ATLANtis

- **DB Demand**: A value can be searched and indexed based on matching as well(we might have to find a word in a paragraph, or a word closest to a particular word).

Elasticsearch	**SQL Database**	**MongoDB**
Can't use foreign keys	Can use foreign keys	Can't use foreign keys
Multi-Match is present	Have to traverse all rows to look for the word in the column	Multi-Match is present
The ranking of results is present	Can't be ranked on basis of closeness	The ranking present in the new version

I went ahead with **Elasticsearch** as Elasticsearch is the best choice in the case when an application requires too many filters or search operations.

- Language Used: Golang (highly scalable language, easy to learn and a fast language).

- Error Tracing, Logs, track performance, Error alerts Sentry because of easy GoLang integration, complete context, and real-time updates.

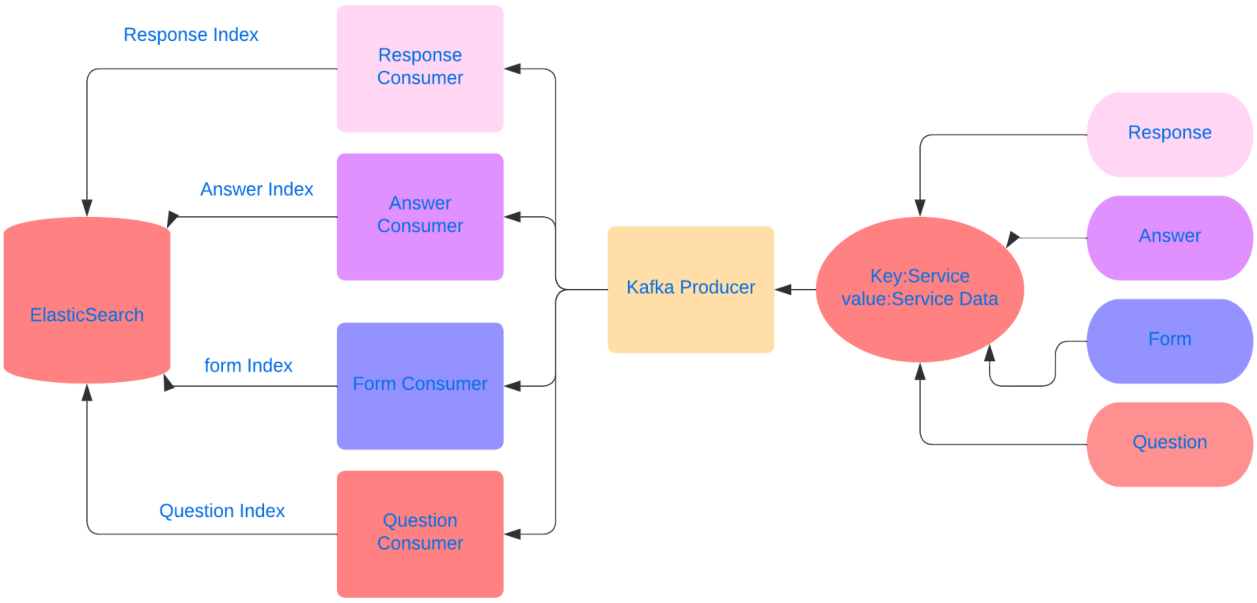
- What to use a Queue system or a non-queue system?
    1. Non-Queue System:
        - **Prods:** Can handle multiple requests on the server and things will have real-time, no waiting for any other microservice.
        - **Cons:** In case of failure the data will be lost forever, and more load on servers.
    2. Queue System:
        - **Pros:** No data will be lost, and no load on servers due even for thousands of data requests.
        - **Cons:** In case of consumer-producer lag delay in data insertion.

Thus I went ahead with Queue System as no data is to be lost.

- Queue Service Requirements:- Need multiple producers and consumers.

RabbitMQ	Kafka
Message Queue platform	Message broker platform
Push-based approach (meaning good for low latency)	Pull-based approach (good for long pooling of data).
10K data per second	1 Million data per second
Dumb consumer smart producer	Dumb producer smart consumer
Acknowledgement based	Policy-based (e.g., ten days)
Flexible routing, many clients, Federation	Durable, zero downtime, scalable, high volume,

Thus **Kafka outweighs RabbitMQ** as a queueing



How is data being stored?

Data is sent to the backend, where it's stored in a queue, from there consumers pick it and store it in the elastic queue. We predefine a unique ID for all the data and store it with the same in the database, the reason being this id can be used by other mechanisms.

