Functional requirements:
Registered users can create campaigns to fund their cause.
Registered users can collect money against a campaign
Registered users can view a campaign by campaign ID
Registered users could contribute to a campaign based on campaign ID
Once a day we flush the collected money into the creator's account

Out of scope:
User registration
Analytics
Recommendation
Search
Guest flow

Non-functional requirements
Low latency
Durability changes made to the campaigns or newly created campaign should not be lost
Eventual consistency
It is a growing system

System APIs:
createCampaign(userId, String campaignName, CampaignDetails campaignDetails) → returns campaignId of the created campaign

getCampaignDetails(userId, campaignId) → Returns CamapignDetails of the given camapignId

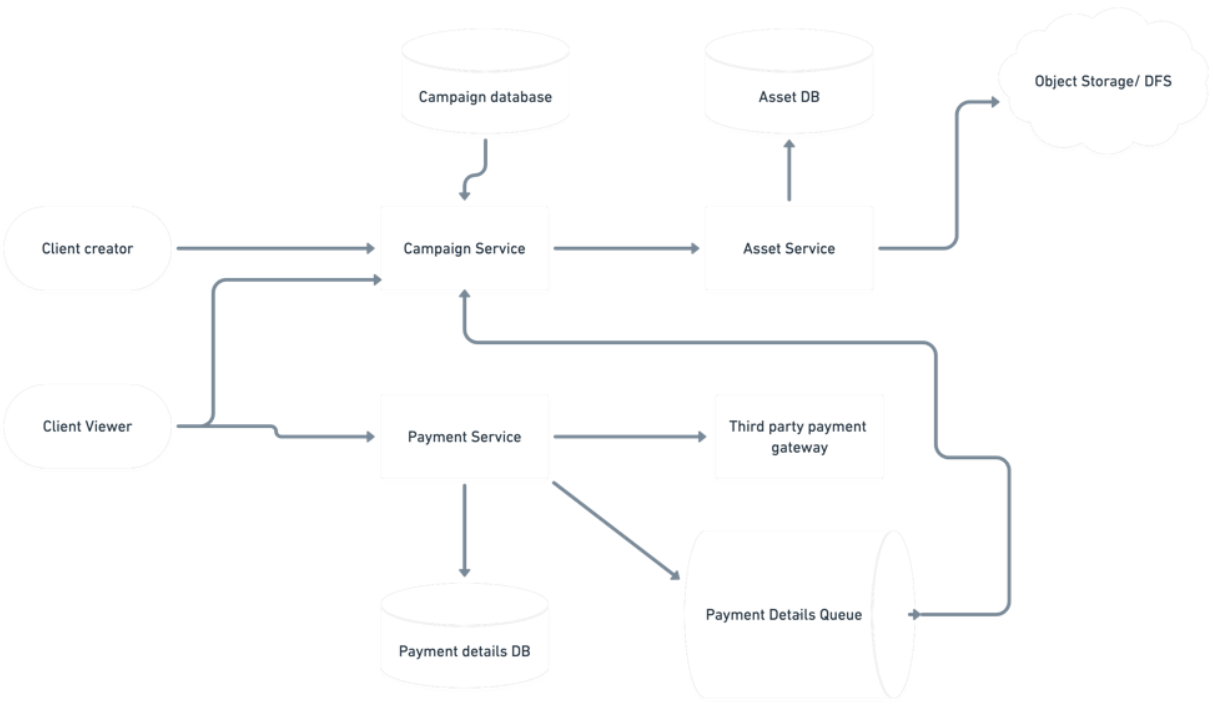
pay(userId, CampaignId, PaymentDetails paymentDetails) → return a boolean if the payment was successful or not also updates the total collected amount against the campaign

Capacity Estimation:
Let's say we have 50 million users with 50K users added daily
Our DAU is 10% of our total users ~ 5 million
A user on an average creates 5 campaigns Total of 50 * 5 ~ 250 million campaigns

On an average 5 million people view campaigns on a daily basis so 5 million/24 ~ 200K per hour/60 ~ 3K per min ~ 500 per sec

So as we can see the load on our system is not that much

Basic Design:



Write Path:

A client creator logs in and is redirected to the Campaign service he enters the campaign details and the campaign service stres these details in the local database and also uploads any assets like pictures, videos to the asset service and the asset service returns the id for the assets.

On the asset service side, the assets are stored in the Object storage/DFS and the mapping of their location is stored in the Asset service DB.

Read path :

Client hits the Campaign service with the Campaign ID and is show the campaign details. Client can further more move to the payments page with the campaign ID the payments service will collect the payment through a third party payment Gateway like PayPal and then return the details. The payment service could also push these details into a payment queue which can have multiple consumers like User payment service which will be responsible for processing the campaign payments for the related user or the campaign service which would be responsible for updating the details to a payment for a campaign.

If sequencing needs to be maintained, we could use multiple queue appropriately.

What our basic design is missing?

Load balancer - which is entry point to our website
Gate way -- which would be responsible for filtering requests and authentication and authorization

Service discovery --- Would be responsible for providing service end points to internal services and also load balancing internally

Caches.

What data can be cached and what will be the optimal cache strategy?

We can have a cache in front of our Campaign service and we can have a cache in front of our asset service or the object storage.

The campaign service cache would cache the campaign details and asset service cache will cache the assets recently used.

What will be the optimal cache eviction policy and why?

We could use an LRU+LFU strategy for cache eviction LRU best suits our need as campaigns are more likely to be accessed in the first few days. And usually campaigns also have a end date.

Incorporating the above changes. Our detailed design would be as follows:

Detailed design: