



Dr. Vishwanath Karad

**MIT WORLD PEACE
UNIVERSITY** | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

BDT Mini Project Report

Topic

Notepad Analytics

Guided By

Prof. Sagar Apune

Group Members

PC14 Jay Mehta

PC20 Yuvraj Singh Lamba

PC22 Keshav Jha

PC25 Vansh Gurnani

Table of Contents

Sr. No.	Topic	Page No.
1	Project Title	1
2	Team Members	1
3	Overview of BDT used in our project	1-3
4	Workflow diagram	4
5	Future Scope and Conclusion	5
6	References	6
7	Screenshots	7-8

1) Project Title

Notepad Analytics using D3.js, PySpark & MongoDB

2) Team Members

PC14 Jay Mehta

PC20 Yuvraj Singh Lamba

PC22 Keshav Jha

PC25 Vansh Gurnani

3) Overview of various big data technologies used in the mini project

a. MongoDB:-

MongoDB is an open source NoSQL database management program. NoSQL (Not only SQL) is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.

MongoDB is used for high-volume data storage, helping organizations store large amounts of data while still performing rapidly. Organizations also use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.

Why is MongoDB used?

Storage: MongoDB can store large structured and unstructured data volumes and is scalable vertically and horizontally. Indexes are used to improve search performance. Searches are also done by field, range and expression queries.

Data integration: This integrates data for applications, including for hybrid and multi-cloud applications.

Complex data structures descriptions: Document databases enable the embedding of documents to describe nested structures (a structure within a structure) and can tolerate variations in data.

Load balancing: MongoDB can be used to run over multiple servers.

The MongoDB document stores information related to a note entry in the project. Here's what each field within the document is storing:

1. "_id": This field holds a unique identifier for the note entry. It serves as a primary key for this particular document.
2. "content": This field stores the actual content of the note.
3. "category": The "category" field allows users to categorize their notes.
4. "owner": This field stores the unique identifier of the user who created the note. It helps associate the note with its respective user.
5. "bookmarked": The "bookmarked" field stores a boolean value (true or false) to indicate whether the note has been bookmarked.
6. "timestamp": This field records the current date and time when the note was created.

These fields collectively store information about the note's content, category, ownership, bookmark status, and the time it was created. This structured data enables users to effectively manage and access their notes within the project.

b. D3.js:-

In our "Notepad" website, we have harnessed the full potential of D3.js, a versatile JavaScript library, to provide users with an engaging and informative experience. D3.js allowed us to create interactive data visualizations that play a central role in helping users understand their activities on the platform. For instance, we developed dynamic charts that illustrate the relationship between the amount of time spent on the website and word count, enabling users to track their productivity over time. These visualizations are not only aesthetically appealing but also functional, making it easier for users to make data-driven decisions about their usage patterns.

