DBMS REPORT

GROUP - 3

Project: Electronic Medical Record WebApplication

Name: Shah Javed Hossain
ID: 1820290

Name: Prantika Roy
ID: 2021040

Name:Munabbirul Saqlain
ID:1821013

Name:Mohammad Sufyan Rahman
ID:1920498

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.

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.

a) Literature Review - 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: [?]

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.

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.

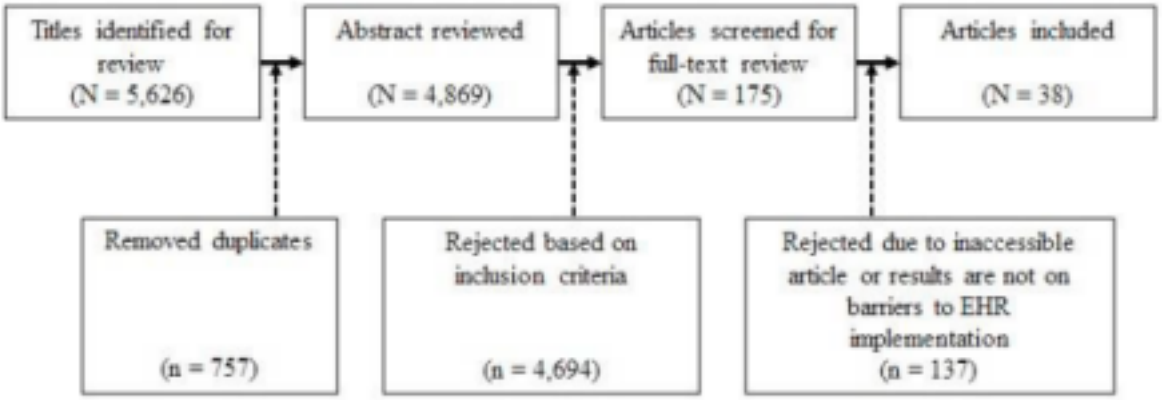


Figure 1: Flowchart of inclusion criteria

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

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.

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.

1.4.1 Paper Link

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

b) Literature Review - 2

2 Paper 2: PatientDataChain: A Blockchain-Based Approach to Integrate Personal Health Records

Journal/Conference Rank: Q1
Publication Year: 2022
Reference: [?]

2.1 Summary

The paper introduces PatientDataChain, a blockchain-based technology designed to address the current lack of a trusted infrastructure for exchanging and storing medical data in the healthcare industry. It highlights the challenges posed by the absence of a platform

for monitoring patients history across the healthcare chain, which creates communication errors. PatientDataChain aims to provide an integrated solution that utilizes wearable device sensors to collect patient data from various healthcare providers. This data is then integrated into a unified Personal Health Records (PHR) system, with the patient retaining ownership of their data.

PatientDataChain is decentralized, it uses blockchain technology, ensures secure, flexible, and reliable data sharing and exchange, enhancing data confidentiality and privacy. The article describes the system's design and implementation. The key contributions of the paper include the concept of making healthcare value chain interoperable.

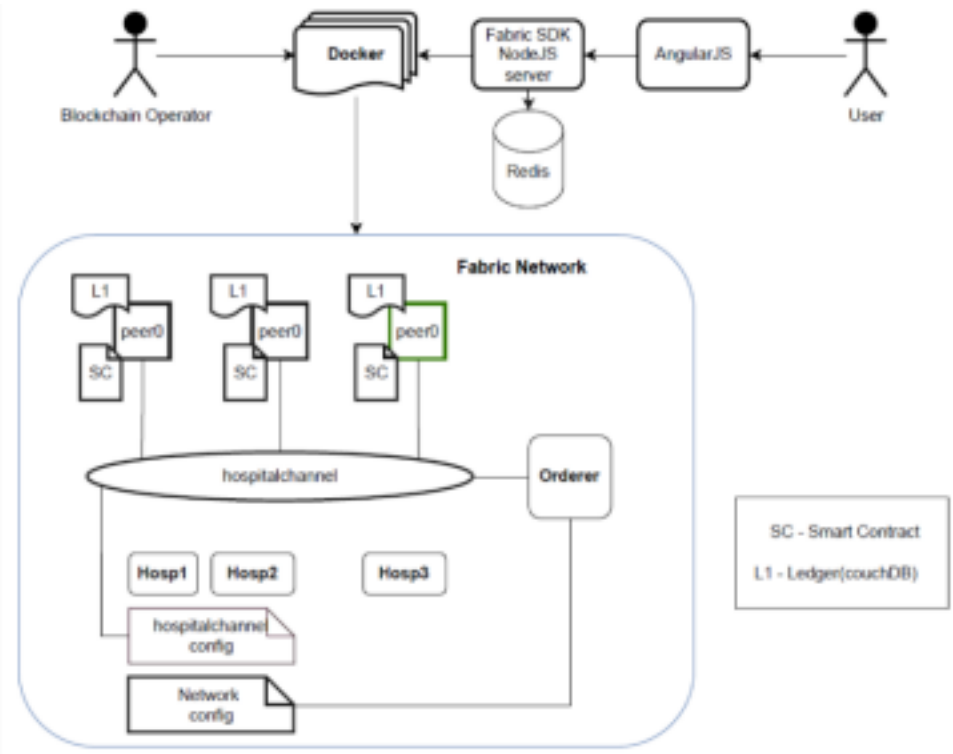


Figure 4: Software architecture diagram for Paper 1.

2.2 Software Architecture

The system uses Modex Blockchain Database (BCDB) which is a hybrid data storage system. Modex BCDB is middleware that exists between clients software and their regular database systems, this enables connection to a blockchain engine and has a similar interface as regular databases making it easier to use without additional training.

The Modex BCDB receives REST API calls directly, using GraphQL. The system can connect to multiple blockchains and database technologies. The main part of PatientDat aChain is the wallet app, functioning as the Personal Health Records (PHR) component. This app serves as a single source of truth, owned by the patient. Patients can grant read/write access to healthcare providers, enabling doctors to view their medical history and create new health records after appointments. Doctors can view patient history and the prescriptions can be stored in the database. Patients can provide access to pharmacies for viewing and even make online reservations for prescribed medications. The pharmacy notifies the patient when their order is ready for pickup. This system offers

Name	Type of System	Description
MedBlocks [56]	Electronic medical records (EMR)	P2P filesystem made immutable using distributed ledger technology, for storing medical records. Strength: improved consensus mechanism to ensure the network is not overloaded
Secure Health Chain [57]	EMR	Stores electronic personal medical records on a blockchain and ensures that only authorized parties, who have been granted permission, can access data.
MedicalChain [58]	EMR	Decentralized platform that enables storage, exchange and use of medical data. It enables the user to give healthcare professionals access to their personal health data.
Doc.AI [59]	Generate insights from medical data	Enterprise AI platform that uses natural language processing, computer vision and blockchain to generate insights from medical data.
OmniPHR [60]	PHR	Distributed system for storing patient health data. It stores different patient datasets into different blocks on the chain
MIStore [61]	Medical Insurance Storage System	A blockchain-based medical insurance storage system

Figure 5: Review of blockchain systems in healthcare

a patient-centric approach to managing health records and facilitates secure interactions with healthcare providers and pharmacies.

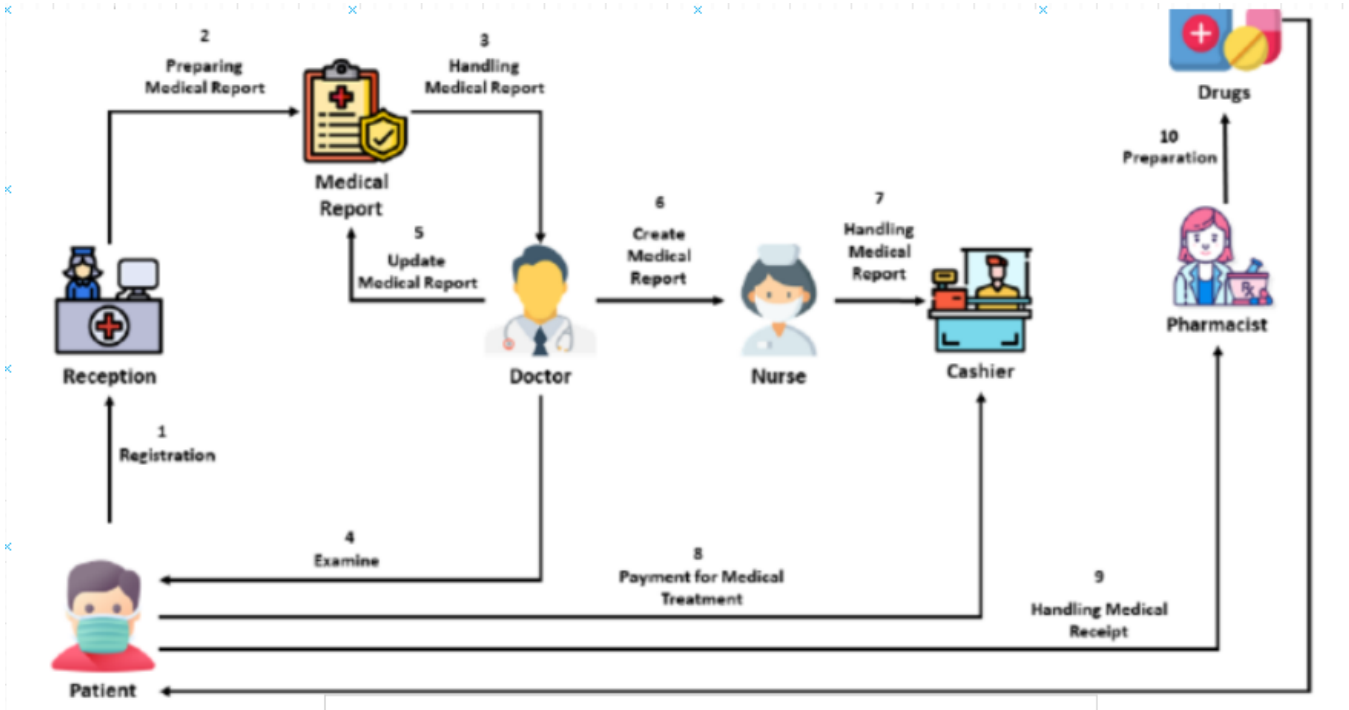
2.2.1 Paper Link

Access the full paper at https://www.mdpi.com/1424-8220/20/22/6538.

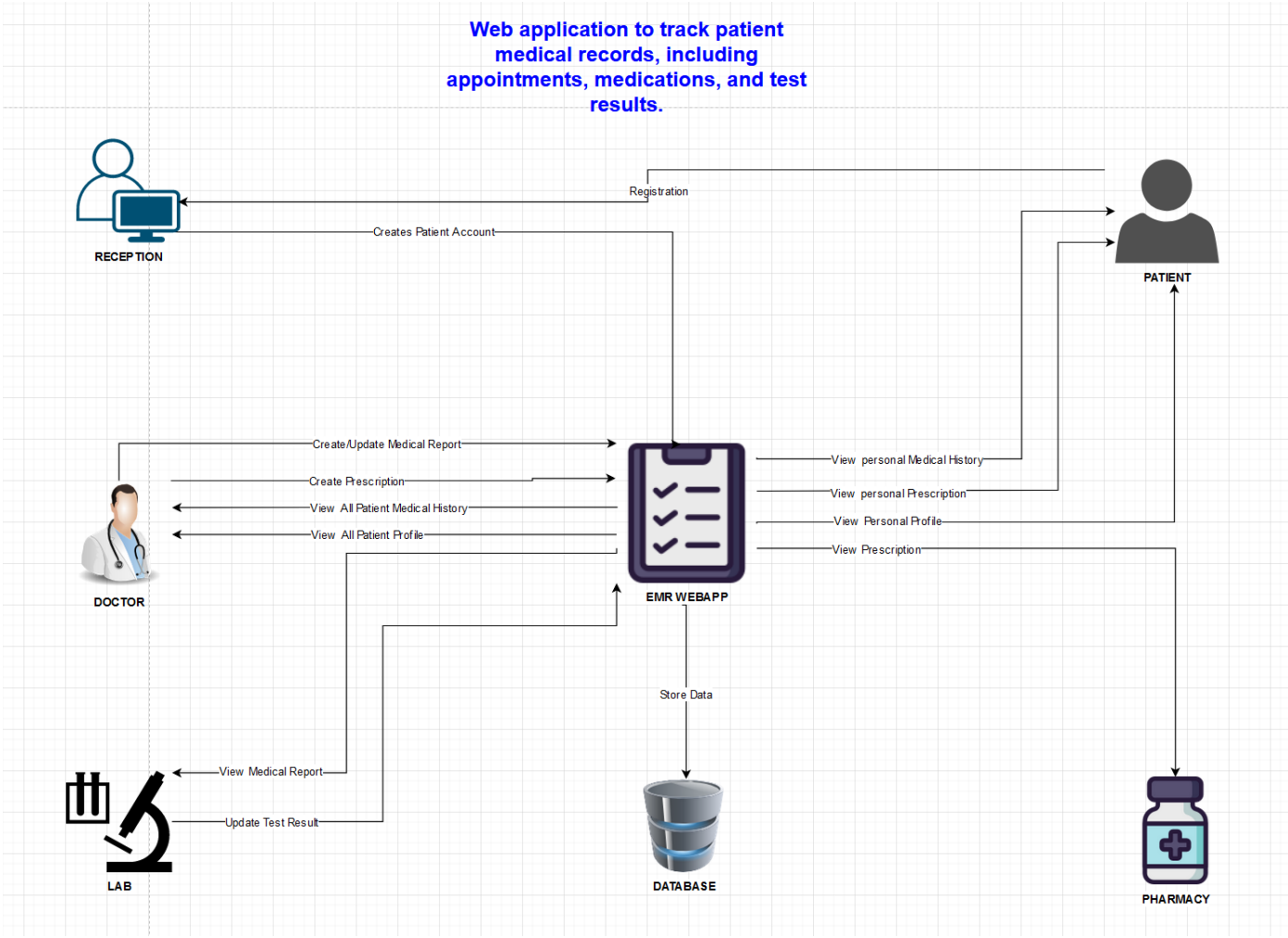
SYSTEM DESIGN

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.

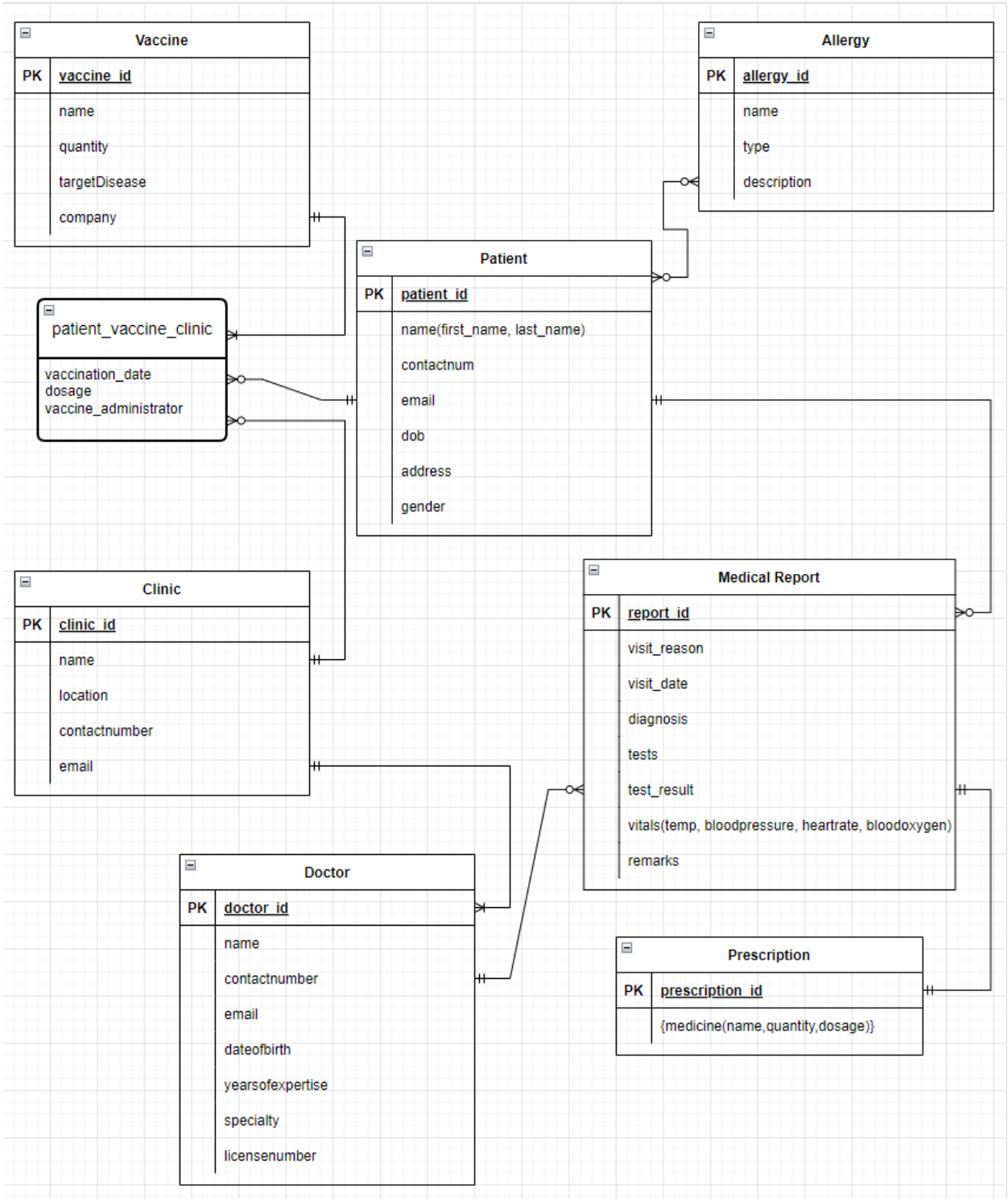
RICH PICTURE - AS IS



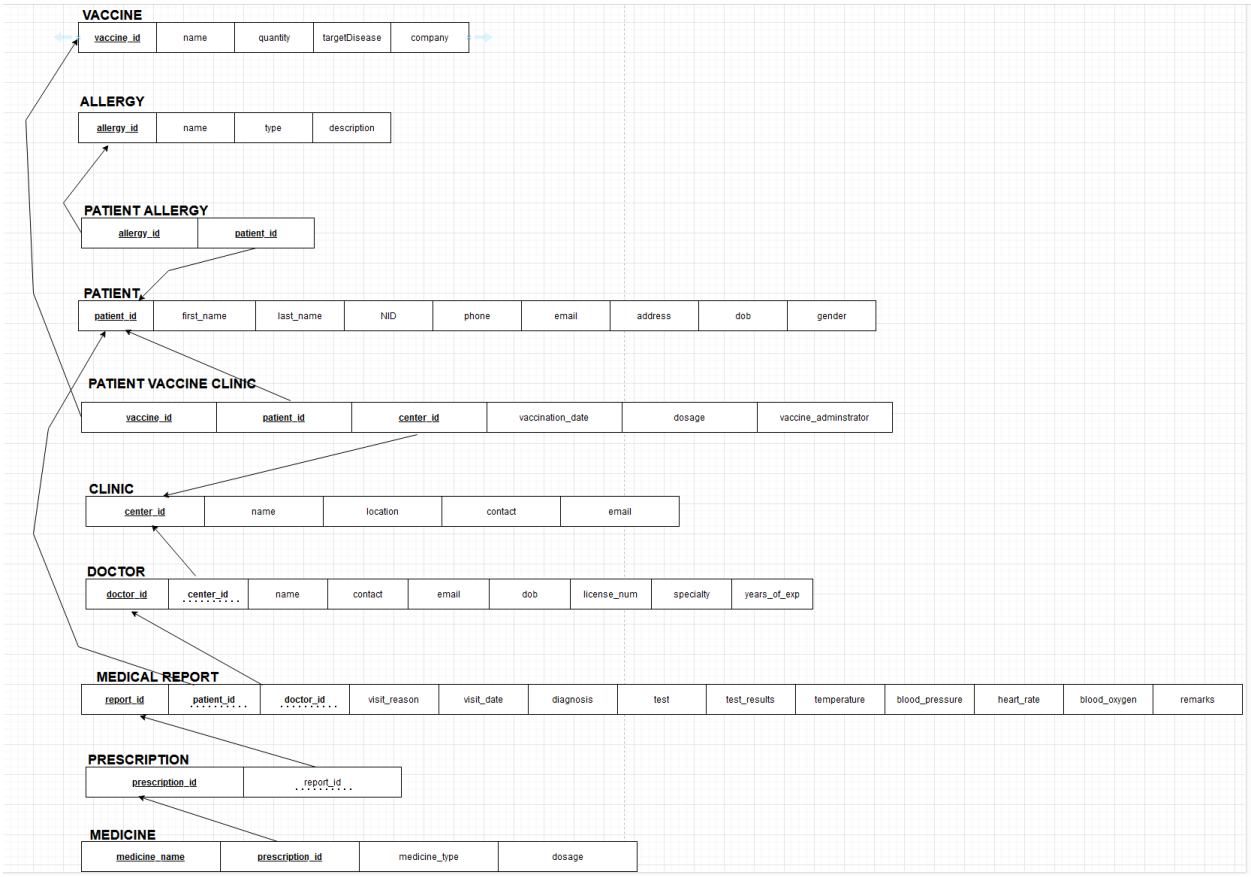
RICH PICTURE - TO BE



ERD


Vaccine
PK
vaccine_id
name
quantity
targetDisease
company
Allergy
PK
allergy_id
name
type
description
Patient
PK
patient_id
name(first_name, last_name)
contactnum
email
dob
address
gender
patient_vaccine_clinic
vaccination_date
dosage
vaccine_administrator
Clinic
PK
clinic_id
name
location
contactnumber
email
Medical Report
PK
report_id
visit_reason
visit_date
diagnosis
tests
test_result
vitals(temp, bloodpressure, heartrate, bloodoxygen)
remarks
Doctor
PK
doctor_id
name
contactnumber
email
dateofbirth
yearsofexpertise
specialty
licensenumber
Prescription
PK
prescription_id
{medicine(name,quantity,dosage)}

RELATIONSHIP SCHEMA



NORMALIZATION

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	

	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
PRESCRIPTION	MEDICINE	PR2		REMARKS	R6

	VACCINATION DATE	PVC1
PATIENT VACCINE CLINIC	DOSAGE	PVC2
	VACCINE ADMINISTRATION	PVC3

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.

