Omdena Equilo Project Article

by

Kshitij Singh, Mohammad Sabik Irbaz , Rasha Salim, Aniket Bhunia, Sambit Das, Sambit Mahapatra, Ashpreet Singh, Karimi

How to Approach the Problem

1. Data Collection

As in any machine learning project, data is the fuel and it is essential to get the right kind of data. This project was no exception. Our pipeline consists of two main parts when evaluating a company or a business in the gender equality aspect. First, What are the experiences of the current and previous employers?, and a survey taken by a business representative to measure how close the company is to an ideal environment, offering feedback and recommendations in the process, all that based on the company responses in the tailored survey.

We already had the survey questions, but going back to the reviews, which were considered an important metric to measure the company's status in supporting gender equality in their work environment. The search for websites that included a large number of reviews and as diverse as possible, began. We ended up with a list of over 30 websites with thousands of reviews each, and we set off to collect them all.

We knew what we needed, reviews that targeted companies from all industries and it should be gender related, that we would use later on use to train a sentiment analysis model and classify which are describing a negative, a neutral, or a positive specific quality, related to gender equality.

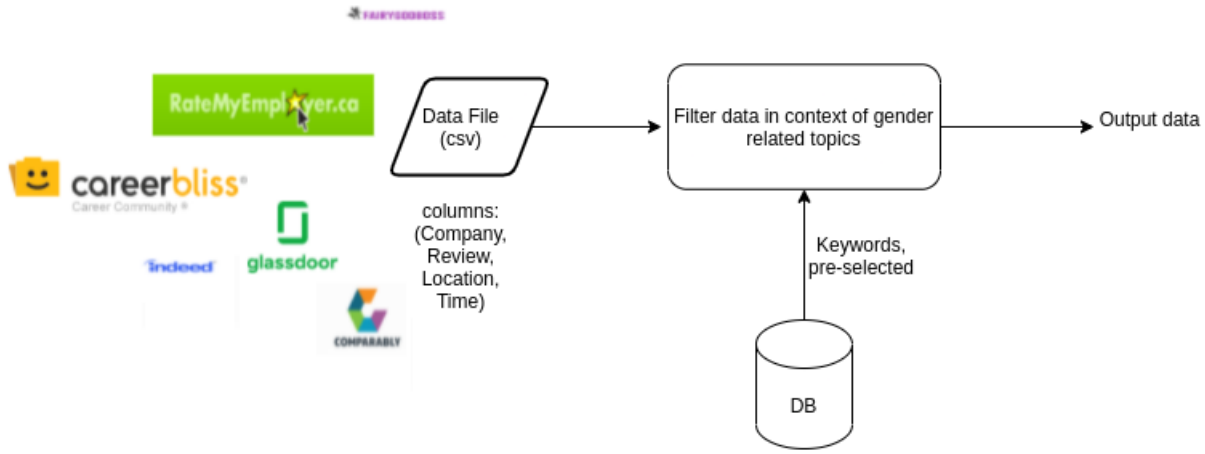


Fig. Data Collection and Filtering

But it was decided to take it one step at a time and collect as many reviews as possible in the first stage, then filter those later to extract the relevant ones (More on that in the upcoming sections).

It turns out though, that web scraping wasn't as straightforward as we thought initially, especially for some of us who were trying it for the first time.

1.1. Technologies Used

Scraping thousands of reviews divided on different websites wasn't something possible to handle by one or two collaborators. We had to stay on schedule and get this data ready to handle for the other tasks to do further processing and work on. Here came the power of teamwork and collaboration. Dozens of people participated in this mission and we ended up with more than what we needed.

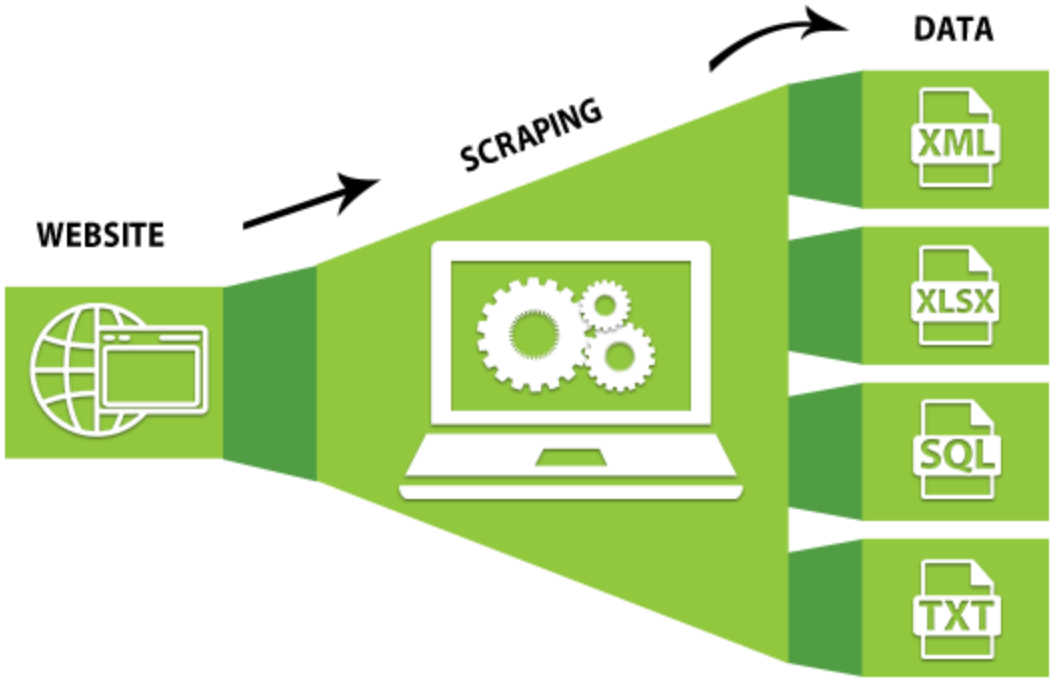


Fig. Web scraping
Image from [Pro web scraping]

The team mainly used *Beautifulsoup* and *Selenium,* both well known Python libraries. Each of these libraries had helped us in totally different types of scenarios.

The challenges of web scraping are well known. Each website has its own structure and design not to mention, their own data sharing policies that we made sure to go through. So at that point each has to get familiar with the website they are scraping and know exactly the location; or here the html tag(s), of the information they are seeking to extract.
(We'll be covering the technical details in an upcoming post), but as a rule of thumb, Beautifulsoup was a great fit, for websites that didn't require login with rather classical design, while on the other hand, Selenium with its automation capabilities, saved us when handling Single Page Application, websites with logins and when there was a need to navigate the page to extract all the reviews; like scrolling down to get more reviews.

