**Real-Time Election Voting - Data Engineering Project**

**Introduction:**

Human beings are political in nature, and elections are a way to ensure free, fair, credible, and transparent methods of putting people in power. The current election system is mainly based on the ballot system, where parties and candidates are listed on paper, and people go to vote for their preferred candidate.

This project proposes building a real-time election voting system to ensure instant and real-time socialization of voting data and results. The system will perform voter accreditation, candidate accreditation, and the voting process in real time.

A group of people in different poses

Description automatically generated

**Functionality:**

The architecture of the project includes parties, candidates, and voters registered on the platform, with their information saved in a Postgres database and streamed simultaneously into Kafka.

A Spark job will listen for events coming into Kafka, consume and aggregate the data, and stream it back into another Kafka topic. Streamlit will listen for events coming into the Kafka topic and visualize the results in real time.

A diagram of a computer

Description automatically generated

* This will show how each candidate is overtaking each other as voters vote for them. In the coding section, we have created a new project, installed the necessary packages, set up the architecture using Docker Compose, and brought up and fired up the Docker instance.

**Data Generation Module:**

The Data Generation module is responsible for creating synthetic voter and candidate data to simulate user interactions within the system. This module utilizes the "Random User Generator" API to generate realistic user profiles.

1. Voter Data Generation

The system fetches random user data from the Random User Generator API with the nationality parameter set to "gb" (Great Britain). The retrieved user data includes information such as name, date of birth, gender, nationality, address, email, phone number, and a profile picture.

Example Voter Data:

A screen shot of a computer

Description automatically generated

2. Candidate Data Generation

Similar to voter data generation, the system fetches random user data with a specific gender parameter to simulate diverse candidates. The generated candidate data includes candidate ID, name, party affiliation, biography, campaign platform, and a photo URL.

**Implementation Details:**

The module uses the Python **requests** library to make HTTP requests to the API. The random module is employed to seed and control the randomness of the generated data. The Data Generation module is invoked to produce synthetic voter and candidate data for populating the system with realistic user profiles. This simulated data is utilized for testing, development, and demonstration purposes.

**Schema:**

Here is the schema for all three tables:

Candidates:

A close-up of a computer code

Description automatically generated

Voters:

A screenshot of a computer code

Description automatically generated

Votes:

A close-up of a computer code

Description automatically generated

**System Components**

* **main.py**: This is the main Python script that creates the required tables on Postgres (candidates, voters , and votes) and handles any data quality errors, if any; it also creates the Kafka topic and creates a copy of the votes table in the Kafka topic. It also contains the logic to consume the votes from the Kafka topic and produce data to voters\_topic on Kafka. The code will ensure that all the tables are created before inserting any.
  + Data-Quality Check: The system will ensure that there are no duplicate voter IDs and candidate IDs, and that each voter is unique.
* **voting.py**: This Python script contains the logic to consume the votes from the Kafka topic (voters\_topic), generate voting data, and produce data to votes\_topic on Kafka. This will ensure that the Kafka topic is properly set up and the data is produced and consumed correctly. It will produce voters' information into the Kafka topic, allowing for real-time data processing and visualization. The code’s logic will handle any errors that may occur during the data production and insertion process.
* **spark-streaming.py**: This Python script contains the logic to consume the votes from the Kafka topic (votes\_topic), enrich the data from Postgres, aggregate the votes, and produce data to specific topics on Kafka.
* **streamlit-app.py**: This is the Python script that contains the logic to consume the aggregated voting data from the Kafka topic as well as Postgres and display the voting data in realtime using Streamlit. It will visualize the real-time data, showing how each candidate is overtaking each other as voters vote for them.

**Data Quality Checks Module**

1. **Referential Integrity Check**

The system performs referential integrity checks between the "votes" and "voters" tables to ensure that each vote has a corresponding voter. Any discrepancies in the referential integrity are flagged for further investigation.

2. **Sanity Row Counts Check**

This check ensures that the number of records in each of the key tables ("voters," "candidates," and "votes") aligns with expectations. Any significant deviations from expected row counts are reported as potential data anomalies.

Below is the screenshot of a few data quality checks:

**A screenshot of a computer code

Description automatically generated**

**Dashboard:**

**A screenshot of a computer

Description automatically generated**

**Conclusion:**

In conclusion, the Real-Time Election Voting System project aims to revolutionize the traditional election process by introducing a dynamic and instantaneous platform. By leveraging cutting-edge technologies such as Kafka, Spark, and Streamlit, the system ensures the swift socialization of voting data and results, fostering transparency and real-time insights.

The project's architecture seamlessly integrates parties, candidates, and voters, storing their information in a robust Postgres database. The use of Kafka facilitates the smooth streaming of data, allowing Spark to aggregate and enhance the information before presenting it to users through a responsive Streamlit interface.

The Data Generation module creatively utilizes the "Random User Generator" API to simulate user interactions within the system, generating synthetic voter and candidate data for testing and development purposes.

The system's components, including main.py, voting.py, spark-streaming.py, and streamlit-app.py, work in harmony to handle data quality checks, manage Kafka topics, and provide an engaging real-time voting experience.

In summary, the Real-Time Election Voting System project not only introduces a novel way of conducting elections but also sets a precedent for incorporating robust data quality practices, ensuring the credibility and dependability of the information presented to users. This project stands as a testament to the seamless integration of technology, data generation, and data quality checks to enhance the democratic process.