The National University of Lesotho

Department of Mathematics and Computer Science

Faculty of Science and technology

CS4430: Distributed Database Systems

Date: 23/04/23

PROJECT DESIGN

	Student Number	Surname, initials	Major
1.	201901918	Mokake N	B.Eng. systems and networks
2.	201903748	Lethoko L.L	B.sc Computer science
3.	201902032	Moeling, K.L.M	B.Eng. systems and networks
4.	201902862	Mahlo M.E	B.Sc Computer science
5.	201904086	Monyake K.S	B.Sc Computer science and mathematics

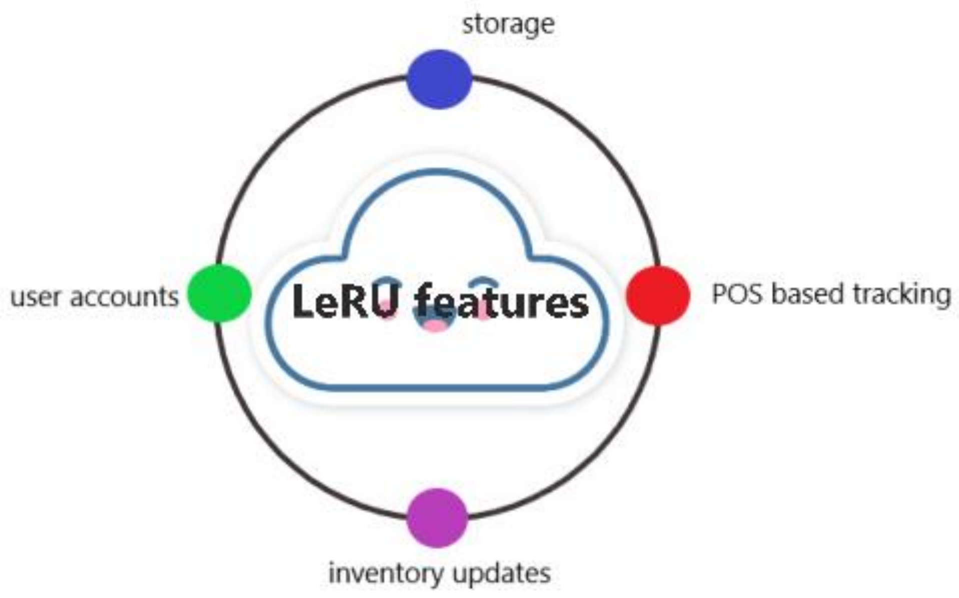
<u>DETAILED REAL-WORLD SCENARIO:</u>

Data modelling

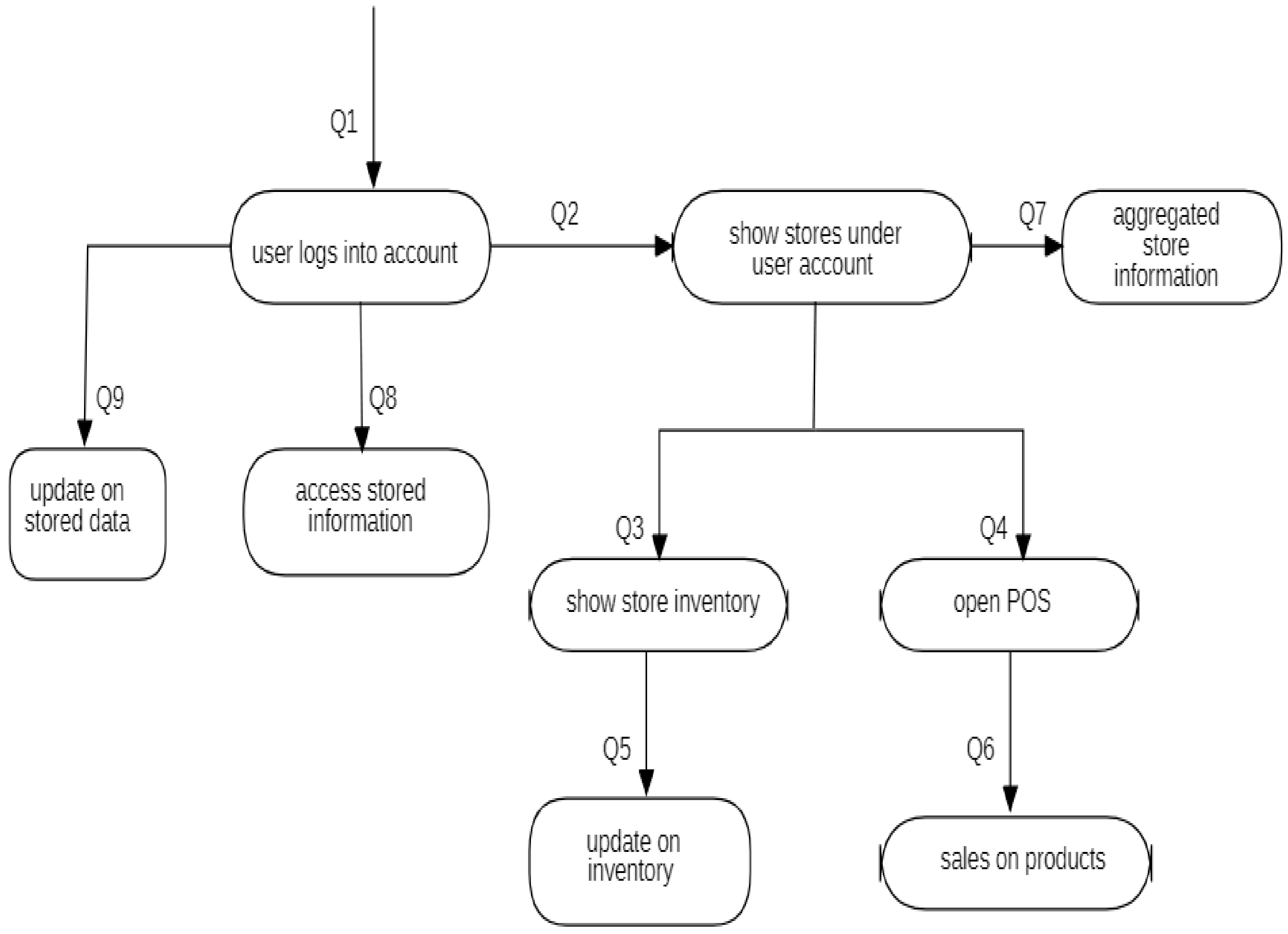
1. **Domain requirements**
2. **Conceptual data model**
3. **Query workflow**
    1. **Application workflow**
4. **Logical data models**
5. **Physical data models**