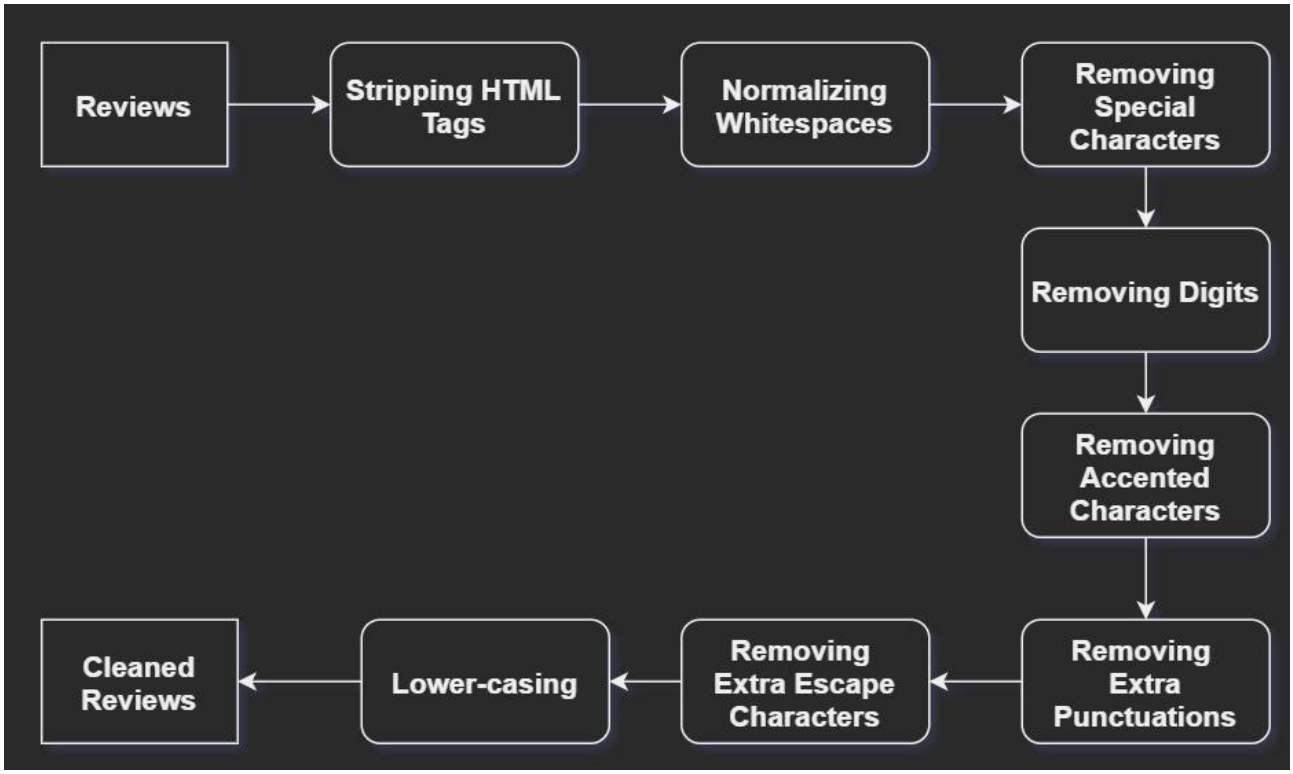
As mentioned earlier, the majority of participants were beginners and had never worked on such a task before. Here the role of people with experience was tested, and they delivered all the way, offering their time and resources and putting people on the right track, which was critical to have the results we were aiming.
We had dozens of different questions, each with different themes and targeting a specific quality in a company, still they needed our attention. Here came the role and amazing work of the data cleaning and processing team.

2. Data Processing

We collected data from different sources. So, it is expected that our data will not be clean and consistent. We go through multiple preprocessing steps to clean our data. Our review data files had 3 types of data for each review (company name, source name and review). We processed them in different ways.

2.1. Processing Reviews

We processed the reviews using the following pipeline:
i) Stripping HTML Tags
ii) Normalizing Whitespaces
iii) Removing Special Characters like emojis
iv) Removing Digits
v) Removing accented characters like (â, î or ô)
vi) Removing extra punctuation marks like (!!!! or ???)
vii) Removing extra escape characters like (\n\n or \r\r)
viii) Lower-casing the whole review text so that it does not remain case-sensitive.



2.2. Processing Company and Review Source names

We processed these two names names using the following pipeline:

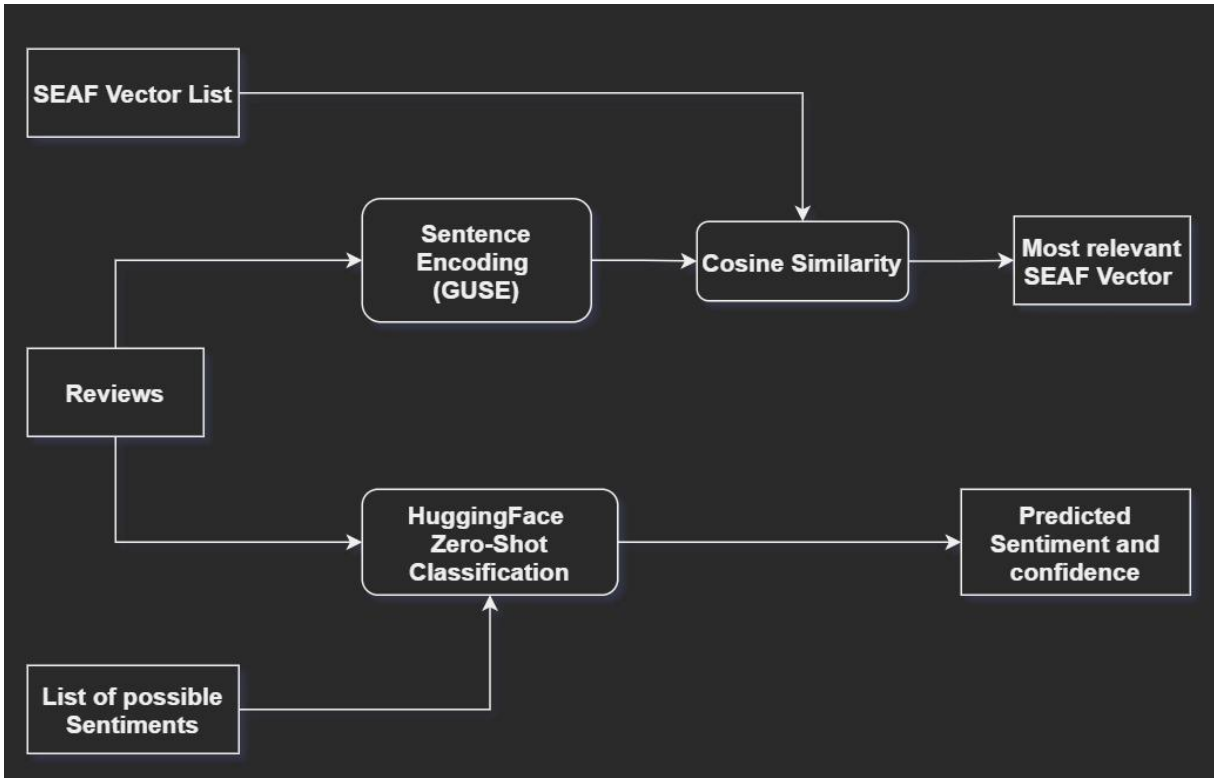
i) Lower-casing the name

ii) Normalizing Whitespaces

iii) Removing the whitespaces with "_"

3. The Machine Learning Phase

Other than survey data, we also used external review data as an additional source of information about a particular company to find out the employee's or ex-employee's point of view. We needed 2 types of detection systems. One was to find out the topic of a review (Topic Modeling) and another was to find out the sentiment (Sentiment Detection).



All our approaches were unsupervised, so at first we built a test dataset for both of the tasks. We tagged our datasets with relevant SEAF vectors and sentiment of that sentence. We tested all our approaches with the test dataset we tagged and finally used the most accurate, confident and most efficient one in the inference period.

Approaches we tried

3.1. Topic Modeling

i) Google Universal Sentence Encoding (GUSE):
ii) Siamese BERT (SBERT)
iii) LDA, Seed-Guided LDA
iv) Keyword extraction with Rake and Yake
v) Keyword extraction with POS tagging

3.1. Sentiment Detection

i) HuggingFace Zero-shot Classification
ii) Flair for Sentiment Analysis
iii) BERT for Sequence Classification
iv) VADER Sentiment Analysis

4. Building The Application

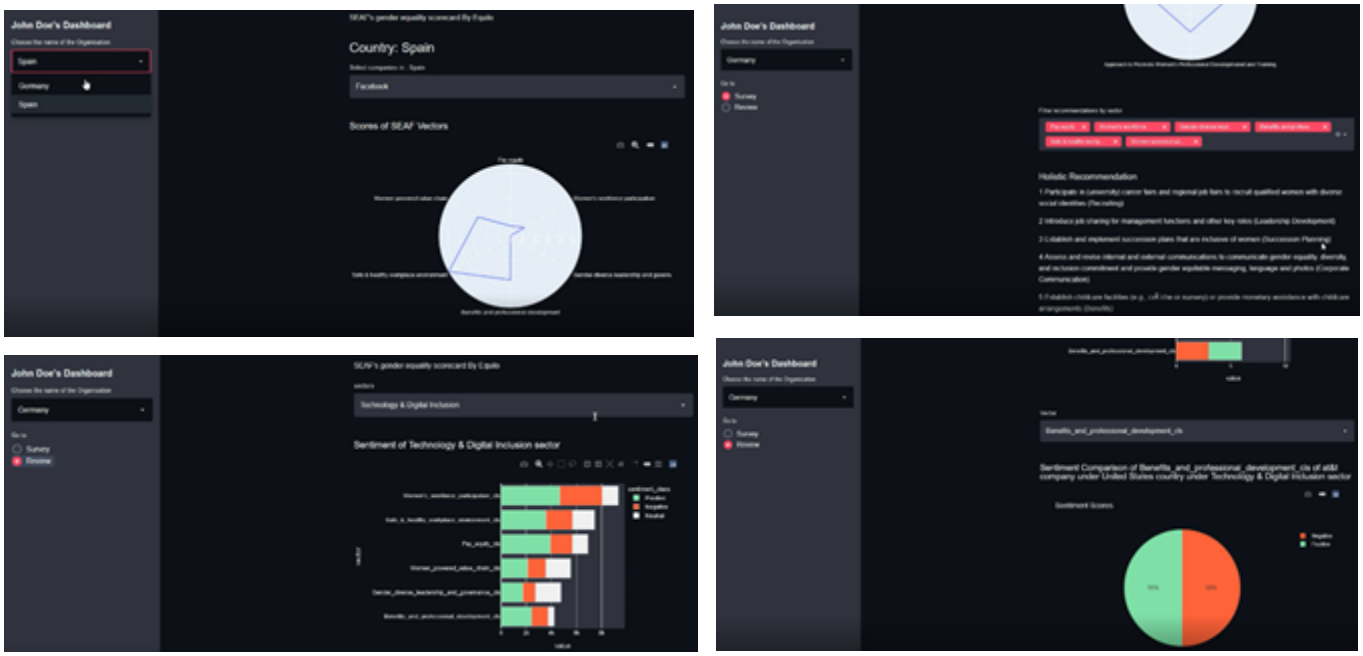
We used react Js for front-end, django for backend and streamlit for building the dashboard.

4.1. Dashboard

Rapid prototyping is an essential part of data science projects, and if that project includes interactive dashboard functionality, then there is no better framework than streamlit. It is a low code dashboard development framework made for data scientists and one can create a beautiful dashboard without the use of HTML and CSS. Let's deep dive into it to understand its functionality.

The project prototype needed multiple pages in the dashboard. Pages will show data based on the selected country of the user. So, we have given the selection box in the sidebar itself. Also on every page, there were multiple selection options that users can choose to get the charts. This interactivity is created very easily using these three features ‘selectbox’, ‘slider’, and ‘multiselect’. We have created a two-pager dashboard. One for review and another for the survey. On the survey page, we are using the APIs that we have created in Django using the response we are creating different spider charts and text recommendations. For the review component, we are showing the data that we have extracted. The data consist of reviews of different companies in different countries with different industries. We have made a different selection box to drill down the data to get a concentric as well as a holistic view.

