UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

IMPLEMENTATION OF A DATA LAKE IN A MICROSERVICES ARCHITECTURE

Ricardo Pinto da Costa Chiolas Macedo

Mestrado em Engenharia Informática

Dissertação orientada por:
Prof. Doutor Diogo Miguel Ferreira Poças

2023

Acknowledgments

I would like to express my sincere gratitude to the many individuals and institutions that have supported me throughout this academic journey. I am grateful to my friends and family for their continuous encouragement and understanding during the difficult stages of this journey. Your belief in me was a constant source of motivation, and for that, I am profoundly grateful.

I extend my heartfelt thanks to my advisors, Diogo and Simão, whose guidance and mentorship were instrumental in shaping the direction and quality of this thesis. Their expertise, dedication, and feedback have been indispensable in refining my research and ensuring its scholarly rigor. I am truly fortunate to have had the opportunity to work with such knowledgeable and supportive mentors.

Furthermore, I would like to express my appreciation to Trust Systems for their support and the opportunity to engage in this thesis. The resources and environment provided by the company significantly enriched the scope of this work, allowing me to explore real-world applications and challenges.

In conclusion, this journey would not have been possible without the collective support and guidance of those mentioned above and many more who contributed in various ways. Your belief in the importance of this research and your encouragement were the driving forces that propelled me towards its successful completion. For this, I am deeply thankful.

Resumo

À medida que o mundo entrou na era da big data, as empresas enfrentaram inúmeros desafios devido ao aumento exponencial do volume, da velocidade e da variedade de dados gerados. Este cenário complexo exigiu uma transformação na forma como as organizações abordavam o armazenamento, processamento e análise de dados. As soluções tradicionais, como as bases de dados relacionais, que um dia foram a espinha dorsal da gestão de dados, rapidamente se mostraram inadequadas para lidar com esta avalanche de dados de formatos diferentes. As empresas perceberam que, para permanecerem competitivas num mercado em constante evolução, era fundamental adotar uma abordagem mais flexível e escalável para a gestão de dados.

No início dos anos 2010, o conceito de data lake surgiu para enfrentar os desafios do big data. Os data lakes são repositórios centralizados e flexíveis que armazenam dados brutos, independentemente da sua origem ou estrutura. Isto representou uma mudança significativa em relação aos modelos tradicionais, nos quais os dados eram estruturados e armazenados em tabelas fixas. Esta abordagem inovadora permitiu que as empresas não só armazenassem grandes volumes de dados, mas também os processassem de maneira eficaz. Os data lakes tornaram-se assim o alicerce sobre o qual as análises de big data foram construídas, capacitando as empresas a explorar dados de diversas fontes. À medida que o conceito de data lake evoluiu, surgiram novas tecnologias e práticas recomendadas para a sua implementação. A integração de data lakes em projetos que seguem uma arquitetura de microserviços destacou-se como uma abordagem eficaz para tirar o máximo proveito desta tecnologia emergente.

Esta tese propõe-se a oferecer uma solução abrangente e prática para a integração de um data lake em projetos fundamentados em microserviços. O objetivo fundamental é capacitar a organização, para a qual este projeto de data lake é concebido, com as ferramentas e o conhecimento essencial para explorar todo o potencial dos data lakes, alinhando-se com as atuais tendências em gestão de dados. Os objetivos desta tese abrangem diversos aspectos cruciais, incluindo uma compreensão dos desafios inerentes ao universo do big data, uma análise das tendências e tecnologias emergentes nos data lakes, uma avaliação comparando as vantagens dos data lakes em relação aos tradicionais data warehouses e, por fim, o desenvolvimento e implementação de uma solução de data lake meticulosamente alinhada com os requisitos específicos da organização.

A tese começa por explorar os conceitos fundamentais de big data, data lake e arquitetura de microserviços. Esta estabelece uma base sólida ao mergulhar numa revisão abrangente da

literatura, analisando as tendências em constante evolução nas arquiteturas e tecnologias de data lake. Nesta revisão da literatura sobre arquiteturas existentes para a implementação de data lakes, são exploradas diversas abordagens, que incluem a arquitetura Lambda, a arquitetura Zone e a arquitetura Pond. As características distintas e vantagens de ambas são analisadas, com o objetivo de identificar possíveis soluções para uma implementação eficaz de um data lake num contexto de microsserviços. Esta análise crítica destas arquiteturas estabelece a base para aprofundar a pesquisa e a discussão na tese, permitindo uma visão mais elaborada sobre como estas arquiteturas se relacionam com os requisitos das empresas na era da big data e como podem ser integradas com sucesso em ambientes de microserviços, promovendo a inovação e a eficiência na gestão de dados.

Quanto às tecnologias, estas são responsáveis pela ingestão de dados de diversas fontes e formatos, o processamento eficiente de dados brutos, o armazenamento escalável e económico de dados, e as soluções de visualização que capacitam os utilizadores a obter insights a partir dos dados armazenados. A ingestão de dados desempenha um papel crítico na recolha de dados de diversas fontes. Tecnologias como o Apache Kafka, o Apache Nifi e o Apache Spark destacam-se nesta camada, permitindo que os dados fluam de forma confiável e segura para o data lake. No que diz respeito ao processamento de dados, a ferramenta mais utilizada em estruturas como os data lakes tende a ser o Apache Spark. Estas ferramentas desempenham um papel fundamental na transformação e análise eficaz dos dados. O armazenamento de dados também é uma consideração crítica, pois os data lakes são projetados para armazenar grandes volumes de dados de forma económica e eficiente. Tecnologias como o HDFS (Hadoop Distributed File System) e sistemas de gerenciamento de objetos, como o Amazon S3, oferecem opções de armazenamento escalável e económico para atender às necessidades das empresas. No que diz respeito à visualização de dados, ferramentas como o Apache Superset, o Power BI e o Tableau desempenham um papel fundamental na apresentação dos dados de maneira compreensível e interativa. Estas ferramentas capacitam os utilizadores a criar painéis interativos e relatórios personalizados que permitem a análise eficaz dos dados armazenados.

Através de uma análise comparativa, este estudo ilumina as vantagens distintas dos data lakes quando contrastados com outras soluções. Especificamente, destaca a notável capacidade dos data lakes de lidar com uma ampla variedade de tipos de dados, a sua escalabilidade inerente e o seu desempenho sólido, ao mesmo tempo que aborda de maneira minuciosa os desafios cruciais relacionados à governança de dados.

O cerne desta tese gira em torno do desenho e implementação de uma solução de data lake, meticulosamente elaborada para se integrar perfeitamente nos projetos baseados em microserviços da empresa, seguindo um conjunto de requistos predefinidos. Esta solução é concebida com base numa arquitetura em camadas, onde cada uma desempenha funções específicas no processo de gestão dos dados. A camada de ingestão de dados, por exemplo, é responsável pela entrada e filtração dos dados brutos no data lake. Nesta camada, tecnologias como Apache Spark e Hadoop, garantem a eficiência na ingestão e no processamento inicial dos dados. Na camada de armazenamento, os dados são organizados e armazenados. Aqui, o sistema incorpora tecnologias

de armazenamento distribuído, como o HDFS, para garantir que os dados estejam disponíveis e seguros. A camada de processamento é alimentada pela poderosa capacidade de processamento paralelo do Apache Spark, permitindo a análise e o processamento avançado dos dados. Por fim, a camada de visualização de dados é mantida pelo Apache Superset, fornecendo ferramentas para criar painéis interativos e consultas eficazes sobre os dados. Todo o sistema do data lake é implementado como uma aplicação Spring, onde o Apache Spark é executado em paralelo. Esta combinação assegura uma integração suave entre as diversas camadas e tecnologias, permitindo que o data lake opere de maneira contínua e eficaz.

Adicionalmente, este trabalho apresenta um estudo de caso que serve como demonstração prática das capacidades de processamento e geração de relatórios do data lake. É realizada uma análise de dados abrangente, e os resultados são traduzidos em gráficos e visualizações detalhadas. Este estudo de caso é uma oportunidade para mostrar as capacidades que um data lake pode proporcionar ao gerir e processar grandes volumes de dados. Os gráficos gerados permitem uma representação clara e concisa das tendências e padrões presentes nos dados, tornando a informação facilmente compreensível para a tomada de decisões. Este exemplo prático evidencia os benefícios tangíveis da solução do data lake implementada, demonstrando como a empresa pode utilizar eficazmente as capacidades de processamento e visualização de dados, impulsionando a inovação e a entrega de valor aos seus clientes.

Com esta tese, a empresa estará melhor preparada para enfrentar os desafios do big data e aproveitar as oportunidades oferecidas pelos data lakes na era moderna da gestão de dados. A integração bem-sucedida do data lake em projetos baseados em microserviços proporcionará à empresa uma vantagem competitiva e *insights* valiosos para a tomada de decisões informadas.

Palavras-chave: Data lake, Big Data, Microsserviços, Spark, Hadoop

Abstract

As the world entered the big data era, companies faced lots of challenges due to the increase in volume, velocity, and variety of data. Traditional relational databases could no longer be an option to tackle this problem. Acknowledging the pressing demand for a revolutionary solution, the concept of the data lake came to fruition in the early 2010s.

This thesis presents a solution for the integration of a data lake into microservices-driven projects, aiming to equip a company with insights and strategies to harness the power of data lakes in modern data management paradigms. The objectives of this thesis encompass a comprehensive understanding of big data challenges, a detailed analysis of evolving data lake trends and technologies, a comparison of data lakes and traditional data warehouses, and the design and implementation of a tailored data lake solution.

It begins by exploring the concepts of big data, data lake, and microservices architecture before delving into a literature review of current trends in data lake architectures and technologies. Through a comparative analysis, this study highlights the advantages of data lakes compared to other solutions, such as their ability to handle diverse data types, scalability, and performance, while addressing challenges related to data governance.

The core of this thesis revolves around designing and implementing a data lake solution, meticulously crafted to seamlessly integrate into the company's microservices projects. This solution is designed using an on-premises architecture and incorporates technologies such as Apache Spark, Hadoop, Apache Superset, and the Spring framework.

Furthermore, a case study is presented highlighting the data lake's processing and reporting capabilities.

Keywords: Data lake, Big Data, Microservices, Spark, Hadoop

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Since the world entered the era of big data, the challenges in managing and making sense of this vast and diverse information landscape have grown exponentially. Data started coming in from a diversity of sources such as online transactions, emails, videos, audio, images, click streams, logs, posts, search queries, sensors, and mobile phones and their applications. It arrived in extreme volume, with a huge variety and at a never before seen velocity [33]. As the name implies, big data refers to heterogeneous, huge amounts of data that are too large, complex, and diverse for traditional databases to store or manage[31].

As a result of the unfitness of traditional relational databases to handle big data, a new concept emerged, the concept of data lake. James Dixon was the first to define this concept in 2010 [31]. Since then, numerous authors have proposed several definitions, making it difficult to find one that unequivocally defines the concept of data lake. Some authors in [10] proposed a definition based on the analysis of 662 papers, defining a data lake as a central repository for storing, processing, and analyzing raw data. It can store a wide range of formats, including unstructured, semi-structured, and structured data sources. Data lakes quickly became popular and were seen as a replacement for data warehouses because they provide a cost-effective and technologically flexible solution to overcome the challenges brought by big data.

Although there is no single architecture that all organizations should use when constructing a data lake, two major generic architectures have been identified in previous literature reviews [31]: zone and pond architecture. The choice of architecture will depend on the needs of each individual or organization to design and implement the architecture that suits them the most. When it comes to technologies used to design and implement data lakes, Apache Hadoop and its ecosystem of open-source tools seem to be the most used option despite the recent appearance of cloud solutions. For data to be usable in a data lake, it is necessary to define mechanisms to catalog and secure data because without those elements the data lake can become a data swamp. A data swap is a situation where a data lake becomes disorganized, poorly managed, and difficult to navigate, making data lose all its value.

Data lakes emerged to tackle the challenges of the big data era, but they also brought challenges such as a lack of consensus and information regarding architectures, technologies, and management policies. Since data lakes have become the new trend because of the value they have

brought to various organizations, it is important to understand and define the best solution to design and implement a data lake based on the ones available. The solution should be suited to the organization's needs so that it can get the most value from its data.

1.1 Motivation

In today's constantly evolving digital world, data has emerged as a vital component for companies, driving strategic decision-making, improving consumer experiences, and stimulating innovation. As organizations attempt to remain competitive, the adoption of new trends and advanced technologies becomes crucial. One such technology that has received a lot of interest is the concept of a data lake bringing organizations to explore the benefits and adopt this new trend.

Companies have traditionally maintained their data in separate systems, each meant to fulfill a distinct purpose or business function. This approach, however, frequently leads to data fragmentation, redundancy, and inefficiencies in data processing and analysis. As data quantities continue to rise massively, the limitations of traditional systems become more evident, preventing a company's ability to take full advantage of its data resources.

In this context, the rise of data lakes has resulted in a paradigm change in how organizations handle and take advantage of their data. By adopting a data lake, the data is no longer spread across multiple systems, enabling companies to take advantage of data sharing and collaboration across departments and teams.

This project is being developed for Trust Systems, a Portuguese company that has operated in the area of Information Security since 2016. As a company that is always looking for new ways to innovate and keep up with new technologies and trends, the idea of this project of implementing a data lake makes all sense in the current context. This project is also the first contact the organization has ever had with this new technological trend.

In addition, the microservices architecture approach used by the organization on its projects constitutes a modern approach to developing scalable and resilient applications. Microservices provide benefits such as increased agility, shorter development cycles, and higher fault tolerance by decoupling complicated systems into smaller, independently deployable services. Incorporating a data lake into a microservices architecture aligns with the organization's desire to upgrade its technology stack and implement best practices in system design.

1.2 Goals

The main goals of this thesis are to provide the organization with a solid understanding of the latest data management paradigms and facilitate the effective integration of a data lake into its microservices-driven projects. These goals can be further divided as :

1. **Comprehending big data realities and challenges**

   The initial goal of this thesis is to dive fully into the world of big data and its multiple

challenges. This thesis intends to give a complete understanding of the data environment that drives the demand for innovative solutions like data lakes by carefully looking into the fundamental characteristics of big data - volume, velocity, variety, and veracity. Understanding the complexities of big data challenges will not only guide decision-making processes, but will also set up the foundation for developing strategies that maximize data extraction, transformation, and loading processes within the organization's growing ecosystem.

2. **Analyzing evolving data lake trends**

   This thesis also aims to perform a comprehensive analysis of the current trends in data lake architectures, technologies, and management policies. The purpose is to provide the organization with a clear path to comprehend and use current data lakes by closely examining how data lakes are evolving, including their detailed design elements and the new, innovative techniques being implemented.

3. **Evaluating Data Lakes vs. Data Warehouses**

   Another essential goal of this study is to properly understand the advantages and disadvantages of data lakes compared to traditional data warehouses. The thesis seeks to highlight the unique benefits of data lakes, through a detailed comparison. At the same time, the study will examine the challenges associated with preserving data quality, managing data rules, and ensuring queries perform effectively. The analysis will assist the organization in making informed decisions about adopting data lakes instead of or in addition to traditional data warehouses, keeping up with the most recent approaches in modern data management.

4. **Designing and implementing a suitable data lake solution**

   The building of a custom-made data lake solution that seamlessly integrates into the organization's projects built on a microservices framework is at the heart of this thesis's objectives. This research presents a solution that makes the most of the data lake by achieving an understanding of what the organization really requires and wants to achieve. This solution will cover topics such as how the data lake is constructed, what technologies are being used, and how data is handled and controlled. All of these features were planned in a way that properly aligns with the organization's microservices approach, ensuring that everything works nicely together in the organization's IT environment.

5. **Testing the quality of the implemented solution**

   The testing phase plays a pivotal role in validating the data lake solution. Rigorous tests confirm its quality and suitability for the organization, ensuring accuracy, speed, and resilience under various conditions. Successful testing assures seamless integration with the microservices architecture, enhancing data accessibility, usability, and analytical capabilities. Most importantly, it empowers the organization with adaptability in a dynamic business landscape.

1.3 Structure of the document

This document is organized to provide a comprehensive overview of the process of establishing a data lake within a microservices architecture that is suited to the organization's needs. Each chapter serves a specific purpose and helps to a comprehensive understanding of the subject.

- **Chapter 2 - Background:** This chapter establishes the foundation for the following chapters by digging into the fundamental principles of big data, data lake, and microservices architecture. It will investigate the context and significance of data lakes in modern data management techniques, setting the stage for the following chapters.
- **Chapter 3 - Related Work:** An in-depth review of existing literature and real-world data lake cases is given here. This chapter extends the research's contextual foundation by providing insights into successful implementations using different types of architectures, challenges encountered, and innovative techniques and technologies used in this context.
- **Chapter 4 - Analysis of Data Lake Architectures:** This chapter looks into different data lake architectures, describing how they are designed and organized. Throughout this analysis, the pros and cons of each architecture presented are discussed as well as the new trends in the architectural paradigm. This helps to understand which architecture aligns best with the organization's goals and data requirements.
- **Chapter 5 - Technologies used in Data Lakes:** This chapter looks into the fundamental technologies required for data lakes. It discusses ingestion techniques, storage options, data processing, and access technologies.
- **Chapter 6 - Design:** The design chapter explores the strategic design choices for incorporating a data lake into a microservices architecture. This chapter also discusses architectural decisions, technological choices, and data governance strategies for arming the organization with the best solution according to its requirements.
- **Chapter 7 - Implementation:** This chapter goes over the practical processes required for implementing the envisioned data lake solution. It covers the process of transforming theoretical conceptions into functional realities, including development processes, tools employed, and difficulties encountered throughout the implementation phase.
- **Chapter 8 - Case Study: Smart Meters in London:** In this chapter a detailed analysis of a dataset regarding smart meters in London is performed. The analysis aims to showcase the capabilities of a data lake in processing data and presenting advanced analytics.
- **Chapter 9 - Conclusion:** The final chapter summarizes the key findings, insights, and lessons learned throughout the research process. It also discusses the extent to which the objectives mentioned in the earlier sections have been met and discusses possibilities for future research and implementation.

Chapter 2

Background

This chapter lays the groundwork for the project by introducing the key concepts that will be discussed. It provides an overview of big data, data lakes, Hadoop, and microservices, setting the stage for their relevance to the project's development and implementation.

2.1 Big Data

Big data's beginnings can be found in the early 1950s when businesses first started storing and analyzing data on massive computers [24]. In the following decades, the development of powerful computer hardware and software made it possible to process even larger and more complex datasets.

Big data refers to large and complex datasets from various sources that are difficult to process using traditional data processing tools. It has grown exponentially due to the increasing usage of internet-connected devices and the generation and sharing of more information online. It is often characterized by what has become known as the 3V's: volume, variety, and velocity. These characteristics help to define the unique challenges and opportunities that big data presents. Volume refers to the large amount of data that needs to be processed, velocity refers to the speed at which data is generated and needs to be processed, and variety refers to the diversity of data types that need to be analyzed (structured, unstructured, and semi-structured data). Over time, other V's have been discussed and presented for a better definition of big data such as veracity which refers to the quality and accuracy of the data, and value which refers to the potential insights and benefits that can be gained from analyzing big data.

The quality and accuracy of big data are other challenges. As data is generated in such large amounts, it can be challenging to ensure that it is accurate and free of errors. This can make it difficult to draw reliable conclusions from the data. To deal with these challenges it is required a far more sophisticated and capable tool than the traditional ones like data warehouses and relational databases and that is where data lakes come in.

In **Figure 2.1**, there are presented the 5V's of bid data (Volume, Variety, Velocity, Veracity, and Value).

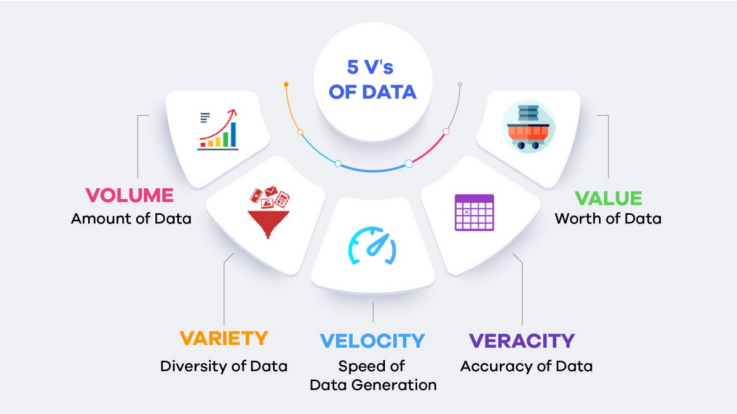

Figure 2.1: The V's of big data [15]

2.2 Data Lake

Although the idea of a data lake has been around for a while, it has recently gained popularity due to the growth of big data and the necessity to store and handle enormous volumes of data in a scalable and cost-effective way, appearing as a solution to tackle the limitations of the traditional data warehouses.