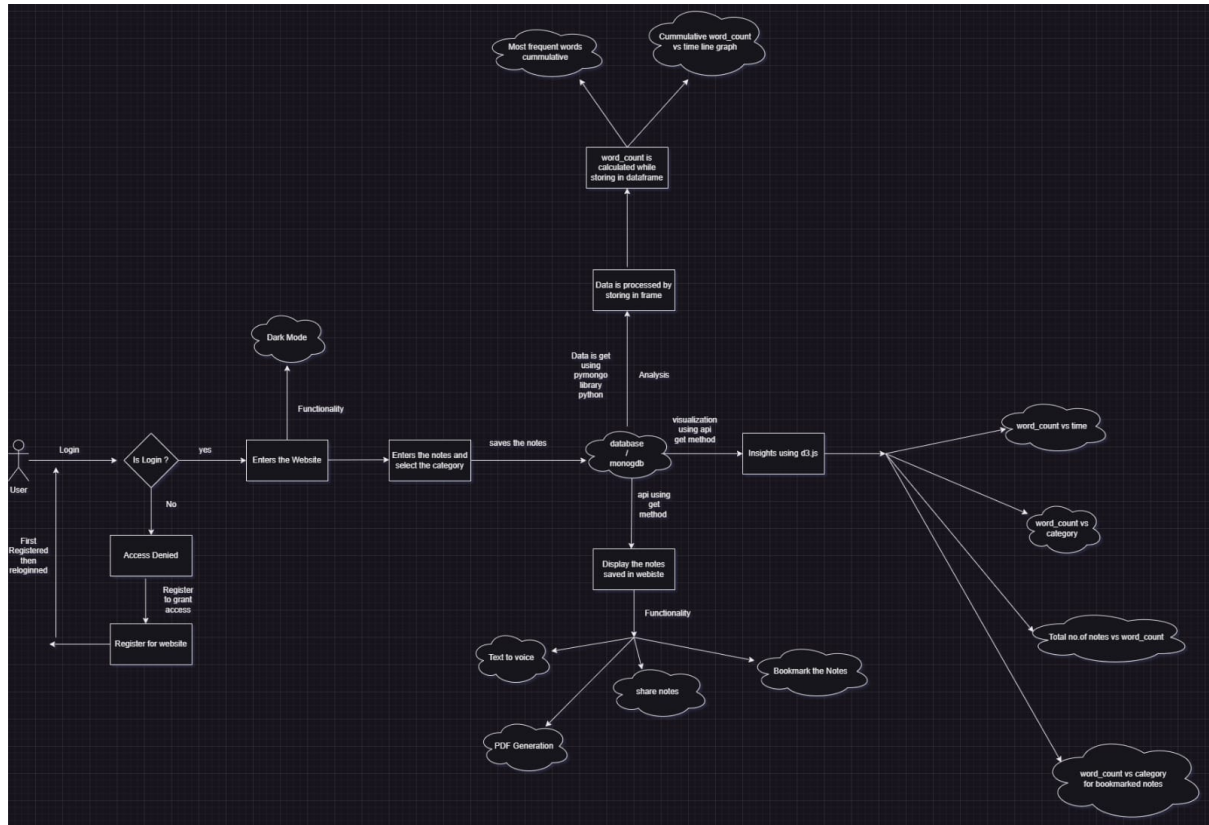
Furthermore, D3.js has been instrumental in crafting categorization graphs and bar charts. Users can categorize their notes, and D3.js empowers these categories to come to life through visually appealing representations. Whether it's tracking bookmarked notes or analyzing usage patterns, D3.js enhances the user experience by providing a graphical, interactive means of understanding their data. The combination of D3.js and other data analysis tools makes our project a comprehensive platform for users to gain insights into their online habits and maximize their productivity.

c. PySpark:-

In the context of our project, PySpark is employed for in-depth data analysis, particularly for word frequency analysis and creating word count vs. time graphs using Plotly. PySpark is a powerful data processing framework that allows you to perform complex operations on the data stored in MongoDB.

PySpark can be utilized to analyze the content field of your MongoDB documents, extracting insights such as word frequencies. This enables users to gain a better understanding of the most common words in their notes and assess their content. Additionally, PySpark can help generate word count vs. time graphs, providing users with a visual representation of how their note-taking habits evolve over time. These capabilities make PySpark an essential tool for deriving meaningful insights from the data stored in MongoDB and enhancing the analytical features of our project.

4) Workflow diagram



5) (i) Future Scope:-

Collaborative Editing: Implement real-time collaboration features that allow multiple users to edit the same document simultaneously. This could be useful for team projects or brainstorming sessions.

Voice Recognition and Dictation: Integrate voice recognition technology to allow users to dictate their notes instead of typing them manually. This can be especially useful for users on the go or those with accessibility needs.

Privacy and Security: Incorporating robust encryption and privacy features to ensure the safety of sensitive information stored in the notepad. For example, using a pin to access notes regarding bank details.

Customizable Templates: Provide users with the ability to create and save custom templates for different types of notes (e.g., meeting minutes, to-do lists, project plans).

Machine Learning for Predictive Text: Implement a predictive text feature that suggests the next word or phrase based on the user's writing patterns and context.

(ii) Conclusion:-

In conclusion, "Notepad" is a robust and versatile project that combines the power of technologies like D3.js, PySpark, and MongoDB to offer users a comprehensive solution for note-taking, data analysis, and time management. With its current feature set and the potential for future expansion, our project provides a user-friendly and data-driven approach to note-taking, fostering productivity and enhancing the overall note-taking experience.