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

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
PATIENT	vaccine_	VARCHA	255	yes	Name of vaccine

VACCINE CLINIC	administrator	R			administration

Table	Attribute	Data Type	Size	Required	Description
ALLERGY	allergy_id	INT	255	yes	Primary key
ALLERGY	name	VARCHAR	50	yes	Name of allergy
ALLERGY	type	VARCHAR	50	yes	Allergy type
ALLERGY	description	VARCHAR	255	yes	Description of allergy

Table	Attribute	Data Type	Size	Required	Description
PATIENT ALLERGY	allergy_id	INT	255	yes	Foreign key from ALLERGY table
PATIENT ALLERGY	patient_id	INT	255	yes	Foreign key from PATIENT table

Table	Attribute	Data Type	Size	Required	Description
MEDICAL REPORT	report_id	INT	255	yes	Medical report ID
MEDICAL REPORT	patient_id	INT	255	yes	Foreign key from PATIENT table
MEDICAL REPORT	doctor_id	INT	255	yes	Foreign key from DOCTOR table

MEDICAL REPORT	visit_reason	VARCHAR	255	yes	Reason of appointment
MEDICAL REPORT	visit_date	DATE	Y-M-D	yes	Date of appointment
MEDICAL REPORT	diagnosis	TEXT	255	yes	Diagnosis of patient
MEDICAL REPORT	test	VARCHAR	255	yes	Test given by doctor
MEDICAL REPORT	test_results	BLOB	BLOB	yes	Lab results in PDF file/format
MEDICAL REPORT	temperature	VARCHAR	56	yes	Body temperature of patient
MEDICAL REPORT	blood_pressure	INT	255	yes	Blood pressure of patient
MEDICAL REPORT	heart_rate	VARCHAR	50	yes	Heart rate of patient
MEDICAL REPORT	blood_oxygen	VARCHAR	50	yes	Blood oxygen level of patient
MEDICAL REPORT	remarks	TEXT	255	yes	Remarks made by doctor

Table	Attribute	Data Type	Size	Required	Description
PRESCRIPTION	prescription_id	INT	255	yes	Primary key
PRESCRIPTION	report_id	INT	255	yes	Foreign key from MEDICAL REPORT table

Table	Attribute	Data Type	Size	Required	Description
MEDICI	medicine_n	INT	255	yes	Medicine name

NE	ame				
MEDICINE	prescription_name	VARCHAR	255	yes	Foreign key from PRESCRIPTION table
MEDICINE	medicine_type	VARCHAR	255	yes	Type of medicine (capsule, tablet, cream, injection, etc.)
MEDICINE	dosage	VARCHAR	255	yes	Dosage of prescribed medicine

The goal of the methodology and implementation strategy is to provide an EMR Web Application that is safe, scalable, and easy to use, while also meeting the unique requirements of healthcare providers and keeping with legal requirements. Maintaining the security of the system and keeping it in line with changing healthcare practices requires regular upgrades and maintenance.

METHODOLOGY

Project:1(React + Nodejs)

FRONTEND Methodology

1. **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.

2. **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.

3. **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.

BACK END Methodology

1. **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.

2. **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.

3. **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.

Project:2 (Laravel)

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.

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.

IMPLEMENTATION

Project:1(React + Nodejs)

FRONTEND Implementation

1. **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.

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

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

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

BACKEND Implementation

1. **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.

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

3. **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).

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

5. **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.

Project:2 (Laravel)

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.

BACKEND Implementation

6. **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.

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

8. **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).

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

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

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.

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.

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.

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.

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.

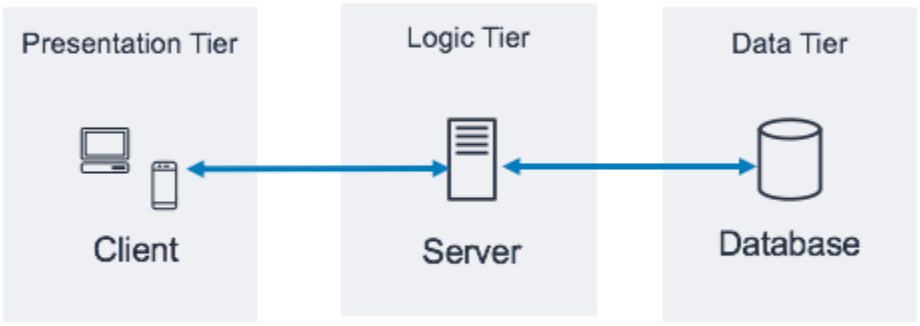
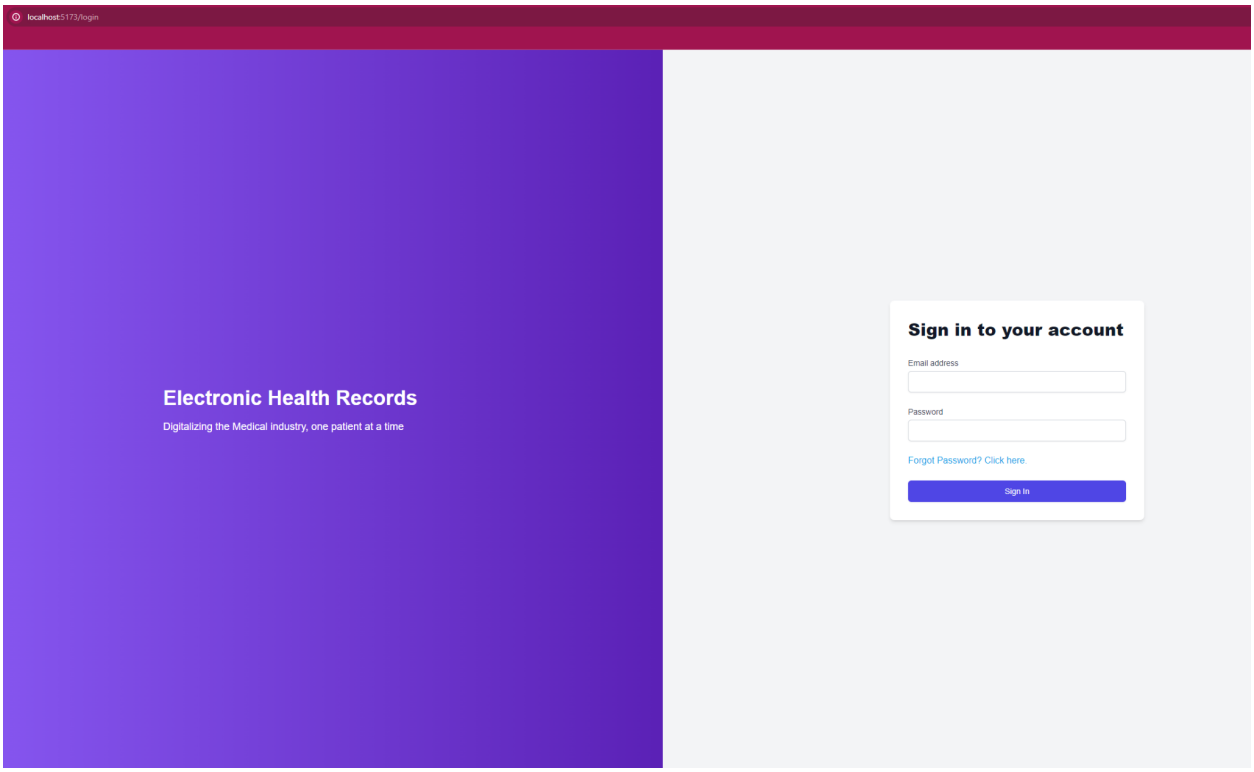


Fig: 3 Tier Architecture

INTERFACE

Project:1 (React + Nodejs)

ALL User: Login



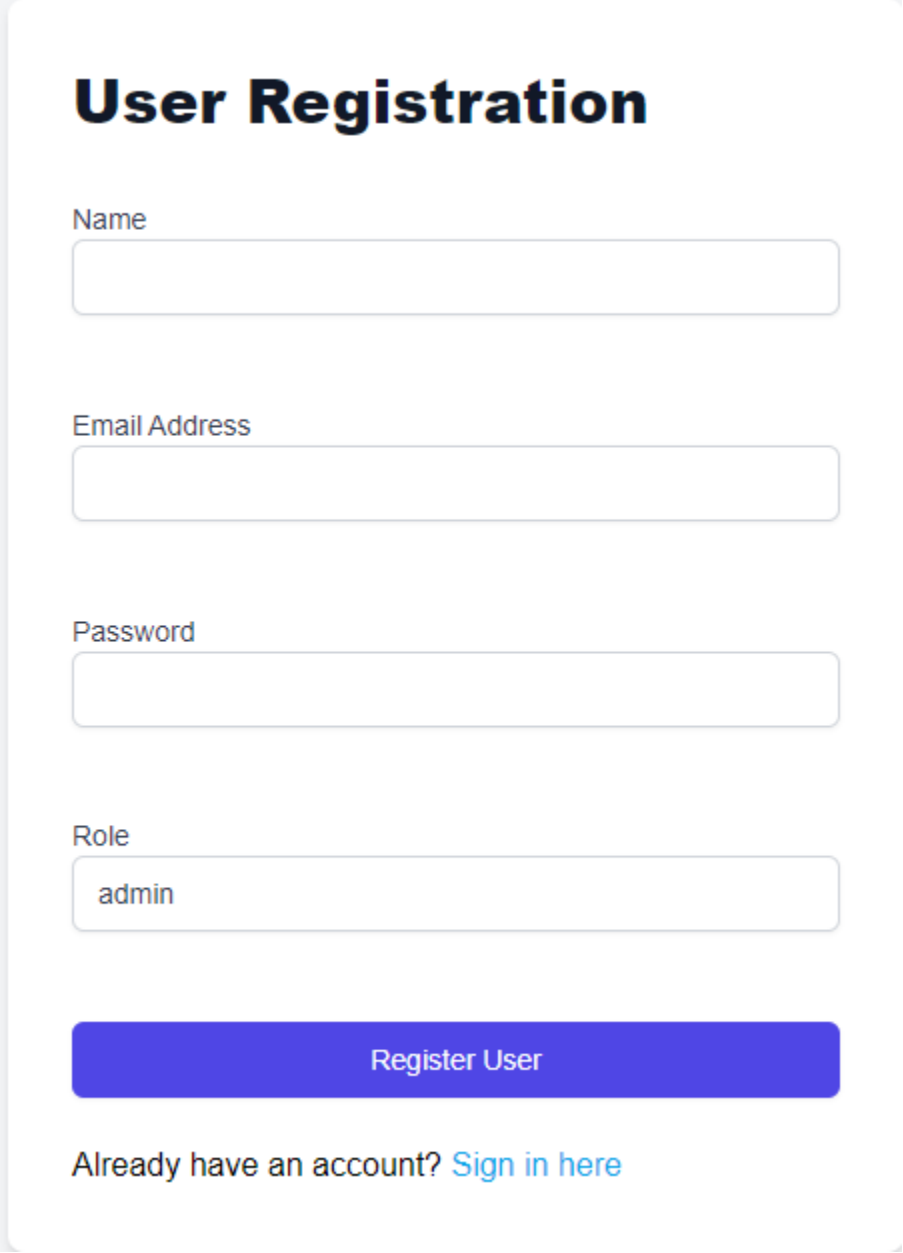
All User Login features (Functionality, mySQL commands)