<u>Data modelling:</u>

- **<u>Application modelling:</u>**



LeRU application workflow model:

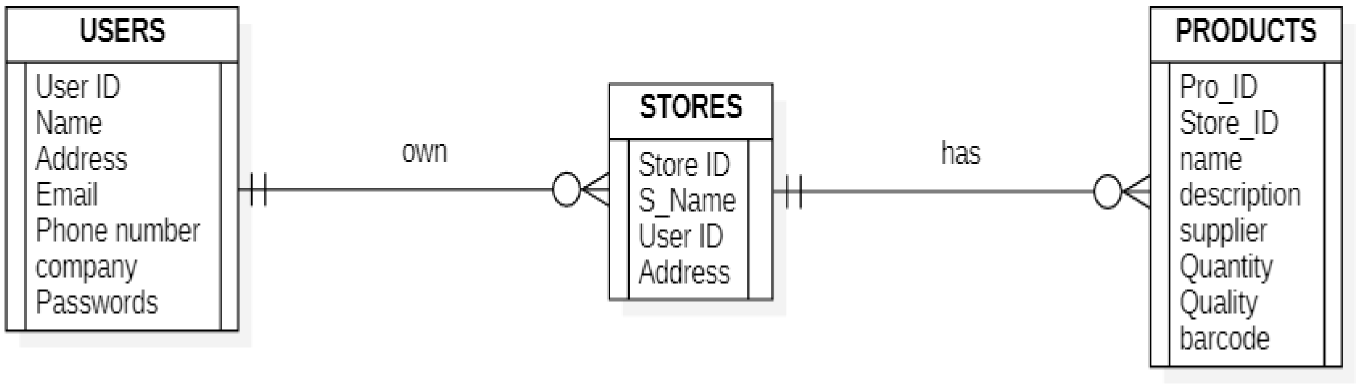


Access and query patterns:

- **Q1:find user with specific details**
- **Q2:find stores associated with user ID**
- **Q3:find inventory data associated with store ID**
- **Q4:retrieve all products saved under specified store ID**
- **Q5:update on inventory table for specific shop ID**
- **Q6:update on inventory table for specific product ID in product table for shop ID**
- **Q7:find all data for stores under specific user ID**
- **Q8:find data under user ID**
- **Q9: insert/delete/update on data banks.**

Conceptual data modelling overview:

- **Entities and entity relationships**



Logical data modelling:

- **Following apache Cassandra data modelling**
  - **The data is highly duplicated in every node and also denormalized**
  - **Cassandra has very low write costs**
  - **Most of the expected queries are write and update operations.**
- **End table designs:**
  - **The end tables will be based on the application model and conceptual model**
  - **End tables will be modelled to meet query demands**
- **Partitioning**
  - **Data partitioning is done by the DBMS.**
  - **Apache Cassandra partitions data using hashed tables and then distributes the different partitions throughout the cluster using the hashed values.**
- **Replication scheme**
  - **For the real world scenario, the replication scheme will be selected as 3. This is because it gives enough data replication to maintain reliability and fault tolerance.**

Real-World System Architecture:

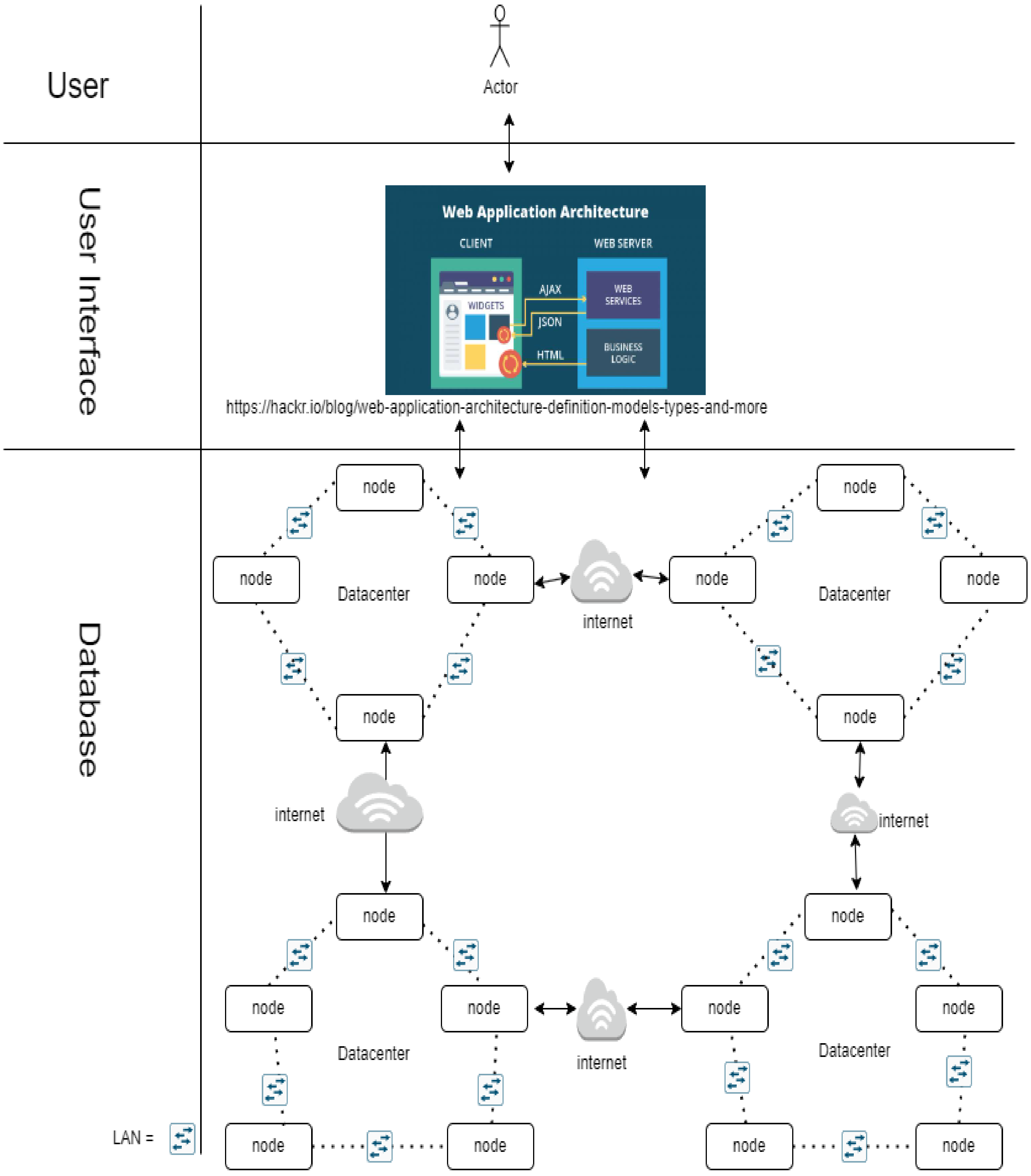
Networking Architecture, Hardware , software architecture(platform and application)

Network topology refers to how the nodes, racks in a cluster and data centers are organized and how they will communicate. In the case of our system, all the nodes in the cluster are located within the same data center hence they would use local network architecture. That is, in real world the nodes would be connected using high speed, low latency networking components, such as 10 GB Ethernet. However, to communicate with nodes from other data centers, WAN would be appropriate; for remote processing.

In terms of hardware architecture, we would use Data Center Architecture, which involves deploying the Cassandra cluster in a data center environment, typically consisting of multiple racks of servers with high-speed network connections and redundant power and cooling systems. The main reasons for this hardware architecture is it provides high reliability and fault tolerance as well as scalability.

Regarding the software architecture, we would use Linux as the operating system and Apache Cassandra as the DDBMS. As for applications, we would use LeRU, a web-based application that provides a user-friendly interface. This application will allow users to access the database systems from anywhere across the country. Moreover, web-based applications are platform-independent, which means they can be accessed from any device or operating system.

System Architecture:



Hardware Architecture: The reason for this architecture is that it provides high performance and easy scalability. It also provides high reliability and fault tolerance. Coming to four data centers, we just

divided Lesotho into its four regions, North, Central, South and West and had each region have its own data centers for high performance.

Networking Architecture: The main reason for this architecture is to maximize performance locally by use of local network architecture and make communication across different data centers possible via the internet.

Software Architecture: LeRU web application typically consists of different components that handle various tasks and functions. The front-end server, back-end server, and database server are three key components that work together to provide a complete web application.

The front-end server is responsible for presenting the user interface and handling user interactions with the application. It typically runs in the user's web browser and is built using HTML, CSS, and JavaScript.

The back-end server is responsible for processing user requests and generating responses. It typically runs on a server and is built using a server-side programming language like Python, Ruby, or PHP.

The database server is responsible for storing and managing data that is used by the web application. It typically runs on a separate server and is designed to handle large amounts of data efficiently.

