

# INDIVIDUAL ASSIGNMENT ONE REPORT

LEVEL 6

COMP60022

## DECISION ANALYTICS HFK2422COM HFK2422COMF

Hand Out Date: 28/07/2024

Hand In Date: 07/10/2024

---

Staffordshire Student Number	Student CB Number	Student Full Name
21035795	CB010406	Kavinda Kethiya Rajapaksha

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Problem Context Status . . . . .	6
1.2	Business Questions . . . . .	8
<b>2</b>	<b>Databases</b>	<b>9</b>
2.1	Concepts . . . . .	9
2.2	Relational and Non-Relational Databases . . . . .	10
2.3	Strength and Weaknesses of Relational and Non-Relational Databases . . . . .	11
2.4	Right Database Option for an application . . . . .	12
<b>3</b>	<b>Data Engineering</b>	<b>13</b>
3.1	Data Engineering Concepts . . . . .	13
3.2	Data Engineering Pipeline . . . . .	16
<b>4</b>	<b>MongoDB</b>	<b>20</b>
4.1	History . . . . .	20
4.2	Concepts . . . . .	20
4.3	Advantages and Disadvantages . . . . .	24
<b>5</b>	<b>MongoDB Application</b>	<b>25</b>
5.1	Database . . . . .	25
5.2	Collections . . . . .	26
5.3	Mongo Shell . . . . .	26
5.4	Mongo Compass . . . . .	32
<b>6</b>	<b>Development of MongoDB Application</b>	<b>36</b>
6.1	Technical Stack and Environment Setup . . . . .	36
6.2	Front-end Development . . . . .	38
6.2.1	Graphical User Interface . . . . .	41
6.3	Back-end Development . . . . .	45
6.4	Testing . . . . .	46
<b>7</b>	<b>Data Analytics</b>	<b>49</b>
7.1	Analytical Questions . . . . .	49
7.2	Hypotheses . . . . .	52



# List of Tables

2.1	Pros and Cons of SQL and NOSQL . . . . .	11
2.2	Advantages and Disadvantages Relational Databases . . . . .	11
2.3	Benifits and Drawbacks of Non-Relational Databases . . . . .	12
4.1	Benefits and Drawbacks of Replication . . . . .	22
4.2	Advantages and Disadvantages of MongoDB . . . . .	24
6.1	CRUD Test Case Results . . . . .	47

# List of Figures

1.1	Nexart Enterprise Logo . . . . .	6
1.2	Nexart Head Office California . . . . .	6
1.3	Nexart Production Plant California . . . . .	7
3.1	Hierarchy of Data Science Necessities . . . . .	13
3.2	Data Engineering Processes . . . . .	14
3.3	ELT Pipeline . . . . .	14
3.4	ETL Pipeline . . . . .	15
3.5	Data Lake with Big Data Infrastructure . . . . .	15
3.6	Apache NiFi Extracting Data From MSSQL Database . . . . .	17
3.7	Apache NiFi Storing Extracted Data in MongoDB Database . . . . .	17
3.8	Apache Spark Transforming Data in MongoDB Database . . . . .	18
3.9	Apache AirFlow Automation . . . . .	18
5.1	Nexart Database Creation . . . . .	25
5.2	Nexart Database Creation Confirmation . . . . .	25
5.3	Shell Collection Creation . . . . .	26
5.4	Collection Creation Verification . . . . .	26
5.5	Client JSON File . . . . .	27
5.6	Branches JSON File . . . . .	27
5.7	Products JSON File . . . . .	28
5.8	Orders JSON File . . . . .	28
5.9	Stock Requests JSON File . . . . .	28
5.10	Client Collection Data Import . . . . .	29
5.11	Branch Collection Data Import . . . . .	29
5.12	Product Collection Data Import . . . . .	29
5.13	Order Collection Data Import . . . . .	29
5.14	Stock Request Collection Data Import . . . . .	29
5.15	Client Data Import Confirmation . . . . .	30
5.16	Branch Data Import Confirmation . . . . .	30
5.17	Product Data Import Confirmation . . . . .	31
5.18	Order Data Import Confirmation . . . . .	31
5.19	Stock Request Data Import Confirmation . . . . .	31
5.20	Document Statistics of Collections . . . . .	32
5.21	Nexart Database Creation Mongo Compass Confirmation . . . . .	32
5.22	Created Collections Visual Confirmation . . . . .	33

5.23	Branch Imported Data Visual Confirmation . . . . .	33
5.24	Client Imported Data Visual Confirmation . . . . .	33
5.25	Product Imported Data Visual Confirmation . . . . .	34
5.26	Order Imported Data Visual Confirmation . . . . .	34
5.27	Stock Request Imported Data Visual Confirmation . . . . .	35
6.1	Technical Stack . . . . .	36
6.2	Integrated Development Environment . . . . .	37
6.3	Project File Structure . . . . .	37
6.4	Admin Product Management Wireframe . . . . .	39
6.5	Routers Wireframe . . . . .	39
6.6	Switches Wireframe . . . . .	40
6.7	Firewalls Wireframe . . . . .	40
6.8	Index Page . . . . .	41
6.9	Product Management Page . . . . .	42
6.10	Routers Page . . . . .	42
6.11	Switches Page . . . . .	43
6.12	Firewalls Page . . . . .	43
6.13	Wireless LAN Controllers Page . . . . .	44
6.14	Wireless Access Points Page . . . . .	44
6.15	Accessories Page . . . . .	45
6.16	Server JS Backend . . . . .	45
6.17	Admin Product Management Backend . . . . .	46
7.1	Question 1.1 Answer . . . . .	50
7.2	Question 1.2 Answer . . . . .	50
7.3	Question 1.3 Answer . . . . .	51
7.4	Question 1.4 Answer . . . . .	51
7.5	Question 2.1 Answer . . . . .	51
7.6	Question 2.3 Answer . . . . .	51
7.7	Question 4.1 Answer . . . . .	51
7.8	Question 3.1 Answer . . . . .	52
7.9	Hypotheses 1 . . . . .	52
7.10	Hypotheses 2 . . . . .	52
7.11	Hypotheses 3 . . . . .	53
7.12	Hypotheses 4 . . . . .	53

# Chapter 1

## Introduction

### 1.1 Problem Context Status

Nexart is an enterprise which manufactures and sells networking devices and accessories to clients. This enterprise was started in 2000 by Alexander Watson. Nexart branches are located all around the globe and the main branch is situated in California, USA.



Figure 1.1: Nexart Enterprise Logo

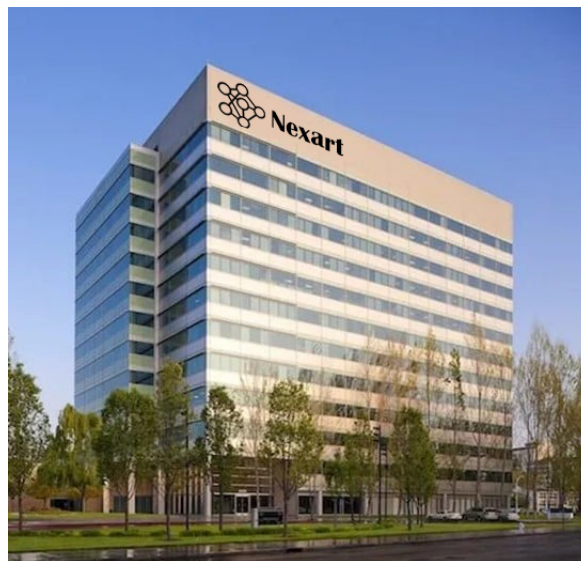


Figure 1.2: Nexart Head Office California



Figure 1.3: Nexart Production Plant California

Manufactured devices and accessories are distributed to Nexart subbranches of other nations via shipping according to the requests of subbranches. Network infrastructure installation companies play the role of Nexart clients. They provide services to government and non-government organizations. After getting quotations approved from the clients, they place orders by logging onto Nexart website. The clients choose their nearest Nexart branch and select the network equipment that they require for the job and proceed to make payments. They fill out the payment details and finalize the payment. Nexart send an email to the client informing about the order that the customer has placed, and the order details include payment amount and network equipment details. In addition, Nexart also provide shipping details and order tracking details in the mail. So that the clients can keep track of the orders. Each of the Nexart subbranches have a stock. When they are out of stock, they order items from their nearest subbranch. In this manner, Nexart optimize their shipping costs. In comparison to past years, the revenue of Nexart has skyrocketed drastically.

Nexart enterprise uses the relational database management system, Microsoft SQL Server to store the records of clients, branches, products, orders, and stock requests. The attributes inherited by the client entity are client id, name, contact information, billing address and registered date. Features inherited by the branch are branch id, name, location, contact information and stock. Device object is inherited by the product id, name, description, and price. The characteristics owned by the order entity are order id, order date, total amount, items, payment, and shipping. The attributes which belong to the Stock Request are stock request id, request date and quantity requested. As the customer base of the Nexart grew rapidly, the databases were unable to handle the requests from clients all around the globe. This resulted in latencies in database operation, lack of interest in customers, losing business edge, increased expenses and under performance of businesses. Apart from the business impacts, the Nexart also suffers from scalability, query latencies, concurrency limits, cost efficiency and schema rigidity issues. Complexity issues and errors have occurred due to maintaining integrity and maintenance. So, the Nexart clients were not satisfied with the service, and they turned towards their main competitor Cisco. This resulted in a revenue decline, and they had to plead with customers to purchase items from them. The CEO of Nexart wanted to get things back on track and he escalated the issue to the IT Department. The data engineers of the IT department started to narrow down issues step by step, isolating irrelevant factors. This process



is discussed thoroughly in the next section.

## 1.2 Business Questions

Data engineers assess the allegations of their clients regarding the web application. With the corporation of network engineers, they monitor network traffic. According to the observation, they noticed that the increase of their clients compared to past. In addition, they notice that Nexart generates bulks of data compared to past. As they produce new network devices and accessories for the market, new features are added to the device table. This results in storage expansion and order processing delays were caused due to the number of queries that I have been fired towards the MSQL database during peak hours. Nexart did not encounter this issue before because of the lower customer base and limited devices on the production line. The issue can be assessed with the following questions.

- What factors caused the revenue loss?
- Why did the revenue decline occur?
- Where did the revenue decline occur?
- When did the revenue decline occur?
- Who is part of the revenue decline?
- What are the measures that can be taken to enhance the accuracy of real-time data processing and downsize query latency?
- How much expenditure can be saved from migrating all the data that is in the MSSQL to MongoDB database?
- What actions can be taken to manage data storage effectively in terms of scaling data?
- How does the database can manage the real-time sensor data that is being generated in production?
- How to ensure redundant connectivity to the database as the number of users grows?
- How does the latencies in query responses can affect customer satisfaction?

Every bit of data generated by the Nexart may possess a value. So, it would be unwise to get rid of the data without accurately assessing it. The next chapter discusses the ways of managing Nexart's generated data in an effective manner.

# Chapter 2

## Databases

### 2.1 Concepts

Nexart stores their daily generated data in a database before processing it in a database. Organized data stored as soft copies in a computer is defined as a database. To manage a database, a Database Management System is required.

They are categorized into two categories, and they are Relational, Non-Relational, NewSQL, In-Memory, Time-Series and Object-oriented Databases. NewSQL database is considered as an enhanced version of a relational database. Scalability acts as the icing for these databases. Google Spanner and CockroachDB are examples for NewSQL databases. Main memory is used by Memory databases to store data. Perfect instances for NewSQL databases are Redis and Memcached. Data generated by IoT devices and financial transactional data are stored in Time-Series databases and time stamped. InfluxDB and TimeScaleDB are examples of TimeSeries databases. Data are stored as objects in Object-Oriented databases. db40 and ObjectDB are instances for Object-Oriented Databases.

Single-Tier, Two-Tier, Three-Tier and N-Tier are database Architectures. In a Single-Tier architecture, the database and application are hosted in the same machine and deployed in development/testing environments. Direct communication between the client and the server is maintained in a Two-Tier architecture and its applications are applications which fall in the range of small to medium. An intermediary layer is added to enhance the capabilities of a Three-Tier architecture, and they are used in web applications, enterprise applications etc. As an enhancement Three-Tier designs add an intermediary layer. N-Tier architecture is used to address complex commercial issues that are related to databases. This architecture surpasses the third architecture by adding more layers. Mostly conglomerate applications use N-Tier architecture for their systems.

The components of the database are Database Engine, Database Schema, Query Processor, Storage Manager, Transaction Manager, Concurrency Control Manager and Recovery Manager. The Database Engine is responsible for accessing and managing the database and this service is considered a core service. The Logical structure of a database is defined as the Logical Schema of a database and it includes tables, views etc. The mechanism which is responsible for interpreting and executing queries is the Query Selector. The storage manager plays the role of managing the

disk space. The transaction manager makes sure that all the transactions happen according to the properties of ACID. The consistency of simultaneous data access is managed by the concurrency control manager. When a database failure occurs, the recovery manager restores the database back to its normal condition.

According to the design aspect of databases, they can be divided into three categories, and they are Conceptual Design, Logical Design and Physical Design. A perfect example for a conceptual design would be entity relationship diagrams. These diagrams illustrate the data model. The Logical Design demonstrates the database in the form of tables, views etc.

To avoid redundancies and enhance data integrity normalization is carried out. 1st, 2nd, 3rd, and 4th normalization are examples of normalization types.

The data are efficiently retrieved by using the database indexing mechanism. Primary, secondary, clustered, and non-clustered indexing are instances for database indexes. Authentication, Authorization, Encryption and Backup and Recovery mechanisms take care of the security just in case of a security breach. Authentication is responsible for verifying the identity of the user while the actions that the verified user can perform are granted by authorization. Encryption mechanism encrypts the data using hashing algorithms before storing or transporting them in the database. Backup and recovery mechanisms make sure that the data can be restored back to its normal condition in case of data loss or corruption. Full back up, incremental backup and differential backup are examples for database backup methods. The entire database is copied to a storage device in full backup method. Incremental backups copy the recently altered data to a storage device. The data changed since the last full backup is copied to a storage device in differential backup method. Restoring data to a certain point in time is defined as point-in-time recovery. Recovering recently committed data by examining the transaction logs is part of the log-based recovery method. Cloud, Distributed and Graph databases are considered as modern trends of databases.

The accurate processing of database transactions is assured by the properties of ACID and their atomicity, consistency, isolation, and durability. A transaction is treated as a one unit in Atomicity while all the standards are maintained under consistency. Isolation mechanism compartmentalizes each transaction until they terminated. Fault tolerances in transactions are ensured by the durability property. The next section draws out the advantages and disadvantages of relational and non-relational databases.

## 2.2 Relational and Non-Relational Databases

Structured data are stored in relational databases using columns and rows while semi-structured and unstructured data are stored in non-relational databases. Structured Query Language is used by relational databases to perform all the database operations while Not Structured Query Language is used by non-relational databases to perform all the database operations. MySQL, PostgreSQL, Oracle and SQL Server are examples of relational databases. There are four types of non-relational databases, and they are Document Stores, Key-Value Stores, Column Stores and Graph Databases. MongoDB and CouchDB are examples for Document Stores while Redis and DynamoDB are instances for Key-Value Stores. Cassandra and HBase are examples for column store NOSQL

databases. Neo4j and ArangoDB are the best examples for graph databases. The table below highlights the main differences between SQL and NOSQL.

SQL	NOSQL
Used in relational databases	Used in non-relational databases
Has static schema design and structure	Has dynamic schema design and structure
Handle complex queries	Handle large volumes of data
Equipped with vertical scalability	Equipped with horizontal scalability
Follows ACID property	Follows CAP property

Table 2.1: Pros and Cons of SQL and NOSQL

Relational and non-relational databases such as MSSQL and MongoDB have ACID in common. Though a transaction is complete or rolled back, atomicity ensures that the database is maintained in a consistent state in MSSQL. Atomicity is applied in a single document update process in MongoDB. Constraints, triggers, and rule mechanisms ensure that consistency is maintained in MSSQL. Consistency is applied for multi-documents in MongoDB, and it is offered via flexible schema design which enables read and write operations. Transparency of uncommitted changes to other transactions is maintained via isolation mechanism in MSSQL. Snapshot mechanism in MongoDB ensures isolation in multi-document transactions. Transactions logs and database backups ensure that durability is maintained in MSSQL while journaling and replica sets ensure durability In MongoDB. The benefits and drawbacks of relational and non-relational databases are discussed in the next section.

## 2.3 Strength and Weaknesses of Relational and Non-Relational Databases

The table given below spotlights the pros and cons of relational databases.

Advantages	Disadvantages
Provides a structure to manage data	Limited Scalability
Compatible with ACID	Performance issues
Extensive Query Capabilities	Difficulty in changing the schema
Avoiding data redundancy and enhancing data integrity	High Cost
Capability of mapping complex relationships between entities	Difficulties in handling semi-structured and un-structured data
Technical Support	Complexity in applications
Security	Unability of handling datasets that scaling.

Table 2.2: Advantages and Disadvantages Relational Databases

The advantages and disadvantages of non-relational databases are emphasized in the table given below.

Advantages	Disadvantages
Scalability	Not Consistent
Availability	Consistency Issues
Higer Performance	Limited Query Capabilities
Flexibility	ACID not supported
Handling of structured and unstructured data	Limited support
Low Expenditure	Complexity in terms of data modeling

Table 2.3: Benifits and Drawbacks of Non-Relational Databases

We have already acknowledged the benefits and drawbacks of relational and non-relational databases. Next step would be to select the right database type for your application and the next chapter addresses about it.

## 2.4 Right Database Option for an application

Applications use relational or non-relational databases to store data according to the project requirements. As we acknowledged before, relational, and non-relational databases use SQL and NOSQL to perform all database operations. If an application has structured data with a static schema and requires multirow transactions, it is advisable to deploy a relational database such as MSSQL. The developers still can go with non-relational databases, but it can limit the functionality of the application sometimes. It may take a toll on the project budget sometimes. If a solution has a dynamic schema and holds semi-structured and unstructured data, it would be ideal to use a non-relational database such as MongoDB. The characteristic of a non-relational database supports an enterprise application which is scaling at a rapid rate. A relational database can be used for this application, but it will downgrade the capabilities of the application even if the option is less costly. NOSQL databases are ideal for solutions which require hierarchical storage structures, scalable features, and functions and where relationships between entities are not important. Maintenance of a database is important for an application to run smoothly without causing any issues. Operations such importing and exporting data occur in the management phase. Hence these tasks are repetitive, automation is required to boost the system's productivity. The next chapter suggests an enhancement to improve the efficiency of the Nexart system.

# Chapter 3

## Data Engineering

### 3.1 Data Engineering Concepts

CTO of Nexart decided to merge Data Science capabilities, Machine Learning and AI with their generated data to increase revenue and to anticipate future revenue losses of the organization. AI's are empowered with quality data. Machine Learning algorithms analyze data and make predictions. The principles of Data Engineering guide in the Data Analysis processes. Data Science and Data Engineering fields have common characteristics. The picture given below demonstrates aspects in IT where data science is applied.

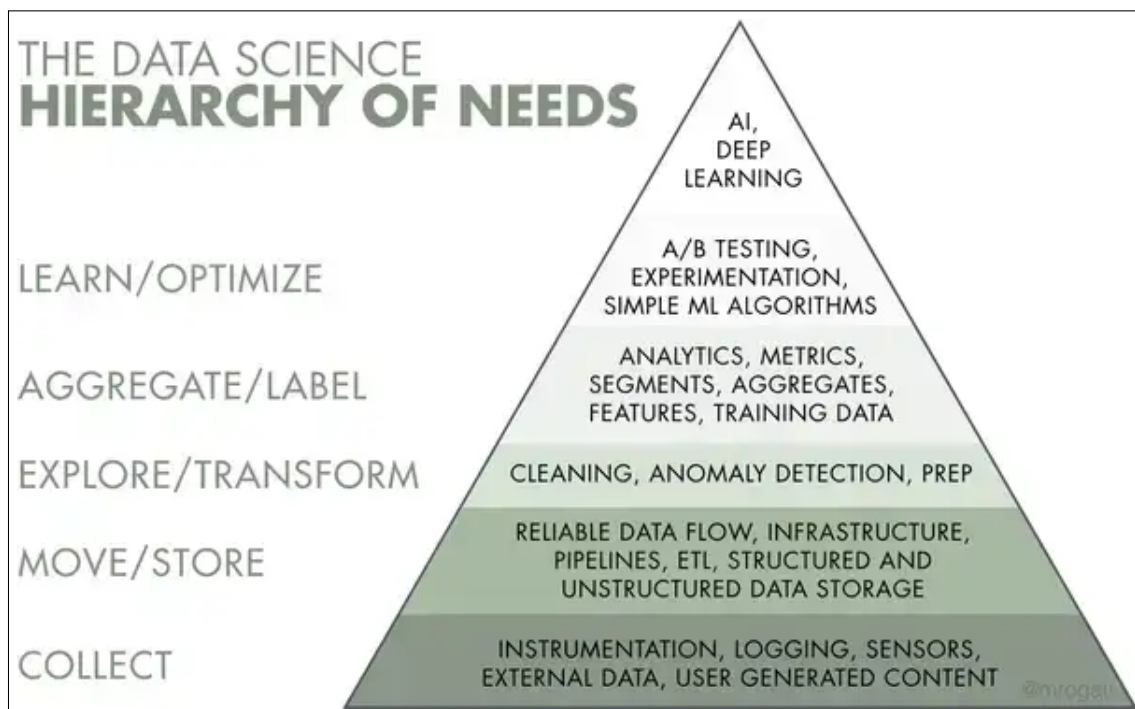


Figure 3.1: Hierarchy of Data Science Necessities

The data engineers of Nexart gather generated data from departments and store in the MongoDB database for data scientists, data analysts, business intelligence developers and other IT specialists

within the Nexart to access at any time. They chose the MongoDB database due to its scalability. The image given below demonstrates the steps that are followed in the data engineering process.



Figure 3.2: Data Engineering Processes

All the collected data from Nexart Departments will be moved to MongoDB database in the data ingestion state. In the data transformation stage, highly demandable data will be cleaned, normalized, and converted to the required format. The users of Nexart that requested data will be served with data in the data serving process. The transparency of data engineering process is maintained through data flow orchestration. This gives the privilege of monitoring data workflows and repairing data quality and performance issues. Automating the steps followed in the data engineering process is defined as a data engineering pipeline. Data from the MSSQL database is moved to MongoDB database with the aid of data engineering pipelines by coupling tools with operations. In addition, data engineers of Nexart could write a script to get weekly/monthly/yearly sales reports. Hence the generation of sales reports is a repetitive task.

Since Nexart enterprise is dealing with big data, the data engineers decided to go with a big data engineering pipeline that uses ELT method. The image below captures the whole ELT process.

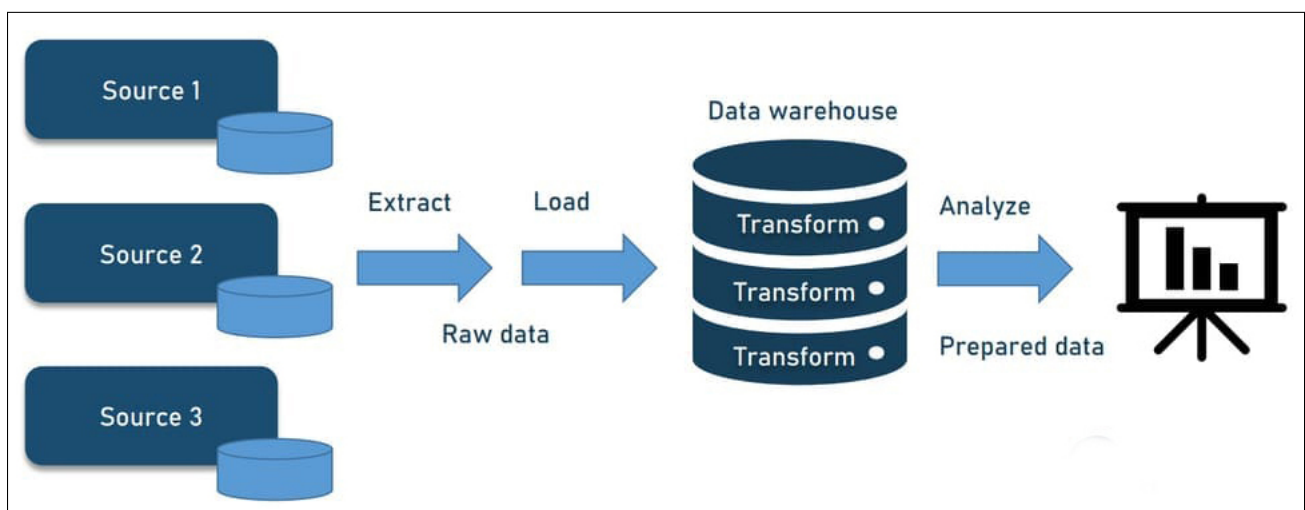


Figure 3.3: ELT Pipeline

ETL functions differently compared to ELT pipelines. We can observe the functionality of ETL from the image given below.

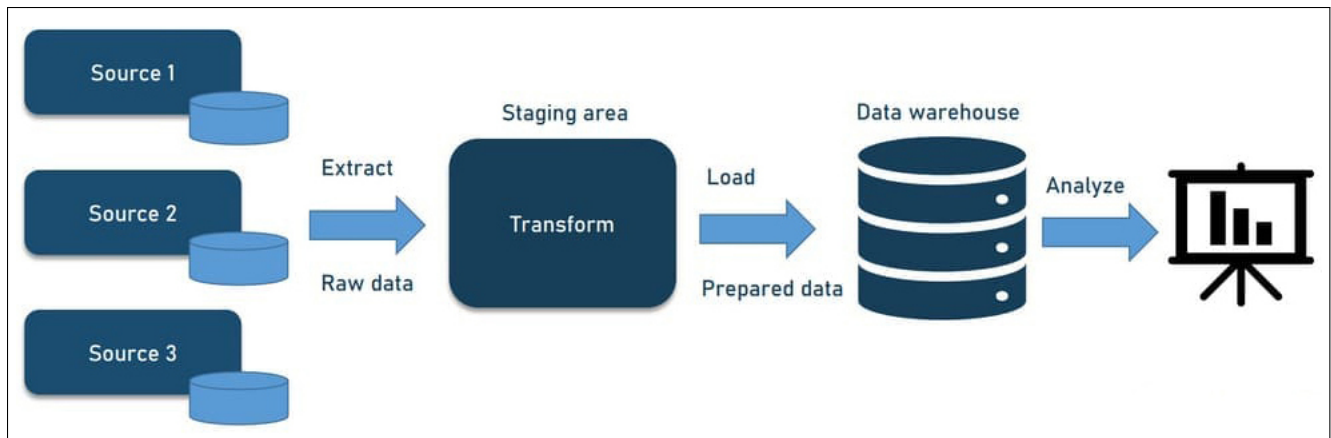


Figure 3.4: ETL Pipeline

Apart from data migrations, Nexart can deploy the pipeline for data wrangling, data integration and copying databases. A data pipeline consists of 7 components, and they are origin, destination, dataflow, storage, processing, workflow, and monitoring. MSSQL and MongoDB play the role of source and destination, respectively. The dataflow defines changes that data undergoes from the source to the destination. As data moves from MSSQL to MongoDB, the data will be stored in the lake.

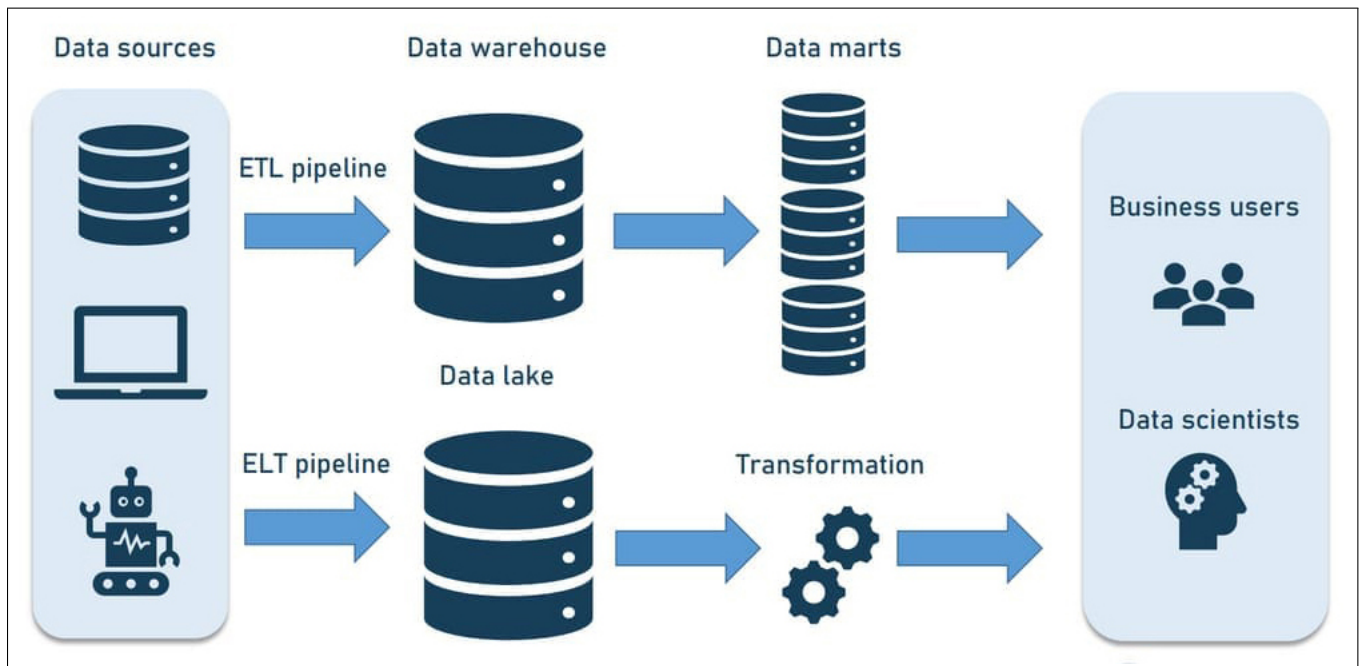


Figure 3.5: Data Lake with Big Data Infrastructure

Processing defines the ingestion, storing and transformation of data from the source to the destination. The workflow defines a set of activities that are carried out from the Source to the destination, and it also highlights dependencies that each task has on them. The monitoring mechanism ensures the data is in proper shape. It would be ideal for Nexart to maintain two data pipelines. One pipeline



should follow batch processing while the other should stream processing. In some instances, Nexart would have to investigate old records to make a business decision and sometimes the corporation would have to deal with real-time data such as data generated from robots on the production line. So, in this case it is better to maintain two data pipelines.