Why not store it in DB and use the auto-generated unique ID?

As we will have to wait for the data to be uploaded and in case the connection is down we will have to create our id and send it back to the Frontend(we're using what we've done earlier as error handling here).
The former solution ensures that we're not dependent on any other service to perform the data upload successfully.

How am I storing forms in the Database?

In my consideration when a form is created, a request is sent to Server "/createForm" only **userId**(form creator ID), the **form name** is needed. A form is created and a **form ID** is sent back to the Frontend.

What is happening here? Instead of directly storing information in the database we're dumping it in the queue and sending a generated text (UUID) as form ID and assigning formed as UUID(so that response is sent fastly to the frontend and since we know the FormID it can be used with a question while they're being stored?).

```
{
  "userId": 6546,
  "formName":"form1"
}
```

But why not store form data in Elasticsearch and queue the data in case of a connection issue?

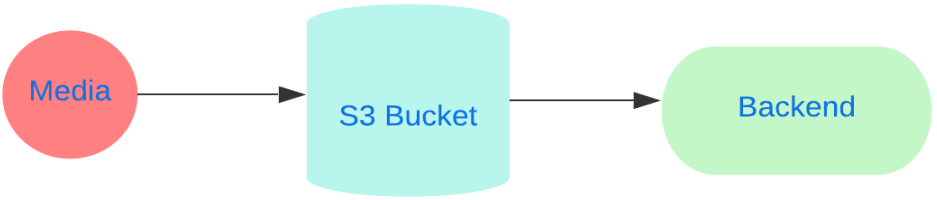
Well this is another good solution and in case a system goes down, the data will be dumped in the queue and used later on but in the case of thousands of data it might bring load on servers, and failure rates might hike.

How do store questions?

Since we have the FormID we can easily store questions **individually** as they can be traced back to the form they belong to.
For storing a form we need the following information :-

Question(the text),**Form**(FormID) ,**ResponseType**(what we expect in answer? Multiple choice,image,video or etc),**Order**(the serial number of the question),**Option**(optional applicable for multiple and single correct answers) ,**MediaAndContact**(optional for audio,video and etc),**Feedback**(optional, we will take the percentage from frontend).

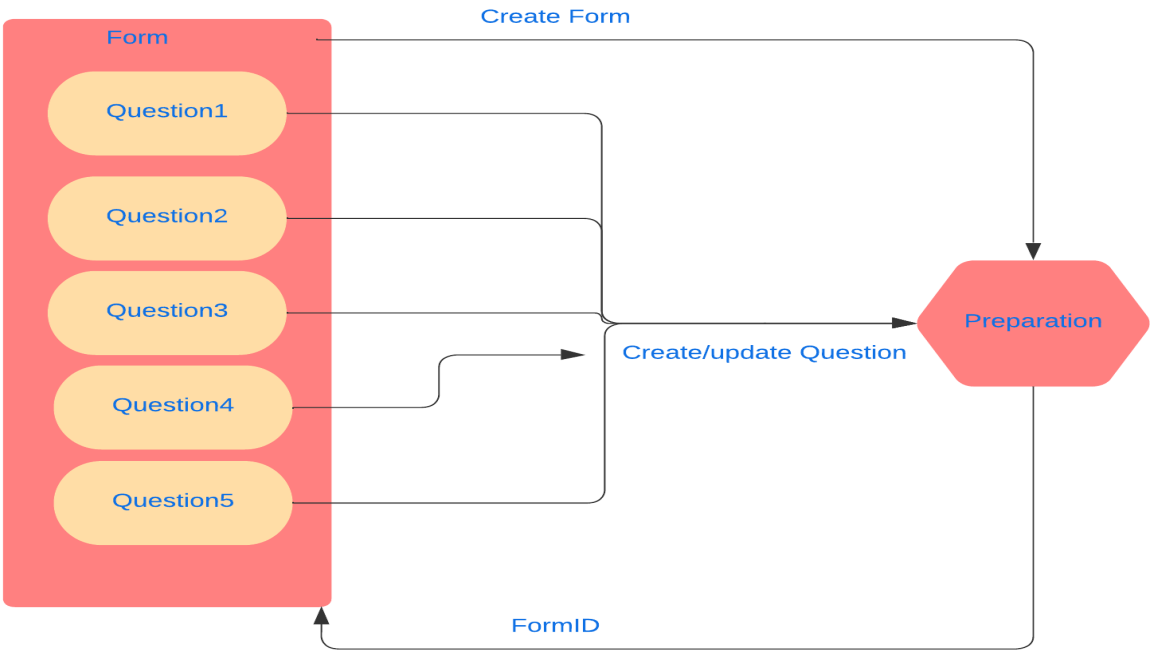
Point to be noted ~ in the case of **MediaAndContact** we expect a string (media should be uploaded and the **CDN link** to be sent to Backend to **reduce the load** on the server of uploading it and speeding the overall backend process)



Since the user can **add additional parameters in the question or remove any question**, I've **not asked for the status** of the form(meaning it's submitted or not, this would mean it's autosaved and the question is added anytime and updated too).