- 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:

```
// 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")
            }
        }
    })
})
```

ALL User: Registration



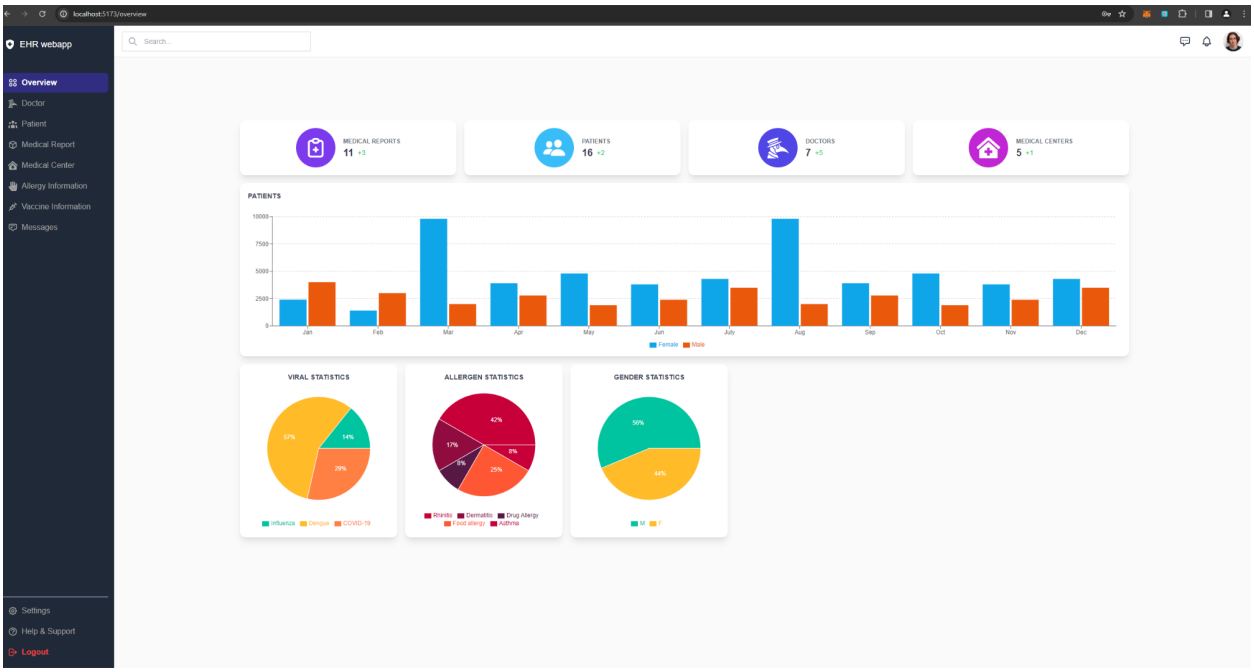
Features

- 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:

```
// 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();
});
```

Admin Dashboard: Overview



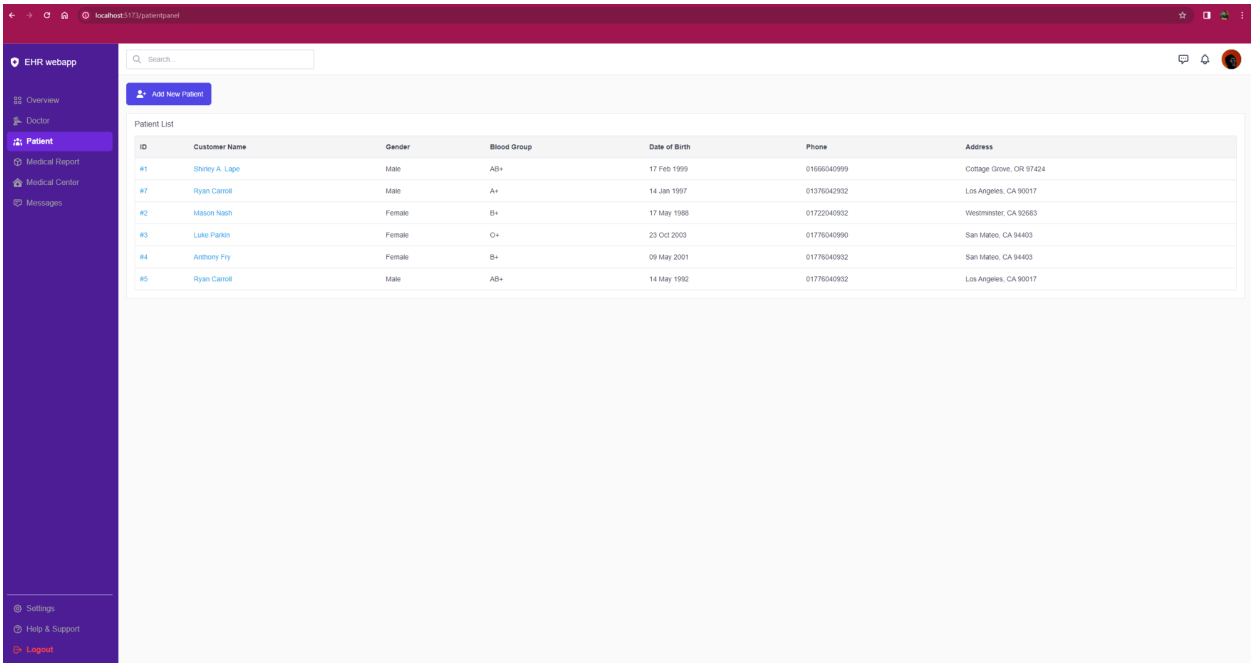
Overview Features

- 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

```
// 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)
    })
```

Admin Dashboard: Patient Panel



Patient Panel Features

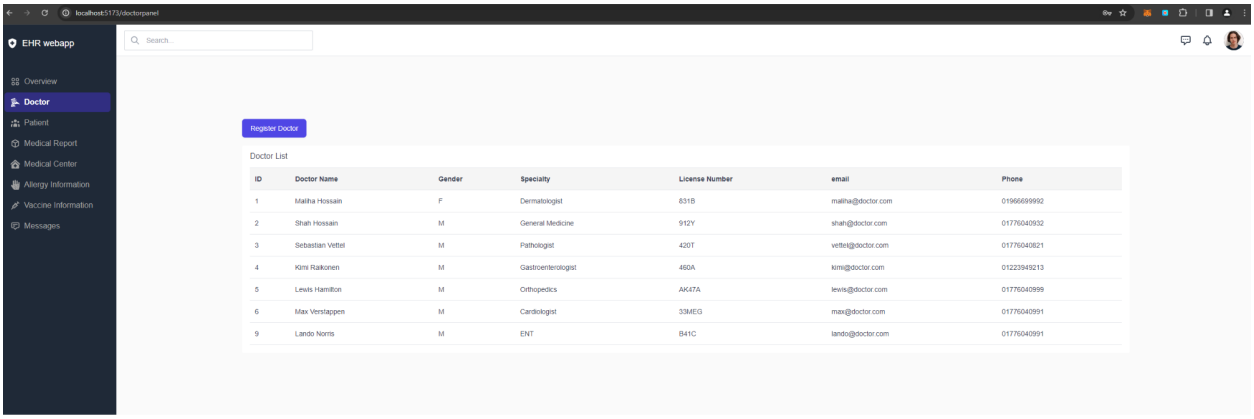
- Display a minimalist table which shows all patient data.
- Data include, patient name, gender, date of birth, address, blood group and phone number.

- 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

```
226     //ALL patient table API
227     app.get('/api/patient_table', (req, res) => {
228         const sqlRetrieve = "SELECT * FROM user"
229         db.query (sqlRetrieve, (err, data) => {
230             if(err) return res.json(err)
231             return res.json(data)
232         })
233     });
```

Admin Dashboard: Doctor Panel



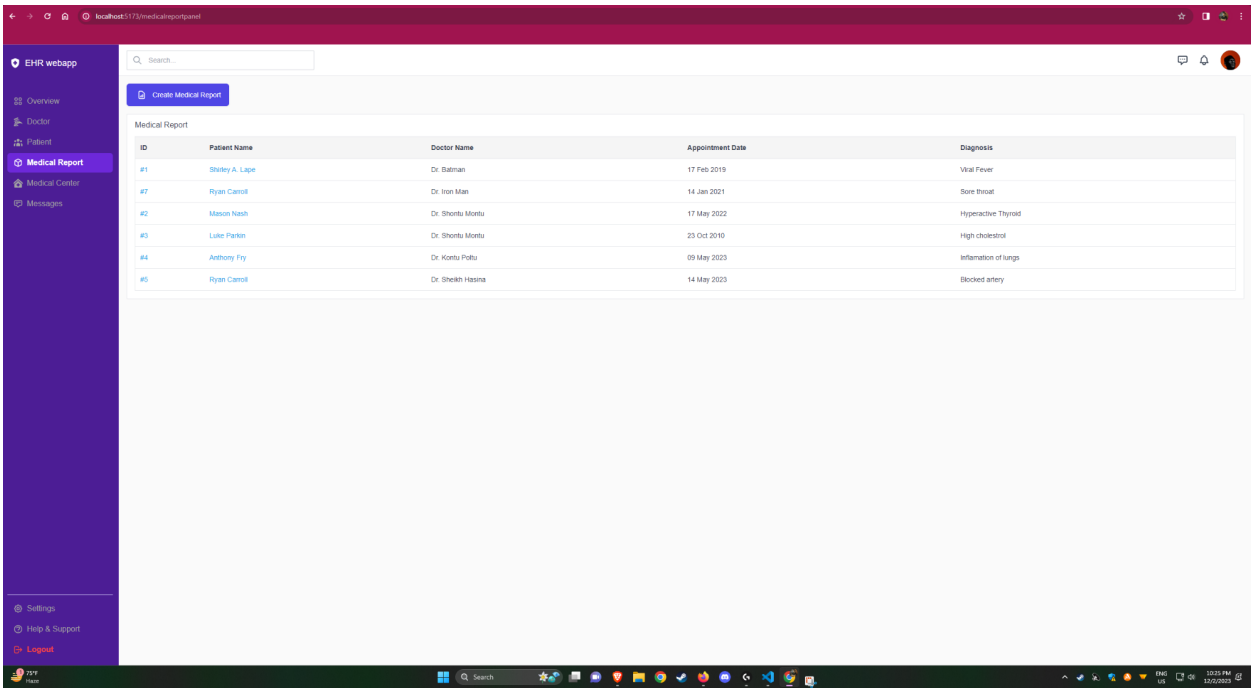
Features

- Shows all doctors
- Can register new doctor

SQL Statements:

```
//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)
    })
});
```

