Docs_on_Blocks – A Defense in Depth Strategy for E-Healthcare

Saad Mohammed
California State University - San Bernardino

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd

Part of the Business Administration, Management, and Operations Commons, Business Analytics Commons, Business Intelligence Commons, Computer and Systems Architecture Commons, Data Storage Systems Commons, Health Information Technology Commons, and the Management Information Systems Commons

Recommended Citation
https://scholarworks.lib.csusb.edu/etd/1164

This Project is brought to you for free and open access by the Office of Graduate Studies at CSUSB ScholarWorks. It has been accepted for inclusion in Electronic Theses, Projects, and Dissertations by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
DOCS_ON_BLOCKS – A DEFENSE IN DEPTH STRATEGY FOR E-HEALTHCARE

A Project
Presented to the Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Information Systems and Technology

by
Saad Mohammed

December 2020
DOCS_ON_BLOCKS – A DEFENSE IN DEPTH STRATEGY FOR E-HEALTHCARE

A Project

Presented to the

Faculty of

California State University,

San Bernardino

by

Saad Mohammed

December 2020

Approved by:

Dr. Frank Lin, Committee Chair, Information & Decision Sciences

Dr. Conrad Shayo, Committee Member

Dr. Jay Varzandeh, Committee Member
ABSTRACT

With the increase in the data breaches and cyber hacks, organizations have come to realize that cyber security alone would not help as the attacks are becoming more sophisticated and complex than ever. E-Healthcare industry has shown a promising improvement in terms of security over the past, but the threat remains. Thus, the E-Healthcare industries are aiming towards a Defense in Depth Strategy approach.

The project here describes how a Defense in Depth Strategy for E-Healthcare system can provide an environment for better security of the data and peer-to-peer interaction with stakeholders. The legacy systems have at some point failed to provide successful results in terms of data accessibility, and storage with changing technologies.

With the proposed model here many of the issues related to modern day healthcare would be solved to make it easier for the related stakeholders. Also, with the health insurance companies as one of the stakeholders in the model makes it even more robust and reliable. One of the most important features of the system being created here is for the people in need of emergency health aid. Thus, making it easier for the patients to avoid the appointments in case of an emergency.
ACKNOWLEDGEMENTS

I would like to express my special thanks and sincere gratitude to my advisor Prof. Frank Lin, who supported me in completing and reviewing my project.

I would also like to thank my graduate professor Dr. Conrad Shayo, for his valuable suggestions and support.

Finally, I want to thank my family and friends, who stood by my side in every situation, supporting me at all hard times and for making me stronger. I am grateful to have them in my life.
# TABLE OF CONTENTS

**ABSTRACT** ............................................................................................................................. iii

**ACKNOWLEDGEMENTS** ........................................................................................................ iv

**LIST OF FIGURES** .................................................................................................................... ix

**CHAPTER ONE: INTRODUCTION** ........................................................................................... 1

1.1 Chapter Overview .................................................................................................................. 1

1.2 Problem Question ................................................................................................................ 1

1.3 Problem Statement .............................................................................................................. 3

1.4 Problem Proposed Solution ................................................................................................ 4

1.5 Project Scope ....................................................................................................................... 5

**CHAPTER TWO: REQUIREMENTS AND SPECIFICATIONS** ................................................. 7

2.1 Chapter Overview .................................................................................................................. 7

2.2 End Users ............................................................................................................................. 7

2.3 System Requirement Specification ....................................................................................... 8

2.4 Actors and Use Cases ........................................................................................................... 8

2.5 Modules ................................................................................................................................ 10

2.6 Resource Requirements ....................................................................................................... 14

  2.6.1 Hardware Requirements .................................................................................................. 14

  2.6.2 Software Requirement: .................................................................................................... 15
2.6.3 Language Requirements for Building the System......................... 15

CHAPTER THREE: SYSTEM DESIGN & ARCHITECTURE ....................... 16

3.1 Chapter Overview ........................................................................ 16
3.2 System Architecture & Design ..................................................... 16
3.3 User Interface ........................................................................... 17
3.4 Communication Interface ............................................................ 17
3.5 Use Case Diagrams ................................................................... 18
  3.5.1 Use Case Diagram for the Doctor .......................................... 18
  3.5.2 Use Case Diagram for the Health Insurance Company .......... 19
  3.5.3 Use Case Diagram for the Patient .......................................... 20
  3.5.4 Use Case Diagram for the Hospital .................................... 21
3.6 E-R Diagram ............................................................................. 22

CHAPTER FOUR: IMPLEMENTATION .................................................. 23

4.1 Account Interface Design ............................................................ 23
  4.1.1 Home Page ........................................................................ 23
  4.1.2 Patient Login ...................................................................... 25
  4.1.3 Patient Profile .................................................................... 25
  4.1.4 Doctor Login ...................................................................... 28
  4.1.5 Doctor Profile ..................................................................... 29
4.1.6 Hospital Login ................................................................. 31
4.1.7 Hospital Profile .............................................................. 32
4.1.8 HIC Login ........................................................................ 35
4.1.9 HIC Profile ................................................................. 35
4.1.10 Emergency Profile .......................................................... 38
4.2 Technical Feasibility .............................................................. 41
4.3 Economic Feasibility .............................................................. 41
4.4 Operational Feasibility .............................................................. 42
CHAPTER FIVE: TESTING ............................................................. 43
5.1 Blackbox Testing ..................................................................... 43
5.2 Whitebox Testing ..................................................................... 43
5.3 String Testing ......................................................................... 43
5.4 Levels of Testing ...................................................................... 45
5.4.1 Unit Testing ......................................................................... 45
5.4.2 Integration Testing ............................................................... 45
5.4.3 System Testing ..................................................................... 46
5.4.4 Validation Testing ............................................................... 46
CHAPTER SIX: CONCLUSION ......................................................... 47
6.1 Summary .............................................................................. 47
6.2 Conclusion ................................................................................................................. 47

APPENDIX A: MISSION .................................................................................................. 49

APPENDIX B: COSTS ...................................................................................................... 51

APPENDIX C: FUNCTIONAL & NON-FUNCTIONAL REQUIREMENTS ........... 53

APPENDIX D: SOURCE CODE ....................................................................................... 60

REFERENCES ................................................................................................................ 86
LIST OF FIGURES

Figure 1: 3-Tier Architecture ................................................................. 17
Figure 2: Use Case Diagram for Doctor .................................................. 18
Figure 3: Use Case Diagram for HIC ....................................................... 19
Figure 4: Use Case Diagram for Patient .................................................. 20
Figure 5: Use Case Diagram for Hospital ............................................... 21
Figure 6: E-R Diagram ............................................................................ 22
Figure 7: Home Page ............................................................................. 24
Figure 8: Patient Login ........................................................................... 25
Figure 9: Patient Login ........................................................................... 26
Figure 10: Patient Personal Information ................................................. 26
Figure 11: Locate Nearest Doctor ........................................................... 27
Figure 12: Make Appointment ................................................................. 28
Figure 13: Doctor Login .......................................................................... 28
Figure 14: Doctor Profile ......................................................................... 29
Figure 15: General Appointment List ..................................................... 30
Figure 16: Generating Health Report ..................................................... 30
Figure 17: Patient Health Report ............................................................ 31
Figure 18: Hospital Login ....................................................................... 32
Figure 19: Hospital Profile ...................................................................... 32
Figure 20: List of Appointments ............................................................. 33
Figure 21: Patient Database ................................................................. 34
Figure 22: Patient Expense Report ................................................................. 34
Figure 23: HIC Login ...................................................................................... 35
Figure 24: HIC Profile .................................................................................... 36
Figure 25: Expense Report ............................................................................. 37
Figure 26: Emergency Profile ......................................................................... 38
Figure 27: Select Doctor ................................................................................ 39
Figure 28: Emergency Appointment Request ................................................ 39
Figure 29: Terms and Conditions for Emergency Appointment ...................... 40
Figure 30: Emergency Appointment List in Doctor Profile .............................. 40
Figure 31: Null Value Input in Patient Info ...................................................... 44
Figure 32: Gantt Chart for Project Progress ................................................... 52
CHAPTER ONE

INTRODUCTION

1.1 Chapter Overview

This chapter identifies and summarizes the problem that the project addresses, which includes the problem statement of the system being developed, solutions for the problem, and the scope involved in the problem being resolved.

1.2 Problem Question

The E-Healthcare industry has evolved over time, back when handwritten ledgers were maintained for the records up until the software to automate the procedure with the AI features showed up and it has come a long way since. With such developments this industry also has seen a lot of failures both in terms of life and revenue.