There seem to be various definitions for the data lake concept as some authors have proposed their own and reviewed others to find a proper one but there is consensus in the definitions as most refer to a data lake as a central repository for storing all kinds of data, both structured and unstructured in their unprocessed state. Large amounts of data can be stored in this kind of repository in a scalable, economical manner, making the data accessible for processing and analysis [31, 10]. The idea of this concept is pretty simple, instead of storing all of the data in a data repository, move the data in its original format into a data lake eliminating the costs of data ingestion, such as transformation. It is no longer necessary to perform changes to the data before storing it, data is stored in its raw format and it is processed as it is needed, being available for analysis by everyone in an organization.[16]

Despite the existence of other technologies, the literature suggests that a distributed file system, like Hadoop, that can manage massive volumes of data and supports parallel processing is often highly associated with the process of building and maintaining data lakes [16, 31]. This allows organizations to store and analyze data from numerous sources, including transactional systems, social media, site logs, and sensors.

Figure2.2 demonstrates the Data Lake's central role in modern data management. At its heart, the data lake serves as the repository for vast volumes of raw and unstructured data. Four smaller circles connect to this core hub, symbolizing important components: "Machine Learning" for sophisticated analytics, "On-Premises Data Movement" for seamless data flow, "Analytics" for data transformation and insights, and "Real-Time Data Movement" for instant data processing. Together, these aspects show the Data Lake ecosystem's versatility and strength, enabling efficient data ingestion, management, analysis, and real-time insights for a wide range of applications.

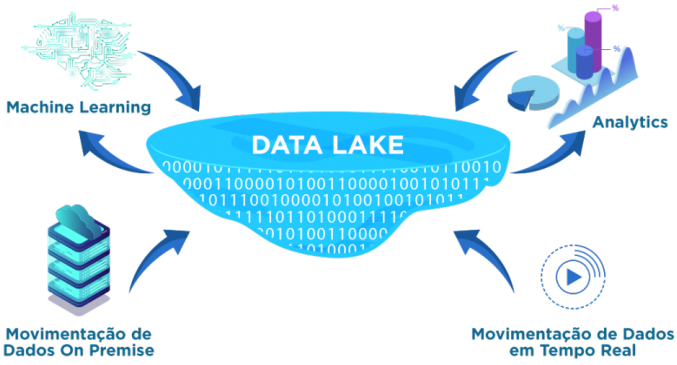

Figure 2.2: Data Lake [7]

2.3 Hadoop

Hadoop is a popular open-source platform for distributed big data processing and storage. It was developed by the Apache Hadoop project and allows companies to efficiently process large amounts of data [35]. The Hadoop platform consists of several components, including the Hadoop Distributed File System (HDFS) for storing data on multiple machines, and MapReduce for processing data in parallel. Hadoop is also commonly used as a platform for creating data lakes due to its ability to store and process large amounts of data efficiently and at a low cost [32]. Data lakes created using Hadoop can provide organizations with valuable insights by allowing them to analyze all of the data they collect including text, photos, audio, and video.

Additionally, Hadoop's support for batch and real-time processing makes it an ideal platform for data lakes. Batch processing provides for the offline processing of vast amounts of data, whereas real-time processing allows for the processing of data as it is generated, offering insights to organizations in real time. Besides its core components, Hadoop also includes several tools and frameworks that can be used by organizations in the creation and management of data lakes. These tools and frameworks will be explored further in Chapter 4.

Figure 2.3 demonstrates the several technologies that are going to be presented and described in Chapter 4, which compose the Hadoop ecosystem.

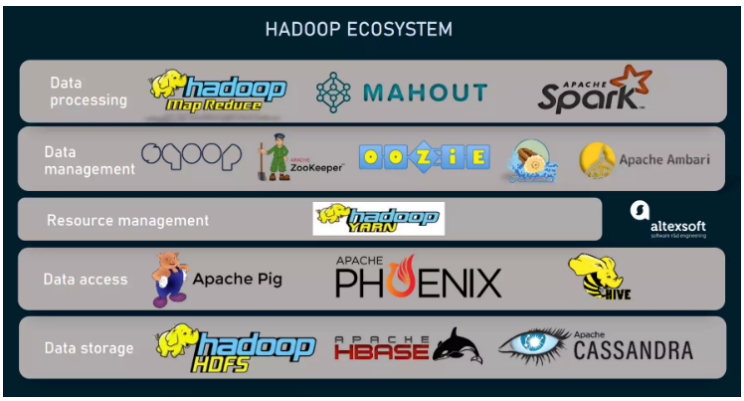

Figure 2.3: Hadoop Ecosystem [14]

2.4 Microservices Architectures

A microservices architecture, commonly known simply as microservices, is an architectural solution that is based on a collection of independently deployable services that have their own business logic and database, and serve a specialized purpose [19]. This enables more flexible management over the application's many components, as well as simpler modification and evolution of specific services without affecting the entire system. Because a microservices architecture is made up of modules that run separately, each service may be built, updated, deployed, and scaled independently of the others [19]. This software development strategy has grown in popularity in recent years because of its ability to facilitate agile development, allow for faster deployment of new features, and increase scalability and reliability [28]. A significant real-world application of a microservices architecture can be seen in e-commerce platforms such as Amazon.

Figure 2.4, showcases a generic microservices architecture with its independent microservices and dedicated databases:

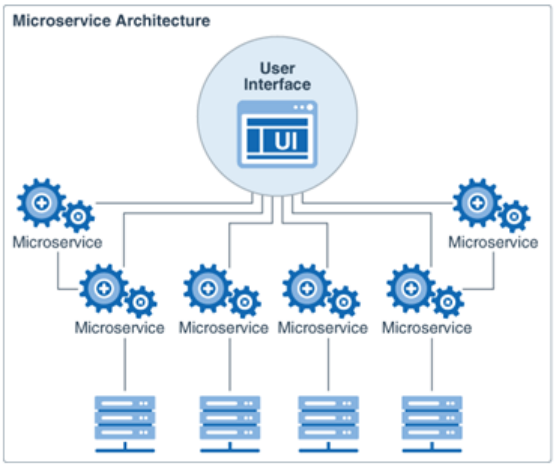

Figure 2.4: Microservices architecture [9]

Chapter 3

Related Work

The implementation of data lakes has received much attention in recent years as organizations seek reliable solutions for handling and extracting value from their constantly growing data. This chapter looks into the work associated with data lake implementation, emphasizing its capabilities, challenges, and a comparison with traditional data warehousing approaches. It also seeks to provide a comprehensive understanding of the considerations, best practices, and lessons gained in installing and leveraging the power of data lakes by reviewing existing literature and studies.

Furthermore, this chapter digs into the main challenges experienced during data lake implementation, such as data quality, governance, and security, as well as the potential benefits and constraints associated with this modern data management approach. By critically examining and synthesizing current knowledge in the field, this chapter establishes the foundation for the succeeding discussions and analyses in this thesis, ultimately contributing to a full understanding of data lake implementations in real-world scenarios.

3.1 The capabilities and value of a Data Lake

Data lakes are increasingly becoming crucial to many organizations' data management strategies. By centralizing the data and with the advancements in Data Science and Machine Learning, a data lake gives organizations a range of capabilities that have never been explored, helping them grow with a lot more business insights as compared to any other system in the organization.

This section looks into the multiple capabilities brought to the forefront by data lakes, revealing how they revolutionize how data is used and increase the importance of data within organizations. Some of these capabilities are described as follows:

- As a central repository, a data lake makes it possible for many teams and departments to access all of the organization's data which helps teams and departments share data among themselves, avoiding the issue of siloed data that happens when several teams or departments each have their own unique data storage systems and are therefore unable to easily share data.

- A data lake can provide a high level of flexibility and versatility. Organizations can store all

of their data in one central location without worrying about the specific format or structure of the data because data lakes store data in its raw, unstructured form. This enables organizations to store and manage data from a wide range of sources, including structured and unstructured data, without the need for data transformation or organization and avoiding costly efforts [16].

- Another advantage of a data lake is its ability to support a diverse set of data analytics and processing tools [23]. Organizations can use a data lake to perform complex data analysis and machine learning tasks using a variety of tools like Python, SQL and R. This allows them to take the most advantages from their data and make decisions based on it.

- Additionally, data lakes can assist organizations in improving data governance, lineage, and security by implementing strict access controls and monitoring the data used to protect sensitive information [23].

- Nonetheless, data lakes can effortlessly scale to accommodate vast amounts of data, making them suitable for businesses experiencing data growth. Whether it's petabytes or exabytes, data lakes can handle it.

3.2 Data Lake vs Data Warehouse

The importance of data lakes and data warehouses in the developing ecosystem of data management systems has grown dramatically. While both entities serve the same basic function as data repositories, they contrast significantly in their architectural bases, features, and intended applications. Data warehouses, which have been around for a long time, have historically been the major choice for organizations to store and manage their data. The creation of data lakes, on the other hand, is a direct response to the growth in big data, constituting a modern approach to data storage and processing that coincides with the latest trends in the industry.

This section compares data lakes and data warehouses, explaining the different characteristics that distinguish them. Data structure, data storage, data processing processes, scalability, and user paradigms are discussed, offering an informative panorama of their differences. This analysis serves as a foundation for determining how these two data management paradigms align with varying organizational needs and objectives.

The differences between data lakes and data warehouses highlight their many uses and applications within the field of data management. Data scientists are the main target audience for a data lake, which can store unstructured, raw data indefinitely to be used now or in the future. The schema can be defined after data storage, allowing for flexibility in integrating various data sources. Big data analytics, predictive analytics, and machine learning all benefit greatly from the support provided by data lakes. Data is only structured as needed in the processing paradigm, which mostly uses an ELT (Extract, Load, Transform) method.

Data warehouses, on the other hand, primarily serve business professionals by storing structured data that has been processed and cleaned for immediate business purposes. The schema, which is established prior to data storage, acts as a blueprint for how the data will be arranged, stored, and structured within the warehouse. The data warehouse is an essential tool for strategic decision-making because of its strengths in data analytics, business intelligence, and data visualization. To ensure that data from diverse sources is cleaned up and transformed before being saved, the processing technique uses an ETL (Extract, Transform, Load) process. This process aligns data from various sources with business analytical requirements.

These differences demonstrate the unique capabilities of both paradigms and their ability to respond to different goals of organizations. Importantly, these two approaches should not be viewed as mutually exclusive alternatives, but rather as complimentary assets that, when wisely integrated, provide an effective solution for organizations. By combining the strengths of data lakes with data warehouses, organizations can take advantage of each solution's benefits, improving insights based on data analytics and strategic decision-making skills. However, it is necessary to evaluate the context where a solution will be implemented to make an informed decision on what approach should be used or if the best solution is to use both of them.

Table 3.1, offers a visual reference, highlighting the key differences between data lakes and data warehouses across the stated categories.

Table 3.1: Data Lake vs Data Warehouse (adapted from [1])

	Data Lake	Data Warehouse
Users	Data from a data lake is typically used by data scientists	Data from a data lake is typically used by business professionals
Data Storage	A data lake can store data indefinitely for present or future usage and contains all of an organization's data in a raw, unstructured form.	A data warehouse stores structured data that has been cleaned up and processed so that it is ready for business needs.
Schema	Schema is defined after the data is stored, accelerating the capture and storage of the data.	Schema is established before the data has been saved.
Analysis	Predictive analytics, machine learning, data visualization, BI, big data analytics.	Data visualization, BI, data analytics.
Processing	ELT (Extract, Load, Transform). This process involves removing the data from its original location for storage in the data lake, and only structuring it as necessary.	ETL (Extract, Transform, Load). Data from various sources is cleaned, and then processed so that it is suitable for business analysis.
Cost	Storage expenses in a data lake are fairly low. Also, managing data lakes takes less time, which lowers operational costs.	Data warehouses are more expensive than data lakes and demand more management effort, which generates higher operational costs.

3.3 Analysis of Data Lake Architectures

A data lake architecture explains the conceptual organization of data within a data lake. It helps to understand where each type of data is located, facilitating the use of data lakes [17]. The first data lake architectures were created to solve the difficulties of storing and analyzing massive amounts of data, and they are still being improved as new technologies and methods are created.

This section analyzes various types of data lake architectures that have evolved in response to the challenges of storing and analyzing massive and diverse data volumes. Furthermore, it presents some of the most used data lake architecture models and provides a discussion on which architecture will fit best according to the organization's needs.

3.3.1 Types of Data Lake Architectures

Each data lake architecture has its own set of strengths and weaknesses but mainly they can be classified into two main categories: on-premises and cloud-based. In addition to these main categories, several hybrid data lake architectures combine on-premises and cloud-based components and some multi-cloud architectures.

Data Lake On-Premises

In this type of architecture, data lakes are built and maintained in a physical infrastructure [36]. This may involve installing and configuring the necessary hardware and software, such as servers, storage, and data processing tools. On-premises architectures provide a high level of control and security since organizations have complete control over the infrastructure and software stack used to build the data lake.

However, the main drawback is that they come with additional expenses and responsibilities since the organization must manage and maintain the infrastructure itself, having to continuously improve its staff's knowledge as technologies evolve [36]. Another concern is that hardware resources must be controlled so that ingested data does not exceed the available storage limit, which might result in considerable data loss [36].

Usually, architectures like the Zone and Pond models, further discussed in section 3.4, are implemented in on-premises environments, frequently using the Hadoop platform.

Cloud-based Architectures

Cloud architectures have been growing in popularity in recent years because of their capacity to provide scalable, cheap, and effective solutions for storing and processing huge volumes of data. These architectures consist of data lakes hosted and managed by third-party cloud providers, which can be public cloud giants like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, or private cloud providers offering customized infrastructure suited to specific business needs.

There are several advantages to choosing a cloud-based architecture [36] such as:

- **Immediate Availability:** Creating a data lake on the cloud is a very simple procedure. Accessing the cloud provider's website and following the provided instructions is all it takes to initiate the setup.
- **Hardware Scalability:** Cloud-based data lakes are designed to scale automatically. They may scale up or down dynamically in response to an organization's changing data demands. This eliminates the need to set up and manage a complete infrastructure, which is a significant benefit over on-premises solutions.
- **Rapid Responsiveness:** Cloud service providers offer quick response times, and technical professionals, while some experience with cloud services is required, can work successfully in these environments.
- **Comprehensive Tutorials:** Cloud service providers typically provide an extensive number of tutorials and documentation. These tools help organizations deploy and manage their cloud-based data lakes more efficiently.

However, it is important to understand that cloud services are based on subscriptions and the long-term cost-effectiveness of these solutions should be carefully evaluated. The total expenses of cloud services must be weighed against the costs of creating and maintaining an on-premises infrastructure, including a team of engineers [36].

It is worth noting that numerous cloud providers, including Microsoft Azure, Google Cloud, and IBM Cloud, provide data lake solutions, extending the range of alternatives available to organizations looking to leverage the power of the cloud for data management. Cloud-based data lake architectures are an appealing option for organizations looking to maximize the value of their data assets while benefiting from the scalability and cost-effectiveness of cloud solutions.

Hybrid Data Lake

This type of architecture involves a combination of the two main architecture types (on-premises and cloud-based) and allows data lakes to be maintained both locally and in the cloud, which can bring a lot of advantages but also some disadvantages [36]. By storing less relevant data locally, the storage costs in the cloud are reduced, making this one of the biggest advantages of this type of architecture. However, there is a big drawback to this solution, which is the higher costs for the organization that must have a skilled team of IT experts and engineers with knowledge in both storage environments to perform the communication between them in a successful way [36].

Multi-cloud Data Lake

Multi-cloud architectures are becoming increasingly popular because they allow organizations to take advantage of the finest cloud services from many cloud providers, leveraging the strengths of each cloud provider, such as increased storage capacity or improved scalability. However, this type of architecture requires greater technical skills, especially in the process of communication

between different cloud providers [36].

To summarize, while creating a data lake, organizations must evaluate both on-premises and cloud architectures, assessing the benefits and trade-offs of each option to decide which is the best match for their particular needs.

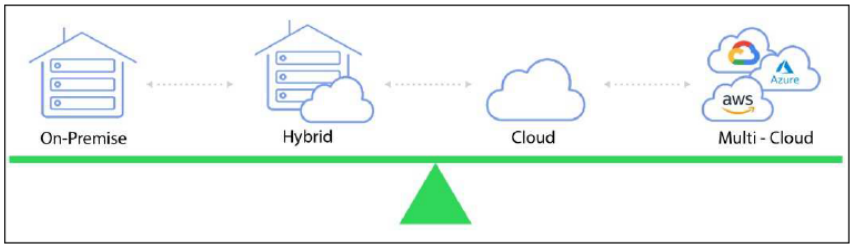

Figure 3.1: Data Lake architectures types [36]

3.4 Data Lake Architecture Models

This section explores the complex world of data lake architectural models. These architectural frameworks serve as the foundation for data lake implementations, determining how data is ingested, organized, and processed across the data lake. Because of the constantly growing volume and diversity of data, organizations seeking to maximize the value of their data must choose the most appropriate architectural model.

Some of the most popular data lake architecture models that have emerged in recent years will be explored. Each model takes a different approach to addressing the challenges of storing and managing big data, but they all have one purpose in common: to enable optimal use of data. According to literature in [31, 17, 20], there are two main variants of data lake architectures models: zone and pond architectures. From the pioneering Lambda Architecture, proposed in [26], to the versatile Lakehouse Architecture, the key features, advantages, and drawbacks of these models will be discussed.

Lambda Architecture

The lambda architecture was one of the first architectures to be proposed in [26]. It intends to offer a balanced method for handling batch and real-time data processing within a single system, putting more focus on data processing and consumption rather than data storage [20].

This architecture describes two processing layers: a batch layer and a speed layer. In some cases, a serving layer is also included like in [27].

The **batch layer** is responsible for performing batch processing on large amounts of data, typically using technologies such as Apache Spark or Hadoop. Here it is possible to access the data stored in persisting memory providing a historical overview of the data [20]. It is also used to do batch analytics.

The **speed layer** is in charge of managing real-time data processing techniques like event-driven processing and stream processing. This layer processes incoming data in almost real-time, enabling fast analysis and response to the data. After being stored in persistent memory, the data is no longer available in the speed layer. It is often built using technologies like Apache Flink or Apache Kafka.

The **serving layer** receives the batch views from the batch layer on a predefined schedule. This layer also receives the near real-time views streaming in from the speed layer. The batch views are indexed here so that they can be queried. The serving layer queues newly arriving data for inclusion in the next indexing run while one indexing task is active [4].

Figure 3.2 showcases a real-world implementation of a data lake lambda architecture, as demonstrated in the context of smart grid big data analytics, which was discussed in detail in [27]. In this architectural design, the lambda architecture serves as a robust framework for efficiently handling and processing massive volumes of data generated by smart grid systems.

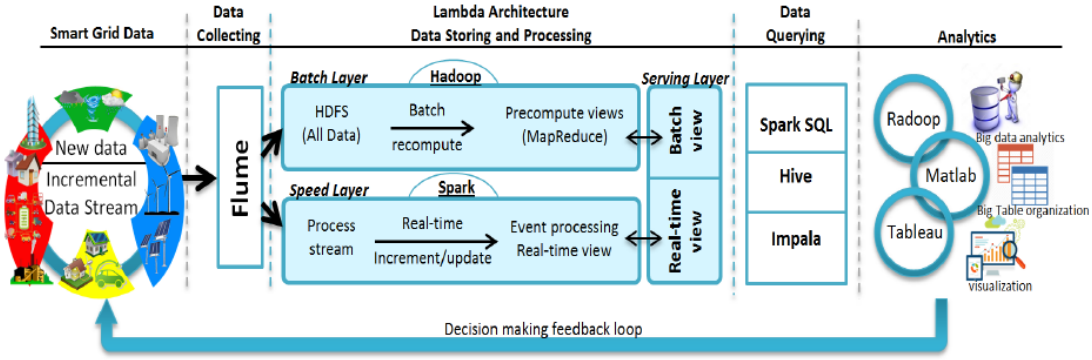

Figure 3.2: Data Lake Lambda Architecture for Smart Grids [27]

Pond Architecture

To organize the different types of data into a structure that can be analyzed, Bill Inmon proposed the data pond architecture model that consists of a set of components (ponds), each one logically separated from another [21, 20].

In his book, he describes five ponds that are represented in figure 3.3:

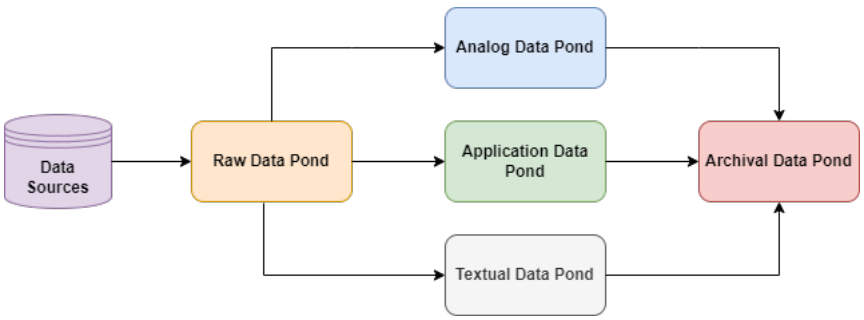

Figure 3.3: Data Pond Architecture

1. **Raw data pond**: This pond holds the ingested data in its raw form and serves as a staging area for the other data ponds, namely, analog, archival, and textual data ponds. Its only purpose is to hold the data, so when data is passed to other ponds, it is no longer available in this pond. Only data that is not used and data that do not fit into any of the other ponds remain in the raw data pond [17]. The next ponds should get the data from the raw data pond as quickly as possible. How tiny the raw data pond is and how quickly data leaves it serve as important measures of its quality.

