The National University of Lesotho

Department of Mathematics and Computer Science

Faculty of Science and technology

CS4430: Distributed Database Design

Task: **Design of**

Due: __- March 2023

Participants:

	Student Number	Surname, initials	Major
1.	201702873	Mothobi, K.B	B.Eng. systems and networks
2.	201901887	Phali, K	B.Eng. systems and networks
3.	201902148	Monyau, K	B.Eng. systems and networks
4.	201902695	Ntaole, M.N	B.Eng. systems and networks
5.	201903611	Nkoe, T.F	B.Eng. systems and networks

Part 1.

The below picture shows the available hospitals in Lesotho .

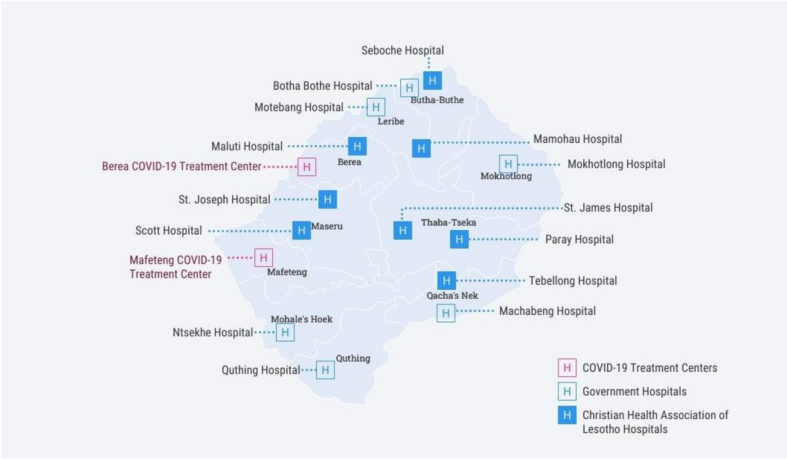
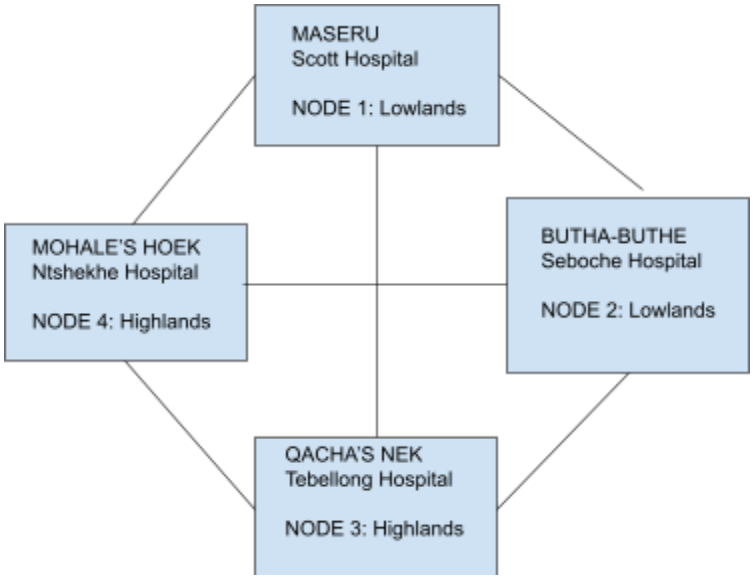
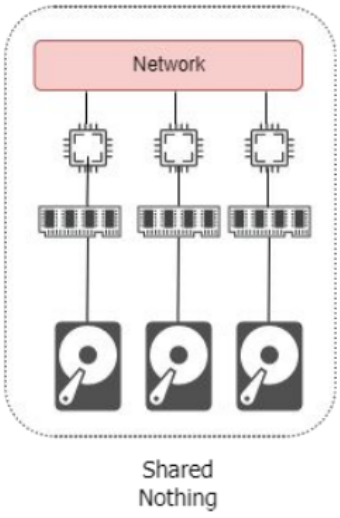


Fig 1

To distribute the system in the real world we use 4 datacenters located at 4 optimal locations with respect to the system





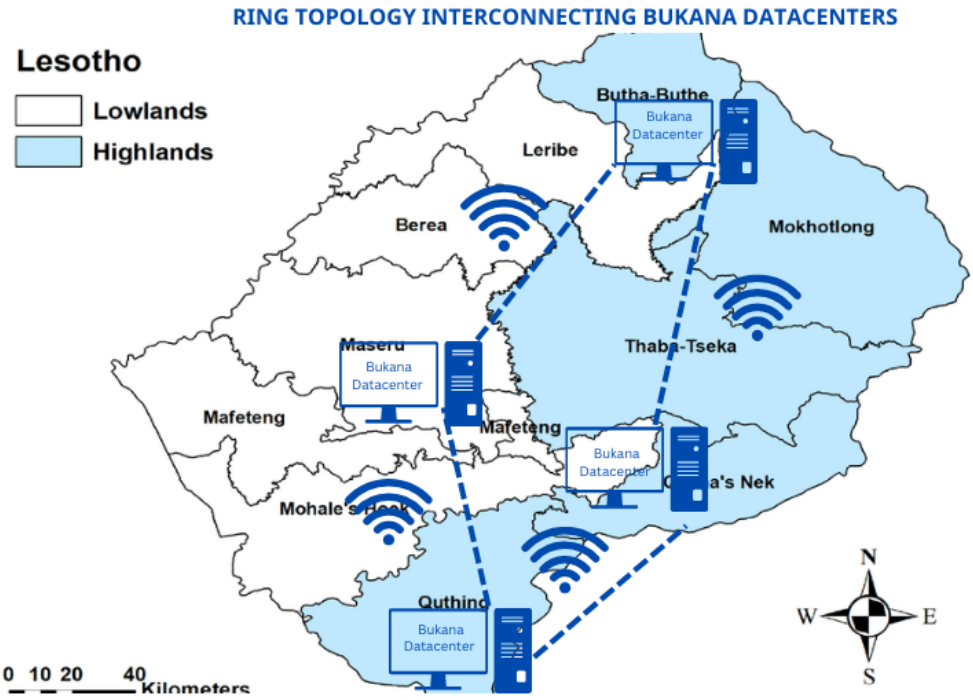

RING TOPOLOGY INTERCONNECTING BUKANA DATACENTERS
Lesotho
Lowlands
Highlands
Butha-Buthe
Bukana Datacenter
Leribe
Berea
Mokhotlong
Thaba-Tseka
Maseru
Bukana Datacenter
Mafeteng
Mafeteng
Bukana Datacenter
Mohale's Hoek
Quthing
Bukana Datacenter
N
W
E
S
0 10 20 40
Kilometers

