BIG DATA PROJECT REPORT

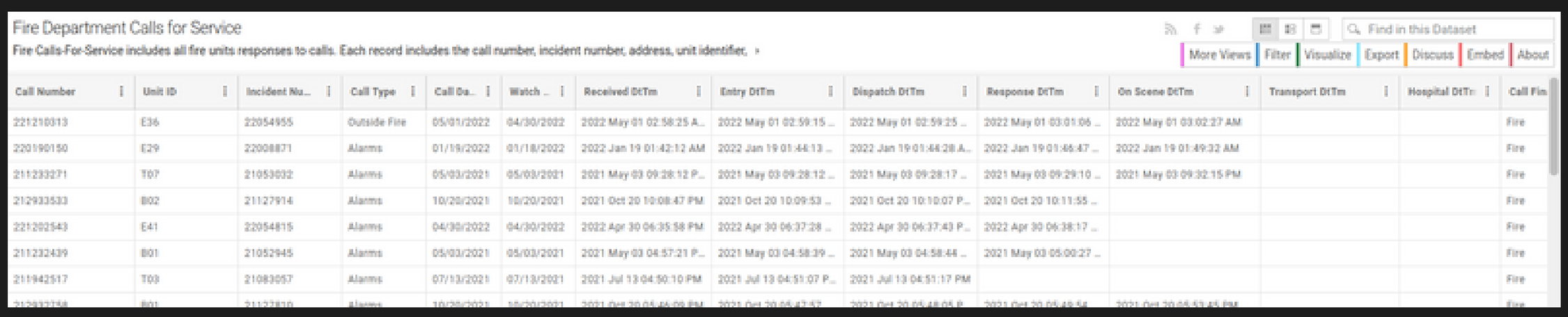
911 FIRE EMERGENCY CALLS

Hadil Helali . Raoua Trimech . Soulaima Kahla
GL 4/1

Datasets

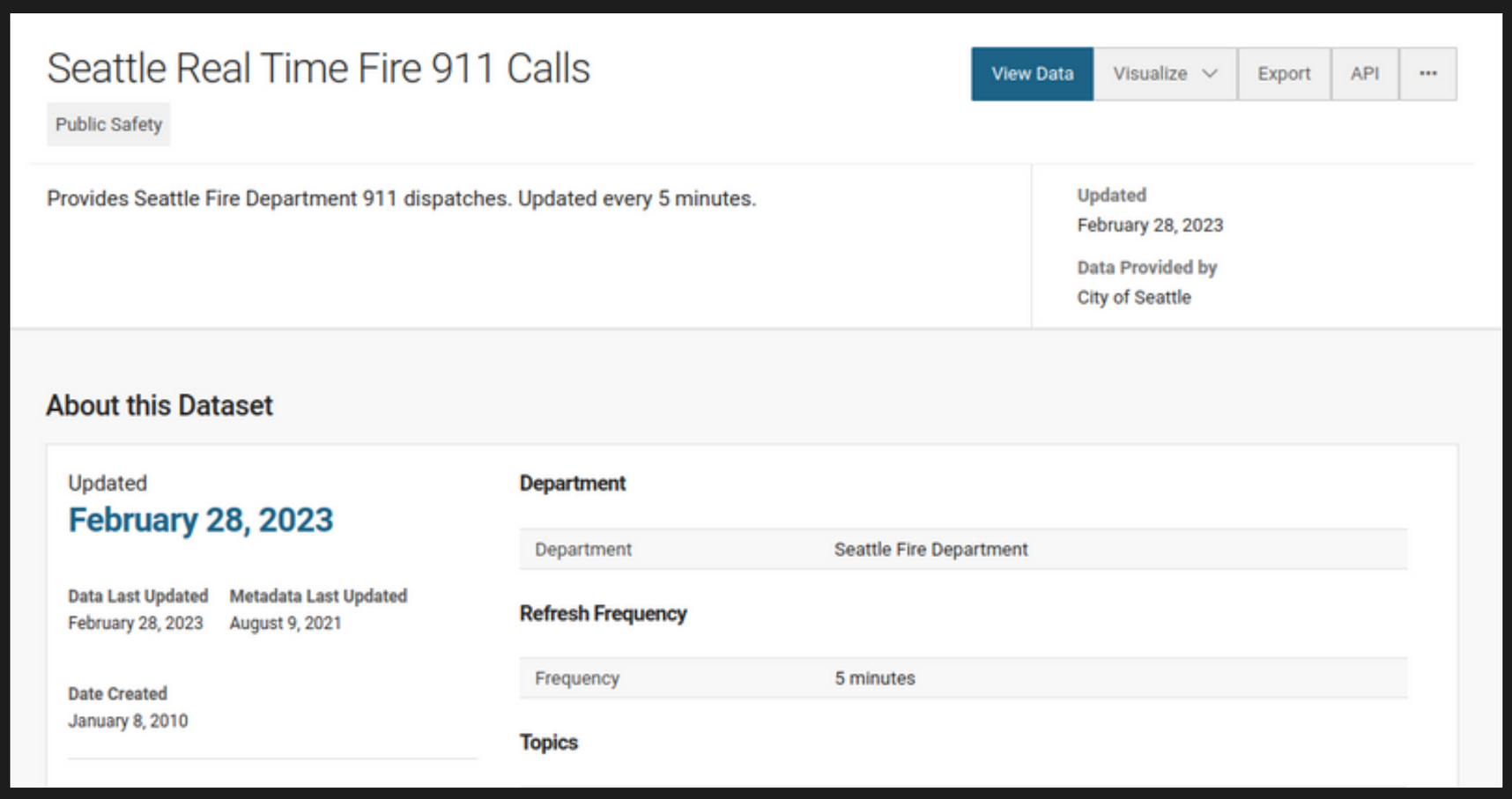
- For the **Batch processing** , we will be working with **911 calls dataset fire department** provided by **DataSF** :

https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek-vuh3/data



- As for the **data stream processing** , we will be working with **Seattle Real Time Fire 911 Calls** provided by **Seattle Open Data** :