We have made independent functions for each chart and are getting data from different APIs. All the charts used here are made with the 'Plotly' library. Besides ‘Plotly’ , Streamlit can handle other visualization libraries too. For each independent function, we used ‘st. cache’. It will make the function faster in loading as the cached function is only being executed once, and every time after that it is hitting the cache. We removed the ‘made by streamlit' at the end of the page and customized it as much as possible. The code is written in such a way so that in the future one can add other pages also alongside the above-mentioned two pages



4.2. FrontEnd (React JS)

Unlike other projects where having a user interface might not come as essential at the proof of concept phase, this project was different. The team already was working hard on creating all the different APIs to handle the survey questions and the reviews and we thought that we needed a basic UI to demonstrate and display the work that had been done across the previous weeks.
So we went with React JS. One small issue here, we had no one in the team who had worked with React before. Here I'd like to point out another important quality in working at an Omdena project, which is the educational aspect. It is a priceless opportunity to jump in and learn about new things that you may be interested in, and that was the case here.

We had the choice of leaving it as it is after all we had a good reason for that, but we made the decision to make the extra effort and do our best to deliver it in the best form possible. Two weeks was the period left to learn React JS and build multiple UIs that interact with our backends. And again we delivered.

It was teamwork again that made this possible as we kept the communication channels open all the time, during the testing of different UIs and making sure it was compatible with the backend.

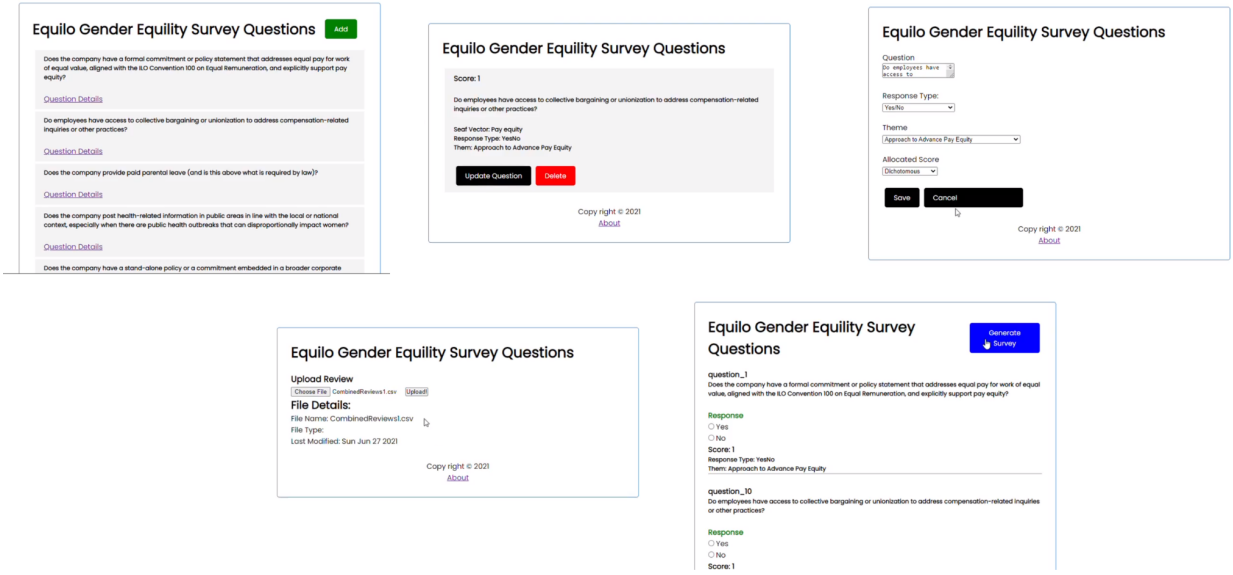


Fig. The UIs that was developed with React JS

We had 5 main interfaces that needed to be developed and integrated with the backend.