FIG 2

Locally at each datacenter we have :

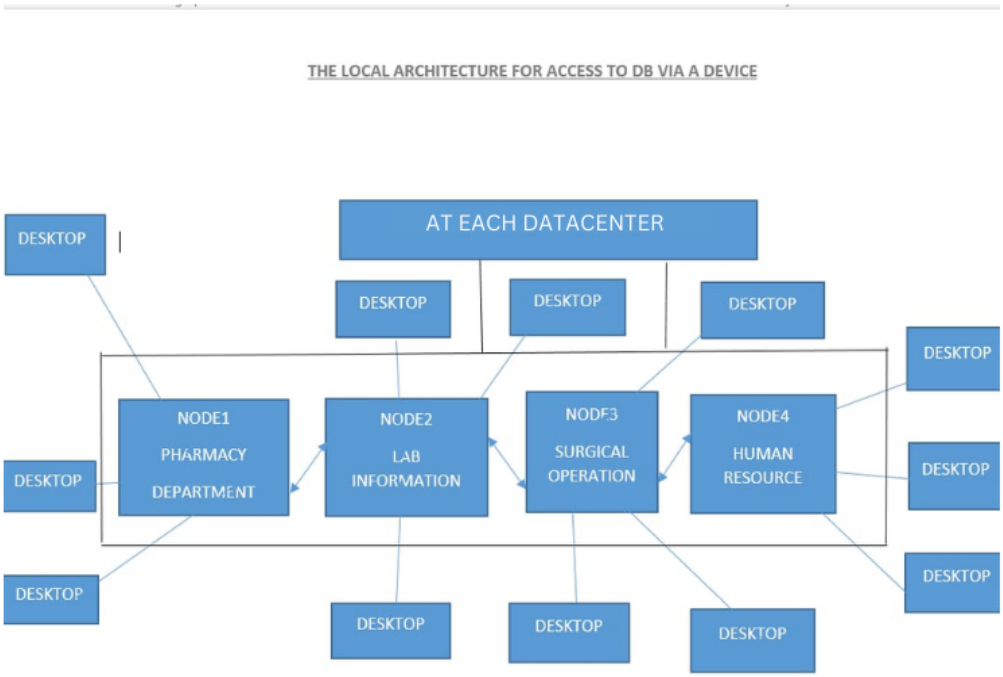


FIG 3

Hardware Architecture:
The proposed patient medical record system will be hosted on cloud-based servers to ensure scalability, reliability, and availability. The system will consist of multiple servers distributed across different geographical locations to provide redundancy and minimize downtime. The servers will be equipped with sufficient processing power, memory, and storage capacity to handle the large volumes of data generated by the system. Additionally, the system will be designed to be compatible with different types of hardware devices, such as laptops, desktops, and mobile devices, to enable users to access patient records from any location.

In a shared-nothing architecture, each node in the system has its own dedicated resources such as CPU, memory, and disk storage, and communicates with other nodes over a network to

coordinate data access and processing. This approach allows for horizontal scalability, as new nodes can be added to the system without requiring changes to the existing node

Networking Architecture:

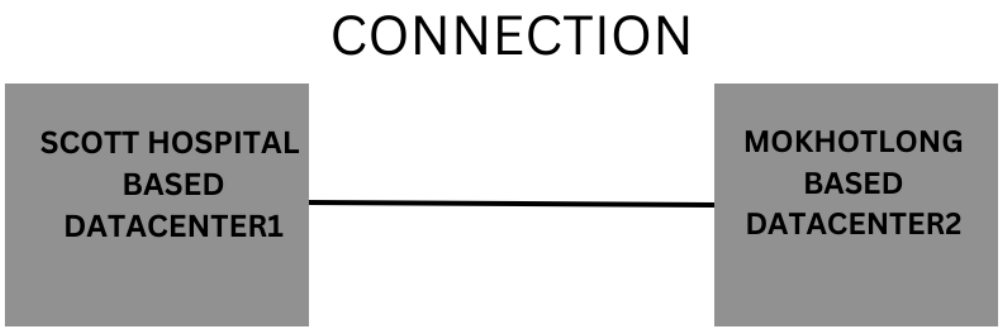
The networking architecture of the system will consist of two components: the local site network and the remote network. The local site network will be established within each hospital or clinic, and it will connect the hospital's or clinic's devices to the local server. The local network will use wired or wireless connections, depending on the specific hospital or clinic's requirements. The remote network will connect the distributed servers to enable data synchronization and replication between them. The remote network will use a combination of leased lines, virtual private networks (VPNs), and internet connections to facilitate secure and reliable data transfer between the servers.