Admin Dashboard: Medical Report Panel



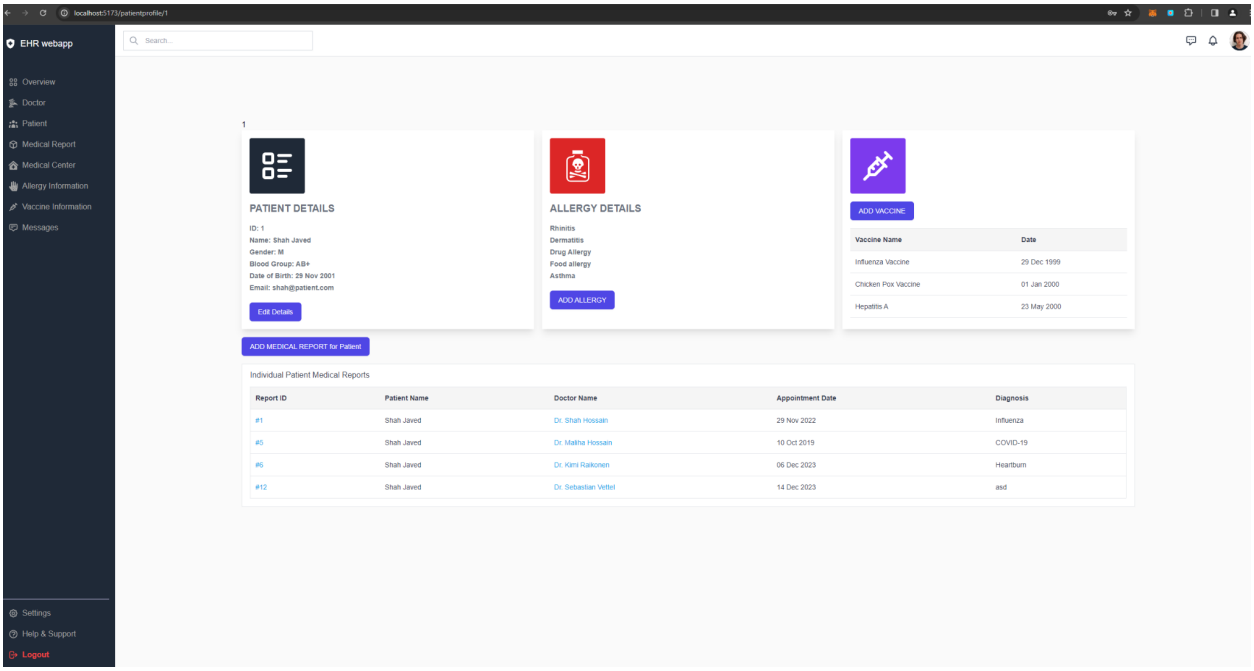
Patient Panel Features

- 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

```
278    //ALL REPORT API
279    app.get('/api/report_table', (req, res) => {
280        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
281        db.query (sqlRetrieve, (err, data) => {
282            if(err) return res.json(err)
283            return res.json(data)
284        })
285    });
```

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

```
//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();
});
```

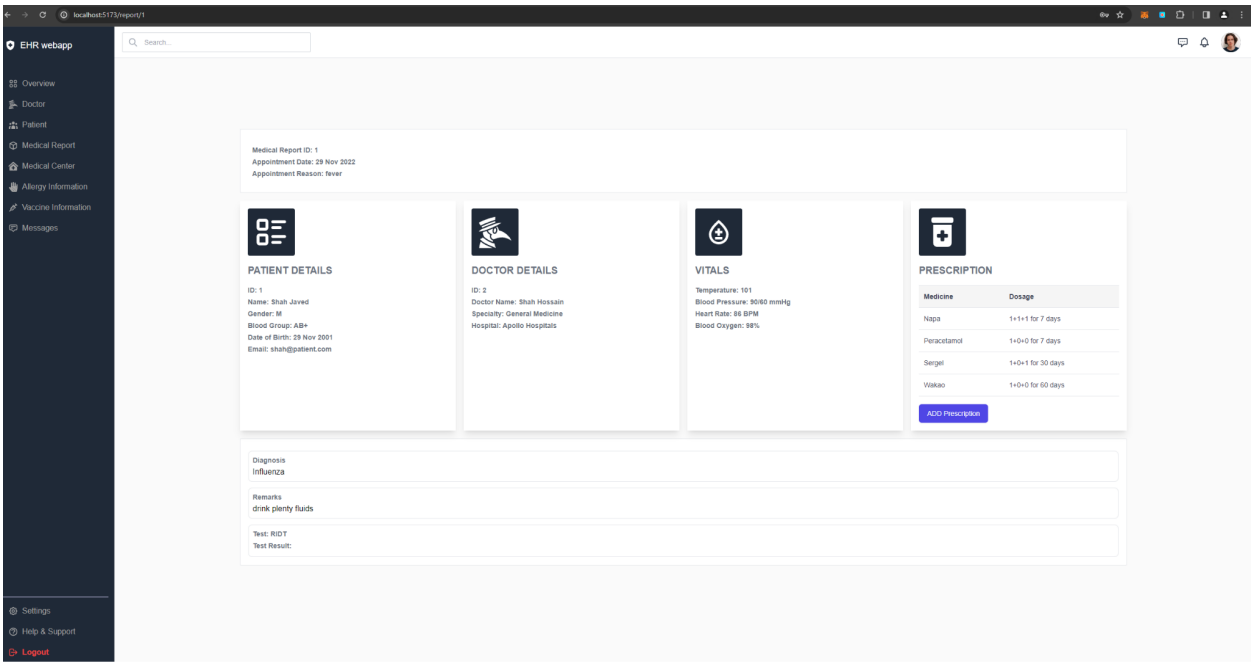
```
//Input VACCINATION TO PATIENT
app.post('/api/add_vaccine/:id', (req, res) => {
    console.log(req.body);
    const patient_id = req.params.id
    const vaccine_id = req.body.vaccine_id
    const vaccination_date = req.body.date
    const vaccine_administrator = req.body.vaccine_adminstrator


    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();
});
```

```
//INDIVIDUAL MEDICAL REPORT API (ALL report for specific patient)
app.get(`/api/patient/medical_report/:id`, (req, res) => {
    const patient_id = req.params.id

    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
    db.query (sqlRetrieve, [patient_id], (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});
```

```
267     //PATIENT ALLERGY API
268     app.get(`/api/patient/allergy/:id`, (req, res) => {
269         const patient_id = req.params.id
270
271         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
272         db.query (sqlRetrieve, [patient_id], (err, data) => {
273             if(err) return res.json(err)
274             return res.json(data)
275         })
276     });
```

```
//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)
    })
});
```

Admin Dashboard: In-depth Medical Report



In depth Medical Report Feature

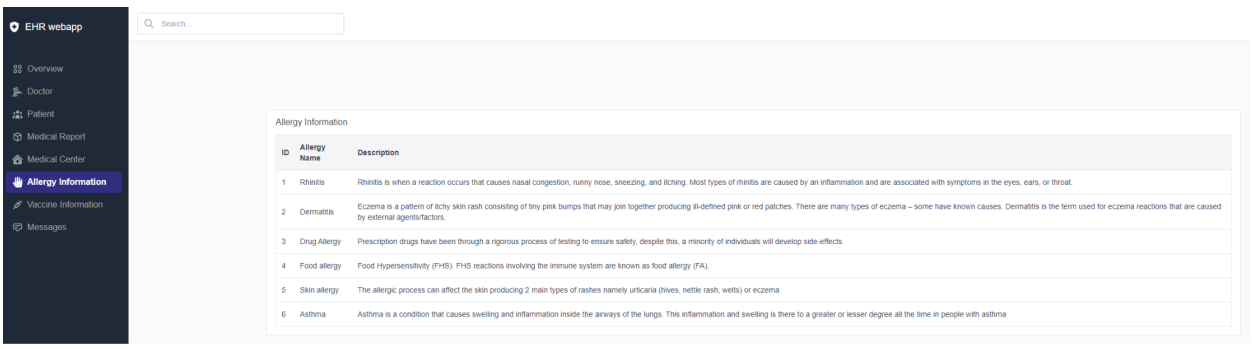
- More in depth details of the medical report.
- Shows vitals of patient for given medical report
- Shows prescription for the given medical report
- Shows doctor details of the given medical report
- Shows patient details of the given medical report
- Ability to add more medicine to prescription

SQL Statement

```
//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();
});
```

```
// 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.special
    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 = prescriptic
    db.query (sqlRetrieve, [report_id], (err, data) => {
        if(err) return res.json(err)
        return res.json(data)
    })
});
```

