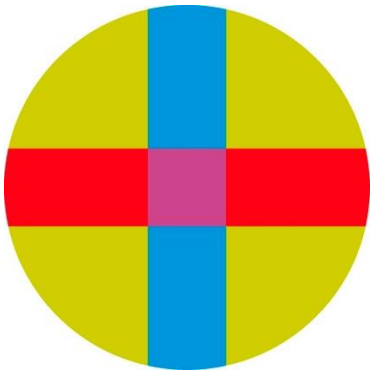
UNIVERSIDAD SAN PABLO - CEU

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA DE SISTEMAS DE INFORMACIÓN



TRABAJO FIN DE GRADO

Módulo de análisis BigData On-premises

On-premises BigData analysis module

Autor: Carlos Alonso Gradillas
Tutor: Jaechang Nam
Cotutor: Mónica Robledo de los Santos

Junio 2024

UNIVERSIDAD SAN PABLO-CEU
ESCUELA POLITÉCNICA SUPERIOR
División de Ingeniería

Calificación del Trabajo Fin de Grado

Datos del alumno

NOMBRE:

Datos del Trabajo

TÍTULO DEL PROYECTO:

Tribunal calificador

PRESIDENTE:	FDO.:
SECRETARIO:	FDO.:
VOCAL:	FDO.:

Reunido este tribunal el ___ /___ /_____, acuerda otorgar al Trabajo Fin de Grado presentado por D./Dña. _____ la calificación de _____

Resumen

El Proyecto Fin de Grado (TFG) se centra en el desarrollo de un prototipo o módulo independiente llamado UNIK para una aplicación existente denominada KUBIC (Korean Unification Big Data Center). El objetivo de KUBIC es recopilar, buscar y analizar datos de investigación profesional en Corea del Sur relacionados con la unificación de Korea. En su cuarto año, el Centro de Big Data sobre Unificación ha alcanzado la fase de funcionamiento estable y se esfuerza por automatizar y optimizar las tareas necesarias para el funcionamiento del centro, incluidas la recopilación, la gestión y el mantenimiento de los datos.

El objetivo de este proyecto, dirigido por el profesor, Jaechang Nam, es investigar y desarrollar un marco que mejore las capacidades de análisis de BigData e implemente un sistema de archivos distribuidos y una solución de almacenamiento más eficientes. Para lograrlo, exploramos la integración de Hadoop y Apache Spark, dos potentes tecnologías para el procesamiento y análisis de BigData. El objetivo principal era aprovechar HDFS de Hadoop para el almacenamiento escalable y las capacidades de procesamiento en memoria de Spark para mejorar el rendimiento del análisis de datos.

Además de mejorar la infraestructura existente, ampliamos la funcionalidad de KUBIC para permitir a los usuarios cargar archivos para su análisis por Spark. Esta función centrada en el usuario facilita el procesamiento de conjuntos de datos externos, lo que permite una capacidad de investigación más completa y flexible.

UNIK (Unificación de Corea) es un proyecto de un año de duración. Durante el primer semestre, el enfoque central fue en desarrollar el ecosistema Hadoop y crear una interfaz fácil de usar para cargar archivos en HDFS. Esto implicó la instalación del clúster Hadoop, la configuración de HDFS y la integración de Spark para el procesamiento avanzado de datos. También diseñamos e implementamos una interfaz de usuario (UI) basada en web para agilizar el proceso de carga de archivos, garantizando la facilidad de uso para investigadores y analistas.

Este proyecto pretende contribuir al avance y perfeccionamiento de la tecnología subyacente en KUBIC, proporcionando una solución robusta y avanzada para la gestión y análisis de datos relacionados con la unificación coreana.

Palabras Clave

- Análisis de Archivos: La incorporación de Spark en la interfaz de usuario permite a los usuarios seleccionar y ejecutar varios algoritmos (trabajos de PySpark) para el análisis de datos, proporcionando capacidades de procesamiento de datos flexibles y potentes.
- Angular: El principal marco de trabajo para el frontend utilizado en la aplicación, ofreciendo una interfaz de usuario robusta y dinámica.
- Apache Spark: Un sistema de computación distribuida de código abierto que ofrece una interfaz para programar clústeres completos con paralelismo de datos implícito y tolerancia a fallos. Es conocido por su velocidad, facilidad de uso y versatilidad en el manejo de diversas tareas de procesamiento de datos.
- Carga de Archivos: Permite a los usuarios cargar sus propios archivos para análisis, asegurando una experiencia interactiva y fácil de usar.
- Hadoop: Un marco de trabajo de código abierto utilizado para el almacenamiento y procesamiento distribuido de grandes conjuntos de datos en un clúster de computadoras. Está diseñado para escalar desde un solo servidor hasta miles de máquinas, cada una proporcionando capacidades de cómputo y almacenamiento locales.
- HDFS (Sistema de Archivos Distribuido de Hadoop): Un componente central de Hadoop, HDFS es un sistema de archivos distribuido diseñado para funcionar con hardware de consumo. Almacena datos en múltiples máquinas, replicando cada fragmento de datos en varios nodos para asegurar la confiabilidad y tolerancia a fallos.
- KUBIC (Centro de Big Data para la Unificación de Corea): Tiene como objetivo recopilar, buscar y analizar datos de investigación profesional en Corea del Sur relacionados con la unificación y Corea del Norte, proporcionando información y conocimientos valiosos.
- Módulo Independiente - UNIK (Unificación de Corea): Un prototipo diseñado para probar el ecosistema Hadoop e implementar la funcionalidad de carga de archivos en el sistema KUBIC, demostrando la aplicación práctica de estas tecnologías.
- Spring Boot: El lenguaje principal para desarrollar APIs en la aplicación, facilitando la creación de aplicaciones basadas en Spring autónomas y de grado de producción.
- YARN (Yet Another Resource Negotiator): Un componente clave del ecosistema de Hadoop, YARN es una capa de gestión de recursos que programa trabajos y gestiona los recursos del clúster. También facilita la integración de Apache Spark en el sistema Hadoop.

Abstract

The Final Degree Project focuses on developing a prototype or independent module called UNIK for an application named KUBIC (Korean Unification Big Data Center). KUBIC aims to collect, search, and analyze professional research data in South Korea related to Korean unification. In its fourth year, KUBIC has reached the stable operation phase and strives to automate and optimize the tasks necessary for the center's operation, including data collection, management, and maintenance.

The objective of this project, as directed by Professor Jaechang Nam, is to investigate and develop a framework that enhances big data analysis capabilities and implements a more efficient distributed file system and storage solution. To achieve this, the integration of Hadoop and Apache Spark, two powerful technologies for big data processing and analytics, was explored. The goal was to leverage Hadoop's HDFS for scalable storage and Spark's in-memory processing capabilities to improve data analysis performance.

In addition to enhancing the existing infrastructure, KUBIC's functionality was extended to allow users to upload files for analysis by Spark. This user-centric feature facilitates the processing of external data sets, enabling more comprehensive and flexible research capabilities.

UNIK (Unification of Korea) is a year-long project. During the first semester, the focus was on developing the Hadoop ecosystem and creating a user-friendly interface for file uploads to HDFS. This involved setting up the Hadoop cluster, configuring HDFS, and integrating Spark for advanced data processing. A web-based User Interface (UI) was also designed and implemented to streamline the file upload process, ensuring ease of use for researchers and analysts.

This project seeks to contribute to the advancement and refinement of the underlying technology in KUBIC, providing a robust and advanced solution for data management and analysis related to Korean unification.

Keywords

- Angular: The main frontend framework used in the application, offering a robust and dynamic user interface.
- Apache Spark: An open-source, distributed computing system that offers an interface for programming entire clusters with implicit data parallelism and fault tolerance. It is known for its speed, ease of use, and versatility in handling several types of data processing tasks.
- File Analysis: Incorporating Spark into the UI allows users to select and run various algorithms (PySpark jobs) for data analysis, providing flexible and powerful data processing capabilities.
- File Upload: Enabling users to upload their own files for analysis, ensuring a user-friendly and interactive experience.
- Hadoop: An open-source framework used for distributed storage and processing of large datasets across a cluster of computers. It is designed to scale from a single server to thousands of machines, each providing local computation and storage capabilities.
- HDFS (Hadoop Distributed File System): A core component of Hadoop, HDFS is a distributed file system designed to run on commodity hardware. It stores data across multiple machines, with each piece of data replicated across several nodes to ensure reliability and fault tolerance.
- Independent Module - UNIK (Unification of Korea): A prototype designed to test the Hadoop ecosystem and implement file upload functionality into the KUBIC system, demonstrating the practical application of these technologies.
- KUBIC (Korea Unification Big Data Center): Aims to collect, search, and analyze professional research data in South Korea related to unification and North Korea, providing valuable insights and information.
- Spring Boot: The primary language for developing APIs in the application, facilitating the creation of stand-alone, production-grade Spring-based applications.
- YARN (Yet Another Resource Negotiator): A key component of the Hadoop ecosystem, YARN is a resource management layer that schedules jobs and manages cluster resources. It also facilitates the integration of Apache Spark into the Hadoop system.

