages, discounts.

3.5 WORK FLOW DESIGN

The workflow diagram delineates a sophisticated healthcare information system designed to streamline patient engagement from initial contact through treatment and follow-up. The system architecture integrates multiple operational facets, including user registration, secure login mechanisms, and patient-doctor interaction. At the onset, new users are systematically onboarded and existing users re-enter through a secure two-factor authentication, ensuring data confidentiality. Upon successful entry, the system affords patients the autonomy to select healthcare providers based on symptoms or preferred consultation mode, whether audio, video, or text-based.

Subsequent to provider selection, the front desk operator facilitates the booking process, interfacing with a comprehensive database that holds patient history and doctor availability. This enables tailored scheduling that takes into account previous medical reports and doctor schedules, culminating in a confirmed consultation appointment. The subsequent workflow encompasses financial transactions where a patient's financial responsibility is communicated through an itemized bill. Secure payment gateways collect payment information and confirm transaction completion, linking directly to the system's billing operations.

Post-consultation, the workflow transitions into report management and prescription fulfillment. Clinical reports are uploaded, contributing to a dynamic patient profile within the database. This patient-centric dossier may trigger ancillary healthcare actions

such as lab test bookings or medication orders. Lab operators interact with the system to provide test availability and manage inventory, while pharmacists engage with the pharmacy database to confirm drug availability and process orders, streamlining the prescription fulfillment process.

Interconnecting these components is a robust feedback mechanism, ensuring data integrity and real-time updates across the system. Each interaction—from booking to billing to prescription management—is meticulously logged, fostering a transparent and efficient healthcare delivery model. In essence, the diagram represents an integrated healthcare management ecosystem that is patient-centered, data-driven, and highly responsive to the multifaceted dimensions of patient care and administrative logistics. This system epitomizes the convergence of healthcare and information technology, striving for operational excellence while prioritizing patient welfare and data security.

The workflow diagram represents a robust healthcare management system that integrates multiple functions essential for modern healthcare delivery. It is patient-focused, data-centric, and likely utilizes advanced technology to ensure efficiency and compliance with health service standards. It encapsulates the complex nature of healthcare operations and the need for streamlined processes to enhance patient care and service delivery.

3.5.1 User Creation and Login:

New users are added to the system's database, while existing users can log in with two-factor authentication (2FA).

3.5.1.1 Booking a Consultation:

- Patients can select a doctor either by symptom description or mode of consultation (audio/video/text) using the front desk operator's database.
- The front desk operator accesses the doctor's availability and the patient's previous reports to help book a consultation.
- Once a consultation is booked, the schedule is confirmed with the doctor, and the consultation confirmation is sent to the patient.

3.5.1.2 Payment Process:

- After booking, the patient’s bill is generated.

- The patient then provides payment information and receives a payment confirmation once the billing is performed.

3.5.1.3 Post-Consultation Process:

- After the consultation, any reports generated are uploaded to the patient database.
- A patient summary is created which can lead to further actions such as booking lab tests or ordering medication.

3.5.1.4 Lab Tests:

- If a lab test is necessary, the lab operator checks for test availability in nearby laboratories using the laboratory datastore.
- The patient or front desk operator selects the required tests, and a booking for the lab test is made.
- Once tests are performed, the lab operator updates the database, which in turn reduces stock if any materials were used.

3.5.1.5 Prescriptions and Pharmacy:

- The patient or pharmacist can book medication, where the pharmacist checks the pharmacy database for availability.
- If the medication is available, an order is placed and then the stock is reduced accordingly.

3.5.1.6 Additional Workflow Links:

- Throughout the process, there are multiple feedback loops and data access points that ensure information is up-to-date and shared among relevant parties.
- For example:Payment information flows to and from the payment confirmation and billing processes. Patient reports and summaries can be used to update the patient database or schedule further consultations or tests.

3.5.1.7 Ancillary Operations:

The system also supports ancillary operations like confirming delivery of orders (medications, lab results), updating databases (laboratory, pharmacy, records), and managing inventory (reduce stock).

3.5.2 User Registration and Authentication :

3.5.2.1 System Admin:

Oversees user creation and maintains the system's integrity. They ensure new users are added correctly and existing users have a secure login mechanism.

3.5.2.2 Records Database:

Acts as a central repository for storing and retrieving patient records. This ensures historical patient data is accessible for medical consultations.