https://data.seattle.gov/resource/kzjm-xkqj.json



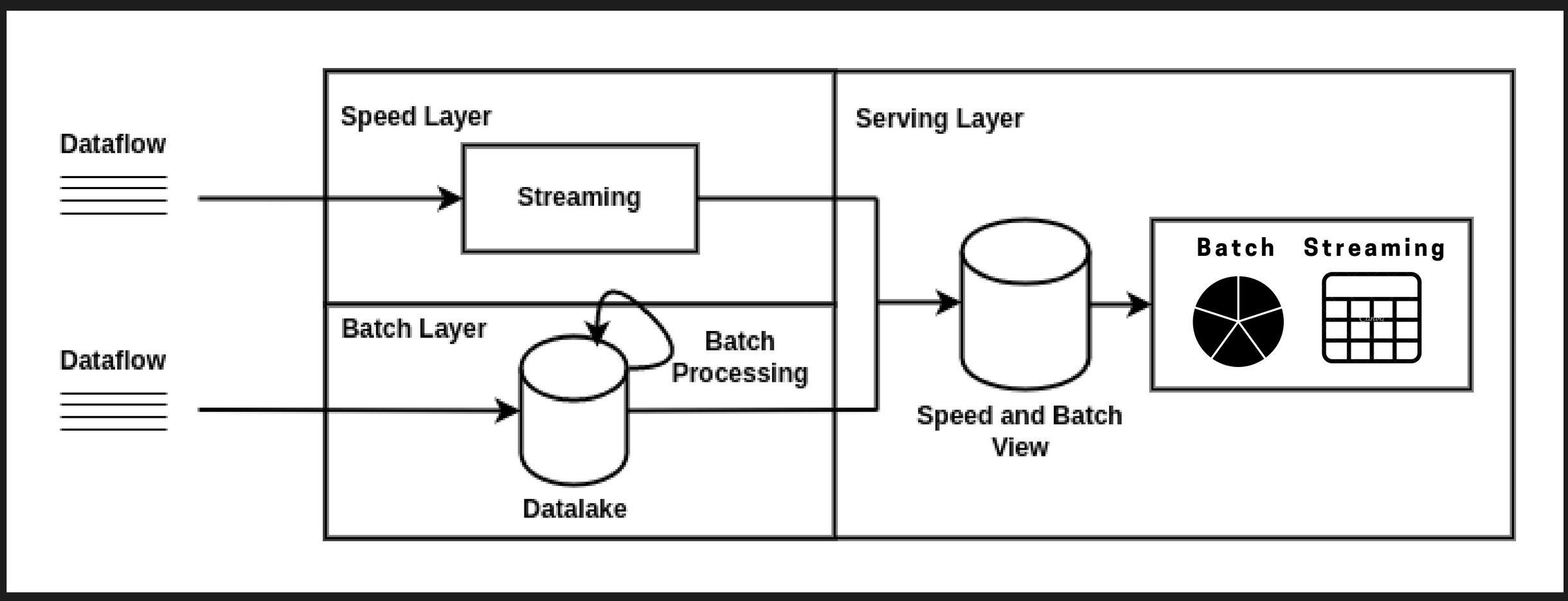
ARCHITECTURE

In this project , we will use **the lambda architecture** :

- **For Data storage**: We will use a distributed file system : **Hadoop HDFS** to store the batch data and **kafka** to store the stream data.

- **For Data processing:** We will use **Apache Spark** for real-time streaming data processing, and Apache Hive for batch data processing.

- **For Data visualization:** For the straming data visualisation we will be using **web interface developed with NodeJs , HTML and CSS , and MongoDB Compass**



BATCH PROCESSING

In the first part of our project, we obtained a **dataset** from the specified website and developed a **Java application in the form of a JAR file** to process the data. The data processing procedure we implemented was relatively straightforward.

We chose to perform a simple treatment on the data by **counting the number of calls per type**.

To accomplish this, we utilized the dataset within our Java application and implemented the necessary logic to **count the occurrences of each call type**. This allowed us to generate **aggregated statistics** based on the call types present in the dataset.

The **data processing ran successfully** as you can see in the accompanying screenshot. The screenshot captures the relevant information and provides visual confirmation of the job's successful execution.

```
23/05/09 21:24:53 INFO mapreduce.Job: Job job_1683663278269_0006 completed successfully
23/05/09 21:24:53 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=1840
                FILE: Number of bytes written=2231295
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2340334074
                HDFS: Number of bytes written=142
                HDFS: Number of read operations=57
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
```

BATCH PROCESSING

```
        Job Counters
                Killed map tasks=1
                Launched map tasks=18
                Launched reduce tasks=1
                Data-local map tasks=18
                Total time spent by all maps in occupied slots (ms)=372119
                Total time spent by all reduces in occupied slots (ms)=9780
                Total time spent by all map tasks (ms)=372119
                Total time spent by all reduce tasks (ms)=9780
                Total vcore-milliseconds taken by all map tasks=372119
                Total vcore-milliseconds taken by all reduce tasks=9780
                Total megabyte-milliseconds taken by all map tasks=381049856
                Total megabyte-milliseconds taken by all reduce tasks=10014720
```

For further technical details on how the job was run , please check the **README** file of our **github repository**