Table of contents

Figure references

Table references

Image references

Chapter 1

Introduction

1.1 Context of the TFG

During the 2023-2024 academic year, while on an exchange program in South Korea, the professor, Jaechang Nam, a member of KUBiC, introduced us to the KUBiC website and tasked us with researching new Big Data frameworks. With complete freedom in research, Hadoop was explored and subsequently proposed a new feature for the system: enabling users to upload their own files.

The development of this TFG consists of:

- User Interface
- Set up Hadoop environment into local computers.
- Create working APIS that connect UI to Hadoop and necessary databases like MongoDB.

The system developed to do such task is called UNIK.

1.2 Objectives

Given the tense situation between North Korea and South Korea, this project aims to provide essential resources for researchers and military personnel to access, analyze, and process Korean unification documents. By leveraging advanced big data technologies, our goal is to facilitate informed decision-making and strategic planning for stakeholders involved in the unification process.

The core objective of this project is to support and guide Korea towards a peaceful and prosperous unification. We believe that by making comprehensive and reliable data readily available, we can empower experts to derive actionable insights, understand historical and current contexts, and develop effective strategies to address the multifaceted challenges of unification.

The project's full development was intended to be completed within a year. During the first semester, a prototype was created, or independent application called UNIK, demonstrating how Hadoop could be integrated into KUBiC. The objective for the following semester is to properly incorporate this feature into the system.

1.3 What is Hadoop

Hadoop is a powerful open-source framework designed to store and process vast amounts of data efficiently using a cluster of commodity hardware. It consists of three core components specifically designed to manage big data:

- HDFS (Hadoop Distributed File System)

Storing massive amounts of data on a single computer is impractical. Instead, HDFS distributes the data across many computers, storing it in blocks. For example, if you have 600MB of data to store, HDFS splits the data into multiple blocks (with a default block size of 128MB) and stores them across several data nodes in the cluster. A key feature of HDFS is fault tolerance; if one node crashes, the data is not lost because HDFS replicates the data across multiple systems.

- MapReduce

After storing the data, it needs to be processed. MapReduce is a programming model that divides the data processing task into smaller parts, which are processed independently on different data nodes. The individual results are then aggregated to produce the final output.

- YARN (Yet Another Resource Negotiator)

Multiple jobs run simultaneously on Hadoop, each requiring resources like RAM and CPU to complete. YARN manages these resources effectively. It consists of the Resource Manager, Node Manager, Application Master, and Containers. The Resource Manager allocates resources, Node Managers handle nodes and monitor resource usage, and Containers hold collections of physical resources.

Hadoop also integrates with other components like Hive, Pig, and Apache Spark, providing a versatile ecosystem for big data processing.

In the project, it was decided to work with Apache Spark rather than MapReduce. Spark offers several advantages over MapReduce:

  - Speed: Spark processes data significantly faster than MapReduce by performing in-memory computations and optimizing execution plans.
  - Ease of Use: Spark provides a more user-friendly API and supports multiple programming languages (Java, Scala, Python, and R), making it accessible to a wider range of developers.
  - Advanced Analytics: Spark includes built-in libraries for advanced analytics, including machine learning (MLlib), graph processing (GraphX), and stream processing (Spark Streaming), which are not natively available in MapReduce.

- Flexibility: Spark can handle batch processing, real-time data processing, and interactive queries, providing a more versatile framework for various data processing needs.

These features make Spark a better choice for the project, enabling us to process large datasets more efficiently and effectively.

1.4 Work organization

Members if this project:

John Song:

Tasks:

- Research on storage.
- Setting up Hadoop ecosystem into Lab's computers.
- Setting up Spark into Hadoop.
- Monitoring and maintenance of Hadoop.
- Creating controller to connect to HDFS and Spark through Spring Boot.
- Developed the Front end for file upload to access HDFS.

Siaw Jia Yuin (Dana):

Tasks:

- Research on BigData.
- Setting up Spark into Hadoop.
- Research BigData algorithms.
- Run Spark jobs on clusters.
- Created PySpark algorithms to run on Hadoop ecosystem.
- Documenting progress.

Carlos Alonso:

Tasks:

- Research file distribution across clusters, and frameworks.
- Setting up computers with Ubuntu, and SSH to have communication between one another.
- Setting up Hadoop ecosystems into Lab's computers.
- Creating controller to connect to MongoDB through Spring Boot.
- Creating Frontend template and routing between pages, focused on the connection between MongoDB files and visualizing the files on the UI.

Chapter 2

Project management

2.1 Life cycle model

Given that this was a team project with the professor serving as overseer, the vision and goals had to be properly defined and their implementation into KUBIC clearly outlined. A waterfall life cycle model was chosen for this purpose. This model provided the necessary structure while allowing flexibility to revisit and refine phases as new areas were discovered or new implementations were required.

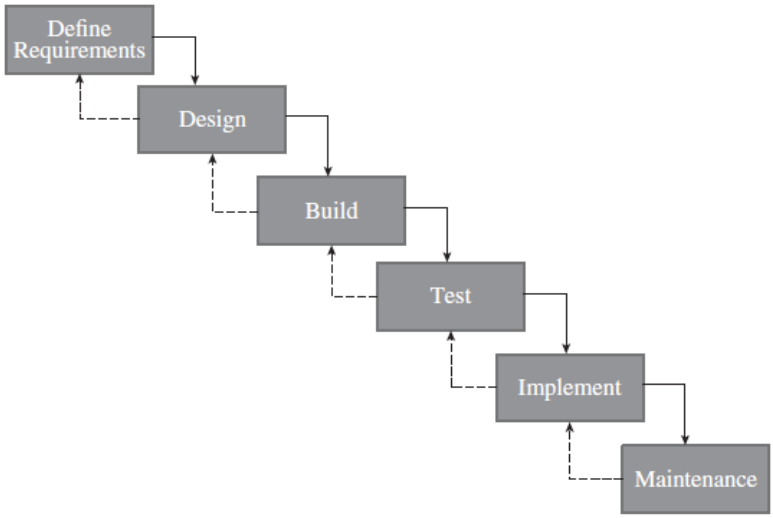


Figure 1. Waterfall lifecycle model

Requirement definition

The phase with the most dedication was undoubtedly the requirement definition. During this phase, the professor introduced us to KUBiC and its goal to incorporate a cloud-based cluster module. Given the brief time for implementation within one semester and the insufficient time to investigate a cloud-based environment and big data analysis module, it was decided to prioritize the latter and consider cloud integration for future development.

This phase was divided into two types of definitions:

- Research Requirements: Determining the main theme of the project.
- System Requirements: Once the project theme was decided, designing how the system should function.

This phase was frequently revisited for new implementations and necessary research.

Design

In this phase of development, two types of design decisions were made:

- Hadoop Environment Design: Deciding on the number of clusters, additional Hadoop services to incorporate, the number of Data Nodes, etc.
- System Design: Designing the APIs, Frontend system, user interface sketches, and more.

With the guidance and constant feedback from the professor, the optimal direction for the work was determined.

Build

During the building phase, there were two IDEs for developing the system:

- Frontend: KUBiC's front-end system was designed in Angular. Given the opportunity to learn a new language and environment, it was decided to embrace and incorporate Angular into UNIK's front end. Development was constructed using Visual Studio Code.
- Backend: Spring Boot was chosen for the backend development. This was the best-known framework among the project members, aiding in connecting various applications together. Development was constructed using IntelliJ IDEA.

Test

The focus was on building a well-structured controller to connect to MongoDB and Hadoop, utilizing PostMan to test the backend HTTP requests to the server. This approach ensured the proper functioning of backend operations and facilitated the efficient identification and resolution of any issues.

Implement

This phase was mainly documentation and preparation to incorporate UNIK into KUBiCs system, guiding future developers on how to properly deploy it to a production environment.

Maintenance

After implementation, the system entered the maintenance phase. This involves regular updates, bug fixes, and enhancements to ensure the system continues to meet user needs and adapt to any changes in the environment or requirements.

While the systems maintenance was a big part of the groups focus, constant problems with the Hadoop clusters occurred, time and effort were invested into solving these problems.

While having the Waterfall life cycle model as a base for development and as the guiding framework, several aspects of Agile development were incorporated to enhance the process. This hybrid approach allowed us to maintain structured phases while benefiting from the flexibility and responsiveness of Agile practices.

Some key aspects of Agile development that were incorporated:

- Weekly meetings were established to assess progress, ensuring alignment with goals and allowing for the quick addressing of any issues that occurred.
- Constant feedback on the development was crucial to know the project was heading in the correct direction.
- Rather than waiting until the end of the project to deliver a complete product, the development was broken down into smaller iterations. Each iteration resulted in a working increment of the project, which testing, reviewing and improvement always came into play.
- Tasks were prioritized based on their importance on the overall project. This prioritization ensured a focus on delivering the most critical features and functionalities first, providing value early in the process.
- Comprehensive documentation of the development process, decisions, and progress was maintained. This transparency helped in tracking progress and provided a clear record for future reference.

