The principles of REST

The 5 principles of REST are:

1. Contract first approach / Uniform Resource identifiers
2. Statelessness
3. Client-Server model
4. Caching
5. Layered architecture

REST stands for Representational State Transfer where representational state is current state of the object.

# Contract first approach/Uniform Resource Identifiers (URI/URL)

The specification of the API is a contract between the app and the server. the app needs to know that it can hit the same URI to get a particular piece of data, every-time! It also needs to know that it’ll get the data in the format that it expects, and that format will not change. This is the Contract and the URI.

# Statelessness

The specification of the API should provide all the information. It does not expect nor assume that any state from previous calls has been preserved. The server should have no knowledge of prior requests. The client needs to provide all the information necessary for the server to provide a response.

# Client-Server Model

The app doesn’t need to bother about how the server stores data, the particular database used, the tech stack, etc. The point of intersection between the server and the app is the database schema.

The two most important points of intersection between the client (front-end) and the server (back-end) are -

* The data format that the client needs
  + The server needs to support this format on the backend otherwise the client/app may not be able to use the API.
* The granularity of the information.
  + Being specific eliminates wasted bandwidth usage. You don’t want to send out the picture album when the request is for one picture. Most times users are paying for data that are TCP/IP overhead or updates with no meaningful change to the data. A TCP/IP header has 40 bytes. If the TCP packet times out or the server doesn’t receive an ACK then the same packet will be sent again. When the cell towers are busy and there’s a premature timeout, the packets will be sent again. Sending only a few bytes of data a.k.a small payloads in these packets can incur a higher percentage of overhead. This is why having an API endpoint for every property of a resource is a bad idea.

# Caching

Caching is the temporary storage of information outside of the server. In between the client and the server, there are many points of presence where the cache can be stored. The cache can be stored near the server like an API gateway cache on AWS. The cache can be stored somewhere in the middle between the client and the server using a 3rd party solution or using a hierarchy of proxy servers that just holds cache data, and are called caching proxy servers. This is usually a shared cache so this cache can be shared by many clients. Lastly, the cache can be stored on the client/in the app/on the device. It is not shared and is only available to the client. This is called a private cache.

# Layered Architecture

Each layer only knows about the layer next to it, and no more. This promotes separation-of-concerns. Each layer is responsible for a specific role. It knows how to do its job, and passes information to and from its immediate neighboring layer to get the job done.