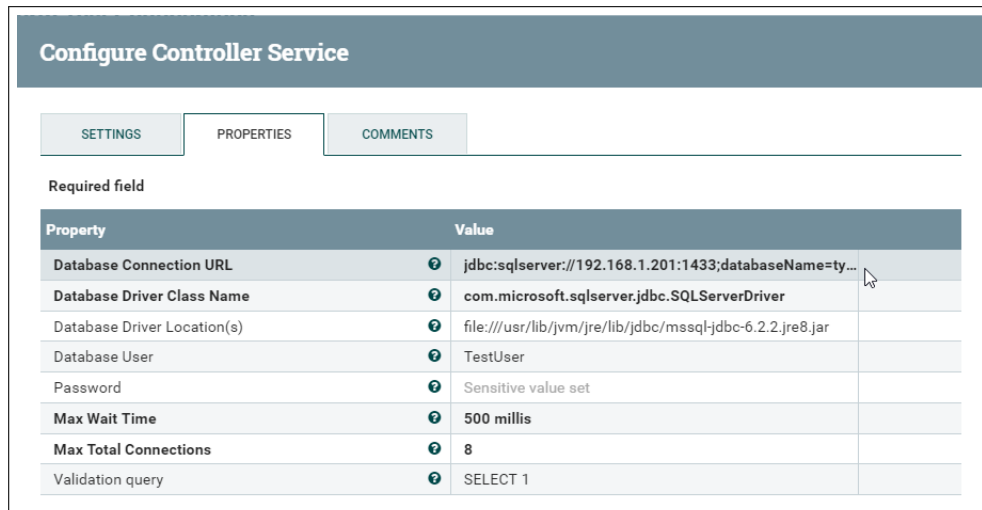
Data pipeline tools branches under ELT, Data Warehousing, Data Lake, Batch Work-flow Schedulers, Real-Time Data Streaming and Big Data tools. All the data preparation and integration tools fall under ETL tools and IBM DataStage Informatica power Center, Oracle Data Integrator, Talend Open Studio are examples for ETL tools. Instances for Data Warehouse tools are Amazon Redshift, Azure Synapse, Google BigQuery, Snowflake and Teradata. AWS, Microsoft Azure, Google Cloud, and IBM are examples of cloud service providers which offer data lakes as tools. Luigi and Azkaban are batch workflow schedulers which declare tasks programmatically along with their dependencies. In addition, they have the capability of monitoring and automating the tasks. Apache Kafka, Apache Storm, Google Data Flow, Amazon Kinesis, Azure Stream Analytics, IBM Streaming Analytics and SQL Stream are tools that are used process real-time data that are generated by machine sensors, IoT sensors etc ale [2023]. The data pipeline tools mentioned do not go well with Big Data. The software developers have developed specific tools to perform heavy duty operations such as prepping data pipelines for big data. Let us review each of these tools. Hadoop and Spark platforms are used for batch processing big data. The Spark streaming analytics service tool enhances the functionality of the Spark platform. Apache Oozie and Apache Airflow tools are used by data engineers to perform scheduling and monitoring for batch jobs. Cloud service providers such as Amazon, Google, IBM, Microsoft, and Alibaba offer their own tools for data experts to create big data pipelines. As we have acknowledged the DNA of data engineering, the next segment discusses how the principles of data engineering are applied towards the Nexart case study.

## 3.2 Data Engineering Pipeline

As we acknowledged before Nexart decided to go with the ELT method for both data pipelines. The data engineers of Nexart are not sure about the way of transforming data and they have an idea of generating insights into the future as well. Due to these reasons, they decided to go with the ELT method. Since Nexart is generating tons of data daily from the production line and other departments, they decided to go with tools that are specifically designed for big data. Let us discuss the data pipelines further.

To increase the revenue, Nexart needs valuable insights from data that been stored in the MSSQL database. These insights are being generated using machine learning. Models need to be fed loads of data to get better insights. So, the model requires processed data from the past and the present and they decided to go with a batch processing data pipeline to get business insights. Apache NiFi, MongoDB, Apache Spark, and Airflow tools will be used for the batch processing data pipeline. In this pipeline there are three phases. Data stored in the MSSQL database needed to be extracted firsthand. Before commencing the first phase, the system engineers of Nexart will determine the peak hours of service by analyzing incoming and outgoing traffic of Nexart. After that the data engineers of Nexart will extract data from MSSQL database according to off peak hours defined by

the data engineer using Apache NiFi software.



**Configure Controller Service**

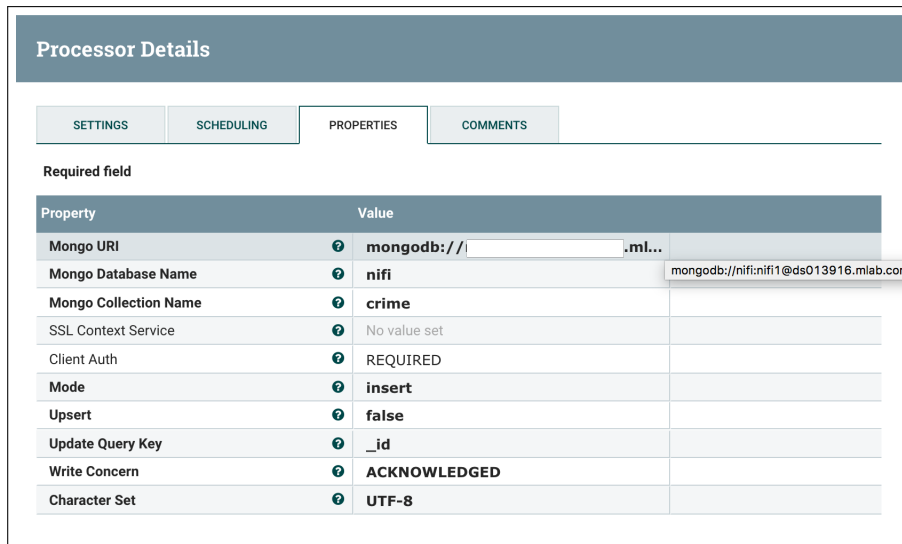
SETTINGS PROPERTIES COMMENTS

Required field

Property	Value
Database Connection URL	jdbc:sqlserver://192.168.1.201:1433;databaseName=ty...
Database Driver Class Name	com.microsoft.sqlserver.jdbc.SQLServerDriver
Database Driver Location(s)	file:///usr/lib/jvm/jre/lib/jdbc/mssql-jdbc-6.2.2.jre8.jar
Database User	TestUser
Password	Sensitive value set
Max Wait Time	500 millis
Max Total Connections	8
Validation query	SELECT 1

Figure 3.6: Apache NiFi Extracting Data From MSSQL Database

This application gives the privilege of setting up data extraction schedules. As a plus point this does not clash the ongoing services of Nexart during peak hours. Extracted data will be needed to store in a storage.



**Processor Details**

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field

Property	Value
Mongo URI	mongodb://[redacted].ml...
Mongo Database Name	nifi
Mongo Collection Name	crime
SSL Context Service	No value set
Client Auth	REQUIRED
Mode	insert
Upsert	false
Update Query Key	_id
Write Concern	ACKNOWLEDGED
Character Set	UTF-8

Figure 3.7: Apache NiFi Storing Extracted Data in MongoDB Database

The schemeless characteristic of MongoDB database addresses the storage problem by storing all extracted raw data from MSSQL database. This enables us to maintain the integrity of data. This

raw data does not have a meaning. So, these data needed transformed to get a better meaning. As a result, transformation takes place using Apache Spark software. All the transformations jobs will be carried out using Spark.

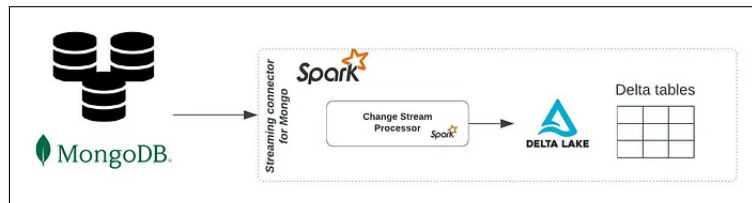


Figure 3.8: Apache Spark Transforming Data in MongoDB Database

Repetitive transformations such as generating weekly reports can be hectic sometimes. As a remedy all the transformation will be automated using Airflow application. Batch processing pipelines are extremely useful in generating weekly/month/yearly sales reports, salary reports etc.

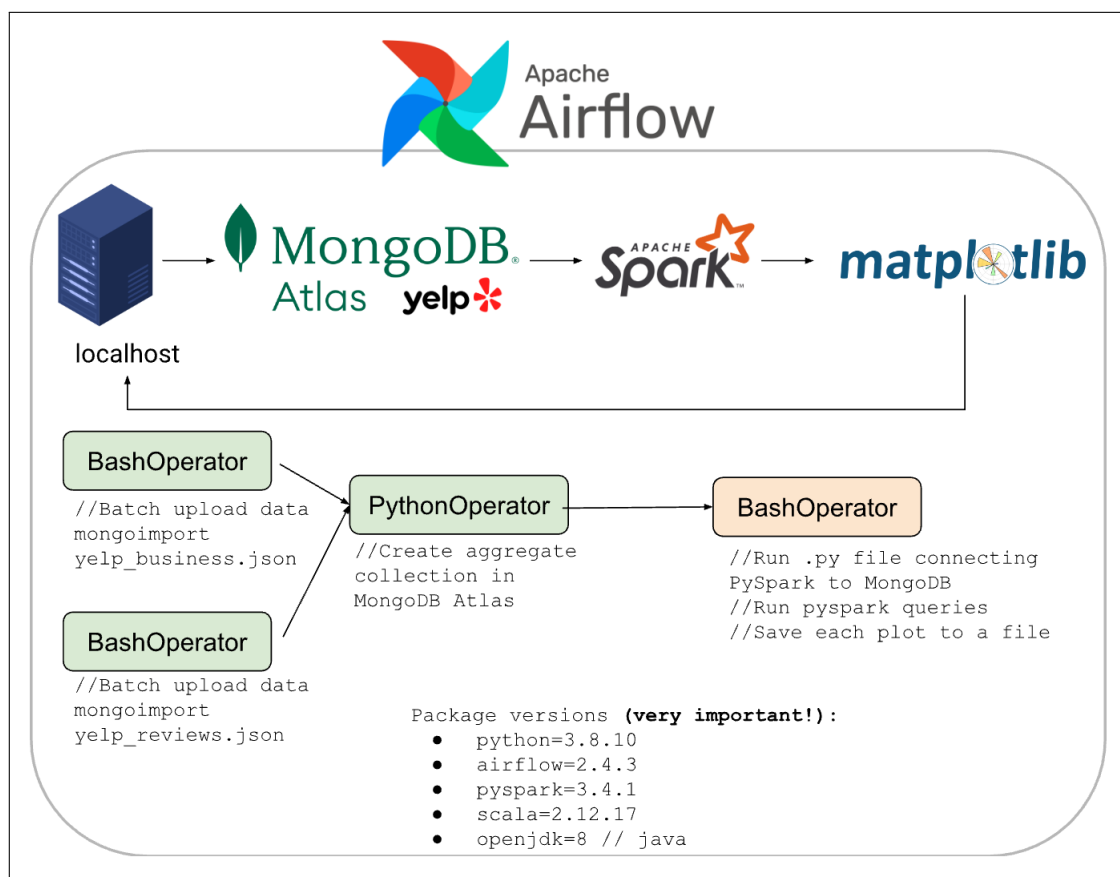


Figure 3.9: Apache AirFlow Automation

According to the golden rule of information, the value of data that is being generated stays constant for a certain period and the value of information starts to fade away with time. Robots in the production generate piles of data daily using sensors in real-time. These data may have a significant value. So, it would be unwise to ignore the data. Since the data experts are dealing with real-time data, they decided to deploy the stream processing method for the second pipeline. The ELT technique is used for the second pipeline due to the benefits offered. This pipeline has three phases too. The data produced by the robot sensors will be captured by the Apache Kafka software as events. Next the captured data will be stored in the MongoDB database in the loading phase. To transform the raw data to meaningful data, Apache Spark application will be used. In some case scenarios robots used to deviate from their normal operational task. The data migration timeline is divided into three stages: pilot phase, full rollout, and post migration optimization. A stream pipeline can solve this issue. As Apache Kafka extracts and stores data MongoDB database, Apache Spark will the raw data and detect for any anomalies in the real-time data. If the application detects an anomaly, it will alert the relevant IT professionals to the event. So that the IT professional could take the necessary steps to troubleshoot any issues in the robot or production line. The data migration timeline is divided into three stages, pilot phase, full rollout, and post migration optimization. In the pilot phase a small amount of data is migrated into MongoDB and tested. If the tested data is valid, all the data are migrated into the database in full rollout stage. Additional tinkering to the data is performed in the post migration optimization phase. Since Nexart have chosen MongoDB to store data, let us dig into the roots of MongoDB from the next section.

# Chapter 4

## MongoDB

### 4.1 History

MongoDB database application is an application developed by Mongo Inc. This organization was founded by Dwight Merriman, Eliot Horowitz, and Kevin Ryan in 2007 Plugge et al. [2010]. The first MongoDB application is introduced to the public as a platform between 2007 and 2008. Next Mongo Inc released their first NOSQL database in 2009. In 2017 MongoDB was listed in NASDAQ. MongoDB version 6 was released in 2022. The title of the application defines nature itself. The Humongous Ness of the application allows users to store a large quantity of data without worrying about the application's structure. MongoDB databases are used in IoT devices, mobile applications, real-time analysis, personalization, catalog management and content management. Enterprises such as Google, Facebook and Ebay deal with their growing data using MongoDB. Let us examine the marvel behind the MongoDB databases starting from the next chapter.

### 4.2 Concepts

The collections in MongoDB act as tables in relational databases while a document in a nonrelational database represents a row. A document has various fields labeled as columns in a relational database. Values are stored as key value pairs in MongoDB. Documents use the BSON format when storing values. The key features of MongoDB are aggregation, gridfs, sharding, document oriented, replication, schema less database, indexing, AD HOC Queries, and high performance noa [2018]. MongoDB uses Data Modeling to store data. The relations between entities and the way the data is stored are defined as data modeling. It helps to build logical databases that are efficient. These databases offer low storage requirements, grant efficient data retrieval and limit redundancy. Through the data modeling process, the following objectives are archived.

- High Data Quality
- Comprehension of dataflow and characteristics
- Development and Maintenance
- Performance

Data models are categorized into three types based on specificity/detail and they are conceptual, logical, and physical data models. One to One, one to many and Many to Many are relationship types that are used in data models. Embedded and Reference data models are two methods used for creating a model. Logical or mathematical operations are executed in MongoDB by using query operators. Query and projection, update and aggregation pipeline operators are examples for query operators.

Regular expressions in NOSQL queries are used to match patterns in a document. List given below points out reasons for using regular expressions.

- Match text and define search patterns in sequences of characters.
- Retrieve data of an undefined field.
- Locate small subsets of data within a collection.

The database administrators can view only the necessary data in a document using the projection feature of MongoDB. Projection features come in handy when removing unnecessary fields from query results, retrieving indexed query results without fetching full documents and filtering data without causing any database performance issues. The sorting method in MongoDB defines the order in which query returns the matching documents in collection. The number of documents retrieved can be limited by using the limit method. Indexes in MongoDB aid in query data in an efficient manner. Avoidance of collection scans, effectiveness of indexing strategies and searching efficiency are achieved through the indexing mechanism. Single field, compound and multi key index are examples of indexes. Creation of a single index in a document is done by using single field index while references to multiple fields in a document is obtained through compound index. The values stored in arrays are indexed by multi key indexes. The data retrieval speed is enhanced by the advanced indexing feature. Geospatial, text and hashed indexing are techniques used in advanced indexing. Geospatial coordinate data are handled by 2d indexes and 2d sphere indexes. Text search queries are powered by text indexes. Entries with hashes are indexed using hash indexes. Gaining accurate results through processing multiple documents by selecting data from a collection is defined as aggregation. Aggregation in MongoDB is performed using groups by clause and complex aggregation operations. It is performed to group values of multiple documents together and to fetch nested data to perform complex operations. In addition, it is used to filter and sort documents and analyze the changes in data. Match, group, sort and project are aggregation pipeline stages. Operations in aggregation are sum, average, minimum, maximum and push. Replication is the process of cloning the same data across multiple MongoDB servers. The intentions behind replication are listed below.

