Electronic Medical Record Web Application(Group-3)

Shah Javed Hossain
ID: 1820290
Section: 04

Prantika Roy
ID: 2021040
Section: 04

Munabbirul Saqlain
ID: 1821013
Section: 01

Mohammad Sufyan Rahman
ID: 1920498
Section: 07

December 17, 2023

1 Introduction

In today's ever-changing healthcare market, efficient management of patient medical information is critical to providing high-quality and tailored treatment. With the introduction of technology, creative ways to improve communication, expedite healthcare procedures, and improve patient outcomes have been developed. Our report, which is in line with this trend, centers on the launch of a web application that aims to completely transform the tracking and management of patient medical records. In addition to centralizing patient data, this web application offers a comprehensive platform that makes it easy to access vital information including appointments, prescriptions, patient medical history and test results. Doctors, patients, hospital/ clinic receptions, labs and pharmacies will have access to use the system. By addressing the issues with current record-keeping practices, the adoption of this web application seeks to promote a more effective, safe, and cooperative approach to healthcare management.

2 Literature Review

This report's Literature Review section critically reviews existing research, scholarly articles, and industry best practices to provide a thorough overview of patient record tracking systems. We intend to derive lessons, identify trends, and analyze the achievements and obstacles experienced by prior projects in similar fields by researching the literature. Key themes will be investigated, including system usability, data security, interoperability, and the influence on healthcare outcomes. We seek to develop a foundation that is both informed and inventive by rooting our work in the current body of knowledge, ensuring that our web application aligns with industry best practices.

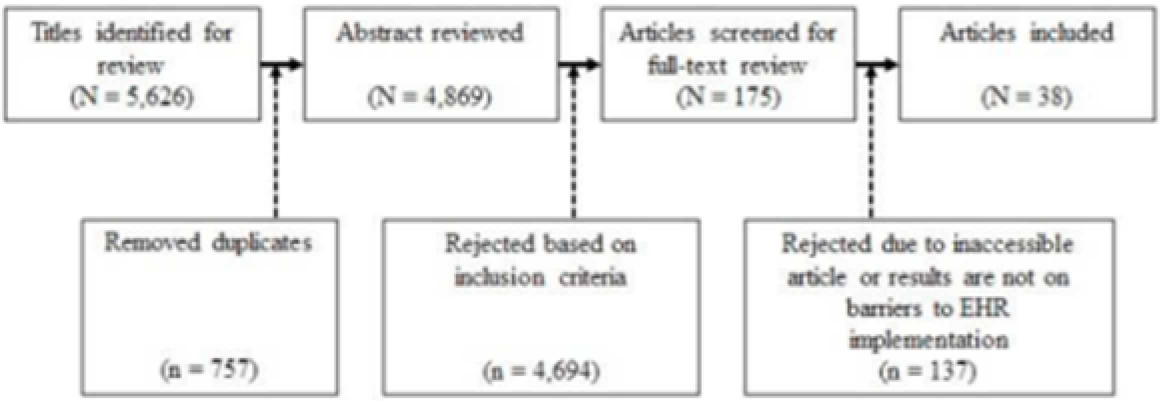

Titles identified for review (N = 5,626)
Abstract reviewed (N = 4,869)
Articles screened for full-text review (N = 175)
Articles included (N = 38)
Removed duplicates (n = 757)
Rejected based on inclusion criteria (n = 4,694)
Rejected due to inaccessible article or results are not on barriers to EHR implementation (n = 137)

Figure 1: Flowchart of inclusion criteria

2.1 Paper 1: Barriers to Electronic Health Record System Implementation and Information Systems Resources: A Structured Review

Journal/Conference Rank: Q1
Publication Year: 2017
Reference: [?]

2.1.1 Summary

The paper discusses a structured literature review that aimed to identify barriers related to Electronic Health Record (EHR) readiness. The review was conducted following the Preferred Reporting Items for Systematic Reviews and Meta-Analysis (PRISMA) guidelines. The search strategy involved specific keywords related to readiness, electronic health, electronic health records, and data resources. Articles were selected based on publication date, language, availability of full text, and their content, specifically related to barriers to EHR implementation or adoption. The identified barriers were categorized into information systems resources, which include people, hardware, software, data, network, and procedures. The most common barriers included user resistance, lack of education and training, lack of IT facilities, concerns about data security, and issues related to data standards and data privacy.

2.1.2 Method of Identification

The barriers were identified by reviewing a collection of scholarly articles. A total of 5626 articles were selected for initial screening and eventually 38 articles met the inclusion criteria. From these articles the barriers were divided into 6 main categories: people, hardware, software, data, network and procedure.

2.1.3 Data Parameters

parameters include elements such as publication date, language, availability of full text, content related to barriers, and the categories of information systems resources (people, hardware, software, data, network, and procedures) used to classify the identified barriers. These parameters were applied to filter and categorize the information within the dataset.

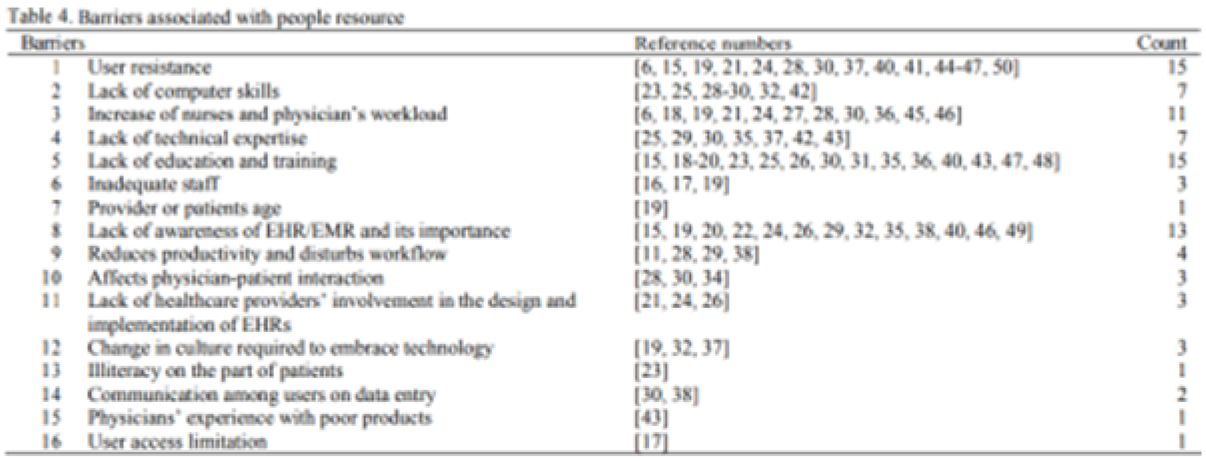


Table 4. Barriers associated with people resource

Barriers		Reference numbers	Count
1	User resistance	[6, 15, 19, 21, 24, 28, 30, 37, 40, 41, 44-47, 50]	15
2	Lack of computer skills	[23, 25, 28-30, 32, 42]	7
3	Increase of nurses and physician's workload	[6, 18, 19, 21, 24, 27, 28, 30, 36, 45, 46]	11
4	Lack of technical expertise	[25, 29, 30, 35, 37, 42, 43]	7
5	Lack of education and training	[15, 18-20, 23, 25, 26, 30, 31, 35, 36, 40, 43, 47, 48]	15
6	Inadequate staff	[16, 17, 19]	3
7	Provider or patients age	[19]	1
8	Lack of awareness of EHR/EMR and its importance	[15, 19, 20, 22, 24, 26, 29, 32, 35, 38, 40, 46, 49]	13
9	Reduces productivity and disturbs workflow	[11, 28, 29, 38]	4
10	Affects physician-patient interaction	[28, 30, 34]	3
11	Lack of healthcare providers' involvement in the design and implementation of EHRs	[21, 24, 26]	3
12	Change in culture required to embrace technology	[19, 32, 37]	3
13	Illiteracy on the part of patients	[23]	1
14	Communication among users on data entry	[30, 38]	2
15	Physicians' experience with poor products	[43]	1
16	User access limitation	[17]	1

Figure 2: Barrier parameters related to people resource

2.1.4 Datasets Used

The dataset in this context refers to the collection of scholarly articles that were reviewed to identify barriers to EHR implementation. This dataset includes 38 articles that met the inclusion criteria for the study.

2.1.5 Paper Link

Access the full paper at https://www.sciencedirect.com/science/article/pii/S1877050917329563.

2.2 Paper 2: A Framework for User-Focused Electronic Health Record System Leveraging Hyperledger Fabric

Journal/Conference Rank: A*
Publication Year: 2023
Reference: [?]

2.2.1 Summary

The paper explores the possibility of integrating health data management system into a private blockchain system, the Hyperledger Fabric (HLF). The solution is based on HLF to solve issues in the current healthcare data management systems. It highlights that the architecture's components are flexible and can be modified to simulate real-world application scenarios, providing insights into managing patient health records' complexity, ease of use, security, scalability, and maintainability. The text also considers theoretical ideas to enhance the system's capabilities through a decentralized blockchain approach.