Data breaches in the healthcare field prove to be one of the most dangerous threats. If patient data falls into the hands of wrong people, it may lead to drastic situations where each and every person involved is vulnerable to the threat. According to an article written by Jessica Davis, around 15 million patient records were compromised in 503 breaches in the year 2018 alone (The 10 biggest Healthcare data breaches of 2019, so far, July 2019). These breaches mostly occur when hackers get access to the hospital servers and they get away with demographic patient health details and financial details.
Another issue that the industry has faced over the years is the cost of treatment. According to a study from the year 2017, the healthcare costs were estimated to be about $3.5 trillion making it one of the country’s largest industries [4].

Health insurance companies have played a vital role in this contribution. There are times when a patient is an emergency and is rushed to the nearest hospital not being bothered about the financial conditions of the patient which turns out to be a huge problem for the patient as his insurance plan is not supported by specific hospital or healthcare center. In a country like United States of America where insurance policies have proved helpful in most of the cases, it also has its demerits in terms of what kind of plan the patient selects. In most of the cases patients end up paying most of the hospital fees instead of insurance plan covering their charges. The reason is because they do not know if the doctor or the healthcare center complies with the patients’ health insurance plan.

Most of the time, if workers do not make direct moves during the yearly open enlistment time frame, they consequently proceed with the previous year’s insurance plan. In any case, regardless of whether that was the correct decision at that point, which usually ends up detrimental. [3] Blindly following the trend of choosing a cheaper health insurance plan may seem profitable, but when the patient is diagnosed at a hospital they may end up paying more than what was necessary if they had selected a better insurance plan.
As indicated by the Kaiser Family Foundation, 49 percent of the United States population gets medical coverage through a business provided plan. Furthermore, a few investigations show that a sizable portion of workers pick unnecessarily costly plans with low deductibles when high-deductible renditions are accessible. [3]

In one of the research papers "Adverse Selection and Inertia in Health Insurance Markets: When Nudging Hurts" (2013) by Benjamin R. Handel, a market analyst at the University of California, Berkeley, the most-mainstream choice was a well being plan with a low, $250 deductible. Professor Handel's examination demonstrated that workers would have set aside cash and gotten similar services, whatever their spending on clinical costs, in the event if they had moved to an arrangement with a $500 deductible.[3]

1.3 Problem Statement

Developing a system which helps in implementing Defense in Depth Strategy in Ehealthcare to prevent data breaches and provide better service to people using the system. Security is imparted via a two-factor authentication to the user.

To help patients find the nearest doctor who would accept the insurance plan of the patient, helping them to automate the report and receipt generation to the user.
To help the doctors to have peer to peer communication with the patient in an emergency and provide them with better service.

1.4 Problem Proposed Solution

This system aims at providing easily accessible healthcare services for the average person. This may include those who may not have the resources to make themselves aware of the options they have that would help reduce the cost of checkup with a doctor or treatment at a hospital and to monitor patient activities, their prescriptions, and payment dues. The system will also help patients in any emergencies find the nearest doctor to avoid any delays and take an appointment directly from them via Peer to Peer communication. The following are the proposed solutions that will help:

➢ **Find the Nearest Doctor:** System helps patients to find the nearest doctor who accepts the insurance type of the patient requesting an appointment. This is done by entering the name of the doctor or by entering the zip code of the place the patient’s residence is from so that it shows the doctors from that area. This feature can in future be developed by adding the GPS location of the doctor to make it easier for the patient.

➢ **Emergency Appointments:** The system makes sure that patients in an emergency are given a priority. The patient need not be a registered user to make an appointment in an emergency, instead go to a dedicated
button where they need to enter basic information of patient and request an appointment by accepting terms and conditions on the form.

- Automated Report Transfer: The Health reports generated after the patient diagnosis is sent to the hospital by the doctor. Once the hospital receives the health report, it generates an expense report based on that and sends it to the HIC, later the expense report is submitted to the patient.

- Selecting an appropriate insurance plan: The Patient information form provides fields to enter the health insurance plan details, so that the system can be updated when the patient’s insurance plan is going to expire, also what doctor would accept that insurance plan to avoid unnecessary expenses.

1.5 Project Scope

This project is an open source website for people willing to have their data managed by electronic healthcare. The system here makes it possible for the patients to upload their information on the website by entering their credentials and confirming it through two factor authentications. The patient can make an appointment with the nearest doctor and in case of an emergency the patient will be given priority based on their medical emergency. The patients will end up paying fairly for the diagnosis as the insurance plan will be matched with a doctor who accepts it. This system is being designed as a front end developed in React, which means its modularity will allow for future iterations to include more robust
features such as a GPS system to help patients locate doctors in an emergency. Overall, this is a one stop solution for all the Ehealthcare requirements, being first of its kind.
CHAPTER TWO

REQUIREMENTS AND SPECIFICATIONS

2.1 Chapter Overview

This chapter is first focused on the four types of end users: Patients, Doctors, Hospitals and Health Insurance Companies. Also, a list of the system requirements and specifications for the system which include the functional and non-functional requirements that the system needs are provided. A Complete description of the requirements and specifications for the Docs_on_Blocks system development which is described in detail in this chapter.

2.2 End Users

There are three types of system users:

a. Patient: They will be responsible to create their own account in the system via the two-factor authentication, and also keep their personal information updated to take appointments with the nearest doctor who accepts their insurance plan and interact with hospitals and insurance companies in the future.

b. Doctor: They are the actors that also needs to be signed in the system to utilize its resources and interact with the patients and confirm or delete the appointments and send patient health reports to the hospital.
c. Hospital: They are users that keeps track of the patient and doctor records and manage the everyday appointments and the send the expense reports to insurance company and health reports to patient.

d. Health Insurance Company: They are the actors that are signed in the system to analyze the medical expenses of the patient and receive feedback on their service.

2.3 System Requirement Specification

A list of all functional and non-functional requirements are listed and referenced in the appendix. This contains a table with a structured collection of information that embodies the requirements of the Docs_on_Blocks System.

2.4 Actors and Use Cases

<table>
<thead>
<tr>
<th>Actors</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Patient</strong></td>
<td>A user who takes an appointment with a nearby doctor who accepts their insurance plan.</td>
</tr>
<tr>
<td><strong>Doctor</strong></td>
<td>A user who confirms or deletes the regular and emergency appointments.</td>
</tr>
<tr>
<td><strong>Hospital</strong></td>
<td>A user who keeps record of all the patient and doctor appointments and their data and makes sure that reports reach right actor.</td>
</tr>
</tbody>
</table>
Health Insurance Company | A user who makes sure that finances of the patient are managed based on the kind of treatment that he or she receives at the hospital by the doctor.

Use cases

Table 2: Use cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register User</td>
<td>For all four actors to create an account.</td>
</tr>
<tr>
<td>Login User</td>
<td>The user logs in the system via two factor authentication.</td>
</tr>
<tr>
<td>View Appointment</td>
<td>Displays patient appointment requests to the doctor.</td>
</tr>
<tr>
<td>Manage Appointment</td>
<td>Doctor and Hospital approves or deletes the appointment request.</td>
</tr>
<tr>
<td>View Hospital Database</td>
<td>Displays the hospital database which includes the patient and doctor info.</td>
</tr>
<tr>
<td>View User Profile</td>
<td>View patient, doctor, or hospital profile by respective user.</td>
</tr>
<tr>
<td>Update User Profile</td>
<td>The profiles of the patient, doctor, and hospital users are updated with their information</td>
</tr>
<tr>
<td>Send Patient Health Report</td>
<td>Doctor generates a health report of the patient and sends it over to hospital which later can be viewed by patient through their account.</td>
</tr>
</tbody>
</table>
Send Patient Expense Report | The expense report generated by the hospital based on the diagnosis in the health report is sent to the respective insurance company and later to the patient profile.

View List of Doctors | To make an appointment, patient selects the nearest doctor by entering their zip code or name of doctor.

Give Feedback | The patient can give feedback to the HIC to let them know about their quality of service.

Send Mail | The doctor, patient and hospital receive mail sent to them by the Health Insurance Company.

View Patient Report | The patient can view their health and expense report sent to them by the hospital and HIC respectively.

2.5 Modules

A design approach that subdivides a system into smaller parts called modules, makes it easier for the system to be used by any stakeholder. This website has five significant modules with its sub-modules as follows:

Patient Login

- Patient Details:
  - The system allows patient to add personal information into the system to be used for further use. The information also includes the health insurance details of the patient and medical history.
Doctor's Nearby:
- Patient can view the list of doctors near them and select one based upon the type of insurance they accept.