Since the questions can be updated at any instant we just need to provide the question id in the body along with changes and it will be updated.

```
{
    "question": "how you doing?",
    "form": "d7d72ecf812",
    "orderNumber": 3,
    "responseType": "text"
}
```



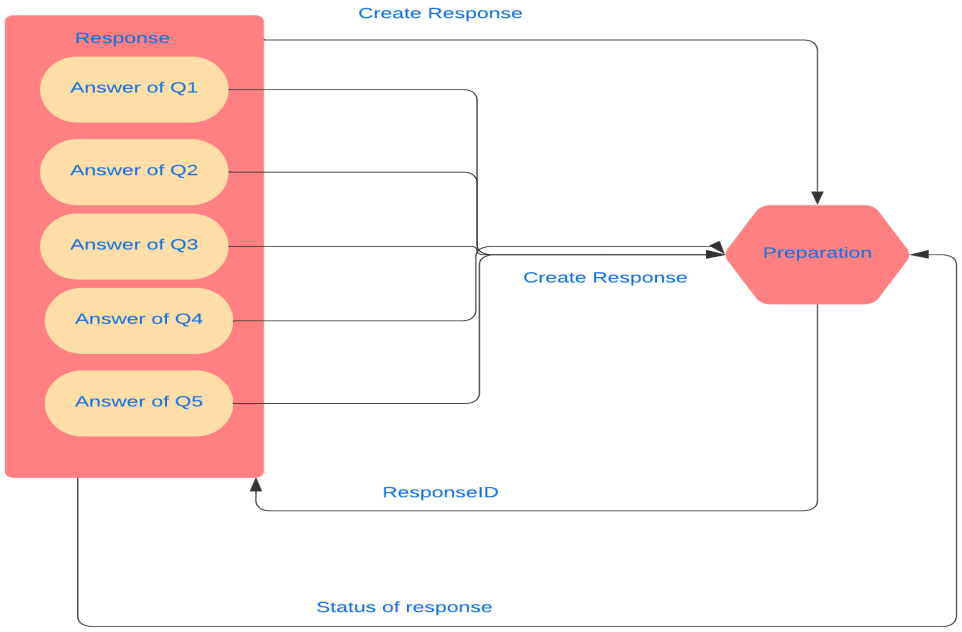
How am I storing responses in the Database?

In my consideration when a response is created, a request is sent to Server "/createResponse" only **userId**(form filler ID), **FormID** to know which form it belongs to, and **status** to know if the response is submitted. A response is created and a **response ID** is sent back to the Frontend.

How do store answers?

Just like questions, we can store answers as we're getting response ID to map the answer to their response. Since answers can be multi-correct or any other param, I'm expecting them to come in an array. This will work for any type of response as a string is expected

```
{
    "answer": [
        "I'm the only option here"
    ],
    "answerType": "single correct",
    "questionId": "955cc7e7520",
    "formId": "b7cb38eec3e",
    "responseId": "de3a24acb9e"
}
```



How to fetch form?