- A page with all the survey questions
- Forms to display a question details with options to delete or edit
- Add a new question form.
- The actual survey form where the users would request a unique survey and post the response to the server.
- In addition to the previous we had another important UI to upload the reviews that would later be used for sentiment analysis after going through the data processing pipeline to extract the valuable ones.

Needless to say we faced some unexpected issues, specially at testing time one of them the infamous CORS error when communicating between URLs with different origins, but was also able to cross that bridge. Again, if you are interested in knowing more about the technical details, stay tuned for a future dedicated post for that also!

But overall working with React was a great experience, and it has such a straightforward and intuitive structure that made it easier to learn, especially if you already have knowledge in HTML and JavaScript.

And that was it. We had all the components ready and we could showcase our work properly and with confidence!

4.3. BackEnd (Django)

All the machine learning and data pipeline is integrated in a backend built with Django. Django is a powerful Python-based web framework for seamless integration and deployment. Our Django App has 4 types of URLs. 2 of them are built-in for Django: admin (for user authentication) and api-auth (for third-party authentication). Other 2 were defined by us: survey(to integrate and build APIs for all survey operations) and review(to integrate and build APIs for all review operations). All our survey data is stored in a firebase NoSQL database.

Survey and Review URLs:
We built 21 APIs to handle survey and review operations:

Only Admin Access:
a) Add a new question to the database.
b) Add a new vector to the database.
c) Add a new recommendation to the database.
d) Add a new theme to the database.
e) Add a new parameter to the database.
f) Get all organizations and countries from the survey responses stored in the database.

g) Get all vectors and themes from the database
h) Get survey responses of a specified user
i) Get a specified recommendation
j) Get a specified parameter
k) Get the country of a specific user
l) Update any question details if needed
m) Update any recommendation details if needed
m) Update any vector details if needed
o) Update any theme details if needed
p) Update any parameter details if need
q) Uploading a file containing annotated reviews and adding to the Google Cloud Store

Public Access:
r) Any user can send a request to take a survey for their company.
s) Provided a survey form any user can fill it up and check the scores. This response is stored to the database automatically.
t) Get all questions from the database
u) Get specific question detail from the database

Firebase Database:

Firestore is a popular NoSQL database that has two components which are documents and collection. Because of the intricate format of reading and writing data in this database, we have created this database in a very particular way to optimize costing. We have created a separate collection for each of the components given by the client. The collections are:

- counter
- parameter
- recommendation
- SEAF_theme
- SEAF_vector
- survey_question
- user