2.2 Planification

The planification of this project had to be very precise and had to be followed accordingly to due dates and time constraints, a GANTT was designed to help us map out the whole scope of the project:

	Nombre	Duracion	Inicio	Terminado
1	**research**	21 days	**1/02/24 8:00**	**29/02/24 17:00**
2	learning feild	20 days	1/02/24 8:00	28/02/24 17:00
3	research big data framework	10 days	12/02/24 8:00	23/02/24 17:00
4	define requiremnets	4 days	26/02/24 8:00	29/02/24 17:00
5	**implementing frameworks/techno**	30 days	**5/03/24 8:00**	**15/04/24 17:00**
6	hadoop implementation	10 days	5/03/24 8:00	18/03/24 17:00
7	set up hadoop environment	20 days	19/03/24 8:00	15/04/24 17:00
8	**Mongo controller**	24 days	**25/04/24 8:00**	**28/05/24 17:00**
9	design	5 days	25/04/24 12:00	2/05/24 13:00
10	implementation	15 days	29/04/24 8:00	17/05/24 17:00
11	testing backend	24 days	25/04/24 8:00	28/05/24 17:00
12	**HDFS controller**	25 days	**29/04/24 8:00**	**31/05/24 17:00**
13	design	20 days	29/04/24 8:00	24/05/24 17:00
14	implementation	3 days	29/04/24 8:00	1/05/24 17:00
15	testing backend	25 days	29/04/24 8:00	31/05/24 17:00
16	**Frontend design**	40 days	**5/04/24 13:00**	**31/05/24 13:00**
17	design	7 days	8/04/24 8:00	16/04/24 17:00
18	development	40 days	5/04/24 13:00	31/05/24 13:00
19	testing	20 days	12/04/24 8:00	9/05/24 17:00
20	documentation	92 days	1/02/24 8:00	7/06/24 17:00

Figure 2. GANTT diagram

Description of each phase:

- Phase 1. Research:

Intensive research was conducted to understand the professor's requirements and KUBiC's working system. Decisions were made regarding the selection of the development language, the Big Data analysis frameworks to be studied, and any technical knowledge that needed to be acquired by the team members.

- Phase 2. Hadoop implementation:

After selecting the framework, further research was conducted to properly set up the Hadoop environment on local computers. This included configuring Hadoop and setting up Ubuntu as the main working environment, as well as connecting the computers via SSH.

- Phase 3. Mongo controller:

Once the API was selected(Spring boot), MongoDB was the first objective, as it served as a learning phase to correctly understand the functionality of an API and then have a better understanding to connect a HDFS controller to the Hadoop environment, designing this controller gave us the fundamentals of HTTP request and how to handle them.

- Phase 4. HDFS controller:

Following the completion of the Mongo controller, a new controller was created. This controller was designed similarly to the MongoDB controller but included additional configuration and requirements.

- Phase 5. Front end Design:

Once the system design was finalized, the focus shifted to designing and developing the frontend system.

2.3 Execution

Due to the extensive scope of the project, minor setbacks in production and research were encountered. Various simplified deployment methods, including Docker, were explored to streamline system usage, but the focus remained on the primary objective. Throughout the project, guidance from the professor was consistent through weekly meetings, seminars, and mentoring sessions, ensuring the development stayed on track.

With the assistance of a well-defined schedule and thorough research, adaptations to risks were made promptly, and deadlines were met. While the project remains under development and is not yet fully completed, an "end of project" delivery is scheduled for the end of August.

Chapter 3

Analysis

3.1 Requirement specification

After understanding the professor's requirements, a system requirement table was developed to better describe and understand the projects necessities, here is the system requirement table:

Index	System Requirements
1	*User Interface*
1.1	User interface should be straightforward and functional
1.2	UI elements should maintain constant appearance throughout the application to reduce user confusion
1.3	User should be able to select as many files as they like
1.4	Maximum file size for uploads is 100MB
1.5	UI components should accurately direct users to the intended destination or page
1.6	Analysis results should be accurate
2	*Controller*
2.1	Controller should be able to access into MongoDB database without error
2.2	Controller should be able to access into HDFS without error
2.3	Controller should be able to utilize Spark to analyze data of selected files
2.4	Controller should use appropriate APIs for system integration
2.5	Controller should be able to correctly handle file upload requests from users

3	***Hadoop Distributed File System (HDFS)***
3.1	System should facilitate multiple machines for file storage
3.2	System should be able to handle files of different formats
3.3	The hardware that contains the DataNode should run Ubuntu 22.04
3.4	The hardware that contains the NameNode should run Ubuntu 22.04
3.5	The YARN Resource Manager should be in the MasterNode for data transactions
3.6	System should have several machines each with a NodeManager
4	***Spark***
4.1	Spark should be able to handle job requests from users
4.2	Accurate results of the analysis should be produced
4.3	Spark clusters should be in the machine which contains the NameNode
4.4	Spark should be able to access all data and process in parallel
4.5	Spark should be well integrated with HDFS and Yarn to support fault tolerance
5	***Operations Requirements***
5.1	System should have at least three machines running
5.2	All three machines should have acces to each other via SSH
5.3	The machines should have sufficient storage capacity
5.4	NameNode should have at least 100GB of SSD
5.5	NameNode should have at least 8GB of RAM
5.6	DataNode should have at least a two core CPU
5.7	DataNode should have at least 4GB of RAM

Table 1. System Requirements Table

3.2 Use case diagram

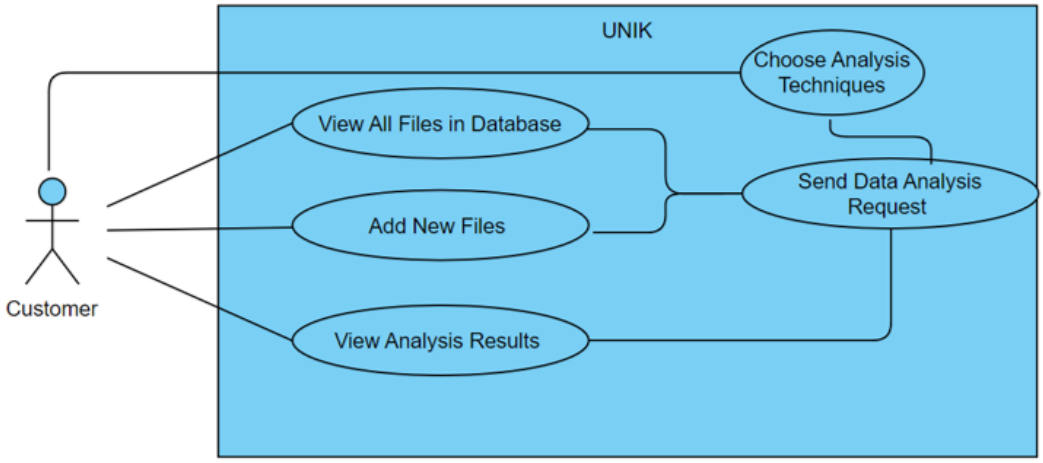


Figure 3. Example of a Use case diagram

The use case diagram provides an overview of the interactions between the user (Customer) and the UNIK module, highlighting the different functionalities offered.

The components and interactions are as follows:

- Customer: Represents the user interacting with the system.
- UNIK: The system/module that connects MongoDB with HDFS and Spark, providing a frontend for user interactions.

The main use cases include:

- View All Files in Database: The customer can view all the files stored in the MongoDB database.
- Add New Files: The customer can upload new files to the MongoDB database from their device.
- Choose Analysis Techniques: The customer selects an analysis technique to be used on the chosen data.
- Send Data Analysis Request: After choosing the analysis technique, the customer sends a request for data analysis.
- View Analysis Results: Once the analysis is completed using Spark, the results are displayed to the customer.

3.3 Security analysis

The security analysis was created to properly secure the users data, as it was mentioned before, the user's access is not going to be handled by the UNIK members as it is already controlled in KUBiC. The main security feature and for copyright issues, the files available only represent the metadata of the files, no file has the option to view it, unless it is your own uploaded file. With the taken into consideration, here is a more detailed explanation into the security analysis described in the project.

- Confidentiality:

Make sure each user has access to only their uploaded files, not other files uploaded by other users.

Use KUBiC to manage user authentication and ensure that access tokens and session cookies are securely handled.

- Integrity:

Make sure the data is not altered.

CORS (Cross-Origin Resource Sharing): Configure CORS policies to allow only trusted domains to interact with the backend.

Logging and Monitoring: Implement comprehensive logging to track data access and modifications. Use tools like Elasticsearch for monitoring and alerting on suspicious activities.

- Availability

Make sure the website is ready to use by all users and keep the files secure.

Redundancy and Backup: Implement redundancy and regular backups to ensure data availability in case of system failures.