2. **Analog data pond**: This pond holds the analog data namely the semi-structured and high-velocity data that typically come from the Internet of Things (IoT), log files, and APIs. In this pond, the volume of data is reduced and restructured to have a workable, manageable, and meaningful volume of data [21].

3. **Application data pond**: This pond is basically a data warehouse, which is populated with data that comes from transactions of executing applications. All the data in this pond is structured and contains value for the organizations. There is a necessity to integrate data in this pond as data may come from various applications, providing analysts with some difficulties.

4. **Textual data pond**: In this pond, unstructured textual data that can come from anywhere is stored. As the text is difficult to analyze in this pond, to perform a deep analysis of the data an ETL (Extract,Transform, Load) process called textual disambiguation is applied, resulting in data that can be stored as analytical records in the database.

5. **Archival data pond**: This pond receives data from the analog, application, and textual data ponds and its main function is to store data that isn't immediately needed for analysis but may be required at a later time. When data is no longer needed, it is moved from its original pond to the archive data pond.

The paper *"A Data Lake Architecture for Monitoring and Diagnosis System of Power Grid"* [25], presents a real-world application of a pond structure for efficient handling of diverse data from power grid sensors and devices. By organizing data into various ponds based on processing stages and characteristics, this architecture enhances the reliability of the power grid by enabling real-time data ingestion, validation, storage, and analytics.

Zone Architecture

Based on the literature in [33], the zone architecture seems to be the most popular option when it comes to building a data lake. Various zones exist in this architecture, each one with a responsibility to the data. Despite the absence of any standard zone architecture, the principle does not change: Depending on how much processing has been done to the data, a zone is assigned to it [17].

There are several alternatives for zone architectures that are proposed and discussed in literature. In [31], a systematic literature review is performed, and the authors talk about three main zones of a zone architecture: Landing Zone, Staged Zone, and Analytics Sandbox.

1. **Landing Zone**: In this zone, data is stored in its raw format. After being stored, data may undergo some basic cleaning and filtering processes so that data can be corrected. Data is always available in its raw form in this zone.

2. **Staged Zone**: Generally, this zone is set up for standardized and cleaned data. Data can come to this zone in two ways: through the landing zone, where data that needs to be processed comes, and directly to this zone if the data doesn't need any type of processing [33].

3. **Analytics Sandbox**: This zone efficiently stores large amounts of data from multiple sources and it is mostly used by data scientists to work on or construct analytical models and is also used for testing purposes. Data arrives in this zone from all levels as it can store structured and unstructured data. Due to the consumption of large amounts of unstructured data, this zone is located in the lower levels of the data lake architecture.

An example of the data flow of this architecture is shown in figure 3.4.

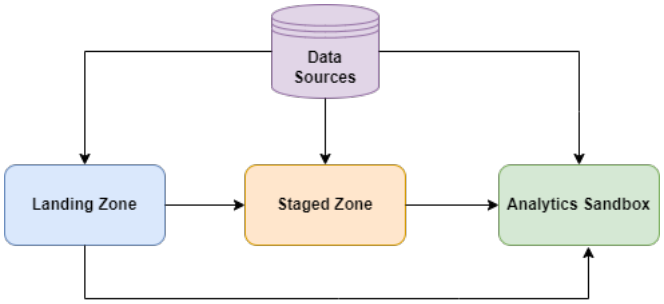

Figure 3.4: Example of a Data Zone Architecture

There are still numerous alternatives for this type of architecture and an overview of these alternatives is made in [18], where the authors evaluate individually different zone models available for industrial use cases due to the lack of consensus on which zones should be included, and assessment of each model. They concluded that the existing zone models could not meet all requirements so they developed a meta-model for zones and based on this meta-model they developed a zone-reference model that specifies six zones: Landing Zone, Raw Zone, Harmonized Zone, Distilled Zone, Explorative Zone, and Delivery Zone.

Lakehouse Architecture

As the name suggests, the data lakehouse architecture is a recent architecture that combines both the data lake and the data warehouse features [30]. It combines data lakes' flexibility, cost-

efficiency, and scalability with data warehouses' data management and ACID (Atomicity, Consistency, Isolation, and Durability) transactions, offering business intelligence and machine learning on all data [11]. To allow both types of queries (analytical and schema on-read), the Data Lakehouse generally stores data in both the Data Lake and the Data Warehouse after successful ETL batch processing [30].

The data lake part of the lakehouse architecture is divided into layers and areas [20]. The extraction and ingestion layer is responsible for collecting the data that will be stored in the landing/staging area, which is a temporary layer whose only purpose is to store data from multiple sources, for further processing. The data then passes to the foundation area where it is persistently stored.

In the Data Warehouse part of the lakehouse architecture, there are three layers defined: manual entry area, base layer, and performance & analytics layer [30]. Here the data previously ingested in the Data Lake part is loaded and integrated through ETL processes. To provide access to the data stored both in the Data Lake and Data Warehouse parts, it is necessary to incorporate a virtualization layer [20].

Databricks Delta Lake, the AWS data lakehouse, Azure data lakehouse, and Oracle data lakehouse are real-world examples of the Lake House Architecture. Databricks creates a unified data lake and data warehouse platform utilizing Apache Spark, a powerful open-source data processing technology.

It is possible to observe in figure 3.5 a proposal of a data lakehouse architecture :

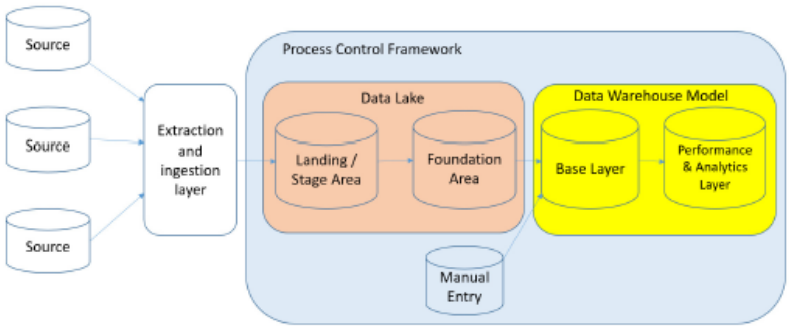

Figure 3.5: Data Lakehouse Architecture [30]

3.4.1 The Challenges of Data Lakes

Despite having become one of the most popular solutions for storing and processing large volumes of data in recent years, implementing and maintaining a data lake can also bring some challenges for organizations that literature can not solve due to existing research gaps.

1. **Complexity of Data Lake Management**: One big challenge is the complexity of managing a data lake. A data lake often involves a variety of technologies and tools, such as data ingestion, storage, processing, and access technologies. The complexity comes from the need

to integrate these different components to ensure they work well together. Organizations must have specific skills and knowledge to manage this challenge and ensure that the data lake performs as desired.

2. **Data Integration Challenges**: The issue of data integration poses another challenge. A data lake frequently involves integrating data from several sources and formats, which may be complicated and take a lot of time. To ensure that data is correctly integrated and can be used successfully in the data lake, organizations must invest in data integration technologies and procedures.

3. **Lack of Comprehensive Implementation Strategies**: A comprehensive strategy for designing and implementing data lakes is also lacking, posing another challenge [17]. While there are numerous architectural conceptions, there is frequently a lack of agreement on best practices. Organizations must define a strategy that matches their specific demands and goals.

4. **Diversity in Data Lake Architectures**: The various and different concepts related to data lake architectures also pose a challenge, as there are no assessments or discussions of the numerous alternatives that exist [17]. There is no generally accepted solution for a data lake architecture, normally it is chosen based on the organization or personal needs [31]. Furthermore, data lake architectures only address the conceptual arrangement of data, they lack some other data lake features, such as data lake modeling or data lake infrastructure [17]. A generic and complete data lake architecture is required to implement data lakes [31, 17].

5. **Data Governance Imperatives**: One of the most important aspects in a data lake seems to be data governance [31, 23, 20]. The need for data governance has even caused some organizations to develop a new job position known as "Chief Data Officer" (CDO) to regulate and oversee the company's data [23]. Data in a data lake is frequently ingested from a number of sources and in a variety of forms, making it challenging to establish clear ownership and accountability for the data. Since data governance addresses the quality and security of the data [31], by not meeting the data management demands, the data lake may be turned into a data swamp, a repository of noisy data that shall have little to no use to organizations [20]. As a result, comprehensive data governance practices are needed for data lakes [17, 31].

Chapter 4

Technologies used in Data Lakes

In this chapter, essential technologies that constitute data lakes, allowing them to ingest, store, process, and analyze massive and varied data, are discussed. The success of any data lake project is strongly dependent on the selection and application of these technologies. Throughout this chapter, all of the key elements and technologies that make up the data lake ecosystem will be analyzed.

Beginning with the analysis of fundamental data ingestion technologies, which will serve as the entry point through which data enters the lake, the analysis will include batch processing as well as real-time streaming, offering insight into the techniques used for seamlessly gathering data from various sources. Following that, some storage solutions are examined, enabling data lakes to efficiently manage the information they house. These options include distributed file systems and object stores.

However, it is essential to understand that data lakes are more than simply data repositories; they are analytical powerhouses. To realize this analytical potential, an investigation of data processing methods, will be carried out. These tools enable organizations to get important insights and detect trends in their data lakes.

From subsections 4.1 to 4.4, several technologies will be presented and their role in the process of building and maintaining a data lake will be described.

4.1 Data Ingestion Technologies

These types of technologies are an essential component, as they are used to move data from physical locations into data lakes [31]. These solutions enable the development of data pipelines for effective data transfer and give organizations quick and simple access to huge amounts of data for analysis. Several open-source data ingestion solutions, such as **Apache Flume and Apache NiFi**, are available. Both of these solutions offer the integration of both structured and unstructured data and are built for scalable and reliable data ingestion. In addition to open-source tools, there are also several commercial data ingestion platforms available, such as Talend, Informatica, and StreamSets.

Table 4.1: Apache Flume: Advantages and Disadvantages

Apache Flume	
Advantages	**Disadvantages**
Efficient log and event data collection. Scalable and extensible architecture. Supports various data sources.	May require configuration for complex data flows. Limited real-time processing capabilities.

Table 4.2: Apache NiFi: Advantages and Disadvantages

Apache NiFi	
Advantages	**Disadvantages**
Powerful data routing and transformation capabilities. User-friendly, web-based interface. Extensive library of processors for data integration.	Can be resource-intensive for large data volumes. Learning curve for complex data flows.

The choice of a data ingestion tool will, in the end, be based on the particular demands and specifications of the organization, including the types of data being ingested, the volume of data, and the required performance and scalability. It is crucial to thoroughly weigh the options and choose the tool that best suits the organization's requirements.

4.2 Data Storage Technologies

As the data lake must be able to handle large volumes of data, often structured and unstructured, and support fast access and processing of the data, it is important to choose carefully the best choice for data storage.

It is possible to store data in data lakes in two ways: through the use of a distributed file system like Hadoop Distributed File System (HDFS) or object storage solutions like Amazon S3 [6]. These file systems are designed to store large volumes of data across multiple nodes, providing both scale and fault tolerance. Data lake architectures frequently use the Hadoop Distributed File System (HDFS) to store their data. There is also an offering from Amazon Web Services (AWS), Amazon S3 [6] which is a very scalable and robust object storage solution that is often used as a data storage layer for data lakes built using the AWS platform.

Table 4.3: Distributed File Systems (e.g., HDFS): Advantages and Disadvantages

Distributed File Systems (e.g., HDFS)	
Advantages	**Disadvantages**
High fault tolerance due to data replication.	Complex to set up and manage.
Scalable for large data volumes.	May not be cost-effective for small-scale deployments.
Designed for batch processing and can handle structured and unstructured data.	Performance can degrade for small files.

Object storage systems, such as Amazon S3, provide scalability and cost-effectiveness, particularly in cloud-based data lake systems. They supports a wide range of data types and formats and have a high level of durability and availability. They may, however, lack integrated data processing capabilities, requiring the use of extra data management tools. They are often used as data storage layers for data lakes, especially when hosted on cloud platforms such as AWS.

Table 4.4: Object Storage (e.g., Amazon S3): Advantages and Disadvantages

Object Storage (e.g., Amazon S3)	
Advantages	**Disadvantages**
Scalable and cost-effective, particularly for cloud-based data lakes.	Limited built-in data processing capabilities.
Supports various data types and formats.	May require additional data management tools.
Highly durable and available.	

Alternatively, relational databases (e.g., MySQL, PostgreSQL) or NoSQL databases (e.g., MongoDB, Cassandra) may be utilized [31]. Relational databases, such as MySQL and PostgreSQL, are excellent at maintaining structured data. They provide good data consistency and ACID (Atomicity, Consistency, Isolation, Durability) transactions, making them suited for complex query applications. They are, however, unsuitable for huge amounts of unstructured data and have limited scalability for tasks that require many reads. Furthermore, schema modifications might be difficult to implement.

Table 4.5: Relational Databases (e.g., MySQL, PostgreSQL): Advantages and Disadvantages

Relational Databases (e.g., MySQL, PostgreSQL)	
Advantages	**Disadvantages**
Excellent for structured data.	Not suitable for large-scale unstructured data.
Strong data consistency and ACID compliance.	Limited scalability for reads.
Support for complex queries.	Schema changes can be challenging.

Since relational databases are not efficient for the storage of semi-structured or unstructured data, NoSQL databases are most often used for storage in data lakes [31]. NoSQL databases such as MongoDB and Cassandra do best at handling semi-structured and unstructured data. They provide horizontal scalability for large datasets as well as schema flexibility for design. However, their consistency mechanism may not be ideal for all use cases, and its scalability for tasks that require many reads may be a constraint. Complex queries might require the inclusion of extra processing layers.

Table 4.6: NoSQL Databases (e.g., MongoDB, Cassandra): Advantages and Disadvantages

NoSQL Databases (e.g., MongoDB, Cassandra)	
Advantages	**Disadvantages**
Ideal for semi-structured and unstructured data.	Eventual consistency may not be suitable for all use cases.
Horizontal scalability for massive datasets.	Limited scalability for read-heavy workloads.
Flexible schema design.	Complex queries may require additional processing layers.

The choice of data storage technology depends on the organization's specific requirements, including data types, volume, performance, and scalability. Careful evaluation and selection of the most suitable technology are essential.

4.3 Data Processing Technologies

The processes of transformation and analysis of large volumes of data stored in the data lake are only possible with the use of data processing technologies. Data transformation, data integration, and data analysis are just a few of the many tasks that these technologies support. One commonly used data processing technology in data lake architectures is **Apache Spark**, which is an open-source, distributed computing solution that allows for in-memory data processing. It is intended to be quick and efficient, and it can be used for a variety of data processing tasks, such as batch processing, stream processing, machine learning, and graph processing.

Table 4.7: Apache Spark: Advantages and Disadvantages

Apache Spark	
Advantages	**Disadvantages**
In-memory processing for faster analytics.	
Supports batch, streaming, machine learning, and graph processing.	May require substantial memory resources.
	Hard to learn for new users.
Strong community support and extensive libraries.	

MapReduce, part of the Hadoop ecosystem, is another data processing technology that is commonly used in data lake architectures. MapReduce is a programming model developed by Google for processing massive datasets and backed by the Hadoop ecosystem. It provides parallel data processing over a distributed cluster and is frequently used in data lake batch processing and data transformation operations. However, it is inefficient in terms of handling fast data, and that is where Spark comes in to solve this problem thanks to its in-memory processing method [31].

Table 4.8: Hadoop Map Reduce: Advantages and Disadvantages

Hadoop Map Reduce	
Advantages	**Disadvantages**
Effective for batch processing on large datasets. Suitable for fault-tolerant processing. Mature and widely adopted.	Slower for iterative algorithms. Requires more coding effort for certain tasks.

Apache Flink [8] is another significant data processing tool. Flink is an open-source stream processing platform for real-time data processing. It is fault-tolerant and scalable, has a low latency stream processing and it supports both batch and stream processing.

Table 4.9: Apache Flink: Advantages and Disadvantages

Apache Flink	
Advantages	**Disadvantages**
Real-time and batch processing support. Low-latency stream processing. Fault-tolerant and scalable	Smaller community compared to Spark and Hadoop. Limited ecosystem of libraries.

In addition to these open-source technologies, commercial data processing systems such as **Amazon EMR** and **Google Cloud Data Fusion** are also available. These technologies offer managed services for data processing pipeline implementation and management, making it easier for organizations to get started with data processing in the data lake.

Table 4.10: Amazon EMR: Advantages and Disadvantages

Commercial Data Processing Services (e.g., Amazon EMR)	
Advantages	**Disadvantages**
Managed services for simplified deployment. Integration with other cloud services. Pay-as-you-go pricing.	May incur ongoing operational costs. Limited control compared to self-hosted solutions.

The volume and complexity of the data being processed, the desired performance and scalability, and the need for real-time processing will be key features when choosing a data processing technology, and it depends on the organization to choose the solution that best fits its needs.

4.4 Data Access Technologies

Data access technologies are used in a data lake architecture to allow users to query and analyze the data contained in the data lake. These technologies, which may include SQL-based query languages, data visualization tools, and business intelligence platforms, allow people to engage with data and extract insights.

Apache Hive is a popular data access solution in data lake architectures. Hive is an open-source data warehouse and SQL-like query language for HDFS data. It enables individuals to use queries similar to SQL to do data analysis and querying on massive datasets stored in the data lake.

Table 4.11: Apache Hive: Advantages and Disadvantages

Apache Hive	
Advantages	**Disadvantages**
Provides SQL-like query language for data analysis. Suitable for querying massive datasets in data lakes. Open-source and widely adopted in Hadoop ecosystems.	May require SQL expertise for effective use. Performance can be a concern for complex queries on large datasets.

Apache Pig is another popular data access technology. Pig is an open-source data processing platform that allows users to write complicated data transformations in Pig Latin, a high-level language and it can be used to clean and prepare data for data lake analysis.

Table 4.12: Apache Pig: Advantages and Disadvantages

Apache Pig	
Advantages	**Disadvantages**
Facilitates complex data transformations through Pig Latin. High-level language for data preparation and cleansing. Open-source and flexible.	Requires familiarity with Pig Latin for effective usage. May not be suitable for all data analysis tasks.

Apache Superset is being recognized as a vital addition to the data access toolset. It is an open-source data exploration and visualization platform with a user-friendly interface for creating

interactive dashboards and viewing data from a variety of sources, including data lakes. The flexibility and scalability of Apache Superset make it an excellent choice for organizations looking to improve their data access and analysis capabilities.

Table 4.13: Apache Superset: Advantages and Disadvantages

Apache Superset	
Advantages	**Disadvantages**
User-friendly interface for data exploration and visualization. Supports interactive dashboards and visualizations. Flexible and extensible open-source platform.	May require integration with other data tools for full functionality. Smaller community compared to some commercial solutions.

In addition to these technologies, commercial data visualization and business intelligence solutions such as **Tableau, QlikView, and Microsoft Power BI** are also available.

Table 4.14: Commercial Data Visualization Tools: Advantages and Disadvantages

Commercial Data Visualization Tools (e.g., Tableau, QlikView, Microsoft Power BI)	
Advantages	**Disadvantages**
Advanced visualization capabilities for data exploration. Interactive dashboards and reporting features. Support for various data sources and formats.	Commercial licenses may involve significant costs. Learning curve for mastering advanced features.

The types of data being accessed, the complexity of the data, and the desired level of interactivity and visualization will dictate the technological solution to choose for data access according to the organization's needs.

Chapter 5

Design

This chapter plays an essential role in understanding the complexities of the data lake module. It explores deeply into the design and decisions taken during this process contributing to a better understanding of the data lake and its components. In the era of big data, the architecture and design of data management systems, such as data lakes, are of the utmost significance. A carefully designed architecture serves as the blueprint for a powerful and efficient data lake module, allowing for the smooth flow of data, processing, and analysis.

5.1 Design Principles

As part of the development of the Data Lake module, software design principles have played an important role in shaping its reliability, maintainability, and overall quality. These principles served as a guide for architectural choices and considerably increased the software's reliability. The main software design ideas that guided the creation of the module are further developed in this section.

1. **DRY (Don't Repeat Yourself)**

   The DRY principle highlights the significance of reusable code. By encapsulating common functionalities into reusable modules and libraries, code redundancy is reduced. This procedure eliminates code duplication and makes updating and maintenance easier.

2. **KISS (Keep It Simple, Stupid)**

   KISS stands for keeping things simple, both in terms of design and execution. Simple, clear solutions were preferred to complex ones to make the codebase easier to maintain and less susceptible to errors.