Allergy Database Page

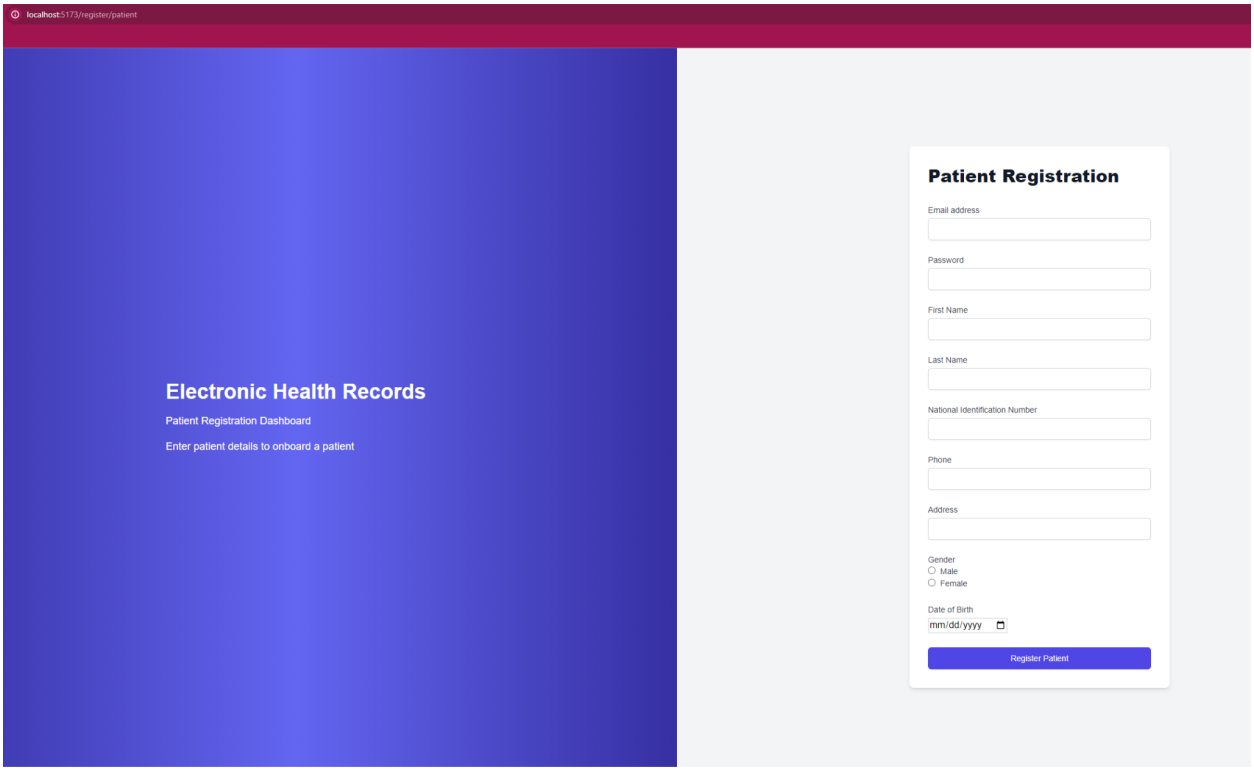


Features

- Shows allergy information from existing database

SQL Statement

Registration: Patient



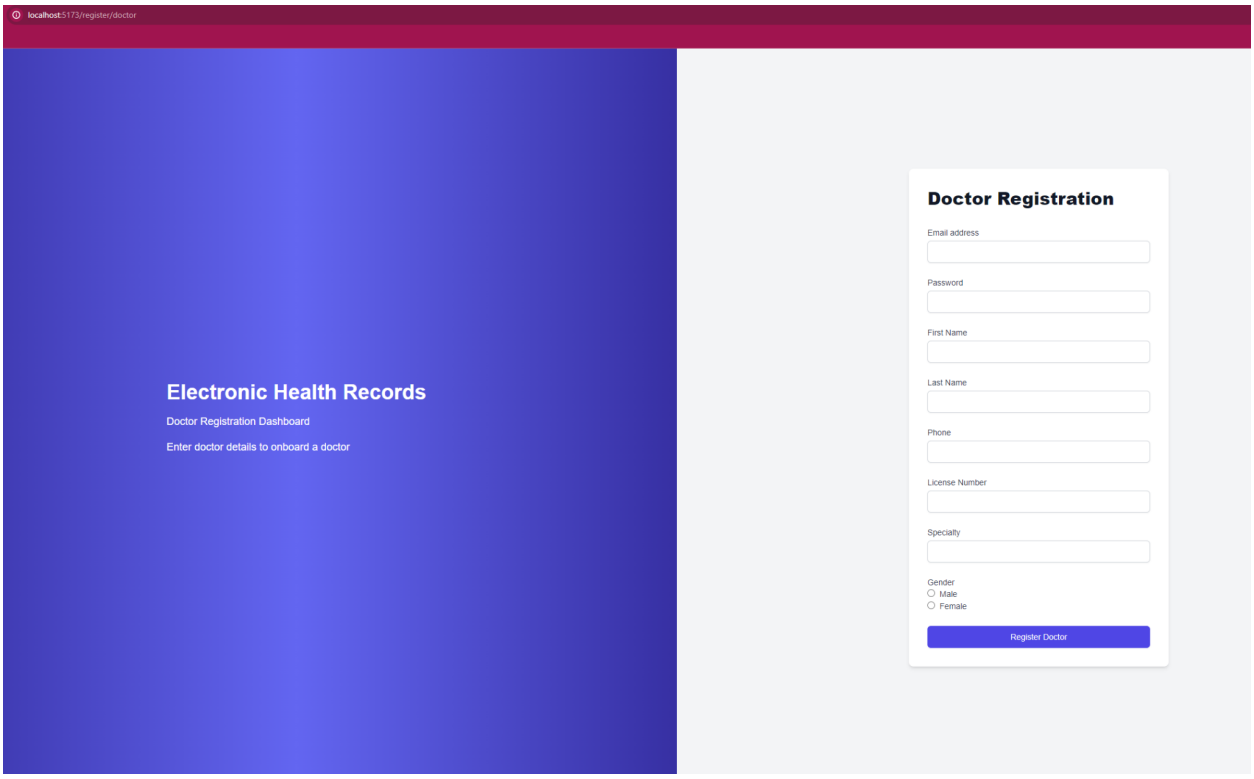
Registration Patient Features(Functionality, mySQL commands)

- Ability to register new patients into the database.

SQL STATEMENT

```
// 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();
});
```

Registration: Doctor



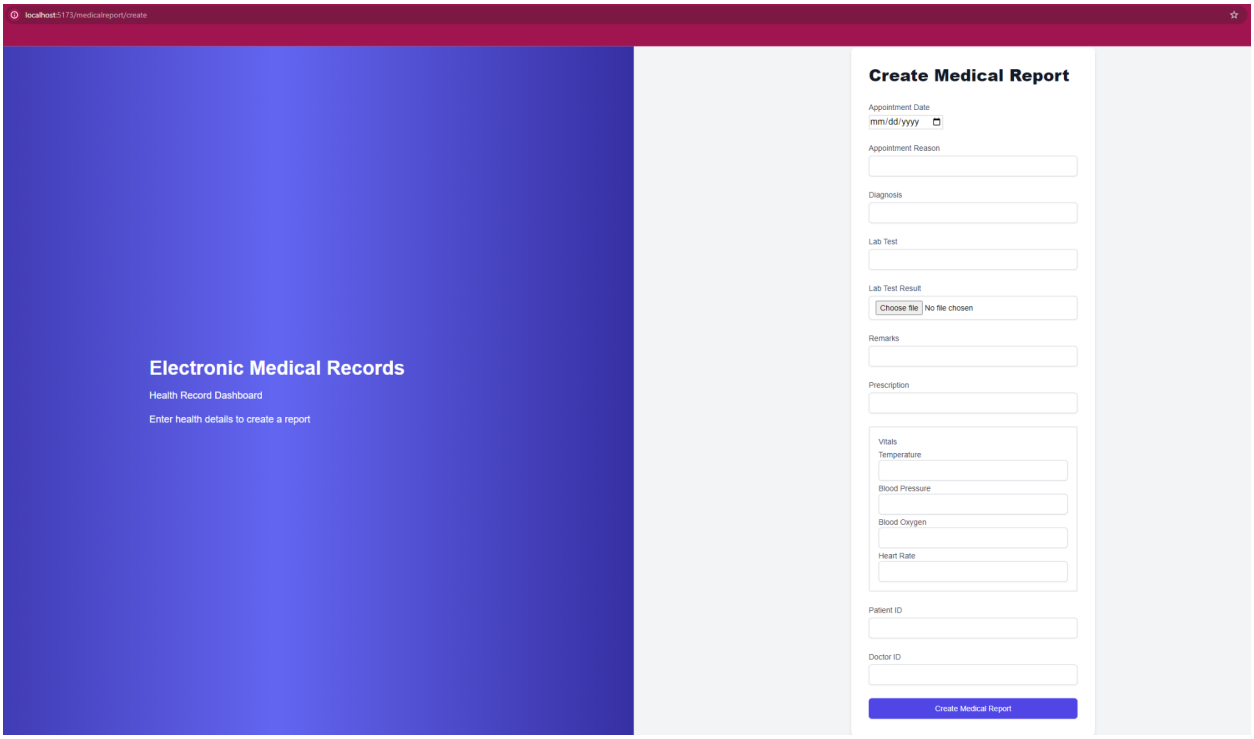
Registration Doctor Features (Functionality, mySQL commands)

- Input form to add new doctor into the database.

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();
});
```

Create: Medical Report



Medical report creation Details (Functionality, mySQL commands)

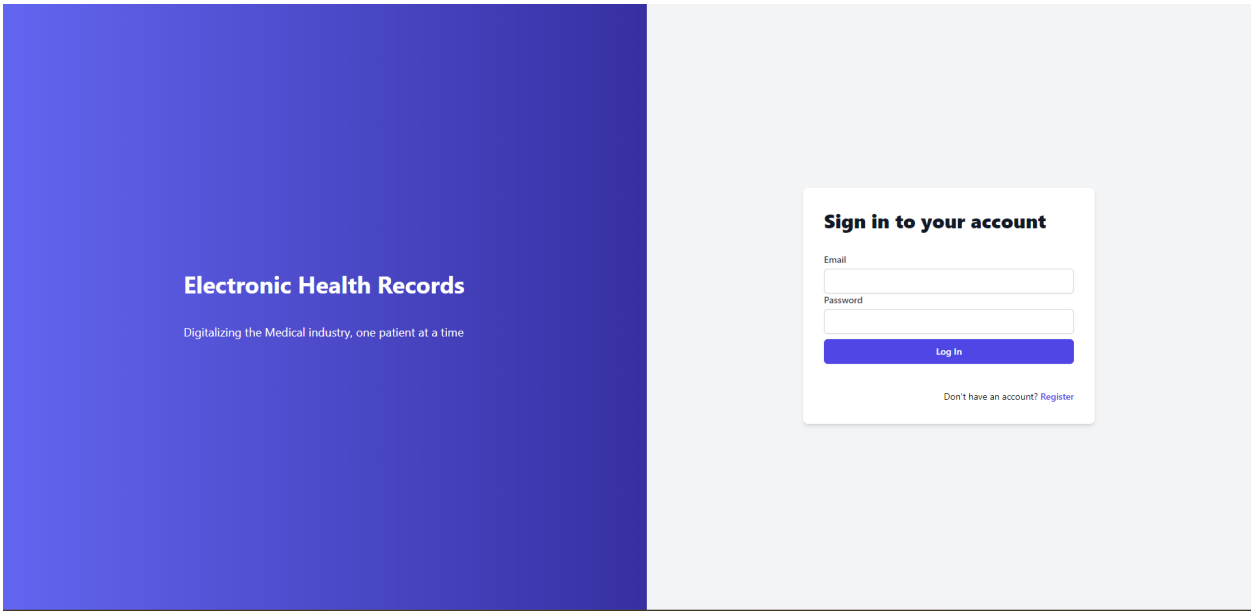
- Input form to make new Medical Report

SQL Statement:

```
//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();
});
```

Project:2 (Laravel)

ALL User: Login



All User Login features (Functionality, mySQL commands)

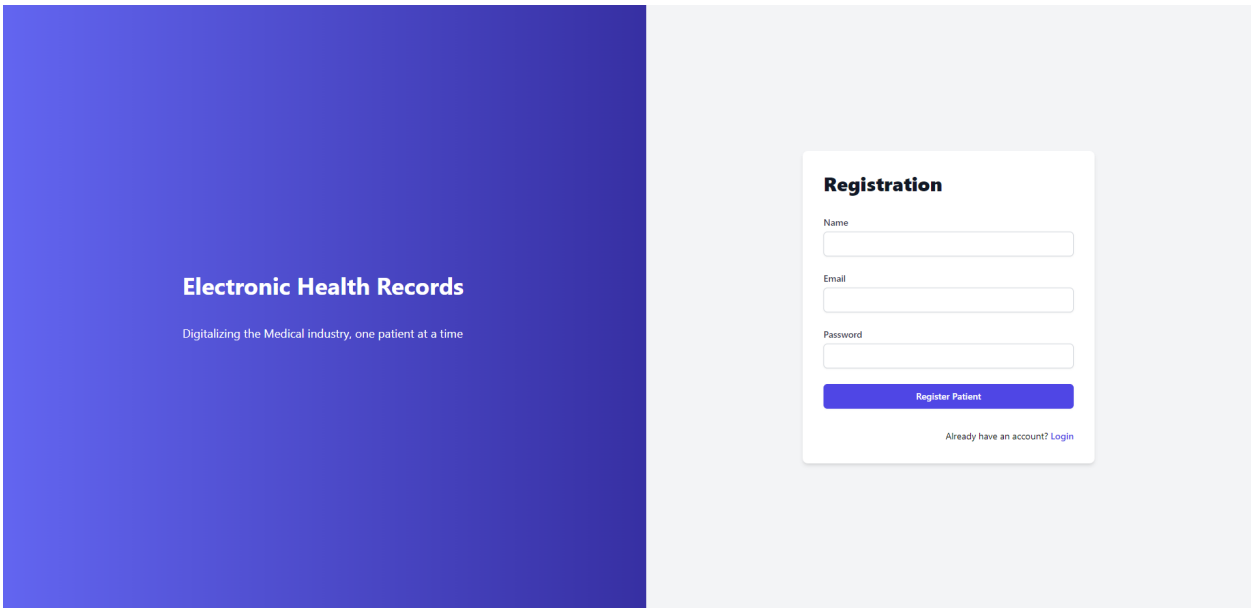
- Checks for user in database, if user doesn't exist, sends "invalid login credentials" error
- Checks for user in database, if user exists but wrong password says "invalid login credentials"

- Checks for user in database, if user exists and types correct password, successfully logs in and generates a session() for future requests

Backend Code:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use App\Providers\RouteServiceProvider;

class LoginController extends Controller
{
    public function login()
    {
        return view ('auth.login');
    }

    public function login_post(Request $request)
    {
        $success = auth()->attempt([
            'email' => request('email'),
            'password' => request('password')
        ]);

        if($success)
        {
            $request->session()->regenerate();
            return redirect()->to(RouteServiceProvider::HOME);
        }
        return back()->with('loginError', 'Invalid login details');
        // return back()->withErrors([
        //     'email' => 'The Credentials do not match.'
        // ])->onlyInput('email');
    }
}
```

