UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

Distributed File System

Submitted by:

Sumukh S Kowndinya	**PES1UG21CS640**
Tarun S Ramanujam	**PES1UG21CS665**
Tejas A Kumar	**PES1UG21CS669**
Thalleen CN	**PES1UG21CS671**

*6th Semester **K** Section*

Prof. Bhargavi Mokashi

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement:

Traditional file systems face scalability and reliability issues when dealing with large amounts of data. It is hard to store massive amounts of data in a centralised store or one particular location, both due to the volumes of data and risk of losing data that comes with such architectures.

Distributed file systems are a solution to these problems because data is not stored at one particular location or a system but distributed amongst multiple systems. This solves both the scale issue and reliability issue since data can be replicated and backups can be kept at different locations.

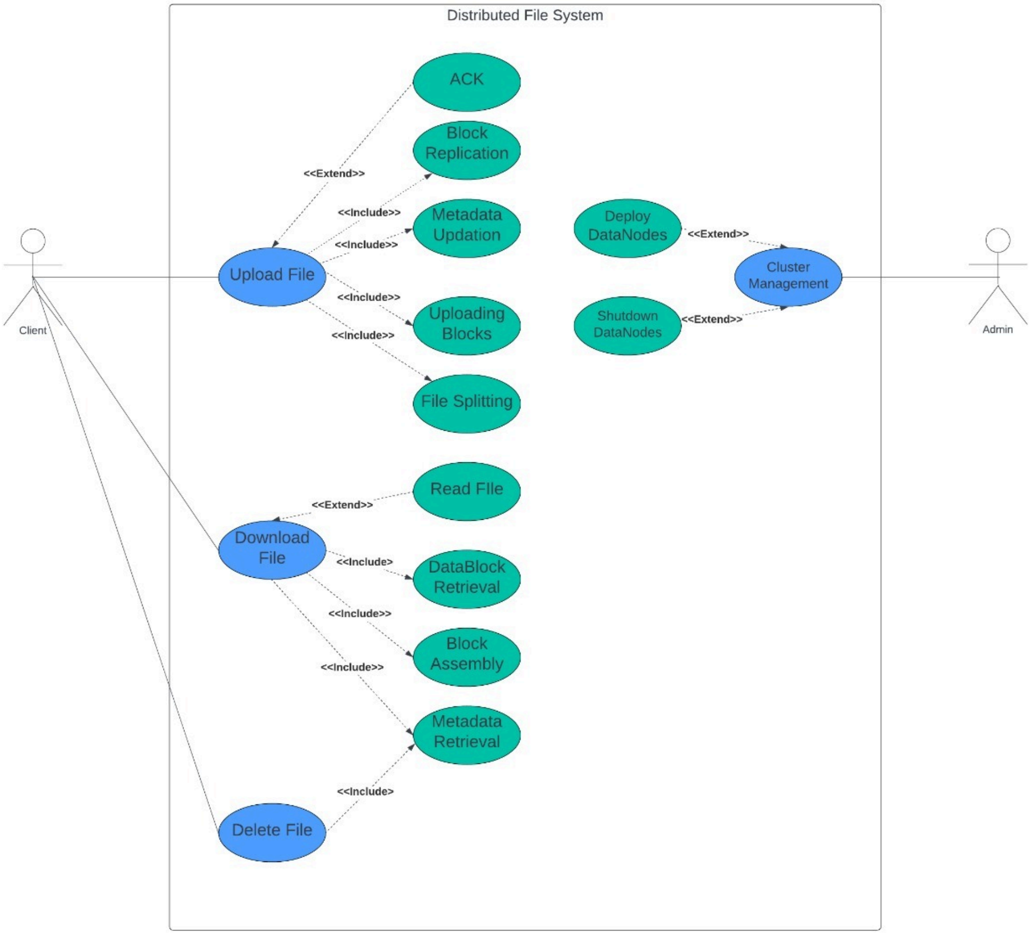
We built a distributed file system in Java that is inspired by the Hadoop Distributed File System(HDFS). The architecture resembles that of HDFS with datanode containers storing the actual data and a namenode container that manages all metadata and facilitates all client operations.

