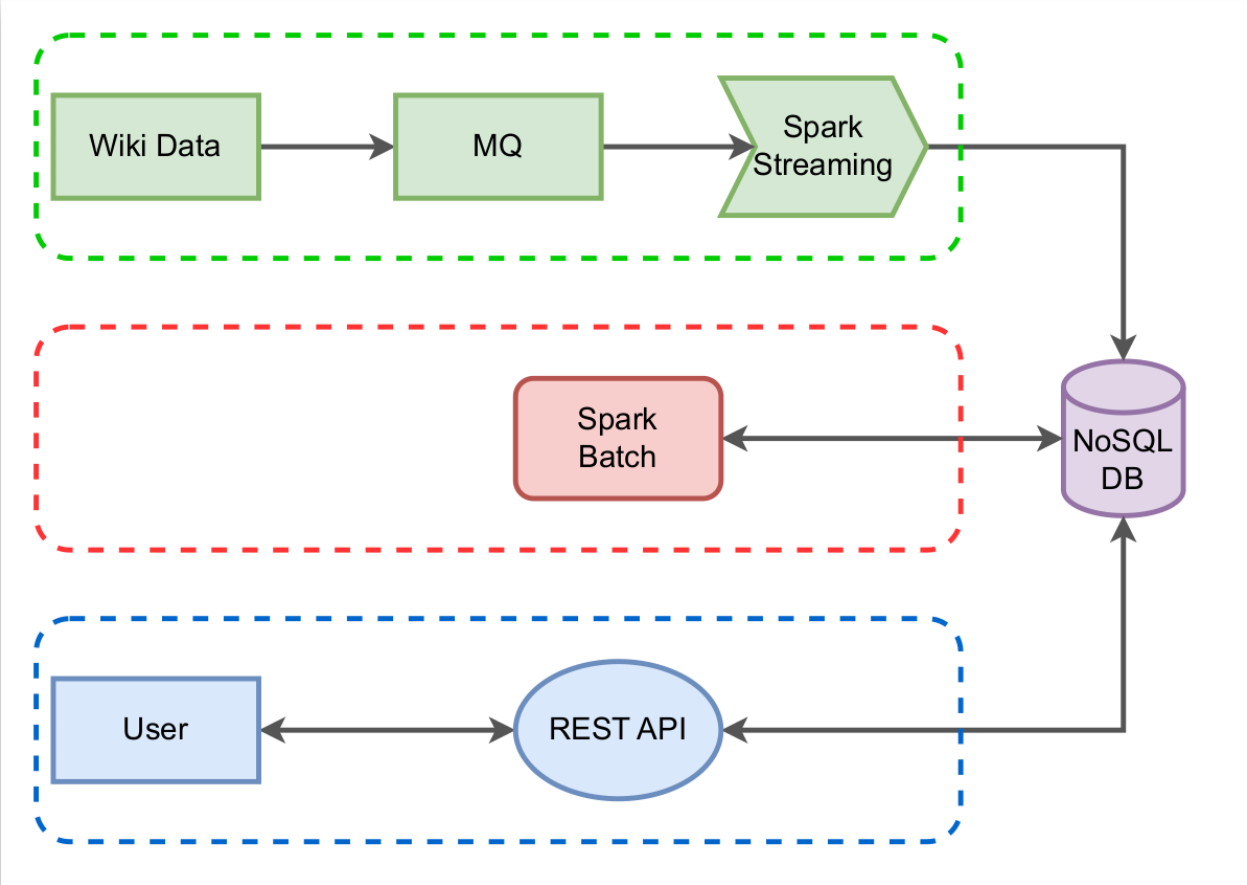
Big Data Project

https://github.com/be-unkind/big-data-ucu-project

Authors:
Anastasiia Havryliv
Yaroslav Romanus

System Design



On this diagram, you can see a high-level overview of our system. It works with a data stream from Wiki - it contains info about newly created pages.

1.) **Data reading pipeline**. It is on our diagram in the green color. Our simple python worker reads this data from the stream and writes a raw version of it into the message queue. We decided to use Kafka in our system. Inspired by the streaming nature of our data, we decided to use the corresponding streaming tool - Spark Streaming. This service is used to process data from message queue and write it into storage, NoSQL DB - Cassandra. More details on the

Data Model and why we decided to use NoSQL (Cassandra) will be in the following section.

2.) **Data processing pipeline.** It is on our diagram in the red color. To create precomputed statistics based on our data for 6 hours, we decided to use a separate service with batch processing - Spark. Simple spark program runs each hour, reads data from Cassandra, aggregates it and writes precomputed results back for storing to the corresponding tables.

3.) **API of our system.** It is on our diagram in the blue color. To interact with the outside world, our system contains REST API, which answers the queries of users: reads data from our DB, processes it and returns results back to the user in the JSON format.