The study finds that using HLF for managing medical data electronically is a practical approach. HLF ensures confidentiality, sensitivity, and immutability of electronic health records (EHRs). Patient data access is maintained securely on the blockchain, improving control for data creators and reducing single points of failure in databases. Blockchain can address healthcare challenges like interoperability when fully implemented at a state or national level. However, the text notes certain unresolved issues, including HLF's limitations related to transaction history for private data and performance concerns, such as transaction processing speed. the paper suggests that a personalized medical system based on Hyperledger Fabric is practical but has room for improvement. It implies that additional components or technologies beyond HLF may be needed to develop a finished product successfully.

2.2.2 Software Architecture

The system is designed to connect multiple hospitals, with the blockchain operator configuring the network and granting access to users. The certificate authority (CA) is used to validate user credentials, and all network elements are interconnected. The application has a server-side code and a smart contract written in JavaScript and using ExpressJS framework for the RESTfull API. The user interface is created using the Angular 11 framework for a good UI experience.. REST API calls facilitate communication between the frontend and backend, with JSON web tokens for authentication. Developers can choose from programming languages like Java, Golang, JavaScript, and Typescript for chaincode and backend development.

In terms of system design, 2 medical institutions are connected through a single channel, the "hospital channel," and more hospitals can be added when connected to the same channel. The data modeling preferences incorporate LevelDB and CouchDB, with CouchDB being chosen for its flexibility. While blockchain can store information, it is often slow and unreliable. CouchDB is used for maintaining the world state information. The ledger serves as the foundation for transaction logs and world state, with all transactions recorded in the transaction log from the genesis block. Additionally, Redis is employed as a key-value database for storing doctor's credentials.

Figure 4: Software architecture diagram for Paper 1.

2.2.3 Paper Link

Access the full paper at https://www.mdpi.com/2078-2489/14/1/51.

2.2.4 Discussion and Future Planning

The recommended areas for future work in the system are as follows:

Data Management: As the system currently stores actual patient data on the blockchain, there is a need to consider more efficient approaches, especially if the system scales to a national or global level. One option is to keep only metadata and access information on the blockchain, with real data stored in a regular DB. This hybrid method could combine the benefits of both approaches, and further development is required in this area.

Scalability: Anticipating the participation of many medical institution joining the network. As organizations may have multiple peers, sub-channels, and various endorsement policies, further research is needed to address the scalability of the system effectively.

Security of EHRs: To ensure the future security of Electronic Health Records (EHRs). The ability to retrieve tx data from private data should be explored as solution to acquire tx data.

Orderer Systems: The system currently many orderer systems in Hyperledger Fabric v2.x to create a fault-tolerant and crash-resistant infrastructure. More work is required

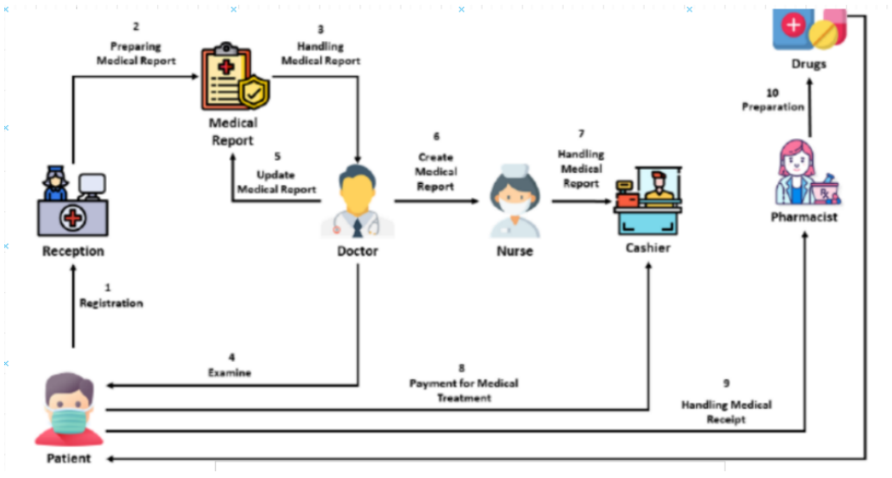


Figure 5: Rich Picture AS-IS

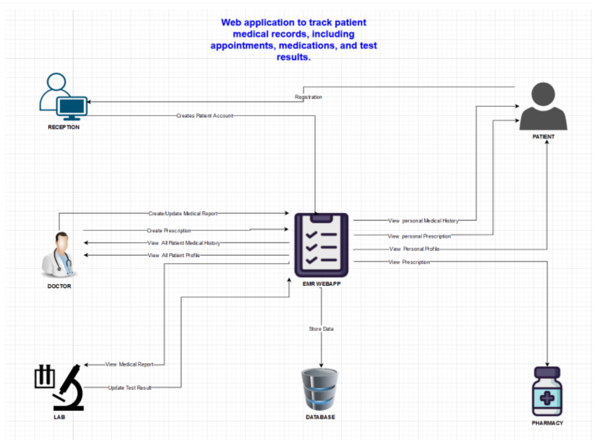

Web application to track patient medical records, including appointments, medications, and test results.
RECEPTION
Registration
Creates Patient Account
PATIENT
Create/Update Medical Report
Create Prescription
View All Patient Medical History
View All Patient Profile
DOCTOR
EMR WEBAPP
View personal Medical History
View personal Prescription
View Personal Profile
View Prescription
Store Data
View Medical Report
Update Test Result
LAB
DATABASE
PHARMACY

Figure 6: Rich Picture To-Be

to improve this functionality, with the goal of improving the reliability and robustness of the system.

Consultation Feature: Since healthcare processes typically involve consultations, the system should consider adding a consultation feature to enhance the user experience for both doctors and patients.

3 System Design

This section of the report navigates into the database management system design, illustrating the As-is Rich Picture of the existing system, To-be Rich Picture of the proposed system, Entity - Relationship Diagram, Schema, Normalized Schema and Data Dictionary that collectively contribute to the application's effectiveness.

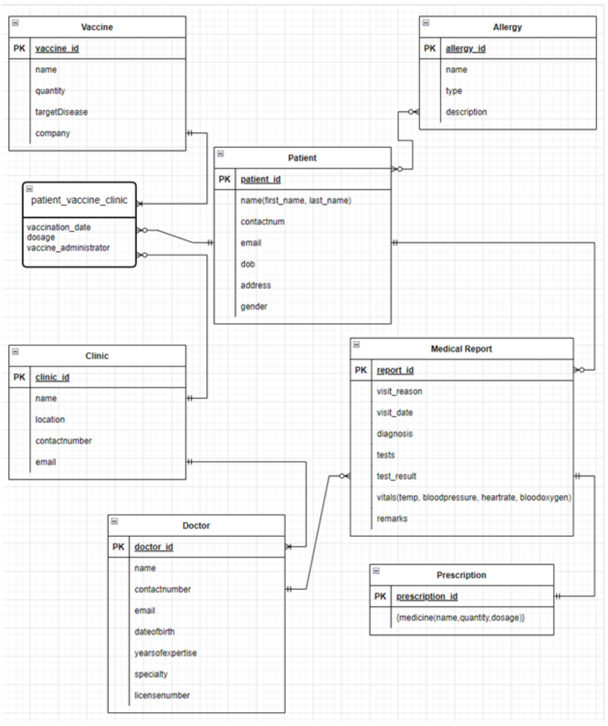


Figure 7: Entity Relationship Diagram

3.1 Rich Picture AS IS

3.2 Rich Picture TO BE

3.3 ERD

Add Image and describe it thoroughly.

3.4 Relation Schema

Add Image and describe it thoroughly.

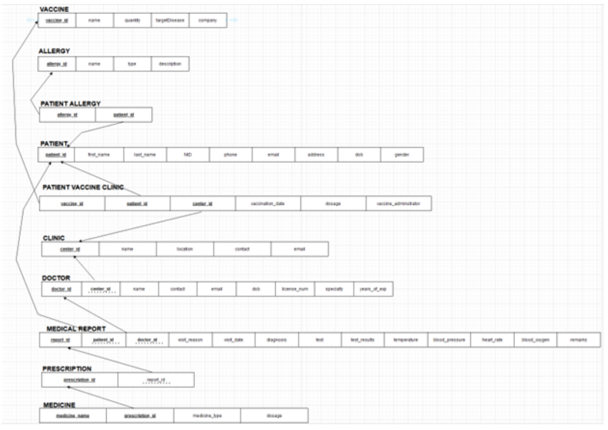
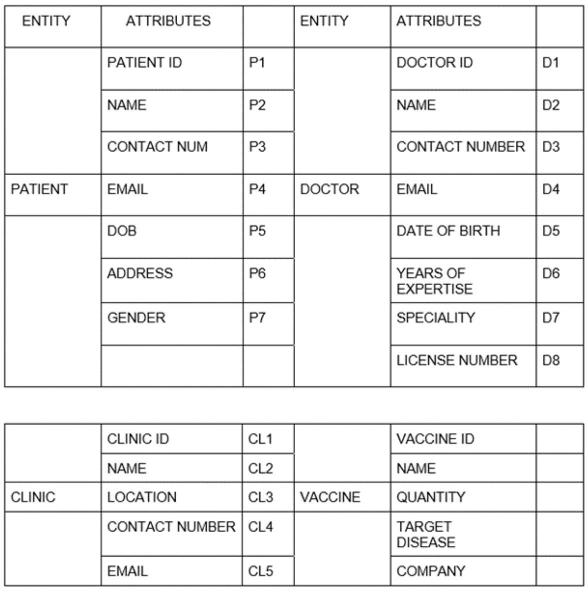