Appointment
- Once the doctor is selected the non-emergency patients can request an appointment with the doctor.

Health Report
- Once the hospital uploads the patients' health report, it will be visible in this module.

Expense Report
- This will appear once the health insurance company sends the expense report to patient after the diagnosis report reaches them.

Doctor Login

Emergency Patients:
- Patients in an emergency can make an appointment without entering their personal information and history, but only in
case of emergency by accepting the terms and conditions of the website. Doctors can view those appointments here.

- **Appointments:**
  - Non-emergency patients can make an appointment with the doctor, and those appointments are viewed by the doctor here.

- **Health Report:**
  - Once the patients are diagnosed by the doctor, the doctor prepares a health report of the patient’s diagnosis and sends it to the hospital to generate an expense report based on the diagnosis.

**Hospital Login**

- **Doctor’s Appointment:**
  - The hospital has the right to accept or cancel an appointment that a patient has made with their doctors.

- **Patient Details Record:**
  - The hospital maintains a database of the patients and it is available in this section.
o Patient’s Expense Report:

- On receiving the patient health report the hospital generates the expense report for the patient and sends it to the insurance company.

Health Insurance Company Login

o Registered Patient’s Record:

- The patients who have a health insurance plan bought from the company that has signed into the system will be shown here in the database.

o Complaints and Mails:

- This section is for the feedback from the people who have taken services from the insurance company that they have enrolled in.

o Patient’s Expense Report:

- The expense report generated by the hospital based on the diagnosis done by the doctor is received by the insurance company and accordingly takes care of the patients’ finances based on their insurance plan.

Emergency Login

o Doctors Nearby:
- This is similar to that of the regular patients login page, where the patients can find a nearest doctor based on their insurance plan type.

  o Emergency Appointment:

- This is where the patients in an emergency can fill out a very few lines of self-information to make an appointment with a doctor.

2.6 Resource Requirements

The Project was built using Visual Studio 2019. The codes were written based on React Native JS, which is an open source JavaScript library to create an effective UI. Due to the libraries available in this environment it is easier to manage the databases and does not need a database system to manage it. GitHub is another open source platform where it makes it possible to host and collaborate the codes, it is a repository that supports HTML, CSS, JAVA, documents, images, etc. Netlify is a cloud service provider which has been used here to manage the data.

2.6.1 Hardware Requirements:

- 500MB-Ram
- 1GB Hard Disk
- Internet Connection
• Intel i3, AMD A6 or equivalent processor

2.6.2 Software Requirement:

• Operating System: Windows 7 or higher
• Internet Explorer, Mozilla Firefox, Google Chrome, or Safari

2.6.3 Language Requirements for Building the System

• Programing Language: JavaScript
• Web-Technology: React Native
• Front- End: Visual Studio 2019
CHAPTER THREE

SYSTEM DESIGN & ARCHITECTURE

3.1 Chapter Overview

This chapter discusses about the design and analysis phase of the system, which includes use case diagrams and ER diagrams.

3.2 System Architecture & Design

The Docs_on_Blocks system is based on the 3-tier architecture, it is designed as a client-server-based system. The first layer of the 3-tier architecture is the Application/ Presentation layer, this is where the front end comes into play, this is what is visible to the person operating on one end of the system. The second layer is the Conceptual/Business logic layer where the interfaces come into play, and the last layer being the Internal/Data access layer is where the databases can be accessed through. Actions like return, update, or manipulate can be performed on the data available. These can be depicted as in the figure below:
These three layers are helpful in flexibility, scalability, and security. Each layer can be secured using different approaches, like the 2nd level in this system is secured by the 2-factor authentication.

3.3 User Interface

The Docs_on_Blocks system categorizes users into four major components: patient, doctor, hospital employees, and health insurance company employees. All the components or users interact with the system through the login interface page.

3.4 Communication Interface

The Docs_on_Blocks system uses React JS framework and the operating system to manage communication between the users and the server.
3.5 Use Case Diagrams

Use case diagrams are a representation of interactions of the user with the system, which shows the relationship between the user and different use cases of a system in which the user/actor is involved. There are four primary end users in the Docs_on_Blocks system: Patients, Doctor, Hospital and HIC’s. This section will discuss separate use case diagrams for the four types of users in the system.

3.5.1 Use Case Diagram for the Doctor

A detailed use case diagram for the Doctor is shown in figure 2.

Figure 2: Use Case Diagram for Doctor
3.5.2 Use Case Diagram for the Health Insurance Company

A detailed use case diagram for the HIC is shown in figure 3.

Figure 3: Use Case Diagram for HIC
3.5.3 Use Case Diagram for Patient

A detailed use case diagram for the Patient is shown in figure 4.

Figure 4: Use Case Diagram for Patient
3.5.4 Use Case Diagram for the Hospital

A detailed use case diagram for the Hospital is shown in figure 5.

Figure 5: Use Case Diagram for Hospital
3.6 E-R Diagram

The entity relationship diagram shows the relationships of the entities and how each of them relates in the Docs_on_Blocks System.

Figure 6: E-R Diagram
4.1 Account Interface Design

The Docs_on_Blocks System is a fully web-based system created with React Native JS. React Native gets together the best parts of native development with React, and one of the finest JS libraries for building effective UI [2]. React can create platform-specific versions of components so that a single code can be shared across different platforms. One team or a person can maintain 2 platforms and share a common technology [2]. React Native supports popular browsers such as Safari, Firefox, Google Chrome, Internet Explorer etc. All user interfaces can be reached dynamically on the server side and can fully update the client pages in real time, giving the user an instant response.

4.1.1 Home Page

This is the homepage of the prototype of the system. The actors involved in the system are shown here as the buttons starting from Patient, Doctor, Hospital, Insurance Company and Emergency.
Each of the users is related to the other based on the kind of interaction between them selected by the respective user of the system. To login to each of the buttons, the user must be signed in the system and needs to go through a two-factor authentication. As each transaction or activity done in the system generates a hash which is linked to the previous hash, any change in any of the hash leads to complete change in the chain of blocks thus keeping the data safe.

As the healthcare industry faces a huge risk of the data piracy or loss, there needs to be an increased security provided to the systems dealing with such volatile data which in the current system is being implemented via two-factor authentication, where a key is sent to the email ID of the user which helps them login the system. When the patient uses the system, they login to the system by creating their own account using their email ID and password as credentials.
4.1.2 Patient Login

This is the sign in page where the patients fill in their credentials, in the system. Here we have requested the patient to enter their name, email, address, and password.

Figure 8: Patient Login

As the two-factor authentication is involved the patient will be requested to enter the key sent to their email. Once the Patient enters the key to login, they can view the following.

4.1.3 Patient Profile

The front end here shows the interactive buttons starting from Patient details, which on clicking requests the complete information of the patient.
Figure 9: Patient Login

![Patient Profile Diagram]

The form that appears to the patient to fill up the data is as in figure 10. This includes medical history of the patient, all the medications taken prior and health insurance details.

Figure 10: Patient Personal Information

![Patient Personal Information Form]
Once the patient has entered all the details requested, the system then updates the records and is stored in the system. When the patient tries to get access to their data again on visiting the site it is available along with all the new appointments and insurance details all at one place. The patient on filling the application then can select a doctor near their location for an appointment, which they can find by entering their zip code or the name of the doctor.

Figure 11: Locate Nearest Doctor

Once the patient clicks on the doctor the patient is willing to take, an appointment with the prompt will be shown on the top of the page that the doctor is selected, and the patient has to enter their details to make an appointment specifying the issue they are facing and type of medications they have taken. As this is a common form of appointment, no special conditions are to be met as in case of patients with an emergency which will be explained later. Once the appointment is requested, it is up to the doctor to accept the appointment or to deny it based on the kind of diagnosis and the problem mentioned. The list of
doctors in this static database is taken from the website
www.ehealthmedicare.com [1].

Figure 12: Make Appointment

![Appointment Form]

4.1.4 Doctor Login

The Doctors who are a part of this system also must sign up with their credentials to provide the service to patients being in contact with the hospital.

Figure 13: Doctor Login

![Doctor Login Form]
4.1.5 Doctor Profile

Once the doctor has signed into the system, they are provided by three options, the first two options deal with the appointment confirmations of patients in emergency and regular appointments, respectively.

Figure 14: Doctor Profile

When the doctor clicks on the Appointment button, they will be presented with a list of all the appointments that needs to be accepted or rejected based on the problem the patient is facing.