Chapter 4

Design and implementation

4.1 System architecture

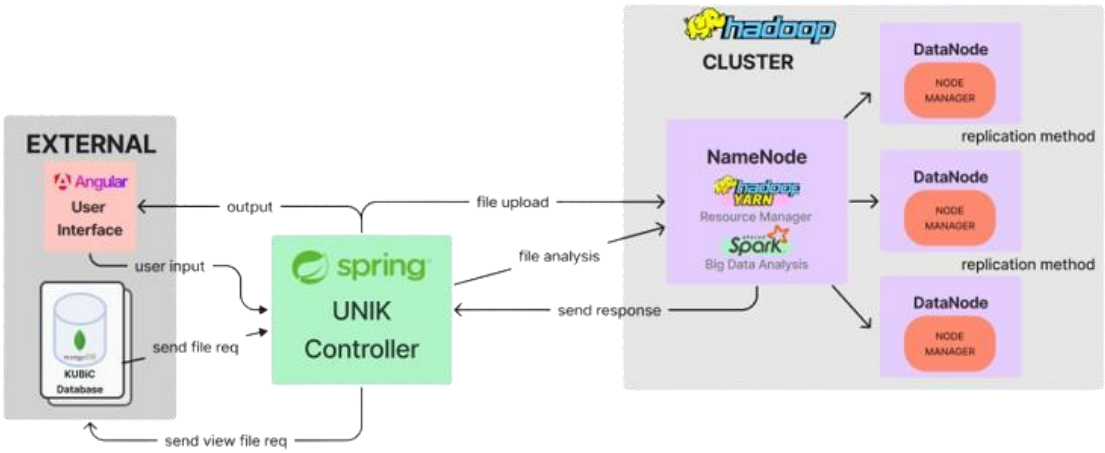


Figure 4. System architecture

The architecture depicted in the systems architecture outlines a data processing system that integrates multiple components, including an Angular-based user interface, a Spring-based controller, and a Hadoop cluster with YARN and Spark for resource management and data analysis.

Components:

- Angular User Interface:
  - Role: Acts as the frontend for the system, allowing users to interact with the application.
  - Function: Collects user input and sends requests to the backend (UNIK Controller). It also displays the output received from the backend.
- KUBiC Database (personal Database):

  As for now, KUBiCs data was not accessed, but it needed to have similar structure for an easier future migration.
  - Role: Stores data that the system might need to process.
  - Function: Provides data upon user request and stores analysis results.
- UNIK Controller (Spring Framework):
  - Role: Serves as the backend controller, handling requests from the user interface.
  - Function: Manages the interaction between the user interface and the Hadoop cluster. It receives user input, processes it, sends files for analysis, and returns the results to the user interface.
- Hadoop Cluster:
  - Components:
    - NameNode: The master node responsible for managing the filesystem namespace and regulating access to files by clients.
    - Data Nodes: Worker nodes that store data and perform computations. Data is replicated across multiple Data Nodes to ensure reliability and fault tolerance.
  - Role: Handles large-scale data storage and processing.
  - Function: Uses YARN (Yet Another Resource Negotiator) for resource management and scheduling tasks, and Spark for big data analysis.

Workflow:

User Input:

- The user interacts with the Angular User Interface to provide input for file selection or request data analysis .
- The user can also upload files via the interface.

Request Handling:

- The user input is sent to the UNIK Controller.
- If a file upload is requested, the file is sent to the Hadoop cluster via the UNIK Controller.

File Upload and Storage:

- The UNIK Controller uploads the file to the Hadoop cluster.
- The NameNode receives the file and stores it across multiple Data Nodes using a replication method to ensure data reliability.

Data Processing and Analysis:

- The UNIK Controller can initiate a data analysis job using Spark on the Hadoop cluster.
- Spark processes the data, leveraging the distributed storage and computation capabilities of Hadoop.

Result Handling:

- The results of the analysis are sent back to the UNIK Controller.
- The UNIK Controller sends the processed data back to the Angular User Interface.
- The Angular User Interface displays the results to the user.

The workflow can also be understood by this sequence diagram:

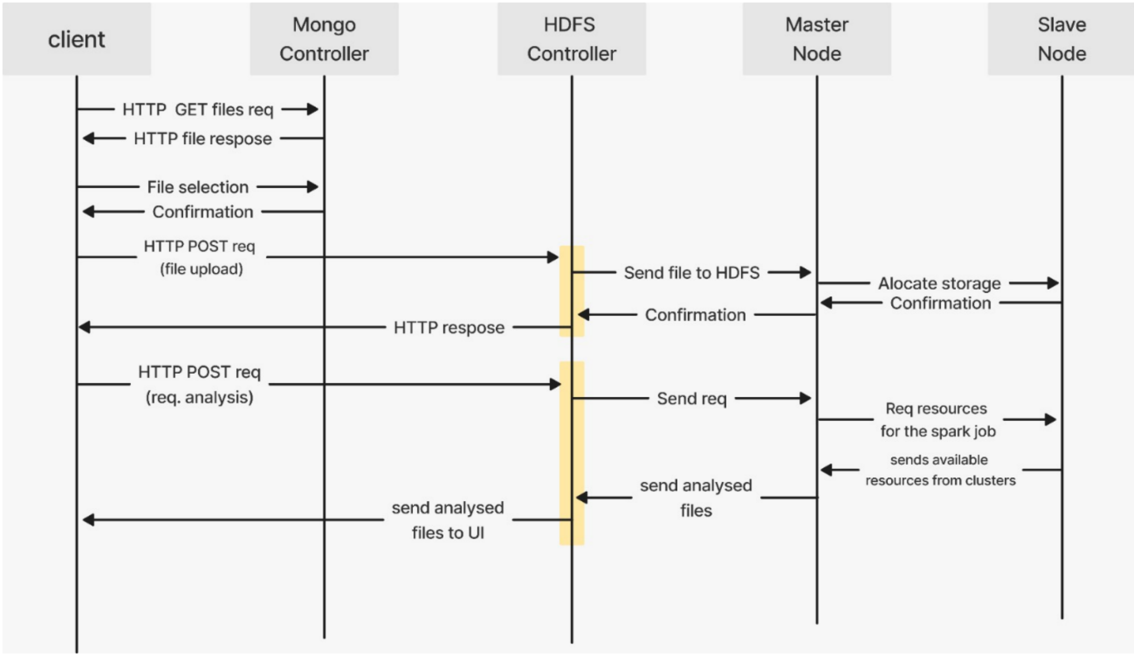


Figure 5. Sequence diagram of user request

To handle user request the system can go through different type of states, represented in this graph:

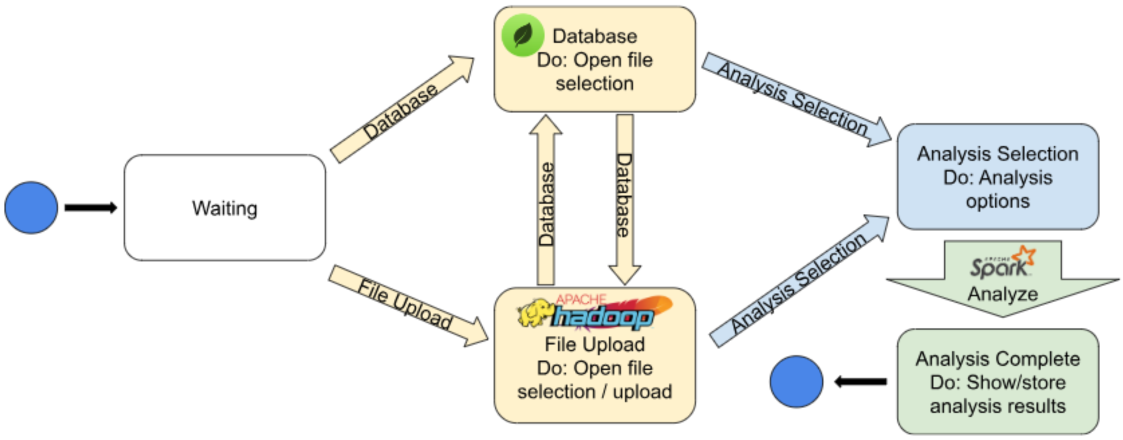


Figure 6. Logical view architecture

4.1.1 Physical architecture

To represent the physical architecture of Hadoop it can be better understood by the following picture.