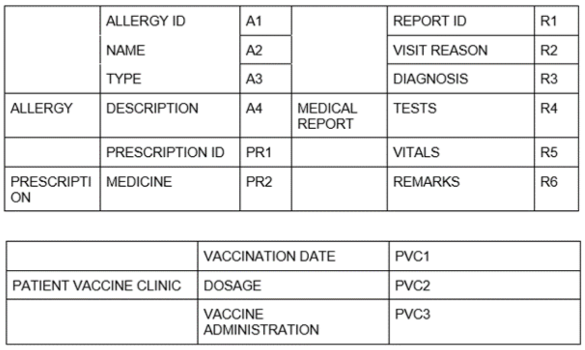
Figure 8: Relational Schema



ENTITY	ATTRIBUTES		ENTITY	ATTRIBUTES	
	PATIENT ID	P1		DOCTOR ID	D1
	NAME	P2		NAME	D2
	CONTACT NUM	P3		CONTACT NUMBER	D3
PATIENT	EMAIL	P4	DOCTOR	EMAIL	D4
	DOB	P5		DATE OF BIRTH	D5
	ADDRESS	P6		YEARS OF EXPERTISE	D6
	GENDER	P7		SPECIALITY	D7
				LICENSE NUMBER	D8

	CLINIC ID	CL1		VACCINE ID	
	NAME	CL2		NAME	
CLINIC	LOCATION	CL3	VACCINE	QUANTITY	
	CONTACT NUMBER	CL4		TARGET DISEASE	
	EMAIL	CL5		COMPANY	

Figure 9: Normalised Schema



	ALLERGY ID	A1		REPORT ID	R1
	NAME	A2		VISIT REASON	R2
	TYPE	A3		DIAGNOSIS	R3
ALLERGY	DESCRIPTION	A4	MEDICAL REPORT	TESTS	R4
	PRESCRIPTION ID	PR1		VITALS	R5
PRESCRIPTI ON	MEDICINE	PR2		REMARKS	R6

	VACCINATION DATE	PVC1
PATIENT VACCINE CLINIC	DOSAGE	PVC2
	VACCINE ADMINISTRATION	PVC3

Figure 10: Normalised Schema cont.

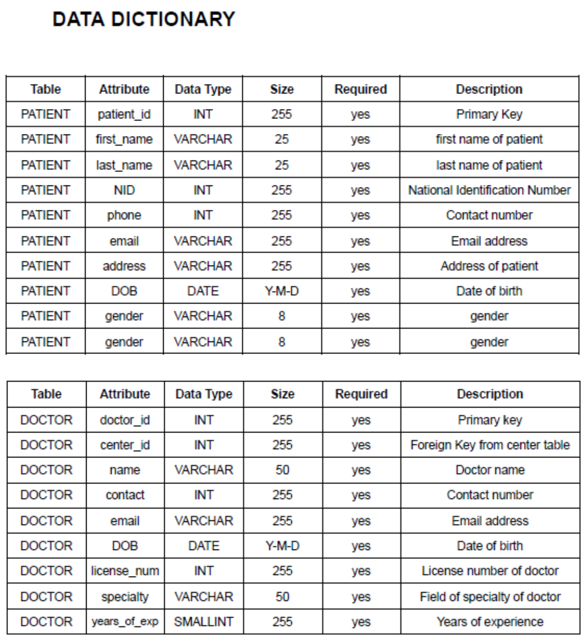
3.5 Normalized Schema / Normalization

P1 - P2,P3,P4,P5,P6,P7 D1 - D2,D3,D4,D5,D6,D7,D8 CL1- CL2,CL3,CL4,CL5 V1 - V2.V3,V4,V5 A1 - A2,A3,A4 PR1- PR2 R1 - R2,R3,R4,R5,R6,R7 PVC1- PVC2,PVC3

Normalization is a process in database design that aims to eliminate redundancy and improve data integrity by organizing data efficiently. It involves breaking down tables into smaller, related tables and establishing relationships between them. As we have properly designed our ERD, it means we have already identified and represented the relationships between entities. In an ideal scenario, a well-designed ERD should reflect a normalized structure to some extent. The normalization process may not be necessary as ERD already exhibits characteristics of normalization, such as minimal redundancy and well-defined relationships.

3.6 Data Dictionary

Add Image and describe it thoroughly.



DATA DICTIONARY

Table	Attribute	Data Type	Size	Required	Description
PATIENT	patient_id	INT	255	yes	Primary Key
PATIENT	first_name	VARCHAR	25	yes	first name of patient
PATIENT	last_name	VARCHAR	25	yes	last name of patient
PATIENT	NID	INT	255	yes	National Identification Number
PATIENT	phone	INT	255	yes	Contact number
PATIENT	email	VARCHAR	255	yes	Email address
PATIENT	address	VARCHAR	255	yes	Address of patient
PATIENT	DOB	DATE	Y-M-D	yes	Date of birth
PATIENT	gender	VARCHAR	8	yes	gender
PATIENT	gender	VARCHAR	8	yes	gender

Table	Attribute	Data Type	Size	Required	Description
DOCTOR	doctor_id	INT	255	yes	Primary key
DOCTOR	center_id	INT	255	yes	Foreign Key from center table
DOCTOR	name	VARCHAR	50	yes	Doctor name
DOCTOR	contact	INT	255	yes	Contact number
DOCTOR	email	VARCHAR	255	yes	Email address
DOCTOR	DOB	DATE	Y-M-D	yes	Date of birth
DOCTOR	license_num	INT	255	yes	License number of doctor
DOCTOR	specialty	VARCHAR	50	yes	Field of specialty of doctor
DOCTOR	years_of_exp	SMALLINT	255	yes	Years of experience

Figure 11: Data Dictionary

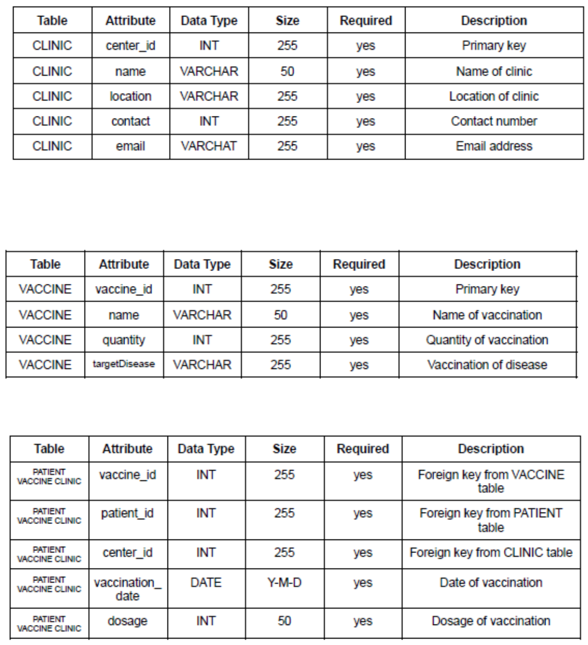


Table	Attribute	Data Type	Size	Required	Description
CLINIC	center_id	INT	255	yes	Primary key
CLINIC	name	VARCHAR	50	yes	Name of clinic
CLINIC	location	VARCHAR	255	yes	Location of clinic
CLINIC	contact	INT	255	yes	Contact number
CLINIC	email	VARCHAT	255	yes	Email address

Table	Attribute	Data Type	Size	Required	Description
VACCINE	vaccine_id	INT	255	yes	Primary key
VACCINE	name	VARCHAR	50	yes	Name of vaccination
VACCINE	quantity	INT	255	yes	Quantity of vaccination
VACCINE	targetDisease	VARCHAR	255	yes	Vaccination of disease

Table	Attribute	Data Type	Size	Required	Description
PATIENT VACCINE CLINIC	vaccine_id	INT	255	yes	Foreign key from VACCINE table
PATIENT VACCINE CLINIC	patient_id	INT	255	yes	Foreign key from PATIENT table
PATIENT VACCINE CLINIC	center_id	INT	255	yes	Foreign key from CLINIC table
PATIENT VACCINE CLINIC	vaccination_ date	DATE	Y-M-D	yes	Date of vaccination
PATIENT VACCINE CLINIC	dosage	INT	50	yes	Dosage of vaccination

Figure 14: Data Dictionary cont.

4 Methodology and Implementation

4.1 Project:1(React + Nodejs)

4.1.1 FRONTEND Methodology

ReactJS: Utilizing ReactJS for the front-end ensures a modular and efficient user interface. React's component-based architecture allows for the creation of reusable UI elements, promoting a maintainable codebase.

Tailwind CSS: Tailwind CSS is a utility-first CSS framework that facilitates rapid UI development. Its utility classes provide a pragmatic approach to styling, enabling a responsive and aesthetically pleasing design.

React Charts: Leveraging React Charts is essential for visualizing health data. The library helps in creating interactive and customizable charts, enabling users to comprehend complex medical information easily.

4.1.2 BACK END Methodology

Node.js: Node.js is chosen for the back-end due to its event-driven, non-blocking I/O model. This is particularly advantageous in handling concurrent requests in a medical application where responsiveness is critical.

Express.js: Express.js is a minimal and flexible Node.js web application framework, providing a robust set of features for web and mobile applications. It simplifies the development of RESTful APIs and integrates seamlessly with Node.js.