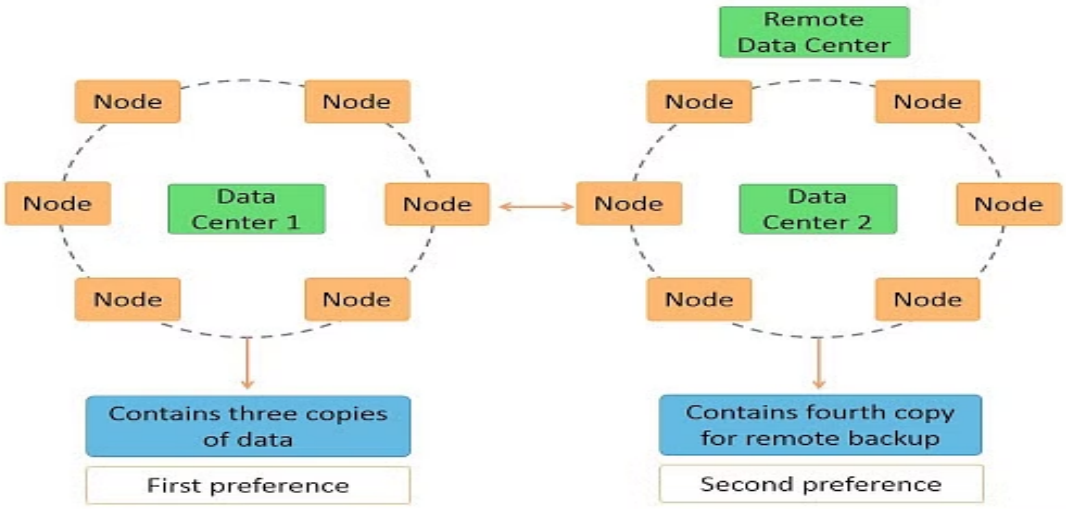
Software Architecture:

The proposed patient medical record system will be designed using a microservices architecture, with each microservice responsible for a specific aspect of the system's functionality. The backend will be implemented using Cassandra, a distributed NoSQL database that is highly scalable and fault-tolerant. The frontend will be built using React, a popular JavaScript library for building user interfaces, to provide a responsive and user-friendly interface. Additionally the design will be designed using Penpot for the user interface. The system will also incorporate security measures such as encryption algorithms (e.g., Caesar cipher), access controls, and authentication mechanisms to ensure data privacy and security. As for applications, there are various options that can be used to build a patient health record system on top of the distributed database platform of our choice. We employ Electronic Medical Record (EMR) software: EMR software is designed specifically for managing patient health records, and can include features such as patient registration, scheduling, medical history, lab results, and more.

Rationale for Design Choices:
The proposed system architecture is designed to address the specific challenges of Lesotho's healthcare system, such as the lack of an integrated medical records system, data privacy and security concerns, and the need for remote access to patient records. The use of cloud-based servers, microservices architecture, and distributed database systems enables the system to be highly scalable, fault-tolerant, and available across different geographical locations. The use of React for the frontend and Cassandra for the backend ensures that the system is responsive, user-friendly, and secure. The use of access controls, encryption algorithms, and authentication mechanisms provides additional security to protect patient data from unauthorized access or manipulation. Overall, the proposed system architecture is designed to meet the objectives of improving access to patient records, providing efficient and effective treatment, and improving health outcomes for patients.

Part 2.
For the experimental part we demonstrate with two datacenters each consisting of a single node
One datacenter in the highlands and the other in the lowlands





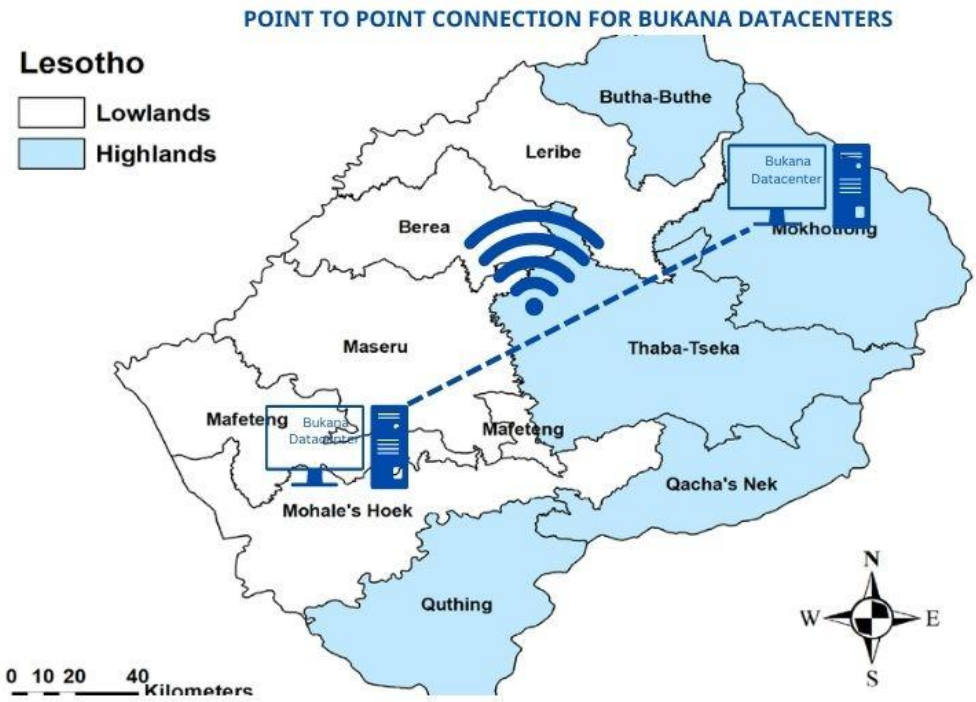


Figure 2.1: Peer-to-Peer nodes to be created in cassandra.