- Data redundancy and high availability.
- Making copies of data across servers.
- Performing data backups and recoveries.

Replication is performed to increase the availability of data and to dampen the effects of a single server loss. MongoDB manages replication through replica sets. Collection of MongoDB nodes are defined as MongoDB nodes. At least three MongoDB nodes are required to form a replica set.

All the writing operations are performed by the primary node while secondary nodes that are under the root node copy data from the primary node. When a primary node fails, it will be elected as a secondary node. The benefits and drawbacks of replication are discussed in the table given below.

Advantages	Disadvantages
Availability	Higher usage of storage and network bandwidth.
Scalability	Higher levels of complexity in operation, configuration, and monitoring.
Enhanced Performance	Lags in propagation.
Disaster Recovery	Challenges in maintenance.
Continuous Operations	Consistency Issues

Table 4.1: Benefits and Drawbacks of Replication

The list given below highlights the limitations of replication.

- Difficulties in handling data and query traffic.
- Inability of MongoDB instances to manage write operations.
- Inability to outsize memory regarding large datasets.
- Higher costs.

The distribution process of data in MongoDB is designated as Sharding. A single dataset is stored in multiple databases. Sharding process adds more memory and processing units or ram into a single server. Sharding is associated with two types of scaling methods, and they are vertical and horizontal scaling. Adding more resources to the server is defined as vertical scaling while horizontal declares of adding more processing units or physical machines to the server or database. Sharding uses an architecture, and this architecture uses sharding clusters. A sharding cluster is formed out of multiple shards, mongos processes and configuration servers. Shard key distributes data among the shards and a shard key is automatically created by the database. The sharding process creates a cluster. A cluster of MongoDB instances forms at least three servers. The list below addresses the pros of sharding.

- Increased storage capacity.
- Increased read/write throughput.
- High availability.
- Facilitates horizontal scaling.

The way multiple documents are logically connected to each other in MongoDB is represented in relationships. Database structures are refined by creating relationships between documents. In addition, they link up the entities in a database and make execution times shorter. Relationships in documents are categorized into embedded and reference relationships. Reference relationships are

achieved through many to many relationships while embedded relationships are achieved through one to one and one to many relationships. Files that are greater than 16MB are stored using GridFS driver in MongoDB. GridFS mechanism divides large files into smaller chunks and stores them as separate documents. The list given below addresses the probable causes of using GridFS.

- Giving the privilege of storing files that are greater than 16MB.
- Giving the access to a portion of a file without loading the whole file.
- Storing and syncing files and metadata across distributed systems.

By default, GridFS uses fs files and fs chunks to store the file's metadata and the chunks. The data processing paradigm for condensing large volumes of data into useful aggregated results is defined as Map Reduce. This technique is mostly used for large datasets. Map, reduce and query functions are executed MapReduce method. Referencing documents stored in multiple collections or databases is performed by using the DBRef option. Ref, ID, and DB parameters are used with the DBRef option. Cover queries use indexed fields to gain efficient results without searching a whole collection. Given below are scenarios when cover queries comes in handy.

- When all the fields in query are part of an index.
- When all the fields returned in query are in the same index.
- When there are no null values in fields.

All data which require an update is stored within a single document to maintain the atomicity. There are certain constraints regarding the atomicity of MongoDB, and they are listed below.

- Zero support for multi document atomic transactions.
- Only allows to perform atomic operations in a single document.
- Atomocity is maintained and executed at the document level.

Set, inc, push, pull and rename are commands used in performing atomic operations. High throughput operations that insert retrieve and delete documents based on insertion order are stored as fixed collections under the capped collections. When collection size reaches over its maximum threshold, it starts to delete the old data autonomously. Due to the following reasons capped collections are used.

- To make sure that recently entered data stay in the database.
- Automate the old data removal process.
- Store cached data that needs regular refreshing.

Capped collections have their own characteristics, and they are listed below.

- Delete operations not allowed to be performed unless it is removed automatically.
- All elements in a collection should have an equal size.



- No indexing is required because of working queues.
- Helps to maintain log files to troubleshoot any errors.

Mechanisms such as authorization and authentication safeguard the MongoDB database from unauthorized access. MongoDB gives the capability to data experts to create users, grant access control for the created users and authenticate users.

Sectionizing a collection of related commands which should be executed with similar session options are defined as session commands. Examples for session commands are `abortTransaction`, `commitTransaction`, `endSessions`, `killAllSessions`, `killAll SessionByPatterns`, `refreshSession`, `killSessions` and `startSession`. Undoing modifications and restoring the database back to its normal condition is performed by `abortTransaction` command. The `commitTransaction` command saves the changes made by operations in a multi-document transaction before ending the transaction. Before terminating a session, the `endSession` command overrides the timeout window. Ongoing operations in a session are terminated by using the `killSession` command. Any sessions that are twined with declared patterns are terminated by the `killSessionbyPattern` command. MongoDB tools such as `bsondump`, `mongodump`, `mongoexport`, `mongofiles`, `mongoimport`, `mongorestore`, `mongostat` and `mongotop` help data engineers in the process of backing up and restoring the databases. MongoDB Atlas is a cloud-based data storage service. Management and monitoring of MongoDB deployments in enterprises are carried out by the MongoDB Ops Manager tool. MongoDB databases have pros and cons. They are discussed in the next section thoroughly.

### 4.3 Advantages and Disadvantages

The MongoDB inherits its benefits and drawbacks, and they are highlighted in the table below.

Advantages	Disadvantages
Open source	Issues related to consistency.
Easy to use	Consumes a lot of resources.
Highly flexible	Less capabilities in creating relations.
Advanced security features	Complexity of managing transactions.
High Availability	Less compatibility
Reliable Indexing	Maintenance and management
Flexible Schema	
High Performance	

Table 4.2: Advantages and Disadvantages of MongoDB

The next section discusses the application of MongoDB's concepts towards the Nexart's web application.

# Chapter 5

## MongoDB Application

### 5.1 Database

We have already acknowledged the issue faced by the Nexart organization. To overcome the issue, the data engineers of Nexart changed the architecture of the database and they moved towards a non-relational database such as MongoDB. The data engineers understood they must make some alterations to meet the growing business' demands. So, they decided to migrate their data to MongoDB. The command given below creates a database in MongoDB database.

```
test> use nexart
switched to db nexart
nexart> |
```

Figure 5.1: Nexart Database Creation

The creation of nexart database can be evaluated by executing the command given below.

```
test> show dbs
admin      40.00 KiB
buzz       72.00 KiB
config     108.00 KiB
gui         820.00 KiB
local      128.00 KiB
nexart      8.00 KiB
test       664.00 KiB
test>
```

Figure 5.2: Nexart Database Creation Confirmation

In prior to the migration, the database engineers map entities, and their attributes of the MSSQL database to MongoDB database. This process is discussed further in the collections section.

## 5.2 Collections

According to concepts of relational databases, all the entities are defined as tables. These tables were converted into collections in the MongoDB database. The tables in the MSSQL database inherited their own columns. These columns were mapped to the fields of the corresponding collections. Each table of the MSSQL database consists of 100 records each. These records are mapped to documents in MongoDB database. According to this scenario clients, branches, products, orders, and stock requests represent tables in relational databases while they are declared as collections in non-relational databases. The figure given below demonstrates the commands that should be executed to create the mentioned collections above.

```
nexart> db.createCollection('branch')
{ ok: 1 }
nexart> db.createCollection('product')
{ ok: 1 }
nexart> db.createCollection('client')
{ ok: 1 }
nexart> db.createCollection('order')
{ ok: 1 }
nexart> db.createCollection('stock_request')
{ ok: 1 }
nexart>
```

Figure 5.3: Shell Collection Creation

The created collections can be verified by executing the command given below.

```
nexart> show collections
branch
client
order
product
stock_request
nexart> |
```

Figure 5.4: Collection Creation Verification

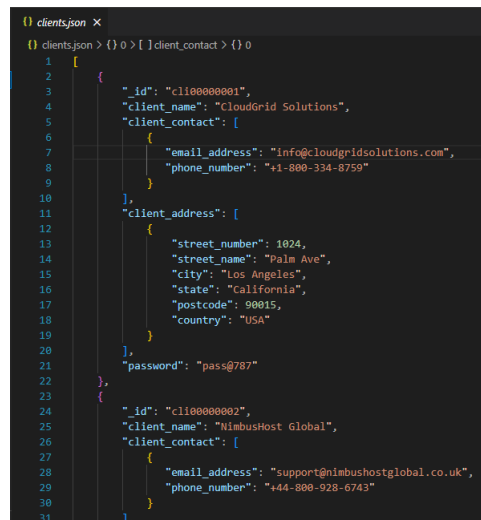
The next segment discusses the application of Mongo Shell regarding the development of Nexart's web application.

## 5.3 Mongo Shell

Before migrating data into the database, a database and tables should be created. While creating the tables primary keys, foreign keys and composite primary keys should be defined. So that it enables us to map the relations between the entities. The database administrators in the past used a command line interface to interact with the application. This applies to the database management systems in the past as well. Before the graphical user interface of MSSQL became known, the administrators used MSSQL cmd to create databases, tables etc. The Shell of MongoDB performs

the same role as the CMD of MSSQL. When creating tables, they were required to provide a structure for the tables. Since MongoDB handles semi-structured and unstructured data well, there is no need to define the table structure. Even though the database, collections and column names should be named properly. So that it would help the software developers of the IT department to populate the data of the database in the Nexart web application. Queries are inserted and executed in Mongo Shell to perform all the database operations. As the collection creation phase is completed, the migration process remains. Data experts of Nexart exported the records in the MSSQL database to json files. The list below shows the data extracted from the tables.

- clients.json



```

1 clients.json x
2 {} clients.json > {} 0 > {} client_contact > {} 0
3 {
4   "_id": "cli00000001",
5   "client_name": "CloudGrid Solutions",
6   "client_contact": {
7     "email_address": "info@cloudgridsolutions.com",
8     "phone_number": "+1-800-334-8759"
9   },
10  "client_address": {
11    "street_number": 1024,
12    "street_name": "Palm Ave",
13    "city": "Los Angeles",
14    "state": "California",
15    "postcode": 90015,
16    "country": "USA"
17  },
18  "password": "pass@787"
19 },
20 {
21   "_id": "cli00000002",
22   "client_name": "NimbusHost Global",
23   "client_contact": {
24     "email_address": "support@nimbushostglobal.co.uk",
25     "phone_number": "+44-800-928-6743"
26   },
27   "client_address": {
28     "street_number": 50,
29     "street_name": "Dexter Avenue",
30     "city": "Montgomery",
31     "state": "Alabama",
32     "postcode": 36101,
33     "country": "USA"
34   },
35   "branch_contact": {
36     "email_address": "alabama@nexart.com",
37     "phone_number": "(205) 555-1234"
38   },
39   "stock": [
40     {
41       "product_id": "prd00000001",
42       "quantity": 100
43     }
44   ],
45   "branch_password": "bra@100"
46 },
47 {
48   "_id": "bra00000001",
49   "branch_name": "Nexart Alabama",
50   "branch_address": {
51     "street_number": 50,
52     "street_name": "Dexter Avenue",
53     "city": "Montgomery",
54     "state": "Alabama",
55     "postcode": 36101,
56     "country": "USA"
57   },
58   "branch_contact": {
59     "email_address": "alabama@nexart.com",
60     "phone_number": "(205) 555-1234"
61   },
62   "stock": [
63     {
64       "product_id": "prd00000001",
65       "quantity": 100
66     }
67   ],
68   "branch_password": "bra@100"
69 },
70 {
71   "_id": "bra00000002",
72   "branch_name": "Nexart Arizona",
73   "branch_address": {
74     "street_number": 50,
75     "street_name": "Dexter Avenue",
76     "city": "Montgomery",
77     "state": "Alabama",
78     "postcode": 36101,
79     "country": "USA"
80   },
81   "branch_contact": {
82     "email_address": "alabama@nexart.com",
83     "phone_number": "(205) 555-1234"
84   },
85   "stock": [
86     {
87       "product_id": "prd00000001",
88       "quantity": 100
89     }
90   ],
91   "branch_password": "bra@100"
92 },
93 ]

```

Figure 5.5: Client JSON File

- branches.json



```

1 branches.json x
2 {} branches.json > ...
3 {
4   "_id": "bra00000001",
5   "branch_name": "Nexart Alabama",
6   "branch_address": {
7     "street number": 50,
8     "street name": "Dexter Avenue",
9     "city": "Montgomery",
10    "state": "Alabama",
11    "postcode": 36101,
12    "country": "USA"
13  },
14  "branch_contact": {
15    "email_address": "alabama@nexart.com",
16    "phone_number": "(205) 555-1234"
17  },
18  "stock": [
19    {
20      "product_id": "prd00000001",
21      "quantity": 100
22    }
23  ],
24  "branch_password": "bra@100"
25 },
26 {
27   "_id": "bra00000002",
28   "branch_name": "Nexart Arizona",
29   "branch_address": {
30     "street number": 50,
31     "street name": "Dexter Avenue",
32     "city": "Montgomery",
33     "state": "Alabama",
34     "postcode": 36101,
35     "country": "USA"
36   },
37   "branch_contact": {
38     "email_address": "alabama@nexart.com",
39     "phone_number": "(205) 555-1234"
40   },
41   "stock": [
42     {
43       "product_id": "prd00000001",
44       "quantity": 100
45     }
46   ],
47   "branch_password": "bra@100"
48 },
49 ]

```

Figure 5.6: Branches JSON File

- products.json

```
1 products.json x
2 {} products.json > ...
3 [
4   {
5     "_id": "prd00000001",
6     "product_type": "switch",
7     "product_model": "NEX-L2S-GE-1",
8     "product_name": "Nexart Layer 2 Ser1 Managable Switch",
9     "product_description": "high performance layer 2 first series switch",
10    "product_price": 210.5,
11    "product_image": ""
12  },
13  {
14    "_id": "prd00000002",
15    "product_type": "switch",
16    "product_model": "NEX-L2S-GE-2",
17    "product_name": "Nexart Layer 2 Ser2 Managable Switch",
18    "product_description": "high performance layer 2 second series switch",
19    "product_price": 220.5,
20    "product_image": ""
21  },
22  {
23    "_id": "prd00000003",
```

Figure 5.7: Products JSON File

- orders.json

```
1 orders.json x
2 {} orders.json > {} 7 > [ ] shipping > {} 0 > carrier
3 [
4   {
5     "_id": "ord00000001",
6     "order_date": "2/1/2024",
7     "total_amount": 1052.5,
8     "client_id": "cli00000001",
9     "branch_id": "bra00000100",
10    "items": [
11      {
12        "product_id": "prd00000001",
13        "quantity": 5,
14        "price": 210.50
15      }
16    ],
17    "payment": [
18      {
19        "payment_id": "pay00000001",
20        "amount": 1052.50,
21        "method": "visa",
22        "payment_date": "2/1/2024"
23      }
24    ],
25    "shipping": [
26      {
27        "shipping_id": "shp00000001",
28        "shipping_date": "2/2/2024",
29        "tracking_number": "tra00000001",
30        "carrier": "UPS"
31      }
32    ]
33  },
34  {
```

Figure 5.8: Orders JSON File

- stock requests.json

```
1 stock_requests.json x
2 {} stock_requests.json > ...
3 [
4   {
5     "_id": "str00000001",
6     "request_date": "2/1/2024",
7     "quantity_requested": 5,
8     "source_branch_id": "prd00000100",
9     "destination_branch_id": "prd00000001",
10    "product_id": "prd00000001"
11  },
12  {
13    "_id": "str00000002",
14    "request_date": "2/2/2024",
15    "quantity_requested": 3,
16    "source_branch_id": "prd00000000",
17    "destination_branch_id": "prd00000002",
18    "product_id": "prd00000002"
19  },
20  {
21    "_id": "str00000003",
```

Figure 5.9: Stock Requests JSON File

Since the MongoDB documents are represented in BSON format, MongoDB is compatible with handling json files as well. In the next phase all json files are imported into the relevant collections of the Nexart database in MongoDB server using mongoimport tool. The progress of the data imports is given below.