MySQL Database: MySQL is employed as the relational database management system (RDBMS) for storing and managing health records. Its reliability, scalability, and support for complex queries make it suitable for an application dealing with sensitive medical data.

4.2 Project:2 (Laravel)

4.2.1 FRONTEND Methodology

HTML: HTML is a fundamental technology for creating and structuring content on the World Wide Web. HTML provides a standardized way to structure content on the web. This standardization ensures consistency and compatibility across different web browsers and platforms. HTML is supported by all major web browsers, making it a universal language for creating content that can be accessed by a broad audience.

Bootstrap: Bootstrap is an open-source front-end framework that facilitates the development of responsive and mobile-friendly web pages. It includes pre-designed components, such as navigation bars, buttons, and forms, as well as a responsive grid system, simplifying the process of building visually appealing and consistent websites.

Charts: Chart is implemented using js charting libraries in conjunction with Bootstrap. Popular charting libraries include Chart.js, D3.js, and others. These libraries allow developers to create various types of charts, such as bar charts, line charts, pie charts, etc., and they can be integrated into Bootstrap-based web projects to visualize data in a more graphical and interactive way.

4.2.2 BACK END Methodology

MVC: MVC, or Model-View-Controller, is a software architectural pattern used in designing and organizing code in applications. It divides the application into three interconnected components: Model, View, Controller.

MySQL Database: MySQL is employed as the relational database management system (RDBMS) for storing and managing health records. Its reliability, scalability, and support for complex queries make it suitable for an application dealing with sensitive medical data.

4.3 Implementation

Add image of the interfaces and describe them thoroughly with it's purpose and functionality.

4.4 Project:1(React + Nodejs)

4.4.1 FRONTEND Implementation

Component Structure: Design a modular component structure in React to ensure a clear separation of concerns. Components should represent different sections of the EHR, such as patient information, medical history, and diagnostic reports.

User Authentication: Implement user authentication using JWT (JSON Web Tokens) to secure access to patient records and comply with privacy regulations.

Responsive Design: Utilize Tailwind CSS to implement a responsive design, ensuring the application is accessible across various devices.

Chart Integration: Integrate React Charts to dynamically generate charts based on patient health data. Use real-time updates for live monitoring of vital signs.

4.4.2 BACKEND Implementation

RESTful API: Design a RESTful API using Express.js to handle requests from the front-end. Implement endpoints for CRUD operations on patient records, appointments, and other relevant data.

1. **Middleware:** Implement middleware for request validation, authentication, and authorization to ensure data security and compliance with privacy standards.
2. **Database Schema:** Create an optimized MySQL database schema to store patient records securely. Implement relationships between tables to represent associations between different entities (e.g., patients, doctors, medical reports).

Data Encryption: Implement encryption mechanisms to safeguard sensitive health data stored in the database.

Error Handling: Implement robust error handling to provide meaningful error messages to users and log errors for debugging purposes. By following these methodologies and implementation details, you can ensure the development of a secure, scalable, and user-friendly Electronic Health Record web application. Regular testing, code reviews, and adherence to coding standards are crucial throughout the development process to maintain the application's integrity and security.

4.5 Project:2 (Laravel)

4.5.1 FRONTEND Implementation

Blade View: Laravel Blade is a templating engine used in the Laravel PHP framework. It provides a concise and expressive syntax for defining views in Laravel applications. Some key features of Laravel Blade include: Inline PHP Code, Template Inheritance, Control Structures, Extending and Including Views, Blade Directives, Escaping Content.

User Authentication: Laravel's built-in authentication system primarily uses sessions to manage user authentication. When a user logs in, Laravel creates a session to keep track of their authentication status. This session persists across subsequent requests, allowing the application to recognize and authenticate the user.

Responsive Design: Utilize Bootstrap to implement a responsive design, ensuring the application is accessible across various devices.

Chart Integration: Integrate Charts using JS scripts to dynamically generate charts based on patient health data. Use real-time updates for live monitoring of vital signs.

4.5.2 BACKEND Implementation

Controller: Design and implement a MVC controller method to handle requests from the front-end. Implement endpoints for CRUD operations on patient records, appointments, and other relevant data.

Middleware: Implement middleware for request validation, authentication, and authorization to ensure data security and compliance with privacy standards.

Database Schema: Create an optimized MySQL database schema to store patient records securely. Implement relationships between tables to represent associations between different entities (e.g., patients, doctors, medical reports).

Data Encryption: Implement encryption mechanisms to safeguard sensitive health data stored in the database.

Error Handling: Implement robust error handling to provide meaningful error messages to users and log errors for debugging purposes.

4.6 ARCHITECTURE

Software Architecture: Overview of 3-Tier Architecture

In the realm of software development, the choice of architecture plays a pivotal role in shaping the efficiency, scalability, and maintainability of a system. Our project adopts a 3-tier architecture, a well-established model that separates the concerns of user interaction, business logic, and data storage into distinct layers. This design ensures a modular and scalable approach to building robust applications.

4.6.1 Tier 1: Presentation Layer (Client App)

The first tier, also known as the presentation layer, is where users interact with the system. In our architecture, this interaction takes place through the Client App, a user-friendly interface designed for seamless communication and intuitive user experience. The Client App is responsible for capturing user input, rendering data, and presenting information in a comprehensible manner. We're using a React+Tailwind app for our client side.

And in Laravel, Blade templates is the user interface which is coded with HTML-CSS which is responsible for capturing user input, rendering data, and presenting information in a comprehensible manner.

4.6.2 Tier 2: Application Layer (Backend Server)

Sitting between the Presentation Layer and the Data Layer, the Backend Server is the heart of our architecture. This layer houses the business logic, ensuring that the application's functionality is executed efficiently. When the Client App requests data or submits user input, the Backend Server processes these requests, interacts with the database, and orchestrates the flow of information. It acts as a bridge, facilitating communication between the user interface and the data storage layer. We're using a node+express framework for our back-end server.

And in Laravel, the Controller requests data or submits user input, the Backend Server processes these requests, interacts with the database, and orchestrates the flow of information. It acts as a bridge, facilitating communication between the user interface and the data storage layer. Here we are using the Controller of MVC architecture.

4.6.3 Tier 3: Data Layer (Database)

The final tier, the Data Layer, involves the storage and retrieval of data. Our chosen database system is queried by the Backend Server to fetch and store information. This separation of concerns ensures that data management is abstracted from the business logic, providing flexibility and scalability. We have implemented secure and efficient database interactions to guarantee the integrity and availability of the stored data. We're using mySQL for our database needs.

4.6.4 Interaction Flow

1. **User Interaction:** Users interact with the Client App, providing input and receiving real-time feedback.

2. **Client-Server Communication:** The Client App sends requests to the Backend Server, seeking data or initiating processes.

3. **Business Logic Execution:** The Backend Server processes requests, executes business logic, and interacts with the database as needed.

4. **Data Retrieval/Storage:** Database queries are performed to retrieve or store data, based on the requirements of the application.

5. **Data Transmission:** The retrieved data is sent back to the Client App, where it is presented to the user while in Laravel data is sent to the Controller for interaction.

This 3-tier architecture ensures a modular and scalable design, allowing for easy maintenance, updates, and future expansions. The clear separation of concerns enhances system flexibility, making it adept at handling evolving requirements while maintaining a robust foundation for optimal performance.

Fig: 3 Tier Architecture

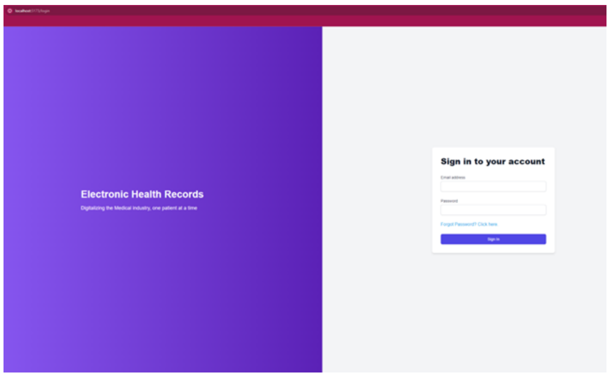

localhost:5173/login
Electronic Health Records
Digitalizing the Medical industry, one patient at a time
Sign in to your account
Email address
Password
Forgot Password? Click here
Sign in

Figure 15: All User Login

5 Interface

Project:1 (React + Nodejs)

5.0.1 ALL User: Login

All User Login features (Functionality, mySQL commands) Checks for user in database, if user doesn't exist, sends "user doesn't exist" error Checks for user in database, if user exists but wrong password says "invalid username or password" Checks for user in database, if user exists and types correct password, successfully logs in and is assigned a JWT token for future requests

SQL Statement:

5.0.2 ALL User: Registration

Features Ability to register user Available roles - admin, patient, doctor During registration, searches for duplicate email, if duplicate found, shows an error 'user already exists'.