3.5.3 Consultation Process

3.5.3.1 Front Desk Operator Database:

This database contains information regarding the doctors schedules and specializations. The front desk operator uses this to facilitate the booking process.

3.5.3.2 Selection of Doctors:

Patients have the flexibility to choose a doctor based on symptoms or preferred mode of consultation. This patient-centric approach is critical for personalized care.

3.5.4 Payment and Billing

3.5.4.1 Payment Information:

The process involves secure handling of payment details, indicating a system that likely complies with standards such as PCI DSS for financial transactions.

3.5.4.2 Billing:

This step is crucial for the financial sustenance of the healthcare provider. It must be both efficient and transparent to maintain trust.

3.5.5 Medical Reporting and Follow-up

3.5.5.1 Upload Reports:

Post-consultation actions include uploading medical reports to the patient database, ensuring the continuity of care.

3.5.5.2 Patient Summary:

A comprehensive summary is essential for maintaining a complete health record, useful for future medical references or consultations.

3.5.6 Laboratory Testing

3.5.6.1 Laboratory Datastore:

This component holds information on available tests, which can be complex given the variety of medical tests available.

3.5.6.2 Book Lab Test:

Scheduling tests must be timely and accurate, reflecting the urgency and necessity of the tests prescribed by the healthcare provider.

3.5.7 Medication Management

3.5.7.1 Pharmacy Database:

Contains a list of medications, their availability, and possibly interactions, which is vital for dispensing the right medication.

3.5.7.2 Place Order:

Indicates a system linked with inventory management, ensuring medications are in stock when needed by patients.

3.5.7.3 System Integration

The workflow diagram shows how each component is interconnected. It's not just about individual tasks but also about how these tasks communicate with one another to form a cohesive system.

3.5.7.4 User-Centric Approach

Throughout the workflow, there is a clear emphasis on the patient's experience, with multiple touchpoints for patient engagement and feedback.

3.5.7.5 Data Privacy and Security

The system appears to have security measures in place, such as 2FA for user login, which is critical given the sensitive nature of healthcare data.

3.5.7.6 Administrative and Operational Efficiency

The diagram suggests a system designed for operational efficiency. For example, the reduction in stock is automated following lab tests, which implies a system that supports inventory management.

3.5.7.7 Technological Infrastructure

The workflow implies a sophisticated technological infrastructure that supports various functionalities such as video consultations, real-time updates to databases, and secure payment processing.

3.5.7.8 Compliance and Standards

The system would need to comply with various healthcare regulations, such as HIPAA in the United States, which governs the privacy and security of patient information.

3.5.7.9 Scalability and Maintenance

Given the complexity of the workflow, the system likely includes features for scalability to accommodate a growing number of users and maintenance routines to ensure the continuous, smooth operation of all components.