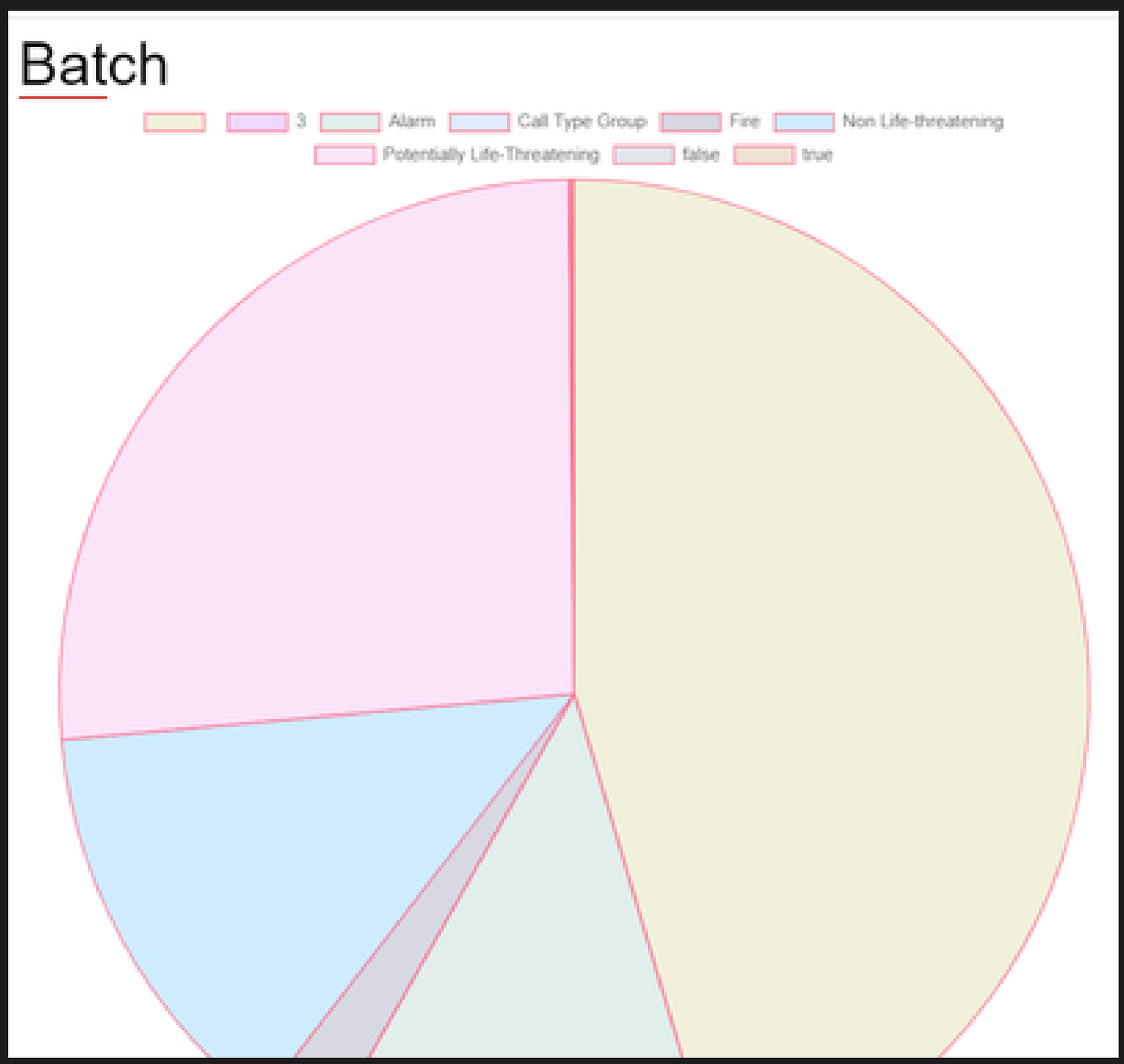
Once we obtained the **desired results**, we proceeded to store the processed data in a **MongoDB database** called **"fire-calls"**

Specifically, we created a collection named **"fire-calls-batch"** within the database as you can see in the sreenshot bellow :

```
> db['fire-calls-batch'].count()
9
> db['fire-calls-batch'].find()
{ "_id" : BinData(3,"WUYmxkYauGsJAG3JAX4zrQ=="), "type" : "", "count" : "2816296" }
{ "_id" : BinData(3,"8UtEiSfAH4np5GJgwGV3nw=="), "type" : "3", "count" : "7" }
{ "_id" : BinData(3,"SU4s3dyrbHRxTbEC68LTtg=="), "type" : "Alarm", "count" : "792276" }
{ "_id" : BinData(3,"GEktlw25V05IKudWCBVbgw=="), "type" : "Call Type Group", "count" : "1" }
{ "_id" : BinData(3,"4ko7NJ5SlslQVQNuxYnotg=="), "type" : "Fire", "count" : "140506" }
{ "_id" : BinData(3,"BEgSQwoCoY1EkT8M2uBojQ=="), "type" : "Non Life-threatening", "count" : "814879" }
{ "_id" : BinData(3,"CUQGDfe8yuV62tvxr9zitg=="), "type" : "Potentially Life-Threatening", "count" : "1627956" }
{ "_id" : BinData(3,"3kpQOo3q/FEFn7bi6vwCmw=="), "type" : "false", "count" : "4423" }
{ "_id" : BinData(3,"8EgznXbEGrZJArQz0fuCoA=="), "type" : "true", "count" : "6108" }
>
```

BATCH PROCESSING

Then , we seamlessly exported the data from our **MongoDB database** in the docker container to **MongoDB Compass**, ensuring easy accessibility for further analysis and visualization.
Building upon this foundation, we proceeded to develop a user-friendly **web interface** using a combination of **Node.js, HTML, and CSS.**



DATA STREAM PROCESSING

In this second part of the project, we developed a **Java application** to process streaming data from an **API.** The processing step involves a **simple filtering of data** to **extract the following fields** for each row:

Incident number | address | type | datetime

To **store the streaming data**, we will utilize **Kafka** as a **message broker**. The application will consist of :

