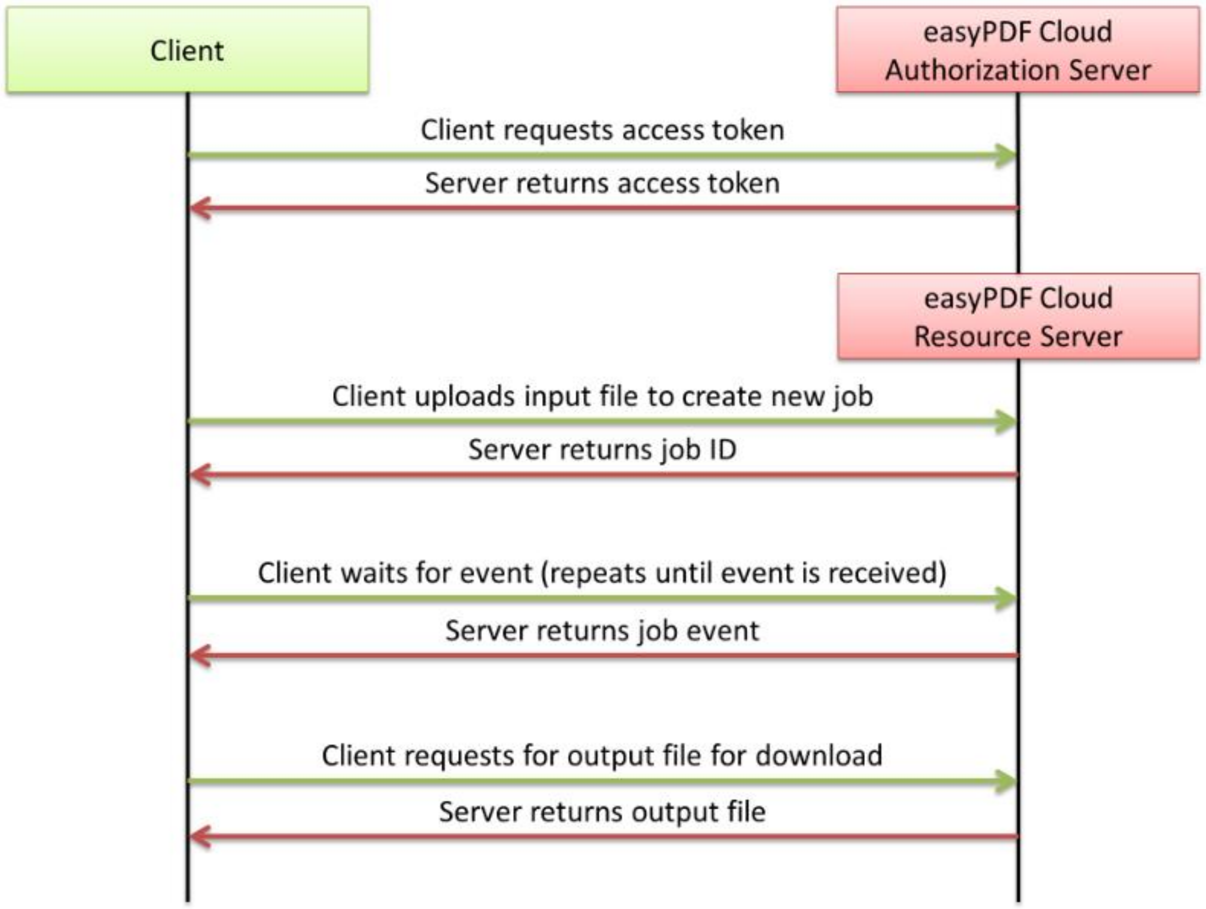
What is a REST API Endpoint?


Client
easyPDF Cloud Authorization Server
Client requests access token
Server returns access token
easyPDF Cloud Resource Server
Client uploads input file to create new job
Server returns job ID
Client waits for event (repeats until event is received)
Server returns job event
Client requests for output file for download
Server returns output file

See above picture for a brief explanation.

Then there is a mass of documentation on the internet:

Intro to REST API's
REST Basics
REST API Developer Guide
IBM REST API Developer Guide
How to use an API
What is Rest

Many companies use the REST API to share information from equipment or from websites.

And below is some information on how to implement it on an ESP device.

ESP32 HTTP GET and HTTP POST with Arduino IDE (JSON, URL Encoded, Text)

How to build a REST API server with ESP32

Also YouTube has lots of information.

What Is REST API? Examples And How To Use It: Crash Course System Design

What Is A RESTful API? Explanation of REST & HTTP

Test equipment:

You can test while using Node Red (use the HTTP GET function).

Postman

There is also a web version of Postman.

There is a Postman tutorial.

In depth:

CRUD on English wiki

CRUD on Dutch wiki

In REST (with http) there is also a CRD mechanism:

	Description	**SQL**	**HTTP (restful)**
C	Create or insert new data	Insert	*http-post* or *http-put*
R	Read or request for data	Select	*http-get*
U	Update data	Update	*http-put* to replace, *http-patch* to modify
D	Delete data	Delete	*http-delete*

What is RESTful API?

RESTful API is an interface that two computer systems use to exchange information securely over the internet. Most business applications have to communicate with other internal and third-party applications to perform various tasks. For example, to generate monthly payslips, your internal accounts system has to share data with your customer's banking system to automate invoicing and communicate with an internal timesheet application. RESTful APIs support this information exchange because they follow secure, reliable, and efficient software communication standards.

What is an API?

An application programming interface (API) defines the rules that you must follow to communicate with other software systems. Developers expose or create APIs so that other applications can communicate with their applications programmatically. For example, the timesheet application exposes an API that asks for an employee's full name and a range of dates. When it receives this information, it internally processes the employee's timesheet and returns the number of hours worked in that date range.