3.5 WORK FLOW DESIGN :

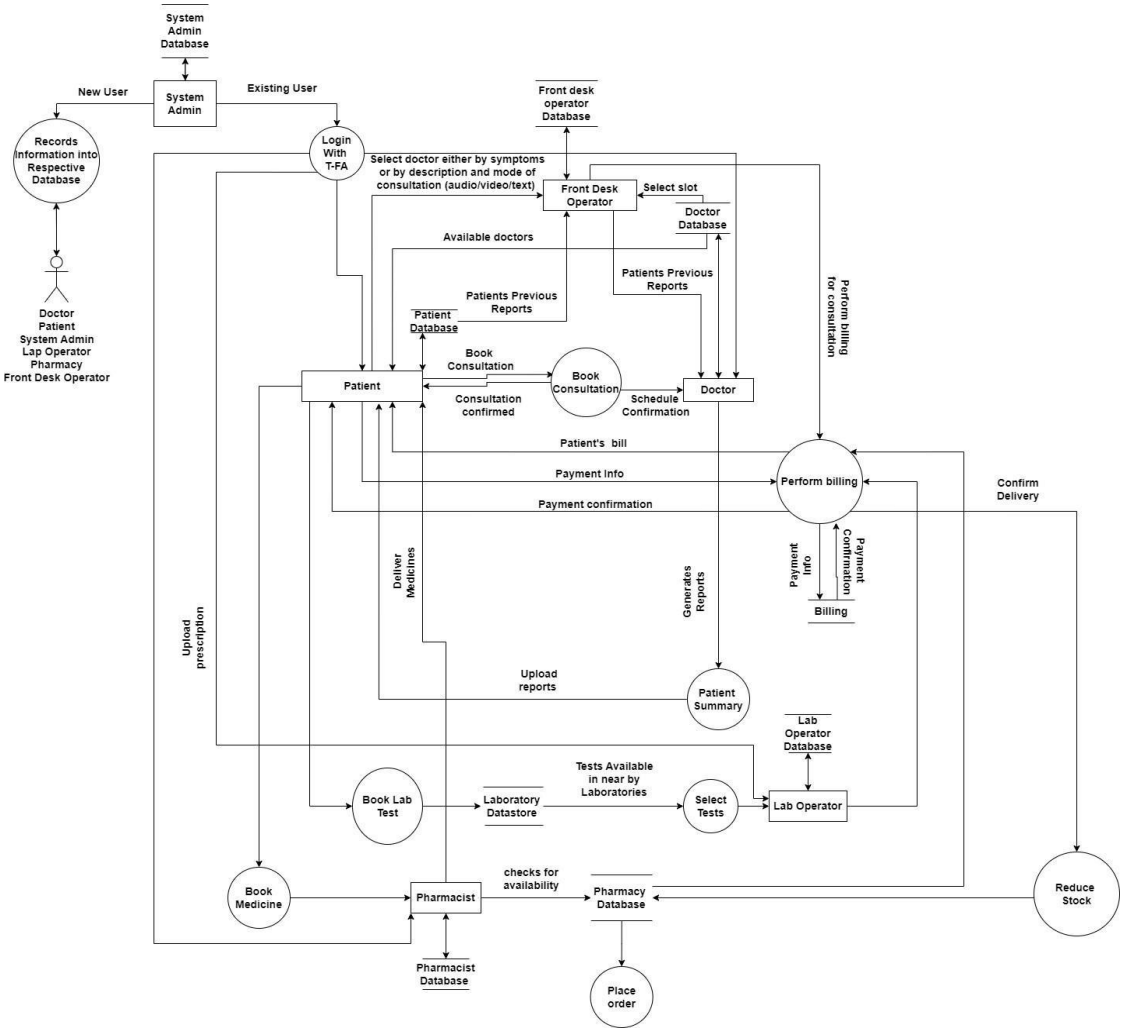

Figure 3.5 WORK FLOW DESIGN

3.6 IMPLEMENTATION STEPS

Step 1: Install Visual Studio Code

- Download and install Visual Studio Code from the official website.
- Open Visual Studio Code after installation.

Step 2: Install Node.js

- Download and install Node.js from the official website (https://nodejs.org/).
- Ensure that Node.js and npm (Node Package Manager) are installed correctly.

Step 3: Install Expo CLI

- Open your terminal or command prompt.
- Install Expo CLI globally by running: npm install -g expo-cli.

Step 4: Create a New React Native Project

- Create a new React Native project by running: expo init MyProjectName.
- Follow the prompts to choose a template and provide your project's name.

Step 5: Navigate to the Project Directory

Use the cd command to navigate to your project directory: cd MyProjectName.

Step 6: Start the Development Server

- Start the Expo development server by running: expo start.
- This will open a web page in your default browser with a QR code.

Step 7: Set Up Expo on Your Mobile Phone

- Install the Expo Go app on your mobile phone from the app store.
- Open the Expo Go app and scan the QR code displayed on the web page opened in the previous step.

Step 8: Implement the Mobile Application

- Open Visual Studio Code and open your React Native project folder.
- Begin implementing your mobile application using React Native, based on your UI designs.
- Save the code and check the Expo Go app on your mobile phone