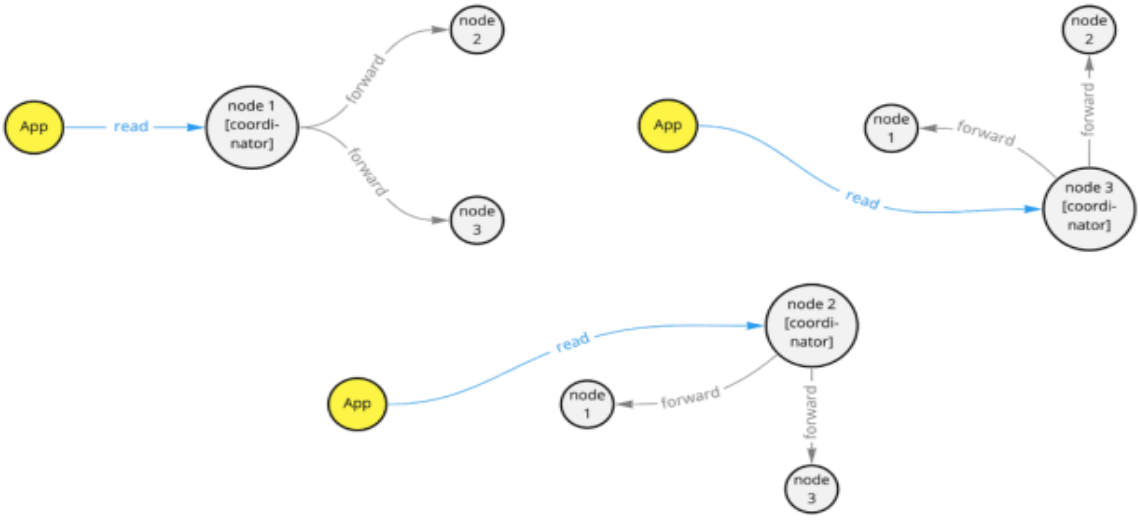


Figure 2.2: Every node is a coordinator in Cassandra

-Hardware architecture.
Host machine with minimum 8GB memory and more than 20GB free disk space.

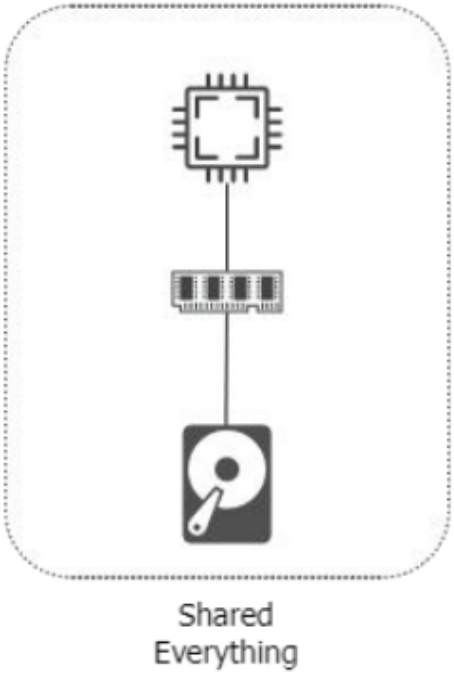


Figure 2.3: share CPU, RAM and Hard Disk

Due to use of cassandra on host machine, the hardware architecture to be deployed will share everything (i.e figure 2.3)

-Networking architecture (Local Site and Remote).

Cassandra enables communication between nodes through a peer-to-peer protocol called the Gossip protocol. The Gossip protocol is used to disseminate state information about the Cassandra cluster between nodes in a decentralized and efficient way.
On the same LAN ,two or more devices can be linked , taking the devices as two datacenters by the following docker commands using ip addresses of the machines
For the master node in datacenter 1 for example the command is

```
 docker run --name master -d -e CASSANDRA_BROADCAST_ADDRESS= 197.220.137.135 -p
7001:7001 cassandra:3.7
```

For the other node in the other datacenter

```
docker run --name worker1 -d -e CASSANDRA_BROADCAST_ADDRESS=197.220.136.142 -p
7001:7001 -e CASSANDRA_SEEDS= 197.220.137.135 cassandra:3.7
```

-Software architecture (Platform and Application).

The backend will be implemented using Cassandra, a distributed NoSQL database that is highly scalable and fault-tolerant. The frontend will be built using React, a popular JavaScript library for building user interfaces, to provide a responsive and user-friendly interface. We employ Electronic Medical Record (EMR) software: EMR software is designed specifically for managing patient health records, and can include features such as patient registration, scheduling, medical history, lab results, and more.

Figure 2.1 shows that four peer-to-peer nodes will be implemented and they will be representing four nodes where two are for highlands being Tebellong hospital node and Ntsekhe hospital node, and another two for lowlands being Scott hospital node and Seboche hospital node.

Figure 2. 2 shows that, in cassandra, every node in the cluster is a coordinator for client requests. This means that each node is capable of receiving requests from clients and coordinating the read or write operations across the cluster to ensure data consistency.When a client sends a request to Cassandra, the request is first received by a client driver or connector. The driver then forwards the request to one of the nodes in the cluster, which becomes the coordinator for that request. The coordinator is responsible for routing the request to the appropriate nodes that hold the data being requested, and for aggregating the results of the operation.

Cassandra is the choice for this project because of many advantages which include:

1. Low Cost because it is open source.
2. High performance
3. High availability

Part 3.