- Client collection data import

```
PS D:\Program Files\MongoDB\Server\7.0\bin> mongoimport --db nexart --collection client --file clients.json --jsonArray
2024-09-30T20:22:12.867+0530    connected to: mongodb://localhost/
2024-09-30T20:22:12.919+0530    100 document(s) imported successfully. 0 document(s) failed to import.
PS D:\Program Files\MongoDB\Server\7.0\bin> |
```

Figure 5.10: Client Collection Data Import

- Branch collection data import

```
PS D:\Program Files\MongoDB\Server\7.0\bin> mongoimport --db nexart --collection branch --file branches.json --jsonArray
2024-09-30T20:17:28.069+0530    connected to: mongodb://localhost/
2024-09-30T20:17:28.107+0530    100 document(s) imported successfully. 0 document(s) failed to import.
PS D:\Program Files\MongoDB\Server\7.0\bin>
```

Figure 5.11: Branch Collection Data Import

- Product collection data import

```
PS D:\Program Files\MongoDB\Server\7.0\bin> mongoimport --db nexart --collection product --file products.json --jsonArray
2024-09-30T20:29:35.109+0530    connected to: mongodb://localhost/
2024-09-30T20:29:35.182+0530    100 document(s) imported successfully. 0 document(s) failed to import.
PS D:\Program Files\MongoDB\Server\7.0\bin> |
```

Figure 5.12: Product Collection Data Import

- Order collection data import

```
PS D:\Program Files\MongoDB\Server\7.0\bin> mongoimport --db nexart --collection order --file orders.json --jsonArray
2024-09-30T20:36:18.230+0530    connected to: mongodb://localhost/
2024-09-30T20:36:18.299+0530    100 document(s) imported successfully. 0 document(s) failed to import.
PS D:\Program Files\MongoDB\Server\7.0\bin> |
```

Figure 5.13: Order Collection Data Import

- Stock Request data import

```
PS D:\Program Files\MongoDB\Server\7.0\bin> mongoimport --db nexart --collection stock_request --file stock_requests.json --jsonArray
2024-09-30T20:41:33.323+0530    connected to: mongodb://localhost/
2024-09-30T20:41:33.357+0530    100 document(s) imported successfully. 0 document(s) failed to import.
PS D:\Program Files\MongoDB\Server\7.0\bin> |
```

Figure 5.14: Stock Request Collection Data Import

Next the data engineers wanted to make sure wither the data is imported into the relevant collections. By executing the commands given below they verified the import process of each collection.

- Clients

```
nexart> db.client.find()
[
  {
    _id: 'cli00000003',
    client_name: 'SkyHub Networks',
    client_contact: [
      {
        email_address: 'contact@skyhubnetworks.com.au',
        phone_number: '+61-1800-112-234'
      }
    ],
    client_address: [
      {
        street_number: 42,
        street_name: 'George St',
        city: 'Sydney',
        state: 'New South Wales',
        postcode: 2000,
        country: 'Australia'
      }
    ],
    password: 'pass@789'
  },
  {
    _id: 'cli00000001',
    client_name: 'CloudGrid Solutions'
```

Figure 5.15: Client Data Import Confirmation

- Branches

```
nexart> db.branch.find()
[
  {
    _id: 'brs00000004',
    branch_name: 'Nexart California ',
    branch_address: [
      {
        'street number': 80,
        'street name': 'J Street',
        city: 'Sacramento',
        state: 'California',
        postcode: 94203,
        country: 'USA'
      }
    ],
    branch_contact: [
      {
        email_address: 'california@nexart.com',
        phone_number: '(213) 555-3456'
      }
    ],
    stock: [ { product_id: 'prd00000001', quantity: 100 } ],
    branch_password: 'bra@130'
  },
  {
    _id: 'brs00000003',
    branch_name: 'Nexart New York ',
    branch_address: [
      {
        'street number': 100,
        'street name': 'Broadway',
        city: 'New York',
        state: 'New York',
        postcode: 10001,
        country: 'USA'
      }
    ],
    branch_contact: [
      {
        email_address: 'newyork@nexart.com',
        phone_number: '(212) 555-1234'
      }
    ],
    stock: [ { product_id: 'prd00000001', quantity: 50 } ],
    branch_password: 'bra@130'
  },
  {
    _id: 'brs00000002',
    branch_name: 'Nexart London ',
    branch_address: [
      {
        'street number': 10,
        'street name': 'Downing Street',
        city: 'London',
        state: 'England',
        postcode: 'W1A 3AD',
        country: 'UK'
      }
    ],
    branch_contact: [
      {
        email_address: 'london@nexart.com',
        phone_number: '(20) 555-5678'
      }
    ],
    stock: [ { product_id: 'prd00000001', quantity: 25 } ],
    branch_password: 'bra@130'
  }
]
```

Figure 5.16: Branch Data Import Confirmation

- Products

```
nexart> db.product.find()
[
  {
    _id: 'prd00000004',
    product_type: 'switch',
    product_model: 'NEX-L25-GE-4',
    product_name: 'Nexart Layer 2 Ser4 Managable Switch',
    product_description: 'high performance Layer 2 fourth series switch',
    product_price: 240.5,
    product_image: ''
  },
  {
    _id: 'prd00000001',
    product_type: 'switch',
    product_model: 'NEX-L25-GE-1',
    product_name: 'Nexart Layer 2 Ser1 Managable Switch',
    product_description: 'high performance Layer 2 first series switch',
    product_price: 210.5,
    product_image: ''
  }
]
```

Figure 5.17: Product Data Import Confirmation

- Orders

```
nexart> db.order.find()
[
  {
    _id: 'ord00000003',
    order_date: '2/3/2024',
    total_amount: 691.5,
    client_id: 'cli00000003',
    branch_id: 'bra00000000',
    items: [ { product_id: 'prd00000003', quantity: 3, price: 230.5 } ],
    payment: [
      {
        payment_id: 'pay00000003',
        amount: 691.5,
        method: 'visa',
        payment_date: '2/3/2024'
      }
    ],
    shipping: [
      {
        shipping_id: 'shp00000003',
        shipping_date: '2/4/2024',
        tracking_number: 'tra00000003',
        carrier: 'UPS'
      }
    ]
  },
  {
    _id: 'ord00000001',

```

Figure 5.18: Order Data Import Confirmation

- Stock Requests

```
nexart> db.stock_request.find()
[
  {
    _id: 'str00000001',
    request_date: '2/1/2024',
    quantity_requested: 5,
    source_branch_id: 'prd00000100',
    destination_branch_id: 'prd00000001',
    product_id: 'prd00000001'
  },
  {
    _id: 'str00000005',
    request_date: '2/5/2024',
    quantity_requested: 2,
    source_branch_id: 'prd00000005',
    destination_branch_id: 'prd00000009',
    product_id: 'prd00000005'
  }
]
```

Figure 5.19: Stock Request Data Import Confirmation

Hence the records displayed at screen are limited to 20 per each iteration, they get the document statics of each collection by executing the commands given below.



```
nexart> db.product.countDocuments();
100
nexart> db.branch.countDocuments();
100
nexart> db.client.countDocuments();
100
nexart> db.order.countDocuments();
100
nexart> db.stock_request.countDocuments();
100
nexart> |
```

Figure 5.20: Document Statistics of Collections

There are differences between the query structures of relational databases and non-relational databases. With the growth of data, the database queries got complex, and some admins were bored of typing the same query repeatedly. In addition, it took some time to insert the query and human errors such as typos occurred as well. In addition, only highly technical professionals of Nexart understood SQL/NOSQL while for other users of Nexart, did not comprehend SQL/NOSQL. So, the developers of MSSQL produced a solution to address these issues.

## 5.4 Mongo Compass

The previous chapter brought up an issue regarding the usability of MSSQL CMD/Mongo Shell. As a remedy for this issue, MSSQL developed a graphical user interface along with a CMD. This resulted in fewer errors and query response times were improved. Even if the query is entered correctly, an admin can mistake enter a wrong value in the field and make the data inefficient. So, the admin is solely responsible for the validity of data. MongoDB Compass functions similarly as MSSQL GUI. As a result of the development of Mongo Compass, technical people who have an average knowledge about databases of Nexart do not need to worry about entering data into the database. The image given below depicts the viewing of the created Nexart database.

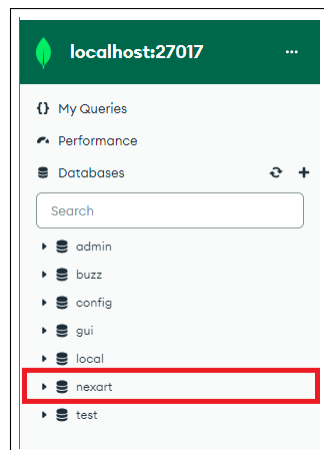


Figure 5.21: Nexart Database Creation Mongo Compass Confirmation

Earlier branch, client, product, order, and stock request collections were created in the Nexart database. The image below shows the created collections visually in Mongo Compass.

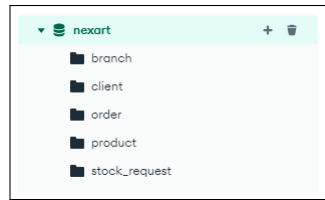


Figure 5.22: Created Collections Visual Confirmation

The data engineers imported data into the collections. All imported data can be verified visually by the list given below.

- Branch Collection

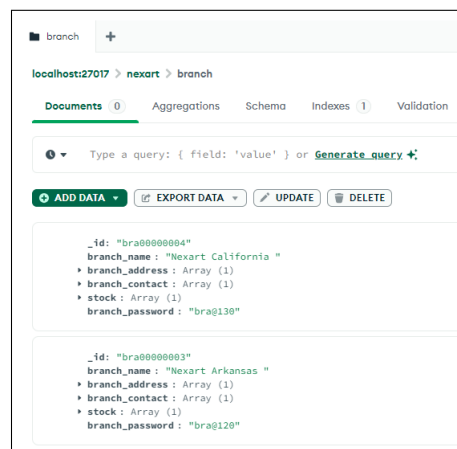


Figure 5.23: Branch Imported Data Visual Confirmation

- Client Collection

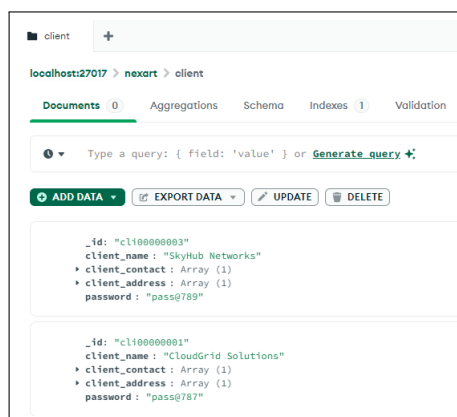


Figure 5.24: Client Imported Data Visual Confirmation

- Product Collection

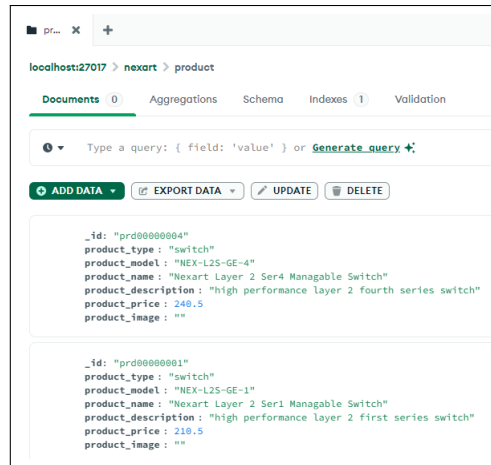


Figure 5.25: Product Imported Data Visual Confirmation

- Order Collection

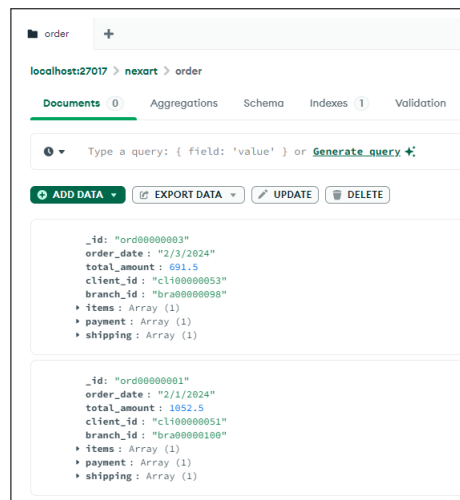


Figure 5.26: Order Imported Data Visual Confirmation

- Stock Request Collection

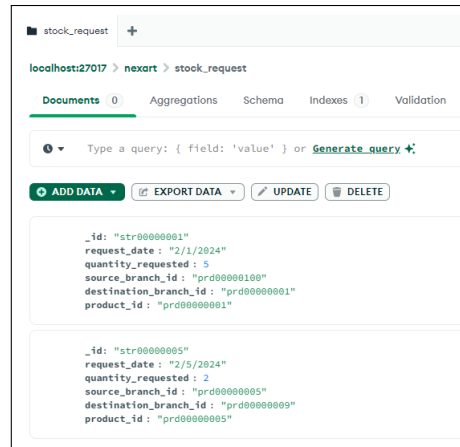


Figure 5.27: Stock Request Imported Data Visual Confirmation

People with average technical knowledge can enter data into the database via MongoDB compass's user-friendly graphical interface. Even though this graphical user interface had a limitation, it is addressed in the next chapter.

# Chapter 6

## Development of MongoDB Application

### 6.1 Technical Stack and Environment Setup

Sometimes graphics are not enough to convey a message. The message should be to understand the users. This applies to the web applications as well. An application contains two parts, and they are front-end and back-end. All the users deal with the front-end of the application while all the processes occur in the back-end of the application. Since the developers of Nexart are familiar with Node JS, Express JS, HTML, CSS, Bootstrap, JavaScript and MongoDB, the development team decided to use the same technical stack to develop the Nexart's web application.



Figure 6.1: Technical Stack

- M - MongoDB
- E - Express JS
- N - Node JS
- H - HTML
- C - CSS
- B - Bootstrap
- J - JavaScript

Before jumping into the front development, the development environment should be organized. Many developers of Nexart prefer Visual Studio Code IDE over other IDEs due to support offered by the vs community. So, all the developers agreed to go with Visual Studio Code.