SQL Statement:

6 Dashboard

6.1 Project 1

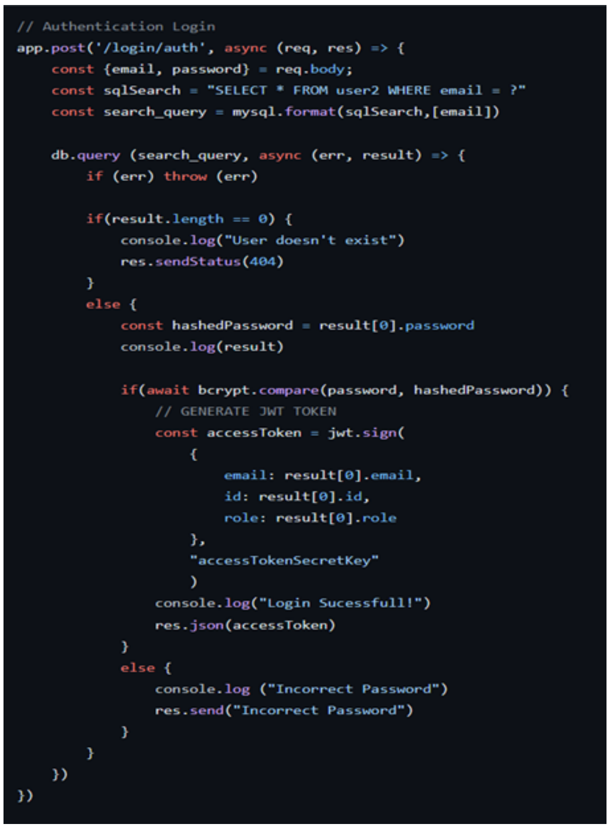
6.1.1 Admin Dashboard: Overview

Overview Features Displays total number of registered doctors Displays total number of registered patients Displays total number of registered hospitals Displays total number of registered medical reports Dynamic Bar chart for total number of male/female patients Dynamic Pie chart showing most common viruses Dynamic Pie chart showing most common diseases Dynamic Pie chart showing most common allergies

SQL Statements for PIECHART:

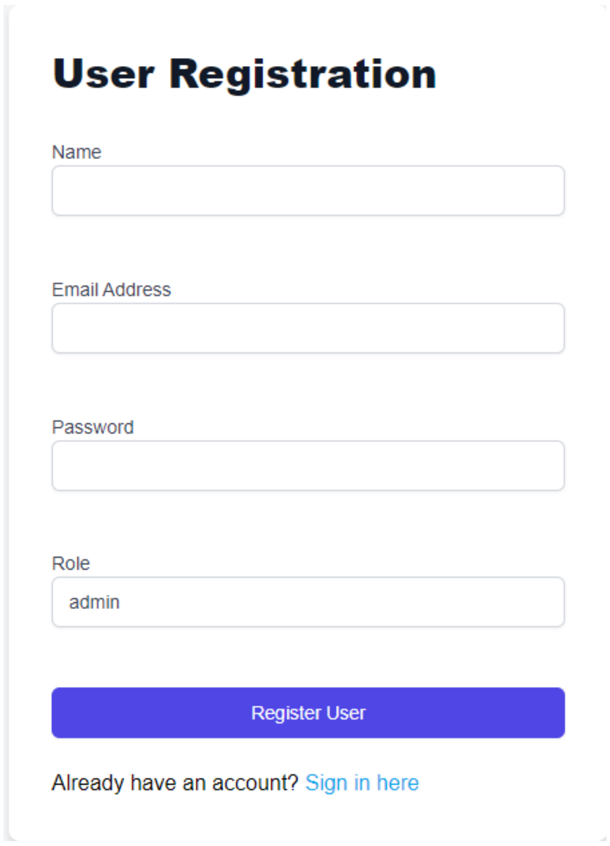
6.1.2 Admin Dashboard: Patient Panel

Patient Panel Features Display a minimalist table which shows all patient data. Data include, patient name, gender, date of birth, address, blood group and phone number.



```
// Authentication Login
app.post('/login/auth', async (req, res) => {
    const {email, password} = req.body;
    const sqlSearch = "SELECT * FROM user2 WHERE email = ?"
    const search_query = mysql.format(sqlSearch,[email])

    db.query (search_query, async (err, result) => {
        if (err) throw (err)

        if(result.length == 0) {
            console.log("User doesn't exist")
            res.sendStatus(404)
        }
        else {
            const hashedPassword = result[0].password
            console.log(result)

            if(await bcrypt.compare(password, hashedPassword)) {
                // GENERATE JWT TOKEN
                const accessToken = jwt.sign(
                    {
                        email: result[0].email,
                        id: result[0].id,
                        role: result[0].role
                    },
                    "accessTokenSecretKey"
                    )
                console.log("Login Sucessfull!")
                res.json(accessToken)
            }
            else {
                console.log ("Incorrect Password")
                res.send("Incorrect Password")
            }
        }
    })
})
```

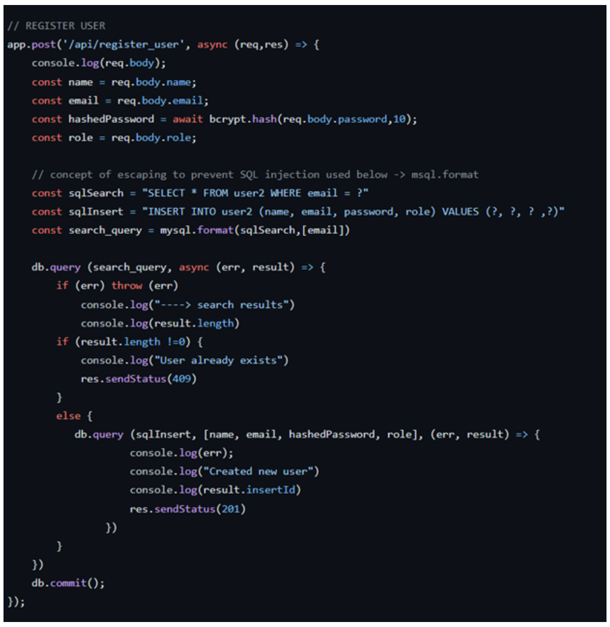
Figure 16: SQL Statement:



User Registration

Name	
Email Address	
Password	
Role	admin

Figure 17: All User Registration



```
// REGISTER USER
app.post('/api/register_user', async (req,res) => {
    console.log(req.body);
    const name = req.body.name;
    const email = req.body.email;
    const hashedPassword = await bcrypt.hash(req.body.password,10);
    const role = req.body.role;

    // concept of escaping to prevent SQL injection used below -> msql.format
    const sqlSearch = "SELECT * FROM user2 WHERE email = ?"
    const sqlInsert = "INSERT INTO user2 (name, email, password, role) VALUES (?, ?, ? ,?)"
    const search_query = mysql.format(sqlSearch,[email])

    db.query (search_query, async (err, result) => {
        if (err) throw (err)
            console.log("----> search results")
            console.log(result.length)
        if (result.length !=0) {
            console.log("User already exists")
            res.sendStatus(409)
        }
        else {
           db.query (sqlInsert, [name, email, hashedPassword, role], (err, result) => {
                console.log(err);
                console.log("Created new user")
                console.log(result.insertId)
                res.sendStatus(201)
            })
        }
    })
    db.commit();
});
```

Figure 18: SQL Statement:

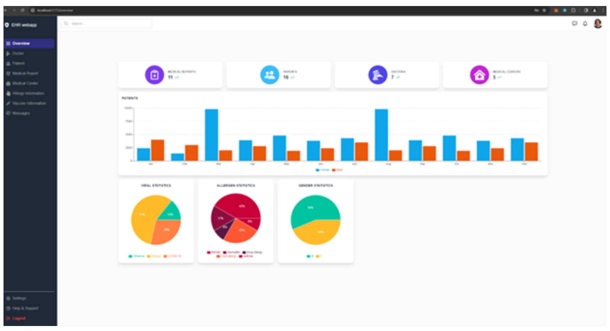
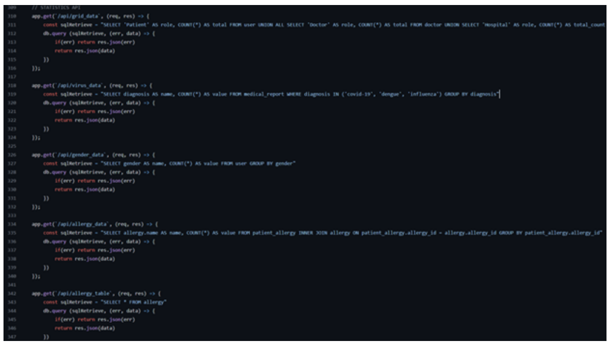
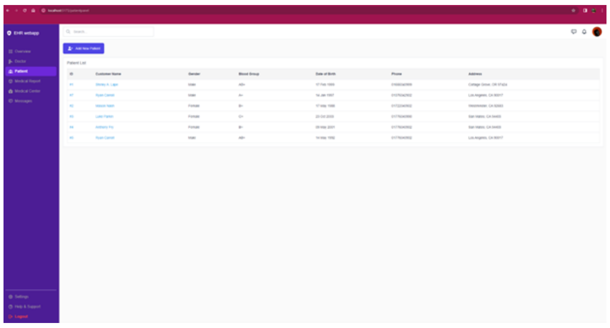