6) References:-

<https://d3js.org/>

<https://www.mongodb.com/>

<https://spark.apache.org/docs/latest/api/python/index.html>

<https://colab.google/>

<https://plotly.com/>


Screenshots of the Project:-

Login


Login

Register

NotePad Home About Insight Summary News ▾ Notes ▾

Enable Dark Mode  Welcome, Vansh Gurnani Logout

Fetch all Notes

Welcome to NotePad 


Enter the text

Category:
Uncategorized ▾

Clear Save Note

Character Count: 0

NotePad Home About Insight Summary News ▾ Notes ▾

Enable Dark Mode  Welcome, Vansh Gurnani Logout

About Us

Vansh Gurnani ▾

Yuvraj Singh Lamba ▾

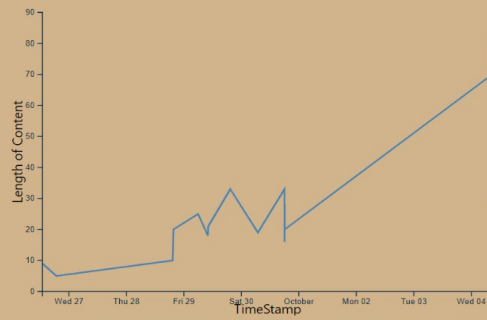
Jay Mehta ▾

Keshav Jha ▾



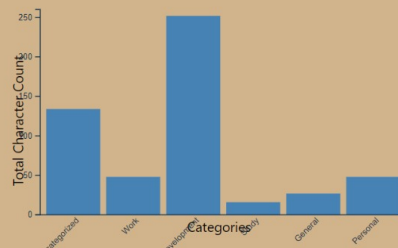
Line Chart

Length & Time Graph



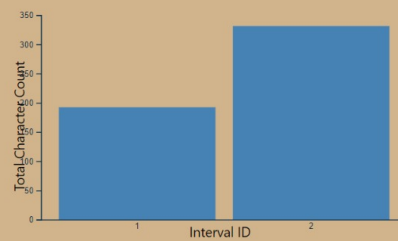
Bar Chart

Category Graph



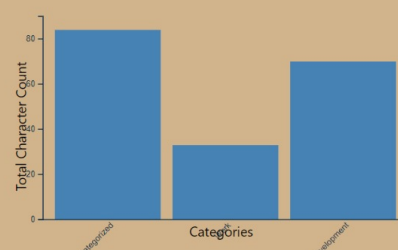
Bar Graph

Total Character Count for Each 10-Note Interval



Bookmark Graph

Category Graph for Bookmarked Notes



```
!pip install findspark
!pip install pyspark
!pip install pymongo
```

```
Requirement already satisfied: findspark in /usr/local/lib/python3.10/dist-packages (2.0.1)
Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-packages (3.5.0)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Requirement already satisfied: pymongo in /usr/local/lib/python3.10/dist-packages (4.6.0)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from pymongo) (2.4.2)
```

```
import pandas as pd
import pymongo
```

```
def fetch_data_from_mongo():
    # Connect to your MongoDB server
    client = pymongo.MongoClient("mongodb+srv://gurnanivansh57:iz64rqtBBQss8iQ7@cluster101.nuwewcc.mongodb.net/tododb?retryWrites=true&w=

    # Specify the database and collection you want to query
    db = client["Notepad"]
    collection = db["notes"]

    try:
        # Query the collection to fetch data
        data = list(collection.find())

        # Check if any data was retrieved
        if data:
            return data
        else:
            print("No data found in MongoDB.")
            return None

    except Exception as e:
        print(f"An error occurred: {e}")
        return None
```

```
def save_to_csv(data, file_name="output.csv"):
    if data:
        try:
            # Convert the data to a DataFrame
            df = pd.DataFrame(data)

            # Save the DataFrame to a CSV file
            df.to_csv(file_name, index=False)
            print(f"DataFrame saved to {file_name}")
        except Exception as e:
            print(f"Error saving to CSV: {e}")
    else:
        print("No data to save.")

if __name__ == "__main__":
    # Fetch data from MongoDB
    fetched_data = fetch_data_from_mongo()

    # Save data to CSV
    save_to_csv(fetched_data, "output.csv")
```