Cassandra

First of all, a few words on why we have chosen to work with NoSQL DB: we have unstructured data. For example for aggregated statistics in precomputed part of the requests, we have nested data structures, to help with easier data storage.
Also, in Category B, in the part with ad-hoc queries, we need fast access to the data, and it is easier to match tables to the requests in greater detail.
So these statements mentioned above were the main reason to stick to the NoSQL databases, and in particular Cassandra DB.

Below is the short description of the tables, each of which was designed specifically for the particular query:

Precomputed reports:

1. Return the aggregated statistics containing the number of created pages for each Wikipedia domain for each hour in the last 6 hours, excluding the last hour.

```
CREATE TABLE domain_stats (
    start_hour TIMESTAMP,
    end_hour TIMESTAMP,
    statistics list<frozen<domain_hour_stats>>,
    PRIMARY KEY (start_hour)
);
```

This table contains the aggregated statistics for each hour. Here we have Primary Key that consist only of **start_hour**, as later in the queries, we will select needed rows by start hour of a specific statistics report.

```
CREATE TYPE domain_hour_stats (
  domain TEXT,
  created_pages INT
);
```

For this table we have statistics as a unique-made type, which represents domain and the quantity of created pages. In the **domain_table** we will have a list, where values are consistent with this type.

2. Return the statistics about the number of pages created by bots for each of the domains for the last 6 hours, excluding the last hour.

```
CREATE TABLE domain_bot_stats (
    start_hour TIMESTAMP,
    end_hour TIMESTAMP,
    statistics list<frozen<domain_hour_stats>>,
    PRIMARY KEY (start_hour)
);
```

This table also has a Primary Key consisting of **start_hour**, but in the statistics column (which is a list of values of type **domain_hour_stats**, as in the previous task), but in this list we will have count only of the pages that were created by bots.

3. Return Top 20 users that created the most pages during the last 6 hours, excluding the last hour. The response should contain user name, user id, start and end time, the list of the page titles, and the number of pages created.

```
CREATE TABLE user_stats (
    start_hour TIMESTAMP,
    end_hour TIMESTAMP,
    statistics list<frozen<user_hour_stats>>,
    PRIMARY KEY (start_hour)
);
```

This page has **start_hour** as its Primary Key, which will allow us to select statistics based on their start hour.

```
CREATE TYPE user_hour_stats (
    user_name TEXT,
    user_id INT,
    created_pages INT,
    page_titles list<text>
);
```

Type user_hour_stats represents information about the user (name of the user, user id, number of pages, created by this user and list of all pages that this user created). This type is used in table user_stats, where it represents the type of values in the statistics list.

Ad-hoc queries:

1. Return the list of existing domains for which pages were created.

```
CREATE TABLE domain_table (
    domain TEXT,
    page_id INT,
    PRIMARY KEY ((domain), page_id)
);
```

This table contains information about every page that was created on every domain. The Primary Key consists of **domain** as Partition Key, and **page_id** as Clustering Key. We are using the Clustering Key along with the Partition Key as all domains are not unique.

2. Return all the pages which were created by the user with a specified user_id.

```
CREATE TABLE user_table (
    user_id INT,
    page_id INT,
    page_title TEXT,
    PRIMARY KEY ((user_id), page_id)
);
```

This table contains information about pages that were created by each user. Here, as a Primary Key we have a combination of **user_id** as a Partition Key and **page_id** as a Clustering Key.

3. Return the number of articles created for a specified domain.

```
CREATE TABLE domain_table (
    domain TEXT,
    page_id INT,
    PRIMARY KEY ((domain), page_id)
);
```

For this query we are using the same table as in the first. And in the query we are counting the number of pages for a specified domain (as it is a Partition Key).

4. Return the page with the specified page_id

```
CREATE TABLE page_table (
    page_id INT,
    page_title TEXT,
    PRIMARY KEY (page_id)
);
```

Here we are storing the data about pages. Primary Key consists only of **page_id**, because it is unique, and it is the only parameter we need in the query.

5. Return the id, name, and the number of created pages of all the users who created at least one page in a specified time range.

```
CREATE TABLE user_time_table (
    user_id INT,
    user_text TEXT,
    page_id INT,
    rev_timestamp TIMESTAMP,
    page_title TEXT,
    PRIMARY KEY ((rev_timestamp), user_id, user_text)
);
```

This table contains information about creation of pages by each user, while also containing the information about time of the pages creation. As we have Primary Key, which consists of **rev_timestamp** as a Partition Key along with **user_id** and **user_text**

as Clustering Keys, we are able to select only pages that were created in the specified time range.

System Work Results

https://github.com/be-unkind/big-data-ucu-project/tree/main/results