Figure 19: Admin Dashboard Overview



```
// STATISTICS API
app.get(`/api/grid_data`, (req, res) => {
    const sqlRetrieve = "SELECT 'Patient' AS role, COUNT(*) AS total FROM user UNION ALL SELECT 'Doctor' AS role, COUNT(*) AS total FROM doctor UNION SELECT 'Hospital' AS role, COUNT(*) AS total_count
    db.query (sqlRetrieve, (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});

app.get(`/api/virus_data`, (req, res) => {
    const sqlRetrieve = "SELECT diagnosis AS name, COUNT(*) AS value FROM medical_report WHERE diagnosis IN ('covid-19', 'dengue', 'influenza') GROUP BY diagnosis"
    db.query (sqlRetrieve, (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});

app.get(`/api/gender_data`, (req, res) => {
    const sqlRetrieve = "SELECT gender AS name, COUNT(*) AS value FROM user GROUP BY gender"
    db.query (sqlRetrieve, (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});

app.get(`/api/allergy_data`, (req, res) => {
    const sqlRetrieve = "SELECT allergy.name AS name, COUNT(*) AS value FROM patient_allergy INNER JOIN allergy ON patient_allergy.allergy_id = allergy.allergy_id GROUP BY patient_allergy.allergy_id"
    db.query (sqlRetrieve, (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});

app.get(`/api/allergy_table`, (req, res) => {
    const sqlRetrieve = "SELECT * FROM allergy"
    db.query (sqlRetrieve, (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
```

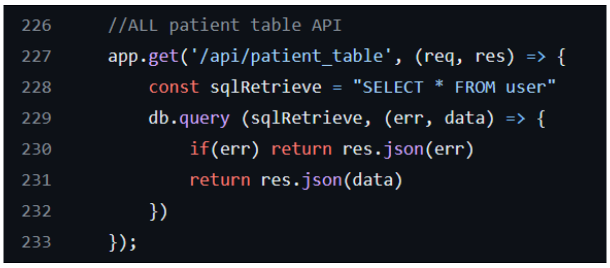
Figure 20: SQL Statement:


Add New Patient

Patient List

ID	Customer Name	Gender	Blood Group	Date of Birth	Phone	Address
#1	Shirley A. Lape	Male	AB+	17 Feb 1999	[illegible]	Cottage Grove, OR 97424
#7	Ryan Carroll	Male	A+	14 Jan 1997	[illegible]	Los Angeles, CA 90017
#2	Mason Nash	Female	B+	17 May 1998	[illegible]	Westminster, CA 92683
#3	Luke Parker	Female	O+	23 Oct 2003	[illegible]	San Mateo, CA 94403
#4	Anthony Fu	Female	B+	09 May 2001	[illegible]	San Mateo, CA 94403
#5	Ryan Carroll	Male	AB+	14 May 1992	[illegible]	Los Angeles, CA 90017

Figure 21: Admin Dashboard: Patient Panel



```
//ALL patient table API
app.get('/api/patient_table', (req, res) => {
    const sqlRetrieve = "SELECT * FROM user"
    db.query (sqlRetrieve, (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});
```

Figure 22: SQL Statement:

Ability to add new patients via the “add new patient” button, which would navigate the user to a form which they have till fill up to add a new patient into the database.

SQL STATEMENT:

6.1.3 Admin Dashboard: Doctor Panel

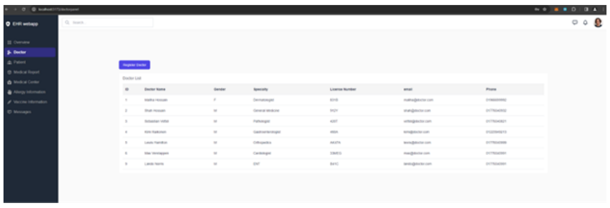
Features Shows all doctors Can register new doctor

SQL Statements:

6.1.4 Admin Dashboard: Medical Report Panel

Patient Panel Features Display a minimalist table which shows all medical report data. Data include, patient name, doctor name, appointment date, appointment reason, diagnosis, lab results, lab test, prescription, remarks. Ability to add a new medical report via the “add new medical report” button, which would navigate the user to a form which they have till fill up to add a new medical report into the database.

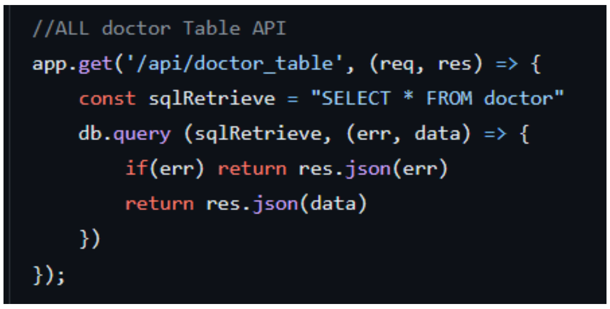
SQL STATEMENT:


Register Doctor

Doctor List

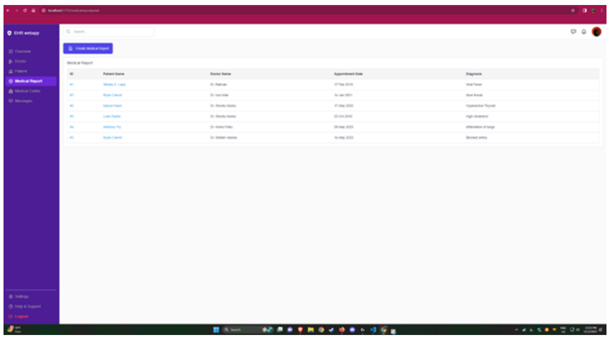
ID	Doctor Name	Gender	Specialty	License Number	email	Phone
1	Maliha Hossain	F	Dermatologist	[illegible]	maliha@doctor.com	[illegible]
2	Shah Hossain	M	General Medicine	[illegible]	shah@doctor.com	[illegible]
3	Sebastian Vettel	M	Pathologist	[illegible]	vettel@doctor.com	[illegible]
4	Kimi Raikkonen	M	Gastroenterologist	[illegible]	kimi@doctor.com	[illegible]
5	Lewis Hamilton	M	Orthopedics	[illegible]	lewis@doctor.com	[illegible]
6	Max Verstappen	M	Cardiologist	[illegible]	max@doctor.com	[illegible]
9	Lando Norris	M	ENT	[illegible]	lando@doctor.com	[illegible]

Figure 23: Admin Dashboard: Doctor Panel



```
//ALL doctor Table API
app.get('/api/doctor_table', (req, res) => {
    const sqlRetrieve = "SELECT * FROM doctor"
    db.query (sqlRetrieve, (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});
```

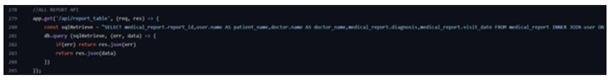
Figure 24: SQL Statement:



Medical Report

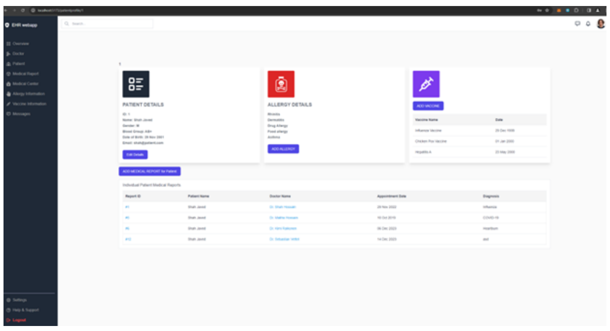
ID	Patient Name	Doctor Name	Appointment Date	Diagnosis
#1	Shirley A. Lape	Dr. Batman	17 Feb 2019	Viral Fever
#7	Ryan Carroll	Dr. Iron Man	14 Jan 2021	Sore throat
#2	Mason Nash	Dr. Shontu Mortu	17 May 2022	Hyperactive Thyroid
#3	Luke Parker	Dr. Shontu Mortu	23 Oct 2010	High cholesterol
#4	Anthony Fry	Dr. Kontu Poltu	09 May 2023	Inflammation of lungs
#5	Ryan Carroll	Dr. Sheikh Hasina	14 May 2023	Blocked artery

Figure 25: Medical Report Panel



```
//ALL REPORT API
app.get('/api/report_table', (req, res) => {
    const sqlRetrieve = "SELECT medical_report.report_id,user.name AS patient_name,doctor.name AS doctor_name,medical_report.diagnosis,medical_report.visit_date FROM medical_report INNER JOIN user ON
    db.query (sqlRetrieve, (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});
```

Figure 26: SQL STATEMENT



PATIENT DETAILS

ID: 1
Name: Shah Javed
Gender: M
Blood Group: AB+
Date of Birth: 29 Nov 2001
Email: shah@patient.com

ALLERGY DETAILS

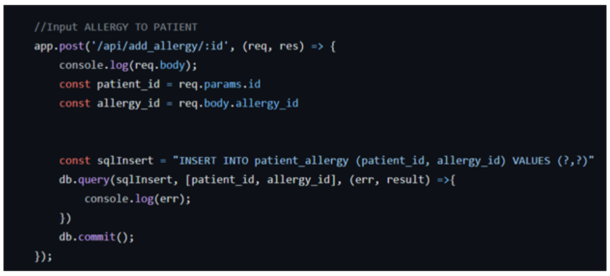
Rhinitis
Dermatitis
Drug Allergy
Food allergy
Asthma

Vaccine Name	Date
Influenza Vaccine	29 Dec 1999
Chicken Pox Vaccine	01 Jan 2000
Hepatitis A	23 May 2000

