Functional and non-functional requirements:

To start, we will outline the use cases and the expected functionalities of the system. We will limit the functionalities to the following scenarios:

- Collect page view data. This functionality tracks the website visits by the users, and the collected data will be used to provide analytics information
- Present the data in a well-formatted way to the customers/merchants. The data collected in the previous step is used to provide analytics to the merchants. This analytics information will be presented using different chart and diagrams through a user interface.
- Service calculates the analytics of pages and monthly visit status

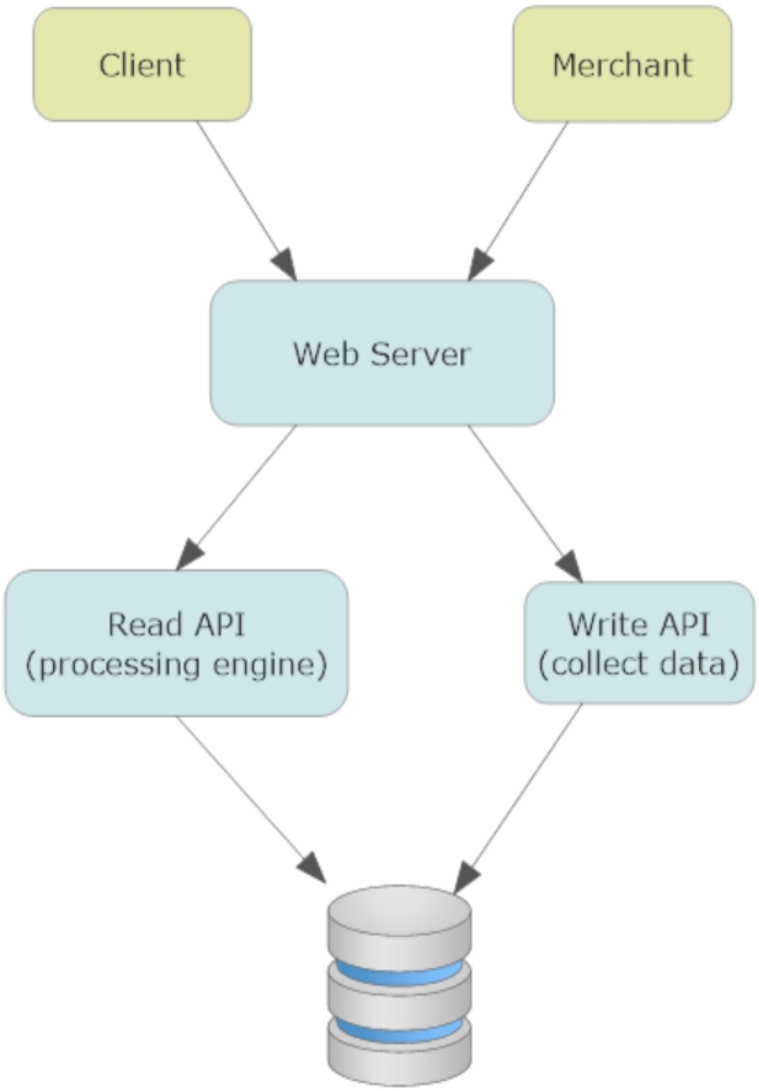
In addition to the above functional requirements, we need to consider the following non-functional requirements to design the system:

- System should be designed in a way that it should be durable and likely to operate continuously without failure for a long time (high availability)
- System should be able to handle a very large number of users at the same time, and it should also prevent overloading resources (scalable design)

In the next pages we will start with a high level design of the system, and then provide the detailed design to handle scalability and high availability.

