# Backend Challenge

**About the Product:**

Collect is a data collection platform that is being used by customers in 50+ countries in over 250 organizations and has powered data collection for over 11 million responses. Its features include team management, multilingual forms, and offline data collection. Our customers use Collect to power their most critical activities — from governments delivering vaccines to small business owners managing their daily inventory, to a zoo monitoring a rare wildlife species.

**Problem Statement:**

The lifecycle of data collection via Collect does not end with the submission of a response. There is usually some post-submission business logic that Collect needs to support over time. Some real-life examples -

1. One of our clients wanted to search for slangs (in local language) for an answer to a text question on the basis of cities (which was the answer to a different MCQ question)

1. A market research agency wanted to validate responses coming in against a set of business rules (eg. monthly savings cannot be more than monthly income) and send the response back to the data collector to fix it when the rules generate a flag

1. A very common need for organizations is wanting all their data onto Google Sheets, wherein they could connect their CRM, and also generate graphs and charts offered by Sheets out of the box. In such cases, each response to the form becomes a row in the sheet, and questions in the form become columns.

1. A recent client partner wanted us to send an SMS to the customer whose details are collected in the response as soon as the ingestion was complete reliably. The content of the SMS consists of details of the customer, which were a part of the answers in the response. This customer was supposed to use this as a “receipt” for them having participated in the exercise.

**Further details**

We preempt that with time, more similar use cases will arise, with different “actions” being required once the response hits the primary store/database. We want to solve this problem in such a way that **each new use case can just be “plugged in”** and does not need an

overhaul on the backend. Imagine this as a whole ecosystem for integrations. We want to optimize for **latency and having a unified interface** acting as a middleman**.**

Design a sample schematic for how you would store forms (with questions) and responses (with answers) in the Collect data store. Forms, Questions, Responses and Answers each will have relevant metadata. Design and implement a solution for the Google Sheets use case and choose any one of the others to keep in mind how we want to solve this problem in a plug-n-play fashion. Make fair assumptions wherever necessary.

Eventual consistency is what the clients expect as an outcome of this feature, making sure no responses get missed in the journey. Do keep in mind that this solution **must be failsafe**, should eventually recover from circumstances like power/internet/service outages, and should **scale to cases like millions of responses** across hundreds of forms for an organization.

There are points for details on how would you benchmark, set up logs, monitor for system health, and alerts for when the system health is affected for both the cloud as well as bare-metal. **Read up on if there are limitations on the third party** ( Google sheets in this case ) too, a good solution keeps in mind that too.

The deliverable upon completion of the task is a zip file containing

- design specification
- a brief but comprehensive explanation for the various approaches you can think of to solve for this and why you went ahead with the approach that you did - i.e a pro/con analysis
- the codebase

It is advised to use a version control system for the codebase, but please avoid creating public repositories on Github/Bitbucket et al.

**Last bit of advice:**

> You could consider signing up for a Collect trial here and explore the product before you begin solving; it might help!