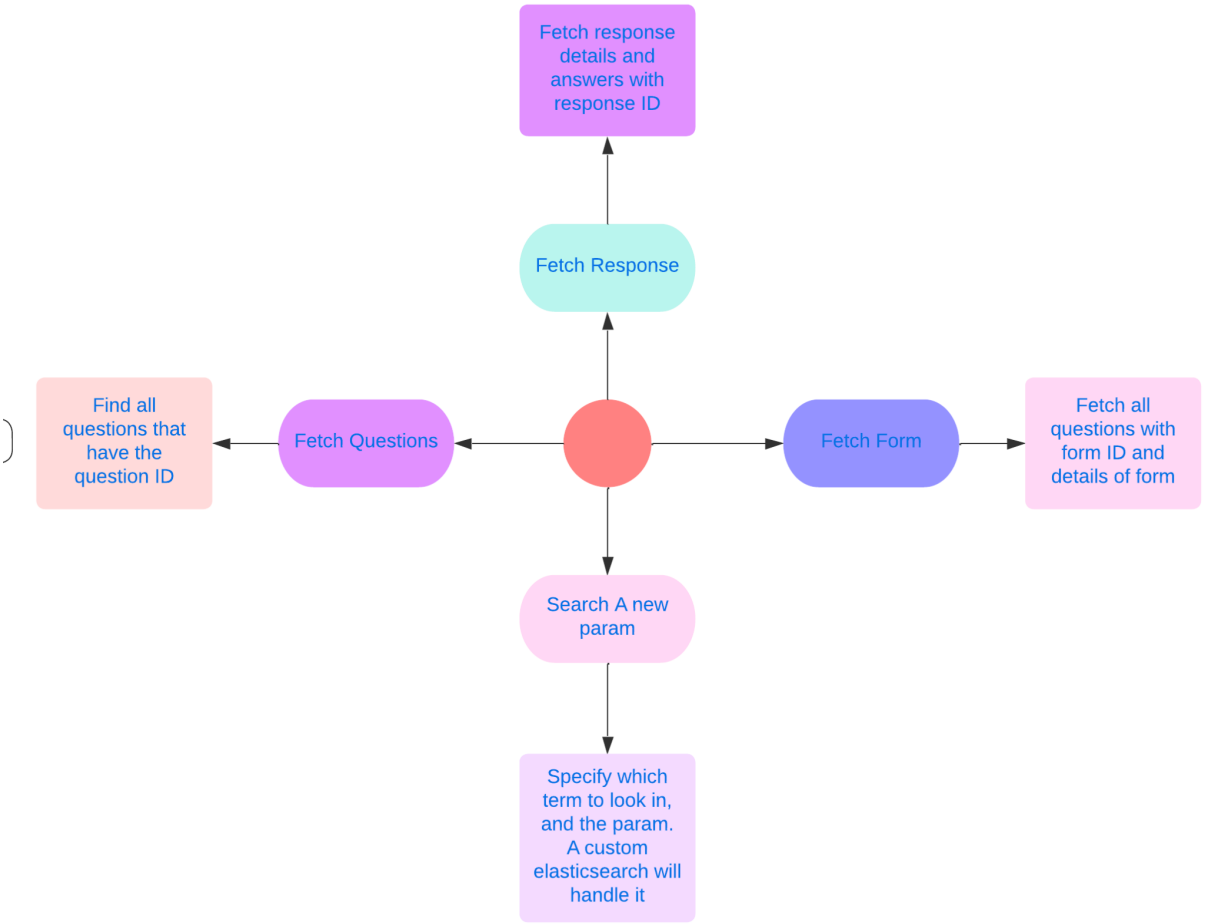
Just the FormID is needed, and questions in the form will be fetched as questions store the FormID as param.

How to fetch a response?

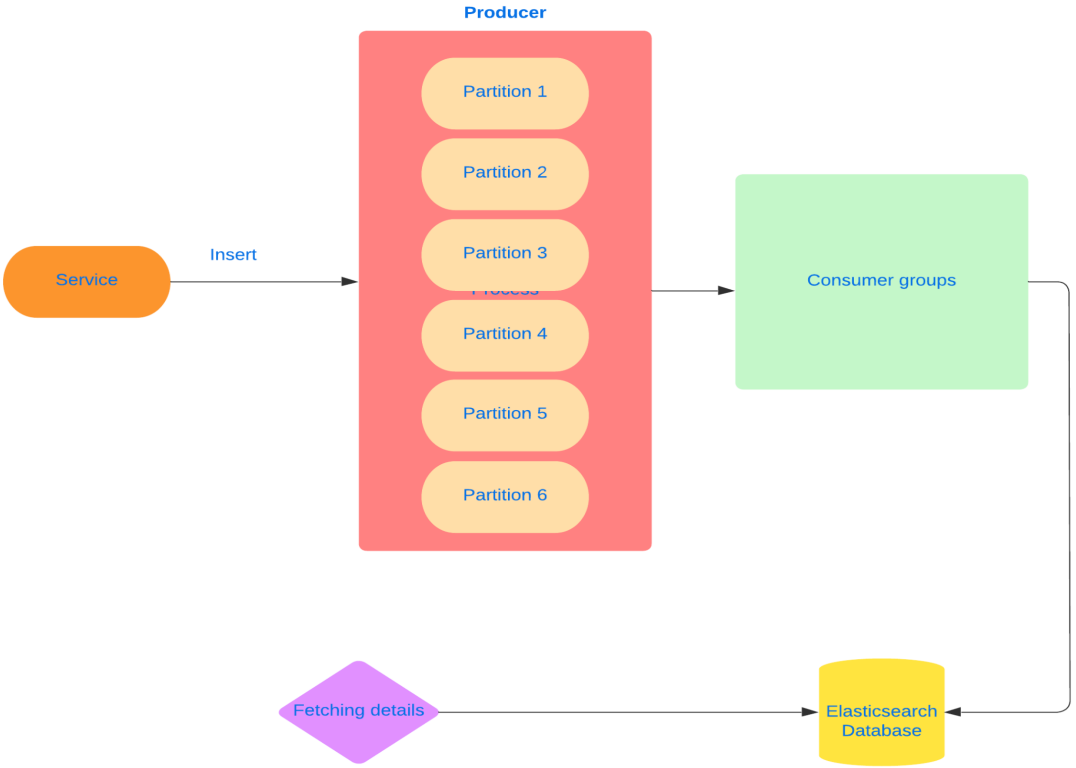
A response ID is fetched, corresponding to that all the answers are fetched and corresponding to those answers, questions are mapped to them and sent to Frontend.