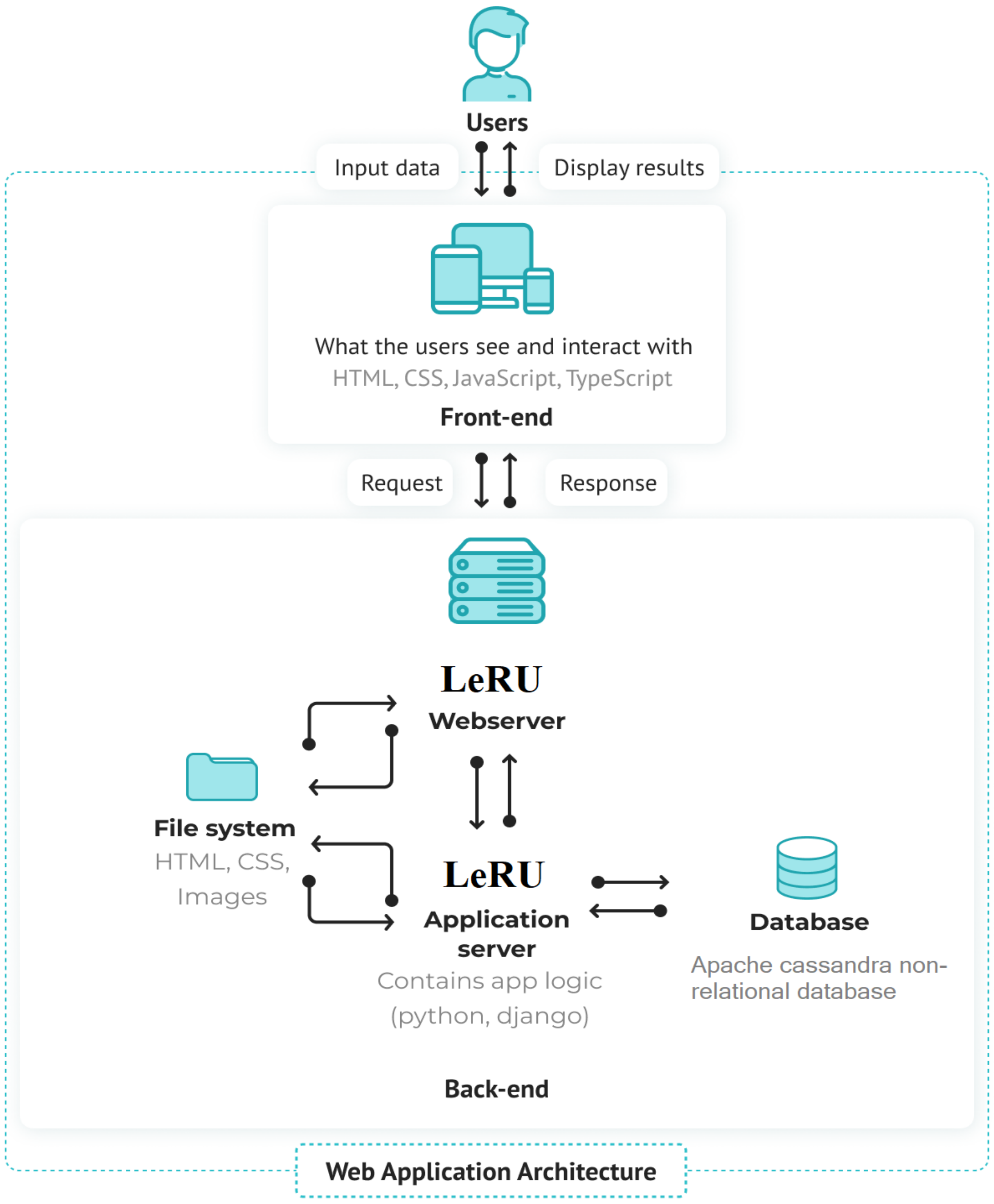
By separating these three components into independent servers, a web application can achieve several benefits:

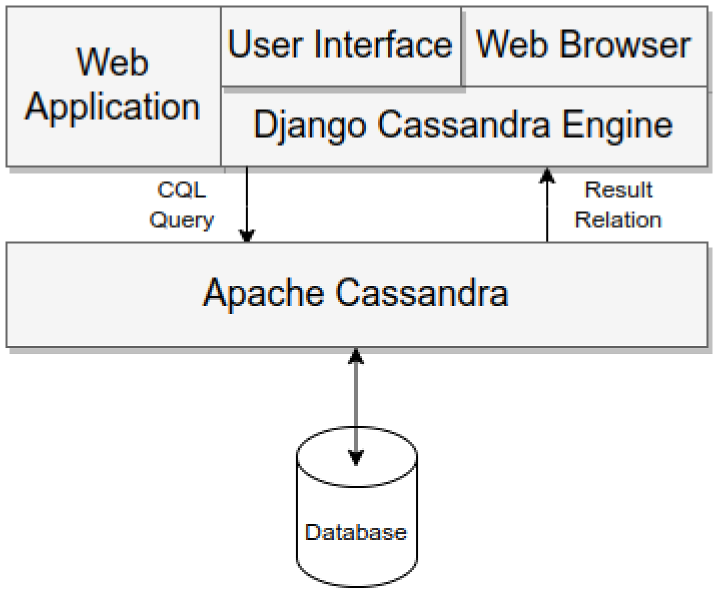
Scalability: Each server can be scaled independently to handle increased traffic or load. For example, if the application experiences a spike in user traffic, the back-end server can be scaled up to handle the increased demand, while the front-end and database servers can remain at their current capacity.