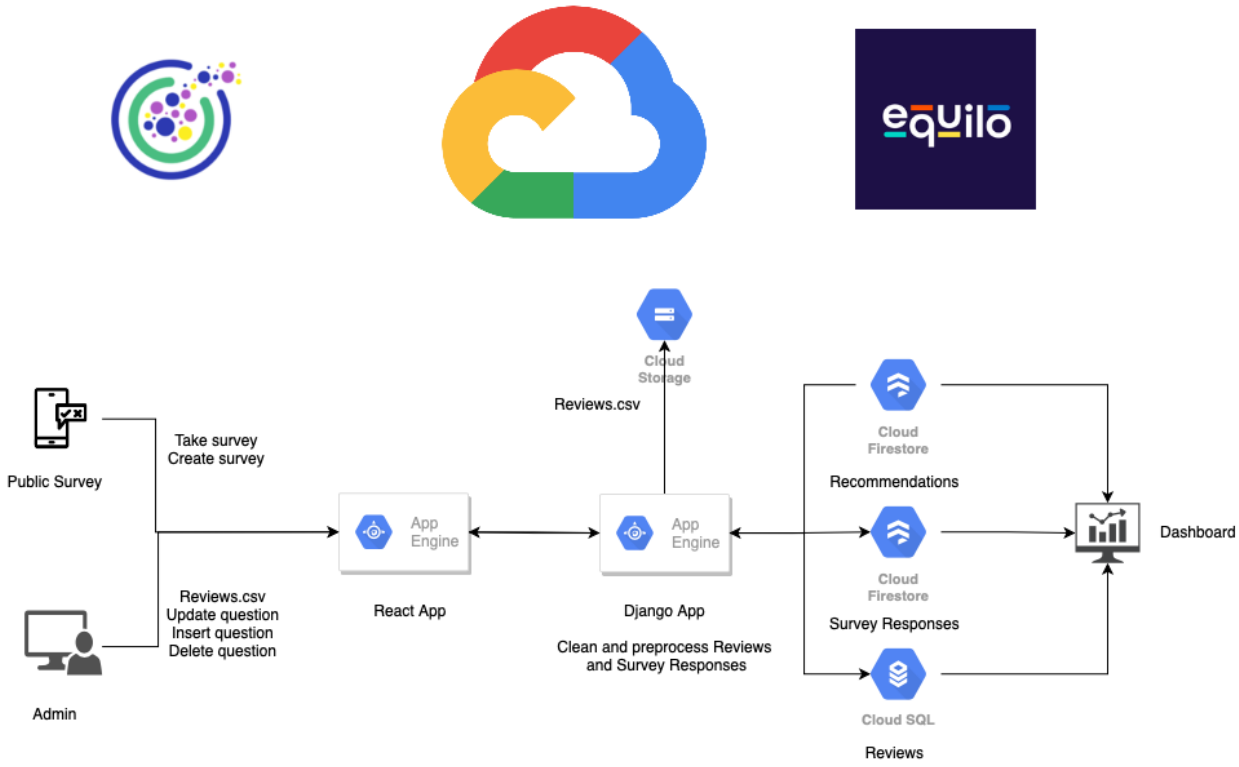
The user collection will contain the user documents. In the User Document, there is a subcollection of the organization. In organization subcollection, the survey responses will be stored which are sent by the user to different companies which will take the survey.

The counter collection is for storing the count of different entities like the question. Theme etc. It will update every time the client will use the Add APIs. For each entity, there are add APIs and update APIs. As the collection are interconnected, one API can update multiple collections. Each update is not restricted to one entity but several entities. Like, update theme API will update details in theme, vector, question collections.

We have two python scripts also for updating the data except for those above-mentioned APIs. One is created to allocate a score for each question. It will run every time the client will add a new question. The second one is for counting the allocated score and determining the holistic recommendation for each survey response.

5. The General Workflow

The entire solution was deployed on Google Cloud Platform(GCP) and firebase. Below is the overall architectural diagram of the solution. Let's go over each component of the solution and their role.



1. Google cloud storage(GCS)
   Google Cloud Storage is a service for storing objects in Google Cloud. An object is an immutable piece of data consisting of a file of any format. In our case the file was a csv file containing reviews. You store objects in containers called buckets. All buckets are associated with a project, and you can group your projects under an organization.
   In essence, GCS was used as a data lake. All the reviews coming in from different sources are first stationed in GCS before further processing can be done.

2. App Engine

   App Engine is a Paas(Platform as a service) offered by Google Cloud Platform. We have hosted our Django and React App on App Engine.
3. Firestore

   Firestore is firebase's latest database for mobile deployment. It builds on the successes of the Realtime Database with a new, more intuitive data model. We used firestore to store the recommendations, survey questions, survey responses, and user data.
4. Cloud SQL

   Cloud SQL is a relational database service offered by Google Cloud Platform. We used it to store processed reviews. Reviews would have fields like review text, date posted, source of review, company name, location, along with scores across different vectors.
5. Dashboard

   Dashboard is the place where a user can see different reviews taken by other users, their recommendations, along with different plots to better understand the organisation gender equality metrics. A few snapshots of the dashboard can be found in the dashboard section above.