- **Kafka producer** responsible for **retrieving data** from the API, processing it, and sending it to the Kafka topic named **'fire-calls'**

```
F230054786 | 26th Ave S / S Jackson St | Natural Gas Odor | 2023-05-06T19:51:00.000
F230054785 | 6901 M L King Jr Way S | Aid Response | 2023-05-06T19:51:00.000
F230054784 | Maynard Ave S / S Lane St | Aid Response | 2023-05-06T19:50:00.000
F230054783 | 7939 Delridge Way Sw | Medic Response | 2023-05-06T19:48:00.000
F230054780 | 4th Ave / Seneca St | Aid Response | 2023-05-06T19:38:00.000
F230054778 | 4857 Rainier Ave S | Aid Response | 2023-05-06T19:37:00.000
F230054776 | 9101 M L King Jr Way S | Aid Response | 2023-05-06T19:34:00.000
F230054775 | 610 2nd Ave N | Aid Response | 2023-05-06T19:29:00.000
F230054774 | 1011 S Weller St | Auto Fire Alarm | 2023-05-06T19:27:00.000
F230054773 | 3814 Ne 90th St | Medic Response | 2023-05-06T19:24:00.000
F230054771 | 1250 1st Ave S | EVENT - Special Event | 2023-05-06T19:13:00.000
F230054772 | 1250 1st Ave S | Single Medic Unit | 2023-05-06T19:13:00.000
F230054770 | 1959 Ne Chelan Ln | Aid Response | 2023-05-06T19:11:00.000
F230054769 | 500 5th Ave | Aid Response | 2023-05-06T19:08:00.000
F230054768 | 407 Ne 45th St | Nurseline/AMR | 2023-05-06T19:05:00.000
F230054767 | 1334 Alaskan Way | Rescue Elevator | 2023-05-06T18:59:00.000
F230054766 | 612 7TH AVE S | Medic Response- Overdose | 2023-05-06T18:55:00.000
F230054765 | 2820 Ne 127th St | Medic Response | 2023-05-06T18:52:00.000
F230054764 | 20th Ave Ne / Ne 77th St | MVI - Motor Vehicle Incident | 2023-05-06T18:52:00.000
F230054762 | 2208 15th Ave W | Auto Fire Alarm | 2023-05-06T18:50:00.000
F230054761 | 2821 Sw Yancy St | Aid Response | 2023-05-06T18:46:00.000
F230054759 | 6515 12th Ave Ne | Aid Response | 2023-05-06T18:41:00.000
F230054757 | 401 Ne Northgate Way | Aid Response | 2023-05-06T18:34:00.000
F230054756 | 15th Ave Ne / Ne 125th St | Aid Response | 2023-05-06T18:34:00.000
F230054755 | 1400 3rd Ave | Aid Response | 2023-05-06T18:30:00.000
F230054754 | 700 N 50th St | Medic Response | 2023-05-06T18:26:00.000
F230054753 | 5449 Ballard Ave Nw | Aid Response | 2023-05-06T18:25:00.000
F230054751 | 1st Ave Ne / N Northgate Way | MVI - Motor Vehicle Incident | 2023-05-06T18:22:00.000
F230054750 | 2208 15th Ave W | Auto Fire Alarm | 2023-05-06T18:21:00.000
F230054748 | 1407 2nd Ave W | Aid Response | 2023-05-06T18:06:00.000
F230054746 | E John St / Broadway E | Aid Response | 2023-05-06T18:05:00.000
F230054744 | 411 3rd Ave | Aid Response | 2023-05-06T18:03:00.000
F230054743 | E Thomas St / Harvard Ave E | Medic Response | 2023-05-06T18:02:00.000
F230054741 | 6727 Rainier Ave S | Medic Response | 2023-05-06T17:56:00.000
F230054740 | 509 3rd Ave | Aid Response | 2023-05-06T17:55:00.000
F230054738 | 10564 5th Ave Ne | Aid Response | 2023-05-06T17:48:00.000
--------------------------------------------------------------------------------

sending the results to the topic 'fire-calls' ...
```

For further technical details on how the application was run, please check the **README** file of our **github repository**

DATA STREAM PROCESSING

- **Kafka consumer** that will retrieve **the processed data** from the **'fire-calls'** topic and store it in the **MongoDB** database named **'fire-calls.'**Specifically, the consumer will save the data in the **'fire-calls-spark'** collection.