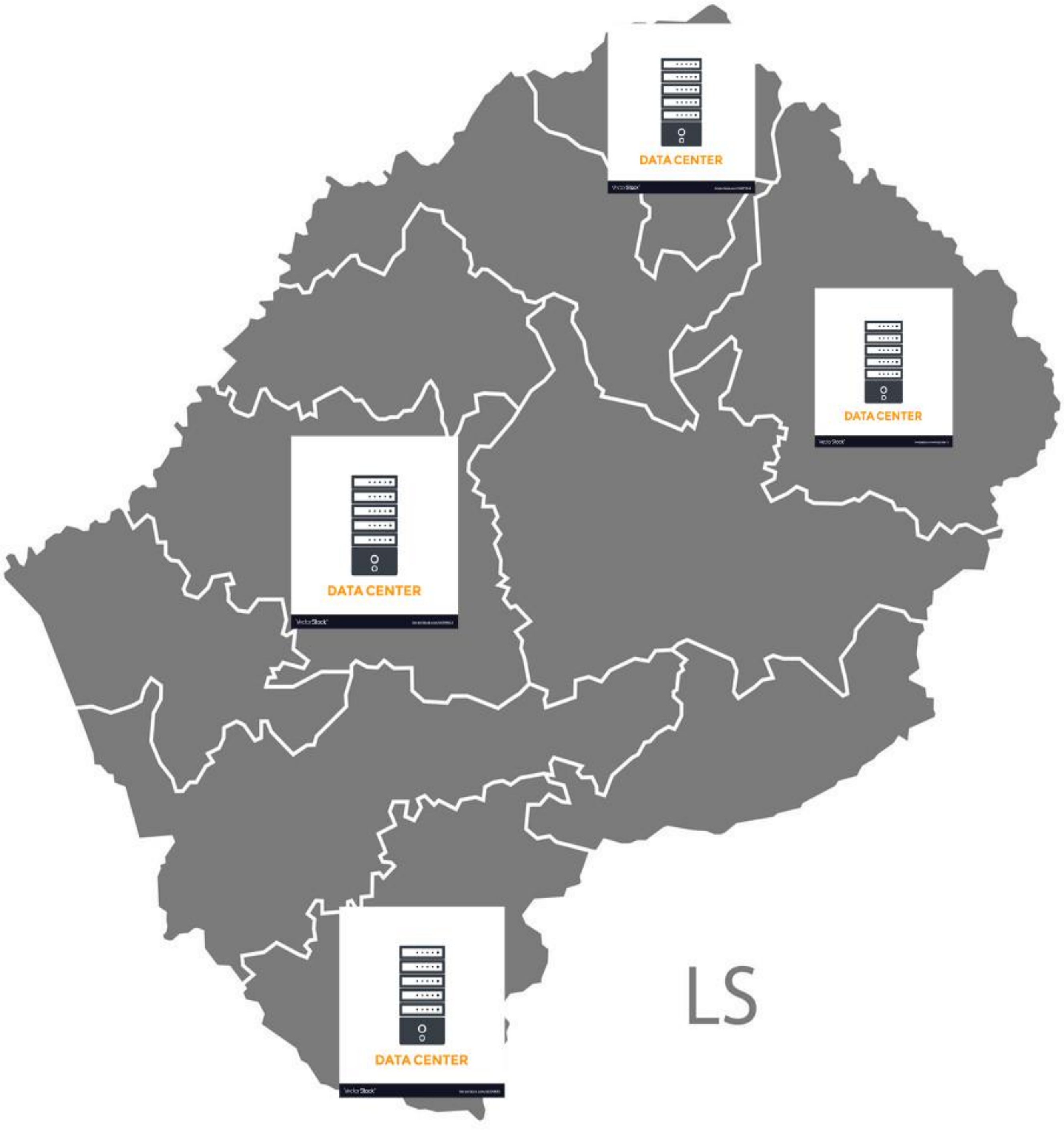
Fault tolerance: By separating the components into independent servers, a web application can be designed to be fault-tolerant. If one server fails, the other servers can continue to function, ensuring that the application remains available.

Security: Separating the components into independent servers can also improve security. By isolating the front-end server from the back-end and database servers, potential attackers have a harder time accessing sensitive information or compromising the application.

Increased performance: By separating the front-end, back-end, and database servers into independent servers can also help improve performance by allowing each server to handle a specific task or function. This means that each server can be optimized for its specific workload, which can lead to faster processing times and improved overall performance.


Users
Input data
Display results
What the users see and interact with
HTML, CSS, JavaScript, TypeScript
Front-end
Request
Response
LeRU
Webserver
File system
HTML, CSS,
Images
LeRU
Application
server
Contains app logic
(python, django)
Database
Apache cassandra non-relational database
Back-end
Web Application Architecture


Web Application
User Interface
Web Browser
Django Cassandra Engine
CQL Query
Result Relation
Apache Cassandra
Database



Data centers across the country.

2. Detailed Distributed Database Design

GCS: Cassandra is a NoSQL database. it is therefore schema less.

Query 1 handler table

find_user_by_userid	
pk	district
ck	user_id
	name
	address
	stores
	phone number
	passwords

Query 2 handler table

find_store_by_userid	
pk	user_id
ck	store_id
	address
	district

Query 3 handler table

find_inventory_data_by_store_id	
pk	store_id
ck	inventory_id
	purchase_data
	durability
	cost

Query 4 handler table

retrieve_product_by_store_id	
pk	store_id
ck	prod_id
	name
	supplier_id
	quantity
	UPC-A code

Query 5 handler table

update_inventory_by_store	
pk	store_id
ck	inventory_id
	purchase_date
	durability
	cost