Or system has 4 datanode containers, a namenode container and a datanode manager container along with a mongoDB database with 2 collections. Our system supports file writes(uploads), file reads(downloads) and file deletion.

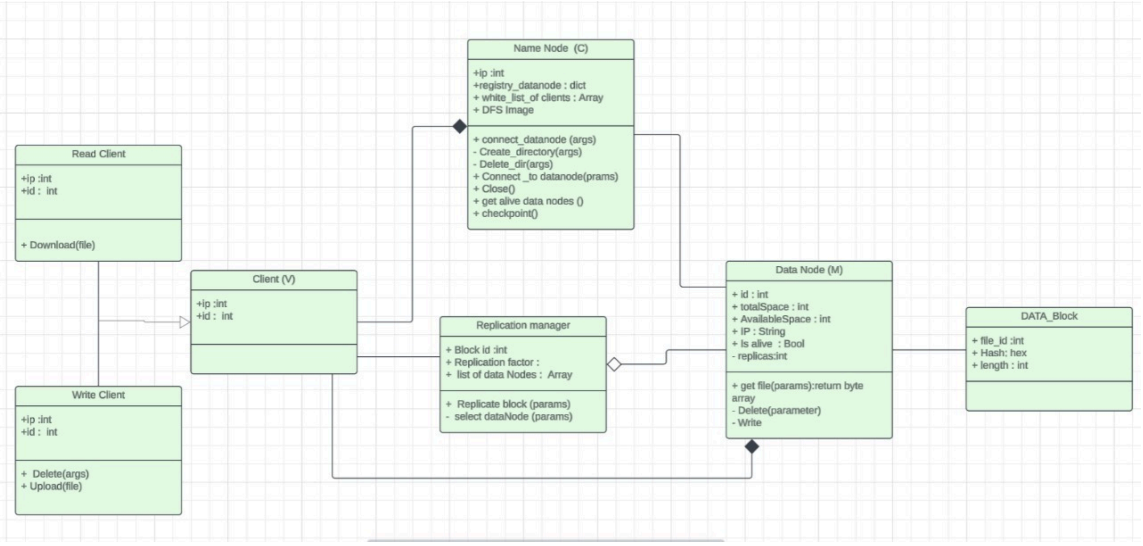
Additionally, we build a client desktop application with Java Swing that allows you to perform all these operations and the backend was purely build in Java with no usage of Swing.

Models:

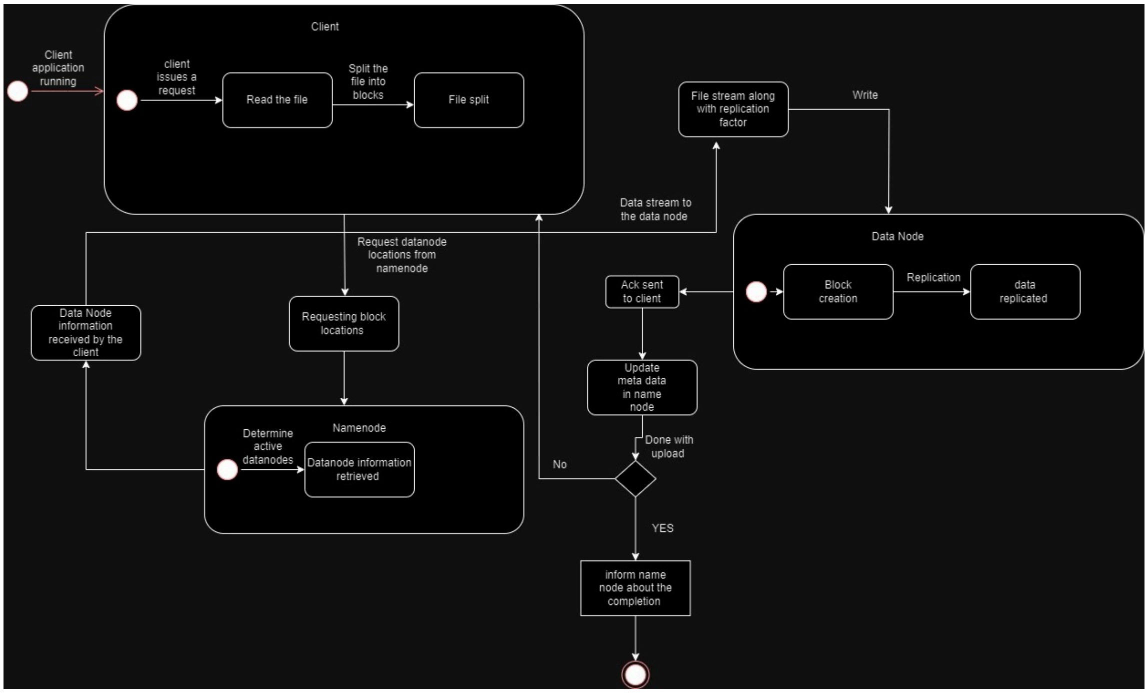
1)Use case



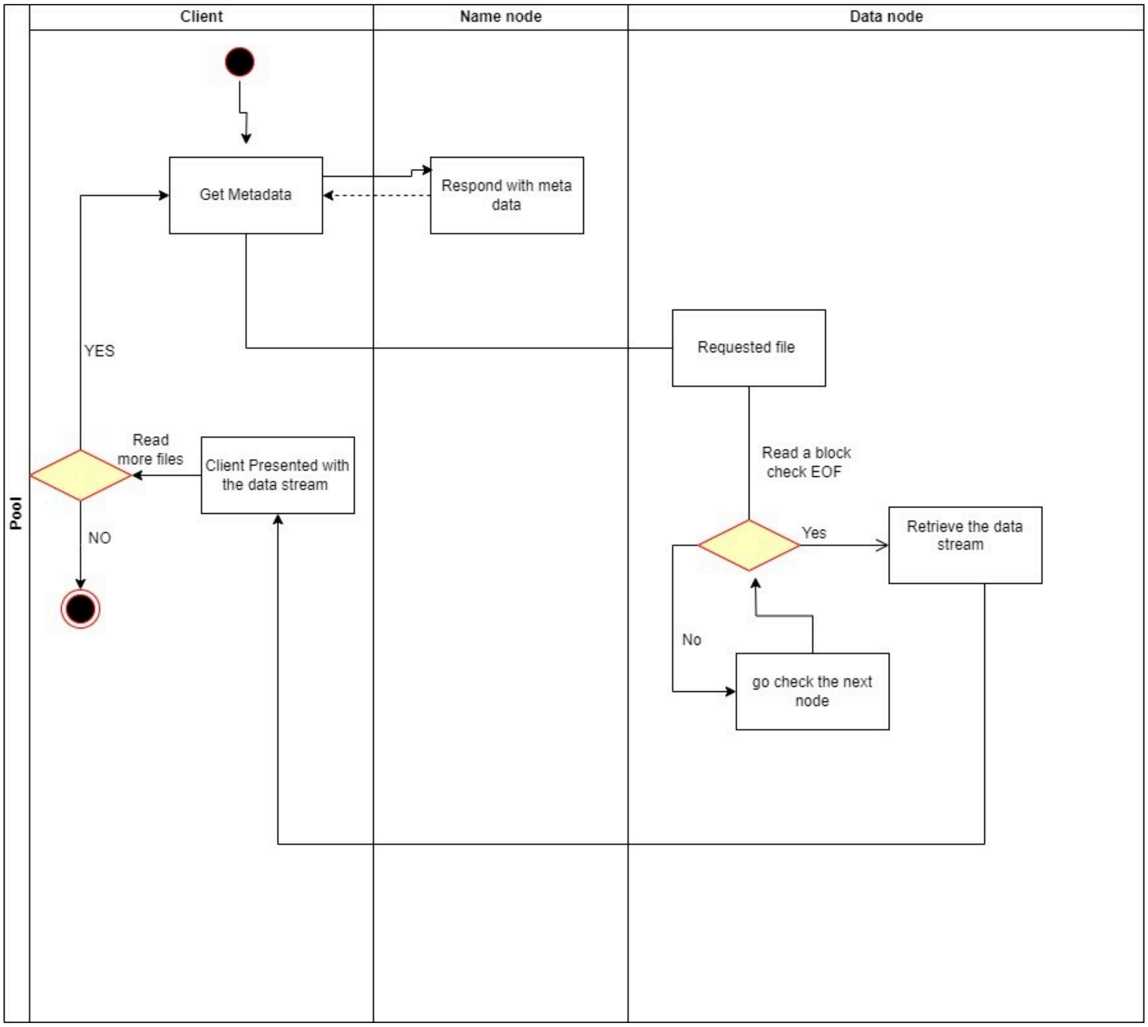
2) Class diagram



3) State diagram


Client application running
Client
client issues a request
Read the file
Split the file into blocks
File split
File stream along with replication factor
Write
Data stream to the data node
Request datanode locations from namenode
Data Node
Ack sent to client
Block creation
Replication
data replicated
Data Node information received by the client
Requesting block locations
Update meta data in name node
Namenode
Determine active datanodes
Datanode information retrieved
Done with upload
No
YES
inform name node about the completion

4) Activity diagram



Architectural Pattern:

The overall system follows the MVC pattern, with the frontend client application acting as the View module, the datanodes acting as the Model since that is where the data is stored and the Namenode acting as the Controller allowing the View and the model to interact with the namenode as the intermediary.

The client/user interacts with the desktop app(View) and issues a request(upload/download/delete), the namenode receives the request(controller), performs metadata checks and forwards the request along with necessary metadata to the Datanodes and the datanode manager(Model).

Design Principles:

The SOLID principles have been followed ensuring efficient and maintainable code. SRP is enforced throughout with classes doing limited and required tasks. OCP has been enforced via abstractions and we also have not cluttered code within individual classes. Liskov substitution has not been violated but not strictly enforced either since it is not very relevant. DIP has been enforced via abstractions and clean code structure and there exist no fat interfaces in the code hence not violating Interface Segregation.

