Premise

Before you start the test fill out this form authorizing us to reach out to you.

Expected total time: >30 hours

Deadline: November 21st, 7:00pm Colombia Time. NO EXCEPTIONS.

For this part of the application, you will need to get creative! This is an open ended challenge, with a few constraints, and some required features. You will be assigned a read-only public data set in Google Cloud BigQuery. Typically these data sets contain multiple tables, each with 20 to 50 million rows. The exact features and purpose of this app are up to you, but they will likely be determined by the data set you will be working with. We require certain features in order to give the challenge structure, ensure that certain necessary skills are being demonstrated, and create a standard objective baseline to judge and compare applicants. Beyond those features (listed below), unlike the first challenge, where you want to take this is up to you!

Requirements

Your application will be composed of a minimum of these components:

- The assigned BigQuery read-only database.
- A graphical web frontend written in Javascript or Typescript.
    - Use either Vue or a React framework.
- A backend written in Python or Java.
    - If you are using Python, you should use either Flask or Django.
    - If you are using Java, you should use either Spring Boot or Dropwizard.
- Your own read-write database. This can be either by MySQL or PostgreSQL. This is where you will store all state specific to your application.
- A Docker compose file that runs the application.
    - It should run a minimum of 3 containers: your frontend, your backend, and a database.

We are requiring specific programming languages, frameworks, and databases for consistency, and because they are heavily used in the Cloud Computing industry. Your submission will be evaluated against a rubric that considers code maturity and industry best practices. This project is structured to highlight your ability to produce great professional-grade software.

This table summarizes choices of languages, databases, and frameworks. Please keep in mind that you are not limited to these– you may use other libraries and frameworks such as SQLAlchemy (an ORM), or templating engines like mustache.

Database	**Backend Language**	**Frontend Framework**
PostgreSQL	Python with one of:	React (Next)

MySQL	Flask Django FastAPI	Angular Vue
	Java with one of: Dropwizard Spring Boot	

User Stories

While much of what the app does is up to you to decide, it must support the below features. These features are described as *user stories.* A user story is a short description of a task or goal the user wants to accomplish, and why they would want to do that.

The story does not describe specifics about how the user accomplishes this goal, or what the user interface is, **unless** it is critical to the goal. In the world of professional enterprise software development, feature specifications are first written as user stories. The details of how the user interface works, and often specifics about how the user accomplishes the goal, are not given. That way the *what and why* are separated from the *how*. This gives developers and designers more freedom to come up with creative solutions to help users accomplish their goals. Isolating the why and the what from the how also makes it easier to see what is a bug or not, because the intention of the feature is written out very explicitly, with no added information. In this case, the how must be determined by you! You should read this short description of how user stories are part of the development process here.

Note that while these user stories are a minimum of the features you should implement, the main focus of the application is still up to you. You can implement additional features, or design these features in such a way that the user is guided to some other kind of goal. At the end of the day, the user might never know there was ever a story called 'visual query builder!' That is a background detail, just like we don't know all of the names of the stories that went into designing the Windows start menu, or the Github repository view.

Visual Query Builder

As a user, I want to be able to build a query without writing any SQL. I should be able to use a combination of checkboxes (for boolean fields), drop down menus, and textboxes in order to specify a query for the data. The exact graphical elements will be determined by what the data set is about.

For instance, on Mercado Libre, you can filter by different price ranges, the used or new condition of objects, and the shipping speed using sliders and check boxes. These are *UI filters* that can help bound a set of search results.

Google also supports boolean logic (although it's really just for super users!) and you can read more about it here. These two references are just to give you an idea of what's possible when it comes to web interfaces and search!

It is not necessary to represent every field or table in the query builder. In fact, that would make the query builder difficult to use! Choose a single use case and limit yourself to six fields or less.

It may make sense for this query builder to perform joins across multiple tables. However, I do not want to be aware of this detail. The purpose of the visual query builder is to be an intuitive interface and hide unnecessary implementation details from the user.