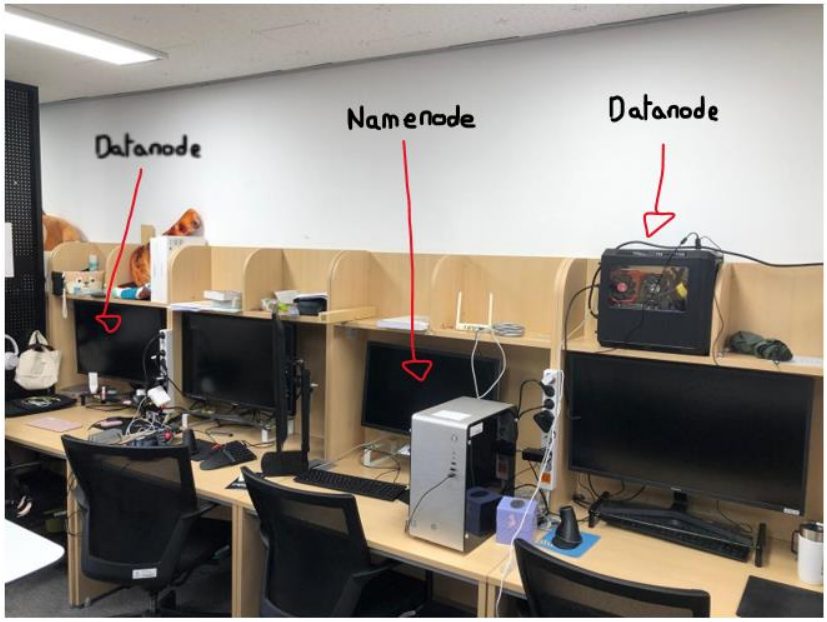


Image 1. Picture representing Hadoop physical architecture

Personal laptops were used to connect to the clusters and through the HDFS controllers, sending HTTP request for file analysis.

(For Hadoop and Spark configuration you can view in the ANNEX.)

The **NameNode** is the master node in Hadoop's HDFS its main function is to manage the filesystem namespace and it controls access to files by clients.

Other essential functions:

- Namespace Management: Maintains the directory tree of all files in the file system and tracks where across the cluster the file data is kept.
- Metadata Storage: Stores metadata about the file system, such as file names, permissions, and locations of data blocks.

- Operations Coordination: Coordinates file system operations such as opening, closing, and renaming files and directories.
- Health Monitoring: Monitors the health of Data Nodes and orchestrates the replication of data blocks for fault tolerance.

On the other hand, the **DataNode** are the worker nodes in HDFS. They store and retrieve blocks of data as instructed by the NameNode. They also handle block creation, deletion, and replication based on the Name Node's instructions.

Main functions:

- Heartbeat Signals: Data Nodes send regular heartbeat signals to the NameNode to indicate that they are operational. If the NameNode does not receive a heartbeat from a DataNode within a specified interval, it marks the DataNode as dead and triggers the replication of its blocks to other Data Nodes to maintain the desired replication factor.
- Block Storage: Data Nodes store the actual data blocks of the files. When a file is uploaded to HDFS, it is split into smaller blocks (default 128 MB), and these blocks are stored across various Data Nodes.
- Replication: Each block is replicated on multiple Data Nodes (default replication factor is 3). This ensures data redundancy and fault tolerance.

4.1.2 Logical view architecture

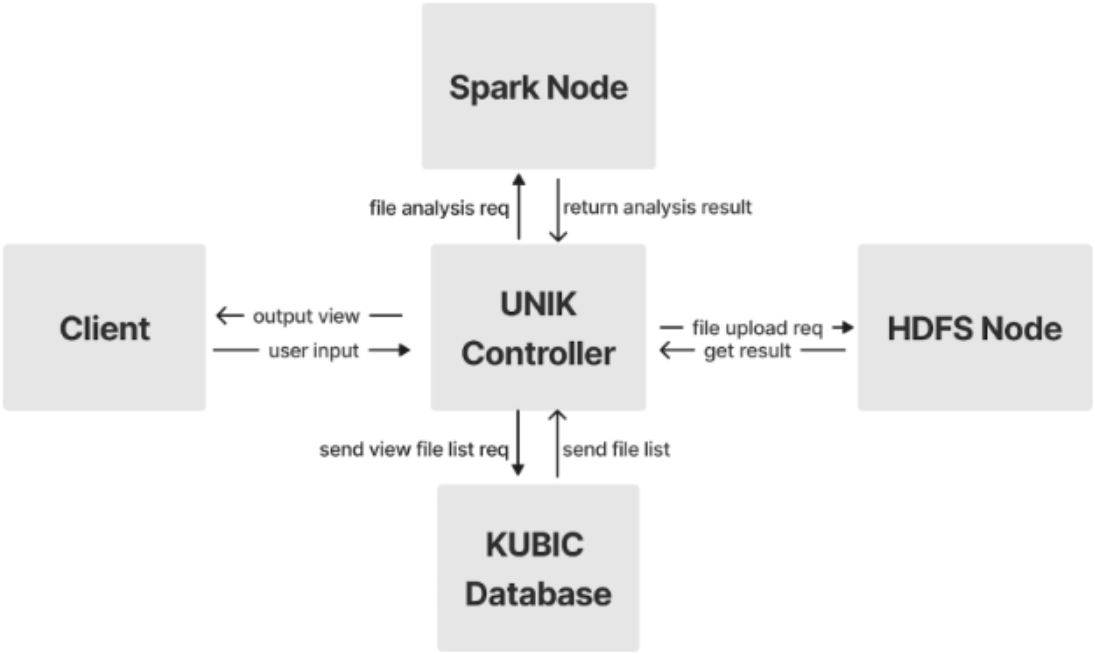


Figure 7. Logical view architecture

4.1.3 Level 2 infrastructure diagram

The development of the system is not currently available to outside users. As of now, the clusters are connected via Ethernet cable to the lab's routers, allowing access only to people connected to that Wi-Fi network. For future development, the main plan is to migrate this to the university's servers to enable public use of Hadoop.

4.2 Data design

Before the explanation of the data design it is important to have in mind that this is a personal database, the credentials are private and only the lab members have access to this database, for security and privacy reasons.

Connection to MongoDB were made through MongoDB Compass. MongoDB Compass is a graphical user interface for MongoDB that allows users to interact with their MongoDB databases visually. It provides a user-friendly way to perform tasks like exploring your data, running queries, visualizing data distributions, and performing CRUD (Create, Read, Update, Delete) operations without writing complex shell commands.

To connect to a private MongoDB instance using Compass, you will typically need the following details:

- Database name.
- Database password.
- Cluster name.

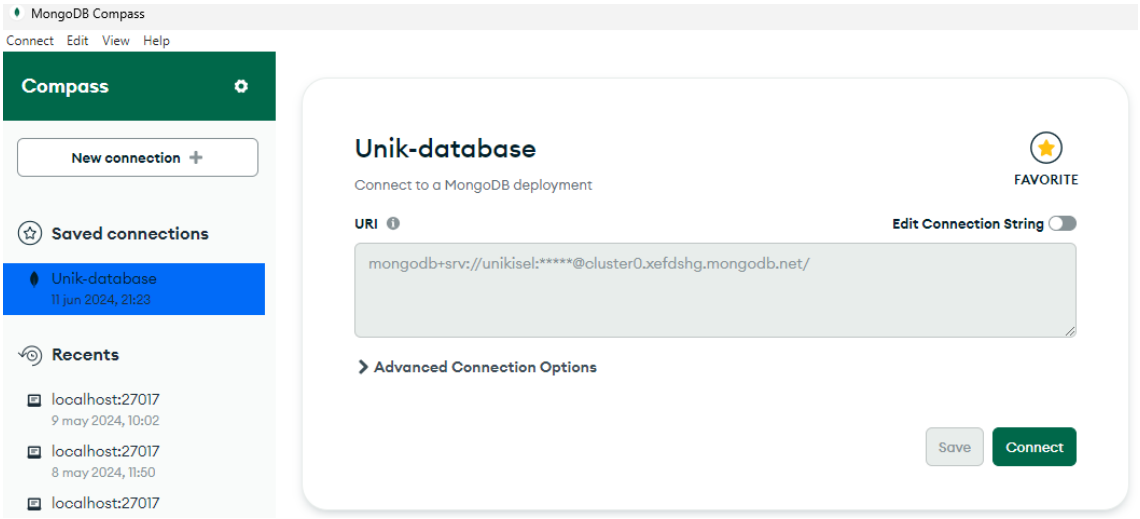


Image 2. Accessing MongoDB via Compass

The database structure:

In the following images you can view how the collections are designed

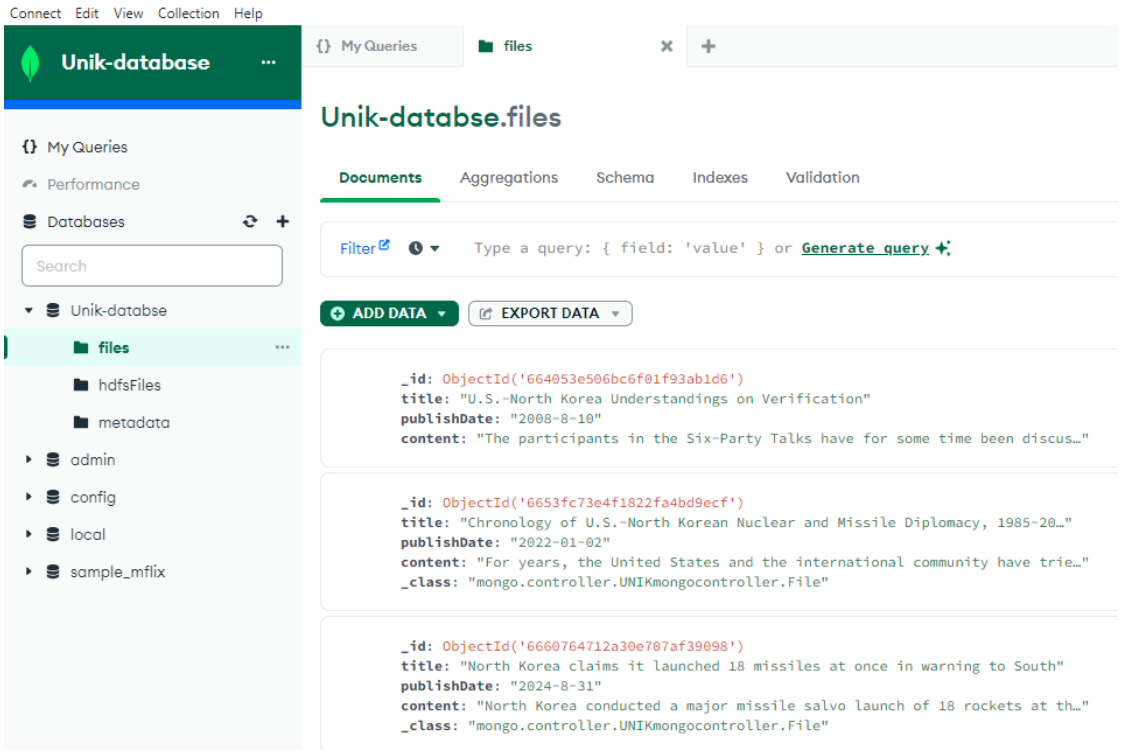


Image 3. File view example

Files contain examples of files with no metadata, primarily used for visualizing titles on the frontend and learn the connectivity through the mongo controller.

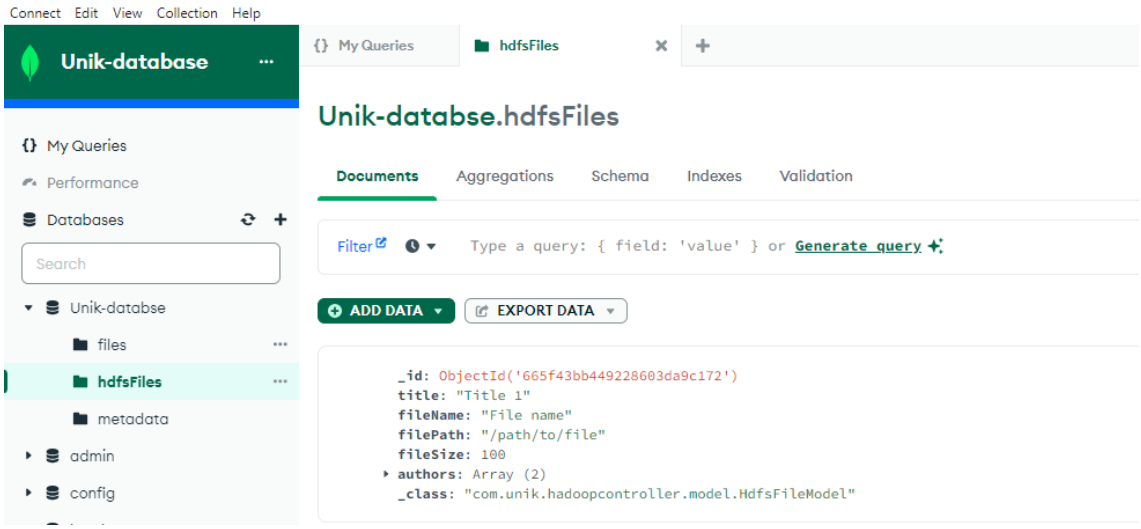


Image 4. HDFS data structure

HDFSFiles has been designed to properly adjust to the HDFS requirements and have a correct access to the Hadoop environment, adding to the collection the files metadata to be properly analyzed.

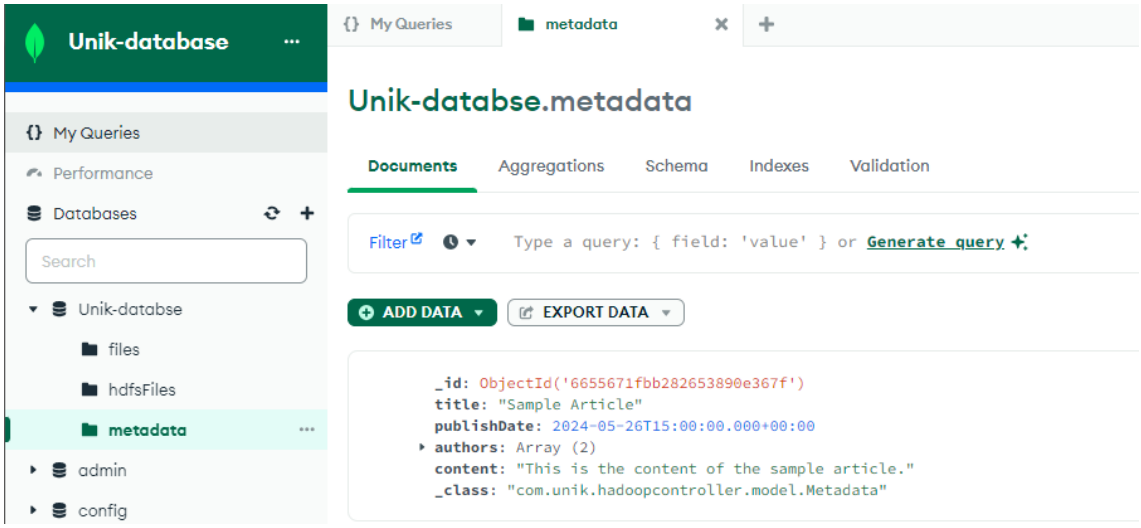


Image 5. File metadata

To the MongoDB collection metadata was added to the existing files so they can also get analyzed, having easier data transfer to the Hadoop and Spark analysis environment.

4.3 User interface design

The main purpose was not to design a user login page because, when this app is added to KUBiC, user authentication will be handled there.

The first page is the welcome page to UNIK. This page contains a welcoming message and three buttons: "Upload Your Files," "View Files," and "View Analysis."

- The "Upload Your Files" button redirects the user to the third page: "Upload Your Files" page.
- The "View Files" button redirects the user to the second page: "All Files in MongoDB" page.
- The "View Analysis" button redirects the user to the fourth page: "Select Analysis Technique" page.

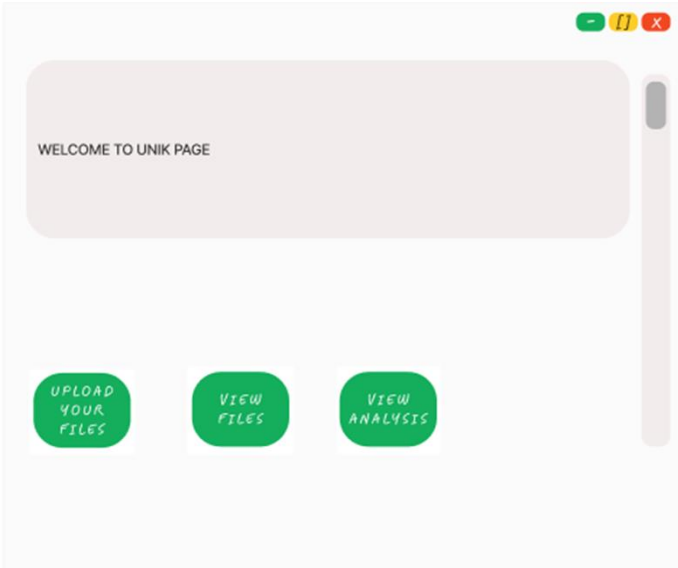


Image 6. Template for Welcome page

Next page represents the files available from MongoDB. The user can freely select available files that they would like to analyze and then move on to the next page, upload your files.

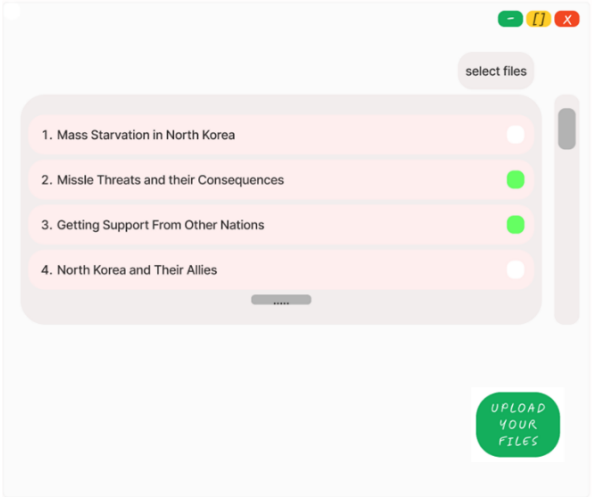


Image 7. Template for MongoDB / available files page

As mentioned before here the user can upload any file they want to get analyzed, the user can freely move between pages to select more files.