Design Patterns:

Creational Patterns:

Factory and Singleton pattern used to create an instance of a database in the Namenode class. A factory class exists which is inherited by a MongoDB Factory that creates an instance of the MongoDB class meaning If ever there is need to migrate to a new database, another factory class can be written instead of modifying the current class. This MongoDB class follows the Singleton pattern since there need only be a single instance of a database.

Behavioural Patterns:

Since we support 3 kinds of requests: Read(GET), Write(POST) and delete(DELETE), we have 3 different handler classes to handle each kind of request separately, hence we used the Chain of responsibility pattern.
The GET handler is the base handler that passes on the responsibility to the POST handler if it cannot handle the request itself, and the POST hander does the same by passing on delete requests to the DELETE handler.(All this is in the namenode code)

Structural Pattern:

The datanode manager follows the Façade pattern by acting as a Façade class for the entire backend of the system, by being the common endpoint for all operations first.

Github Link:

https://github.com/Sumz-K/DistributedFileSystem

(The language split shows HTML as the most dominant language since the dependency files are HTML(in lib), the code is in the src folder and is completely in Java)

Individual Contribution:

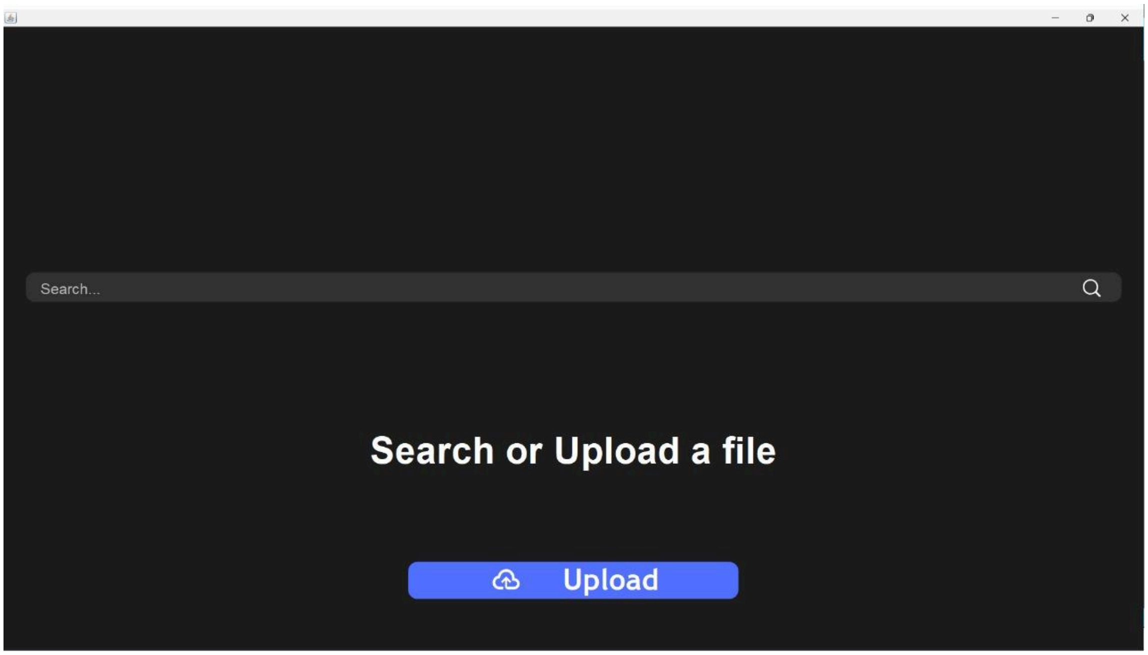
Sumukh: Worked on the backend of the system with special focus on Namenode and the database.