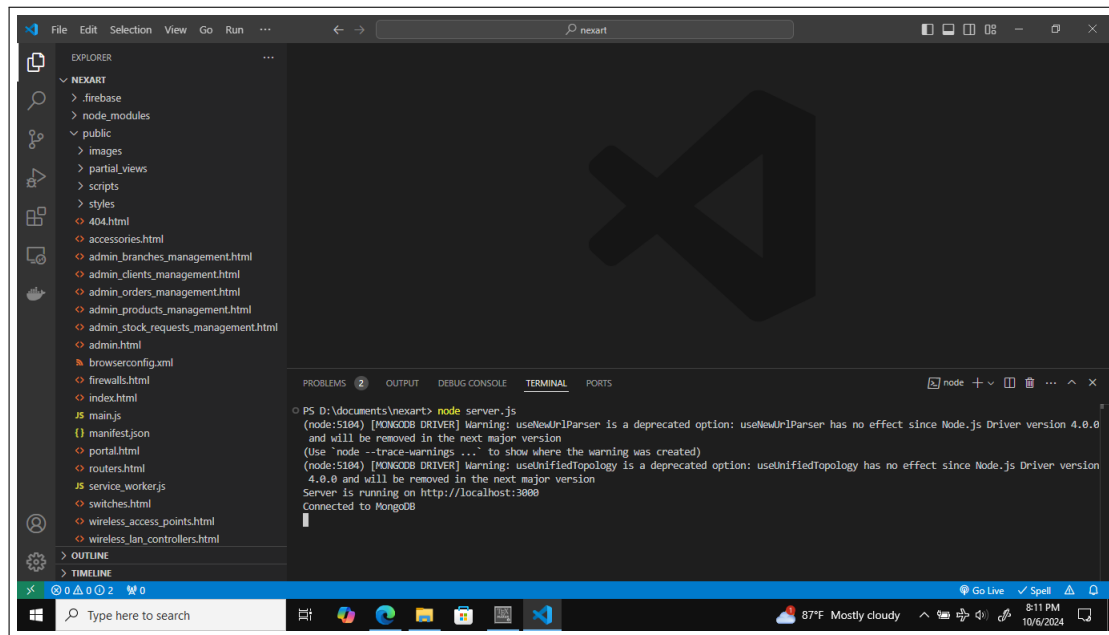


Figure 6.2: Integrated Development Environment

Housekeeping is a main aspect in application development. This makes sure that every is organized. In addition, it makes error troubleshooting easier. The Nexart web application's file structure is given below.

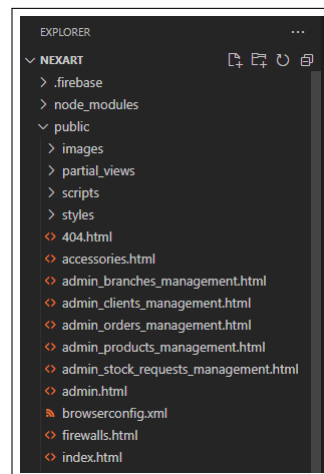


Figure 6.3: Project File Structure

As the technical stack and environment have already been organized, the front-end development of the application commenced from the next chapter onwards.

## 6.2 Front-end Development

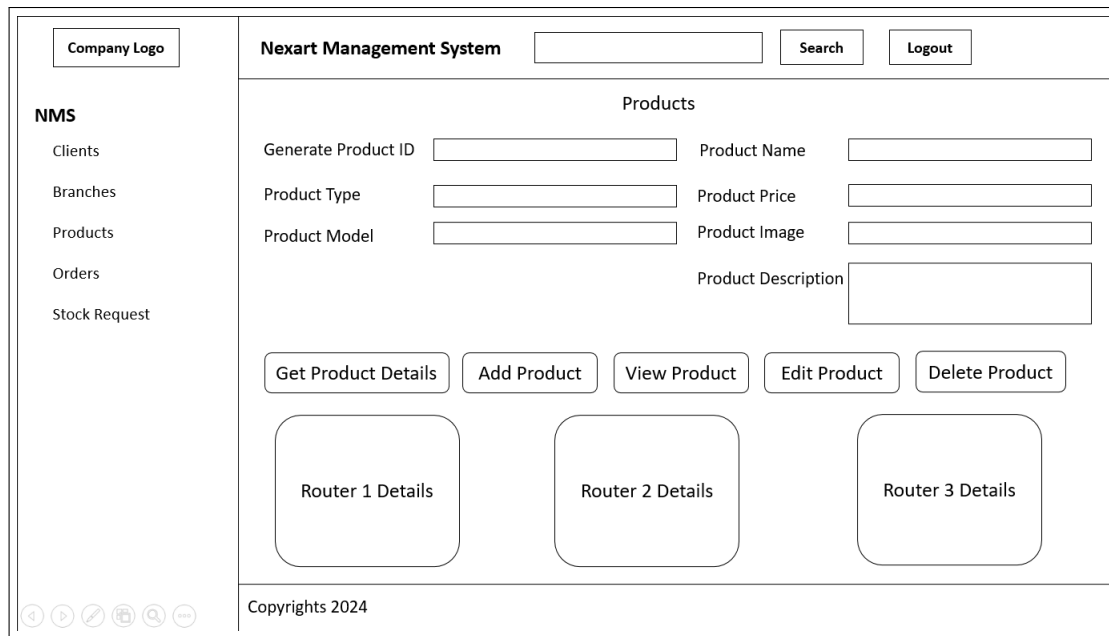
Before discussing the front-end of the application, let us get to know about the technical users in society. We can categorize technical people into 3 categories, and they are listed below.

- People who have less technical knowledge.
- People who have average technical knowledge.
- People who have high technical knowledge.

Let us explore facts about each technical group. People with less technical people are digitally literate people who use digital devices such as smart phones to complete some their daily/non-daily activities such as browsing the web, streaming music/videos online, replying to emails etc. Average technical personnels have more technical knowledge in comparison to people with less technical knowledge. They are also digitally literate who perform tasks such as word processing, spreadsheet processing, slide processing, using simple database management systems such as Microsoft Office Access etc. Highly technical people are digitally literate as averagely technical people, but they perform advanced tasks such as software development, writing SQL/NOSQL queries to manipulate data, writing scripts to automate tasks etc. So highly technical people are capable of handling command line interfaces and graphical user interfaces of database management systems while averagely technical users capable of only handling graphical user interfaces of databases. People with less technical capabilities are not capable of handling either command line interfaces or graphical user interfaces of databases.

This applies to the Nexart enterprise as well. Nexart web application can be accessed by the user categorizes that are mentioned in the above paragraph. As for instance circuit programmers, administrators and floor staff represents highly technical, averagely technical, and less technical people accordingly. In some cases, people who have less technical capabilities need to store data in the database to process a task such as monthly payroll. So, they need to interact with MongoDB database indirectly to process the payroll. This is where the front-end of the web application comes into play. This gives the privilege to the less technical people to interact with the database. Before developing the front-end wireframes were sketched out and they are demonstrated below.

- Admin Product Management Wireframe

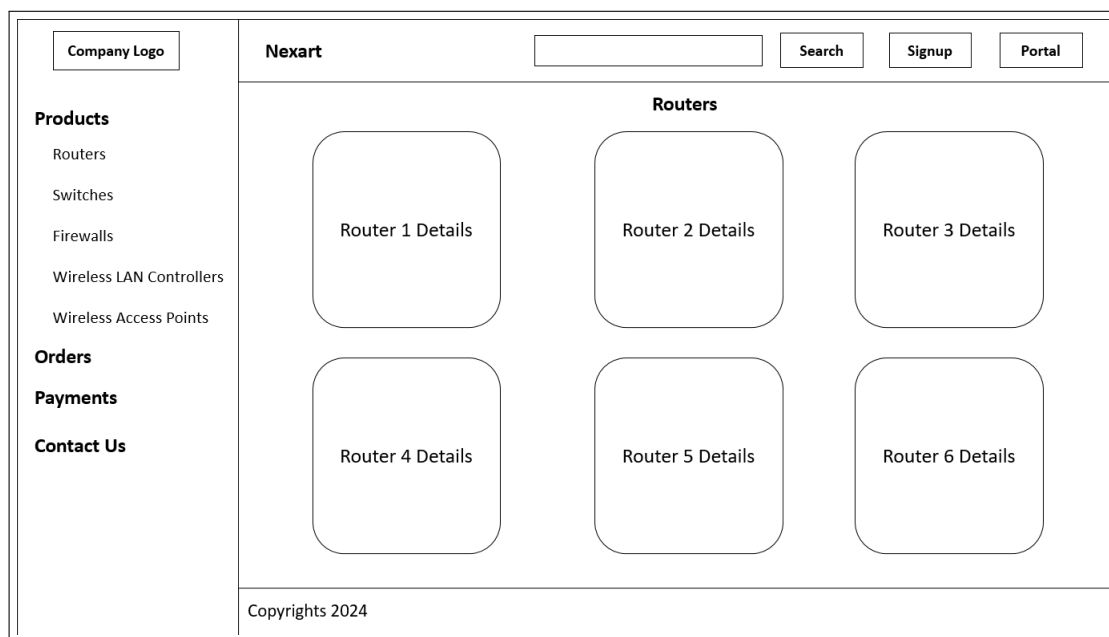


The wireframe for the Admin Product Management interface is divided into three main sections: a left sidebar, a top header, and a main content area.

- Left Sidebar:** Contains a "Company Logo" at the top, followed by the "NMS" title and a list of menu items: Clients, Branches, Products, Orders, and Stock Request.
- Top Header:** Features the "Nexart Management System" title, a search bar, and a "Logout" button.
- Main Content Area:**
  - Products Section:** Includes form fields for "Generate Product ID", "Product Name", "Product Type", "Product Price", "Product Model", "Product Image", and "Product Description". Below these are five buttons: "Get Product Details", "Add Product", "View Product", "Edit Product", and "Delete Product".
  - Router Details Section:** Displays three large rounded rectangular boxes labeled "Router 1 Details", "Router 2 Details", and "Router 3 Details".
- Footer:** Located at the bottom left, it contains a row of small circular icons and the text "Copyrights 2024".

Figure 6.4: Admin Product Management Wireframe

- Routers Wireframe



The wireframe for the Routers interface is structured similarly to the Admin Product Management wireframe, with a left sidebar, a top header, and a main content area.

- Left Sidebar:** Features a "Company Logo" at the top, followed by the "Products" title and a list of menu items: Routers, Switches, Firewalls, Wireless LAN Controllers, and Wireless Access Points. Below this are three additional sections: "Orders", "Payments", and "Contact Us".
- Top Header:** Displays the "Nexart" title, a search bar, and two buttons: "Signup" and "Portal".
- Main Content Area:**
  - Routers Section:** Shows a grid of six rounded rectangular boxes, each labeled "Router 1 Details" through "Router 6 Details".
- Footer:** Positioned at the bottom left, it includes the text "Copyrights 2024".

Figure 6.5: Routers Wireframe

- Switches Wireframe



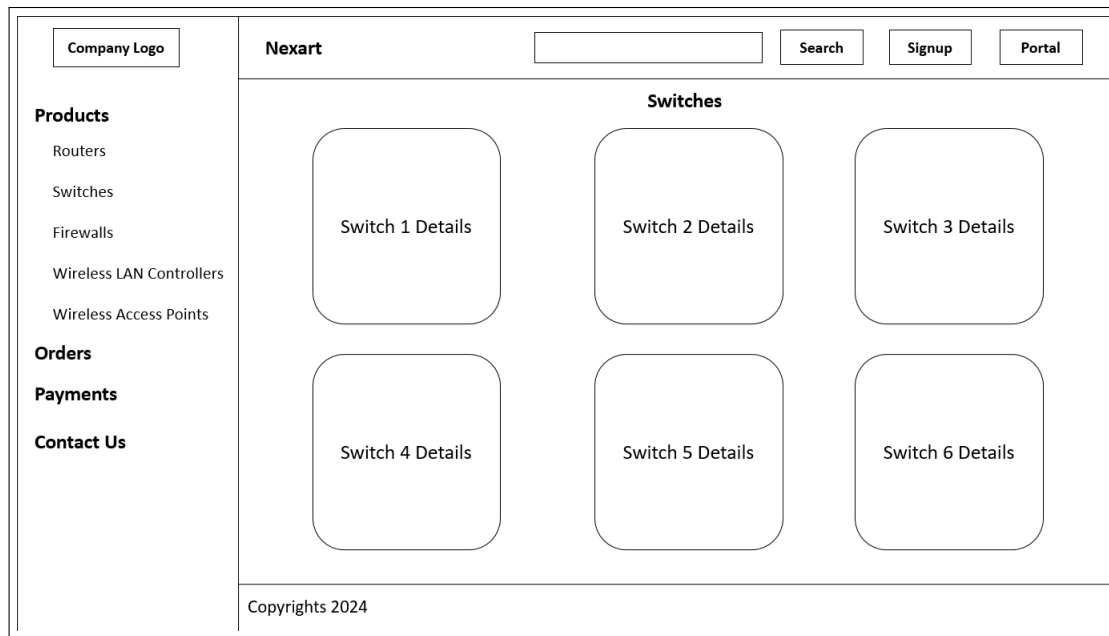


Figure 6.6: Switches Wireframe

- Firewalls Wireframe

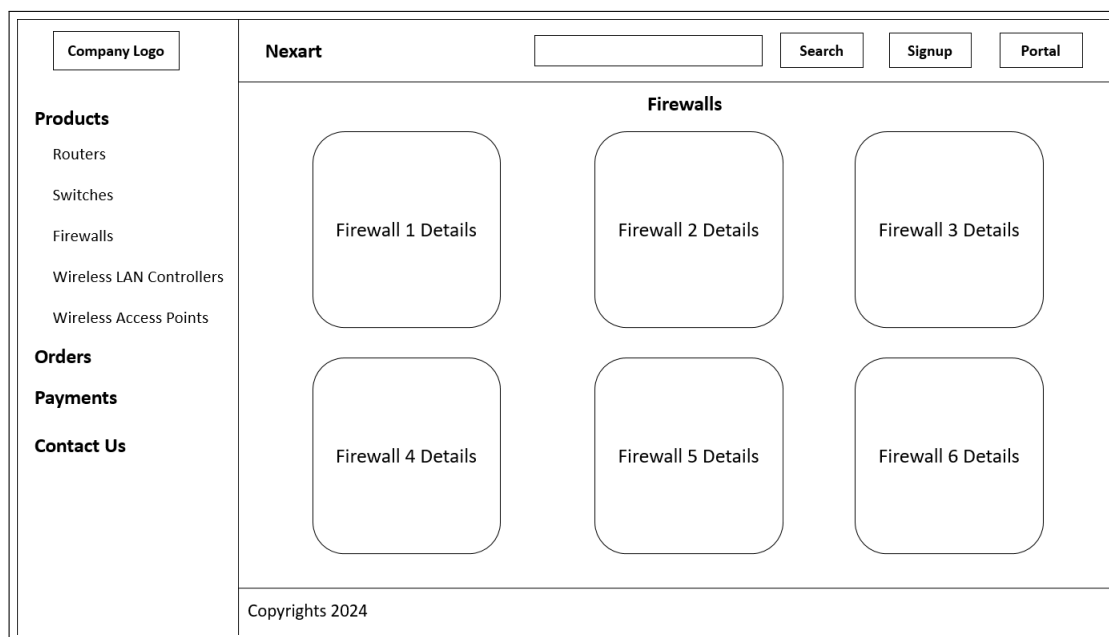


Figure 6.7: Firewalls Wireframe

In the next chapter these wireframes are brought back to life.

## 6.2.1 Graphical User Interface

In order to develop the graphical user interfaces, HTML and Bootstrap technologies are used, and the developed graphical user interface is illustrated below.