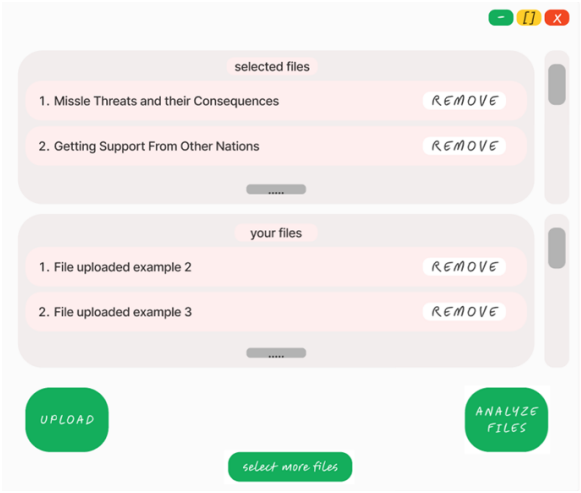


Image 8. Template for HDFS / upload files page

Lastly the user can select an algorithm that they would like to perform on their files, having the chance to select multiple analysis. These requests are handled and sent to Hadoop and Spark.

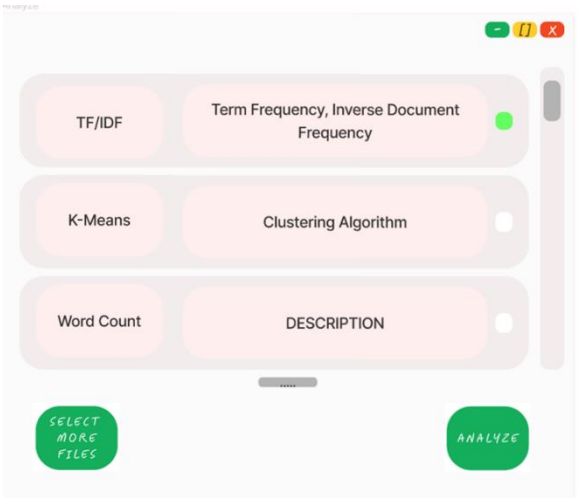


Image 9. Template for analysis page

Once the files have been analyzed, the user gets the result.

Image 10. Template for the result page

4.4 Class diagram

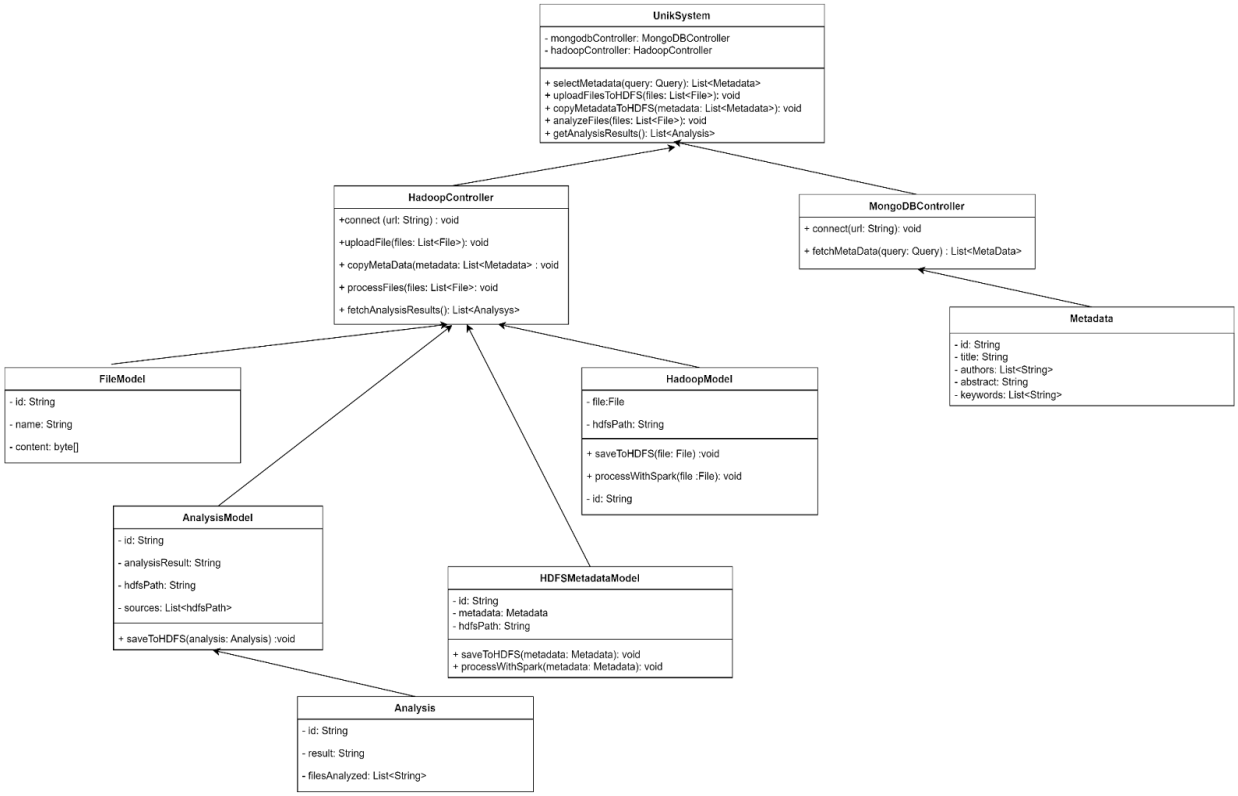


Figure 8. Class diagram

As previously discussed, the functionalities of the system have been thoroughly defined. The class diagram provides a structural overview of the system, highlighting the classes, attributes, and relationships within it. This diagram is crucial for understanding the architecture and the interactions between different components.

From the UNIK system there are two main classes HadoopController and MongoDBController each one with the required necessities described in the architecture.

4.5 Construction environment

While working on a group project it was important to know which task, objective and risk had to be prioritized.

These are the tools that helped us monitor and develop the project:

- Monitoring Tasks and Workflow: **Notion and Clickup**

Notion is an all-in-one workspace tool that allows you to manage tasks, projects, and documentation efficiently. It is highly customizable and can be used for a variety of purposes such as note-taking, project management, task tracking, and collaboration.

It was an essential tool to monitor tasks and ensuring that all team members were on the same page and that project milestones were tracked and met.

- Version Control: **GitHub**

GitHub is a web-based platform for version control and collaboration that uses Git. It allows multiple people to work on projects simultaneously without interfering with each other's work. It was mainly used to allow the team to collaborate on code development, track changes, and manage the codebase efficiently.

- Development Tools (IDE):
    - **Visual Studio Code (VS Code) with Angular**:

VS Code is a well-known source code editor developed by Microsoft. VS Code was used for developing the Angular-based user interface, providing a robust environment for writing, and debugging TypeScript and HTML and CSS.

    - **IntelliJ IDEA with Spring Boot:**

IntelliJ IDEA is a comprehensive IDE developed by JetBrains, known for its powerful features and support for a variety of programming languages, including Java and Spring Boot.

- Testing the Backend: **Postman**

Postman is a popular API testing tool that allows developers to test API endpoints, inspect responses, and automate testing.

Request Building: Create and send HTTP requests with various methods (GET, POST, PUT, DELETE).

Automated Testing: Write test scripts to automate API testing.

- Building Tools: **Maven**

Maven is a build automation tool primarily used for Java projects. It helps manage project dependencies, build processes, and project documentation.

- Database: **MongoDB**

MongoDB is a NoSQL database that uses a document-oriented data model. It is known for its scalability, flexibility, and performance. Documents (data records) can have varying structures, making it easy to store complex data.

Hadoop environment construction in the ANNEX

4.6 Reference to the software repository

The Projects GitHub repository:

https://github.com/carlosalongra/Aplicacion_TFG

Other repositories that may be useful:

KUBiCs repository: https://github.com/KUBIC-HGU

KUBiCs website: kubic.handong.edu

Note: it requires a VPN connection to South Korea to access the website.

Chapter 5

System validation

5.1 Test plan

A test plan was developed to make sure the requirements preestablished were properly fulfilled, and with the correct functionality.

The test established are:

- Unitary testing

  Main function is to verify that each individual components works as intended.

  - Angular: test each component, service and routing of the application
    - Tested via Karma, using command **ng test**
  - Spring boot: test the connection of the controllers, make the proper file class in Java to match the one on the Mongo database to have correct acces to it.
    - Tested via PostMan for each CRUD operations on existing files.

- Integration testing

  Ensure that different modules or services interact correctly.

  - Angular: test the output when connected to the spring controller, variables located on each service with the assigned IP address.
    - Tested via command ng e2e

- Spring boot: checked connectivity to other platforms.
  - Tested via PostMan to check connectivity to the desired IP address, in this case, each individual cluster and MongoDB, the IP addresses are located in the following file:

HadoopController/blob/master/src/main/resources/application.properties

- End to end testing (E2E)

Tested the connection from the user interface to correctly access Hadoop. Also created multiple test of deleting, adding and getting files form post mas and seeing the output on the front end.