ALL User: Registration

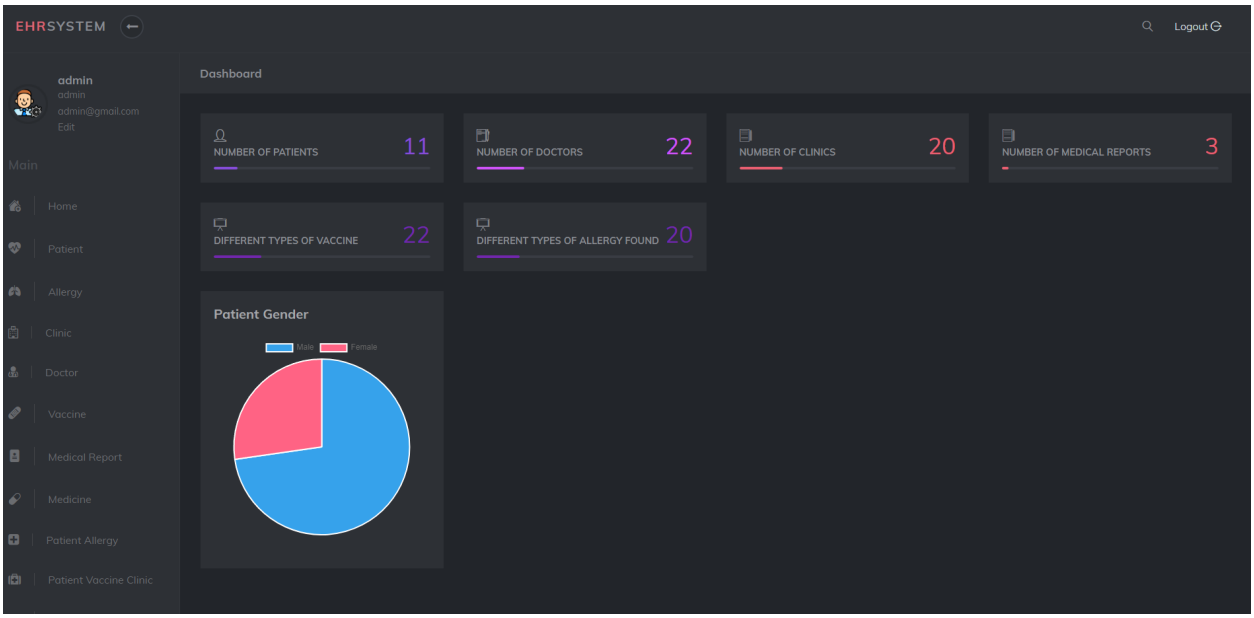


- Ability to register user
- Available roles - admin, patient, doctor
- During registration, searches for duplicate email, if duplicate found, shows an error ‘user already exists’.

Backend Code:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use App\Providers\RouteServiceProvider;

class RegisterController extends Controller
{
    public function register()
    {
        return view ('auth.register');
    }

    public function register_post()
    {
        $user = User::create([
            'name' => request('name'),
            'email' => request('email'),
            'password' => Hash::make(request('password'))
        ]);

        request()->validate([
            'name' => ['required', 'string' , 'max:255'],
            'email' => ['required', 'email', 'max:255'],
            'password' => ['required', 'string', 'min:8', 'confirmed']
        ]);

        Auth::login($user);

        return redirect()->to(RouteServiceProvider::HOME);
    }
}

```

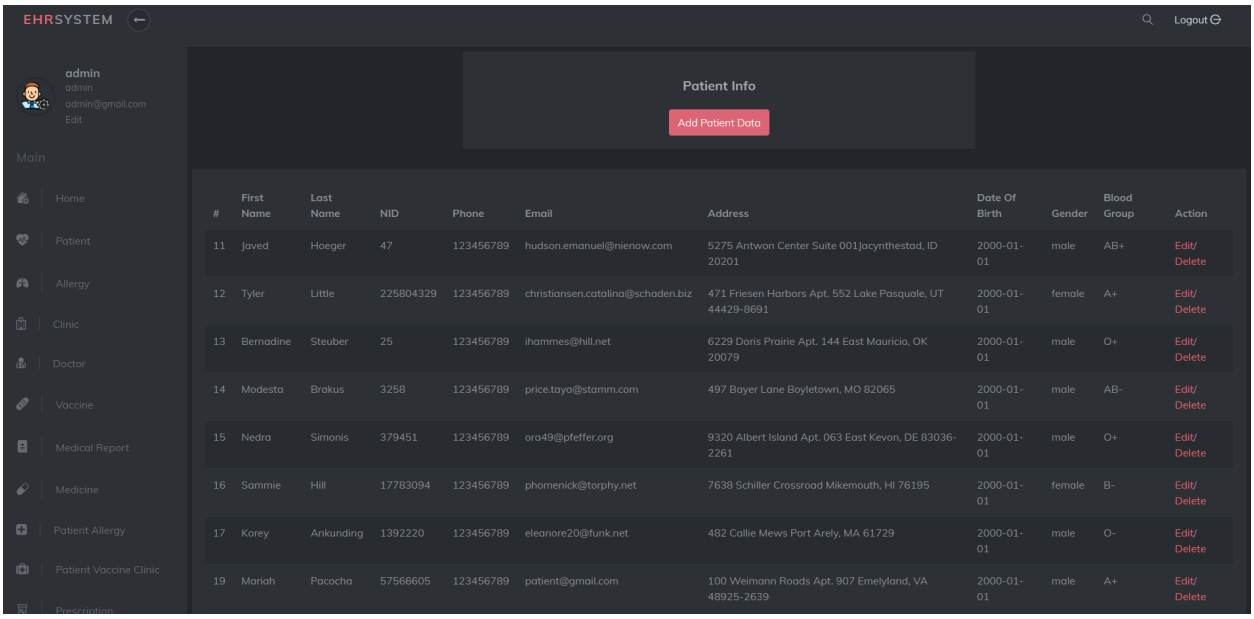
Admin Dashboard: Overview



Code of PIECHART:

```
        <div class="col-lg-3">
                <div class="pie-chart chart block">
                  <div class="title"><strong>Patient Gender</strong></div>
                  <div class="pie-chart chart margin-bottom-sm">
                    <canvas id="piePatientGender"></canvas>
                  </div>
                </div>
              </div>
    </section>

</div>
<?php
  $patientmale = DB::table('patients')->where('gender', 'male')->count();
  $patientfemale = DB::table('patients')->where('gender','female')->count();
?>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
  var ctx = document.getElementById('piePatientGender').getContext('2d');
  var piePatientGender = new Chart(ctx, {
    type: 'pie',
    data: {
      labels: ['Male', 'Female'],
      datasets: [{
          data: [<?php echo $patientmale; ?>, <?php echo $patientfemale; ?>],
        backgroundColor: ['#36a2eb', '#ff6384']
      }]
    },
    options: {
      responsive: true,
      maintainAspectRatio: false
    }
  });
</script>
```

Admin Dashboard: Patient Panel



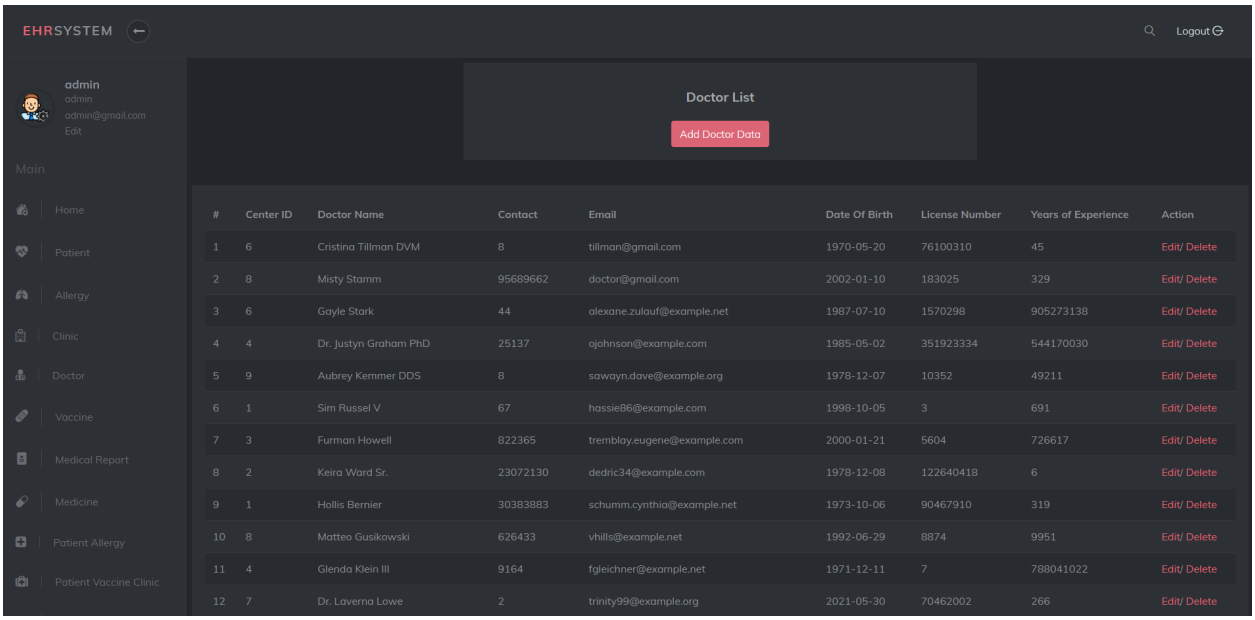
Patient Panel Features

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

Backend Code to view data:

```
public function patient()
{
    $patient = Patient::all();
    return view('admin.pages.patient', compact('patient'));
}
```

Admin Dashboard: Doctor Panel



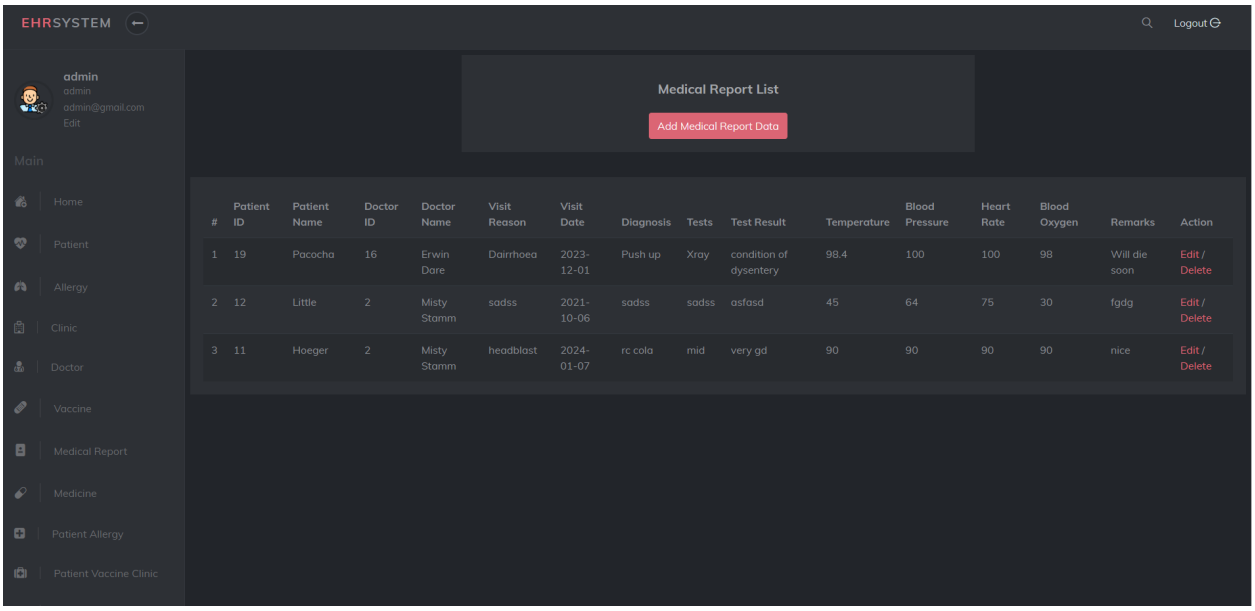
Features

- Shows all doctors
- Can register new doctor

Backend to view data:

```php
public function doctor()
{
    $doctor = Doctor::all();
    $clinic = Clinic::all();
    $cl = Clinic::all();
    return view('admin.pages.doctor', compact('clinic', 'doctor', 'cl'));
}
```

Admin Dashboard: Medical Report Panel



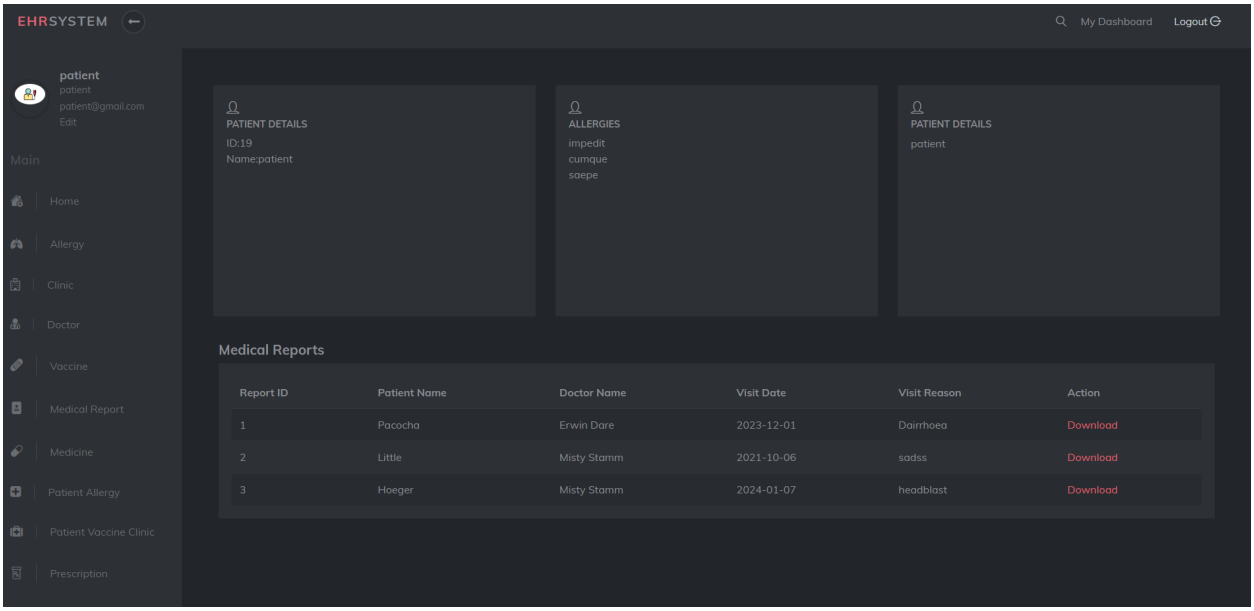
Patient Panel Features

- 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 databas

Backend to view data:

```php
public function medical_report()
{
    $patient = Patient::all();
    $pt = Patient::all();
    $doctor = Doctor::all();
    $doc = Doctor::all();
    $medicalreport = DB::table('medical_reports')
        ->join('patients','medical_reports.patient_id','=','patients.id')
        ->join('doctors','medical_reports.doctor_id','=','doctors.id')
        ->select('medical_reports.*','patients.last_name','doctors.doctor_name', 'doctors.email', 'patients.email')
        ->get();
    return view('admin.pages.medicalreport', compact('patient', 'doctor','medicalreport', 'pt', 'doc'));
}
```

Admin Dashboard: In-depth Medical Report



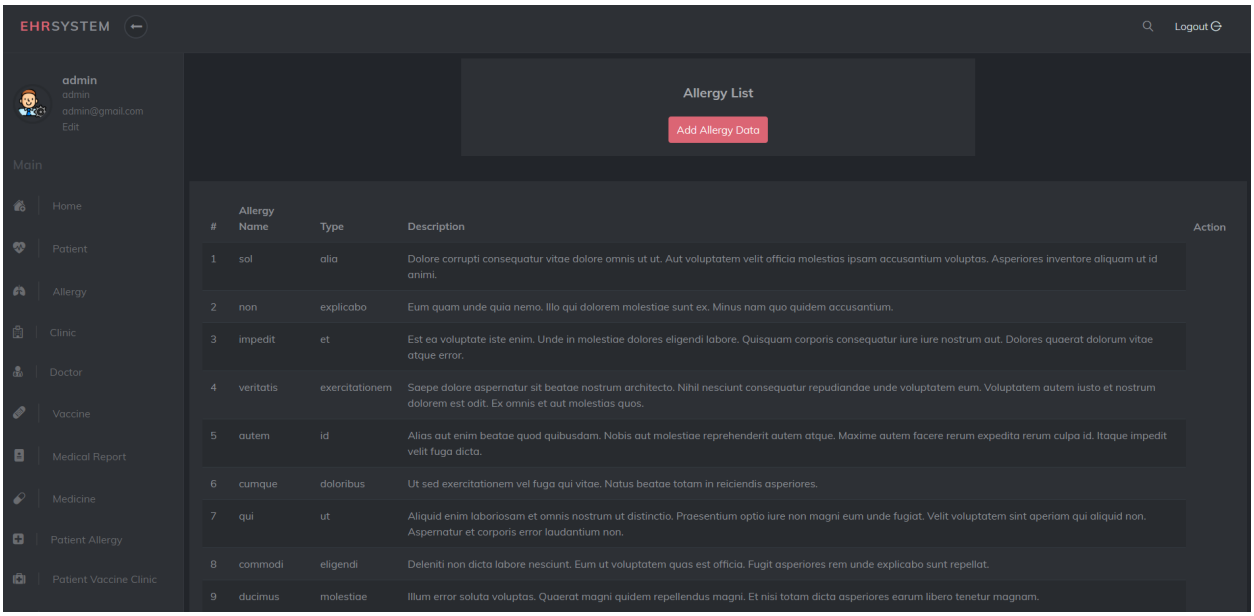
User Profile Feature:

- 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

Backend to view data:

```php
public function patientdata($id)
{
    $pd = DB::table('patients')
        ->join('patient_allergies','patients.id','=','patient_allergies.patient_id')
        ->join('allergies','patient_allergies.allergy_id','=','allergies.id')
        ->select('patients.*','patient_allergies.*','allergies.allergyname','allergies.type')
        ->where('patients.id','=',$id)
        ->get();

    $pd1 = $pd;

    $mlr = DB::table('medical_reports')
        ->join('patients','medical_reports.patient_id','=','patients.id')
        ->join('doctors','medical_reports.doctor_id','=','doctors.id')
        ->select('medical_reports.*','patients.last_name','doctors.doctor_name', 'doctors.email', 'patients.email')
        ->get();

    return view('admin.pages.patientdata',compact('mlr','pd','pd1'));
}
```

Allergy Page



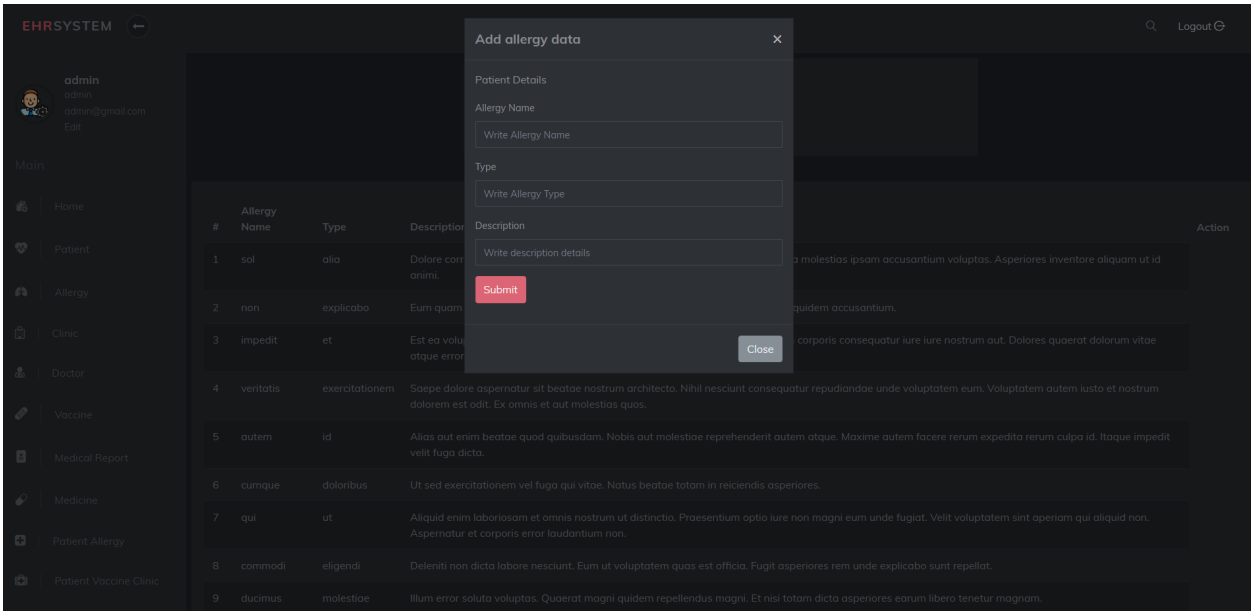
Features

- Shows allergy information from existing database

Backend to view data:

```php
public function allergy()
{
    $allergy = Allergy::all();
    return view('admin.pages.allergy', compact('allergy'));
}
```

Create Allergy Data:



Features

- Can add more Allergy Data.

Backend:

```php
public function allergy_submit(Request $request)
{
    $allergy = new Allergy();

    $allergy->allergyname = $request->name;
    $allergy->type = $request->type;
    $allergy->description = $request->description;

    $allergy->save();

    return redirect()->back();
}
```

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.

CONCLUSION & 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