- Index Page

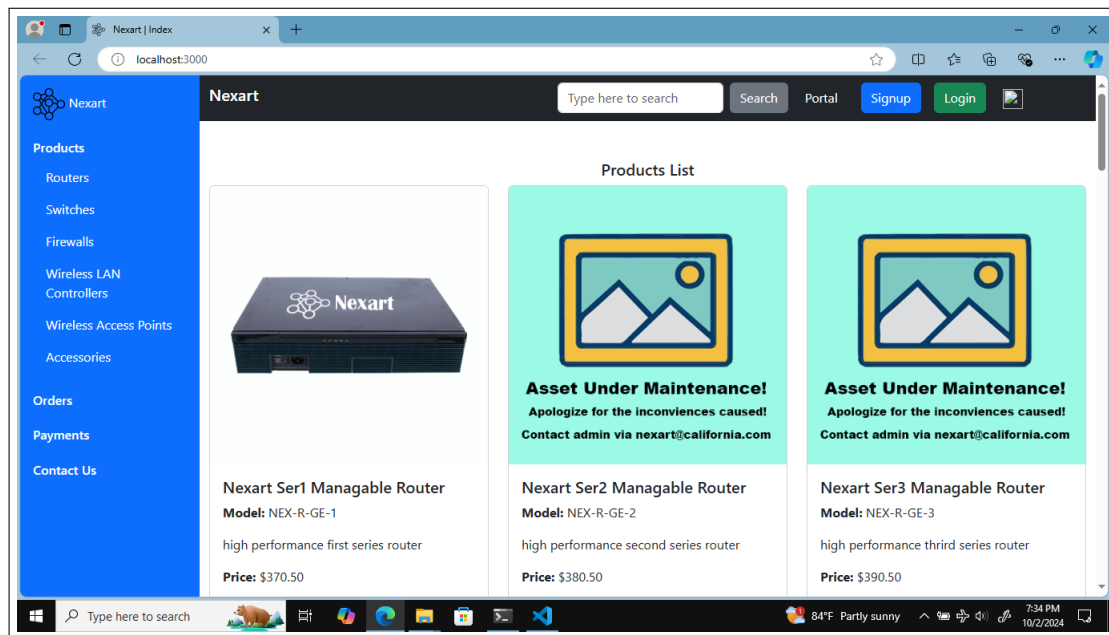


Figure 6.8: Index Page

- Product Management Page

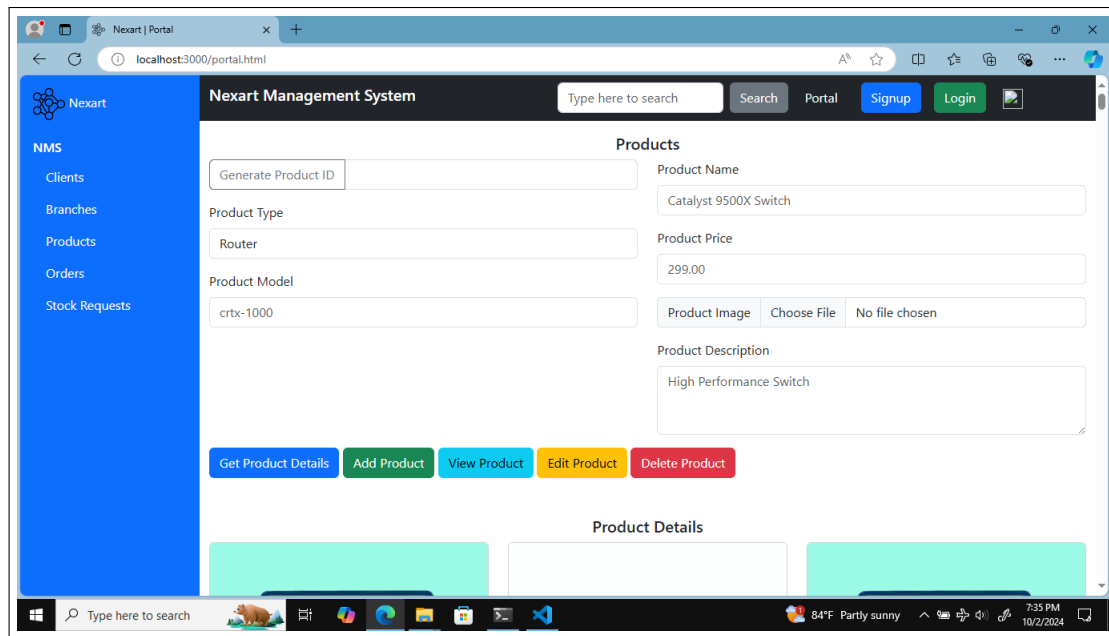


Figure 6.9: Product Management Page

- Routers Page

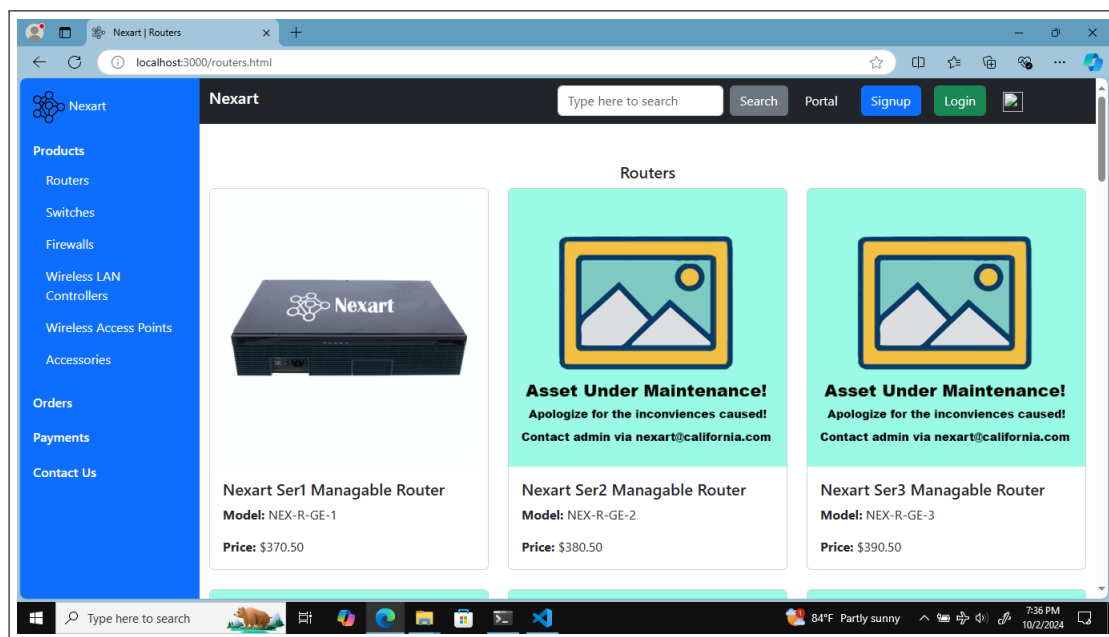


Figure 6.10: Routers Page

- Switches Page

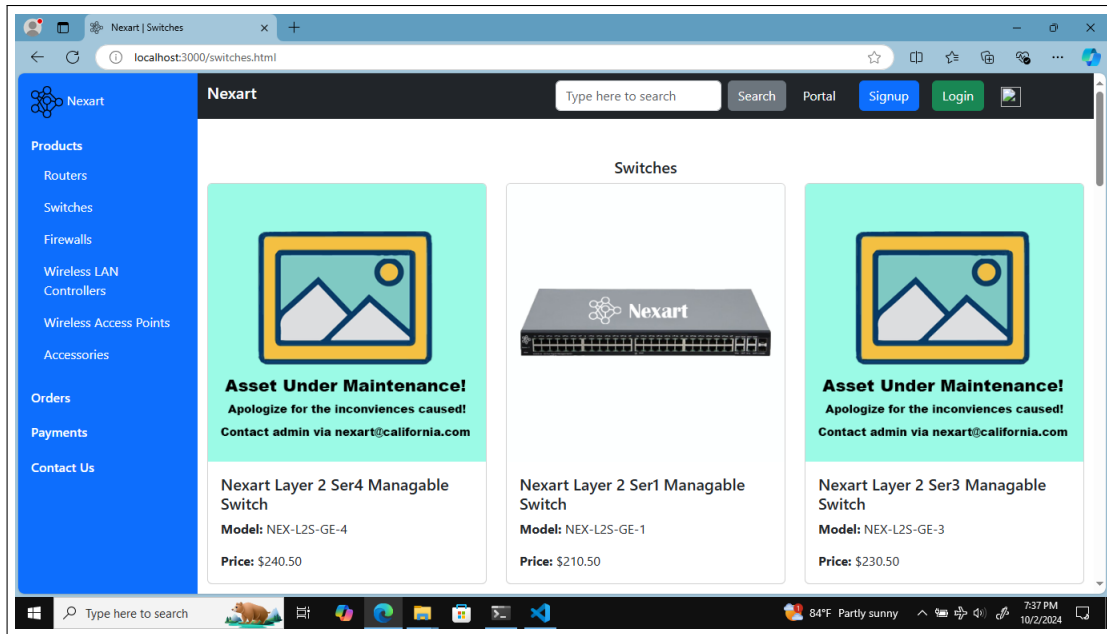


Figure 6.11: Switches Page

- Firewall Page

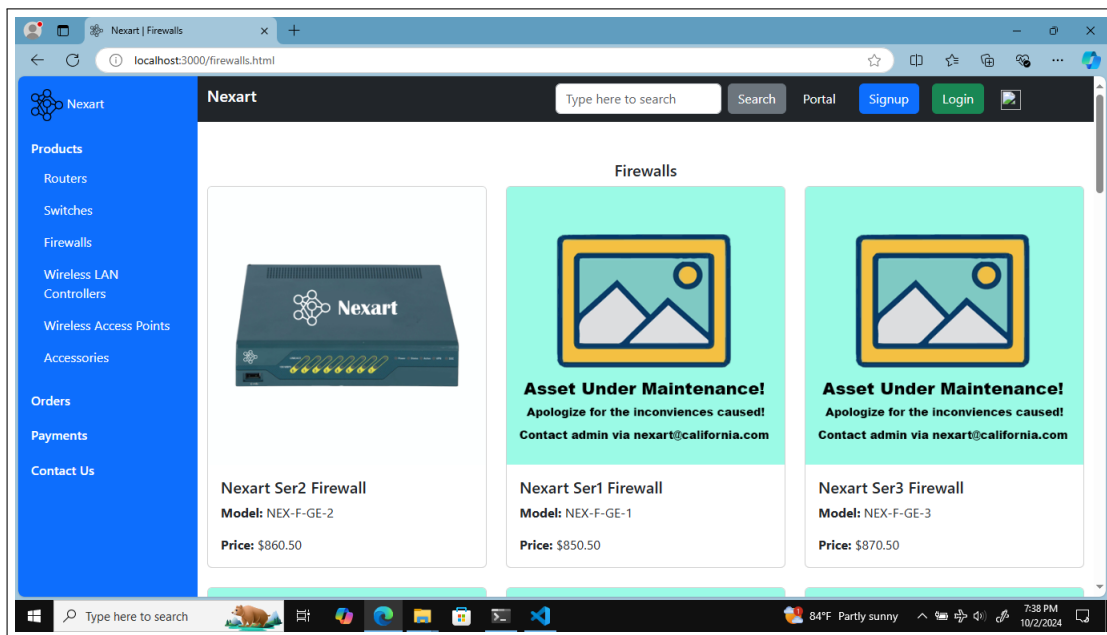


Figure 6.12: Firewalls Page

- Wireless LAN Controller Page

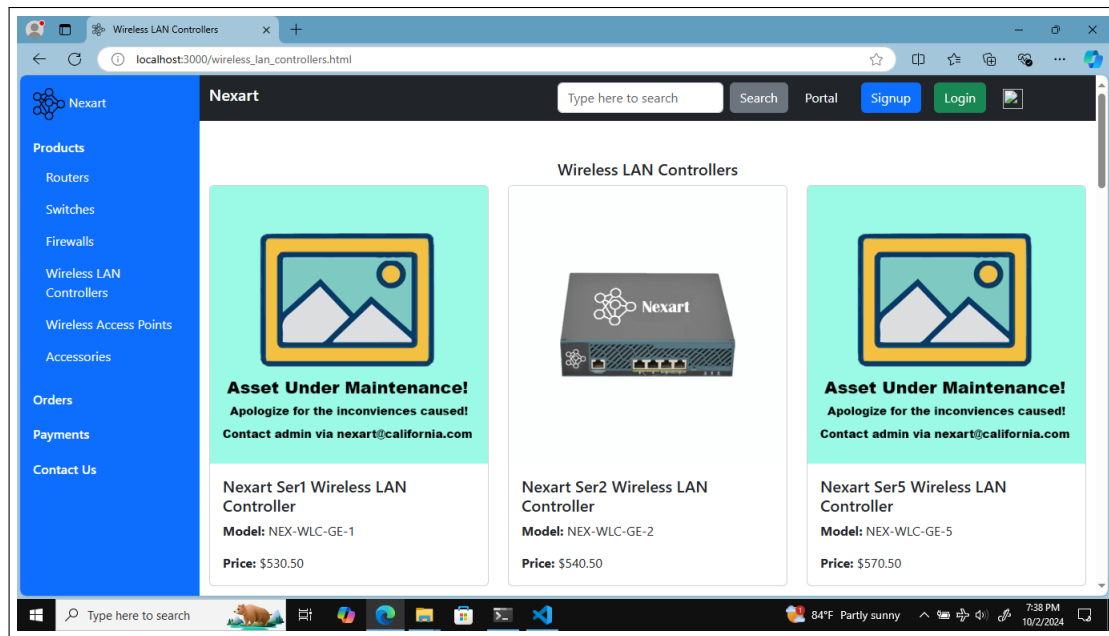


Figure 6.13: Wireless LAN Controllers Page

- Wireless Access Point Page

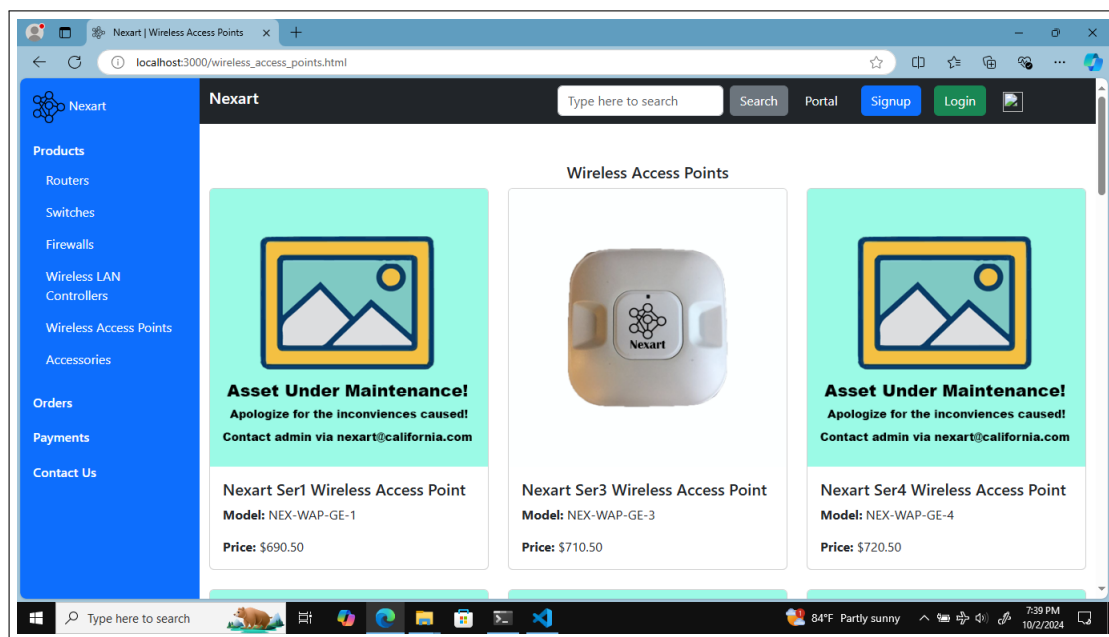


Figure 6.14: Wireless Access Points Page

- Accessories Page

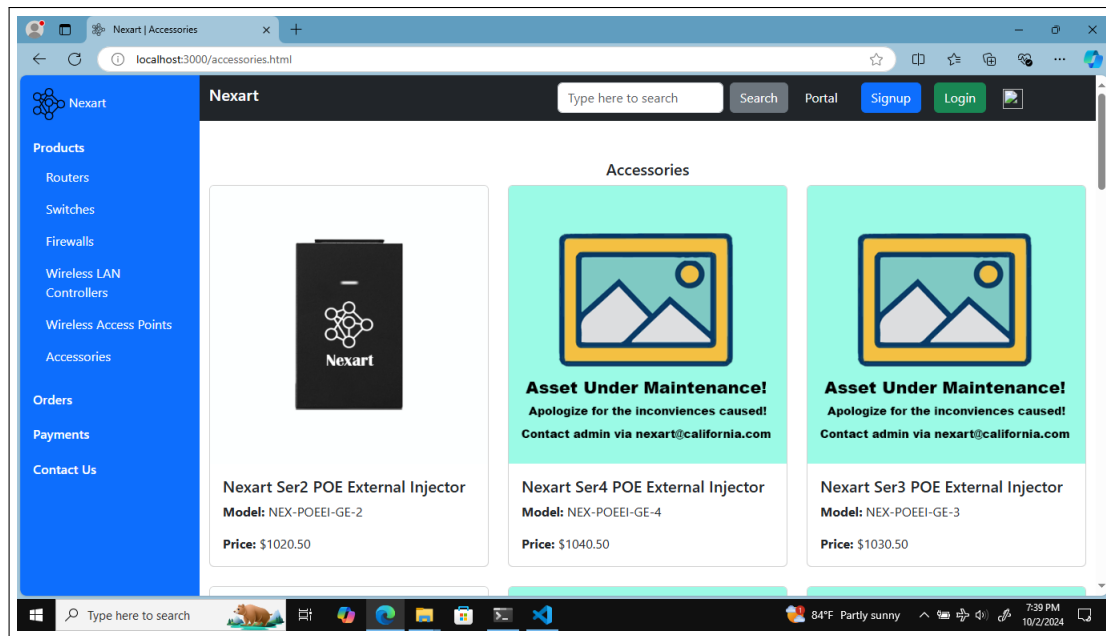


Figure 6.15: Accessories Page

Graphical User Interfaces does not complete the whole development phase of the application. Front-end of the application needs to communicate with the databases to perform all crud operations. Back-end of the application connects the front-end to the database. The next segment chapter discusses the development of back-end.

## 6.3 Back-end Development

The back-end of the application is developed using JavaScript. Scripts that are related to back-end are shown below.

- Server JS

```

JS server.js  X
JS server.js > ...
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const bodyParser = require('body-parser');
4  const multer = require('multer');
5  const path = require('path');
6
7  const app = express();
8  const PORT = process.env.PORT || 3000;
9
10 // MongoDB Connection
11 mongoose.connect('mongodb://localhost:27017/nexart', {
12   useNewUrlParser: true,
13   useUnifiedTopology: true
14 });
15
16 // Check if connection is successful
17 const db = mongoose.connection;
18 db.on('error', console.error.bind(console, 'MongoDB connection error:'));
19 db.once('open', () => {
20   console.log('Connected to MongoDB');
21 });
22

```

Figure 6.16: Server JS Backend

- Admin Product Management

```
<script>
document.addEventListener('DOMContentLoaded', function () {
    fetchProducts();

    document.getElementById('product_id_gen').addEventListener('click', function () {
        generateProductId();
    });

    document.getElementById('add_product').addEventListener('click', function () {
        addProduct();
    });

    document.getElementById('get_product').addEventListener('click', getProductDetails);
    document.getElementById('edit_product').addEventListener('click', editProduct);

    document.getElementById('view_product').addEventListener('click', function () {
        const productId = document.getElementById('product_id').value;
        if (!productId) {
            alert('Please enter a valid Product ID');
            return;
        }
        viewProduct(productId);
    });
});
```

Figure 6.17: Admin Product Management Backend

The server JavaScript file is the main script of this application. Functionality of the scripts of webpages relies upon the server JavaScript file. In this manner backend development comes into a whole. The functionality of the system is evaluated in the next chapter.

## 6.4 Testing

Without system testing and bug fixing, the whole development phase is not complete. The application cannot be hosted online unless the testing phase is completed. So, the development team requested the quality assurance engineer to evaluate the system thoroughly. For starters he tested the crud functionality of the products in the application by inserting values and triggering events such as clicks in admin product management page. The results of the crud functionality are given below.

Test Case ID	Description	Precondition	Test Procedure	Test Inputs	Expected Output	Actual Output	Results
TC0001	Admin adding a product into the database.	Admin should be logged in and system should be up and running.	Filling up the input fields and click on add product button	product id, product name	Product added successfully	Product added successfully	Product added successfully!
TC0002	Admin viewing a product.	Admin should be logged in and system should be up and running.	Insert product id in and click on view product button	product id	Product fetched!	Product fetched!	Product fetched successfully
TC0003	Admin editing a product.	Admin should be logged in and system should be up and running.	Fill up the necessary fields and click on edit product button	product id, product name	Product updated successfully!	Product updated successfully!	Product updated successfully!
TC0002	Admin deleting a product.	Admin should be logged in and system should be up and running.	Insert product id in and click on remove product button	product id	Product deleted successfully!	Product deleted successfully!	Product deleted successfully

Table 6.1: CRUD Test Case Results

In this manner the QA tested crud operations for other collections as well. In addition, he conducted unit and integration testing for all the scripts in the application. Since the testing phase has already been completed, the nexart web application is ready to be deployed online. Even though the system is fully developed, maintenance should be carried out to address the technical issues that are about to occur in the future. The effectiveness of the implemented data engineering pipeline is evaluated



in the next segment.

# Chapter 7

## Data Analytics

### 7.1 Analytical Questions

Values insights about the business can be drawn from the documents stored in the collections. According to the perspective of the business we can derive questions, and these questions are divided into several categories. The question categories and the relevant question for each category are listed below.

#### 1. Orders and Client Behavior

- What is the most ordered product item across all branches?
- Which product item have been ordered a lot in quantity wise?
- Which payment method is preferred by most of the clients?
- Which branch has received the most orders?
- What is the frequency of making repeat orders?

#### 2. Stock and Inventory Management

- Which branch receives the highest amount of stock requests?
- How long does it take branch to receive a requested product?
- What is the product item in stocks that has the highest quantity across all branches?
- Which product items are low in quantity wise and have made the highest requests?
- What is the relationship between stock requests and client orders?

#### 3. Branch and Operational Efficiency

- Which branch generates the highest sales?
- How long does it take for a product shipment to get delivered to the client?
- Which shipping carriers take the least amount of time to deliver products to clients?
- Which branch generates the least sales?
- What product prices are different branch wise and what is their impact on sales?

## 4. Client Satisfaction and Payment Analysis

- What is the correlation between the payment method and client order volume?
- What impact does the shipping cost and times of products have on client's address?
- Which branch gets the most complaints and returns?
- How do quicker deliveries affect client order amounts?

The questions mentioned in the above help us understand the status of the business. The answers to some queries are listed below.

- What is the most ordered product item across all branches?

```
test> use nexart
switched to db nexart
nexart> db.order.aggregate([
... { $unwind : "$items" },
... { $group: { _id: "$items.product_id", total_quantity: { $sum: "$items.quantity" } } },
... { $sort: { total_quantity: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'prd00000050', total_quantity: 5 } ]
nexart> |
```

Figure 7.1: Question 1.1 Answer