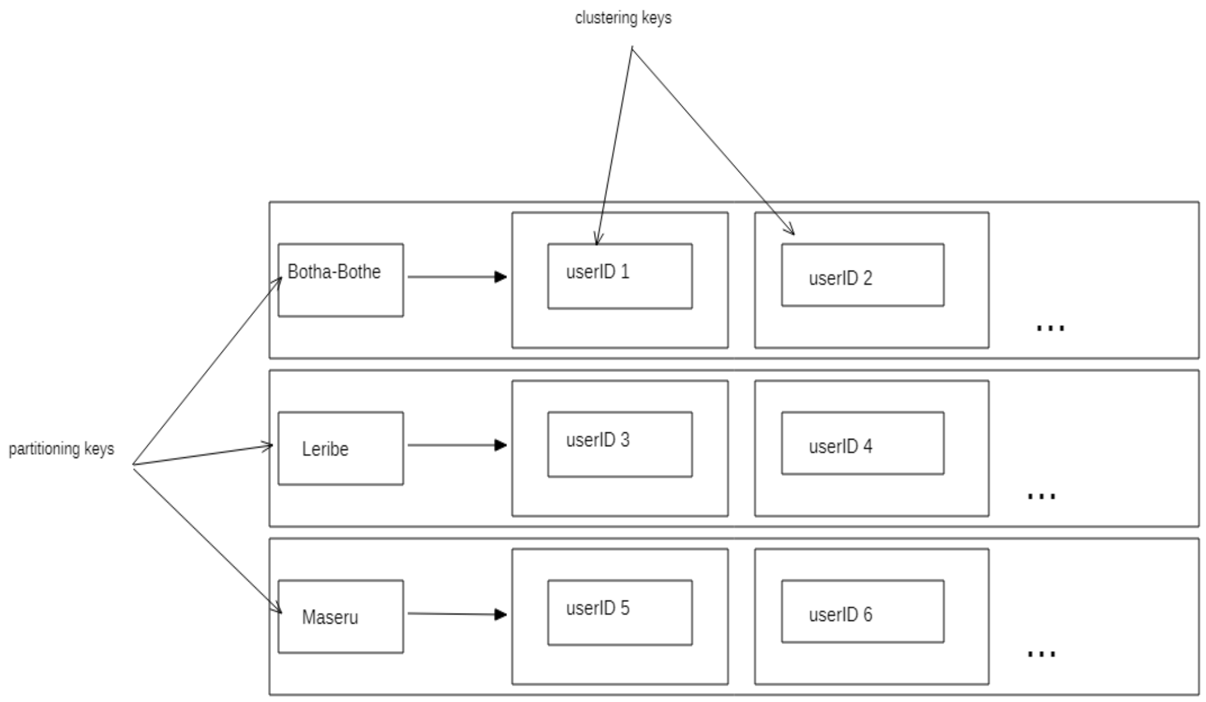
Query 6 handler table

update_product_quantity_bystore	
pk	store_id
ck	product_id
	name
	supplier_id
	quality
	description

query 7 handler table

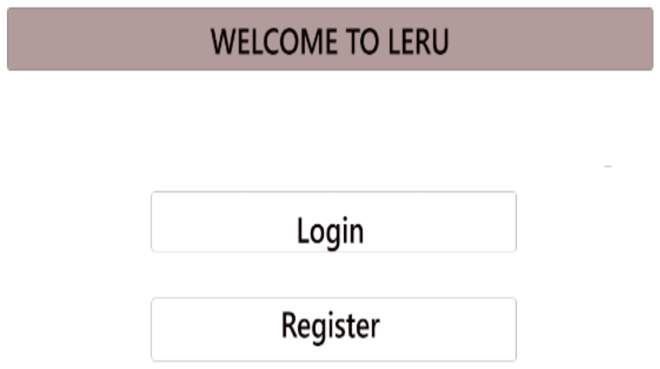
find_storedata_byuserid	
pk	User_id
ck	store_id
	name
	supplier_id
	quality
	description

Partitioning and clustering in Cassandra for table "find_user_by_userID":