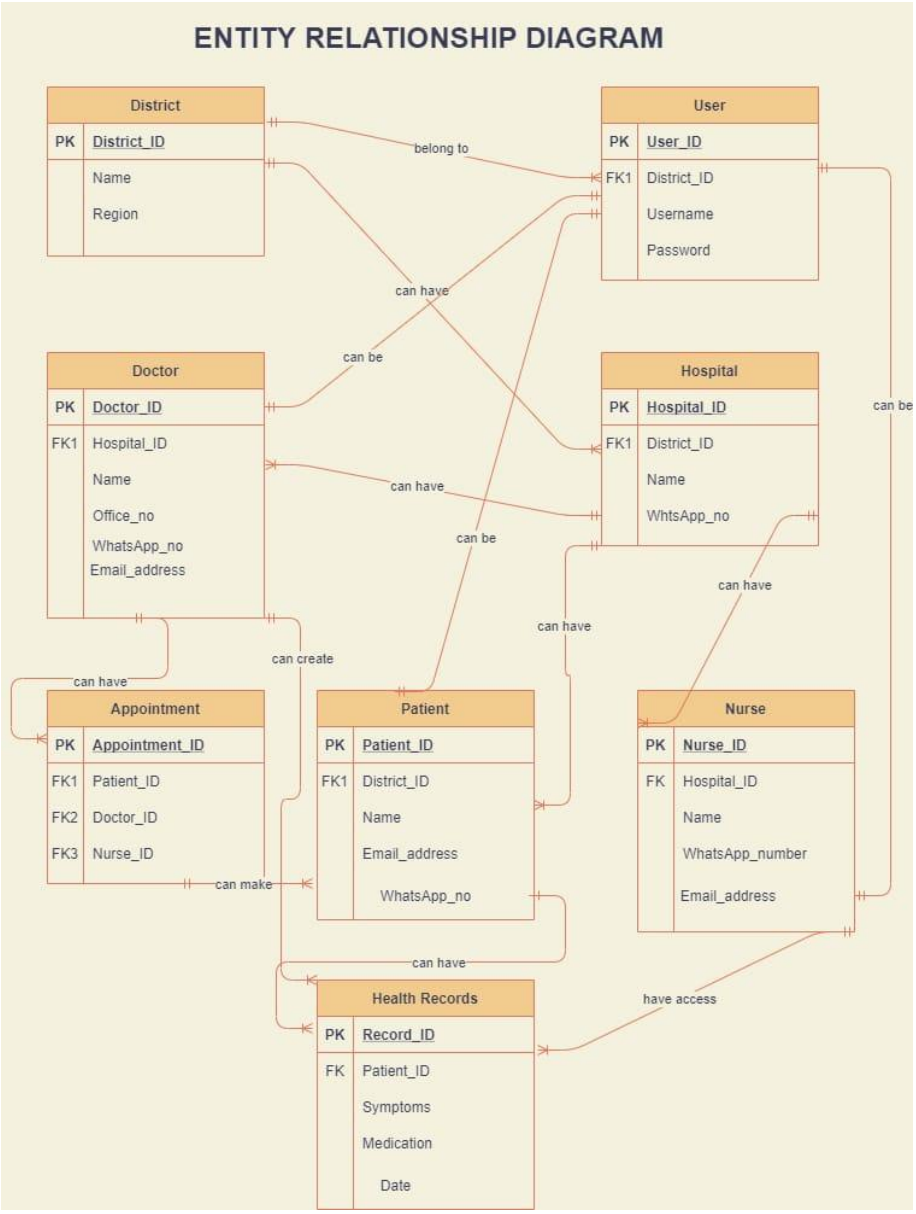
ENTITIES

HOSPITAL (Hospital_id,District_id,name,whatsApp_no)
DISTRICT (District_id,name,region)
PATIENT(Patient_id,name,District_id, email_address,whatsApp_no)
USER (User_id,username,password,ID,district_id)
DOCTOR (Doctor_id, Hospital_id,name,office_number,whatsApp_no,email_address)
NURSE (Nurse_id,Hospital_id,name,whatsApp_no,email_address)
HEALTH_RECORD(Record_id,Patient_id,symptoms,medication,date)
APPOINTMENT (Appointment_id,Patient_id,Doctor_id,date,time)

RELATIONS

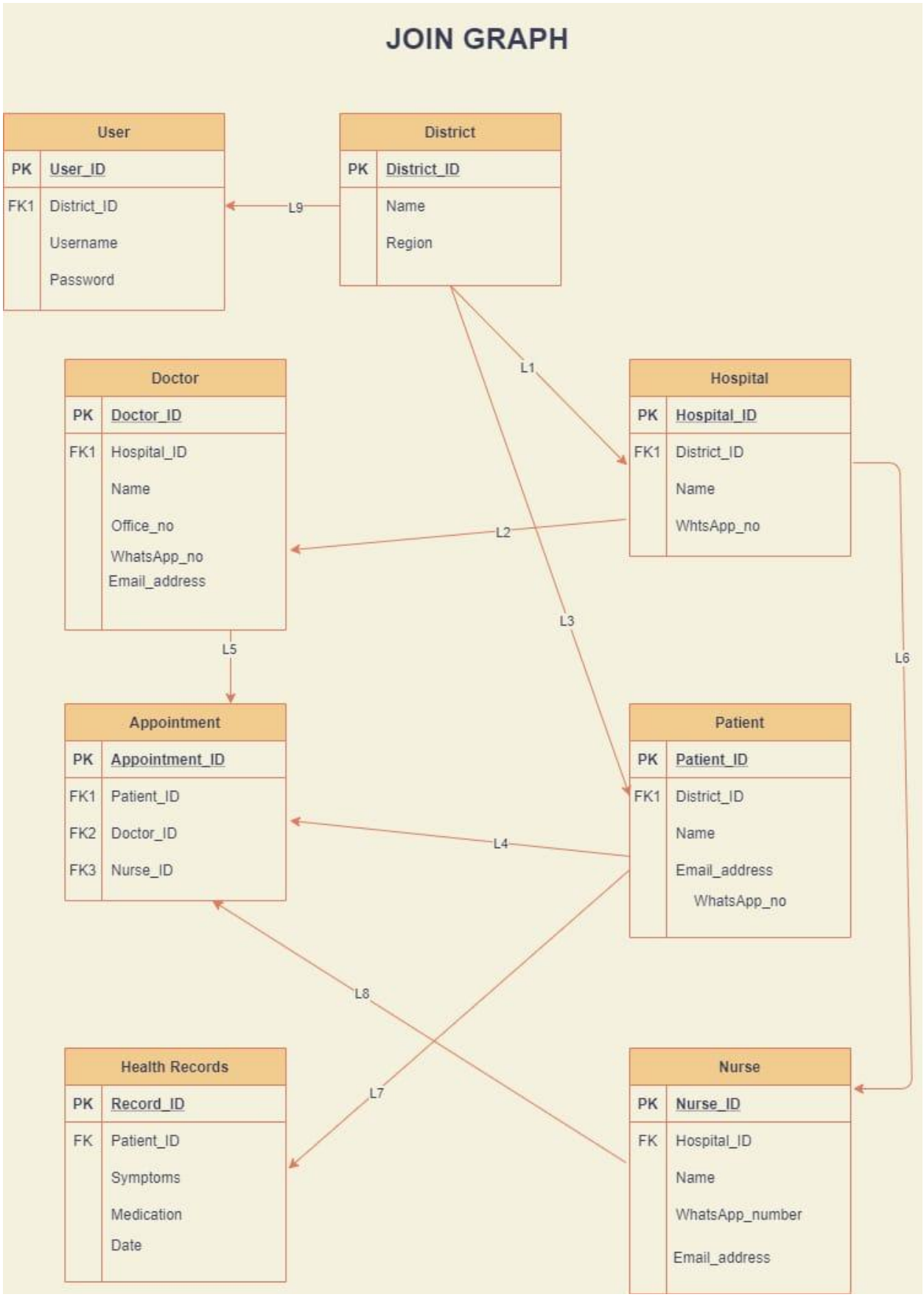
- *A HOSPITAL is located in one DISTRICT, but a DISTRICT can have many HOSPITALs.*
- *A PATIENT can have many HEALTH_RECORDs, but a HEALTH_RECORD belongs to only one PATIENT.*
- *A patient can have many appointments , but one appointment can be for only one patient*
- *A doctor can have many appointments , but only one appointment can belong to one doctor*
- *A DOCTOR can create and update many HEALTH_RECORDs, but a HEALTH_RECORD can only be associated with one DOCTOR.*
- *A NURSE can create and update many HEALTH_RECORDs, but a HEALTH_RECORD can only be associated with one NURSE.*
- *A USER can be associated with either a doctor , nurse or patient , but a patient or doctor or nurse can only be associated with one user account .*