Tejas: Worked on the datanode manager, datanodes and hosting the entire system.

Tarun: Built the frontend from scratch, adding local file system interaction and all other functionalities.

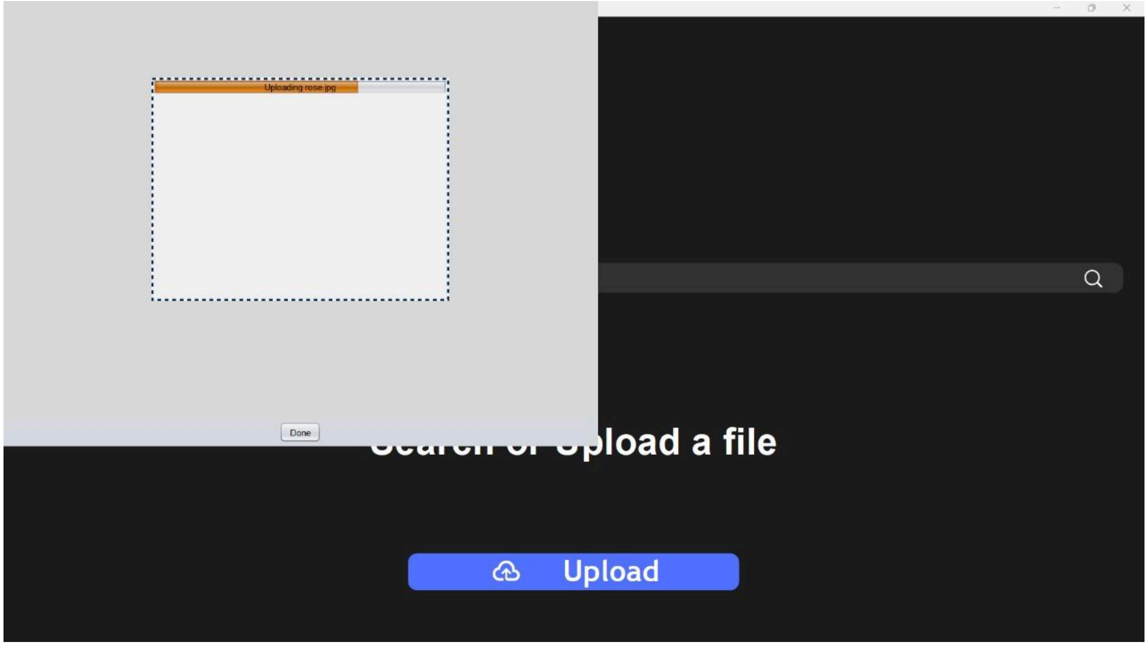
Thalleen: Worked on creating datanodes and helped with the frontend.