CHAPTER-4

Results and Observations

4.1 DATA ACCESS LAYER :

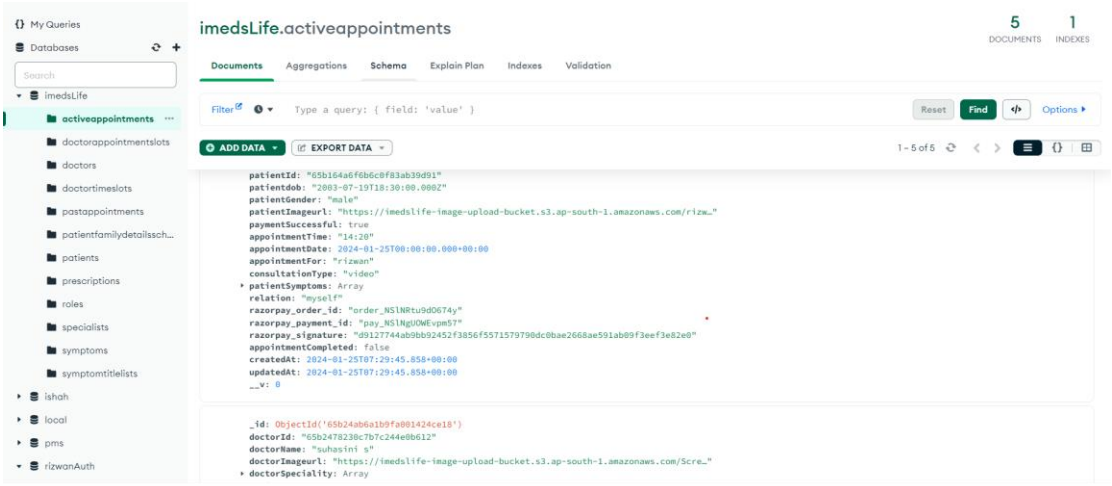

Fig 1. Mango DB database

The left panel lists the collections within a database named imedsLife, including activeappointments, doctorappointmentlots, doctors, doctortimeslots, pastappointments, patientfamilydetailssch..., patients, prescriptions, roles, specialists, symptoms, symptomtitlelists, along with other collections not fully visible.

The main panel displays a document from the activeappointments collection. Fields in the document include patientId, patientdob, patientGender, patientImageUrl, paymentSuccessful, appointmentTime, appointmentDate, appointmentFor, consultationType, patientSymptoms, relation, razorpay_order_id, razorpay_payment_id, razorpay_signature, appointmentCompleted, createdAt, updatedAt, and a version number (__v). There are placeholders for arrays, indicated with [Array], and some fields have been expanded to show their values. Below this document, there is another document that belongs to a different collection (possibly doctors), with fields like _id, doctorId, doctorName, doctorImageUrl, doctorSpeciality, and more that are not fully visible in the image.The document details for activeappointments indicate various information about an appointment, including patient identifiers, their date of birth, gender, an image URL, payment status, and

timestamps for the creation and modification of the record. There is also reference to a

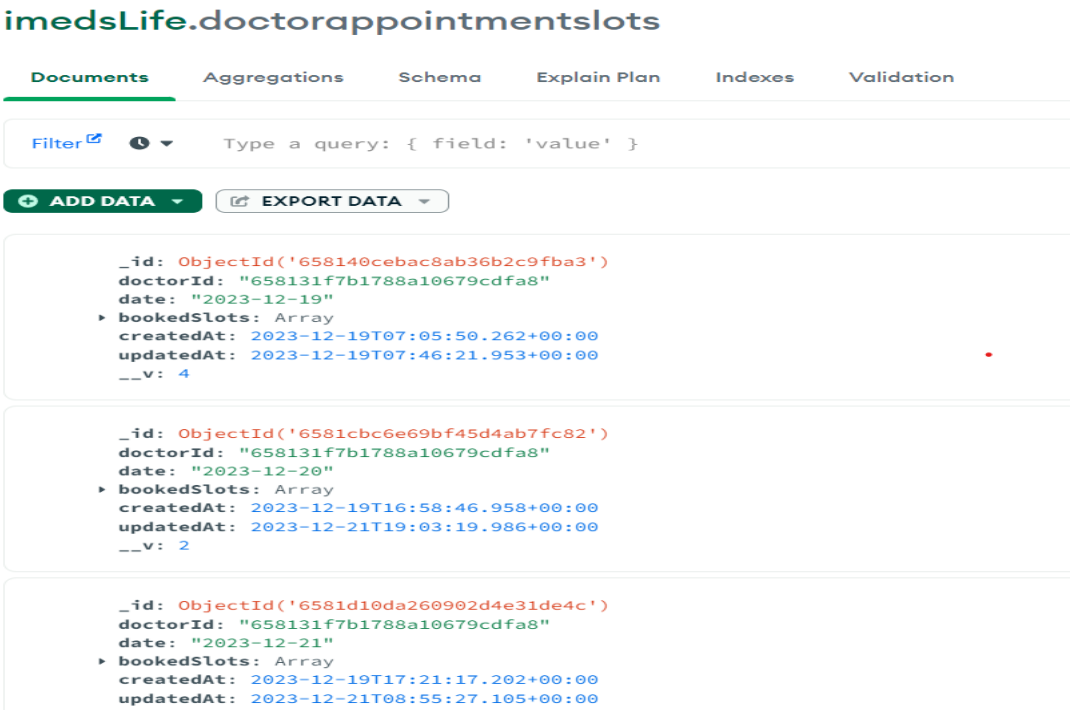

Fig 2 Doctor Appointment Collection

payment gateway with identifiers related to Razorpay, an Indian payment gateway.