- Which product item have been ordered a lot in quantity wise?

```
nexart> db.order.aggregate([
... { $unwind: "$items" },
... { $group: { _id: "$items.product_id", total_quantity: { $sum: "$items.quantity" } } },
... { $sort: { total_quantity: -1 } }
... ])
[
  { _id: 'prd00000032', total_quantity: 2 },
  { _id: 'prd00000063', total_quantity: 5 },
  { _id: 'prd00000053', total_quantity: 4 },
  { _id: 'prd00000067', total_quantity: 2 },
  { _id: 'prd00000012', total_quantity: 2 },
  { _id: 'prd00000017', total_quantity: 4 },
  { _id: 'prd00000005', total_quantity: 1 },
  { _id: 'prd00000025', total_quantity: 1 },
  { _id: 'prd00000004', total_quantity: 2 },
  { _id: 'prd00000070', total_quantity: 2 },
  { _id: 'prd00000069', total_quantity: 1 },
  { _id: 'prd00000082', total_quantity: 4 },
  { _id: 'prd00000022', total_quantity: 4 },
  { _id: 'prd00000009', total_quantity: 4 },
  { _id: 'prd00000077', total_quantity: 3 },
  { _id: 'prd00000013', total_quantity: 3 },
  { _id: 'prd00000047', total_quantity: 5 },
  { _id: 'prd00000058', total_quantity: 1 },
  { _id: 'prd00000019', total_quantity: 3 },
  { _id: 'prd00000081', total_quantity: 3 }
]
Type "it" for more
nexart> |
```

Figure 7.2: Question 1.2 Answer

- Which payment method is preferred by most of the clients?

```
nexart> db.order.aggregate([
... { $unwind: "$payment" },
... { $group: { _id: "$payment.method", total_payments: { $sum: 1 } } },
... { $sort: { total_payments: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'paypal', total_payments: 51 } ]
nexart> |
```

Figure 7.3: Question 1.3 Answer

- Which branch has received the most orders?

```
nexart> db.order.aggregate([
... { $group: { _id: "$branch_id", total_orders: { $sum: 1 } } },
... { $sort: { total_orders: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'bra00000008', total_orders: 1 } ]
nexart> |
```

Figure 7.4: Question 1.4 Answer

- Which branch receives the highest amount of stock requests?

```
nexart> db.stock_request.aggregate([
... { $group: { _id: "destination_branch_id", total_requests: { $sum: 1 } } },
... { $sort: { total_requests: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'destination_branch_id', total_requests: 100 } ]
nexart> |
```

Figure 7.5: Question 2.1 Answer

- What is the product item in stocks that has the highest quantity across all branches?

```
nexart> db.branch.aggregate([
... { $unwind: "$stock" },
... { $group: { _id: "$stock.product_id", total_quantity: { $sum: "$stock.quantity" } } },
... { $sort: { total_quantity: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'prd00000001', total_quantity: 10000 } ]
nexart> |
```

Figure 7.6: Question 2.3 Answer

- Which branch generates the highest sales?

```
nexart> db.order.aggregate([
... { $unwind: "$payment" },
... { $group: { _id: "$payment.method", total_orders: { $sum: 1 }, total_amount: { $sum: "$total_amount" } } },
... { $sort: { total_orders: -1 } }
... ])
[
  { _id: 'paypal', total_orders: 51, total_amount: 107146 },
  { _id: 'visa', total_orders: 49, total_amount: 103634 }
]
nexart> |
```

Figure 7.7: Question 4.1 Answer

- What is the correlation between the payment method and client order volume?

```
nexart> db.order.aggregate([
... { $group: { _id: "branch_id", total_sales: { $sum: "$total_amount" } } },
... { $sort: { total_sales: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'branch_id', total_sales: 210780 } ]
nexart> |
```

Figure 7.8: Question 3.1 Answer

## 7.2 Hypotheses

- It is assumed that clients prefer wireless access points of the latest generations but according to the data, clients prefer wireless access points of older generations.

```
test> use nexart
switched to db nexart
nexart> db.order.aggregate([
... { $unwind : "$items" },
... { $group: { _id: "$items.product_id", total_quantity: { $sum: "$items.quantity" } } },
... { $sort: { total_quantity: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'prd00000050', total_quantity: 5 } ]
nexart> |
```

Figure 7.9: Hypotheses 1

- It is assumed that clients have ordered more switches in quantity wise but according to the data, clients ordered more routers considering the quantity.

```
nexart> db.order.aggregate([
... { $unwind: "$items" },
... { $group: { _id: "$items.product_id", total_quantity: { $sum: "$items.quantity" } } },
... { $sort: { total_quantity: -1 } }
... ])
[
  { _id: 'prd00000032', total_quantity: 2 },
  { _id: 'prd00000063', total_quantity: 5 },
  { _id: 'prd00000053', total_quantity: 4 },
  { _id: 'prd00000067', total_quantity: 2 },
  { _id: 'prd00000012', total_quantity: 2 },
  { _id: 'prd00000017', total_quantity: 4 },
  { _id: 'prd00000005', total_quantity: 1 },
  { _id: 'prd00000025', total_quantity: 1 },
  { _id: 'prd00000004', total_quantity: 2 },
  { _id: 'prd00000070', total_quantity: 2 },
  { _id: 'prd00000069', total_quantity: 1 },
  { _id: 'prd00000082', total_quantity: 4 },
  { _id: 'prd00000022', total_quantity: 4 },
  { _id: 'prd00000009', total_quantity: 4 },
  { _id: 'prd00000077', total_quantity: 3 },
  { _id: 'prd00000013', total_quantity: 3 },
  { _id: 'prd00000047', total_quantity: 5 },
  { _id: 'prd00000058', total_quantity: 1 },
  { _id: 'prd00000019', total_quantity: 3 },
  { _id: 'prd00000081', total_quantity: 3 }
]
Type "it" for more
nexart> |
```

Figure 7.10: Hypotheses 2

- It is assumed that clients prefer to pay using visa card but according to the data the clients prefer to pay using PayPal.

```
nexart> db.order.aggregate([
... { $unwind: "$payment" },
... { $group: { _id: "$payment.method", total_payments: { $sum: 1 } } },
... { $sort: { total_payments: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'paypal', total_payments: 51 } ]
nexart> |
```

Figure 7.11: Hypotheses 3

- It is assumed that most of the Nexart clientele is based in the state of California but according to the data most of the Nexart clientele is based in the state of Florida.

```
nexart> db.order.aggregate([
... { $group: { _id: "$branch_id", total_orders: { $sum: 1 } } },
... { $sort: { total_orders: -1 } },
... { $limit: 1 }
... ])
[ { _id: 'bra00000000', total_orders: 1 } ]
nexart> |
```

Figure 7.12: Hypotheses 4

These hypothesis helps Nexart in making enhancements, trying new strategies and making business decisions. The next chapter sums up all the details in the above chapters.

# **Chapter 8**

## **Conclusion**

The hypothesis discussed in the above section are useful when making out important business strategies. As overall the Nexart web application functions as intended according to the testing results. Scaling businesses generates tons of data. In order to handle big data, NOSQL databases are required. This gives the privilege of storing large volumes of data without worrying about the schema. Migrating data from one database to another database requires time and effort. Data pipelines addresses these issues by preparing an effective plan without disrupting the daily transactions of Nexart. Bottom line, data pipelines are vital in solving problems that are associated with data.

# Bibliography

Aggregation operations. <https://www.mongodb.com/docs/manual/aggregation/>. Accessed: 2024-10-7.

Extract data from PDF to excel, CSV, JSON, read & generate PDF, barcodes. <https://bytescout.com>, July 2018. Accessed: 2024-10-7.

Data engineering: Data warehouse, data pipeline and data eng. <https://www.altexsoft.com/blog/what-is-data-engineering-explaining-data-pipeline-data-warehouse-and-data-engineering/> March 2023. Accessed: 2024-10-7.

Eelco Plugge, Peter Membrey, and Tim Hawkins. *The definitive guide to MongoDB*. APress, Berlin, Germany, 1 edition, September 2010.