Individual Patient Medical Reports

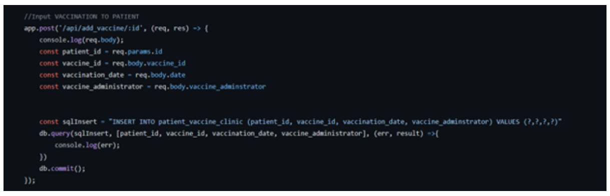
Report ID	Patient Name	Doctor Name	Appointment Date	Diagnosis
#1	Shah Javed	Dr. Shah Hossain	29 Nov 2022	Influenza
#5	Shah Javed	Dr. Maliha Hossain	10 Oct 2019	COVID-19
#6	Shah Javed	Dr. Kimi Raikkonen	06 Dec 2023	Heartburn
#12	Shah Javed	Dr. Sebastian Vettel	14 Dec 2023	asd

Figure 27: User Profile



```
//Input ALLERGY TO PATIENT
app.post('/api/add_allergy/:id', (req, res) => {
    console.log(req.body);
    const patient_id = req.params.id
    const allergy_id = req.body.allergy_id


    const sqlInsert = "INSERT INTO patient_allergy (patient_id, allergy_id) VALUES (?,?)"
    db.query(sqlInsert, [patient_id, allergy_id], (err, result) =>{
        console.log(err);
    })
    db.commit();
});
```

Figure 28: SQL STATEMENTS



```
//Input VACCINATION TO PATIENT
app.post('/api/add_vaccine/:id', (req, res) => {
    console.log(req.body);
    const patient_id = req.params.id
    const vaccine_id = req.body.vaccine_id
    const vaccination_date = req.body.date
    const vaccine_administrator = req.body.vaccine_adminstrator


    const sqlInsert = "INSERT INTO patient_vaccine_clinic (patient_id, vaccine_id, vaccination_date, vaccine_adminstrator) VALUES (?,?,?,?)"
    db.query(sqlInsert, [patient_id, vaccine_id, vaccination_date, vaccine_administrator], (err, result) =>{
        console.log(err);
    })
    db.commit();
});
```

Figure 29: SQL STATEMENTS

6.1.5 Admin Dashboard: User Profile

User Profile Feature: Shows individual patient details Show patient allergy details Shows patient vaccine details Can add allergy to patient Can add vaccine to patient Patient based medical report list Can add patient based medical report
SQL STATEMENTS

6.1.6 Admin Dashboard: In-depth Medical Report

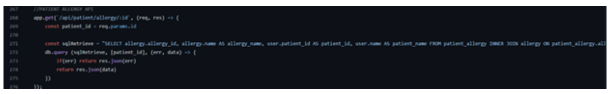
In depth Medical Report Feature More in depth details of the medical report. Shows vitals of patient for given medical report Shows prescription for the given medical report Shows doctor details of the given medical report Shows patient details of the given medical report Ability to add more medicine to prescription
SQL Statement:

6.1.7 Allergy Database Page

Features Shows allergy information from existing database

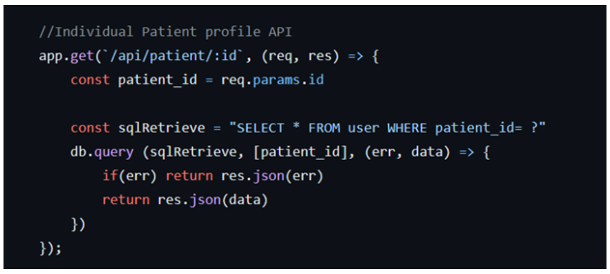
6.1.8 Registration: Patient

Registration Patient Features(Functionality, mySQL commands) Ability to register new patients into the database. SQL STATEMENT



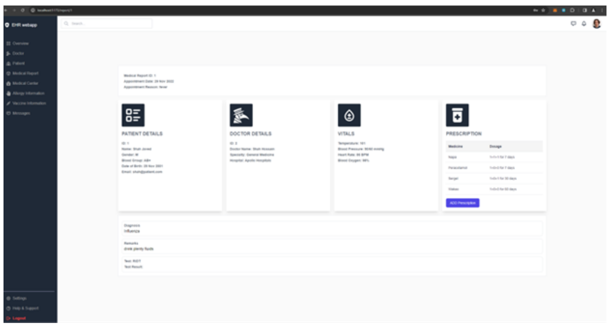
```
//PATIENT ALLERGY API
app.get('/api/patient/allergy/:id', (req, res) => {
    const patient_id = req.params.id

    const sqlRetrieve = "SELECT allergy.allergy_id, allergy.name AS allergy_name, user.patient_id AS patient_id, user.name AS patient_name FROM patient_allergy INNER JOIN allergy ON patient_allergy.all
    db.query (sqlRetrieve, [patient_id], (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});
```

Figure 30: SQL STATEMENTS



```
//Individual Patient profile API
app.get(`/api/patient/:id`, (req, res) => {
    const patient_id = req.params.id

    const sqlRetrieve = "SELECT * FROM user WHERE patient_id= ?"
    db.query (sqlRetrieve, [patient_id], (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});
```

Figure 31: SQL STATEMENTS



Medical Report ID: 1
Appointment Date: 29 Nov 2022
Appointment Reason: fever

PATIENT DETAILS
ID: 1
Name: Shah Javed
Gender: M
Blood Group: AB+
Date of Birth: 29 Nov 2001
Email: shah@patient.com

DOCTOR DETAILS
ID: 2
Doctor Name: Shah Hossain
Specialty: General Medicine
Hospital: Apollo Hospitals

VITALS
Temperature: 101
Blood Pressure: 90/60 mmHg
Heart Rate: 88 BPM
Blood Oxygen: 98%

PRESCRIPTION

Medicine	Dosage
Napa	1+0+1 for 7 days
Paracetamol	1+0+0 for 7 days
Sergel	1+0+1 for 30 days
Viatose	1+0+0 for 60 days

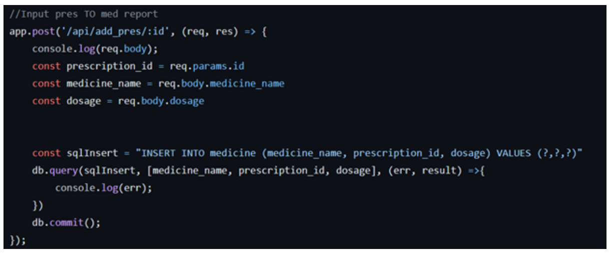
ADD Prescription

Diagnosis
Influenza

Remarks
drink plenty fluids

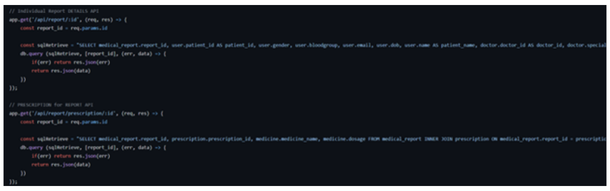
Test: RDT
Test Result:

Figure 32: SQL STATEMENTS



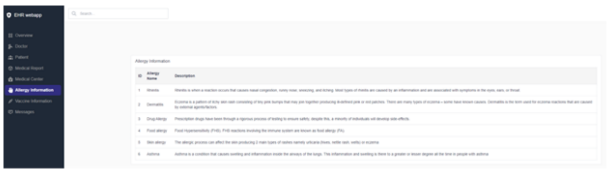
```
//Input pres TO med report
app.post('/api/add_pres/:id', (req, res) => {
    console.log(req.body);
    const prescription_id = req.params.id
    const medicine_name = req.body.medicine_name
    const dosage = req.body.dosage


    const sqlInsert = "INSERT INTO medicine (medicine_name, prescription_id, dosage) VALUES (?,?,?)"
    db.query(sqlInsert, [medicine_name, prescription_id, dosage], (err, result) =>{
        console.log(err);
    })
    db.commit();
});
```

Figure 33: SQL STATEMENTS



```
// Individual Report DETAILS API
app.get('/api/report/:id', (req, res) => {
    const report_id = req.params.id

    const sqlRetrieve = "SELECT medical_report.report_id, user.patient_id AS patient_id, user.gender, user.bloodgroup, user.email, user.dob, user.name AS patient_name, doctor.doctor_id AS doctor_id, doctor.speciali
    db.query (sqlRetrieve, [report_id], (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});

// PRESCRIPTION for REPORT API
app.get('/api/report/prescription/:id', (req, res) => {
    const report_id = req.params.id

    const sqlRetrieve = "SELECT medical_report.report_id, prescription.prescription_id, medicine.medicine_name, medicine.dosage FROM medical_report INNER JOIN prescription ON medical_report.report_id = prescriptio
    db.query (sqlRetrieve, [report_id], (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});
```

Figure 34: SQL STATEMENTS



Allergy Information

ID	Allergy Name	Description
1	Rhinitis	Rhinitis is when a reaction occurs that causes nasal congestion, runny nose, sneezing, and itching. Most types of rhinitis are caused by an inflammation and are associated with symptoms in the eyes, ears, or throat.
2	Dermatitis	Eczema is a pattern of itchy skin rash consisting of tiny pink bumps that may join together producing ill-defined pink or red patches. There are many types of eczema – some have known causes. Dermatitis is the term used for eczema reactions that are caused by external agents/factors.
3	Drug Allergy	Prescription drugs have been through a rigorous process of testing to ensure safety, despite this, a minority of individuals will develop side effects.
4	Food allergy	Food Hypersensitivity (FHS). FHS reactions involving the immune system are known as food allergy (FA).
5	Skin allergy	The allergic process can affect the skin producing 2 main types of rashes namely urticaria (hives, nettle rash, welts) or eczema
6	Asthma	Asthma is a condition that causes swelling and inflammation inside the airways of the lungs. This inflammation and swelling is there to a greater or lesser degree all the time in people with asthma