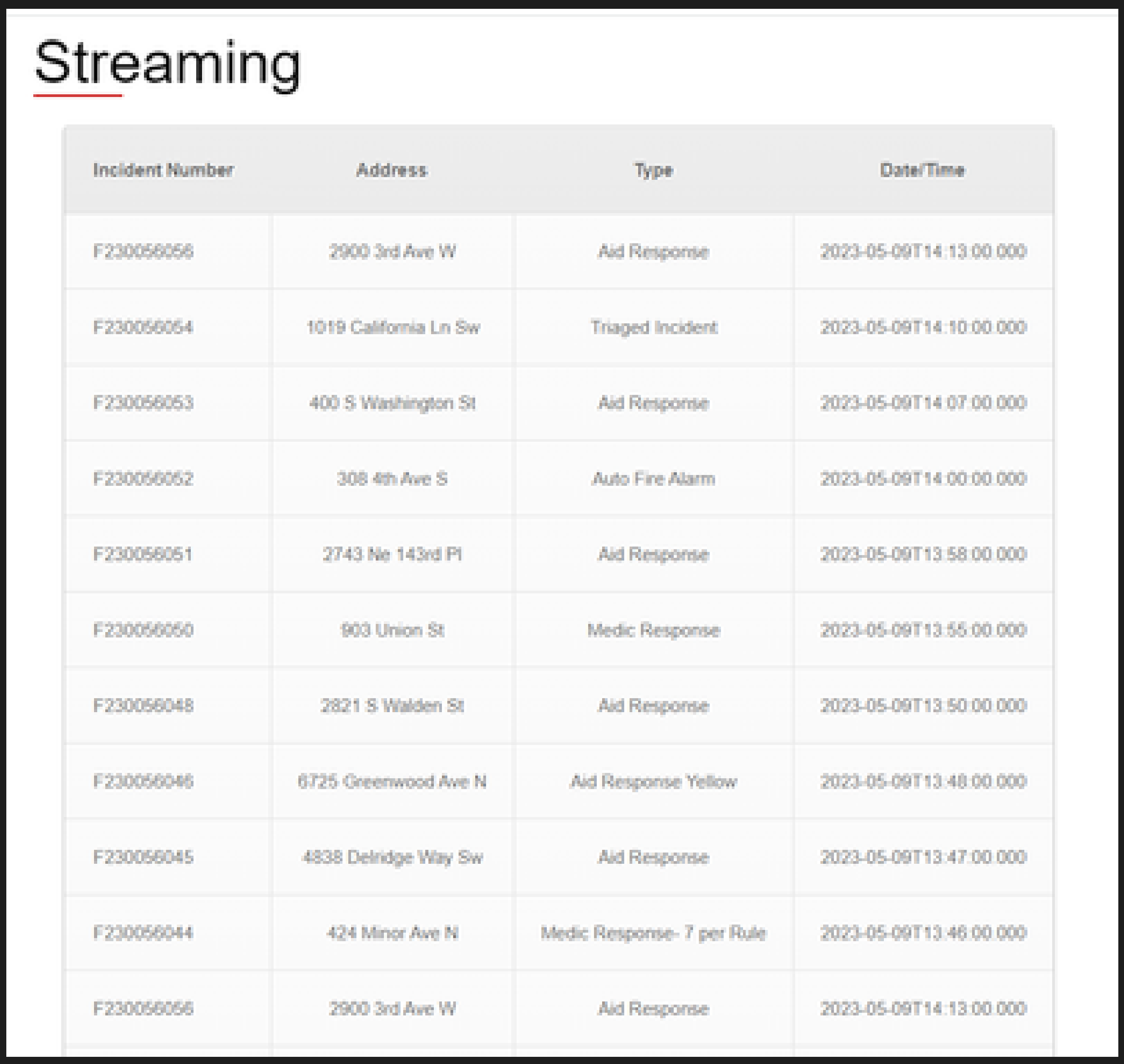
```
F230054769 | 500 5th Ave | Aid Response | 2023-05-06T19:08:00.000
F230054768 | 407 Ne 45th St | Nurseline/AMR | 2023-05-06T19:05:00.000
F230054767 | 1334 Alaskan Way | Rescue Elevator | 2023-05-06T18:59:00.000
F230054766 | 612 7TH AVE S | Medic Response- Overdose | 2023-05-06T18:55:00.000
F230054765 | 2820 Ne 127th St | Medic Response | 2023-05-06T18:52:00.000
F230054764 | 20th Ave Ne / Ne 77th St | MVI - Motor Vehicle Incident | 2023-05-06T18:52:00.000
F230054762 | 2208 15th Ave W | Auto Fire Alarm | 2023-05-06T18:50:00.000
F230054761 | 2821 Sw Yancy St | Aid Response | 2023-05-06T18:46:00.000
F230054759 | 6515 12th Ave Ne | Aid Response | 2023-05-06T18:41:00.000
F230054756 | 15th Ave Ne / Ne 125th St | Aid Response | 2023-05-06T18:34:00.000
F230054757 | 401 Ne Northgate Way | Aid Response | 2023-05-06T18:34:00.000
F230054755 | 1400 3rd Ave | Aid Response | 2023-05-06T18:30:00.000
F230054754 | 700 N 50th St | Medic Response | 2023-05-06T18:26:00.000
F230054753 | 5449 Ballard Ave Nw | Aid Response | 2023-05-06T18:25:00.000
F230054751 | 1st Ave Ne / N Northgate Way | MVI - Motor Vehicle Incident | 2023-05-06T18:22:00.000
F230054750 | 2208 15th Ave W | Auto Fire Alarm | 2023-05-06T18:21:00.000
F230054748 | 1407 2nd Ave W | Aid Response | 2023-05-06T18:06:00.000
F230054746 | E John St / Broadway E | Aid Response | 2023-05-06T18:05:00.000
F230054744 | 411 3rd Ave | Aid Response | 2023-05-06T18:03:00.000
F230054743 | E Thomas St / Harvard Ave E | Medic Response | 2023-05-06T18:02:00.000