3. **YAGNI (You Ain't Gonna Need It)**

   YAGNI advises developers to hold off on implementing new features unless they are genuinely required. By following this rule, engineering too complex is avoided and the module's primary goals are kept in mind.

4. **Single Responsibility Principle (SRP)**

   The single responsibility principle dictates that a class should have only one responsibility and not more than one reason to change. In this data lake project, every method follows SRP by performing a specific action, thus enhancing code clarity and maintainability.

5. **Inversion of Control (IoC)**

   In software design, Inversion of Control (IoC) transfers dependency management responsibility to a single container. IoC ensures loose coupling between classes, improving flexibility, extensibility, and testability. By managing dependencies, the IoC container makes the code more modular, manageable, and adaptable to change. This enhances the overall quality and dependability of the software system.

5.2 System Architecture

This section provides a fundamental view of the data lake's structure and how its key components interact. It includes a diagram that offers a high-level visualization of the system's design and the way its essential parts work together. This diagram acts as a guide to help us understand the system's technology, processes, and data flows. This analysis provides the means to understand how the system manages large amounts of data and supports data analysis and decision-making.

The data lake is structured into distinct layers, each with well-defined responsibilities contributing to the overall functionality and effectiveness of the data lake ecosystem. Each architectural layer will be explored, providing an understanding of the data flow and how data is ingested, stored, processed, and made accessible for various analytical and business needs.

5.2.1 Data Ingestion Layer

The Data Ingestion Layer is the initial entry point for data into the data lake module. It is crucial for efficient ingestion of diverse data sources, typically from relational databases or CSV files. Apache Spark plays a crucial role in handling data, facilitating extraction from different sources, transforming it as needed, and transferring it to the data lake storage layer. Spark's batch processing capability ensures efficient ingestion and preparation of large volumes of data for subsequent processing and analysis.

5.2.2 Data Processing Layer

The Data Processing Layer is crucial for transforming, cleaning, and analyzing raw data in the data lake. Spark, known for its batch and streaming processing capabilities, is used in this layer for data processing, machine learning, and insights. Its versatility and scalability make it an essential component in managing the massive data in the data lake.

5.2.3 Data Storage Layer

In the data storage layer, data comes from the ingestion layer where is safely and effectively stored to allow for quick access and retrieval. Data also flows to and from the data processing layer. Hadoop Distributed File System (HDFS) is the primary storage technology used. HDFS delivers the resilience and scalability required to manage huge amounts of data and is explored further in subsection 5.3.3. It can store data in various formats such as CSV (comma-separated values) and parquet, a columnar storage file format that is often used in the context of big data and data lakes for efficient columnar storage.

Furthermore, HDFS integrates with Apache Spark, the chosen data processing engine, increasing the efficiency of the data lake ecosystem. This integration enables Spark to directly access HDFS data, avoiding the need for complicated data transfers and ensuring data is quickly available for processing.

5.2.4 Data Access Layer

The Data Access Layer acts as the user interface for interacting with the data in the data lake. In this layer, Apache Superset, an efficient data visualization and business intelligence tool, takes center stage. Superset enables users, such as data scientists and business experts, to easily access and view data. Superset's user-friendly interface makes it possible to create interactive dashboards, execute queries, and derive actionable insights from data stored in HDFS and analyzed by Spark. This layer connects raw data to relevant data, enabling decisions to be based on valuable data throughout the company.

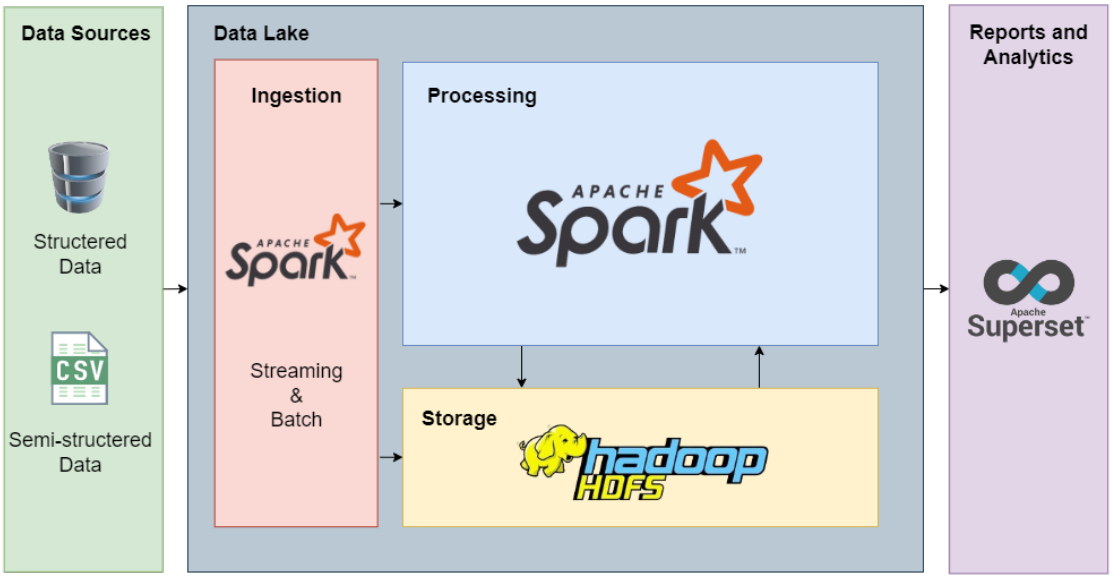

Figure 5.1: Data Lake Architecture

5.3 Technology Stack

This section describes and analyzes the diversified technology stack that powers the data lake design. These technologies were carefully selected to meet the company's specific needs and preferences, focusing on cost-effectiveness, scalability, and the ability to accommodate a wide variety of data types and workloads. This section builds on chapter 4, as it takes the comparison made in that chapter and explores more the technologies that were used in the data lake. Beginning with Postgres databases and progressing through HDFS, Apache Spark, and Apache Superset, an analysis of each technology is performed, detailing how it operates in the data lake and the benefits it provides.

5.3.1 Software Requirements

Before getting into the various technologies, it's critical to highlight the company's initial software requirements. The following requirements guided the selection process:

1. **Spark**: Since the beginning of this project, one of the certainties was the use of Apache Spark as the main technology for data processing in the data lake. Although the company had never used this tool before, they were aware of its high-speed data processing capabilities and comprehensive support for a variety of data sources and large data volumes in both batch and real-time. Consequently, they were interested in gaining first-hand experience with this tool and using it within the data lake.

2. **HDFS On-Premises Data Storage over Cloud Solutions**: The choice to use the Hadoop Distributed File System (HDFS) on-premises was largely motivated by economic reasons. HDFS, being the most used open-source technology when it comes to data lake implementation, makes it the most logical choice for data storage. By hosting HDFS in local infrastructure, cloud service fees are eliminated while maintaining complete control over our data storage environment. This decision, allows us to optimize the value of our data lake implementation while staying within budget limits.

3. **Superset Over Power BI for Data Visualization**: During the search for an efficient data visualization and exploration tool that is cost-effective, one solution stood out for its advanced features and affordability. Apache Superset emerged as the clear winner in this selection process, owing to its open-source nature and high level of customizability that perfectly aligns with the requirements of the company.

5.3.2 Apache Spark

Apache Spark is the main technology in both the Data Ingestion and Data Processing layers of the data lake architecture, where it plays an important role in acquiring, transforming, and processing data from diverse sources. This section provides an analysis of Apache Spark's significance, functionality, and contributions to both layers of our data lake ecosystem.

Spark is an open-source distributed computing framework that revolutionized big data processing. Internet giants like Netflix, Yahoo, and eBay have used it at scale. Spark has grown into the largest open-source community in big data, with 1000+ contributors from 250+ enterprises [12]. It was developed as a faster and more adaptable alternative to Hadoop MapReduce. To understand Spark's vital role in data lake design, one must first understand its fundamental architecture, capabilities, and deployment methods.

Cluster Computing

Spark runs in a distributed cluster computing environment, where data and processing are distributed effectively over a cluster of machines. This design combines the computing capacity of several nodes, resulting in fast data processing. A typical Spark cluster is made up of two parts:

- **Master node**: The master node is in charge of coordinating tasks, scheduling jobs, and managing the overall execution of Spark applications. It keeps track of critical information about the cluster's status and resource availability, ensuring that concurrent Spark applications are allocated resources efficiently.
- **Worker nodes**: Also known as slaves, they are in charge of carrying out tasks set by the master node. Spark executors run tasks and store data in memory on each worker node. Communication between worker nodes and the master node ensures that Spark operations run smoothly.

Resilient Distributed Dataset (RDD)

Apache Spark uses the Resilient Distributed Dataset (RDD) programming model, which represents an immutable collection of objects that can be distributed and executed in parallel across a computing cluster. RDDs hide most of the computational complexity, and offer benefits like resilience, distribution, and dataset-like features, making them essential for Spark [13].

Spark's Architecture

Apache Spark's architecture revolves around a distributed computing model designed for the efficient handling of extensive data sets. Its components work in concert to deliver robust performance:

At the core is the **Driver Program**, acting as the brain of a Spark application. It manages tasks distributed across the cluster and defines the application's control flow. The **Cluster Manager**, supporting various managers like Apache Mesos, Hadoop YARN, and its standalone cluster manager, oversees resource management and monitoring, ensuring efficient resource allocation. **Executors**, situated on worker nodes, are responsible for data storage, computation execution, and concurrent task handling. They maintain a direct line of communication with the driver program. In YARN cluster mode, an **Application Master** effectively manages tasks and optimizes resource

scheduling. **Worker Nodes**, comprising individual machines, execute tasks and store data, forming the core of Spark's architecture. This collaborative, distributed framework empowers Spark to handle vast data sets efficiently, making it a versatile choice for data-intensive applications across various domains.

Figure 5.2 depicts the spark's architecture and all of its components described above giving perspective on how data flows.

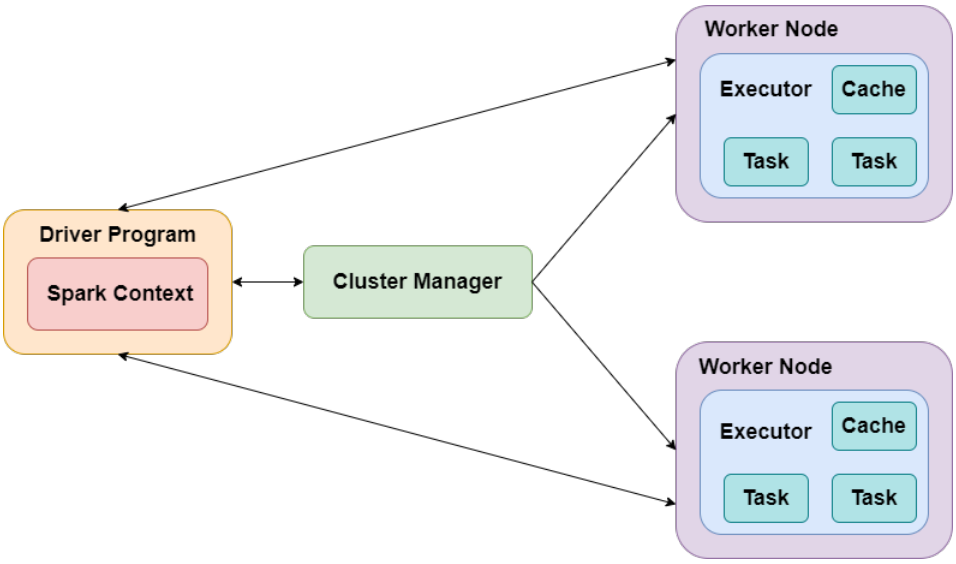

Figure 5.2: Spark's Architecture

Deployment Modes

Apache Spark offers multiple deployment modes to support different use cases:

- **Standalone Mode:** Spark works in standalone mode using its integrated cluster manager. This mode is appropriate for development, testing, and smaller workloads.
- **Apache Hadoop YARN (Yet Another Resource Negotiator):** Spark can easily connect with a Hadoop YARN cluster, which manages resources and schedules jobs, leveraging Hadoop's powerful capabilities. This mode is especially beneficial for companies with existing Hadoop clusters.
- **Apache Mesos:** Spark may be deployed using Apache Mesos, a general-purpose cluster manager. This mode is appropriate in circumstances where numerous distributed applications coexist and efficiently share resources.

Versatile Processing and In-Memory Computing

Spark is a versatile platform for data processing, supporting **batch processing, real-time data streaming**, **machine learning (MLlib)**, and **graph processing (GraphX)**. Its in-memory computation capability caches data in memory, significantly improving performance, speed, and efficiency in data input and processing.

Ease of Use, Community, and Scalability

Apache Spark offers high-level APIs in **Java, Scala, Python, and R**, with built-in SQL libraries for easier data searching and **MLlib** for complex machine learning tasks. The Spark ecosystem includes **Spark SQL** for structured data querying and **Spark Streaming** for real-time data processing. With its scalability and robust open-source community, Apache Spark is a valuable solution for big data processing.

The diagram in **Figure 5.3** illustrates the Spark ecosystem, providing a clear visualization of the programming languages, components, and cluster modes discussed in this section.

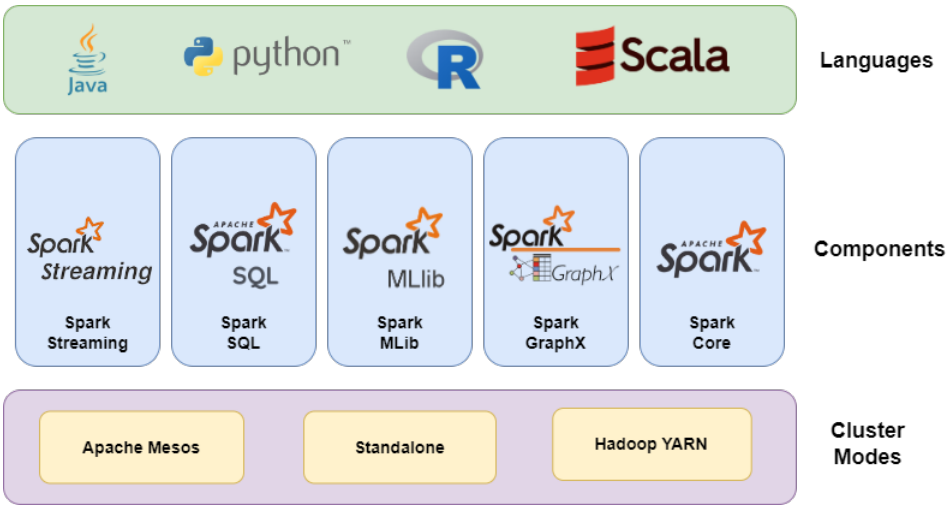

Figure 5.3: Spark Ecossystem

As a powerful tool, Spark is used worldwide in the development of a wide range of applications. A real-world use case is **Netflix**. Spark is the recommendation engine used by Netflix. Netflix analyzes user activity and preferences with the use of Spark's machine-learning skills to suggest movies and TV series [34].

5.3.3 Hadoop

The capacity to efficiently and securely store massive amounts of data is at the heart of any data lake design. The Hadoop Distributed File System (HDFS), part of the Hadoop Ecosystem, serves as the storage layer of the data lake. This subsection explores how Hadoop and HDFS work, explaining their fundamental position in the data lake, its underlying components, data storage processes, and unique advantages.

What is Hadoop?

Apache Hadoop is a Java-based open-source, scalable, and fault-tolerant platform. It processes enormous amounts of data effectively using a cluster of commodity (low-cost) hardware. Hadoop is not only a storage system, but also a data storage and processing platform. It provides an efficient framework for performing tasks over numerous cluster nodes. A cluster is a group of

computers that are linked together via LAN (Local Area Network). Apache Hadoop allows for concurrent data processing on multiple machines at the same time.

Hadoop components

Hadoop can be configured on just one machine, but its true potential emerges when a cluster of machines is used. It can grow in real-time to thousands of nodes without any downtime. Hadoop is made up of three major components:

- **Hadoop HDFS:** Hadoop Distributed File System (HDFS) is the storage unit of Hadoop.
- **Hadoop MapReduce:** Hadoop MapReduce is the Hadoop processing unit. This software framework is designed to create apps that can handle massive volumes of data [2].
- **Hadoop YARN :** Hadoop YARN is a resource management component of Hadoop. It processes and executes data for batch, stream, interactive, and graph processing, all of which are stored in HDFS [2].

How does Hadoop work?

Hadoop adopts a Primary-Secondary architecture. There is a primary node and n slave nodes, where n might be more than 1000. The Primary controls, maintains, and monitors the slaves, whilst the slaves do the real job [29]. In the Hadoop architecture, Primary must be implemented on good hardware as it is the centerpiece of the Hadoop cluster. The Primary maintains metadata (information about data), whereas slaves are nodes that store data. To execute any action, the client application connects to the primary node.

1. The client application sends the data to Hadoop, which then divides the data into blocks of size 128 Mb (by default). Then the data is sent by HDFS to different nodes in the cluster.
2. After all file blocks have been put on datanodes, the data may be processed by the user.
3. When a MapReduce (or Apache Spark) processing job is initiated, the primary schedules the job (provided by the user) on particular nodes.
4. The output will be pushed back to HDFS once all nodes finish processing the data. The metadata is stored by the Namenode daemon, while the actual data is stored by the Datanode daemons.

In simple terms, clients send Hadoop data and programs. HDFS manages metadata and the distributed file system. The input/output data is then processed and converted by Hadoop MapReduce. Finally, YARN distributes jobs across the cluster.

Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) is a distributed file system designed to store and handle massive amounts of data on commodity hardware clusters. It is an essential component of the Apache Hadoop ecosystem, enabling companies to handle and analyze massive information. HDFS is the technology that serves as the storage layer in this data lake project because of its many features and advantages that will be discussed further down.

Essential Components of HDFS

HDFS is composed of two main components: the **NameNode** and multiple **DataNodes**.

- **NameNode**: HDFS has a master server called NameNode that manages the file system namespace and tree structure. It keeps track of metadata such as file names, permissions, and hierarchy, while the actual data is stored in DataNodes. This design enables scalability and fault tolerance.
- **DataNode**: DataNodes are worker nodes responsible for storing and managing data blocks. They manage data storage, retrieval, and replication. They respond to read and write requests and report back to the NameNode about the health and status of data blocks, assuring data integrity and availability. Data is partitioned into fixed-size blocks and spread over several DataNodes for redundancy and parallel access.

Figure 5.4, showcases the HDFS architecture and the interactions among its different nodes, providing a comprehensive overview of its structure and functionality.

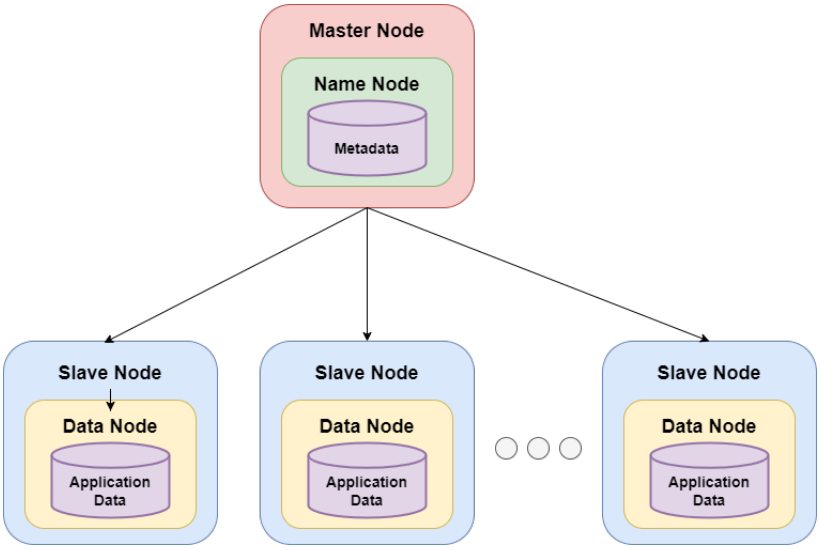

Figure 5.4: HDFS Base Architecture

Data Storage in HDFS

HDFS employs a unique approach to data storage that is distinct from traditional file systems. It follows a block-based storage as it stores data in fixed-size blocks. This design choice, as opposed

to storing data in variable-sized blocks or files, brings significant benefits for data management within the file system in terms of scalability, fault tolerance, and data processing.

Data Replication for Fault Tolerance

HDFS puts great importance on data fault tolerance. To guarantee resilience against hardware failures or data corruption, data is duplicated over many DataNodes. Each data block is typically replicated three times by default, however, this replication factor is configurable. If one copy fails, the data may still be recovered from the other copies, guaranteeing high availability.

Write-Once, Read-Many Model

HDFS follows a write-once, read-many approach, which is ideal for cases in which data is mostly written once and then read several times. This concept is consistent with many big data use cases in which data is intensively collected, processed, and analyzed but is rarely updated once stored.