 DataFrame saved to output.csv

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, length, concat_ws, collect_list
from pyspark.sql.types import TimestampType
from pyspark.sql.functions import to_timestamp

# Initialize a Spark session
spark = SparkSession.builder.appName("TotalCharacterCount").getOrCreate()

# Load data from the CSV file
df = spark.read.csv("output.csv", header=True, inferSchema=True)

# Define the expected timestamp format
expected_timestamp_format = "yyyy-MM-dd HH:mm:ss.SSS"
```

```
# Filter out rows with invalid timestamps
df = df.withColumn("timestamp", to_timestamp(col("timestamp"), expected_timestamp_format))
df = df.na.drop(subset=["timestamp"])

# Calculate the total character count for each content
df = df.withColumn("total_character_count", length(col("content")))

# Group data by 'timestamp' and concatenate 'content' values
grouped_df = df.groupBy("timestamp").agg(concat_ws(" ", collect_list("content")).alias("combined_content"))

# Calculate the total character count for the combined content
grouped_df = grouped_df.withColumn("total_character_count", length(col("combined_content")))

# Show the result
grouped_df.show(truncate=False)
```

```
+-----+-----+
|timestamp|combined_content|
+-----+-----+
|2023-09-28 19:38:16.288|hello bookmark notes|
|2023-09-28 19:18:55.792|hello test|
|2023-09-30 18:02:47.187|hello study test|
|2023-09-30 18:05:00.912|hello personal notes testing|
|2023-09-27 10:12:48.233|Hello yuvraj|
|2023-09-30 18:01:58.539|hello test study notes|
|2023-09-26 18:43:47.312|hello|
|2023-09-29 19:36:44.374|hello test 2|
|2023-09-30 17:59:33.357|hello development testing notes 1|
|2023-09-29 05:55:08.614|hello bookmark testing 12|
|2023-09-27 10:13:05.916|Hello world|
|2023-10-04 08:49:57.47 |Hadoop is a framework work to handle data|
|2023-10-06 15:24:43.545|000 and the CountryCode is 'USA'. It selects all columns from the matching rows in the CITY table.|
|2023-09-29 10:08:28.603|hello bookmark test 3|
|2023-10-23 17:48:21.181|"In a small coastal town named Blue Harbor, residents enjoyed the simple pleasures of life. The weather was|
|2023-10-23 17:49:48.879|In this enchanted forest, the trees held secrets of ages past, with oak, pine, and maple standing tall and|
|2023-10-06 15:22:51.076| which can require more computational resources. Engineers and designers must strike a balance between achi|
|2023-09-29 19:22:38.758|hello work testing 10000000000000|
|2023-10-06 15:23:25.761|A linear-phase filter is a type of filter, typically a digital filter, that exhibits a linear relationship
```

```
|2023-09-30 18:05:35.629|hello personal notes
```

```
+-----+
```

```
only showing top 20 rows
```

```
pip install plotly
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15.0)
```

```
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (8.2.3)
```

```
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly) (23.2)
```

```
import pandas as pd
```

```
import plotly.express as px
```

```
from pyspark.sql.functions import to_date, sum
```

```
# Group data by 'timestamp' and calculate the total character count for each timestamp
```

```
time_vs_character_count_df = grouped_df.groupBy("timestamp").agg(sum("total_character_count").alias("total_character_count"))
```

```
# Convert the timestamp to a date to simplify the x-axis in the plot
```

```
time_vs_character_count_df = time_vs_character_count_df.withColumn("date", to_date("timestamp"))
```

```
# Convert the PySpark DataFrame to a Pandas DataFrame
```

```
pandas_df = time_vs_character_count_df.select("date", "total_character_count").toPandas()
```