imedsLife.doctorappointmentlots collection within a MongoDB database management system. The interface highlights three distinct records (documents) from the collection, each record possessing a unique Object Identifier (_id) and associated with a specific doctorId. Each document lists a date, representing the day for which appointment slots are recorded. An Array placeholder suggests that the bookedSlots field contains multiple entries, although their specifics are not visible in the image. Additionally, two timestamps are present: createdAt, marking the creation time of the record, and updatedAt, indicating the most recent update time. The interface also features a versioning field (__v) which suggests the version of the document schema. The screenshot exemplifies a database interface designed for managing and tracking doctor appointment slots, showcasing the database's structure and capabilities for organizing such data. The interface also includes options for filtering, adding, and exporting data, alongside a query bar for searching within the collection.

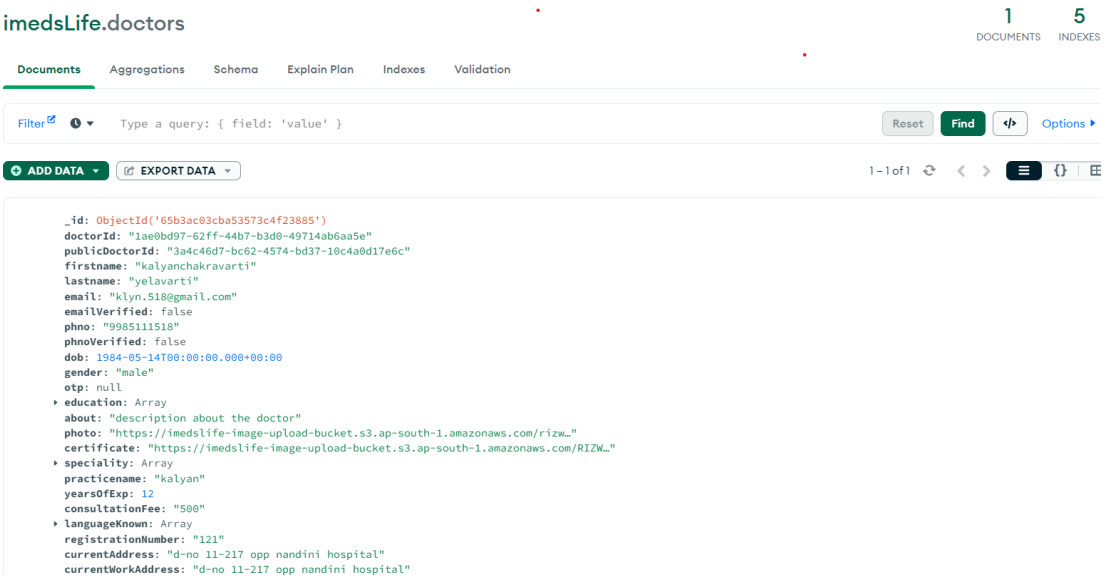

Figure 3 Doctors collection

The image displays a section of the imedsLife.doctors collection within a MongoDB database management interface. This particular document contains comprehensive information about a doctor. The fields include a unique identifier (_id), the doctor's ID (doctorId), and a publicly shareable doctor ID (publicDoctorId). Personal information such as the doctor's full name (firstname and lastname), email address, phone number, and date of birth (dob) are visible. There are also fields indicating whether the email and phone have been verified (emailVerified, phoneVerified), gender, and the timestamp for document creation (createdAt). The document provides additional professional details of the doctor such as a brief bio (about), a profile photo URL (photo), educational background (education), speciality, the name of the practice (practiceName), years of experience, consultation fee, languages known, registration number, and the current workplace address (currentWorkAddress). It is noted that speciality and languagesKnown are arrays, which may contain multiple items, and the actual data for these fields is not shown in the screenshot. The interface features controls such as filtering and exporting data, and there are options for performing database queries displayed at the top.