I want to be able to press a button marked "Run Query" to fetch the data based on my graphical input. The query should only be run when I press the button.

Visual Summary of Queried Data

As a user, I want to see a graphical representation after I run a query, so I can quickly understand the data.

This visualization can be any picture, like a line graph, bar graph, pie chart, or something else. It may make sense for the visualization to be multiple pictures.

Be creative! Make it pretty!

Save Query With Name, Username, and Comment

As a user, I want to be able to save a query created in the visual query builder, so that I can come back to it later, or so that other people can use it.

I want to be able to give the query a name, and a comment. When it is saved, my username should be recorded.

Note: You do not need to build login and password functionality. However, the user should be able to specify their username somewhere while using the app.

Show All Saved Queries

As a user, I want to be able to see a list of all of the saved queries. This list should be able with the columns: name, comment, and the username that created it.

Comment on Query

As a user, I want to be able to comment on other user's queries, so that I can collaborate on data exploration. The comment should appear with my username below the description of the query in the list of all saved queries.

Select Saved Query

As a user, I should be able to select a saved query from the list of saved queries, so I can come back to my work later. When I select a query, it should be loaded into the visual query builder.

Persistence

As a system administrator, I want to be able to shut down the application and boot it back up again with all data, such as saved queries, restored. My system should be able to suffer a power outage with no loss of data.

Multiplayer Functionality

As a system owner, I want two or more separate users, with two separate usernames, to be able to use the app at the same time, so I can support more data exploration.

Rubric

You will be graded on three main areas. Each area will be given a grade between 1 to 4, with 1 being the lowest. We chose 4 points because we did not want any 'neutral' grades for any areas.

Note that it is entirely possible to get a 4 in one area, and a 1 in a different area.

Feature Completeness

Whether the user stories were comprehensively implemented and work as intended.

1	Many user stories are not implemented. Or most features have critical bugs that do not allow the user story to be executed.
2	User stories are implemented but there are major bugs that make the experience unpleasant.
3	User stories are implemented correctly and there are few or no major bugs.
4	Wow! User stories are implemented and fit together to create an intuitive user experience. The app has a professional look and feel. Additional user stories beyond the ones required may be implemented that make sense to help the user accomplish their intended goals, and fit within the theme of the app.

Data Pipeline Quality

Focus on understanding the data you're working with, data hygiene, and reproducibility.

1	Data source has clearly not been properly studied and processed. Data queries are written in raw SQL and non optimized and unorganized.
2	Some basic data quality checks are in place, and queries are minimally organized on the backend.
3	Signs of mature data pipeline quality practices are in place such as filtering for outliers, type checks, reproducible data pipelines, and ORM query management.
4	Wow! Are you a professional data engineer? Processing and filtering is done through mature ORM and ETL frameworks, queries are modular and type checked, all data CRUD operations are sanity checked, and error and processing messages are clear. Even data unit tests are in place.

Code Maturity

Focus on maintainability, coding best practices, and the ability for other people to collaborate on this code in the future.

1	• Code is not documented. • No unit or integration tests. • Code organization is poor. • Debugging information is leaked via print statements. • Security best practices are disregarded. Ex: protecting against SQL injection attacks.
2	• Some code comments. • Some unit or integration tests. • Code is separated into loose modules.
3	• Code is organized in packages by functionality. • Functions are kept short and have a single use. • Security best practices are followed. • Good documentation. • Good unit tests.
4	• Code is structured with class and function reuse in mind. • Excellent code documentation. • Excellent unit tests with nearly complete code coverage.

	• How another developer could contribute to the code is obvious through documentation or great modularity.

System Quality

Similar to code quality, with a focus on collaboration and maintainability. However this includes considerations around how the application interacts with the Cloud.