F230054741 | 6727 Rainier Ave S | Medic Response | 2023-05-06T17:56:00.000
F230054740 | 509 3rd Ave | Aid Response | 2023-05-06T17:55:00.000
F230054738 | 10564 5th Ave Ne | Aid Response | 2023-05-06T17:48:00.000
-------------------------------------------------------------------------------

configuring MongoDB and establishing connection ...
23/05/09 21:56:44 INFO driver.cluster: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelection
out='30000 ms', maxWaitQueueSize=500}
retrieving the database then the collection ...
inserting the records ...
23/05/09 21:56:44 INFO driver.cluster: Cluster description not yet available. Waiting for 30000 ms before timing out
23/05/09 21:56:44 INFO driver.connection: Opened connection [connectionId{localValue:201, serverValue:205}] to localhost:27017
23/05/09 21:56:44 INFO driver.cluster: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=
DALONE, state=CONNECTED, ok=true, version=ServerVersion{versionList=[2, 6, 10]}, minWireVersion=0, maxWireVersion=2, maxDocumentSize=16777216, logicalSe
nTimeoutMinutes=null, roundTripTimeNanos=153802}
23/05/09 21:56:44 INFO driver.connection: Opened connection [connectionId{localValue:202, serverValue:206}] to localhost:27017
records inserted ...
23/05/09 21:56:44 INFO driver.connection: Closed connection [connectionId{localValue:202, serverValue:206}] to localhost:27017 because the pool has been
sed.
```

```
Type "it" for more
> db['fire-calls-spark'].count()
151000
> db['fire-calls-spark'].count()
153000
> db['fire-calls-spark'].count()
157000
```