ER-DIAGRAM


ENTITY RELATIONSHIP DIAGRAM
District
PK District_ID
Name
Region
belong to
User
PK User_ID
FK1 District_ID
Username
Password
can have
can be
Doctor
PK Doctor_ID
FK1 Hospital_ID
Name
Office_no
WhatsApp_no
Email_address
Hospital
PK Hospital_ID
FK1 District_ID
Name
WhtsApp_no
can be
can have
can be
can have
can have
can create
can have
Appointment
PK Appointment_ID
FK1 Patient_ID
FK2 Doctor_ID
FK3 Nurse_ID
Patient
PK Patient_ID
FK1 District_ID
Name
Email_address
WhatsApp_no
Nurse
PK Nurse_ID
FK Hospital_ID
Name
WhatsApp_number
Email_address
can make
can have
Health Records
PK Record_ID
FK Patient_ID
Symptoms
Medication
Date
have access

JOIN GRAPH

JOIN GRAPH


User
PK User_ID
FK1 District_ID
Username
Password
District
PK District_ID
Name
Region
L9
L1
Doctor
PK Doctor_ID
FK1 Hospital_ID
Name
Office_no
WhatsApp_no
Email_address
Hospital
PK Hospital_ID
FK1 District_ID
Name
WhtsApp_no
L2
L3
L5
L6
Appointment
PK Appointment_ID
FK1 Patient_ID
FK2 Doctor_ID
FK3 Nurse_ID
Patient
PK Patient_ID
FK1 District_ID
Name
Email_address
WhatsApp_no
L4
L8
L7
Health Records
PK Record_ID
FK Patient_ID
Symptoms
Medication
Date
Nurse
PK Nurse_ID
FK Hospital_ID
Name
WhatsApp_number
Email_address

Fragmentation

Applying PHF on the District relation we get:

From , Simple predicates are:
P1 = “Lowlands”,
P2 = “Highlands”
Pr = {P1, P2}

Resulting complete and minimal predicates are:
Pr’ = {P1}, since P2 = ¬P1

Now generating minterms:
M1 = P1
M2 = ¬P1
M’ = {M1, M2} = M ; since already simplified

Therefore the resulting fragments are M1 = “Lowlands” and M2=”Highlands”.

DISTRICT_1

district_id	name	region
MSU	Maseru	Lowlands
LRB	Leribe	Lowlands
BB	Botha-Bothe	Lowlands

DISTRICT_2

district_id	name	region
MKG	Mokhotlong	Highlands
QSK	Qacha’s nek	Highlands
0QUT	Quthing	Highlands

Now on Hospital we perform derived fragmentation

Firstly starting with the Hospital Relation:
We perform semi-join with the fragments of the primary source relation and the car relation
Formula :

HOSPITALi = HOSPITAL ⋉ DISTRICTi, 1 ≤ i ≤ 2 , where Districtj =σ(Fj)(District), 1 ≤ j≤2

HOSPITAL1 = HOSPITAL ⋉ DISTRICT1 where DISTRICT1 = σREGION=”LOWLANDS” (DISTRICT)
HOSPITAL2 = HOSPITAL ⋉ DISTRICT2 where DISTRICT2 = σREGION=”HIGHLANDS” (DISTRICT)

Hospital_1

Hospital_id	Name	WhatsApp_number	District_id
777	Scott	+266 51711116	MSU
888	Motebang	+266 51272227	LRB
102	Seboche	+266 57766437	BB
101	Botha-Bothe	+266 51444748	BB

Hospital_2

Hospital_Id	Name	WhatsApp_number	District_id
133	Quthing Hospital	+266 51666669	QUT
171	Mokhotlong	+266 51444440	MKG
191	Machabeng	+266 51666661	QSK