On the other hand, Hadoop testing was primarily done on the command line from the Name node, uploading files to HDFS or submitting them to Spark with the following command:

- spark-submit --master yarn --deploy-mode cluster ./pi.py 3
  - check if its uploaded: hdfs dfs -ls /user/hadoop/document_samples

Chapter 6

Conclusions and future works

6.1 Conclusions

First and foremost, this project has had a significant personal impact on me. Having the rare opportunity to spend the final year of my college degree in South Korea has been a true gift. Additionally, being accepted into a research lab to develop such a project has been an incredibly rewarding experience. I would like to express my absolute appreciation and thankfulness to Professor Nam Jaechang, who provided me with this opportunity and guided me throughout the process.

Moreover, having such dedicated, hardworking, and invested lab members has made this experience even more fulfilling. Their support and collaboration have been invaluable.

On the technical side, this project, though challenging at times, has significantly expanded my knowledge in various areas:

- Teamwork: Working closely with my peers has taught me the importance of effective communication, collaboration, and the ability to work harmoniously in a team setting.
- Code Development: I have improved my coding skills, learning new programming languages and frameworks such as Angular and Spring Boot, and applying best practices in software development.
- Software Engineering Methodologies: This project has provided practical experience in applying software engineering methodologies, including testing, monitoring, project management, and more.

- Project Planning: I have gained a deeper understanding of project planning and management, including defining requirements, designing systems, building and testing solutions, and ensuring timely delivery.
- Research Lab Skills: Working in a research lab environment has taught me essential research skills, such as conducting thorough literature reviews, designing and conducting experiments, analyzing data, and documenting findings. I have also learned the importance of adhering to research ethics and maintaining meticulous records of all research activities.

Overall, this project has been a transformative experience, enhancing both my technical abilities and personal growth. I am grateful for the opportunity to work in such an enriching environment and look forward to applying these skills and experiences in my future endeavors.

6.2 Future works

The future work for this project has a very wide range of possibilities. First, we will migrate UNIK into KUBiC and ensure the implementation is correct. The most significant change involves moving Hadoop's environment to a cloud-based system. During our research, we dedicated substantial time to exploring how to incorporate Hadoop into the cloud and identified various solutions.

Several well-known companies offer easy access to multiple clusters and provide a guided framework for working with and monitoring these clusters. However, these solutions can be very expensive to maintain consistently. The companies that were considered:

- Amazon EMR (Elastic MapReduce)
- Google Cloud Platform (Dataproc)
- Microsoft Azure (HDInsight)

The main focus was on Google Cloud Platform's Dataproc. Dataproc is a Google-managed, cloud-based service for running big data processing, machine learning, and analytic workloads on the Google Cloud Platform. It provides a simple, unified interface for managing clusters of machines, running jobs, and storing data. Dataproc emerged as a response to the growing demand for efficient and scalable data processing solutions, particularly in the era of big data. It offers a free trial of $300 for cloud services.

Although Dataproc provided easy access to clusters, we needed a more consistent solution. Dataproc consumed a lot of resources, and maintaining a non-stop cluster quickly became expensive.

In our search for a more sustainable approach, we came across OpenStack. OpenStack is an open-source cloud computing platform that enables users to build and manage both public and private clouds. Within OpenStack, the Sahara project is designed specifically to provision and manage big data processing frameworks like Hadoop and Spark. Sahara simplifies the process of deploying and scaling these frameworks by providing pre-configured templates and automated cluster management. This allows for more efficient use of resources and a cost-effective solution for running Hadoop in a cloud environment.

Bibliography

[1] Apache Hadoop. (n.d.). Retrieved from https://hadoop.apache.org

[2] Apache Software Foundation. "Apache Spark." Retrieved from. https://spark.apache.org

[3] White, Tom. Hadoop: The definitive guide. " O'Reilly Media, Inc.", 2012. https://books.google.es/books?hl=en&lr=&id=drbI_aro20oC&oi=fnd&pg=PR5&dq=what+is+hadoop&ots=t1xmugiWf5&sig=0oNjsXE295sMCGV_EHivHDe7LCE&redir_esc=y#v=onepage&q=what%20is%20hadoop&f=false

[4] K. Aziz, D. Zaidouni and M. Bellafkih, "Real-time data analysis using Spark and Hadoop," 2018 4th International Conference on Optimization and Applications (ICOA), Mohammedia, Morocco, 2018, pp. 1-6, doi: 10.1109/ICOA.2018.8370593.

[5] Dey, Abhik. (2023, June 25). Apache Hadoop 3.3.6 installation on Ubuntu 22.04. Medium. https://medium.com/@abhikdey06/apache-hadoop-3-3-6-installation-on-ubuntu-22-04-14516bceec85

[6] Angular. (n.d.). Tutorials. Retrieved from https://angular.dev/tutorials

[7] Borthakur, D. (2008). HDFS architecture guide. Hadoop Apache project, 53(1-13), 2. Retrieved from https://docs.huihoo.com/apache/hadoop/1.0.4/hdfs_design.pdf

[8] MongoDB, Inc. (n.d.). Getting started. MongoDB. Retrieved from https://www.mongodb.com/docs/manual/tutorial/getting-started/

[9] Spring. (n.d.). Uploading files. Spring Guides. Retrieved from https://spring.io/guides/gs/uploading-files

Annex I – Hadoop and cluster configuration

In this part of the document you will find the hadoop and shh configuration of each machine, have in mind all of the configuration has been done with Ubuntu 22.04. As mentioned before in the physical architecture, three computers were used to construct the hadoop environment, one Name node and two Data nodes.

For the ssh configuration you can view in the following document:

Setting up machines

Hadoop configuration:

Setting up Hadoop and Spark

Submitting a Spark Job via console:

Submitting spark Job

You can also view demo example of the application; this can be helpful to better understand the functionality of the system. It is important to note that this application is not available for external users, because of the private database credentials and the hadoop configuration on the personal computers. If you wish to set up the environment you can follow the guides above.

Here you can find the steps user request and files are processes in the hadoop environment and how each component work together to process data

Writing Data:

Step 1: The client contacts the NameNode to initiate a file write operation.

Step 2: The NameNode provides the client with the block locations and data Nodes to write to.

Step 3: The client writes data directly to the assigned Data Nodes. Each block of data is written to multiple Data Nodes.

Step 4: Data Nodes confirm the block writes to the NameNode.

Step 5: The NameNode updates its metadata to reflect the new file and its block locations.

Reading Data:

Step 1: The client contacts the NameNode to read a file.

Step 2: The NameNode responds with the metadata and the locations of the data blocks.

Step 3: The client directly contacts the relevant Data Nodes to read the blocks of data.

Step 4: Data Nodes serve the data blocks to the client.

Fault Tolerance:

Replication: Data blocks are replicated across multiple Data Nodes (the default replication factor is 3). This ensures data availability in case of node failure.

Heartbeat and Block Reports: Data Nodes regularly send heartbeat signals and block reports to the NameNode to report their health and block status. If the NameNode

does not receive a heartbeat from a DataNode within a certain time, it marks the DataNode as dead and re-replicates the blocks to other Data Nodes.

Diagram Integration:

NameNode: Coordinates the file upload and data processing requests. It manages the distribution and replication of data blocks across the Data Nodes.

Data Nodes: Store and process the actual data. They handle file read/write requests from the client as well as replicate and manage data blocks as instructed by the NameNode.

UNIK Controller: Acts as the mediator between the Angular user interface and the Hadoop cluster, forwarding user requests to the NameNode and processing the responses.

You can view all the connection configurations on the application.properties from the hadoop controller:

```
spring.hadoop.fsUri=hdfs://192.168.0.230:9000
spring.data.hadoop.fs-uri=hdfs://192.168.0.230:9000
spring.data.hadoop.hdfs-root-dir=/user/hadoop
spring.hadoop.config.fs.defaultFS=hdfs://192.168.0.230:9000
spring.hadoop.config.dfs.client.use.datanode.hostname=true
spring.hadoop.config.dfs.replication=1

# MongoDB properties
spring.data.mongodb.database=${env.MONGO_DATABASE}
spring.data.mongodb.uri=mongodb+srv://${env.MONGO_USER}:${env.MONGO_PASSWORD}@${env.MONGO_CLUSTER}
#spring.data.mongodb.uri=mongodb://${env.MONGO_USER}:${env.MONGO_PASSWORD}@ac-mkvq4bl-shard-00-00.xefdshg.mongodb.net:27017,ac-mkv

# Logging properties
logging.level.org.springframework=DEBUG
logging.level.org.apache.hadoop=DEBUG
logging.level.com.unik.hadoopcontroller.service.HdfsDirectService=DEBUG

# Spark properties
spark.app.name=SparkJob
spark.master=yarn
spark.deploymode=cluster
```

Image 11. Image representing the connection to the clusters and to MongoDB