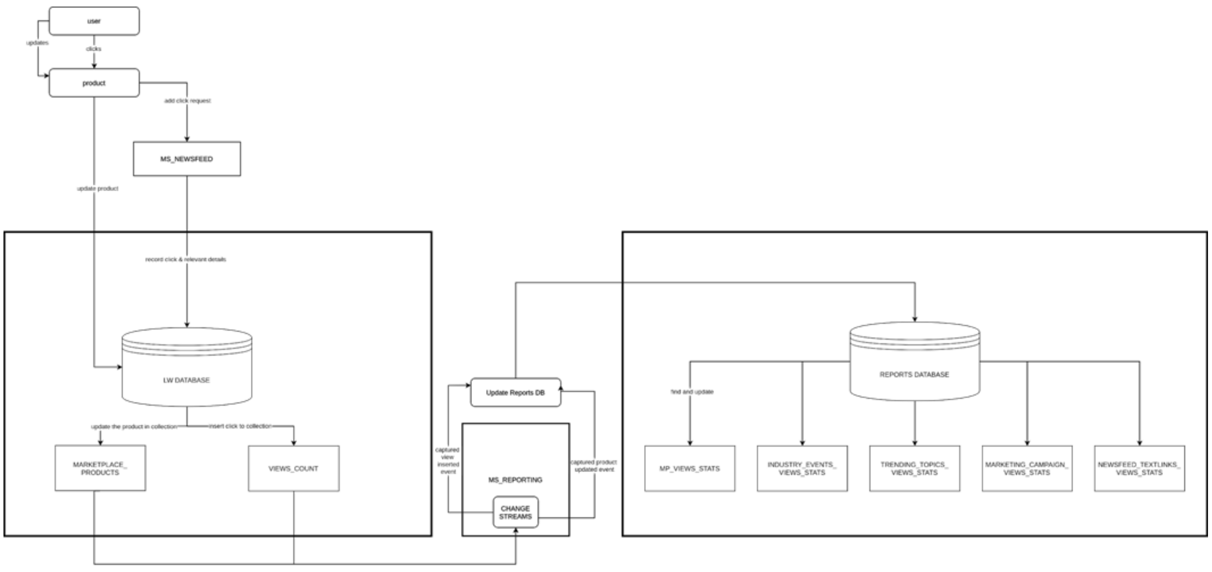
What's it about?
This case study is about the challenging yet interesting problem that I solved for Leafwire. The project is a subset of work I've done for the leafwire project. I was given the responsibility to handle the analytical section of Leafwire. Leafwire monitors user interactions to provide the users a better experience of using the application and to produce reports. All the analytics data was stored in one Mongo DB collection, which initially fits the purpose of producing reports and understanding usage of users.

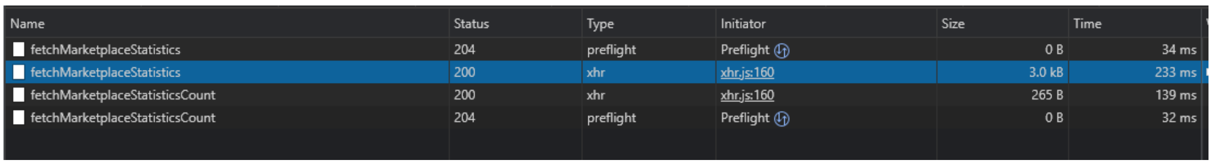
Problem Statement
As the user base starts growing the analytics data starts to grow exponentially, I noticed if this growth continues, it would impact the performance greatly, up to an extent where while producing reports would consume the 100% CPU usage and make the app unusable for the users. I had to develop a solution to the problem to overcome the possible downsides in the near future. The application was taking around 25 seconds to produce 1 report.

Solution
I created a microservice based architecture based solution integrating leafwire API to other microservices.



Microservice 1: Data Processor API
I designed this microservice, in a way using Mongo Change Stream, so when a db operation takes place in the original db, it will trigger an event automatically. So whenever a record is created in the large analytics collection, it will trigger an event. This microservice would process the data and add them into separate tables. It will add the combined count of analytics data per day, and every day would have one record of the total analytics data.

Microservice 2: Reporting API
I designed this microservice, to produce reports from the processed data and only this api can access the reports data collection that's separated by the data processor.This reporting API is independently responsible for the reporting so the processing time of reports would not impact the other parts of the application. By combining these two microservices along with the leafwire API, we were able to solve the issue and my solution is still handling data that grows at the rate of millions per week and the average processing time for reports is just around 0.2 seconds. So with my solution I was able to reduce the processing time by a huge margin(25 seconds to 0.2 seconds). In the below screenshot you can see the response time for the generated report (233ms).



Name	Status	Type	Initiator	Size	Time
fetchMarketplaceStatistics	204	preflight	Preflight	0 B	34 ms
fetchMarketplaceStatistics	200	xhr	xhr.js:160	3.0 kB	233 ms
fetchMarketplaceStatisticsCount	200	xhr	xhr.js:160	265 B	139 ms
fetchMarketplaceStatisticsCount	204	preflight	Preflight	0 B	32 ms