1	• Application requires troubleshooting to start up.
2	• Configuration is hardcoded in the application, instead of in an editable config file. • State is maintained in the app. • Database queries are inefficient.
3	• All state is in the database. • Configuration is changeable via a config file. • Database queries are reasonably efficient.
4	• Database queries are efficient. • The system has excellent logging. • The app handles errors gracefully, and presents an explanatory and helpful human readable message to the user. For example, if BigTable becomes unreachable, the user will know why the app is failing.

Submitting Your Application

Your submission must include these items:

- A Github repository where we can browse the code. The Github repository will also contain the following items in the root directory.
- A file called USERGUIDE.md. This will be a short user guide on how a person with no programming experience would use your app.
- The USERGUIDE.md should include a Youtube link to a video of you explaining something. More details below (Explainer Video).
- A docker compose file. After running docker-compose up, we should be able to point a web browser at http://localhost:9000 and see your application. That means that your application must be running on port 9000!

To submit, you must invite Github user @kakaner (Karen) and @lazyvalue (Scott) as collaborators to the repo, AND fill out this form before the deadline. Please note again, that there are no exceptions– we must receive these notifications (both as collaborators and the form) by the exact deadline!

Explainer Video [English!!!!!!!!!]

An important part of software engineering is teamwork! As software engineers, we need to explain complex concepts to other members of the team, and convince them that our ideas and solutions are correct. We will also be expected to mentor more junior engineers and teach software concepts to people with non-technical backgrounds. Successful software companies are made by many people, not all of them engineers. We may need to help sales people understand why a feature is complicated to build, or troubleshoot a problem with an important company client. Understanding who you are talking to is really important.

Submit a 3 to 5 minute long video of you explaining to a junior engineer how docker compose works. The video should record your screen as you run docker compose command and talk about what is happening. Describe what the lines in the docker compose file do. Bonus points if you can talk about how you used Github Actions or some other CI/CD!

Then describe the cost and user scaling properties of your application. How much will it cost to run for 10 users? How about for 100 or 10,000 or 10,000,000 users? How many queries can it save before you need to refactor it? What other changes will you need to make to the app when you reach 10,000 users? When you reach 10,000,000 users? When will you need extra infrastructure like load balancers?

Hints and Tips

Github allows for free package storage up to 500mb. You can use this to store your docker images.

A lot of people like d3.js for visualizing pretty graphs.

Public Dataset Assignments

Please find here your cedula and your dataset!

Dataset	Cédula
A: The World Bank's Education Dataset	1058963679 1000595857 1001250630 1000885201 1001635870 1234095444 1193471614 1000604256 1005897813

	1006464055 1006325694 1003845154 1045492052 1001044935 1001822184 1001781997 1091674562 1088236109 1017209111 1216724594 1001651994 1192805665 1020442684 1015417330 1005332572 1001168564 1045742921 1044602713 1031640279 1075676056 1001131543 1036681527 1140887650
B: Google Search Trends	1235044452 1002528294 1152472496 1007316123 1000403325 1003527653 1006107372 1109185879 1109662923 1000003072 1045025694 1001883069 1193224863 1140829793 1017270327 1006073728 1061793477 1116547319 1005755109 1034656098 1143404079 1113620642 1193090421 43605279 1004754681 1088341715 1010147614

	1005753821
C: US Forest Service Analysis	1016075322 1005640648 1052836096 1007479370 1000747677 1073535334 1006170256 1193029891 1116070867 1000782413 1115189286 1001943284 1032876368 1006121867 1037669179 1151966570 1022389029 1140873583 1085301509 1000646367 1090396866 1013613966 1014205722 1055479435
D: World Births Dataset	1007236176 1003408831 1000412691 1216726912 1001463871 1193378401 1006325694 1005865617 1003616362 1071171822 1096670253 1044606552 1025140377 1067836791 1125348235 1002545071 1006189569 1007961917 1001419175 1136889403 1097402869 1067961607 1090479997 77106311