The Emergency patients button shows the list of appointment requests made by patients in emergency only. This is shown in figure 15.
Once the doctor confirms the appointment, the patient can visit the doctor on a specific day and get diagnosed. The diagnosis of each patient is generated as a health report and sent to the hospital in which the doctor works and is registered. The hospital then generates an expense report based on the health report sent. Figure 16 shows the format of the health report generated.

Figure 16: Generating Health Report
The health report when sent to the hospital is analyzed and returned to the patient, which can be viewed from the patient profile by clicking on the Health Report button. The report will appear as such in figure 17.

Figure 17: Patient Health Report

4.1.6 Hospital Login

For hospitals to perform the tasks in the system, they would also have to be a part of Docs_on_Block. Each hospital must have to enter their credentials and login via 2 factor authentication done through email. Figure 18 shows the hospital login page.
4.1.7 Hospital Profile

Once the hospital logs into the system, it is provided with three buttons namely Doctor’s Appointments, Patient Details Record, and Patient’s Expense Report this is shown in figure 19.
Hospitals can manage all the Doctor’s appointments either by accepting them or rejecting based on the criteria of a patient’s health insurance plan and availability which is shown in figure 20.

Figure 20: List of Appointments

The second button is the patient detail record, which is a database of all the patients that have visited the hospital. This form displays the details of the patient, in order to view the patient history the system requests the employee to enter their credentials to provide an extra layer of security to the patient records so that the hierarchy of the employee is followed. For example, a clinical staff does not need to know about the surgery details of a patient like that of his/her surgeon. This can be seen in figure 21. Once the Employee of the hospital submits there credentials the patient details are revealed according to the level of data visibility.
The third button in the hospital profile is to create an expense report of the patient, which is created by the hospital by viewing the health report of the patient generated by the doctor who diagnosed them. The expense report consists of patient details and the cost of the treatment, which is submitted to the HIC. Figure 22 shows the form for the expense report.
4.1.8 **HIC Login**

To find a doctor who accepts the insurance plan that the patient has which may reduce a huge amount of over payment for the diagnosis done is a challenge. The Docs_on_Blocks system focuses on providing a permanent solution for this problem by locating the nearest doctor based on the zip code entered. The HICs have to be a part of the system as well to provide the service. This is done by entering the credentials and going through two factor authentication. The login page is as shown in figure 23.

Figure 23: HIC Login

4.1.9 **HIC Profile**

The profile of HIC consists of three interactive buttons namely: Registered Patient Records, Complaints and Mails, and Patient’s Expense Report. Figure 24 shows the frontend for the HIC profile.
The HIC generates an expense report for the patient once it receives the health report from the hospital. After the expense report is generated based on the kind of insurance plan of the patient, if the extra amount has to be covered by the patient themselves then the report will be forwarded to the patient’s profile so that the balance amount can be paid to the company. Figure 25 shows the expense report sent to the patient.
The first button of the HIC profile has the list of people who have bought an insurance plan from that company and are enrolled in the system. The second button deals with the feedback of the people who have received services from the HIC. Feedback is necessary for any business to grow, as the improvements needed can be figured out through the client responses. Thus, here the HIC can improve its plans by the help of patient feedback.

Example: An international student who needs to comply with health industry standards to maintain their visa status in the country must buy a specific insurance plan. But in case of an emergency the student goes to a nearby hospital complaining about a pain in their tooth. The doctor performs the diagnosis and lets the patient go after prescribing a few pain killers. After a month, the student receives an envelope with an expense report of the hospital to pay $300 which is not covered by the insurance plan. Thus, if the student knew
earlier that the hospital did not accept this insurance plan, they would have made sure to look for the one who does. The Docs_on_Blocks system helps them eradicate this disadvantage by figuring out which doctor near them accepts their plan, so that they don’t end up paying out of pocket.

4.1.10 Emergency Profile

The emergency profile can be accessed by a nonmember in case of an emergency. But the clauses or terms must be agreed before they make an appointment request. The profile shows two buttons namely, Doctors Nearby and Emergency Appointment as shown in fig 26.

Figure 26: Emergency Profile

The working of this module is similar to the appointment request in the Patient profile, but the layout of the form that needs to be filled out to take an emergency appointment is a lot less compared to the prior. Once the doctor is
selected as shown in figure 27, the appointment request has to be made as shown in figure 28 and 29.

Figure 27: Select Doctor

Figure 28: Emergency Appointment Request
Once the Emergency appointment has been made this is visible to the doctors in their profile. The button is positioned first to give it a priority as compared to regular appointments.

Figure 30 shows the format of Emergency Appointments as seen by the doctors in their profile. The doctor can either agree or cancel the appointment requested made by the patient based on the type of emergency.
4.2 Technical Feasibility

This step basically deals with how we decide to deliver a product to a client or customer. This includes the technology involved in it and labor and all the other charges involved to have a finished product into the market. [6] This step confirms whether the product can be used in real time based on the following conditions:

- Hardware and software technology to develop this system exists (MS Visual Studio 2019, React JS, Web browser).

- This system can be restructured in the future to make it more robust (GPS system if included to locate the doctors nearby would make it more helpful for the people using it).

- The Docs_on_Blocks system is a static website for now, which can yield the results fast without having to connect to a server every time to fetch the results.

4.3 Economic Feasibility

This project is economically feasible because there is no investment involved to generate the product as all the resources used are open source products like GitHub, and code snippets which are fetched from open sources as well. The time required to build this is about 4 months, but to include more features it is possible to do it in couple months based on the requirement. Even the clients using this software need not invest to get the services for now. As the Docs_on_Blocks system is the first of its kind it has more room for improvement
in the near future. The users using it will be hugely benefitted economically, without even investing anything. Thus, all these reasons make it more favorable as the system’s economic benefits are exceeding the costs [7].

4.4 Operational Feasibility

This step focuses on high output with low input. The input may be in the form of manpower, money, and time. In the case of Docs_on_Blocks system the work force was less, and if there was more man force the project would have been completed a lot quicker, but it would compromise its resources. Even though the project took time to be built, it being first of its kind, it has a good future to it making it operationally feasible. React JS being increasingly popular these past few years has a brighter future in upcoming time. With its robust features the system can go on for a longer period than expected. [7] With the system working on any browser as a front-end tool only, it has minimal or no chance of failures, which makes it more acceptable. [7]
CHAPTER FIVE

TESTING

5.1 Blackbox Testing

This is basically the behavioral testing done on the product or system to check the efficiency of design, internal structure, and implementation. The system is checked for missing functions, interface errors and the data structures. [8] In case of Docs_on_Blocks, the system was tested in the development and implementation phase for this testing method.

5.2 Whitebox Testing

This software testing method is performed by the tester himself; this is also known as open box testing or transparent testing. Different kinds of inputs are provided to get an appropriate output. Thus, Docs_on_Blocks system was tested with this method to determine the outputs generated which may be essential in this case.

5.3 String Testing

The product has been tested with different kinds of inputs for the patient’s name, and addresses, doctor’s reports, and for the expense reports by health insurance companies, etc.
All these were tested using the following datatypes: string, null data (which prompts if not entered for specific entries), alphanumeric characters, and numbers.

This testing needs to be done as we do not know what kind of inputs the user might enter in order to get services. Thus, this testing is crucial to know what outputs are generated on inserting wrong values and the right values. Example: a person cannot enter their name in the section of phone number as it strictly allows only numbers. Therefore, in the Docs_on_Blocks system we can see that the prompts turn red on the wrong input or null values in each form to be filled by different users. Figure 31 shows the prompts after inputting null values.

Figure 31: Null Value Input in Patient Info
5.4 Levels of Testing

Four levels of software testing involves:

a. Unit testing
b. Integration testing
c. System testing
d. Acceptance testing

5.4.1 Unit Testing

Unit Testing is where each individual components of a software are tested. This individual component may either be a function or a procedure. Developers take care of this testing phase. This determines if there is any issues in the codes by the developer and corrected then and there. [9] In the system developed the unit testing was performed during the development phase itself.

5.4.2 Integration Testing

Testing the interface between two software units or modules is known as integration testing and focuses on determining the correctness of the interface. The ideal behind the integration testing is to find the issues involved in the interaction between integrated units. This testing takes place only when the unit testing is completed. [9] Docs_on_Blocks system has gone through this phase of testing as well.
5.4.3 System Testing

System testing makes sure whether specified requirements of a product are met or not. When various components are integrated into one place and then the whole thing goes under integration testing this becomes system integration system. Docs_on_Blocks system was tested through system testing.