How to fetch a particular value?

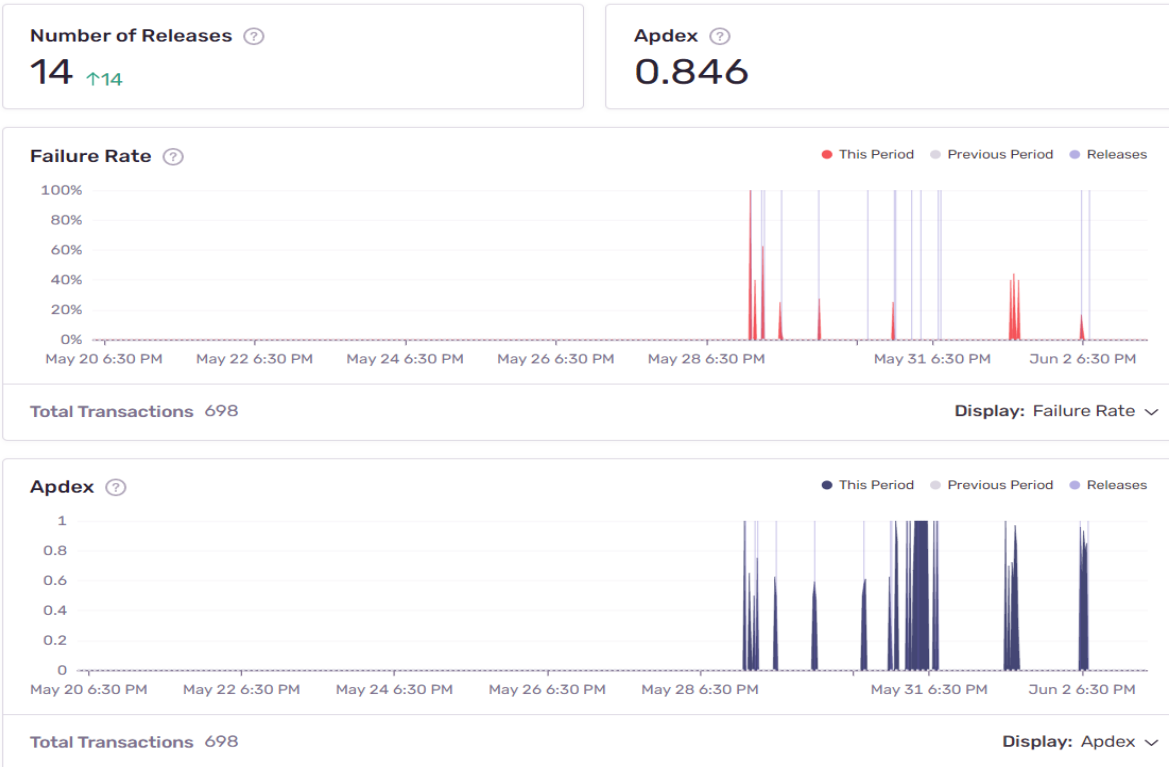
The parameter that is to be searched for the value and the term to be passed, the multi_match function can help fix this problem, we also get the freedom to choose which **index**(table equivalent in ElasticSearch is to be searched).