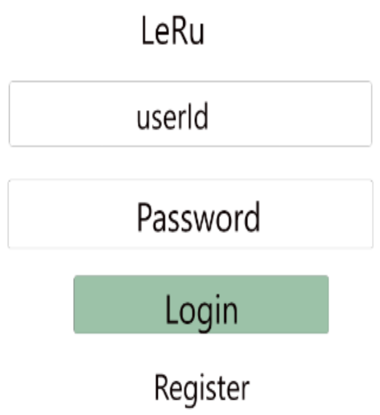


<u>UI/UX Design</u>

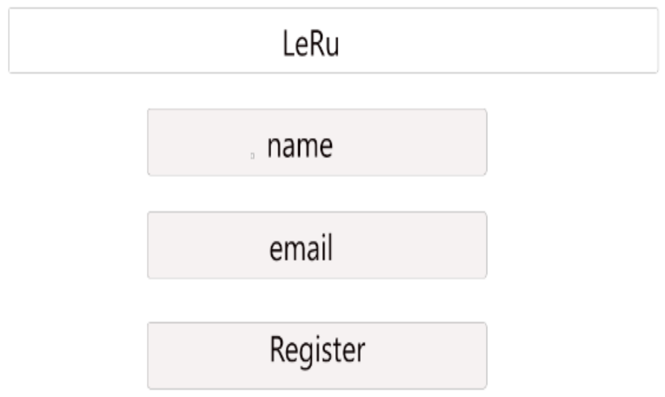
HOME PAGE



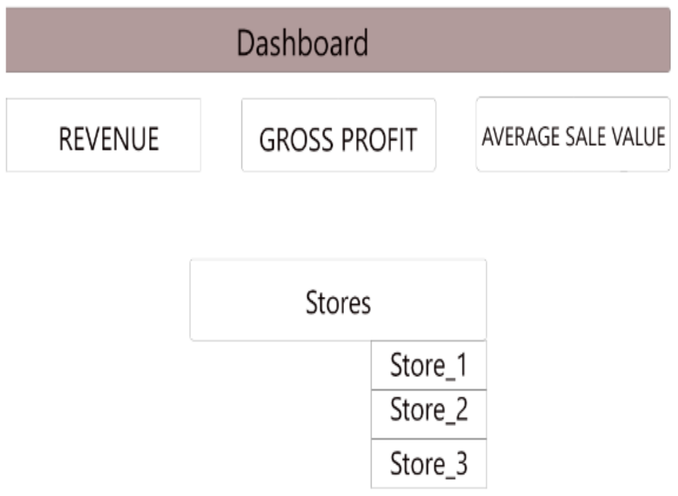
LOGIN PAGE

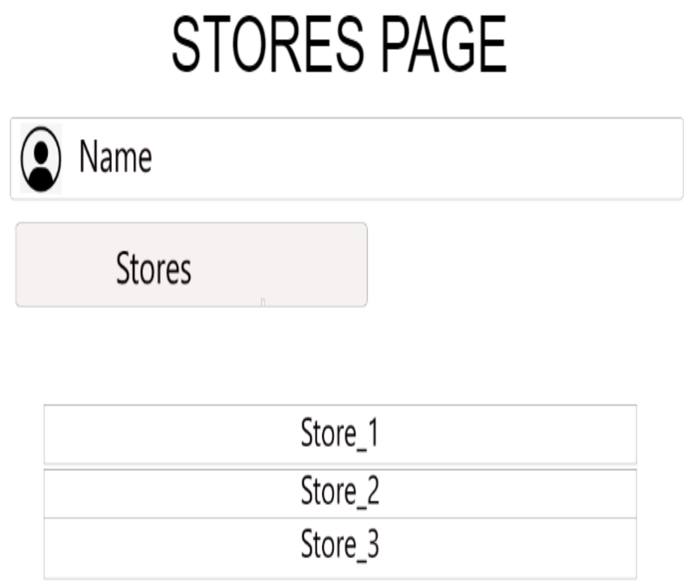


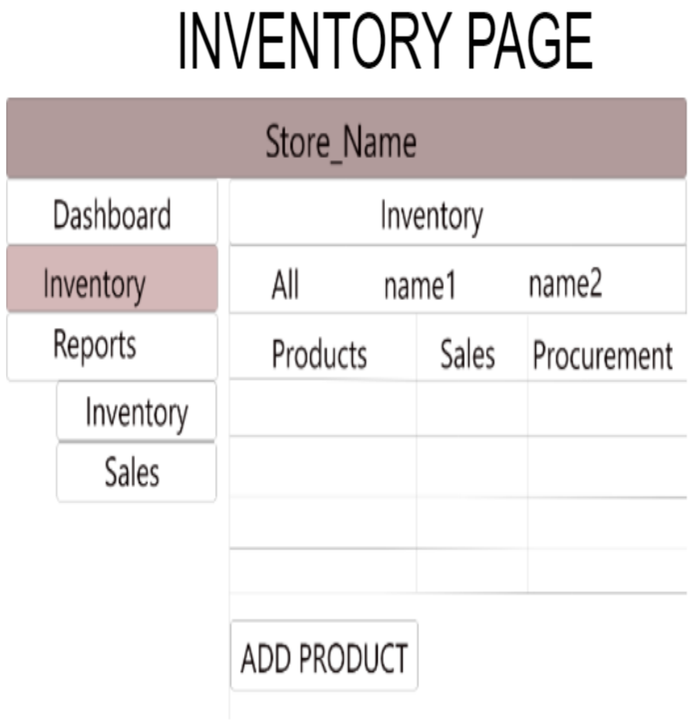
REGISTRATION PAGE

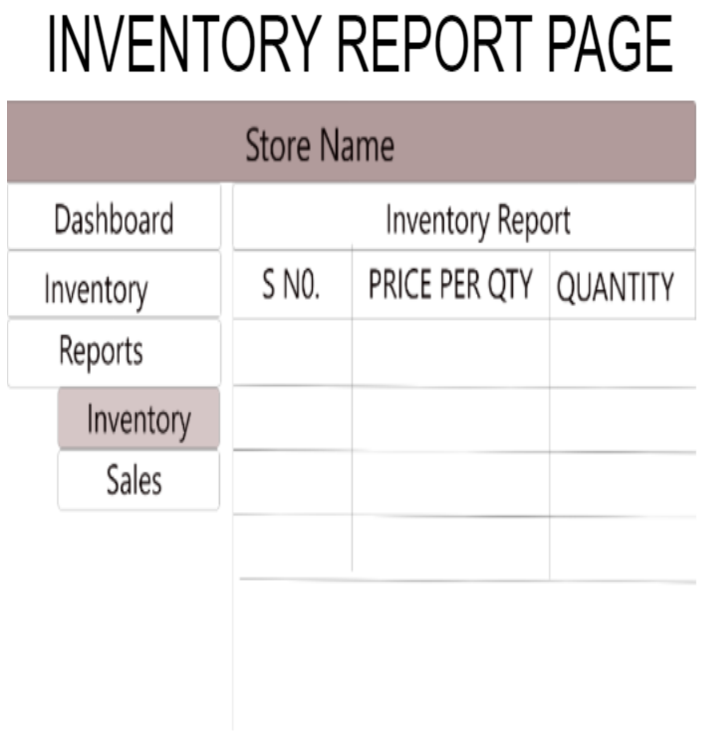


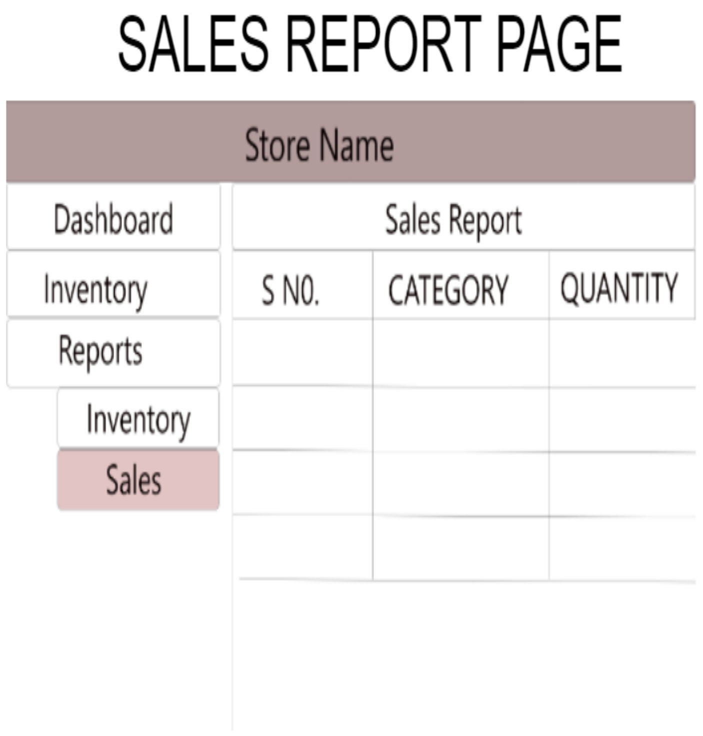
DASHBOARD PAGE





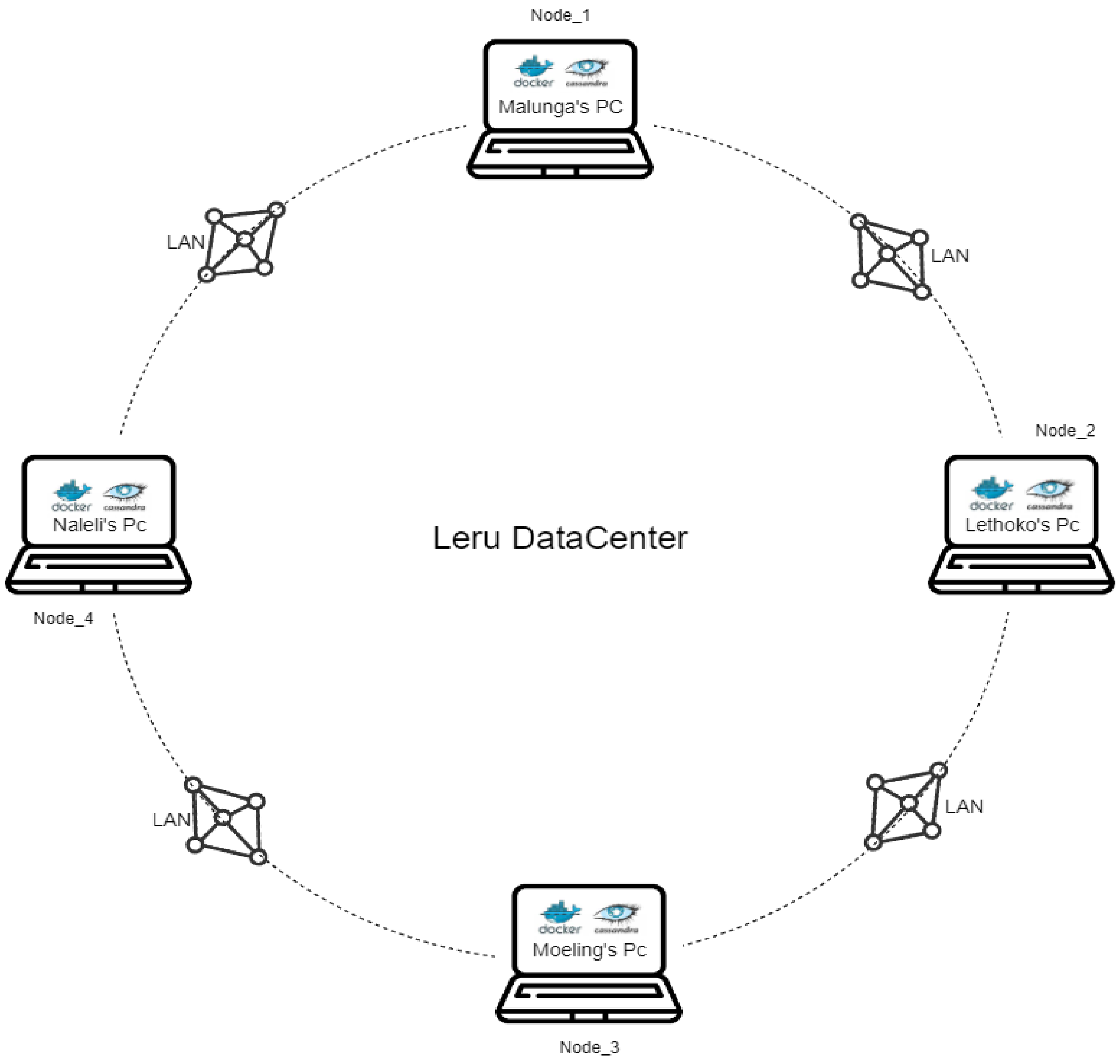






.

Detailed Demonstration System Architecture Diagram.



In the above diagram, we will have four laptops, each running a containerized instance of Cassandra in Docker, with each PC acting as a node in the Cassandra cluster to simulate the 4 datacenters in the real world plan. The database running on these machines has a key space named "LeRU" with a replication factor of 2. We have chosen a replication factor of 2 to strike an effective balance between data redundancy and storage overhead. With a replication factor of 2, two copies of data will be stored across different nodes in the Cassandra cluster, providing the necessary redundancy to ensure

data durability and availability in the event of node failures or network issues. This approach is logically correct and helps mitigate the risk of data loss or unavailability.

Now we are going to test our system using a web application. Our four laptops, connected together, will act as our data centers. We chose a web application for testing because it will allow us to simulate real-world scenarios in a controlled environment. Web applications provide a user-friendly interface that allows us to interact with the system, input data, and trigger different functionalities. This will help us verify that the system responds correctly to different inputs and performs as expected.

Lastly, the primary reason for using a four-node ring in this scenario is to assess the system's ability to handle data replication and fault tolerance. By having multiple nodes in the cluster, each replica can store a copy of the data, providing redundancy and ensuring data durability. This helps in evaluating the system's ability to handle failures, such as node failures or network issues, without losing data or causing disruptions.

Test Plan:

case	**description**	**expected _output**
1.Connect devices(nodes)	**Multiple devices that represents nodes in different districts in Lesotho but these nodes will be within the same LAN at the same location. These nodes will form a cluster.** **Use IP addresses to connect different devices.**	**-All nodes should be able to** **Find other devices on a cluster. This will be achieved by :nodetool status command.**
2. Switch off another node(s).	**Turning down some nodes in a cluster. This will be testing availability.**	**-The expectation is to have the ability to query and access data from a down node(s). If this happens, we will be assured that replication factor worked.**
3. Run queries that we designed in our application workflow.	**Making sure that all the application queries solve the problem domain we defined.**	**-The output should be what the query implies e.g ," select * from find_store_by_user_id where id = 1"** **This command should list all the stores associated with the specific user.** **-There are more queries we can test using.**