5.4.4 Validation Testing

Validation testing or acceptance testing is a method to ensure whether the software meets the customer requirements or not. Testers and stakeholders use this testing method. [9] Acceptance testing is only a functional testing unlike system testing which is both functional and nonfunctional. Docs_on_Blocks system has been satisfactorily tested using validation testing.
CHAPTER SIX

CONCLUSION

This chapter describes achievements from the project and the conclusions derived from it.

6.1 Summary

The project was a combined effort of all the courses taken under MSIST program for the past 2 years. The project idea was built when one of my friend’s Health Insurance plan did not cover the consultation charges for a tooth ache, it became clear that a system needs to be designed to solve this issue. The Docs_on_Blocks system is first of its kind; thus, it is a prototype with some robust features that can be developed in the future to improvise its functionality.

6.2 Conclusion

Docs_on_Blocks system has been developed keeping in mind the users who are from different aspects of the healthcare field, the focus being mainly to benefit a patient. This system being a website can be accessed from any browser, also it can be used from a mobile device as it is developed using React JS.

This user-friendly interface consists of 5 modules namely: Patient, Doctor, Hospital, HIC, and Emergency patients. Each of the user, except an Emergency patient, must be a part of the system by logging in with their credentials to carry
out any operations. The tools used in this project are Visual Studio 2019, Netlify cloud services, GitHub, etc. The user does not need to have any of these tools as it is only a front-end prototype the website will be enough to interact with the users in the system and get the expected output.

Recommendations for the project would be to add the GPS locating system for the doctors to make it easier for the patients to get to the nearest location. Also the whole system can be built on Blockchain technology, which would help the authentication process, by eradicating the two factor authentication as the blocks generate the hash code and this will end the necessity of any manager of the system and make it completely automated and secured. As the healthcare data is volatile and prone to data breaches in the past years, the Blockchain will help end this trend.
APPENDIX A

MISSION
MISSION

The mission of the Docs_on_Blocks System is to provide an automated Ehealthcare environment to the end users namely: Patient, Doctor, Hospital and HICs which would help in providing a solution to current shortcomings in the industry.

Vision

Improving efficiency of the current healthcare trends, reducing time of appointment confirmations, budget management of patients with proper insurance plans, and securing the data from breaches or data leaks.

Values

- **Initiative**
  - Updating system with latest technology and features on a regular basis.

- **Communication**
  - Keeping in touch with the stakeholders for constant updates on the improvements needed for the system.

- **Service**
  - Docs_on_Blocks system is expected to provide excellent customer service to the stakeholders/clients, with regular check on feedbacks, periodic updates.

- **Direction**
  - The system strives to maintain the ethics and continue with providing good service along with the data security.

- **Quality & Safety**
  - The system is expected to provide best of the quality product being first of its kind

50
APPENDIX B

COSTS
Figure 32: Gantt Chart for Project Progress

The Docs_on_Blocks system will be commercialized online free of cost. Being first of its kind, it has no competitors yet in the market. Based on the response it may be monetized in future. The Docs_on_Blocks system will be provided as an open source tool for everyone in the healthcare industry. The system will generate revenue through partnerships with healthcare departments, doctors, hospitals, and health insurance companies.
APPENDIX C

FUNCTIONAL & NON-FUNCTIONAL REQUIREMENTS
# TABLE 3: Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Details</th>
<th>Type</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>The Docs_on_Blocks system shall create new user accounts.</td>
<td>• Registration</td>
<td>Must-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>The Docs_on_Blocks system shall allow login to users to their prospective account.</td>
<td>• Authentication</td>
<td>Must-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>The Docs_on_Blocks system shall authenticate credentials through 2-factor authentication.</td>
<td>• Authentication</td>
<td>Must-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>The Docs_on_Blocks system shall monitor the users log.</td>
<td>• Product</td>
<td>Must-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>The Docs_on_Blocks system shall set a minimum qualification requirement for doctor.</td>
<td>• User Interface</td>
<td>Should-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td>The Docs_on_Blocks system shall set a Emergency Patients form for nonregistered users.</td>
<td>• User Interface</td>
<td>Should-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td></td>
</tr>
</tbody>
</table>
| R7  | The Docs_on_Blocks system shall search for nearest doctor by inputting their name or zip code. | • Product  
• Functional | Should-Have |
|-----|-----------------------------------------------------------------------------------------------|-------------------------------|-------------|
| R8  | The Docs_on_Blocks system shall display patient information on patient profile.               | • Product  
• Functional | Should-Have |
| R10 | The Docs_on_Blocks system shall allow patients to provide any relevant health history, including earlier diagnosis, medications, etc. | • Product  
• Functional | Must-Have |
| R11 | The Docs_on_Blocks system shall require patient appointment request for doctor access that patient records. | • User Interface  
• Functional | Must-Have |
| R12 | The Docs_on_Blocks system shall allow patients to include their health insurance plan details in their info. | • User Interface  
• Functional | Must-Have |
| R13 | The Docs_on_Blocks system shall allow the patient to view their health reports sent to them by hospital. | • User Interface  
• Functional | Must-Have |
| R14 | The Docs_on_Blocks system shall notify the patient on selecting a Doctor.                      | • Order  
• Functional | Must-Have |
<table>
<thead>
<tr>
<th><strong>R15</strong></th>
<th>The Docs_on_Blocks system shall allow Hospital to send a copy of the patient’s health report to patient.</th>
<th>• Order</th>
<th>Must-Have</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td>Have</td>
</tr>
<tr>
<td><strong>R16</strong></td>
<td>The Docs_on_Blocks system shall allow the Hospital to send a copy of expense report to HIC.</td>
<td>• Order</td>
<td>Must-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td>Have</td>
</tr>
<tr>
<td><strong>R17</strong></td>
<td>The Docs_on_Blocks system enables allow the Doctor to cancel an appointment.</td>
<td>• Order</td>
<td>Must-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Non-Functional</td>
<td>Have</td>
</tr>
<tr>
<td><strong>R18</strong></td>
<td>The Docs_on_Blocks system shall notify the patient if the appointment is canceled</td>
<td>• Order</td>
<td>Must-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Non-Functional</td>
<td>Have</td>
</tr>
<tr>
<td><strong>R19</strong></td>
<td>The Docs_on_Blocks system shall allow the patients in emergency to make an appointment.</td>
<td>• Order</td>
<td>Must-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Functional</td>
<td>Have</td>
</tr>
<tr>
<td><strong>R20</strong></td>
<td>The Docs_on_Blocks system shall give preference to patients in emergency.</td>
<td>• Performance</td>
<td>Should-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Non-Functional</td>
<td>Have</td>
</tr>
<tr>
<td><strong>R21</strong></td>
<td>The Docs_on_Blocks system shall log in the Doctor and the Patient within 5 seconds.</td>
<td>• Performance</td>
<td>Should-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Non-Functional</td>
<td>Have</td>
</tr>
<tr>
<td><strong>R22</strong></td>
<td>The Docs_on_Blocks system shall login Hospital and Insurance Company result in less than 6 seconds.</td>
<td>• Performance</td>
<td>Should-Have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Non-Functional</td>
<td>Have</td>
</tr>
</tbody>
</table>
| R23 | The Docs_on_Blocks system shall show the profile of Patient to respective Hospital employee in less than 10 seconds. | • Performance  
• Non-Functional | Should-Have |
|-----|---------------------------------------------------------------------------------------------------------------|---------------------------------|------------|
| R24 | The Docs_on_Blocks system shall show the profile of Hospital and Insurance company in less than 15 seconds.  | • Performance  
• Non-Functional | Must-Have |
| R25 | The Docs_on_Blocks system shall store all transaction data.                                                | • Availability  
• Non-Functional | Must-Have |
| R26 | The Docs_on_Blocks system shall be accessible 24 hours a day 360 days.                                       | • Availability  
• Non-Functional | Must-Have |
| R27 | The Docs_on_Blocks system shall allow 5,000 transactions per day.                                            | • Capacity  
• Non-Functional | Must-Have |
| R28 | The Docs_on_Blocks system shall allow 10 transactions per second at peak time.                               | • Capacity  
• Functional | Should-Have |
| R29 | The Docs_on_Blocks system shall use a browser at its user interface.                                        | • Compliance To-  
Standards  
Non-Functional | Must-Have |
<table>
<thead>
<tr>
<th>R30</th>
<th>The Docs_on_Blocks system shall support the last 3 versions of any web browsers.</th>
<th>• Compliance To- Standards • Functional</th>
<th>Must-Have</th>
</tr>
</thead>
<tbody>
<tr>
<td>R31</td>
<td>The Docs_on_Blocks system shall organize the list of appointments by date.</td>
<td>• Product • Functional</td>
<td>Must-Have</td>
</tr>
<tr>
<td>R32</td>
<td>The Docs_on_Blocks system shall allow users to recover the password by clicking &quot;forgot password.&quot;</td>
<td>• Authentication • Functional</td>
<td>Must-Have</td>
</tr>
<tr>
<td>R33</td>
<td>The Docs_on_Blocks system shall operate in various mobile devices with internet connection.</td>
<td>• Performance • Non-Functional</td>
<td>Should-Have</td>
</tr>
<tr>
<td>R34</td>
<td>The Docs_on_Blocks system shall accept all kinds of patients.</td>
<td>• Compliance To- Standards Functional</td>
<td>Must-Have</td>
</tr>
<tr>
<td>R35</td>
<td>The Docs_on_Blocks system shall validate the expenses generated by HIC.</td>
<td>• Payment • Functional</td>
<td>Must-Have</td>
</tr>
<tr>
<td>R36</td>
<td>The Docs_on_Blocks system shall check the patient’s personal information before processing the appointment confirmation.</td>
<td>• Compliance To- Standards Functional</td>
<td>Must-Have</td>
</tr>
<tr>
<td>R37</td>
<td>The Docs_on_Blocks system shall allow HIC to view registered patients record.</td>
<td>• Compliance To-Standards Functional</td>
<td>Must-Have</td>
</tr>
<tr>
<td>-----</td>
<td>------------------------------------------------------------------------</td>
<td>---------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>R38</td>
<td>The Docs_on_Blocks system shall allow patients to accept the terms &amp; conditions for emergency appointments.</td>
<td>• Compliance To-Standards Functional</td>
<td>Must-Have</td>
</tr>
<tr>
<td>R39</td>
<td>The Docs_on_Blocks system shall secure the browsing.</td>
<td>• Security Non-Functional</td>
<td>Must-Have</td>
</tr>
<tr>
<td>R40</td>
<td>The Docs_on_Blocks system shall provide two-factor verification.</td>
<td>• Security Non-Functional</td>
<td>Must-Have</td>
</tr>
</tbody>
</table>
APPENDIX D