Figure 35: Allergy database

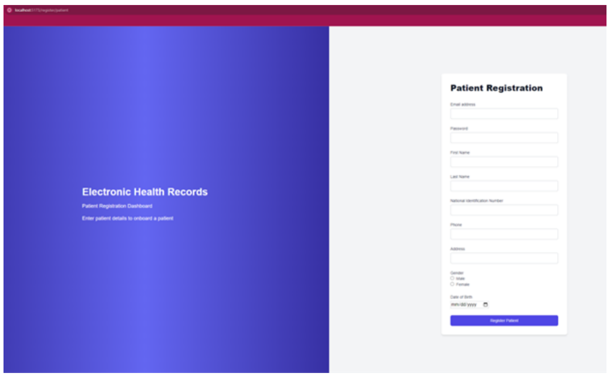
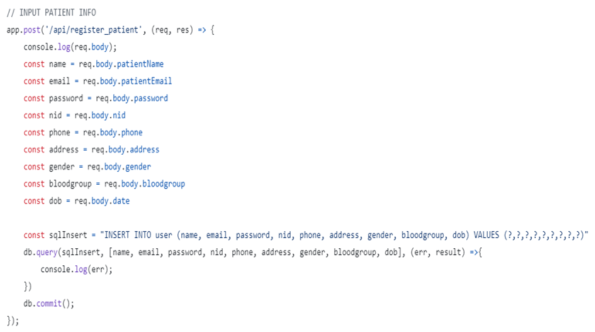

localhost:5173/register/patient
Electronic Health Records
Patient Registration Dashboard
Enter patient details to onboard a patient
Patient Registration
Email address
Password
First Name
Last Name
National Identification Number
Phone
Address
Gender
Male
Female
Date of Birth
mm/dd/yyyy
Register Patient

Figure 36: Patient registration



```
// INPUT PATIENT INFO
app.post('/api/register_patient', (req, res) => {
    console.log(req.body);
    const name = req.body.patientName
    const email = req.body.patientEmail
    const password = req.body.password
    const nid = req.body.nid
    const phone = req.body.phone
    const address = req.body.address
    const gender = req.body.gender
    const bloodgroup = req.body.bloodgroup
    const dob = req.body.date

    const sqlInsert = "INSERT INTO user (name, email, password, nid, phone, address, gender, bloodgroup, dob) VALUES (?,?,?,?,?,?,?,?,?)"
    db.query(sqlInsert, [name, email, password, nid, phone, address, gender, bloodgroup, dob], (err, result) =>{
        console.log(err);
    })
    db.commit();
});
```

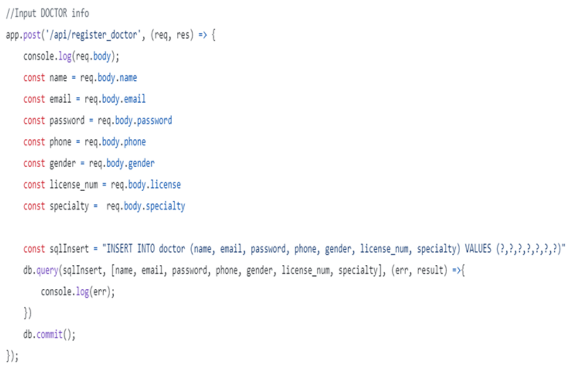
Figure 37: SQL STATEMENT

6.1.9 Registration: Doctor

Registration Doctor Features (Functionality, mySQL commands) Input form to add new doctor into the database.

6.1.10 Create Medical Report

Medical report creation Details (Functionality, mySQL commands). Input form to make new Medical Report SQL Statement:



```
//Input DOCTOR info
app.post('/api/register_doctor', (req, res) => {
    console.log(req.body);
    const name = req.body.name
    const email = req.body.email
    const password = req.body.password
    const phone = req.body.phone
    const gender = req.body.gender
    const license_num = req.body.license
    const specialty =  req.body.specialty

    const sqlInsert = "INSERT INTO doctor (name, email, password, phone, gender, license_num, specialty) VALUES (?,?,?,?,?,?,?)"
    db.query(sqlInsert, [name, email, password, phone, gender, license_num, specialty], (err, result) =>{
        console.log(err);
    })
    db.commit();
});
```

Figure 38: SQL STATEMENT

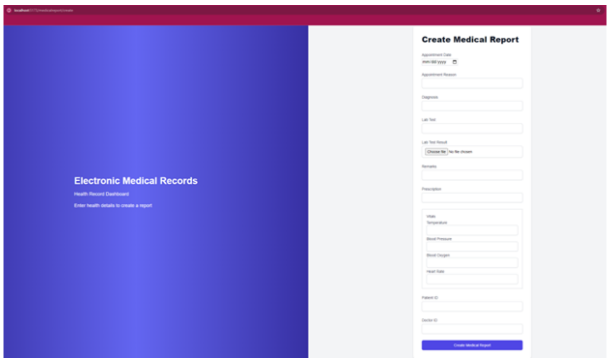
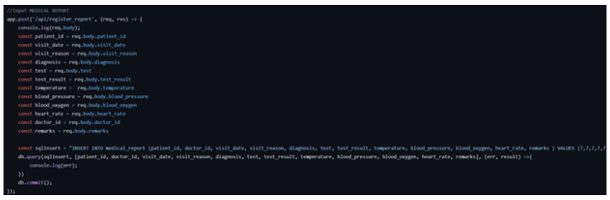


Figure 39: Create Medical Report



```
//Input MEDICAL REPORT
app.post('/api/register_report', (req, res) => {
    console.log(req.body);
    const patient_id = req.body.patient_id
    const visit_date = req.body.visit_date
    const visit_reason = req.body.visit_reason
    const diagnosis = req.body.diagnosis
    const test = req.body.test
    const test_result = req.body.test_result
    const temperature =  req.body.temperature
    const blood_pressure = req.body.blood_pressure
    const blood_oxygen = req.body.blood_oxygen
    const heart_rate = req.body.heart_rate
    const doctor_id = req.body.doctor_id
    const remarks = req.body.remarks

    const sqlInsert = "INSERT INTO medical_report (patient_id, doctor_id, visit_date, visit_reason, diagnosis, test, test_result, temperature, blood_pressure, blood_oxygen, heart_rate, remarks ) VALUES (?,?,?,?,?
    db.query(sqlInsert, [patient_id, doctor_id, visit_date, visit_reason, diagnosis, test, test_result, temperature, blood_pressure, blood_oxygen, heart_rate, remarks], (err, result) =>{
        console.log(err);
    })
    db.commit();
});
```

Figure 40: SQL STATEMENT

7 Result Analysis

During our research for EHR webapps, we learned a lot about how the electronic health record systems work. We learned how patient-doctor interaction works. When we started to design the web application, we got to learn more in depth, the database concepts like ERD, normalization, schema and data dictionary. When we got to the programming, we learned authentication and CRUD operations and SQL queries. We had many challenges on the way. Overall it was a very good learning experience.

8 Conclusion and Future Work

- Integration of the EHR webapp directly with hospitals and clinics where these bodies can directly pull data from the server. We can use the NID of the patients to uniquely identify users across medical institutes.
- Integration with laboratories and pharmacies to check for patient prescription and tests and update them from individual platforms
- Enhanced UX/UI experience for all stakeholders
- Integration of Advanced Analytics, real time analytics to show data on epidemic/pandemic.
- Development of a mobile application
- Enhance security

References

[1] Jaillah Mae Gesulga, Almarie Berjame, Kristelle Sheen Moquiala, Adrian Galido, Barriers to Electronic Health Record System Implementation and Information Systems Resources: A Structured Review, Procedia Computer Science, Volume 124, 2017, Pages 544-551, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2017.12.188. (https://www.sciencedirect.com/science/article/pii/S187705091732956)

[2] Campion, Francis X, MD; Richter, James M, MD. The Journal of Medical Practice Management : MPM; Tampa Vol. 27, Iss. 1, (Jul/Aug 2011): 50-6, (https://www.proquest.com/scholarly-journals/high-level-adoption-electronic-health-records/docview/910125805/se-2)

[3] Amatayakul, Margret. (2009). Electronic Health Records: A Practical Guide for Professionals and Organizations.

[4] Professor Dr. Phyllis J. Watson, Head of School of Health Information Management (formerly), Faculty of Health Sciences, University of Sydney, Posted-1 Oct 2006 Originally published-1 Oct 2006, Medical Records Manual: A Guide for Developing Countries,

[5] Ajami S, Arab-Chadegani R. Barriers to implement Electronic Health Records (EHRs). Mater Sociomed. 2013;25(3):213-5. doi: 10.5455/msm.2013.25.213-215. PMID: 24167440; PMCID: PMC3804410.

[6] https://sharps.org/wp-content/uploads/STABLEIN-ESS-Annual-Meeting.pdf