Advantages of HDFS

The use of HDFS in the data lake architecture comes from several significant benefits that make it particularly appropriate according to the requirements:

1. **Scalability**: HDFS has excellent scalability in terms of both data volume and node growth. As the storage requirements of the data lake expand, it is possible to effortlessly add more storage nodes to the cluster, guaranteeing that it is capable of handling even the most significant data growth.

2. **Fault Tolerance**: The fault tolerance mechanisms built into HDFS, such as data replication and the separation of metadata and data, ensure that our data remains available even in the event of hardware failures or data corruption.

3. **Cost-Effectiveness**: By using HDFS, we benefit from having access to a cost-effective storage option. HDFS is open-source and free to use, thus there are no license fees like there are with other commercial storage systems.

4. **Support for Diverse Data Types**: Given that HDFS can store data in its raw, unstructured form, it is perfect for supporting a variety of data types and formats, whether structured, semi-structured, or completely unstructured.

When it comes to Hadoop, the possible use cases are almost endless [3]. For instance, retailers like **Marks & Spencer (M&S)** use it for predictive analytics and inventory optimization. **JPMorgan Chase** relies on it for portfolio management and risk modeling. Healthcare providers use Hadoop to simplify access to patient data, and institutions like **Harvard University** use it for academic research, including genetic studies.

5.3.4 Apache Superset

This section explores the significance of data visualization, the functions of Apache Superset, its features, and how it enhances the capabilities of the data lake.

Data visualization plays an important role in converting raw data into meaningful insights. It is crucial for making informed decisions and effective communication. It helps users identify trends, patterns, and anomalies, making it essential for data professionals.

Apache Superset is an open-source data exploration and visualization framework that complements the goals of the data lake. Its primary features and benefits make it an excellent alternative for improving data access and visualization:

- **Rich Visualization Capabilities**: Superset provides a wide range of visualization choices, such as interactive charts, graphs, heatmaps, and geographic visualizations. Users can easily develop interacting dashboards, allowing for dynamic study of data from several perspectives.
- **Ease of Use**: The user-friendly interface of Superset ensures that people with varied degrees of technical experience may leverage its potential. Its simple drag-and-drop interface makes generating and modifying visualizations simple and straightforward.
- **Data Source Flexibility**: Apache Superset easily connects to a wide range of data sources. It supports a wide range of databases, data warehouses, and file formats, making it an adaptable tool for accessing and displaying data stored throughout the data lake.
- **Interactive Dashboards**: Superset's dynamic dashboards are one of its most notable features. Directly from the dashboard, users may examine data, apply filters, and dive down into specifics. This feature allows users to interact with data in real-time and draw valuable insights.
- **Community-Driven Development**: Because Apache Superset is open source, it has an active community of developers and users. This user-driven approach offers continual improvement, regular updates, and a variety of tools for users in need of assistance.

Some of Superset's most famous real-world use cases include **Airbnb**. Apache Superset was developed by Airbnb's data team and continues to be widely used across the company [22]. Superset assists Airbnb's data analysts and decision-makers in developing interactive dashboards and gaining knowledge about market trends, guest behavior, and host performance.

5.3.5 Spring Boot

The data lake module is built on top of the Spring Framework, which provides a comprehensive programming and configuration model for Java-based applications. By using the Spring Framework, developers can develop loosely coupled, highly maintainable, and scalable systems, promoting the use of best practices in the development of software.

Spring Boot, a Spring framework extension, is a flexible and efficient platform for developing stand-alone, production-grade Spring-based applications [5]. Spring Boot is the essential technology for developing and delivering the data lake solution. It extends the Spring Framework and provides conventions, auto-configuration, and production-ready defaults to minimize code and configuration. Its goal is to accelerate the application development and deployment process.

Some of the key features of Spring Boot are described below:

- **Rapid Development**: Spring Boot facilitates development by providing a set of conventions and auto-configuration features. This makes it possible for developers to quickly create powerful applications without the need for excessive repetitive code. This rapid development capacity speeds up the implementation process in the data lake project.
- **Embedded Containers**: With Spring Boot, applications can be packaged and run as standalone executables using embedded containers (such as Tomcat and Jetty). A feature like this is particularly valuable for architectures such as data lakes, where deployment and scaling are important aspects to consider.
- **Spring Ecosystem Integration**: Spring Boot interacts easily with other Spring projects, making it easier to leverage various Spring capabilities and libraries. This integration ensures consistency when developing the data lake module and also makes it easier to integrate the data lake module into other spring-based projects.

The data lake is built using Spring Boot and follows a microservices architecture, promoting modularity, scalability, and flexibility. Various microservices handle data ingestion, processing, storage, and access. In addition to simplifying the integration of Apache Spark for data processing, Spring Boot ensures seamless coordination and control of Spark jobs within the workflow of the data lake.

Figure 5.5, depicts the many technologies that form this data lake's technological stack.

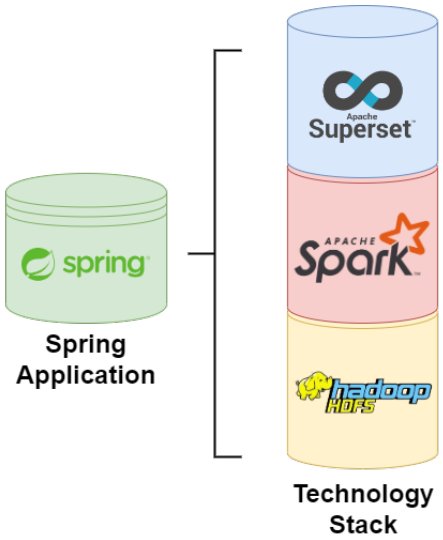

Figure 5.5: Techonology Stack

Chapter 6

Implementation

In the chapter on implementation, the emphasis switches from design theory to actual implementation details. This chapter explores into code structures, configurations, and the operational aspects of each layer and component, providing a clear perspective of the Data Lake module's functionality.

6.1 Development Environment

This section provides an overview of the fundamental components that formed the basis of the data lake module's development. It includes the decision on development tools, version control procedures, hardware setups, and operating systems. Topics such as the testing environment, agile development methodology, and challenges encountered are discussed. This overview sets the stage for a deeper examination of the module's architecture, implementation, and technological components in subsequent sections.

Development Tools

The project was mostly developed using the Java programming language, which was chosen not only for its versatility and strong ecosystem support, but also because it is the primary language used by the company. IntelliJ IDEA was the preferred Integrated Development Environment (IDE), providing an intuitive and efficient coding environment. Furthermore, command-line interfaces were used to configure certain components of the development environment. The Apache Superset environment was configured using an Ubuntu-based command line, while Hadoop and Spark components were set up using the Windows command prompt.

Version Control

Effective version control is essential in software development, even in solo projects. Git was selected as the version control solution for this project. This choice enabled the effective administration of source code changes, tracked development progress, and guaranteed that each important code change or feature addition was adequately documented and versioned. Git repositories were critical to ensuring code integrity throughout the project.

Operating Systems

The development process was spread across many operating systems. Windows environments were mostly utilized for coding and testing. The Apache Superset environment, on the other hand, required the use of an Ubuntu-based command-line interface. This adaptable strategy enabled the use of alternative operating system strengths for certain development tasks, allowing flexibility throughout the development process.

Development Methodology

The project was structured around Agile development principles, focusing on iterative development and responsiveness to evolving requirements.

Testing Environment

The development process included robust testing techniques. Unit and integration testing were carried out within the IntelliJ IDEA environment, using industry-standard technologies such as JUnit and Mockito. Postman, a powerful API testing tool, was essential in thoroughly validating the functionality and reliability of API endpoints. This rigorous testing technique guaranteed the module's resilience and compliance with functional requirements.

Deployment Environment

The Data Lake module is designed for use on-premises on local computers. However, progressive containerization ideas employing Docker containers have begun. This strategic choice sets the project for potential future deployment scenarios such as cloud-based or containerized systems, which improves scalability and deployment flexibility.

Dependencies

Within the Spring Boot application, Maven, a powerful build automation tool, effectively managed project dependencies. This automatic dependency management guaranteed that all necessary libraries, frameworks, and components were incorporated properly into the project. Maven led to a more efficient development workflow by making it easier to manage external dependencies.

6.2 Code Structure

This section discusses the importance of a well-structured codebase in software development. The code structure affects a project's maintainability, readability, and collaboration among developers. Here, the chosen code structure is explained, and how it helps the project's objectives while adhering to best practices. The analysis shows how smart code architecture contributes to the project's success and longevity.

6.2.1 Package Structure

The package structure of a software project is critical for organizing and managing the project's source code and resources. It establishes the hierarchical organization of directories and packages, grouping relevant classes, interfaces, and resources.

Figure 6.1 illustrates this project's package structure highlighting the organization of classes throughout the project. Each package and its contents will be described further giving a better understanding of how this project is organized.

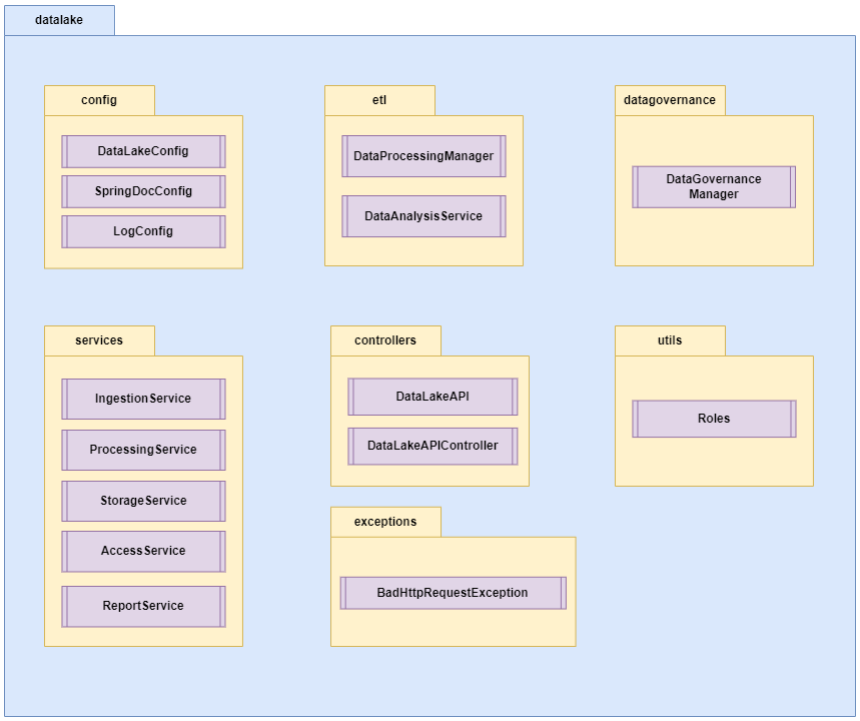

Figure 6.1: Data Lake's Package Diagram

- **datalake:** This package serves as the core of the data lake module, containing all the essential components for data ingestion, storage, processing, and access.
- **config:** The config package stores configuration files and settings necessary for the proper functioning of the data lake module, such as API documentation, Spark, and logging configuration.
- **etl:** The etl (Extract, Transform, Load) package encapsulates data transformation and loading processes. It is responsible for storing classes related to data analysis and processing, such as the classes responsible for dataset analysis presented further in chapter 7.
- **datagovernance:** Within the datagovernance package, data governance and metadata management functionalities are implemented to maintain data lineage and quality.
- **services:** The services package contains all the service classes responsible for handling

various operations within the data lake, such as data ingestion, storage, processing, and access. This is where all the layers described in section 5.2 are implemented.

- **controllers:** This package encompasses controllers that expose RESTful API endpoints, enabling external users to interact with the data lake.
- **utils:** The utils package provides utility classes and functions that assist in various internal operations.
- **exceptions:** Exception classes within the exceptions package handle and communicate errors or exceptional situations that may occur during the module's execution.

6.2.2 Class Description

Here the main classes that contribute to the efficient functioning of the data lake will be thoroughly examined, looking into its attributes and methods. As classes within the same package often share similar characteristics, this class description aligns with the package structure specified in 6.2.1, guaranteeing a logical and cohesive flow. Each class has a set of methods that based on its role perform specific operations within the data lake. To appropriately handle uncommon events, robust error-handling methods are incorporated in each class, ensuring transparency and user understanding.

DataLakeConfig

The `DataLakeConfig class`, included in the config package, is essential for setting and initializing the Apache Spark environment within the Data Lake module. It makes use of Spring Framework's dependency injection to supply beans for Spark objects. Beans are pre-configured objects in Spring that help with code structure and usage. Spring provides these objects as needed, managing their creation and setup.

This class defines **two beans**: `sparkSession()` and `sparkContext()`. The @Bean annotation instructs Spring to utilize these methods to construct and configure the objects necessary for Apache Spark interaction.

One of the main functions of this class is to create and maintain SparkSessions using the `sparkSession()` method. This session is essential for executing Spark operations and is configured with key parameters. It's worth noting that this session is called "DataLakeApp" and is configured to execute locally, leveraging all available CPU cores with the *"local[*]"* parameter. Furthermore, this session is set up to interact with the Hadoop Distributed File System (HDFS) at `"hdfs://localhost:9000"`. Other configurations may be seen further down in the code snippet.

Listing 6.1 showcases the `DataLakeConfig` class, responsible for configuring a SparkSession essential for data processing. It sets up crucial parameters, such as the application name, master URL, and Hadoop file system settings. This configuration ensures that the data lake application can handle and process data effectively while preserving system stability and fault tolerance.

```
@Configuration
public class DataLakeConfig {

    @Bean
    public SparkSession sparkSession() {
        SparkSession sparkSession = SparkSession.builder()
                .appName("DataLakeApp")
                .master("local[*]")
                .config("spark.hadoop.fs.defaultFS",
                "hdfs://localhost:9000")
                .config("spark.sql.streaming.checkpointLocation",
                "hdfs://localhost:9000/checkpoint_dir")
                .getOrCreate();

        return sparkSession;
    }
```

Listing 6.1: DataLakeConfig Class

LogConfig

The `LogConfig` class is essential for setting the Log4j logging framework. It offers a simple method, `configureLog4j()`, that uses the previously defined `log.xml` configuration file to initialize Log4j, simplifying configuration and assuring consistent and reliable logging across the entire application. Logging is crucial to identify issues, track behavior, and ensure security and reliability. The `LogConfig` class guarantees proper configuration, contributing to app stability and maintainability.

Listing 6.2 introduces the `LogConfig` class implementation.

```
public class LogConfig {
    public static void configureLog4j() {
        String logConfigFile = "log.xml";
        Configurator.initialize(null, logConfigFile);
    }
}
```

Listing 6.2: LogConfig Class

SpringDocConfig

The `SpringDocConfig` class is essential for configuring and generating OpenAPI documentation for the Data Lake module. OpenAPI documentation simplifies integration and development with comprehensive information on endpoints, operations, input parameters, response formats, and authentication mechanisms.

The `SpringDocConfig` class makes use of SpringDoc, a Spring Framework extension, to automate the generation of OpenAPI documentation. It accomplishes this by introducing a

bean, `dataLakeAPI()`, which builds an OpenAPI instance customized for the Data Lake API automatically. This instance contains important API information, such as the title, description, version, and licensing details. This automatic setup simplifies the documentation generating process, providing developers with comprehensive and up-to-date documentation without the need for guidance from developers.

Listing 6.3 showcases the `SpringDocConfig` class implementation.

```
@Configuration
public class SpringDocConfig {

    @Bean
    public OpenAPI dataLakeAPI() {
        return new OpenAPI().info(new Info()
                .title("Data Lake API")
                .description("Operations to perform on the data lake")
                .version("v1")
                .license(new License().name("Apache 2.0")
                .url("http://springdoc.org")));
    }
}
```

Listing 6.3: SpringDocConfig Class

DataAnalysisService

The `DataAnalysisService` class is a service component in the Data Lake module that is responsible for performing dataset analysis as part of a case study. Its major role is to coordinate different data analysis operations through the use of methods from the `DataProcessingManager` class. The `performAnalysis()` method contains the main functionality of this class. When its called this method executes all the methods from the `DataProcessingManager` class. When the `DataAnalysisService` class is instantiated, it initializes the necessary dependencies for data analysis. It connects to the `DataGovernanceManager` and `DataProcessingManager`, which are in charge of data governance and advanced data processing.

Constructor The class's constructor takes a `SparkSession` as a parameter, providing access to Spark's powerful data processing capabilities. This allows for efficient data handling and analysis.

Methods

- **`checkFileIntegrity(String path):`** This method validates the integrity of a given file path by utilizing the `DataGovernanceManager`, ensuring that data is reliable and accurate.

- **`performDataAnalysis():`** This method orchestrates data analysis operations, including generating hourly energy consumption statistics, adding metadata, and performing other analyses as needed.

- **`addMetadata(String dataPath, String metadata):`** Responsible for adding metadata to a specified data path, enhancing data governance and lineage within the Data Lake.

DataGovernanceManager

The `DataGovernanceManager` class, which is included in the datagovernance package, is essential for governing and managing data in the Data Lake module. This class uses various methods to assure data integrity, metadata management, and access control, which are critical for preserving the data lake's quality and security.

One of the most important characteristics of this class is its ability to efficiently manage data information. It keeps a metadata map that links data paths to their corresponding metadata. It adds or modifies metadata associated with a specific data path. This metadata contains critical information about the data, helping its comprehension and management.

Constructor The class's constructor takes a `SparkContext`, which is essential for establishing connections and managing data within Hadoop Distributed File System (HDFS).

Methods

- **`updateDataMetadata(String dataPath, String metadata):`** This method updates metadata to improve data governance and provide information about the data's origin and context.

- **`classifyData(String dataPath, String classification):`** Offers data classification and tagging capabilities, enabling data organization based on classification criteria.

- **`logDataAccess(String userId, String dataPath, String action):`** Facilitates data access auditing by logging data access events, collecting information such as the user, the accessed data path, and the action performed. This leads to enhancing security and compliance measures.

- **`isPathValid(String filePath):`** Validates the existence of a file path within HDFS, ensuring data integrity and accessibility.

Listing 6.4 showcases the `updateMetadata()` method. This method is in charge of updating metadata for specific data files in the Data Lake. It collects metadata and verifies its accuracy, which is critical for good data governance and traceability.

```
public void updateDataMetadata(String dataPath, String metadata) {

    // Update metadata associated with the data
    dataMetadataMap.put(dataPath, metadata);
    metadataLogger.info("Metadata for file "+dataPath+" : "+metadata);
}
```

Listing 6.4: Method for Updating Metadata

IngestionService

The **IngestionService** class is a vital component of the Data Lake project, responsible for the seamless ingestion of data from external sources into the data lake. It is essential to ensure the Data Lake is kept up to date with relevant external data, allowing for real-time data analysis and decision-making. In this case, the class is designed to ingest data from a PostgreSQL database since the data used by the company in their projects is stored in this type of relational database.

Constructor: The class constructor takes two parameters: a `SparkSession sparkSession` instance for efficient data processing, and a `DataGovernanceManager dataGovernanceManager` instance responsible for overseeing data governance and lineage.

Methods:

- **`ingestData():`** This method is responsible for data ingestion in the data lake. It ingests data from external databases using the Spark JDBC library. It takes two parameters: the database name and an array of table names to ingest. Furthermore, it stores the fetched data persistently in HDFS as parquet files. This method combines SQL query execution with Apache Spark's JDBC capabilities, as well as data governance duties such as data access logging and lineage maintenance to ensure data quality and regulatory compliance.
- **`scheduledIngestData():`** This method is a scheduled task that periodically triggers data ingestion from an external database into the data lake, ensuring the data lake is kept up-to-date. The scheduled interval is set to 60 seconds but is adjustable.

The code snippet in **Listing6.5**, represents the `ingestData(String dbName, String []tableNames,String jdbcUrl).`

```
public ResponseEntity<Void> ingestData(String dbName,
String[] tableNames, String jdbcUrl) {

    try {

        // Parameters verification ...

        for (String table : tableNames) {
```

```
            String query = "SELECT * FROM " + table;

            // Data retrieval from the database using Spark JDBC
            Dataset<Row> data = sparkSession.read()
                    .format("jdbc")
                    .option("url", jdbcUrl)
                    .option("dbtable", "(" + query + ") AS tempTable")
                    .option("user", username)
                    .option("password", password)
                    .load();

            // Save data in HDFS as Parquet files
            data.write().mode("overwrite").parquet(hdfsPath);

            // Data governance and lineage ...

        }

        return ResponseEntity.accepted().build();
    } catch (BadHttpRequestException e) {
       \\ Exception Handling
    }
}
```

Listing 6.5: Data Ingestion Method

StorageService

The `StorageService` class is a crucial component of the data lake architecture, responsible for managing the storage and retrieval of data within the Hadoop Distributed File System (HDFS). It serves as a bridge between external data sources and the data lake, facilitating data ingestion and deletion operations.

Constructor: The class constructor accepts two parameters: a `SparkSession sparkSession` instance for efficient data processing, and a `DataGovernanceManager dataGovernanceManager` instance responsible for overseeing data governance and lineage.