SOURCE CODE
import React from "react";
import { Link, Switch, Route } from "react-router-dom";

import Tile from "./shared/components(Tile).Tile";
import basestyles from "./shared/styles/ba
sestyles.module.css";
import homesty les from "./shared/styles/home.module.css";
import PatientRoutes from "./patient/pages/PatientRegisterRoutes/PatientRegister";
import DoctorRoutes from "./doctor/pages/DoctorRegisterRoutes/DoctorRegister";
import InsuranceRegister from "./insurance/pages/InsuranceRegisterRoutes/InsuranceRegister";
import HospitalRegister from "./hospital/pages/HospitalRegisterRoutes/HospitalRegister";
import EmergencyRoute from "./emergency/pages/EmergencyRoutes/EmergencyRoutes";

function Home() {
  return (
    <div className={homestyles.containerHome}>
      <div className={basestyles.container}>
        <h1>
          <p>docs</p>
          <p>on</p>
          <p>blocks</p>
        </h1>
      </div>
      <div className={basestyles.container}>
        <Link to="/patient">
          <Tile name="Patient" type="round" icon="user" />
        </Link>
        <Link to="/doctor">
          <Tile name="Doctor" type="round" icon="user-md" />
        </Link>
        <Link to="/hospital">
          <Tile name="Hospital" type="round" icon="user-md" />
        </Link>
      </div>
    </div>
  )
}
PATIENT PROFILE

import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";
import { RingLoader } from "react-spinners";
import Backicon from "./shared/components/Backicon/Backicon";
import Tile from "./shared/components/Tile/Tile";
import styles from "./patientprofile.module.css";
import baseStyles from "./shared/styles/basestyles.module.css";

function PatientProfile() {
    const [pageLoading, setPageLoading] = useState(true);

    useEffect(() => {
        // Your code here
    })
}
setTimeout(() => setPageLoading(false), 3000);
}, []);

return (
<div className={styles.phome}>
<div className={baseStyles.container}>
{pageLoading ? ( 
<div className={styles.spinnerContainer}>
<RingLoader size={100} color="white" /></div>
</div>
) : (
</div>
</div>
</div>
</div>
</div>
</div>
</div>

<Backicon onClick={() => this.props.history.goBack()} />
<h1 className={styles.title}>Patient Profile</h1>
<div className={styles.first}>
<Link to="/patient/patient-details">
<Tile name="Patient Details"
style={{
height: "200px",
width: "250px",
fontSize: "2.5rem",
backgroundColor: "rgba(255, 255, 255, 0.619)",
}} />
</Link>
</div>
</div>
</div>
</div>

<Link to="/patient/doctors-nearby">
<Tile name="Doctor's Nearby"
style={{
height: "200px",
width: "250px",
fontSize: "2.5rem",
backgroundColor: "rgba(255, 255, 255, 0.619)",
}} />
</Link>
</div>
</div>
</div>
</div>
</div>

<Link to="/patient/appointments">
<Tile name="Appointment"
style={{
height: "200px",
width: "250px",
fontSize: "2.5rem",
backgroundColor: "rgba(255, 255, 255, 0.619)",
}} />
</Link>
</div>
</div>
</div>
</div>
</div>
height: "200px",
width: "250px",
font-size: "2.5rem",
background-color: "rgba(255, 255, 255, 0.619)",
}
 />
</Link>
</div>
<Link to="/patient/health-reports">
<Tile
name="Health Report"
style={{
height: "200px",
width: "250px",
font-size: "2.5rem",
background-color: "rgba(255, 255, 255, 0.619)",
}}
 />
</Link>
<Link to="/patient/expense-report">
<Tile
name="Expense Report"
style={{
height: "200px",
width: "250px",
font-size: "2.5rem",
background-color: "rgba(255, 255, 255, 0.619)",
}}
 />
</Link>
</div>
</div>
});
export default PatientProfile;
import React, { Component } from "react";
import styles from "./appointments.module.css";
import basestyles from "../../shared/styles/basestyles.module.css";
import Header from "../../../shared/components/Header/Header";
import Backicon from "../../../shared/components/Backicon/Backicon";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

class Appointments extends Component {
  state = {
    patientAppointmentOverview: "",
  };

  componentDidMount() {
    let appointments = JSON.parse(localStorage.getItem("patient_appointment"));
    this.setState({
      patientAppointmentOverview: appointments ? appointments.data : [],
    });
  }

  handleRemove = (id) => {
    let filteredAppointments = this.state.patientAppointmentOverview.filter(
      appointment => appointment.id !== id
    );
    console.log(this.state.patientAppointmentOverview);
    this.setState({
      patientAppointmentOverview: filteredAppointments,
    }, () => {
      localStorage.setItem("patient_appointment",
      JSON.stringify({
        data: this.state.patientAppointmentOverview
      }));
    });
  }

  scheduleAppointment = () => {
    toast.info("Appointment scheduled...!", {
      autoClose: 2000,
    });
  };
}
render() {
  return (
    <>
    <div className={styles.backgroundappoint}>
      <Header name="All Appointments" />
      <Backicon onClick={() => this.props.history.goBack()} style={{
          background: "linear-gradient(60deg, rgba(2, 12, 29, 0.644) 0%, rgba(67, 184, 192, 0.548) 98%")",
          color: "white",
        }}
    />
    <div className={basestyles.inputWidth}>
      <div className={styles.allapt}>
        <table>
          <thead>
            <tr>
              <th>ID</th>
              <th>Name</th>
              <th>Problem</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            {this.state.patientAppointmentOverview &&
              this.state.patientAppointmentOverview.map((appointment) => ( 
                <tr key={appointment.id}>
                  <td>{appointment.id}</td>
                  <td>{appointment.Name}</td>
                  <td>{appointment.TypeOfProblemHeSheIsFacing}</td>
                  <td className={styles.acceptTrash}>
                    <i className="far fa-check-square" onClick={() => this.scheduleAppointment()}
                      >
                    <ToastContainer />
                  </i>
                </tr>
              ))
            }
          </tbody>
        </table>
      </div>
    </div>
  )
}
HOSPITAL
PATIENT EXPENSE REPORT

import React from "react";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

import styles from "./patientexpensesreport.module.css";
import basestyles from "../../../shared/styles/basestyles.module.css";
import Input from "../../../shared/components/Input/Input";
import Textarea from "../../../shared/components/Textarea/Textarea";
import Button from "../../../shared/components/Button/Button";
import Header from "../../../shared/components/Header/Header";
import Backicon from "../../../shared/components/Backicon/Backicon";