```
# Sort DataFrame by date
```

```
pandas_df = pandas_df.sort_values('date')
```

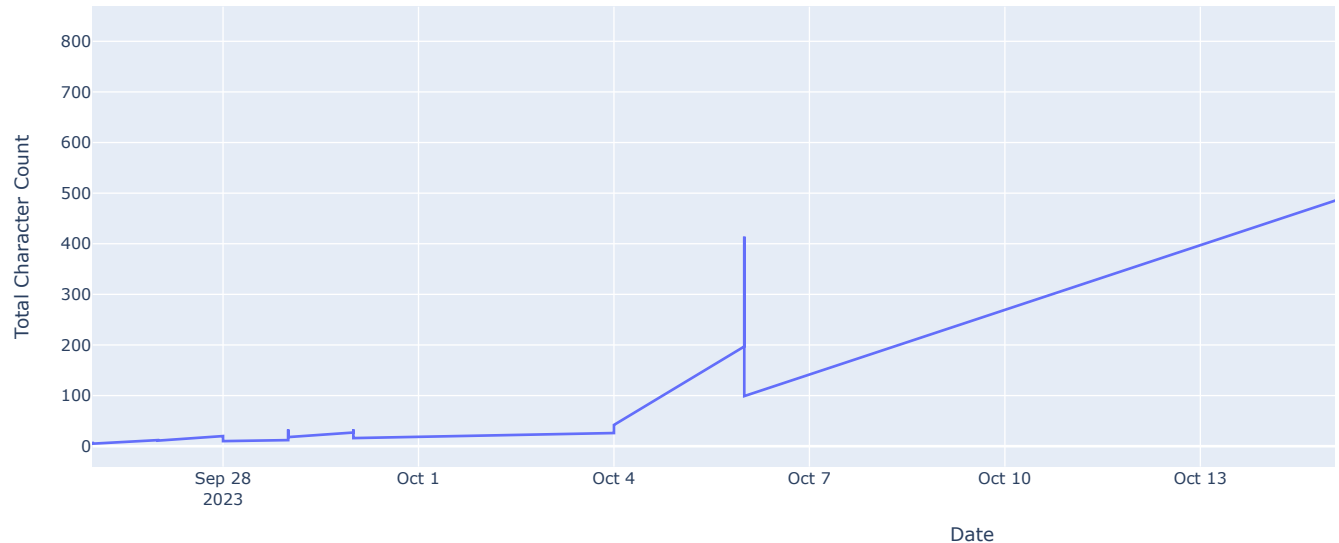
```
# Create an interactive line graph using Plotly
```

```
fig = px.line(pandas_df, x='date', y='total_character_count', labels={'date': 'Date', 'total_character_count': 'Total Character Count'},  
              title='Time vs. Total Character Count')
```

```
# Show the interactive plot
```

```
fig.show()
```

Time vs. Total Character Count



```
from pyspark.sql.functions import lower, explode, split
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
import matplotlib.pyplot as plt

# Initialize a Spark session
spark = SparkSession.builder.appName("WordFrequencyAnalysis").getOrCreate()

# Load data from the CSV file
```

```
# Load data from the CSV file
df = spark.read.csv("output.csv", header=True, inferSchema=True)

# Tokenize and preprocess the text data using PySpark DataFrame operations
df = df.withColumn("content", lower(col("content")))
df = df.select(explode(split(df.content, r'\s+')).alias("word"))
df = df.select(col("word")).filter(col("word").rlike(r'^[a-z]+$'))

# Calculate word frequencies
word_frequencies = df.groupBy("word").count()

# Get the most common words and their frequencies
common_words = word_frequencies.orderBy("count", ascending=False).limit(10)

# # Show the top 10 most common words and their frequencies
# common_words.show()

# Collect the data to be plotted
word_data = common_words.collect()

# Extract words and frequencies from the collected data
words = [row.word for row in word_data]
frequencies = [row['count'] for row in word_data]

# Create a bar chart to visualize word frequencies
plt.bar(words, frequencies)
plt.xlabel("Words")
plt.ylabel("Frequency")
plt.title("Top 10 Most Frequent Words")
plt.xticks(rotation=45)
plt.show()

# Stop the Spark session
spark.stop()
```