4.2 DATA STORAGE LAYER

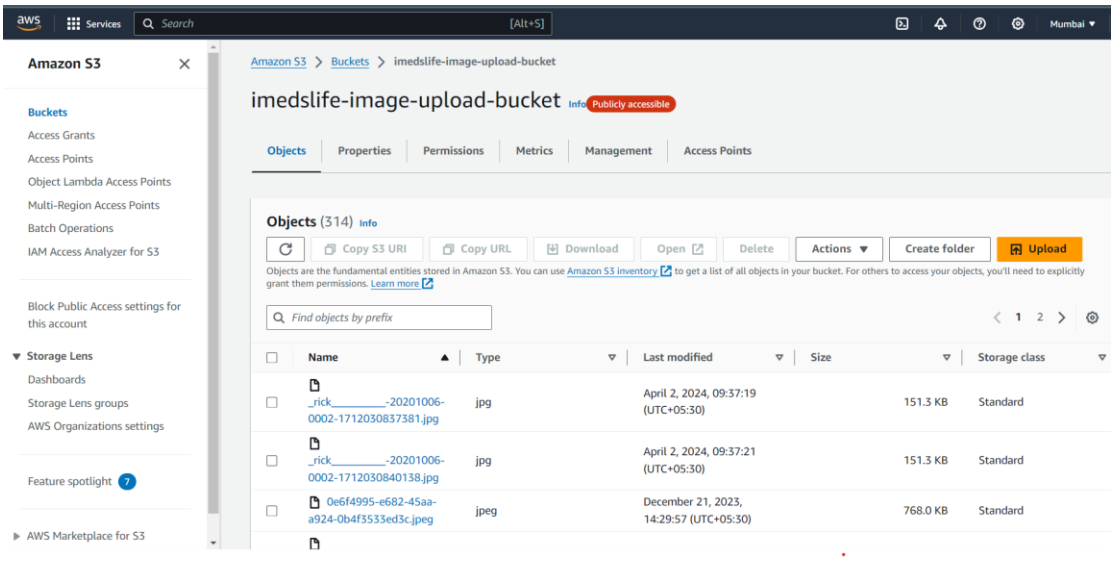

Figure 4 Data storage in AWS S3

The image depicts the Amazon Web Services (AWS) management console focused on the Simple Storage Service (S3). Specifically, it shows an S3 bucket named "imedsLife-image-upload-bucket," which is indicated as being publicly accessible. In the bucket, there is a list of objects, presumably files, which are displayed with their names, type (all are JPEG images), the last modified date and time, their size in kilobytes, and the storage class, which is marked as "Standard" for the listed objects. The names of the files follow a pattern but are partially obscured, with visible fragments suggesting they may contain identifiers or dates. Notably, two files with similar names, distinguished by a trailing identifier, were modified on April 2, 2024, and are the same size. Another listed file was last modified on December 21, 2023, and is significantly larger. This interface is part of AWS's cloud storage solution that provides users with the capability to store and retrieve any amount of data at any time. It is widely used for various purposes, such as hosting website content, backups, and archiving. The navigation pane on the left-hand side offers a range of options including creating buckets, accessing permissions, and viewing storage analytics, signifying the extensive features provided by AWS S3 for managing and monitoring stored data.