const reportintialState = {
  Name: "",
  Phone: "",
  Diagnose: "",
  Treatmentcost: "",
  NameError: "",
  PhoneError: "",
}
class PatientExpenseReport extends React.Component {
state = reportInitialState;

change = (e) => {
this.setState({
[e.target.name]: e.target.value,
});
};

validate = () => {
let NameError = "";
let PhoneError = "";
let DiagnoseError = "";
let TreatmentcostError = "";

if (!this.state.Name) {
NameError = "Name cannot be empty";
}
if (!this.state.Phone) {
PhoneError = "Phone number cannot be empty";
}
if (!this.state.Diagnose) {
DiagnoseError = "This field cannot be empty";
}
if (!this.state.Treatmentcost) {
TreatmentcostError = "This field cannot be empty";
}
if (NameError || PhoneError || DiagnoseError || TreatmentcostError) {
this.setState({
NameError,
PhoneError,
DiagnoseError,
TreatmentcostError,
});

return false;
}
return true;
};

onSubmit = (e) => {
  e.preventDefault();
  const isvalid = this.validate();
  if (isvalid) {
    toast.info("Uploading Report...!", {
      autoClose: 2000,
    });
    const expense_report = {
      Name: this.state.Name,
      Phone: this.state.Phone,
      Diagnose: this.state.Diagnose,
      Treatmentcost: this.state.Treatmentcost,
    };
    localStorage.setItem(
      "expense_report_info",
      JSON.stringify(expense_report)
    );
    console.log(this.state);
    this.setState(reportintialState);
  }
}

render() {
  return (  
    <>
      <div className={styles.backgroundpic}>  
        <Header name="Expense Report" />
        <Backicon  
          onClick={() => this.props.history.goBack()}
          style={{
            background:  
              "linear-gradient(60deg, rgba(2, 12, 29, 0.644) 0%, rgba(67, 184, 192, 0.548) 98%)",
            color: "white",
          }}>
          <div className={basestyles.inputWidth}>  
            <form>
<label className={styles.hrName}>Name</label>
<Input
placeholder="Name"
type="text"
value={this.state.Name}
name="Name"
onChange={((e) => this.change(e))}
/>
<div className={styles.error}>{this.state.NameError}</div>

<label className={styles.hrphone}>Phone</label>
<Input
type="tel"
placeholder="1-555-777-6666"
value={this.state.Phone}
name="Phone"
onChange={((e) => this.change(e))}
/>
<div className={styles.error}>{this.state.PhoneError}</div>

<label className={styles.hrprob}>Diagnose</label>
<Textarea
placeholder="Accident, Flu, Injuries, Colds, Allergies, and Headaches..etc"
value={this.state.Diagnose}
name="Diagnose"
onChange={((e) => this.change(e))}
/>
<div className={styles.error}>{this.state.DiagnoseError}</div>

<label className={styles.hradvise}>Treatment Cost</label>
<Input
type="text"
placeholder="$"
value={this.state.Treatmentcost}
name="Treatmentcost"
onChange={((e) => this.change(e))}
/>
<div className={styles.error}>{this.state.TreatmentcostError}</div>
export default PatientExpenseReport;

INSURANCE REGESTERED PATIENT'S RECORD

import React from "react";
import styles from ".//patientsrecord.module.css";
import Backicon from "../../../shared/components/Backicon/Backicon";
import Header from "../../../shared/components/Header/Header";
import basestyles from "../../../shared/styles/basestyles.module.css";
import RecordBtn from "../../../shared/components/Button/RecordBtn";
import Modal from "react-modal";

Modal.setAppElement("#root");
class PatientsRecord extends React.Component {
  state = {
    patientdetailsdata: "",
    modallIsOpen: false,
}
componentDidMount() {
let completePatientDetails = [];
let patientDetails = JSON.parse(
localStorage.getItem("patient_personal_info")
);
let healthReports = JSON.parse(localStorage.getItem("health_report_info"));
for (let i = 0; i < patientDetails.existingPatientData.length; i++) {
let data = {
patientInfo: patientDetails.existingPatientData[i],
healthReport: healthReports.existingHealthReports[i],
};
completePatientDetails.push(data);
}
this.setState(
completePatientDetails,
);
}

renderModal = () =&gt; {
if (this.state.modalType === "PATIENT_INFO") {
return (
&lt;h1 className={styles.modalheading}>Patient Details&lt;/h1&gt;
&lt;h4 className={styles.modalstyle}&gt;
FirstName:&lt;/h4&gt;
&lt;p&gt;(this.state.modalData.FirstName)&lt;/p&gt;
&lt;/div&gt;
&lt;div className={styles.modalstyle}&gt;
Last Name:&lt;/h4&gt;
}
<p>{this.state.modalData.LastName}</p>
</div>
<div className={styles.modalstyle}>
<h4>Phone:</h4>
<p>{this.state.modalData.PhoneNumber}</p>
</div>
<div className={styles.modalstyle}>
<h4>Date Of Birth:</h4>
<p>{this.state.modalData.DateOfBirth}</p>
</div>
<div className={styles.modalstyle}>
<h4>Street Address:</h4>
<p>{this.state.modalData.StreetAddress}</p>
</div>
<div className={styles.modalstyle}>
<h4>Street Address 2:</h4>
<p>{this.state.modalData.StreetAddress2}</p>
</div>
<div className={styles.modalstyle}>
<h4>City:</h4>
<p>{this.state.modalData.City}</p>
</div>
<div className={styles.modalstyle}>
<h4>State:</h4>
<p>{this.state.modalData.State}</p>
</div>
<div className={styles.modalstyle}>
<h4>ZipCode:</h4>
<p>{this.state.modalData.ZipCode}</p>
</div>
<div className={styles.modalstyle}>
<h4>Country:</h4>
<p>{this.state.modalData.Country}</p>
</div>
<div className={styles.modalstyle}>
<h4>Home Phone:</h4>
<p>{this.state.modalData.HomePhoneNumber}</p>
</div>
<div className={styles.modalstyle}>
<h4>Hepatitis, MMR, B-vacc:</h4>
<p>{this.state.modalData.Hepatitis_MMR_B_vacc}</p>
</div>
Medical History:

{this.state.modalData.Significant_Medical_History}

Medication taken regularly:

{this.state.modalData.List_any_medication_taken_regularly}

Allergies:

{this.state.modalData.Allergies}

Do You Have Medical Insurance:

{this.state.modalData.DoYouHaveMedicalInsurance}

Name Of Insurance Company:

{this.state.modalData.NameOfInsuranceCompany}

Insurance Company Address:

{this.state.modalData.InsuranceCompanyAddress}

Insurance Company Address 2:

{this.state.modalData.InsuranceCompanyAddress2}

Insurance Company City:

{this.state.modalData.InsuranceCompanyCity}

Insurance Company State:

{this.state.modalData.InsuranceCompanyAddressState}

Insurance Company Zipcode:

{this.state.modalData.InsuranceCompanyZipcode}
<div className={styles.modalstyle}>
<h4>Insurance Company Country:</h4>
<p>{this.state.modalData.InsuranceCompanyCountry}</p>
</div>
<div className={styles.modalstyle}>
<h4>Policy Number:</h4>
<p>{this.state.modalData.PolicyNumber}</p>
</div>
<div className={styles.modalstyle}>
<h4>Policy Expiry Date:</h4>
<p>{this.state.modalData.PolicyExpiryDate}</p>
</div>
<RecordBtn
  btnName="Close"
  onClick={() => this.closeModal()}
  style={{
    padding: "10px 50px",
    margin: "10px",
    fontSize: "1.1rem",
  }}
/>
</>

if (this.state.modalType === "HEALTH_REPORT") {
  return (
    <>
      <h1 className={styles.modalheading}>Patient Health Report</h1>
      <div className={styles.modalstyle}>
        <h3>Name - </h3>
        <p>{this.state.modalData.Name}</p>
      </div>
      <div className={styles.modalstyle}>
        <h3>Age - </h3>
        <p>{this.state.modalData.Age}</p>
      </div>
      <div className={styles.modalstyle}>
        <h3>Phone - </h3>
        <p>{this.state.modalData.Phone}</p>
      </div>
      <h3>Past Medical History - </h3>
    </>
  )
}
<p>{this.state.modalData.PastMedicalHistory}</p>
<h3>Medications Used Before - </h3>
<p>{this.state.modalData.MedicationsUsedBefore}</p>
<h3>Diagnostic Testing - </h3>
<p>{this.state.modalData.DiagnosticTesting}</p>
<h3>Treatment Advised - </h3>
<p>{this.state.modalData.TreatmentAdvised}</p>
<table>
<thead>
<tr>
<th>ID</th>
<th>Patient Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

