Explain the rest naming conventions

In order to provide consistent developer experience across many APIs and over a long period of time, all names used by an API should be:

- simple
- intuitive
- consistent

This includes names of interfaces, resources, collections, methods, and messages.

Since many developers are not native English speakers, one goal of these naming conventions is to ensure that the majority of developers can easily understand an API. It does this by encouraging the use of a simple, consistent, and small vocabulary when naming methods and resources.

- Names used in APIs should be in correct American English. For example, license (instead of licence), color (instead of colour).

- Commonly accepted short forms or abbreviations of long words may be used for brevity. For example, API is preferred over Application Programming Interface.

- Use intuitive, familiar terminology where possible. For example, when describing removing (and destroying) a resource, delete is preferred over erase.

- Use the same name or term for the same concept, including for concepts shared across APIs.

- Avoid name overloading. Use different names for different concepts.

- Avoid overly general names that are ambiguous within the context of the API and the larger ecosystem of Google APIs. They can lead to misunderstanding of API concepts. Rather, choose - specific names that accurately describe the API concept. This is particularly important for names that define first-order API elements, such as resources. There is no definitive list of names to avoid, as every name must be evaluated in the context of other names. Instance, info, and service are examples of names that have been problematic in the past. Names chosen should describe the API concept clearly (for example: instance of what?) and distinguish it from other relevant concepts (for example: does "alert" mean the rule, the signal, or the notification?).

- Carefully consider use of names that may conflict with keywords in common programming languages. Such names may be used but will likely trigger additional scrutiny during API review. Use them judiciously and sparingly.

NodeJS Frameworks

- Express.js

  Express is an open-source framework, developed and maintained by NodeJS foundation. Express.js is designed to function as a middleware and a routing framework. It is a part of MEAN stack for backend development, along with MongoDB database and AngularJS for building frontend.

- Hapi.js

  Hapi.js is created and maintained by Walmart. It's a NodeJS framework for building web applications and services. Hapi.js is a configuration-centric framework that has integrated support for input validation, authentication, caching, cookies, routing, server methods etc.

- Locomotive

  Locomotive is a Node JS framework for building web applications. With its significant support to MVC patterns and REST principles ,it enable developers to build well-architected applications. Developers already familiar with Ruby on Rails Development can find it easier to get started with Locomotive.

- Total.js

  Total.js is a MVC framework for building responsive web apps using HTML, CSS, and Java script. This NodeJS framework works well with front-end frameworks such as AngularJS, ReactJS, Ember, Bootstrap, and Backbone.js. It is compatible with database, such as MongoDB, CouchDB, SQLite, PostgreSQL, and MYSQL.

- Koa.js

  Written by the developers of Express.js, this server framework for NodeJS allows developers to build web applications and APIs. Koa is popular for its error handling capabilities and avoiding callbacks.

- Sails.js

  Sales.io is a MVC framework for NodeJS that enables development of server-side applications in Java script. This framework is popular for building chat applications, multiplayer games, dashboards, and data-driven APIs.

Full Stack NodeJS Frameworks

- Meteor

  Meteor is an open-source framework for building web and mobile apps. Meteor integrates well with React, AngularJS, Cordova, MongoDB, and NPM. With this NodeJS framework, developers have the opportunity to build apps using one language- Java script, in all environments- web browser, application server, and mobile device.

- Keystone.js

  Keystone.js allows building database-driven websites, apps, and APIs in NodeJS. Built on MongoDB and Express.js, it is best suited for building scalable applications such as web portals, eCommerce apps, Content Management System (CMS), and RESTful web services.

- Mean.IO

  Mean. Io is a open-source JS software stack for building dynamic websites and web apps. The MEAN stack includes MongoDB, Express.js, AngularJS, and Node.js for building frontend and backend for web apps.

- Socket.io

  Socket.IO is best suited for apps that need real-time, bidirectional, and event-based communication. Socket.IO aims to make real time apps possible in every browser and mobile device, blurring the differences between the different transport mechanisms. It supports multiple transports, such as Web Sockets, Flash sockets, long polling and more, automatically falling back when a transport fails.

DDOS

DDoS is short for Distributed Denial of Service. DDoS is a type of DOS attack where multiple compromised systems, which are often infected with a Trojan, are used to target a single system causing a Denial of Service (DoS) attack. Victims of a DDoS attack consist of both the end targeted system and all systems maliciously used and controlled by the hacker in the distributed attack.

In a DDoS attack, the incoming traffic flooding the victim originates from many different sources – potentially hundreds of thousands or more. This effectively makes it impossible to stop the attack simply by blocking a single IP address; plus, it is very difficult to distinguish legitimate user traffic from attack traffic when spread across so many points of origin.

Types of DDoS Attacks

There are many types of DDoS attacks. Common attacks include the following:

Traffic attacks:

Traffic flooding attacks send a huge volume of TCP, UDP and ICPM packets to the target. Legitimate requests get lost and these attacks may be accompanied by malware exploitation.

Bandwidth attacks:

This DDoS attack overloads the target with massive amounts of junk data. This results in a loss of network bandwidth and equipment resources and can lead to a complete denial of service.

Application attacks:

Application-layer data messages can deplete resources in the application layer, leaving the target's system services unavailable.