Now for the Doctor Relation:
DOCTOR= DOCTOR ⋉HOSPITALi, 1 ≤ i ≤ 2 , where Districtj =σ(Fj)(District), 1 ≤ j≤2

DOCTOR1 = DOCTOR ⋉ HOSPITAL1
DOCTOR2 = DOCTOR ⋉ HOSPITAL2

doctor1

DOCTOR_ID	Name	WhatsApp_number	HOSPITAL_iD
999	Fane	+266 51337334	777
101	Retshepile	+266 51444748	888
121	Liabiloe	+266 51555859	102

doctor2

DOCTOR_ID	Name	WhatsApp_number	HOSPITAL_iD
105	Katleho	+266 51666669	131
109	Mothobi	+266 51444440	171
110	Masilo	+266 51555559	191

Now for the Nurse Relation:
NURSEi= NURSE ⋉HOSPITALi, $1 \leq i \leq 2$, where Districtj =σ(Fj)(District), $1 \leq j \leq 2$

NURSE1 = NURSE ⋉ HOSPITAL1
NURSE2= NURSE ⋉ HOSPITAL2

NURSE1

NURSE_ID	Name	WhatsApp_number	HOSPITAL_iD
200	Lerato	+266 51337334	777
201	Retshepile	+266 51444748	888
202	Mpho	+266 51555859	102

NURSE 2

NURSE_ID	Name	WhatsApp_number	HOSPITAL_iD

301	Mpholo	+266 51666669	133
302	Papiki	+266 51444440	171
303	Kosaka	+266 59752384	191

NOW PERFORMING DERIVED ON PATIENT RELATION
PATIENTi= PATIENT ⋉ DISTRICTi, 1 ≤ i ≤ 2 , where Districtj =σ(Fj)(District), 1 ≤ j≤2

PATIENT1 = PATIENT ⋉ DISTRICT1 where DISTRICT1 = σREGION=”LOWLANDS” (DISTRICT)
PATIENT2 = PATIENT ⋉ DISTRICT2 where DISTRICT2 = σREGION=”HIGHLANDS” (DISTRICT)

PATIENT1

PATIENT_ID	Name	WhatsApp_number	DISTRICT_ID
23	Tselane	+266 51337334	MSU
24	Makoetje	+266 51444748	LRB
25	Mpho	+266 51555859	BB

PATEINT2

PATIENT_ID	Name	WhatsApp_number	DISTRICT_ID
28	Ntori	+266 51666669	QUT
29	Bopaki	+266 51444440	MKG
30	Nkoe	+266 51555559	QSK

Now for the Health_Records Relation:

Health_Recordsi= DOCTOR ⋉HOSPITALi, 1 ≤ i ≤ 2,

Health_Records1 = **Health_Records** ⋉ Patient1
Health_Records2 = **Health_Records** ⋉ Patient2

HEALTH_RECORD1

Record_id	Patient_id	symptoms	medication	Date
23	23	headache	aspirin	12/02/2023
24	24	Nausea and vomit	zofran	12/02/2023
25	25	Sore throat	Strepsils	12/02/2023

HEALTH_RECORD2

Record_id	Patient_id	symptoms	medication	date
281	28	sore throat	Strepsils	12/02/2023
291	29	nausea	zofran	12/02/2023
301	30	headache	aspirin	12/02/2023

Now for the Appointment Relation:
Appointmenti = **Appointment**⋉Patient, 1 ≤ i ≤ 2 ,

Appointment1 = **Appointment** ⋉ Patient1
Appointment2= **Appointment** ⋉ Patient2

Appointment1

Appointment_id	Patient_id	Doctor_id	Time	Date
23	23	999	13:00hrs	12/02/2023
24	24	101	14:00hrs	12/02/2023
25	25	121	15:00hrs	12/02/2023

Appointment2

Appointment_id	Patient_id	Doctor_id	Time	date
281	28	105	9:00am	12/02/2023
291	29	109	10:00am	12/02/2023
301	30	110	11:00am	12/02/2023

Now performing DHF on user relation
USERi= USER ⋉ DISTRICTi, 1 ≤ i ≤ 2 , where Districtj =σ(Fj)(District), 1 ≤ j≤2

USER1 = USER ⋉ DISTRICT1 where DISTRICT1 = σREGION=”LOWLANDS” (DISTRICT)
USER2 = USER ⋉ DISTRICT2 where DISTRICT2 = σREGION=”HIGHLANDS” (DISTRICT)

USER1

USER_ID	userName	password	DISTRICT_ID	ID
1	tselane1	122333	MSU	23
2	katleho	34324423	LRB	105
3	Mpho	3445253	BB	202

USER2

USER_ID	userName	password	DISTRICT_ID	ID
7	Bopaki1	3424243	QUT	29
8	mothobi	44343	MKG	109
10	kosak	555559	QSK	303

Allocation

Hash-based allocation: This strategy involves using a hash function to map each fragment to a specific node in the cluster. This approach ensures that each fragment is stored on a specific node, which reduces data movement and improves performance. However, it requires a good hash function that can distribute fragments evenly across the nodes.A hash function to use can be Message-Digest-5 .The replication strategy Network Topology Strategy: This strategy takes into account the physical location of nodes in the cluster and ensures that replicas are placed in different data centers or racks for better fault tolerance.