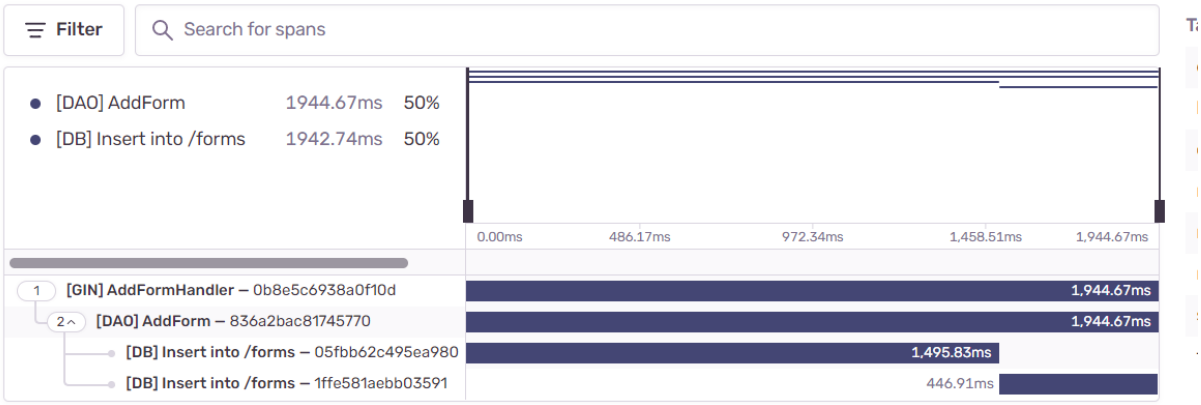
The overall schema (connecting database to queue to service(form, question, response, answers**))**



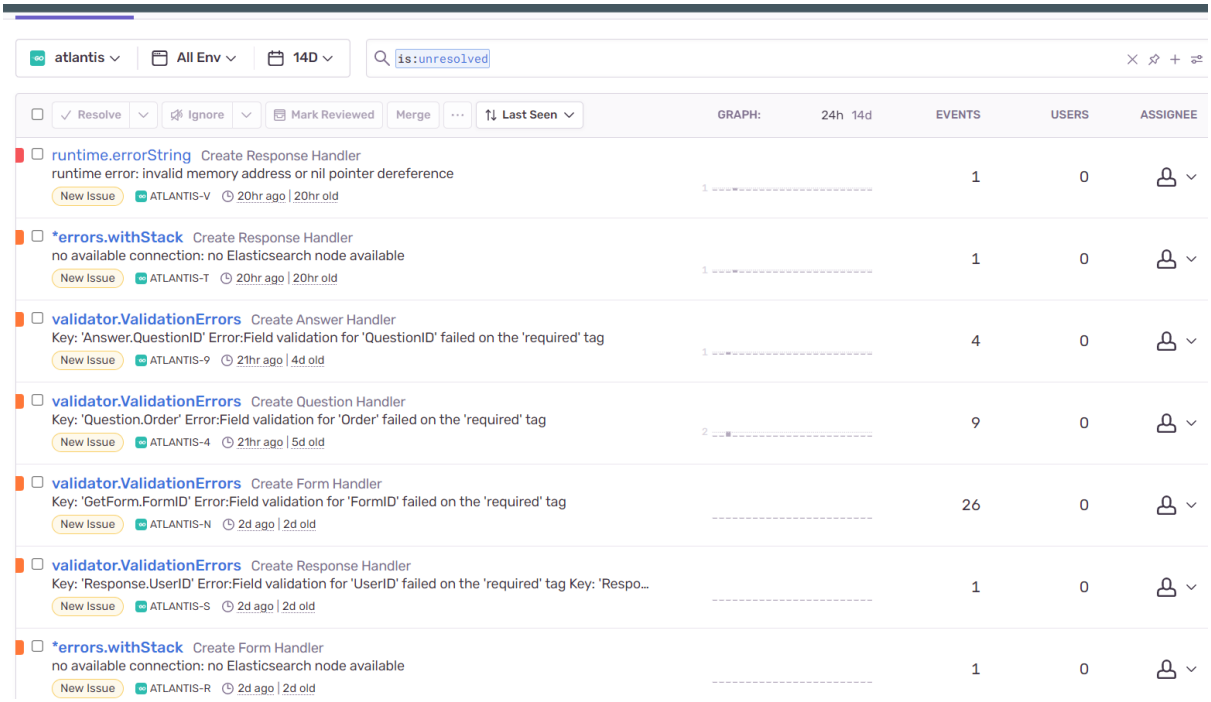
Sentry homepage



Sample Transaction



Error Handling



Transaction Sentry