Screenshots:



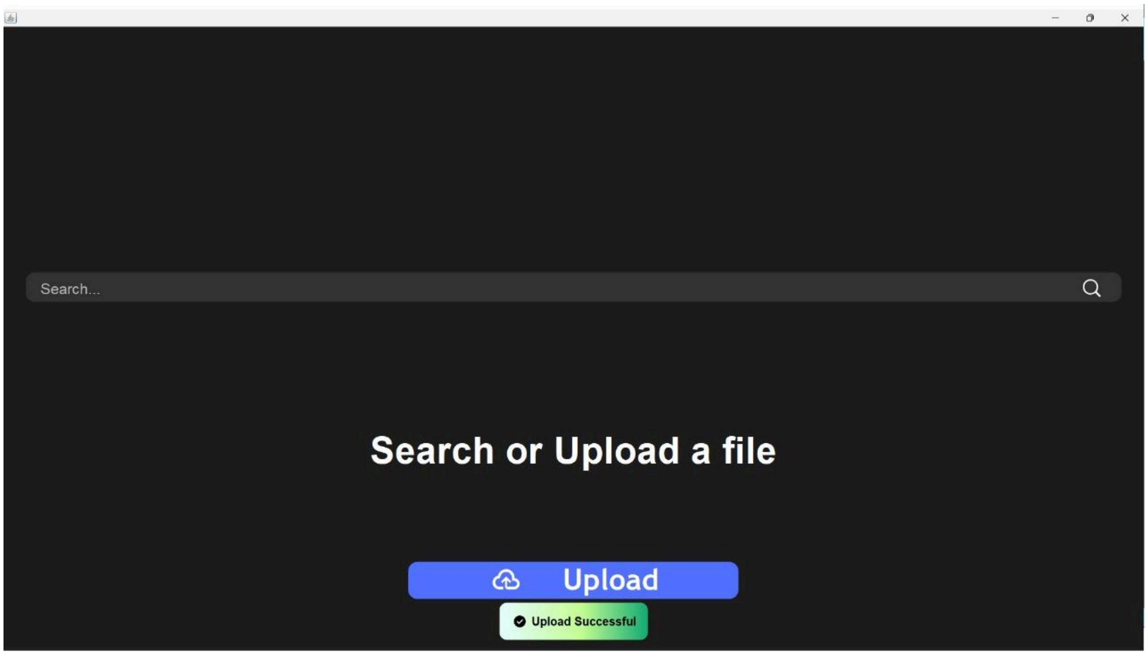
Search or Upload a file

home page



Uploading rose.jpg

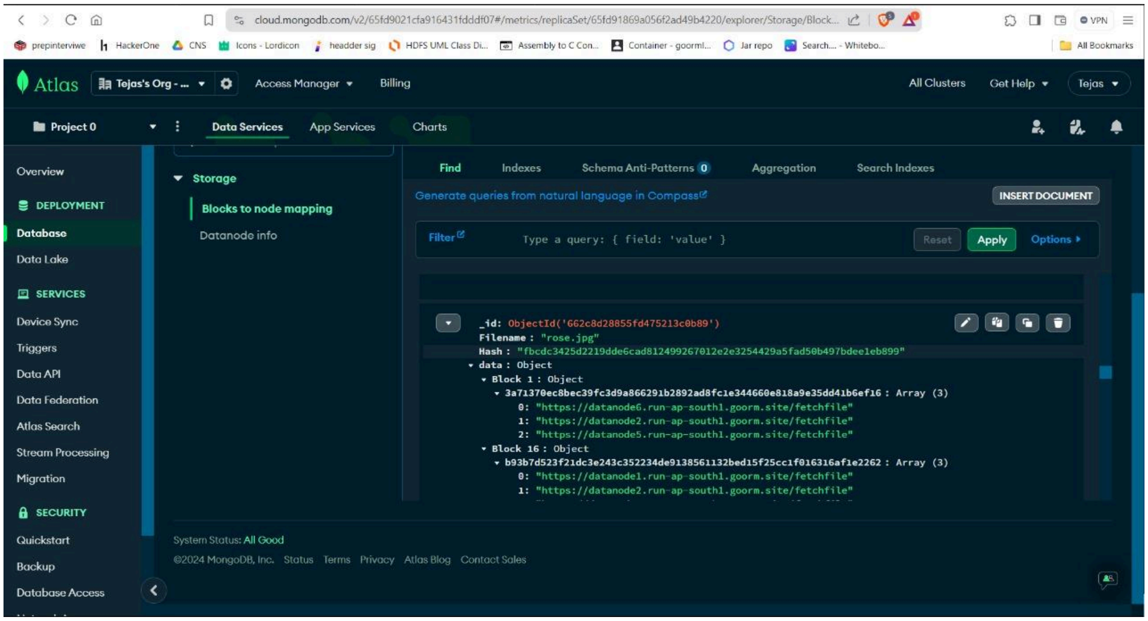
uploading



Search or Upload a file

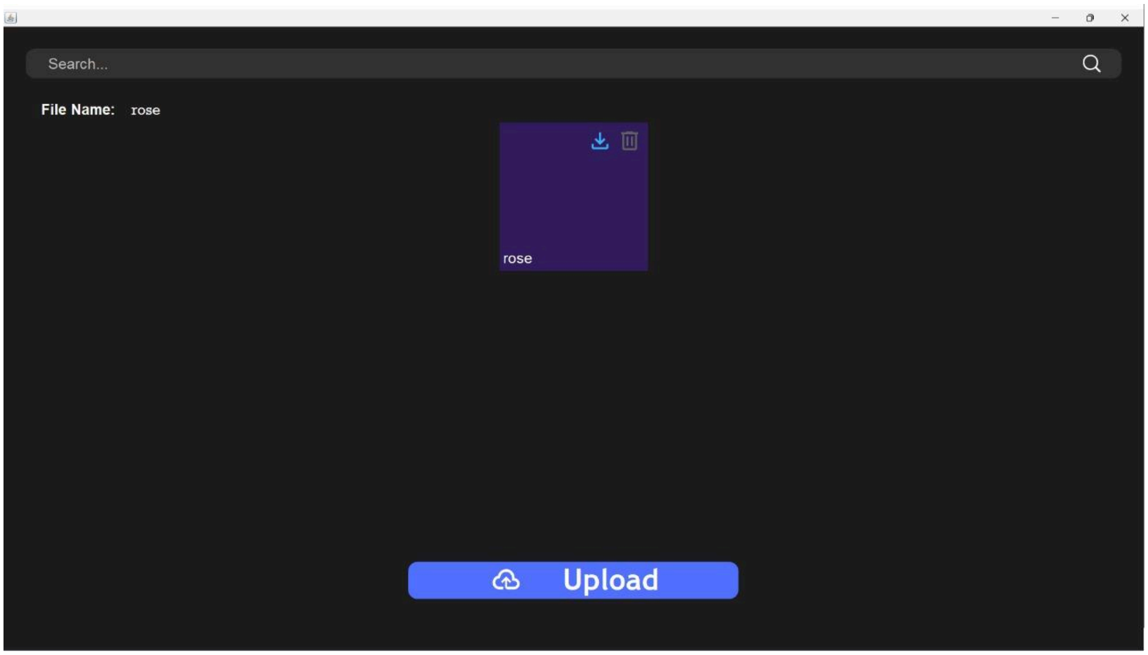
Upload Successful

uploaded



```
_id: ObjectId('662c8d28855fd475213c0b89')
Filename : "rose.jpg"
Hash : "fbcdc3425d2219dde6cad812499267012e2e3254429a5fad50b497bdee1eb899"
▾ data : Object
  ▾ Block 1 : Object
    ▾ 3a71370ec8bec39fc3d9a866291b2892ad8fc1e344660e818a9e35dd41b6ef16 : Array (3)
        0: "https://datanode6.run-ap-south1.goorm.site/fetchfile"
        1: "https://datanode2.run-ap-south1.goorm.site/fetchfile"
        2: "https://datanode5.run-ap-south1.goorm.site/fetchfile"
  ▾ Block 16 : Object
    ▾ b93b7d523f21dc3e243c352234de9138561132bed15f25cc1f016316af1e2262 : Array (3)
        0: "https://datanode1.run-ap-south1.goorm.site/fetchfile"
        1: "https://datanode2.run-ap-south1.goorm.site/fetchfile"
```

all the block to node mappings are stored in the database



File Name: rose

can view/download/delete the file