Voting Demographic Analyzer

## By: Navtejinder Singh & Savio Moonnumackel

Table of Contents

[1 Application Overview 2](#_Toc53221502)

[1.1 Voting demographic analyzer 2](#_Toc53221503)

[1.2 User interaction 2](#_Toc53221504)

[2 Components used 2](#_Toc53221505)

[2.1 Google Products used 2](#_Toc53221506)

[2.1.1 Kubernetes Enging 2](#_Toc53221507)

[2.1.2 Cloud SQL 2](#_Toc53221508)

[2.1.3 BigQuery 2](#_Toc53221509)

[2.2 Languages Used 2](#_Toc53221510)

[3 Architecture 2](#_Toc53221511)

[4 Design 2](#_Toc53221512)

[5 Implementation Plan 2](#_Toc53221513)

[6 Test Plan 2](#_Toc53221514)

# Application Overview

## Voting demographic analyzer

As the presidential elections of the USA are on the verge everyone is wondering who will win the election. Everyone is making their own predictions to who will be the winner but, if a human makes the prediction a little bit of bias is always involved to eliminate that bias we are creating a super cool and efficient web app/ website that will take some mandatory inputs, analyze them and will predict which presidential candidate is a person most likely to vote for. In addition to this we will also give the combined output of the inputs so far for example if we have received inputs from 100 users and out of those 100, 51 voted for candidate A and 49 voted for candidate B. Then we will roll out the overall result announcing the winner with the percentage of the vote share.

## User interaction

Once the user is on the voting demographic analyzer, he will see a button there which says, “start the prediction”. once he clicks that button, he will be taken to a new page where he is required to give some input for example the user’s age, state of residency, profession, salary etc. After a little bit of processing (less than a second) user will receive a output saying what is he most likely to vote for and a result based on the past data which will display who is winning at the time.

Given below is the flow chart:

Diagram

Description automatically generated

# Components used

## Google Products used

### Google Kubernetes Engine

We will use google Kubernetes engine to deploy the website. We will first package our website into a docker image and then we will push that docker image to the google container registry (turning on the GCR API). Then we will create a GKE cluster and we will deploy our website to that GKE cluster. We are using GKE because we can avail a lot of benefits provided by the Google Kubernetes Engine such as auto-scaling, load-balancing, easy upgrades and many more.

### Cloud SQL

We will google cloud SQL to store our website data into google cloud. We are using this service because of the following reasons:

1. It is easily/highly scalable.

2. It is highly available.

3. It is secure.

4. Fully managed and many more benefits

### Big Query

We will use big query to find the current result as to who is leading in the election/who is likely to win the election based on the inputs from the previous as well as current users.

Benefits of Big query are as follows:

1. It is salubrious for large data.

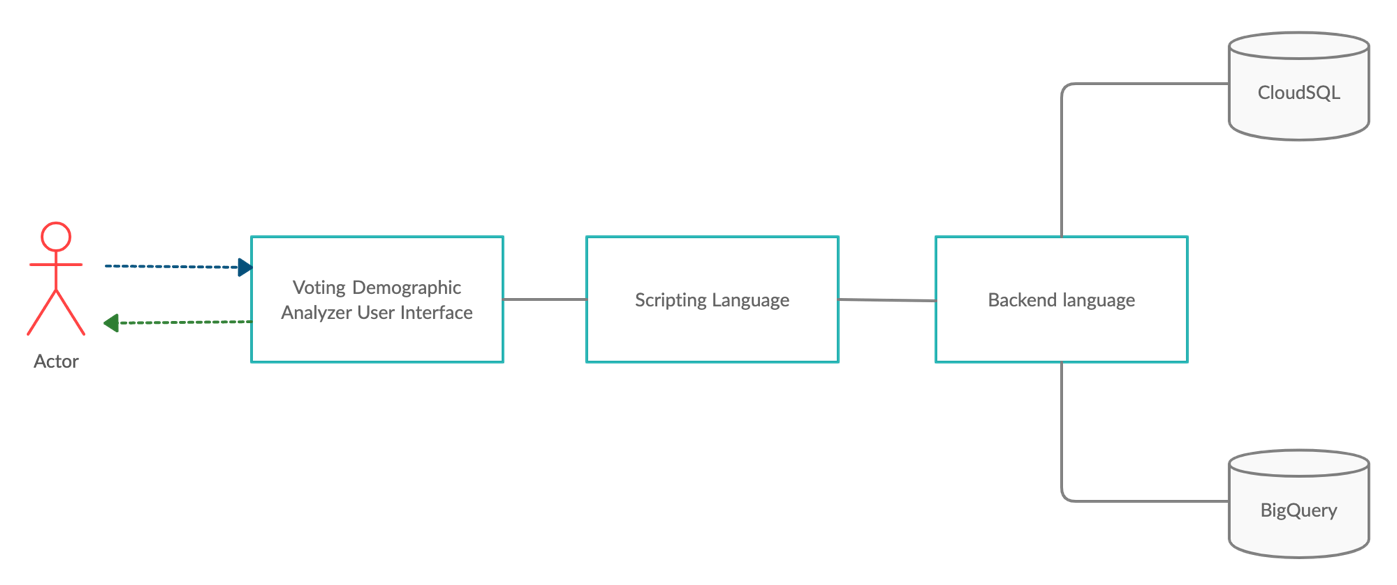
2. It is fully managed

3. It is secure

# Architecture

Voting demographic analyzer accepts data from a front end user through HTML forms. This data is then parsed using the scripting language. This parsed data is then stored in the data base and we use a backend language to hook the application logic with the database persistence layer. Periodically, based on a configurable time interval or delta change to the demographic data, BigQuery will pull transactional data from the DataBase and provide analytical data. This data then travels in the reverse direction through the backend logic to the front end and is provided to the UI user.

This flow can be better visualized as below,

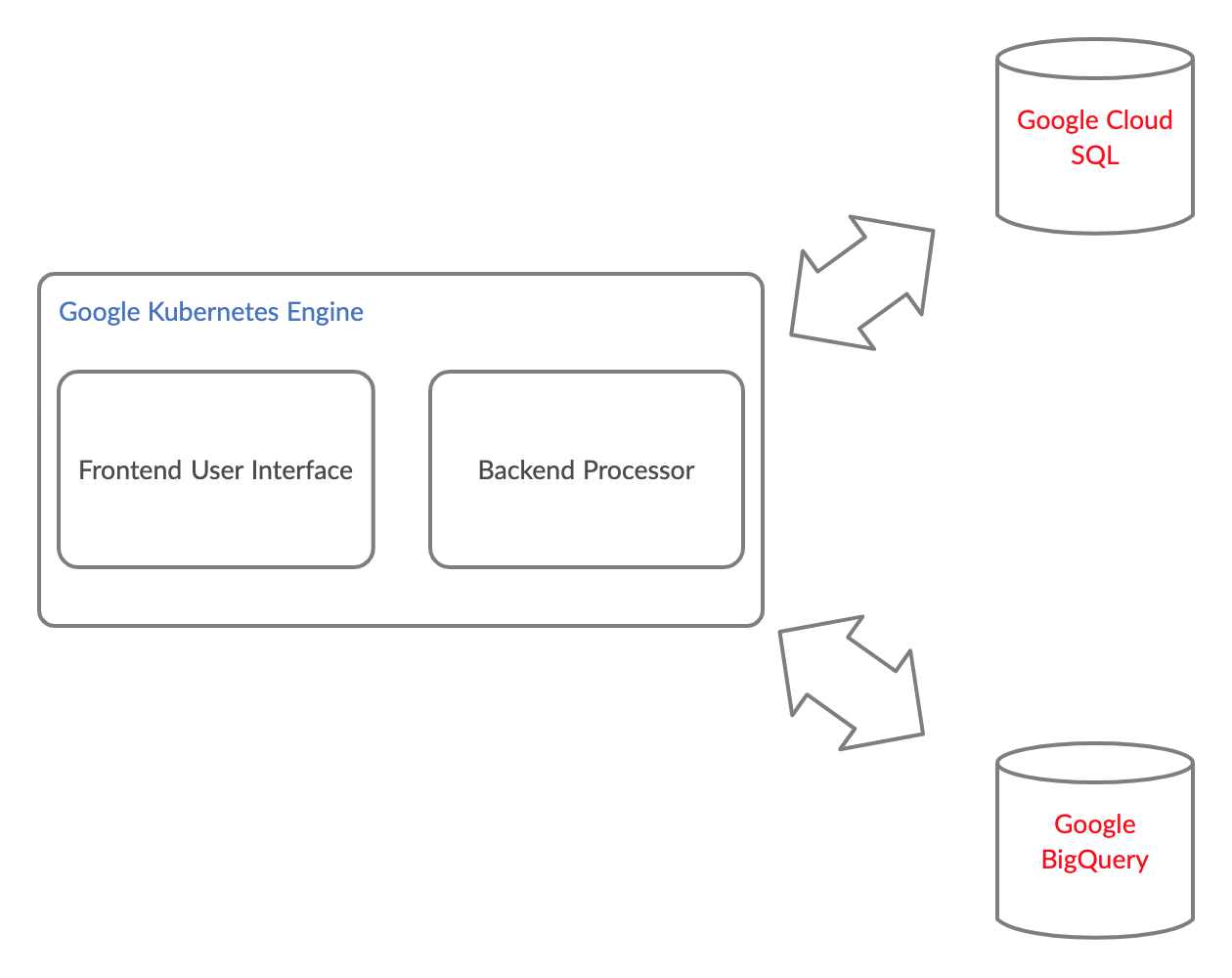


The architecture of Voting Demographic Analyzer can be visualized as 3 main blocks for simplicity,

1. Front End User Interface
2. Backend Processing
3. Data Persistence layer

The front end is responsible for accepting input data from the user and allowing a seamless interaction with the user. The backend processor will process the data entered by the user and will know what to do with it. It will store the data in a database and based on configurable values, it will interact with BigQuery to allow for data analytics. This configurable parameter was chosen as BiqQuery can tend to be expensive and charges per interaction. Now based on financial requirements we can configure the frequency of these transactions.

The Frontend User Interface and the Backend processor will be packaged as a single container which will be hosted via Google Kubernetes Engine. This container will then interface with Google Cloud SQL and Google BigQuery.



# Design

## Languages

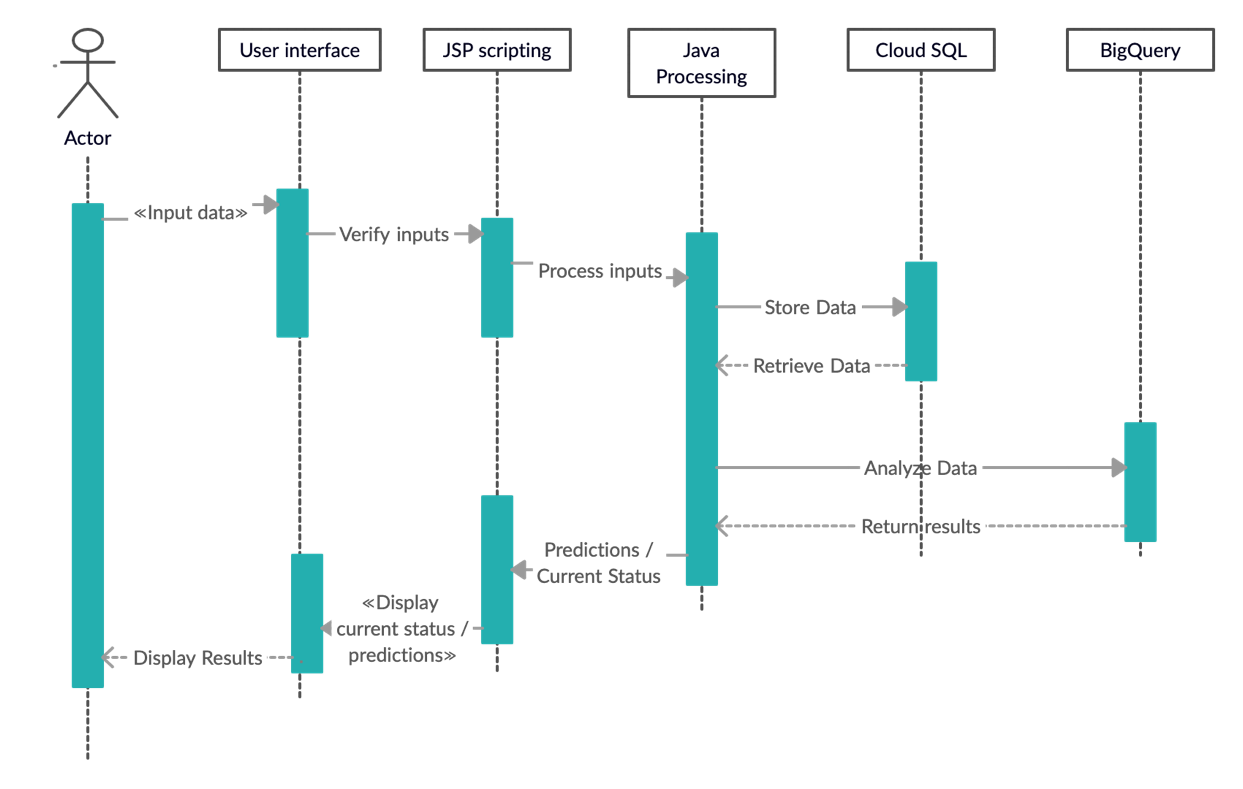
The proposed languages decided for each portion can be seen in the table below.