Methods:

- **`deleteData()`:** This method is responsible for removing data from the data lake. It takes a `filePath` parameter, which specifies the location of the data file to be deleted. To connect with HDFS and remove the provided file, the method uses Hadoop's FileSystem API. It also incorporates data governance and lineage tracking to ensure that data access events are recorded for auditing purposes.
- **`uploadData()`:** The `uploadData()` method allows data files to be uploaded to the data lake. It takes a `MultipartFile` object containing the data file and a `filePath`

parameter indicating the desired location within the data lake. The method checks and cleans the file path before saving the incoming file to a local temporary file and uploading it to HDFS. It incorporates data governance characteristics, similar to the delete operation, by logging data access events, updating metadata, and classifying the data.

The code snippet in **Listing 6.6** showcases the data upload method within the `StorageService` class.

```
public ResponseEntity<Void> uploadData(@RequestParam MultipartFile file, @RequestParam String filePath) {

    // Parameters verifcation

    try {
        ...
        // Save the multipart file to a temporary local file
        File tempFile = File.createTempFile("temp-",
        file.getOriginalFilename());

        file.transferTo(tempFile);

        // Get the Hadoop file system
        Configuration hadoopConfig = sparkSession.sparkContext()
                                        .hadoopConfiguration();

        FileSystem hdfs = FileSystem.get(hadoopConfig);

        // Upload the file to the data lake
        Path hdfsPath = new Path(completeFilePath);
        hdfs.copyFromLocalFile(false, true, new Path(tempFile
                                .getAbsolutePath()), hdfsPath);

        // Data governance and lineage ...
        // Delete the temporary local file
        tempFile.delete();

        return ResponseEntity.accepted().build();
    } catch (IOException e) {
        // Handle file upload error
    }
}
```

Listing 6.6: Data Upload Method

ProcessingService

The `ProcessingService` class is an essential part of the data lake architecture, and it is used to conduct data queries and facilitate data retrieval and processing. It makes use of the capabilities of Apache Spark to conduct complex operations on the data stored in the data lake.

Constructor: The class constructor has two parameters: a `SparkSession` instance for interacting with Spark data processing capabilities and a `DataGovernanceManager` instance responsible for handling data governance and lineage.

Methods: This class's methods are set to interact with data from a company's project to demonstrate its functionality but they can be adapted to other projects depending on specific requirements. However, the main function of this class's methods remains the same, which is to execute queries on specific data. The class is composed of the following methods:

- **`executeQuery()`:** This method handles the execution of data queries. It takes several parameters, including `userId`, `databaseName`, `tableName`, and `query`. The method processes the query, applies data access restrictions based on user permissions, and logs data access events. In this case, the method is set to query a specific table from a company's project data , the "site_data" table. It applies additional filters based on the user's permission to access the data. Then, if the user is allowed to access the data, the query result is converted to JSON format and returned as a `ResponseEntity` object.
- **`getGroupFromUser()`:** This private method retrieves the user's group from the "users_group_role_group" table, a company's project data table. It takes parameters such as the `userId`, `databaseName`, and `tableName` and executes the corresponding SQL query using SparkSQL. The method logs data access events and returns the user's group as a string.
- **`getAllowedSites()`:** This private method fetches the allowed site IDs for a user's group from the "rel_groups__site_data" table. It requires `userId`, `databaseName`, and `group` as input. The method logs data access events and returns the allowed site IDs as a list of strings.

The code snippet in **Listing 6.7** shows the `executeQuery()` method.

```
public ResponseEntity<String> executeQuery(
        @PathVariable String userId,
        @RequestParam String databaseName,
        @RequestParam String tableName,
        @RequestParam String query) {

    try {

        // Get the user's correspondent group from the table
        String group = getGroupFromUser(userId, databaseName,
        "users_group_role_group");

        // Get the user's allowed sites
        List<String> allowedSiteIds= getAllowedSites(userId,
                                        databaseName,group);
```

```
        // Parameter validation and query preparation ...

         Dataset<Row> siteDataTable = sparkSession.read()
                                          .parquet(fullPath);

        // Data governance and lineage
         siteDataTable.createOrReplaceTempView("site_data");
         result = sparkSession.sql(filteredQuery);

        // Write the dataset to a PostgreSQL table
         result.write()
                .format("jdbc")
                .option("url",
                    "jdbc:postgresql://localhost:5432/postgres")
                .option("dbtable", "query_result")
                .option("user", "postgres")
                .option("password", "postgres")
                .mode("overwrite")
                .save();

        // Convert the result to JSON format
        String jsonResult = result.toJSON().collectAsList()
        .toString();

        return ResponseEntity.ok(jsonResult+","+filteredQuery);

    } catch (BadHttpRequestException e) {
        // Exception handling for bad requests
    } catch (Exception e) {
        // Exception handling for other errors
    }
}
```

Listing 6.7: Query Execution Method

AccessService

The `AccessService` class embodies the data accessibility layer within the Data Lake ecosystem. It acts as a gateway, enabling controlled and secure retrieval of data while offering the capability to generate tailored reports based on user-defined queries.

Constructor: The constructor of the `AccessService` class initializes the service with three essential components. It accepts the following parameters: an instance of `SparkSession` for orchestrating interactions with Spark and Hadoop Distributed File System (HDFS), an instance of `DataGovernanceManager` for integrating data governance functionalities, and an instance of `ReportService` for report generation based on user queries.

Methods:

- **`getData()`:** This method serves as the main interface through which users can access data stored in the Data Lake. It requires the database and table names to be specified, allowing for the seamless retrieval of the desired data. The class guarantees that data access events are meticulously logged, capturing critical parameters such as user identification, targeted table, and operation type (usually ”read”).

- **`generateReport()`:** This method enables users to generate detailed reports based on specific queries. Users can submit unique SQL queries that are suited to their specific data requirements. The `AccessService` class passes the query as a parameter, and in collaboration with the `ReportService` class, produces informative dashboards with custom charts. The method returns a URL that leads to a dashboard where users can access the results of the query. These reports contain valuable insights drawn from the query results, improving the ability to make decisions based on data.

ReportService

The `ReportService` class is an essential component of the Data Lake architecture, designed to facilitate dynamic data reporting and visualization through seamless integration with Apache Superset. It allows users to create interactive and shareable dashboards.

Constructor: The constructor for the `ReportService` class initializes its internal components, including an HTTP client for interacting with Superset's API and a Gson object for JSON handling. It does not require any parameters.

Methods:

- **`configureSupersetForPostgreSQL()`:** This method is used to configure Superset to connect to PostgreSQL databases. It orchestrates the configuration process, including obtaining an authorization token for Superset, and ultimately returns the ID of the created database in Superset.

- **`generateReport()`:** This method facilitates the dynamic creation of data reports and interactive dashboards within the Data Lake environment using a JSON string (`jsonData`) to represent the data. It configures Superset for PostgreSQL with an authentication token acquired. It establishes a dataset along with an associated dashboard. Finally, generates a table chart for data visualization, and offers a permalink for convenient access and effortless sharing of the dynamic content.

- **`createDatabaseConnection()`:** This private method creates a database connection in Apache Superset using the provided information. It sends an HTTP POST request to the Superset API to create the connection and returns the database ID.

- **`createDataset()`:** Another private method that creates a dataset in Apache Superset. It sends an HTTP POST request to the Superset API with dataset details, including the database ID, dataset name, and data in JSON or CSV format.
- **`createDashboard()`:** This private method creates a dashboard in Apache Superset. It sends an HTTP POST request to the Superset API with the dashboard's title and other relevant information.
- **`createChart()`:** Another private method responsible for creating a chart (visualization) in Apache Superset. It sends an HTTP POST request to the Superset API with details like dataset ID, dashboard ID, and chart type.
- **`obtainAuthorizationToken()`:** A private method for obtaining an authorization token from Apache Superset. It sends an HTTP POST request with Superset credentials to obtain an access token.
- **`extractAccessTokenFromResponse()`:** A helper method to extract the access token from the HTTP response obtained during authorization.
- **`getPermalinkForDashboard()`:** This method retrieves a URL for a specific Superset dashboard. It requires two parameters: a `dashboardId`, the unique identifier of the Superset dashboard, and a `authToken`, an authorization token obtained for Superset authentication. The retrieved permalink URL allows users to access and share the dashboard, enhancing collaboration and data exploration.

The code snippet in **Listing 6.8** offers a concise overview of the `generateReport()` method's functionality. This method connects to Apache Superset and a PostgreSQL database. Furthermore, it handles the construction of critical components that, when combined, constitute a shareable and interactive dashboard.

```
public String generateReport(String jsonData) {

    // Configure Superset for PostgreSQL
    String databaseID=configureSupersetForPostgreSQL(
        "SUPERSET_USERNAME", "SUPERSET_PASSWORD",
        "DB_NAME", "DB_HOST","DB_USER", "DB_PASSWORD");

    // Obtain the authorization token
    this.authToken = obtainAuthorizationToken(
        "SUPERSET_USERNAME", "SUPERSET_PASSWORD");

    // Create dataset and get dataset ID
    long datasetId = createDataset(databaseID,datasetName,
            jsonData, authToken);

    // Create dashboard with the dataset
    long dashboardId=createDashboard(dashboardTitle, datasetId);
```

```
    //Create chart and get chartID
    long chartId= createChart(datasetId,dashboardId,authToken);

    //Get permalink for dashboard
    String dashboardURL= getPermalinkForDashboard(dashboardId,
        authToken);

    return "dashboardURL";

}
```

Listing 6.8: ReportService Class

DataLakeAPI

The `DataLakeAPI` is an interface class that defines a set of standardized methods that serve as endpoints for communication and interaction with the Data Lake system. These methods provide various essential operations related to data management and access within the Data Lake architecture. This interface consists of the following methods already described in this section:

- **`uploadData(MultipartFile file, String filePath)`**
- **`getData(String dbName, String tableName)`**
- **`executeQuery(String userId, String databaseName, String tableName, String query)`**
- **`deleteData(String tableName)`**
- **`ingestData(String dbName,String[] tableNames,String jdbcUrl)`**

DataLakeAPIController

The `DataLakeAPIController` class implements the methods from the `DataLakeAPI` interface. It is responsible for exposing a set of RESTful endpoints dedicated to data management and retrieval. These endpoints enable external systems and users to interact efficiently with the Data Lake's core functionalities.

Constructor: The constructor initializes the `DataLakeAPIController` class with essential services, including `IngestionService`, `StorageService`, `ProcessingService`, and `AccessService`. These services are crucial for the execution of data-related operations within the Data Lake.

The endpoints exposed by this class are explored further in section 6.3.

The code snippet in **Listing 6.9** provides an overview of the `ingestData()` method. This method is thoughtfully annotated with Swagger API documentation annotations, such as `@Opera`

`tion, @ApiResponses` and `@Parameter`, to provide precise and well-structured documentation for the API endpoint.

```
@PostMapping("/ingest")
@Operation(summary = "Ingest data into the Data Lake",
    description = "Ingest data into the Data Lake.")
@ApiResponses(value = {
        @ApiResponse(responseCode = "200", description
            = "Successfully ingested data"),
        @ApiResponse(responseCode = "400", description
            = "Bad request"),
        @ApiResponse(responseCode = "500", description
            = "Internal server error")})
public ResponseEntity<Void> ingestData(@Parameter(description
    = "Database name") @RequestParam String dbName,
    @Parameter(description = "Array of table names")
    @RequestBody String[] tableNames,
    @Parameter(description = "JDBC URL")
    @RequestParam String jdbcUrl){

    return ingestionService.
        ingestData(dbName,tableNames,jdbcUrl);
}
```

Listing 6.9: Ingest Data Method

6.2.3 Class Relationships

There are some relationships and interactions in the data lake ecosystem that stand out and are most important.

1. **`DataLakeAPIController` - Layers Services**

   The `DataLakeAPIController` acts as the central orchestrator, utilizing methods from various service classes like IngestionService, StorageService, ProcessingService, and AccessService to expose operations through endpoints.

2. **Layers Services - `DataGovernanceManager`**

   The `DataGovernanceManager` class serves as the central hub for enforcing data governance policies across all Data Lake service classes. It guarantees that data is ingested, stored, processed, and accessed in accordance with governance rules. This interaction ensures data integrity and regulatory compliance.

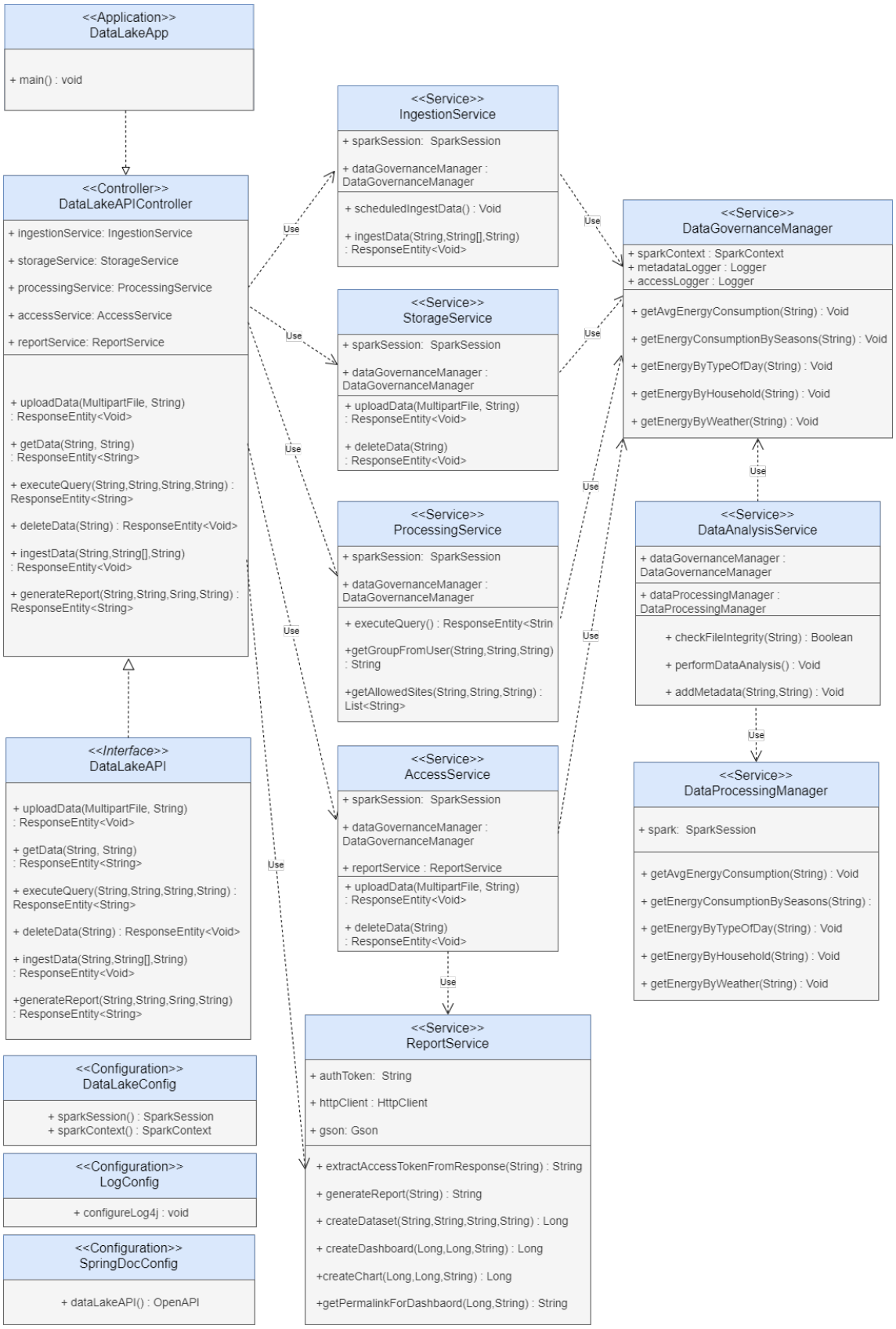

Figure 6.2: Data Lake's Class UML Diagram

A Class UML Diagram illustrates a software system's structure and interactions between classes. The diagram in **figure 6.2** presents an overview of the classes and their relationships in the context of the Data Lake project, providing useful insights into the project's design.

6.3 REST API

The Data Lake module exposes a RESTful API for facilitating system interactions. This API follows industry standards and includes endpoints for data ingestion, retrieval, deletion, querying, and report production. This section provides an overview of the main features of the Data Lake REST API.

A REST API (Representational State Transfer Application Programming Interface) is a set of rules that allows software programs to communicate over the internet. It interacts with resources represented by URLs using typical HTTP methods such as GET, POST, PUT, and DELETE. REST APIs are commonly used to exchange structured data across different software components.

6.3.1 REST API Endpoints

The Data Lake module exposes several REST API endpoints to manage and interact with data stored in the system. Below are concise descriptions of these endpoints:

/api/data/{dbName}/{tableName} - GET

Description: Retrieve data by specifying the database name (`dbName`) and table name (`tableName`).

HTTP Method: GET

Parameters: `dbName` (Path), `tableName` (Path)

Response Codes: 200 (Success), 404 (Not Found), 500 (Server Error)

/api/ingest - POST

Description: Ingest data by providing database name, table names, and a JDBC URL.

HTTP Method: POST

Request Body: JSON Object

Response Codes: 200 (Success), 400 (Bad Request), 500 (Server Error)

/api/delete - DELETE

Description: Delete data by specifying the file path.

HTTP Method: DELETE

Parameters: `filePath` (Request Parameter)

Response Codes: 204 (Success), 400 (Bad Request), 500 (Server Error)

/api/data/upload - POST

Description: Upload data by providing the data file and target file path.

HTTP Method: POST

Parameters: `file` (Multipart Form Data), `filePath` (Request Parameter)

Response Codes: 204 (Success), 400 (Bad Request), 500 (Server Error)

/api/executeQuery/{userId} - GET

Description: Execute a query by specifying user ID, database name, table name, and query.

HTTP Method: GET

Parameters: `userId` (Path), `databaseName` (Request), `tableName` (Request), `query` (Request)

Response Codes: 200 (Success), 400 (Bad Request), 500 (Server Error)

/api/generateReport/{userId} - GET

Description: Generate a report for a specific user by specifying user ID, database name, table name, and query.

HTTP Method: GET

Parameters: `userId` (Path), `databaseName` (Request), `tableName` (Request), `query` (Request)

Response Codes: 200 (Success), 400 (Bad Request), 500 (Server Error)

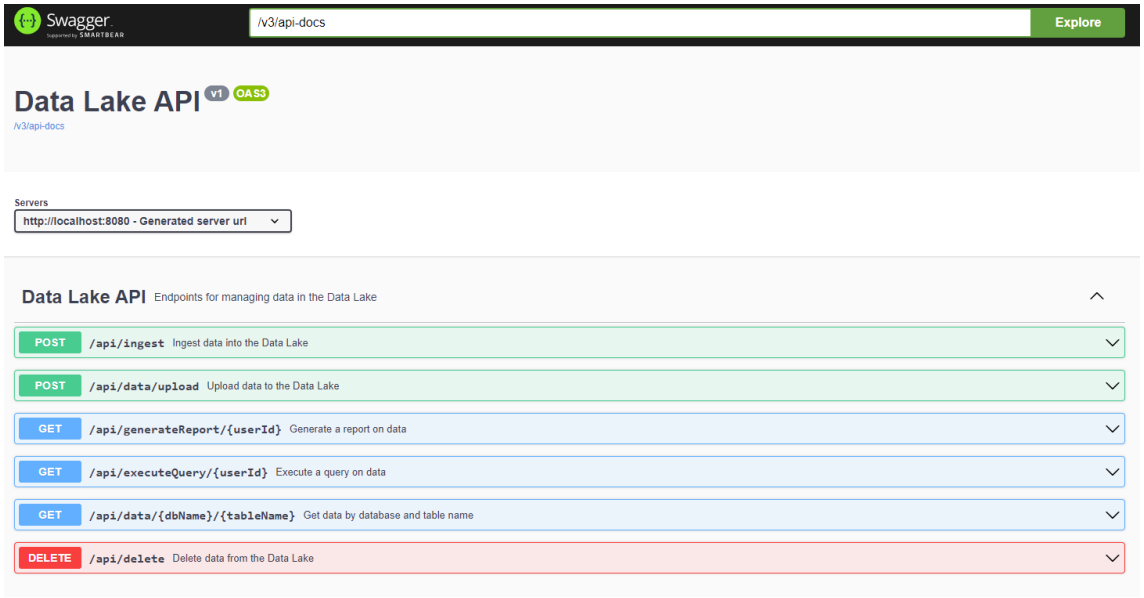

Figure 6.3: Data Lake API in Swagger UI

The Data Lake REST API is documented using SpringDoc and Swagger. This documentation serves as a valuable resource for developers and users seeking to understand and utilize the API effectively. It provides details on each endpoint, including supported HTTP methods, expected

request parameters, response formats, and example usage. The API documentation is available at `http://localhost:8080/swagger-ui/index.html#/` on deployment.