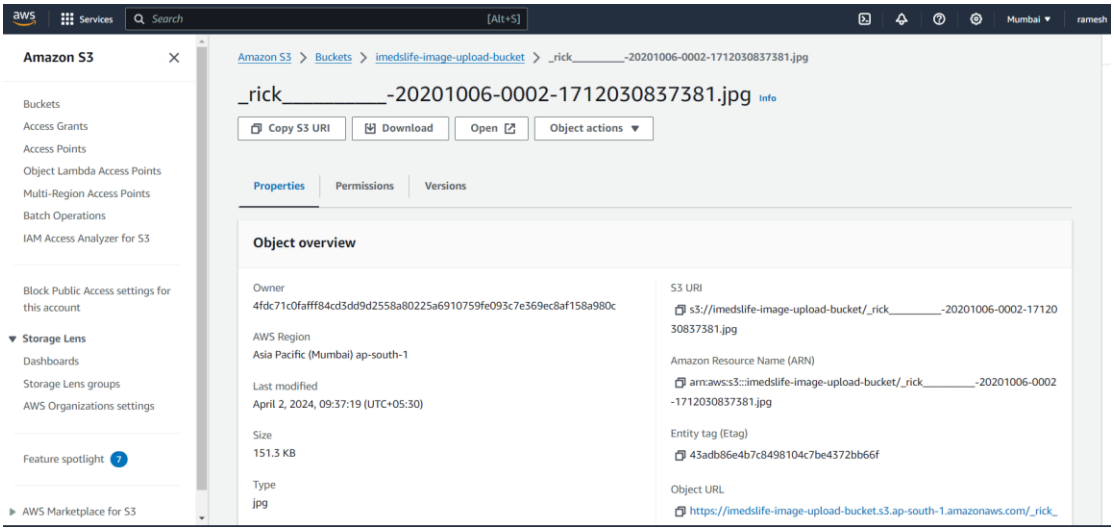

Figure 5 aws file image in aws s3

The image is a screenshot from the Amazon Web Services (AWS) S3 console, presenting the details of a specific object within the S3 bucket named 'imedsLife-image-upload-bucket'. The object in question is a JPEG image file named 'rick___-20201006-0002-1712030837381.jpg'. The user interface provides a detailed 'Object overview', including the 'Owner' identification, the 'AWS Region' where the bucket is hosted (Asia Pacific, Mumbai, ap-south-1), and the date and time the file was last modified (April 2, 2024, 09:37:19 UTC+05:30). The size of the file is listed as 151.3 KB. The screenshot also shows the S3 URI, Amazon Resource Name (ARN), Entity tag (Etag), and an Object URL, which provides a direct link to the file. These details are crucial for managing and referencing the object within the AWS ecosystem. On the left panel of the console, there are navigation options to access various S3 features, such as Buckets, Batch Operations, and Storage Lens, as well as settings for IAM Policies and Public Access. The account holder appears to be 'ramesh', as seen in the top right corner of the console.

CHAPTER-5

Conclusion and Future Study

5.1CONCLUSION:

In conclusion, the development of an online consultation website is a valuable and effective solution for providing convenient and accessible healthcare services. This website offers numerous benefits for both healthcare professionals and patients as well and the online consultation website prioritizes privacy and security. By implementing robust data encryption and secure server systems, it ensures the confidentiality of patient information and medical records. Patients can confidently share sensitive details with healthcare providers, fostering trust and confidentiality.

Lastly, the online consultation platform promotes efficiency and productivity. Healthcare professionals can manage their schedules more effectively, reducing waiting times and optimizing their workflow. Patients experience reduced waiting times and enjoy the convenience of accessing healthcare services from the comfort of their homes.

FUTURE STUDY:

Phase-1:

Developing the Web Application for the Health Care Management System.

Phase-2:

Developing Mobile Application for the Health Care Management System.

Phase-3:

Apply Authentication & Security in Web application and Mobile application. Identify and handle OWASP API Security the Top 10 Vulnerabilities 2019 in APIs exposed by the web application.

Implement TLS v1.2 in APIs for securing data transfer between web server and mobile/web application. Implementation of logging and registration for all users in the proposed system. Design registrations, schedules, calendars, e-prescription through dashboard.

References

[1] International Journal of Healthcare management, Volume 14, Issue 4 (2021). Research on health administration, healthcare system, e-Health, patient satisfaction, healthcare marketing, medical tourism, health policy and reform.

[2] Hospital management system using web technology ISSN 0193 4120, May 2020: Kotapati Saimanoj, Grandhi Poojitha, Khushbu Devendra Dixit, Laxmi Jayannavar.

[3] Hospital Management and Control System ISSN 25158260, Volume 7, June 2020 (European Journal of Molecular & Clinical Medicine): Ashmita Gupta, Ashutosh Niranajan, School of Medical Sciences & Research, Uttar Pradedesh.

[4] HAMS: An Integrated Hospital Management System to Improve Information Exchange, CISIS- June, 2020: Francesco Lubrano, Federico Stirano, Gluseppe Varavallo, Fabrizio Bertone, Olivier Terzo.

[5] Hospital management System (International Research Journal of Engineering & Technology)- Volume: 07 Issue: 04 | April 2020: Pranjali Anpan, Roshni Udasi, Susneha Jagtap, Shon Thakre, Chalika Kamble.

[6] Review on the Smart Hospital Management System Technologies: RST Journal, Spring 2019: Najeh Lakhoua.

[7] Hospital Management System – International Journal for Research in Engineering Application & Management: M. Sowmya, D. AnilChandraVarma, M.Sailaja, M. Venugopalarao, T. Prasanth.