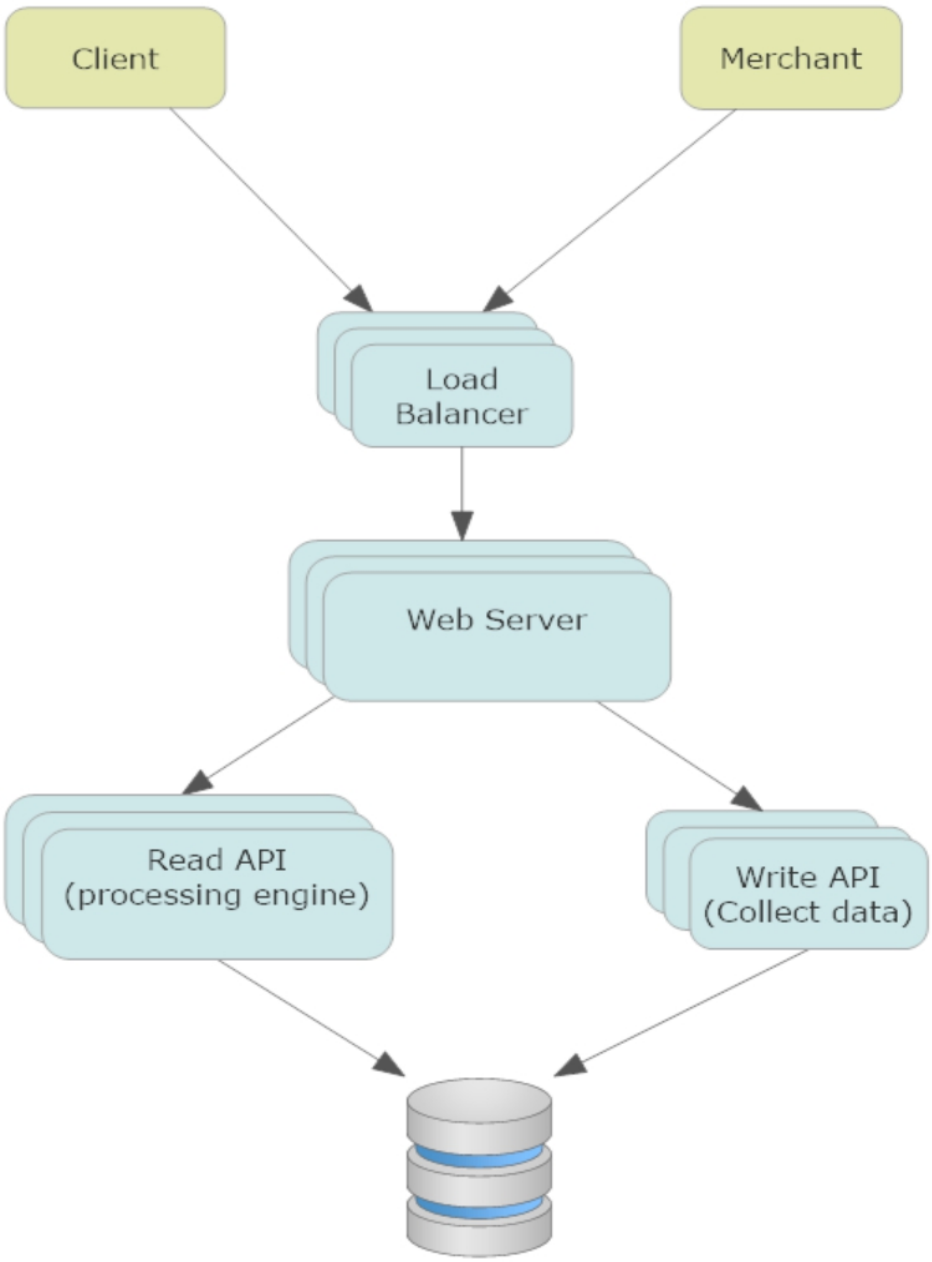
High level design:

Web pages tracked by Google Analytics contains a tracking code or script that sends an event with every interaction (page load in our case) from the web page to the Google Analytics web server. The events generated from user is stored in the analytics database and scheduled for aggregation processing. The data collected from user events are used by a processing engine in order to provide analytics to the merchants. The processed data is passed to the presentation layer in order to provide different charts to the merchants.
Also third party APIs are provided in order to integrate with other systems. In this case, the client is not an end user, it is a another system which is going to be integrated with our analytics server and use the data and process it within their own system.
The following is the high level architecture of the system which represents the above mentioned functionality:



Scale the design:

As the system is going to be used by a very large number of users (billions of page visits) and large number of merchants, we need to make our system scalable. **Horizontal scaling** can be used to improve scalability and availability, as we can add more machines into the pools of resources behind a load balancer. Using load balancer helps to prevent requests from going to unhealthy servers, prevent overloading resources, and eliminate single points of failure. Also it can decrypt incoming requests and encrypt server responses, so backend servers do not have to perform these potentially expensive operations. Also, in order to eliminate the risk of single point of failure for our load balancer, we need to have more than one load balancer in our system.



We are going to use relational database in the system to track and store the page visits. Since we need to design a highly available system, we need to have master-slave (active - passive) architecture for our database. The master serves reads and writes, replicating writes to one or more slaves, which serve only reads. If the master goes offline, the system can continue to operate in read-only mode until a slave is promoted to a master or a new master is provisioned. Also we can provide memory cache for our read operations, as reading from cache is much faster than reading from database.
Since the users of our system and the merchants are going to be geographically distributed across the world, we need to have multiple data centers in different regions, and the DNS can route the user to closest geographical data center to reduce network latency.