6.4 Deployment

The deployment of the Data Lake module is a critical step in guaranteeing its seamless functioning. The Data Lake module operates within a Java 11 runtime environment using Windows as the operating system. Java 11 provides stability, security, and performance improvements, ensuring that the application runs smoothly. This section provides an overview of the deployment procedure, including the configuration of Hadoop and Spark clusters, the installation of Superset, and the execution of the Data Lake application.

6.4.1 On-Premises Deployment

The Data Lake module is currently deployed on an on-premises Windows environment, specifically in a local development machine. This deployment configuration has been chosen for its flexibility during the development and testing phases.

6.4.2 Hadoop Configuration

Hadoop, an essential component for distributed storage and processing, is configured to operate effectively in the local environment. The specific version used is Hadoop 3.3.1.

The Hadoop cluster is intentionally configured with a single Namenode and a single Datanode, primarily driven by the current low data volume requirements. This choice offers simplicity and efficiency for the current setup. However, it's worth noting that this configuration is easily adaptable, allowing for seamless expansion by adding more Datanodes when data volume escalates in the future.

The `core-site.xml` and `hdfs-site.xml` files include critical configuration properties such as:

core-site.xml

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Here, the property `fs.defaultFS` sets the default Hadoop File System (HDFS) to `hdfs://localhost:9000`, indicating that Hadoop should use the local HDFS running on port 9000.

hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///C:/Hadoop/hadoop-3.3.1/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///C:/Hadoop/hadoop-3.3.1/datanode</value>
  </property>
</configuration>
```

These configurations ensure that Hadoop works properly on the local machine. Additionally, `dfs.replication` is set to 1, indicating that HDFS replicates data blocks once for redundancy in the local environment. The paths specified for `dfs.namenode.name.dir` and `dfs.datanode.data.dir` determine where Hadoop stores its critical metadata and data blocks.

Hadoop Cluster Initialization

After installation and configuration, to initialize the Hadoop cluster, the following commands are executed:

```
$ hdfs namenode -format
$ start-dfs.sh
$ start-yarn.sh
```

The `hdfs namenode -format` command initializes the Namenode, while `start-dfs.sh` and `start-yarn.sh` start the HDFS and YARN services, respectively. These commands set the Hadoop cluster in motion, enabling it to distribute, store, and process data efficiently.

Hadoop services, such as the Namenode, Datanode, and ResourceManager, use specific ports. **The Namenode uses port 9870, Datanode uses port 9864 and ResourceManager uses port 8088.**

Cluster Health and Usage

It is critical to keep track of the cluster's health. Various Hadoop commands and web interfaces can be used to verify the cluster's status and usage. The Hadoop Web UI, for example, gives

information on cluster usage, data nodes, and resource management. It is accessible in `http://localhost:9870/`.

6.4.3 Spark Configuration

Apache Spark, a powerful data processing framework, is also integrated into the deployment environment. The specific version used is Spark 3.1.2.

Setting HADOOP_CONF_DIR in spark-env.sh

In order to ensure seamless integration between Apache Spark and Hadoop, it's necessary to set the `HADOOP_CONF_DIR` environment variable. This variable specifies the location of Hadoop's configuration files, which Spark needs to communicate with HDFS and operate effectively.

To set `HADOOP_CONF_DIR`, the following line was added to the `spark-env.sh` configuration file:

```
export HADOOP_CONF_DIR=/path/to/your/hadoop-3.3.1/etc/hadoop
```

To update the Hadoop configuration, it is necessary to replace `/path/to/your/hadoop-3.3.1/etc/hadoop` with the actual path to the Hadoop configuration directory. By configuring `HADOOP_CONF_DIR` in this manner, Spark will correctly locate Hadoop's configuration, enabling smooth interaction between the two frameworks.

spark-defaults.conf

The `spark-defaults.conf` file contains Spark configuration properties, including:

```
spark.master                    local[*]
spark.eventLog.enabled           true
spark.driver.memory              2g
spark.executor.memory            2g
spark.executor.cores             2
```

In this configuration, Spark runs in local mode using all available CPU cores (`local[*]`). Additionally, memory allocation for both the driver and executor processes is set to 2 GB.

Spark uses various ports for communication and monitoring, including **Spark Master Web UI: 4040, Spark Worker Web UI: 4041, and Spark Application UI: A dynamic port (e.g., 4042, 4043, etc.)**.

Spark Cluster Configuration

The Spark cluster is made up of one master node and one worker node. The master node manages the cluster, while the single worker node performs tasks in parallel. This distributed configuration improves processing power and efficiency, making it well-suited for large-scale data processing workloads.

The decision to use only one worker node is primarily motivated by two factors: the relatively smaller volume of data being processed and the need to keep the deployment simple and cost-effective. Given the current data volume, a single worker node is adequate to handle the task. It provides the ability to increase the number of worker nodes in the future as data volumes increase, ensuring that the system remains adaptive to changing needs.

6.4.4 Superset Installation

To enable data visualization capabilities within the Data Lake project, Apache Superset was installed on a Linux-based environment using Windows Subsystem for Linux (WSL). Due to compatibility issues and a lack of native support for Windows, this approach provided a more efficient installation process. The following steps outline the installation procedure, which can be referenced in detail in the tutorial: Apache Superset Installation in Windows.

1. Installing pip some setup dependencies like virtual environment(venv)

```
$ sudo apt install python3-pip
$ pip3 install virtualenv
$ sudo apt-get install python3-venv
$ python3 -m venv supersetdata
$ .supersetdata/bin/activate
```

2. Install Superset and its dependencies

```
$ pip install --upgrade pip setuptools
$ pip install apache-superset
```

3. Initialize the Superset database and create an admin user

```
$ superset db upgrade
$ export FLASK_APP=superset
$ superset fab create-admin
```

4. Initialize Superset and start the Superset web server

```
$ superset init
$ superset run -p 8088
```

After completing these steps, Apache Superset can be accessed using a web browser at `http://127.0.0.1:8088/`. It provides robust data visualization capabilities, enhancing the Data Lake's functionality.

6.4.5 Data Lake Application

The heart of the system, the Data Lake Spring application, is executed within the IntelliJ IDEA development environment. This application interacts seamlessly with the configured Hadoop and Spark clusters, allowing for efficient data ingestion, processing, storage, and access. All logs are configured to be displayed in the console, providing visibility into the activities of both Spark and HDFS.

Data Lake Application Configuration

The Data Lake Spring application is highly configurable to meet specific data processing and management requirements. Below are some essential configuration settings found in the 'application.properties' file:

- **Multipart File Size Limits:** The application is configured to handle file uploads with specific size constraints.
  - `spring.servlet.multipart.max-file-size=10MB` limits individual file uploads to a maximum of 10 megabytes.
  - `spring.servlet.multipart.max-request-size=20MB` sets the maximum request size, ensuring that requests, including all uploaded files, do not exceed 20 megabytes.
- **Springdoc Integration:** Springdoc is integrated for efficient API documentation.
  - `springdoc.show-actuator=true` enables the display of actuator endpoints in the API documentation.
  - `springdoc.packagesToScan=tese.project.datalake.controllers` specifies the package(s) to scan for REST API documentation. In this case, it focuses on the 'tese.project.datalake.services' package.
  - `springdoc.pathsToMatch=/api/**` defines the paths to match for API documentation generation. Specifically, it targets endpoints under the '/api/' path.

These configurations ensure that the Data Lake Spring application operates within defined file size limits and generates comprehensive API documentation through Springdoc. Furthermore, these configurations can be changed according to specific requirements.

6.4.6 Integration with other projects

The Data Lake module is intended to function as an independent data management system that can be easily integrated into other projects. Its fundamental functionality includes data ingestion, data storage, processing, and access through a set of clearly defined API endpoints. These endpoints are designed to meet the different data requirements of various projects inside the company.

Nonetheless, its methods are also susceptible to change and the inclusion of new functionalities. Consequently, projects looking to leverage the Data Lake can seamlessly integrate with it by making API calls to the defined endpoints. This approach allows projects to initiate the necessary operations and efficiently handle the responses, simplifying integration and ensuring that the Data Lake remains a valuable and flexible resource for all.

6.5 Testing

Testing is crucial in guaranteeing the data lake module's reliability, functionality, and correctness. It is critical to ensure that the system operates as intended, handles varied scenarios gracefully, and responds correctly to various inputs. A comprehensive testing approach was developed to meet these objectives, leveraging the power of industry-standard tools and techniques like JUnit, Mockito and Postman.

6.5.1 Unit Testing with JUnit

JUnit, a widely used Java unit testing library, served as the testing process's foundation. Unit testing involves examining individual units or components of the data lake in isolation to validate their behavior. This granular technique enables a thorough examination of the functionality of each class and method, ensuring that they function as expected.

Unit tests were written for some classes to cover a wide range of scenarios, including normal use cases, boundary conditions, and error handling. For example, the ingestion service was tested to ensure that it properly handles different data formats, while the access service was validated to provide the expected responses for different request types.

6.5.2 Mocking Dependencies with Mockito

Mockito, a powerful mocking framework, was essential in isolating data lake classes from their external dependencies. This isolation enabled the creation of controlled and predictable testing scenarios. Mockito allowed tests to focus solely on the functionality of Data Lake components by generating mock objects that imitate the class's behavior.

Mockito was used to test the processing service, for example. Mock objects were used to simulate classes in the Data Lake, resulting in a controlled environment in which the processing logic could be rigorously validated.

6.5.3 Postman API Testing

API endpoints are the gateways to the data lake, making their reliability and correctness crucial. **Postman**, a versatile API testing tool, played a crucial role in evaluating the functionality of these endpoints. It facilitated HTTP request execution, enabling seamless interaction with the API. It also made it easier to examine API answers to ensure their accuracy and reliability.

One important part of this testing was comparing API responses to actual database data to ensure data was correct. Furthermore, error-handling scenarios were tested to ensure that actions like file deletion produced the required error messages or status codes when trying to access data previously deleted. Postman's wide testing skills were essential in improving the overall quality and robustness of the Data Lake system.

Figure 6.4 illustrates a sample test scenario involving an endpoint and its corresponding JSON body response, validating it against the expected content.

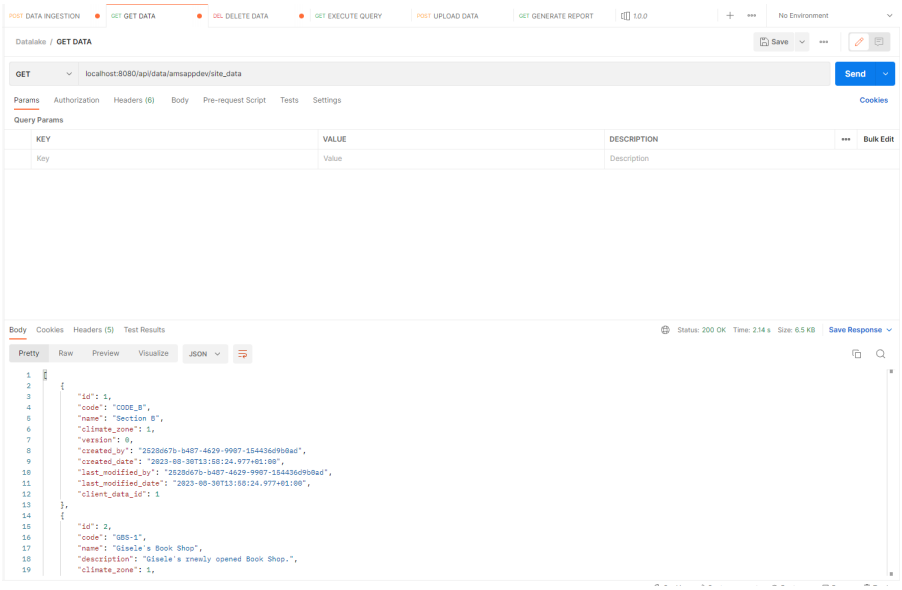

Figure 6.4: Testing endpoints with Postman

6.6 Challenges Faced

The development of the Data Lake system was a journey marked by various challenges and obstacles that required innovative solutions and persistent effort. Below are some of the key challenges encountered during the project.

6.6.1 Learning Curve of Spark and Hadoop

One of the hurdles we encountered was the learning curve associated with Apache Spark and Hadoop. These technologies form the backbone of our Data Lake. Although they offer power they come with a complex ecosystem and a vast array of libraries. It required an amount of time and dedication to become skilled, in these technologies and effectively set up the Spark and Hadoop cluster for our Data Lake. Understanding the intricacies of Spark, Hadoop and their Java libraries was crucial to take full advantage of their potential, for data processing and storage.

6.6.2 Superset Configuration and Integration

Integrating Apache Superset into the Data Lake system presented its own set of challenges. Configuring Superset, especially in a Linux-based environment, proved to be a non-trivial task. The lack of comprehensive documentation and support resources made this process even more demanding. Several bug fixes and environment changes were required to guarantee that Superset worked properly within the Data Lake ecosystem.

Another significant challenge was configuring the `ReportService` class to interact seamlessly with Superset. The Superset API, while a powerful tool for creating data visualizations and dashboards, lacked detailed documentation. This resulted in an iterative process of trial and error when attempting to make HTTP requests to the API for dashboard generation. Understanding the structure of the request body and its parameters required meticulous exploration and testing.

6.6.3 Availability of Representative Big Data

Working with big data presented its own unique challenge. Acquiring and managing a dataset that accurately represented the scale and complexity of big data proved to be difficult. Local machine limitations made it challenging to work with large datasets, and finding a truly representative big data source was a task in itself. As a practical alternative, the project team chose to use project data for testing and integration.

In conclusion, overcoming these challenges required a combination of technical expertise, perseverance, and creative problem-solving. Each obstacle was an opportunity to learn and innovate, ultimately contributing to the successful development and deployment of the Data Lake system.

6.7 Limitations and Future Enhancements

The implementation of the Data Lake system, while robust and functional, has certain limitations that warrant attention. Furthermore, there are opportunities for future enhancements to elevate the system's capabilities and address these limitations.

6.7.1 Limitations

1. **Data Governance Approach**: The current implementation relies on in-memory data structures like maps to manage data governance metadata. Even though this approach works well for smaller scenarios, it may not be scalable when data governance requirements are extensive. Future enhancements should consider transitioning to a database-based approach for storing metadata, offering more scalability and robustness.

2. **Security and Authentication**: The security measures, in the Data Lake are currently limited. Mainly depends on the security measures implemented on the client side. To enhance security it is crucial to establish authentication and authorization mechanisms within the Data Lake API. By adopting token-based authentication an extra layer of security can be added to ensure that authorized users have access to and control over the data, in the system.

3. **Superset Integration Bug**: An existing limitation in the Superset integration arises from a known bug that currently lacks available fixes. This issue relates to retrieving the URL of the generated dashboard with data visualizations. No current solution were found until this moment to resolve this bug.

6.7.2 Future Enhancements

1. **Expanded API Functionality**: More API endpoints may be added in the future to support a variety of data operations. By continually enriching the API's capabilities, the Data Lake system can become even more versatile and adaptable to diverse data requirements.

2. **Enhanced Security Measures**: The implementation of security features should be a top priority. This involves not only authentication and authorization but also encryption of data at rest and during transmission.

3. **Integration with Identity Providers**: To streamline user management and authentication, integrating the Data Lake with identity providers (e.g., LDAP, OAuth) can be beneficial. This would enable seamless single sign-on (SSO) and simplify user access control.

4. **Dockerization**: The Data Lake's future roadmap includes the adoption of Docker containers, unlocking several benefits such as enhanced scalability, portability, and simplified deployment. Containerization ensures the Data Lake can seamlessly adapt to diverse environments while maintaining stability and version control.

5. **Integration Testing**: Comprehensive integration testing will be employed to fortify the Data Lake's reliability. It involves examining the system's behavior when its various components work together. Key areas of scrutiny include API endpoint interactions, data ingestion and processing pipelines, data access and retrieval, data governance, error handling, scalability, and performance. Integration testing ensures that the Data Lake operates cohesively as a dependable data management solution.

The Data Lake module can evolve into a more comprehensive and secure data management solution by addressing these limitations and embracing future enhancements. These enhancements will not only strengthen its functioning but will also keep it relevant in a constantly shifting data landscape.

Chapter 7

Case Study: Smart Meters in London

This chapter presents a practical case study designed to illustrate several key data lake capabilities, including data processing, reporting, and analysis. The primary aim of this study is to leverage an open-source dataset related to smart meters in London to highlight the inherent advantages and capabilities of data lakes. The selection of this dataset was a collaborative decision with the company, motivated not only by the dataset's intriguing content and analytical potential but also due to its substantial size (10GB), approaching the scale of big data.

7.1 Smart Meters Dataset Overview

The dataset used in this case study is sourced from Kaggle and is titled "Smart Meters in London." It provides a collection of data related to smart meter usage within the Greater London area. The dataset offers a valuable glimpse into energy consumption patterns as it contains the energy consumption readings for a sample of 5,567 London households that took part in the UK Power Networks-led Low Carbon London project between November 2011 and February 2014.

In addition to energy consumption data, the dataset includes supplementary information derived from households' classification according to the ACORN system. ACORN is a classification system commonly employed in the UK, designed to categorize households based on social demographic indicators. This classification adds a socio-economic context to the dataset, enabling deeper insights into energy usage patterns across various demographics.

Furthermore, the dataset incorporates weather data that corresponds to the period of energy consumption measurements improving the analysis potential.

The dataset is publicly available and can be accessed through the following link: `https://www.kaggle.com/datasets/jeanmidev/smart-meters-in-london`.

7.2 Data Preparation

In this section, the critical steps taken during the data preparation phase, which encompassed data acquisition, preprocessing, and integration into the data lake, are discussed. The process aimed to ensure the quality and suitability of the dataset for subsequent analysis and reporting.

DataProcessingManager

The `DataProcessingManager` class is crucial for this case study as it's responsible for all the processing operations. It makes good use of Apache Spark's capability to perform multiple data transformations and calculations. This class allows the Data Lake to extract valuable insights from the dataset. Before executing the methods of this class, some data cleaning was performed, removing null values and discarding irrelevant values for the analysis.

Constructor: This class obtains a `SparkSession` object upon initialization using dependency injection. It is critical for running Spark operations smoothly and provides the computational power required for complicated data processing tasks.

Methods: The `DataProcessingManager` class provides a number of methods, each of which is designed for a distinct data processing activity. Each of these methods corresponds to a metric that is set for analysis. Among these methods are:

- **`getHourlyEnergyConsumption()`**: Aggregates energy consumption data into hourly intervals, computes averages, and stores the results.
- **`getAvgEnergyConsumption()`**: Calculates the daily average energy consumption and identifies patterns in energy use.
- **`getEnergyConsumptionBySeasons()`**: Analyzes energy consumption by seasons and stores the findings.
- **`getEnergyByTypeOfDay()`**: Groups and summarizes energy data by day type (weekday, weekend and public holiday).
- **`getEnergyByHousehold()`**: Combines energy consumption data with household information, allowing analysis by household characteristics.
- **`getEnergyByWeather()`**: Integrates weather data with energy consumption, providing insights into the correlation between weather conditions and energy usage.

Once the data processing operations are completed, the class generates the results and saves them in parquet files in HDFS. This storage method enhances data retrieval and analysis efficiency. Parquet files were selected as the storage format due to their columnar storage structure, which brings several benefits. Columnar storage in Parquet allows for more efficient data compression and encoding, hence requiring less storage space. Moreover, it ensures faster query performance since only the relevant columns are read during query execution, reducing I/O operations. This format is particularly useful for analytical tasks, thus making it an optimal choice for storing processed data in HDFS.

The provided code snippet in **Listing 7.1** illustrates a method within this class, specifically the `getEnergyByDayType()` method. It provides insight into the data processing procedure, demonstrating how Spark's capabilities are harnessed to process data and store the results in HDFS.

```
public void getEnergyByDayType(String path) {

    // Load data from CSV file
    Dataset<Row> data = spark.read().option("header", true)
        .csv(path);

    // Convert date strings to date type
    Dataset<Row> formattedData = data.withColumn("Date",
    to_date(col("day"), "yyyy-MM-dd"));

    // Select columns
    Dataset<Row> result = formattedData.select("Date",
    "energy_mean", "energy_sum")
        .groupBy("Date")
        .agg(avg("energy_mean").as("average_energy_mean"),
            sum("energy_sum").as("total_energy_sum"));

    // Write the results to a Parquet file in HDFS
    result.write()
        .mode("overwrite")
        .parquet("hdfs://localhost:9000/output/
        energy_consumption_by_days.parquet");
}
```

Listing 7.1: Method for Calculating Energy Consumption by Day Type

7.3 Analysis

This section summarizes the key findings of this case study's analysis. While this analysis case study does involve a comprehensive examination of data, its primary goal is to compare the data visualization capabilities provided by the data lake to standard relational databases. Furthermore, it emphasizes the seamless integration with data visualization tools like Power BI, highlighting the important role of these capabilities in enhancing data exploration and decision-making processes.

The choice of Power BI as the preferred data visualization tool was a deliberate decision made together with the company, primarily driven by the tool's exceptional capabilities in the realm of data visualization, surpassing the alternatives that were scrutinized.

The data processing workflow began within the data lake, where Apache Spark was employed to process and transform the data. Subsequently, the processed data was stored in HDFS in the form of Parquet files. Finally, the data was loaded into Power BI using its HDFS connector, and visualizations were generated from the data.

For analysis, data from the years 2011 and 2014 was excluded due to insufficient values for all days of the year and lower data coverage compared to other years.

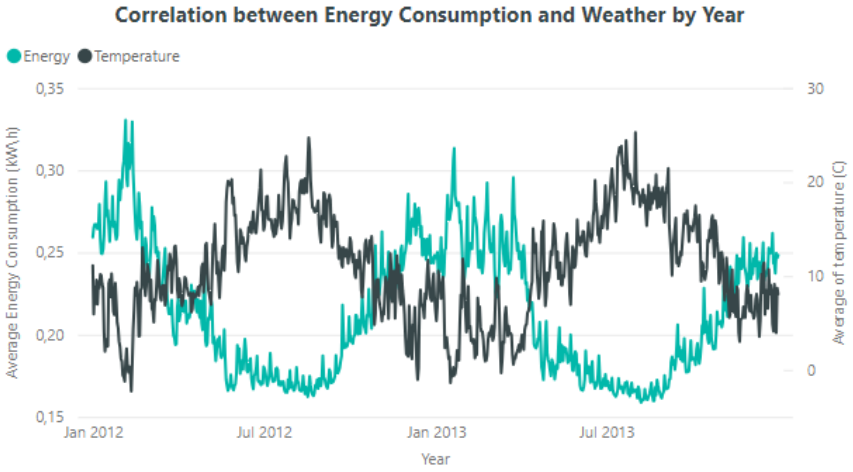

Figure 7.1: Correlation between Energy Consumption and Weather by year

The graphic depicted in 7.1 presents a correlation between daily energy consumption averages and weather by year. It shows a symmetry where energy consumption and temperature display an inverse relationship over time. The symmetry between energy consumption and temperature suggests a strong seasonal effect on energy usage. As temperatures rise, typically during summer months, energy consumption tends to decrease. This decline is likely attributed to the reduced demand for heating during warmer weather. Conversely, during colder months, energy consumption increases possibly due to higher demand for heating.

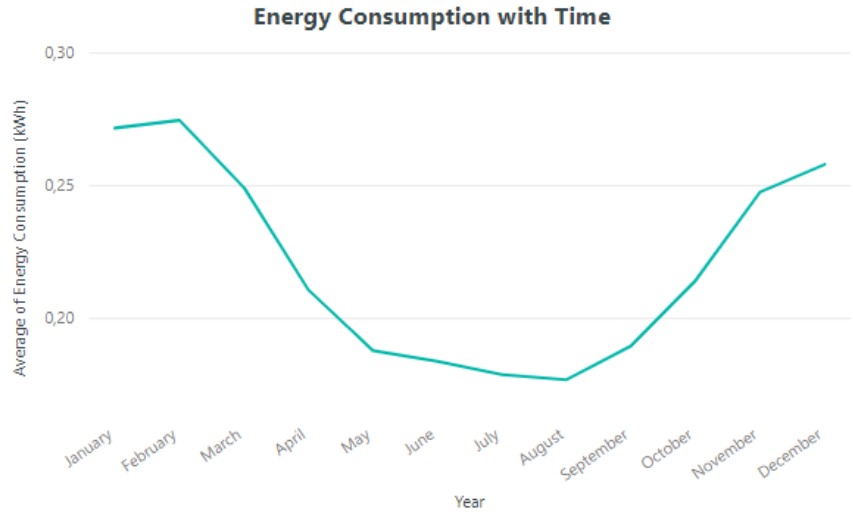

Figure 7.2: Energy Consumption with time

The line graph in 7.2 depicts daily energy consumption averages over time. It shows a characteristic pattern that roughly resembles a ”U” shape, yet with flatter beginnings and finishes. The most noticeable aspect of this pattern is the clear decline in energy usage during the sum-

mer months, which corresponds with the period of higher temperatures and is consistent with the previous analysis of the correlation between energy consumption and temperature.

The ”U” form represents a strong seasonality in energy usage, with significant fluctuations throughout the year. Energy use is lowest during the summer months when temperatures are rather high. This might indicate a reduced demand for heating, which is a significant contributor to energy use, particularly during the colder seasons. In contrast, as it begins to approach the colder months, the line chart shows a steady rise in energy consumption. This growing tendency is strongly tied to the reduction in winter temperatures, which needs greater usage of heating equipment.

This insight provides a comprehensive understanding of the factors influencing energy consumption, emphasizing the strong influence of seasonal temperature changes. By combining the data from the energy consumption line graph and the correlation analysis between energy consumption and temperature, clear inferences can be drawn about the crucial role that weather conditions play in influencing energy consumption patterns over time. This knowledge can be useful in establishing plans to optimize energy usage and improve energy efficiency, especially during the various seasons of the year.

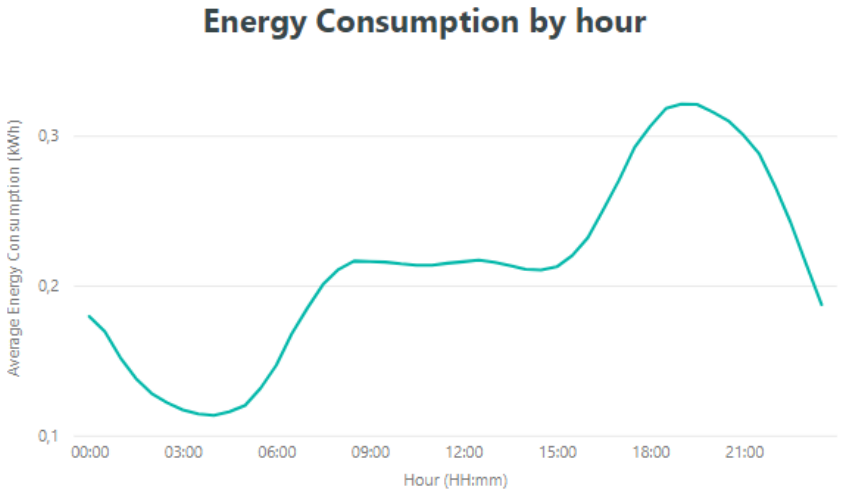

Figure 7.3: Energy Consumption by Hour

The hourly energy consumption line chart reveals a particular pattern that resembles a stretched ”U” shape with distinct features at different times of the day. During the early morning hours, from 00:00 to 6:00, the chart demonstrates the form of a ”U,” with energy consumption at its lowest point at around 4:00. This low energy demand can be attributed to the fact that most households are in a state of minimal activity during this time, with many people asleep. Consequently, there is reduced usage of energy-intensive appliances and lighting, leading to lower overall energy consumption. Between 8:00 and 15:00, the energy consumption profile is more consistent, with some slight fluctuations. This consistency can be attributed to the working hours of a large percentage of the population. During this time, energy consumption remains relatively steady as households and businesses operate at regular capacity, using a consistent amount of energy to power various

activities. The late evening hours show a different pattern. Energy consumption starts rising in the shape of an inverted ”U,” with a peak between 18:30 and 20:00. This is the time when people return home from work and engage in a variety of evening activities. As a result, energy usage surges due to the simultaneous operation of appliances, lighting, and heating or cooling systems. Following the evening peak, energy consumption gradually decreases as households wind down and people retire for the night. The pattern forms a gentle descending slope as the night progresses.

The hourly consumption chart, in conjunction with our analysis of energy consumption over time, provides valuable insights into how daily habits and activities influence energy usage. These findings highlight the significance of comprehending the daily patterns of energy consumption, as this knowledge can be used to adopt energy-saving initiatives and optimize resource allocation throughout the day.

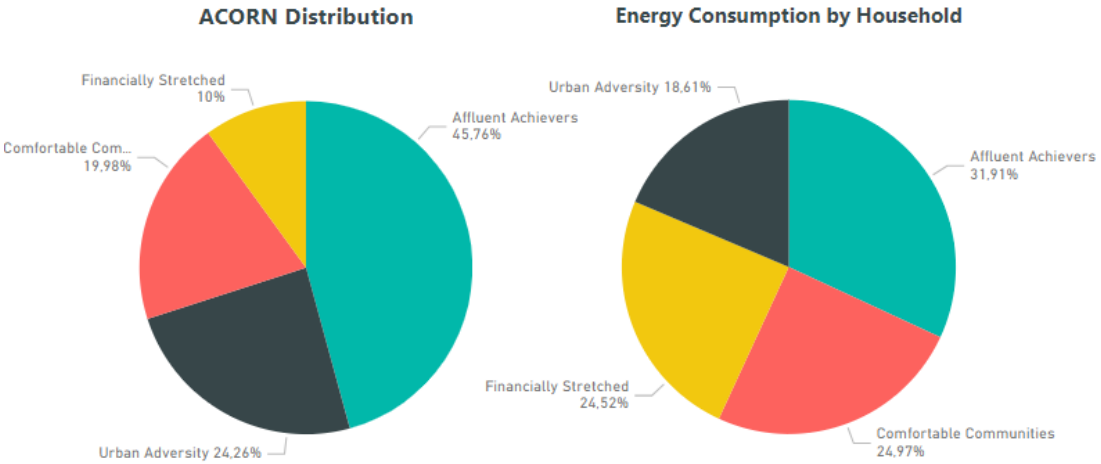

Figure 7.4: Correlation between Energy Consumption and Household

The pie charts in 7.4 illustrate energy consumption by households, categorized according to the UK ACORN classification system. It reveals intriguing insights into the dynamic interplay between socioeconomic factors and energy consumption patterns. Four primary categories emerge: ”Affluent Achievers”, ”Comfortable Communities”, ”Financially Stretched”, and ”Urban Adversity”.

The chart on the left illustrates the distribution of energy consumption records within this dataset among various ACORN categories. Notably, the ”Affluent Achievers” category holds the largest share, representing almost half of the records (45.76%). The second-largest category is ”Urban Adversity,” accounting for 24.26% of the records, followed by ”Comfortable Communities” at 19.98%. Finally, the ”Financially Stretched” category contains the smallest proportion of energy records, standing at 10%.

The chart on the right illustrates the average energy consumed by each ACORN category.

The ”Affluent Achievers” represent the category that has the highest energy consumption (31.91%) of the energy consumed. This category includes wealthy households with significant financial resources. Their relatively high energy consumption can be attributed to larger living

spaces, increased appliance usage, and a greater demand for climate control systems. It reflects the lifestyle associated with wealth. On the other hand, the "Comfortable Communities" (24.97%) category represents neighborhoods characterized by moderate wealth. These households have high energy consumption, often due to the presence of families, with multiple members contributing to energy demands.

In contrast, the "Financially Stretched" (24.52%) and "Urban Adversity" (18.61%) categories had lower energy use percentages. These categories typically consist of households with limited financial means. The energy consumption within these categories can be traced to smaller living spaces, reduced appliance usage, and an overall need for economizing resources.

Furthermore, the data highlights the importance of focused energy policies, infrastructural development, and social programs. Adapting these approaches to meet the specific needs of each ACORN category can result in more sustainable and equitable energy usage patterns. It is a critical step toward closing socioeconomic gaps and providing affordable, efficient, and ecologically friendly energy services to all parts of society.

The remaining analysis is accessible through a published dashboard on the Power BI service. [1]

[1] https://app.powerbi.com/reportEmbed?reportId=ea0617a0-9940-403b-8d3d-e58051a127a6&autoAuth=true&ctid=6b81d0e1-94b9-4d82-b284-99bb32dec71b

Chapter 8

Conclusion

In summary, this thesis embarked on a significant journey, driven by the main goal of seamlessly integrating a data lake into the company's projects that follow a microservices architecture. In order to achieve this goal extensive work had to be performed evolving a lot of effort.

First and foremost, the thesis delved into comprehending the realities and challenges posed by the era of big data. It focused on some key characteristics of big data such as volume, velocity, variety, and veracity. This understanding was crucial in developing a solution for the data lake implementation. After exploring some key concepts, a comparative analysis shed light on the distinct advantages offered by data lakes, including their innate ability to effortlessly and efficiently manage diverse data types and seamlessly scale as data requirements grow.

Furthermore, the analysis of current data lake trends and architectures was crucial in providing the company with the knowledge required to make informed decisions. This analysis involved a deep dive into the evolving landscape of data lake technologies, design elements, and management practices. It enabled the identification of trends that align with the company's goals, positioning them to proactively manage their data resources.

Alongside the previously established requirements, this analysis led to the choice of a layered architecture for the data lake, with clearly defined layers responsible for data ingestion, storage, processing, and access. The adoption of a layered architecture brought several advantages. It enabled the modular development and maintenance of individual components, resulting in a more simplified and organized project structure. Each layer was particularly designed to fulfill its function effectively, encouraging code reusability and scalability. Data flowed seamlessly from one layer to another, ensuring a consistent and controlled data journey.

Choosing the appropriate technologies for each layer was a significant process. In the data ingestion layer, Apache Spark performed efficiently, handling large amounts of data formats. The storage layer relied on Hadoop Distributed File System (HDFS) and Apache Parquet for scalable and structured data storage. For data processing, Apache Spark emerged as a powerful and versatile tool, while the data access layer leveraged Spring Boot for API development and Apache Superset for interactive data visualization.

The adoption of an API for integration with other projects, especially within a microservices architecture, has brought significant benefits to the Data Lake module. This decision has pro-

moted modularity, encouraged reusability, and simplified data interactions. The API provides a standardized, abstraction layer for data access through HTTP endpoints, optimizing development and simplifying data interactions.

The successful implementation of effective data governance features has been essential in maintaining data quality, effectively handling metadata, and assuring clear data lineage. These features have made a substantial contribution to data integrity and trustworthiness. By efficiently implementing data governance, the system constantly considers high data quality standards and provides clear visibility into data lineage, which is a critical component of dependable data management.

The successful integration with Apache Superset expands the module's capabilities by enabling users to access rich data visualizations and dashboards. This empowers users to gain insights from their data quickly and effectively.

The rigorous testing approach, utilizing JUnit and Mockito, has fortified the Data Lake module's reliability and correctness. It provides confidence in the module's ability to perform consistently in real-world scenarios.

Nevertheless, as with any project, certain limitations were encountered. The reliance on in-memory data structures for data governance, while suitable for smaller scenarios, may prove less scalable in extensive data governance requirements. Moreover, the security measures in the data lake are currently limited, mainly relying on security mechanisms implemented at the client end. Future enhancements will necessitate the adoption of robust authentication and authorization mechanisms within the Data Lake API, enhancing security measures. Finally, the integration with Apache Superset poses a challenge. A known bug with no quick fixes impedes the retrieval of the URL for created dashboards containing data visualizations. This issue may temporarily limit flawless integration with Apache Superset until a solution is provided.

Looking ahead, the project sets the stage for future developments and enhancements. The adoption of Docker containers for deployment offers improved scalability, portability, and deployment ease. Furthermore, integration testing will strengthen the solution's robustness, ensuring its reliability in a variety of scenarios.

In conclusion, this thesis has equipped the company with the knowledge, insights, and strategies to harness the revolutionary potential of data lakes within a microservices architecture. By achieving its objectives, this thesis has laid the foundation for the company to thrive in an era where data stands as a strategic asset of unparalleled significance, enabling data-driven decisions, innovation, and competitive advantage.

Bibliography

[1] Data lake vs data warehouse: 6 key differences — qlik. https://www.qlik.com/us/data-lake/data-lake-vs-data-warehouse. Accessed on 2024-03-02.

[2] Databricks hadoop distributed file system (hdfs) glossary. https://www.databricks.com/glossary/hadoop-distributed-file-system-hdfs. Accessed on 2024-03-02.

[3] Databricks hadoop glossary. https://www.databricks.com/glossary/hadoop. Accessed on 2024-03-02.

[4] Snowflake lambda architecture. https://www.snowflake.com/guides/lambda-architecture#:~:text=Lambda%20architecture%20is%20a%20data,for%20responding%20to%20user%20queries. Accessed on 2024-03-02.

[5] Spring boot. https://spring.io/projects/spring-boot. Accessed on 2024-03-02.

[6] Amazon. Amazon s3. https://aws.amazon.com/s3/, 2002. Accessed on 2024-03-02.

[7] Amazon Web Services. What is a data lake? https://aws.amazon.com/pt/big-data/datalakes-and-analytics/what-is-a-data-lake/. Accessed on 2024-03-02.

[8] Apache Flink. Apache flink: Stateful computations over data streams. https://flink.apache.org/. Accessed on 2024-03-02.

[9] Hossein Ashtari. What are microservices? definition, examples, architecture, and best practices for 2022. https://www.spiceworks.com/tech/devops/articles/what-are-microservices/, Apr 2022. Accessed on 2024-03-02.

[10] Julia Couto, Olimar Borges, Duncan Ruiz, Sabrina Marczak, and Rafael Prikladnicki. A mapping study about data lakes: An improved definition and possible architectures. pages 453–458, 07 2019.

[11] Databricks. What is a data lakehouse? https://www.databricks.com/glossary/data-lakehouse#:~:text=What%20is%20a%20Data%20Lakehouse,(ML)%20on%20all%20data., Mar 2022. Accessed on 2024-03-02.

[12] Databricks. About spark. https://www.databricks.com/spark/about, 2023. Accessed on 2023-09-12.

[13] Databricks. What is apache spark. https://www.databricks.com/glossary/what-is-apache-spark, 2023. Accessed on 2023-09-12.

[14] Editor. Hadoop vs spark: Main big data tools explained. https://www.altexsoft.com/blog/hadoop-vs-spark/, Jun 2021. Accessed on 2024-03-02.

[15] Excelsior. Big data, explained: The 5v s of data. https://medium.com/@get_excelsior/big-data-explained-the-5v-s-of-data-ae80cbe8ded1, 2022. Accessed on 2024-03-02.

[16] Huang Fang. Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem. In *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 820–824, 2015.

[17] Corinna Giebler, Christoph Gröger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang. Leveraging the Data Lake - Current State and Challenges. In *Proceedings of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2019)*, 2019.

[18] Corinna Giebler, Christoph Gröger, Eva Hoos, Holger Schwarz, and Bernhard Mitschang. A Zone Reference Model for Enterprise-Grade Data Lake Management. In *Proceedings of the 24th IEEE Enterprise Computing Conference (EDOC 2020)*, 2020.

[19] Chandler Harris. Microservices vs. monolithic architecture. https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith. Accessed on 2024-03-02.

[20] Tomislav Hlupić, Dražen Oreščanin, Domagoj Ružak, and Mirta Baranović. An overview of current data lake architecture models. In *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1082–1087, 2022.

[21] Bill Inmon. *Data Lake Architecture: Designing the Data Lake and avoiding the garbage dump*. Technics publications, 2016.

[22] Josue J. Apache superset: A modern real-time data exploration and visualization platform. https://t.ly/medium-superset, 2023. Accessed on 2024-03-02.

[23] Tomcy John and Pankaj Misra. *Data lake for enterprises*. Packt Publishing Ltd, 2017.

[24] Doug Laney et al. 3d data management: Controlling data volume, velocity and variety. *META group research note*, 6(70):1, 2001.

[25] Ying Li, AiMin Zhang, Xinman Zhang, and Zhihui Wu. A data lake architecture for monitoring and diagnosis system of power grid. In *Proceedings of the 2018 Artificial Intelligence and Cloud Computing Conference*, AICCC '18, page 192–198, New York, NY, USA, 2018. Association for Computing Machinery.

[26] Nathan Marz. How to beat the cap theorem. `https://aphyr.com/posts/322-how-to-beat-the-cap-theorem`, 2011. Accessed on 2024-03-02.

[27] Amr A. Munshi and Yasser Abdel-Rady I. Mohamed. Data lake lambda architecture for smart grids big data analytics. *IEEE Access*, 6:40463–40471, 2018.

[28] Sam Newman. *Building microservices*. " O'Reilly Media, Inc.", 2021.

[29] Carlos Oeiras. Uma breve introdução do hadoop hdfs (hadoop distributed file system) (1/2). `https://t.ly/medium-hdfs`, 2020. Accessed on 2023-09-13.

[30] Dražen Oreščanin and Tomislav Hlupić. Data lakehouse - a novel step in analytics architecture. In *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1242–1246, 2021.

[31] Sonam Ramchand and Tariq Mahmood. Big data architectures for data lakes: A systematic literature review. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1141–1146, 2022.

[32] Margy Ross and Ralph Kimball. *The data warehouse toolkit: the definitive guide to dimensional modeling*. John Wiley & Sons, 2013.

[33] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)*, pages 42–47. IEEE, 2013.

[34] upGrad. Top 3 apache spark applications: Use cases & why it matters. `https://t.ly/medium-spark`, August 2020. Accessed on 2024-03-02.

[35] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[36] Elisabeta Zagan and Mirela Danubianu. Cloud data lake: The new trend of data storage. In *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–4, 2021.