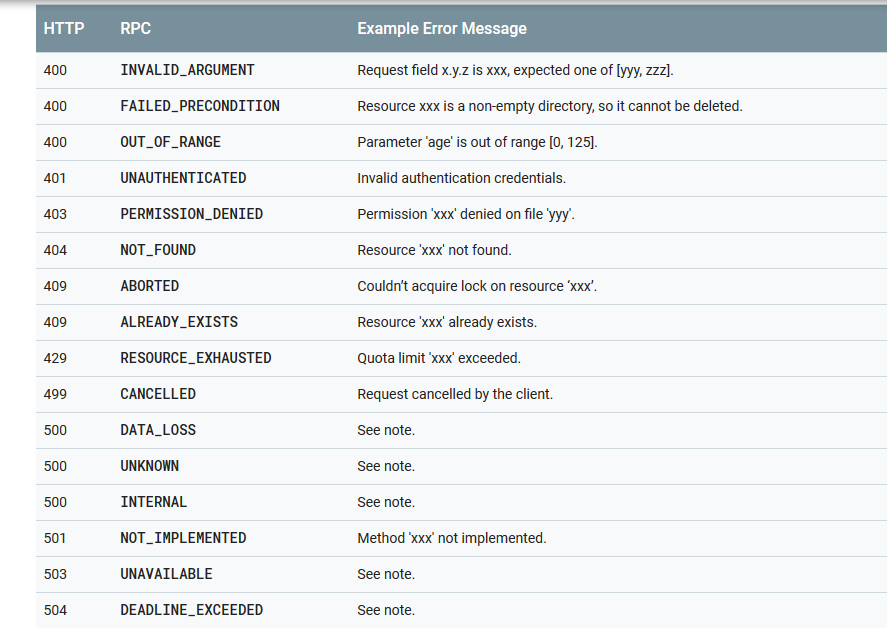
**APIS**

Develop simple, consistent and easy to use APIs

// This is a calendar event resource name.  
"//calendar.googleapis.com/users/john smith/events/123"  
  
// This is the corresponding HTTP URL.  
"https://calendar.googleapis.com/v3/users/john%20smith/events/123"



**Notes**

API Key

* An application programming interface key (API key) is a code passed in by computer programs calling an application programming interface (API) to identify the calling program, its developer, or its user to the Web site. API keys are used to track and control how the API is being used, for example to prevent malicious use or abuse of the API (as defined perhaps by terms of service).
* Best practices for securely using API keys (Google)
  + When you use API keys in your Google Cloud Platform (GCP) applications, take care to keep them secure. Publicly exposing your credentials can result in your account being compromised, which could lead to unexpected charges on your account. To keep your API keys secure, follow these best practices:
    - Do not embed API keys directly in code: API keys that are embedded in code can be accidentally exposed to the public, for example, if you forget to remove the keys from code that you share. Instead of embedding your API keys in your applications, store them in environment variables or in files outside of your application's source tree.
    - Do not store API keys in files inside your application's source tree: If you store API keys in files, keep the files outside your application's source tree to help ensure your keys do not end up in your source code control system. This is particularly important if you use a public source code management system such as GitHub.
    - Restrict your API keys to be used by only the IP addresses, referrer URLs, and mobile apps that need them: By restricting the IP addresses, referrer URLs, and mobile apps that can use each key, you can reduce the impact of a compromised API key. You can specify the hosts and apps that can use each key from the GCP Console Credentials page and then create a new API key with the settings you want, or edit the settings of an existing API key.
    - Restrict your API keys to be usable only for certain APIs: If you have multiple APIs enabled in your project and your API key should only be used with some of them, restrict usage of that key to those APIs. You can specify the allowed APIs for each key from the GCP Console Credentials page and then create a new API key with the settings you want, or edit the settings of an existing API key.
    - Delete unneeded API keys: To minimize your exposure to attack, delete any API keys that you no longer need.
    - Regenerate your API keys periodically: You can regenerate API keys from the GCP Console Credentials page by clicking Regenerate key for each key. Then, update your applications to use the newly-generated keys. Your old keys will continue to work for 24 hours after you generate replacement keys.