|  |  |  |
| --- | --- | --- |
| SI No | Functionality/Component | Language/Tool |
| 1. | Frontend | HTML, CSS, Bootstrap |
| 2. | Backend scripting | JSP |
| 3. | Backend processing | Java |
| 4. | Persistence Layer | SQL |

## Workflows

The major workflows in the application can be described as follows:

1. User inputs data to user interface, as a HTML form
2. JSP pulls this data and verifies that the data is actually in the correct format and no erroneous input was received
3. Java processes this data and hooks it with the cloud SQL
4. The java processor can store data and retrieve data from the cloud SQL
5. The Java processor requests BigQuery to analyze data, BigQuery returns the results
6. Java processor returns data to JSP
7. JSP returns data to a HTML page



# Implementation Plan

We intend to implement the project in a collaborative manner however due to the current circumstances of Covid-19 we will try to do as much as possible virtually and hold personal meetings only at milestones.

To harbor a collaborative approach, we will be storing all our data in github at the below address:

<https://github.com/saviomoon/CloudVotingAnalyticsProject>

The above project is currently private and access can be granted when requested.

|  |  |  |
| --- | --- | --- |
| Component | Owner | Date |
| Frontend UI | Navtejinder Singh | Oct 28 |
| JSP scripting | Savio Moonnumackel | Oct 28 |
| Java Processing | Savio Moonnumackel | Oct 28 |
| Cloud SQL | Savio Moonnumackel | Oct 28 |
| BigQuery | Navtejinder Singh and Savio Moonnumackel | Nov 12 |
| Additional Features | Navtejinder Singh and Savio Moonnumackel | Nov 26 |
| Testing | Navtejinder Singh and Savio Moonnumackel | Nov 30 |

# Test Plan

|  |  |
| --- | --- |
| Requirement | Test Procedure |
| Functionality testing | It is function specific based on different cases. |
| Usability testing | Different users will test the web app. |
| Compatibility testing | Based on different browsers and operating systems. |
| Performance testing | Test the speed of the response. |

# Teammate Contribution

|  |  |  |
| --- | --- | --- |
| Section | Author | Approval |
| Application Overview | Navtejinder Singh | Savio Moonnumackel approves.  Navtejinder Singh |
| Components Used | Navtejinder Singh | Savio Moonnumackel approves.  Navtejinder Singh |
| Architecture | Savio Moonnumackel | Savio Moonnumackel approves.  Navtejinder Singh |
| Design | Savio Moonnumackel | Savio Moonnumackel approves.  Navtejinder Singh |
| Implementation Plan | Savio Moonnumackel | Savio Moonnumackel approves.  Navtejinder Singh |
| Test Plan | Navtejinder Singh | Savio Moonnumackel approves.  Navtejinder Singh |