{this.state.completePatientDetails &&
this.state.completePatientDetails.map((patient, index) => ( |
| <tr key={index}> |
| <td>{index}</td> |
| <td>{patient.patientInfo.FirstName} {patient.patientInfo.LastName} |
| <RecordBtn btnName="Patient Details" onClick={() =>
this.openModal( |
| patient.patientInfo, |
| "PATIENT_INFO" |
| ) |
| }) |
| / > |
| <Modal isOpen={this.state.modalIsOpen} style={|
| overlay: { |
| background: |
| "linear-gradient( 60deg, rgba(2, 12, 29, 0.644) 0%, rgba(67, 184, 192, 0.548) 98%)"|
| }, |
| content: { |
| color: "#091135", |
| maxWidth: "600px", |
| margin: "0 auto", |
| }, |
| }) |
| > |
| {this.renderModal()} |
| </Modal> |
| <RecordBtn btnName="Patient Health Report" onClick={() =>
| }}
this.openModal(
  patient.healthReport,
  "HEALTH_REPORT"
)}

EMERGENCY
EMERGENCY APPOINTMENT

import React, { Component } from "react";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import basestyles from "../../../shared/styles/basestyles.module.css";
import styles from "../../../patient/pages/Appointments/appointments.module.css";
import Textarea from "../../../shared/components/Textarea/Textarea";
import Input from "../../../shared/components/Input/Input";
import Button from "../../../shared/components/Button/Button";
import Header from "../../../shared/components/Header/Header";
import Backicon from "../../../shared/components/Backicon/Backicon";
import DoctorsNearby from "../../../shared/components/DoctorsNearby/DoctorsNearby";
import searchbar from "../../../shared/components/SearchBar/searchbar.module.css";
import RecordBtn from "../../../shared/components/Button/RecordBtn";
import Modal from "react-modal";

const initialState = {
  Name: "", 
  Age: "", 
  Phone: "", 
  TypeOfProblemHeSheIsFacing: "", 
  NameError: "", 
  AgeError: "", 
  PhoneError: "", 
  TypeOfProblemHeSheIsFacingError: "", 
  checked: false, 
};

class EmergencyAppointments extends Component {
  state = {
    initialState, 
    modallsOpen: false, 
  };

  openModal = (e) => {
    e.preventDefault();
    this.setState({ modallsOpen: true });
  };
  closeModal = () => {
    this.setState({ modallsOpen: false });
  };

  inputchange = (e) => {
    this.setState({
      [e.target.name]: e.target.value,
    });
  };

  changeCheckedStatus = () => {
    this.setState((prevState) => {
      return {
        checked: !prevState.checked,
      };
    });
  }
}
componentDidMount() {
  if (!localStorage.getItem("emergency-appointment")) {
    localStorage.setItem("emergency-appointment",
    JSON.stringify({ data: [] })
  );
  }
  if (localStorage.getItem("doctor_selected")) {
    let doctor = localStorage.getItem("doctor_selected");
    this.setState({ DoctorSelected: JSON.parse(doctor) });
  }
}

validate = () => {
  let NameError = "";
  let AgeError = "";
  let PhoneError = "";
  let TypeOfProblemHeSheIsFacingError = "";

  if (!this.state.Name) {
    NameError = "Name cannot be empty";
  }

  if (!this.state.Age) {
    AgeError = "Age cannot be empty";
  }
  if (!this.state.Phone) {
    PhoneError = "Phone number cannot be empty";
  }
  if (!this.state.TypeOfProblemHeSheIsFacing) {
    TypeOfProblemHeSheIsFacingError = "This field cannot be empty";
  }

  if (NameError ||
    AgeError ||
    PhoneError ||
    TypeOfProblemHeSheIsFacingError

function handleSubmit(e) {
  e.preventDefault();
  const isValid = this.validate();
  if (isValid) {
    if (isValid && !this.state.checked) {
      toast.error("Please agree to the terms and conditions", {
        autoClose: 2000,
      });
      this.setState({
        NameError: "",
        AgeError: "",
        PhoneError: "",
        TypeOfProblemHeSheIsFacingError: "",
      });
      return;
    }
    let existingData = localStorage.getItem("emergency-appointment");
    existingData = JSON.parse(existingData);
    existingData.data.push({
      ...this.state,
      ...(id: existingData.data.length + 1),
    });
    localStorage.setItem("emergency-appointment", JSON.stringify(existingData));
    toast.info("Appointment Scheduled...!", {
      autoClose: 2000,
    });
  }
}
autoClose: 2000,
});
// clear form
this.setState(initialState);
}
}
render() {
return (<>  
<div className={styles.backgroundappoint}>
<Header name="Appointment" />
<Backicon onClick={() => this.props.history.goBack()} style={{
background: "linear-gradient(60deg, rgba(2, 12, 29, 0.644) 0%, rgba(67, 184, 192, 0.548) 98%),
color: "white",
}} />
<div className={basestyles.inputWidth}>  
<div className={styles.docName}>
{this.state.DoctorSelected && (  
<div className={searchbar.doctorlist}>
<DoctorsNearby doctor={{  
Doctor: this.state.DoctorSelected.Doctor,
Speciality: this.state.DoctorSelected.Speciality,
Address: this.state.DoctorSelected.Address,
}} />
</div>
})
</div>
</form>
<label className={styles.ptName}>Name</label>
<Input placeholder="Name" type="text" name="Name" value={this.state.Name} onChange={((e) => this.inputchange(e))}
</div>
<div className={styles.errors}>{this.state.NameError}</div>

<label className={styles.ptAge}>Age</label>
<Input
  type="text"
  min="15"
  max="100"
  step="1"
  placeholder="ex: 22"
  name="Age"
  value={this.state.Age}
  onChange={(e) => thisInputChange(e)}
/>
<div className={styles.errors}>{this.state.AgeError}</div>

<label className={styles.ptphone}>Phone</label>
<Input
  type="tel"
  placeholder="1-555-777-6666"
  name="Phone"
  value={this.state.Phone}
  onChange={(e) => thisInputChange(e)}
/>
<div className={styles.errors}>{this.state.PhoneError}</div>

<label className={styles.ptprob}>
  Type of Problem he/she is facing
</label>
<Textarea
  placeholder="Allergies, Colds and Flu, Headaches..etc"
  name="TypeOfProblemHeSheIsFacing"
  value={this.state.TypeOfProblemHeSheIsFacing}
  onChange={(e) => thisInputChange(e)}
/>
<div className={styles.errors}>
  {this.state.TypeOfProblemHeSheIsFacingError}
</div>

<div className={styles.terms}>
  <div>
    <input type="checkbox"
checked={this.state.checked}
onClick={this.changeCheckedStatus}
style={{
  height: "18px",
  width: "18px",
  cursor: "pointer",
  verticalAlign: "middle",
  marginTop: "10px",
  marginLeft: "15px",
}}
/>
</div>
</div>
<RecordBtn
btnName="I Agree to the terms and conditions"
onClick={e => this.openModal(e)}
style={{
  color: "red",
}}
/>
</div>
<Modal
isOpen={this.state.modalIsOpen}
style={{
  overlay: {
    background: "linear-gradient( 60deg, rgba(2, 12, 29, 0.644) 0%, rgba(67, 184, 192, 0.548) 98%)",
  },
  content: {
    color: "#091135",
    maxWidth: "600px",
    margin: "0 auto",
    padding: "25px",
  },
}}>  
<div className={styles.termsconditions}>
<h2>Terms and Conditions</h2>
<ul>
<li>This appointment should be filled only in the emergency
Cases
<li>Fill out the perfect address to reach</li>
<li>Extra Charges will be applied if we found the case is not emergency</li>
</ul>

<RecordBtn btnName="Close"
onClick={()=> this.closeModal()}
style={{
padding: "10px 50px",
margin: "10px",
fontSize: "1.1rem",
}}
/>
</Modal>
</div>
<Button onClick={(e) => this.onSubmit(e)}
buttonName="Submit"
style={{
color: "white",
margin: "20px 15px 30px",
padding: "10px 10px",
width: "90%",
maxWidth: "500px",
}}
>
</Button>
<ToastContainer />
</form>
</div>
</div>
</>
);

export default EmergencyAppointments;
REFERENCES