**MVC**

Design Principles

* Separation of Concerns
* DRY

**Notes**

**REST API**

GET − Provides a read only access to a resource.

PUT − Used to create a new resource.

DELETE − Ued to remove a resource.

POST − Used to update a existing resource or create a new resource.

OPTIONS − Used to get the supported operations on a resource.

GET: retrieve resources

POST: create resources

PUT: update resources

DELETE: delete resources

**Resources**

Resources will be the targets of the actions, in our case Articles and Users, and they have their own endpoints:

/articles

/users

**Handling Errors**

HTTP defines over 40 standard status codes that can be used to convey the results of a client’s request. The status codes are divided into the five categories presented here:

1xx: Informational - Communicates transfer protocol-level information

2xx: Success -Indicates that the client’s request was accepted successfully.

3xx: Redirection - Indicates that the client must take some additional action in order to complete their request.

4xx: Client Error - This category of error status codes points the finger at clients.

5xx: Server Error - The server takes responsibility for these error status codes.

**Google API**

## What is a REST API?

A REST API is modeled as collections of individually-addressable resources (the nouns of the API). Resources are referenced with their [resource names](https://cloud.google.com/apis/design/resource_names) and manipulated via a small set of methods (also known as verbs or operations).

Standard methods for REST Google APIs (also known as REST methods) are List, Get, Create, Update, and Delete. Custom methods (also known as custom verbs or custom operations) are also available to API designers for functionality that doesn't easily map to one of the standard methods, such as database transactions.

**NOTE**: Custom verbs does not mean creating custom HTTP verbs to support custom methods. For HTTP-based APIs, they simply map to the most suitable HTTP verbs.

## Design flow

The Design Guide suggests taking the following steps when designing resource- oriented APIs (more details are covered in specific sections below):

* Determine what types of resources an API provides.
* Determine the relationships between resources.
* Decide the resource name schemes based on types and relationships.
* Decide the resource schemas.
* Attach minimum set of methods to resources.

## Resources

A resource-oriented API is generally modeled as a resource hierarchy, where each node is either a simple resource or a collection resource. For convenience, they are often called as a resource and a collection, respectively.

* A collection contains a list of resources of **the same type**. For example, a user has a collection of contacts.
* A resource has some state and zero or more sub-resources. Each sub-resource can be either a simple resource or a collection resource.

For example, Gmail API has a collection of users, each user has a collection of messages, a collection of threads, a collection of labels, a profile resource, and several setting resources.

While there is some conceptual alignment between storage systems and REST APIs, a service with a resource-oriented API is not necessarily a database, and has enormous flexibility in how it interprets resources and methods. For example, creating a calendar event (resource) may create additional events for attendees, send email invitations to attendees, reserve conference rooms, and update video conference schedules.

## Methods

The key characteristic of a resource-oriented API is that it emphasizes resources (data model) over the methods performed on the resources (functionality). A typical resource-oriented API exposes a large number of resources with a small number of methods. The methods can be either the standard methods or custom methods. For this guide, the standard methods are: List, Get, Create, Update, and Delete.

Where API functionality naturally maps to one of the standard methods, that method **should** be used in the API design. For functionality that does not naturally map to one of the standard methods, custom methods **may** be used. [Custom methods](https://cloud.google.com/apis/design/custom_methods) offer the same design freedom as traditional RPC APIs, which can be used to implement common programming patterns, such as database transactions or data analysis.

## Examples

The following sections present a few real world examples on how to apply resource-oriented API design to large scale services. You can find more examples in the [Google APIs](https://github.com/googleapis/googleapis) repository.

### Gmail API

The Gmail API service implements the Gmail API and exposes most of Gmail functionality. It has the following resource model:

* API service: gmail.googleapis.com
* A collection of users: users/\*. Each user has the following resources.
  + A collection of messages: users/\*/messages/\*.
  + A collection of threads: users/\*/threads/\*.
  + A collection of labels: users/\*/labels/\*.
  + A collection of change history: users/\*/history/\*.
  + A resource representing the user profile: users/\*/profile.
  + A resource representing user settings: users/\*/settings.