Part 4

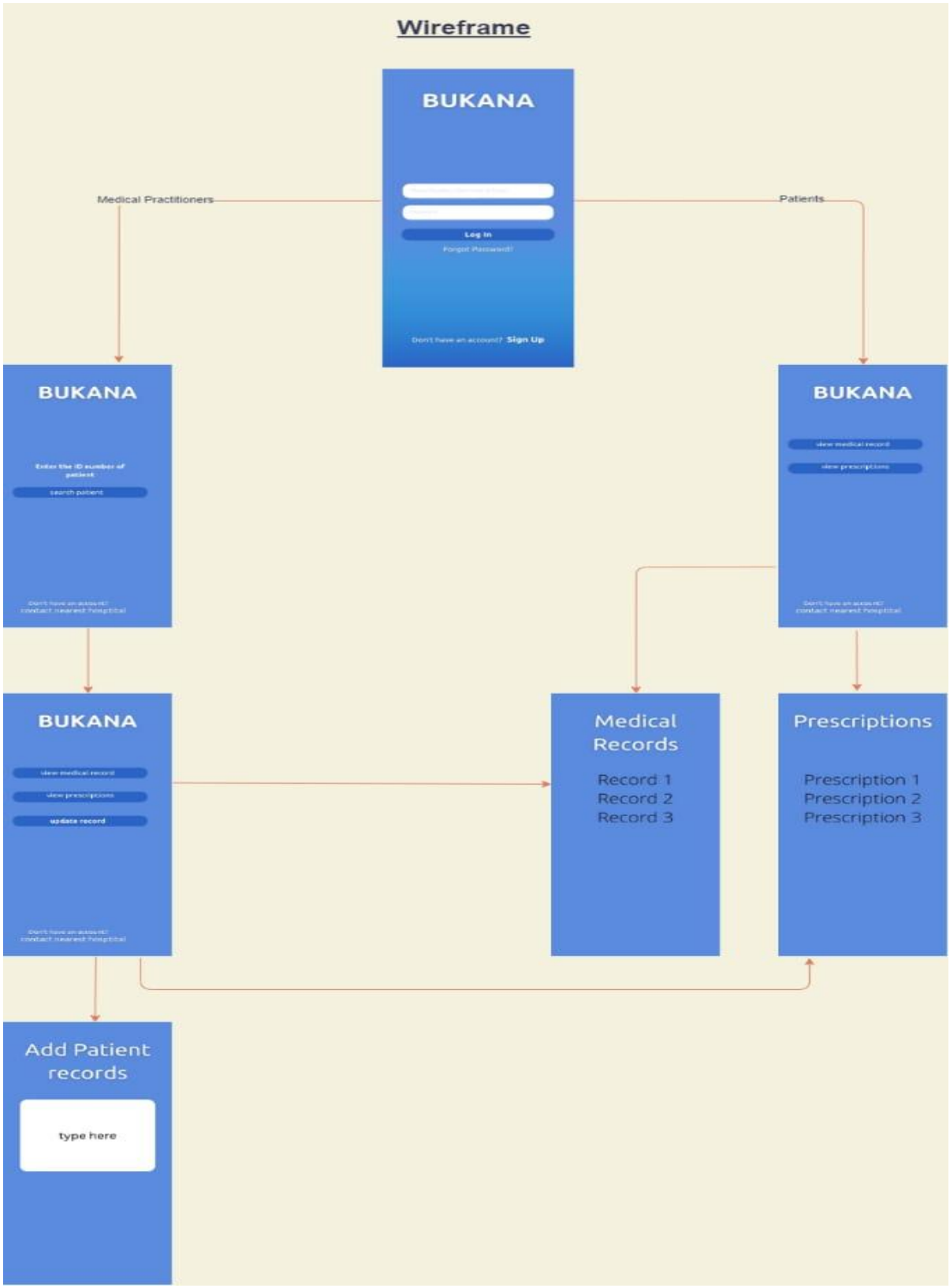


Figure 4.1

The wireframe shown in figure 4.1 above shows that all the medical practioners and patients will log in to the Bukana system. Practitioners will be able to search the patients by providing patient IDs, they can also see, modify, and add prescriptions and records. While on the other hand, patients logged in the system can only view their prescriptions and records.

Part 5

Test Case	Test Objective	Expected Outcome
1	Functionality Testing	The system will be tested for functionality and compliance with the specifications. All features of the system will be tested to ensure they are working correctly and producing accurate results.
2	Performance Testing	The system will be tested for performance and scalability. The test will determine how well the system performs under normal and peak load conditions. The test will also evaluate the system's ability to handle large volumes of data without compromising performance.
3	Security Testing	The system will be tested for security and protection against unauthorized access, data breaches, and other security vulnerabilities. The test will

		include testing the access controls, authentication mechanisms, and encryption algorithms to ensure that they are working effectively.
4	Usability Testing	The system will be tested for usability and user-friendliness. The test will involve evaluating the user interface, navigation, and overall user experience of the system.
5	Compatibility Testing	The system will be tested for compatibility with different hardware devices and software applications. The test will evaluate the system's ability to work with different devices and software applications without encountering compatibility issues.
6	Integration Testing	The system will be tested for integration with other healthcare systems and applications. The test will evaluate the system's ability to integrate with other healthcare systems and applications without encountering integration issues.

References

1. https://stackoverflow.com/questions/71063388/how-do-i-connect-to-a-remote-cassandra-instance-running-in-docker-to-retrieve-da
2. https://canva.com
3. https://www.analyticsvidhya.com/blog/2022/09/using-docker-to-create-a-cassandra-cluster/
4. Apache Cassandra Documentation: https://cassandra.apache.org/doc/latest/
5. DataStax Academy: https://academy.datastax.com/
6. "Cassandra: The Definitive Guide" by Jeff Carpenter and Eben Hewitt (O'Reilly Media, 2016)
7. "Learning Apache Cassandra" by Sandeep Yarabarla and Vijay Parthasarathy (Packt Publishing, 2015)
8. "Mastering Apache Cassandra - Second Edition" by Nishant Neeraj (Packt Publishing, 2017)
9. Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review, 44(2), 35-40.
10. Hewitt, B., & Cottingham, D. (2015). Cassandra: The Definitive Guide (2nd ed.). O'Reilly Media.
11. Das, S. (2016). Learning Apache Cassandra - Manage Fault Tolerant and Scalable Real-Time Data. Packt Publishing.