The above screenshot shows how the **resulted data** is **increasingly** stored in the **mongoDB database**

DATA STREAM PROCESSING

To maintain consistency and ensure a seamless user experience, we have opted to employ the same **visualization** approach for both batch processing and real-time data streaming in our project using both **MongoDB Compass** and **Web interface** that is updated everytime we refresh the page to get the stream.



Streaming

Incident Number	Address	Type	Date/Time
F230056056	2900 3rd Ave W	Aid Response	2023-05-09T14:13:00.000
F230056054	1019 California Ln Sw	Triaged Incident	2023-05-09T14:10:00.000
F230056053	400 S Washington St	Aid Response	2023-05-09T14:07:00.000
F230056052	308 4th Ave S	Auto Fire Alarm	2023-05-09T14:00:00.000
F230056051	2743 Ne 143rd Pl	Aid Response	2023-05-09T13:58:00.000
F230056050	903 Union St	Medic Response	2023-05-09T13:55:00.000
F230056048	2821 S Walden St	Aid Response	2023-05-09T13:50:00.000
F230056046	6725 Greenwood Ave N	Aid Response Yellow	2023-05-09T13:48:00.000
F230056045	4838 Delridge Way Sw	Aid Response	2023-05-09T13:47:00.000
F230056044	424 Minor Ave N	Medic Response- 7 per Rule	2023-05-09T13:46:00.000
F230056056	2900 3rd Ave W	Aid Response	2023-05-09T14:13:00.000

For further information about the web interface implementation, please check our **github repository**