### Cloud Pub/Sub API

The pubsub.googleapis.com service implements the [Cloud Pub/Sub API](https://cloud.google.com/pubsub), which defines the following resource model:

* API service: pubsub.googleapis.com
* A collection of topics: projects/\*/topics/\*.
* A collection of subscriptions: projects/\*/subscriptions/\*.

**NOTE:** Other implementations of the Pub/Sub API may choose different resource naming schemes.

### Cloud Spanner API

The spanner.googleapis.com service implements the [Cloud Spanner API](https://cloud.google.com/spanner), which defines the following resource model:

* API service: spanner.googleapis.com
* A collection of instances: projects/\*/instances/\*.
  + A collection of instance operations: projects/\*/instances/\*/operations/\*.
  + A collection of databases: projects/\*/instances/\*/databases/\*.
  + A collection of database operations: projects/\*/instances/\*/databases/\*/operations/\*.
  + A collection of database sessions: projects/\*/instances/\*/databases/\*/sessions/\*

**OAuth is a specification for authorization**

OAuth 2.0 is a specification for authorization, but NOT for authentication. RFC 6749, [3.1. Authorization Endpoint](https://tools.ietf.org/html/rfc6749#section-3.1) explicitly says as follows:

The authorization endpoint is used to interact with the resource owner and obtain an authorization grant. The authorization server MUST first verify the identity of the resource owner. The way in which the authorization server authenticates the resource owner (e.g., username and password login, session cookies) is **beyond the scope of this specification**.

**OAuth authentication?**

Authentication deals information about "who one is". Authorization deals information about "who grants what permissions to whom". Authorization flow contains authentication as its first step. It is the reason people are often confused.

There are many libraries and services that use OAuth 2.0 for authentication. It is often called "social login" and It makes people more confused. If you see "OAuth authentication" (not "OAuth authorization"), it is a solution using OAuth for authentication.

**OpenID Connect**

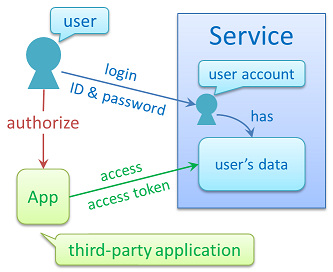
OpenID 1.0 and OpenID 2.0 are old specifications for authentication. Those who made the specifications expected people to use OpenID for authentication. However, some people began to use OAuth 2.0 for authentication (not for authorization) and OAuth authentication has prevailed rapidly.

From a viewpoint of OpenID guys, authentication based on OAuth was not secure enough, but they had to admit that people preferred OAuth authentication. As a result, OpenID guys decided to define a new specification, [**OpenID Connect**](http://openid.net/connect/), on top of OAuth 2.0.

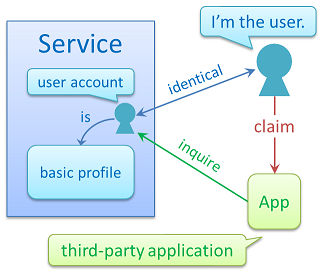
Yes, this has made people much more confused.

**One-sentence definitions of OAuth 2.0 and OpenID Connect**

[**OAuth 2.0**](https://tools.ietf.org/html/rfc6749) is a framework where a user of a service can allow a third-party application to access his/her data hosted in the service without revealing his/her credentials (ID & password) to the application.

[](https://i.stack.imgur.com/zI0Gp.png)

[**OpenID Connect**](http://openid.net/connect/) is a framework on top of OAuth 2.0 where a third-party application can obtain a user's identity information which is managed by a service.

[](https://i.stack.imgur.com/89Wmh.png)

(Sorry, these definitions are excerpts from the [overview](https://www.authlete.com/documents/overview) page of my company)

**Definitions from a viewpoint of implementors**

**Authentication** is a process to determine the subject (= unique identifier) of an end-user. There are many ways to determine the subject. ID & password, fingerprints, iris recognition, etc.

**Authorization** is a process to associate the subject with the requested permissions and the client application that requested the permissions. An access token represents the association.