You can think of a web API as a gateway between clients and resources on the web.

Clients

Clients are users who want to access information from the web. The client can be a person or a software system that uses the API. For example, developers can write programs that access weather data from a weather system. Or you can access the same data from your browser when you visit the weather website directly.

Resources (servers)

Resources are the information that different applications provide to their clients. Resources can be images, videos, text, numbers, or any type of data. The machine that gives the resource to the client is also called the server. Organizations use APIs to share resources and provide web services while maintaining security, control, and authentication. In addition, APIs help them to determine which clients get access to specific internal resources.

What is REST?

Representational State Transfer (REST) is a software architecture that imposes conditions on how an API should work. REST was initially created as a guideline to manage communication on a complex network like the internet. You can use REST-based architecture to support high-performing and reliable communication at scale. You can easily implement and modify it, bringing visibility and cross-platform portability to any API system.

API developers can design APIs using several different architectures. APIs that follow the REST architectural style are called REST APIs. Web services that implement REST architecture are called RESTful web services. The term RESTful API generally refers to RESTful web APIs. However, you can use the terms REST API and RESTful API interchangeably.

The following are some of the principles of the REST architectural style:

Uniform interface

The uniform interface is fundamental to the design of any RESTful web service. It indicates that the server transfers information in a standard format. The formatted resource is called a representation in REST. This format can be different from the internal representation of the resource on the server application. For example, the server can store data as text but send it in an HTML representation format.

Uniform interface imposes four architectural constraints:

1. Requests should identify resources. They do so by using a uniform resource identifier.
2. Clients have enough information in the resource representation to modify or delete the resource if they want to. The server meets this condition by sending metadata that describes the resource further.
3. Clients receive information about how to process the representation further. The server achieves this by sending self-descriptive messages that contain metadata about how the client can best use them.
4. Clients receive information about all other related resources they need to complete a task. The server achieves this by sending hyperlinks in the representation so that clients can dynamically discover more resources.

Statelessness

In REST architecture, statelessness refers to a communication method in which the server completes every client request independently of all previous requests. Clients can request resources in any order, and every request is stateless or isolated from other requests. This REST API design constraint implies that the server can completely understand and fulfill the request every time.

Layered system

In a layered system architecture, the client can connect to other authorized intermediaries between the client and server, and it will still receive responses from the server. Servers can also pass on requests to other servers. You can design your RESTful web service to run on several servers with multiple layers such as security, application, and business logic, working together to fulfill client requests. These layers remain invisible to the client.

Cacheability

RESTful web services support caching, which is the process of storing some responses on the client or on an intermediary to improve server response time. For example, suppose that you visit a website that has common header and footer images on every page. Every time you visit a new website page, the server must resend the same images. To avoid this, the client caches or stores these images after the first response and then uses the images directly from the cache. RESTful web services control caching by using API responses that define themselves as cacheable or noncacheable.

Code on demand

In REST architectural style, servers can temporarily extend or customize client functionality by transferring software programming code to the client. For example, when you fill a registration form on any website, your browser immediately highlights any mistakes you make, such as incorrect phone numbers. It can do this because of the code sent by the server.

What are the benefits of RESTful APIs?

RESTful APIs include the following benefits:

Scalability

Systems that implement REST APIs can scale efficiently because REST optimizes client-server interactions. Statelessness removes server load because the server does not have to retain past client request information. Well-managed caching partially or completely eliminates some client-server interactions. All these features support scalability without causing communication bottlenecks that reduce performance.

Flexibility

RESTful web services support total client-server separation. They simplify and decouple various server components so that each part can evolve independently. Platform or technology changes at the server application do not affect the client application. The ability to layer application functions increases flexibility even further. For example, developers can make changes to the database layer without rewriting the application logic.

Independence

REST APIs are independent of the technology used. You can write both client and server applications in various programming languages without affecting the API design. You can also change the underlying technology on either side without affecting the communication.

How do RESTful APIs work?

The basic function of a RESTful API is the same as browsing the internet. The client contacts the server by using the API when it requires a resource. API developers explain how the client should use the REST API in the server application API documentation. These are the general steps for any REST API call:

1. The client sends a request to the server. The client follows the API documentation to format the request in a way that the server understands.
2. The server authenticates the client and confirms that the client has the right to make that request.
3. The server receives the request and processes it internally.
4. The server returns a response to the client. The response contains information that tells the client whether the request was successful. The response also includes any information that the client requested.

The REST API request and response details vary slightly depending on how the API developers design the API.

What does the RESTful API client request contain?

RESTful APIs require requests to contain the following main components:

Unique resource identifier

The server identifies each resource with unique resource identifiers. For REST services, the server typically performs resource identification by using a Uniform Resource Locator (URL). The URL specifies the path to the resource. A URL is similar to the website address that you enter into your browser to visit any webpage. The URL is also called the request endpoint and clearly specifies to the server what the client requires.

Method

Developers often implement RESTful APIs by using the Hypertext Transfer Protocol (HTTP). An HTTP method tells the server what it needs to do to the resource. The following are four common HTTP methods:

GET

Clients use GET to access resources that are located at the specified URL on the server. They can cache GET requests and send parameters in the RESTful API request to instruct the server to filter data before sending.

POST

Clients use POST to send data to the server. They include the data representation with the request. Sending the same POST request multiple times has the side effect of creating the same resource multiple times.

PUT

Clients use PUT to update existing resources on the server. Unlike POST, sending the same PUT request multiple times in a RESTful web service gives the same result.

DELETE

Clients use the DELETE request to remove the resource. A DELETE request can change the server state. However, if the user does not have appropriate authentication, the request fails.

HTTP headers

Request headers are the metadata exchanged between the client and server. For instance, the request header indicates the format of the request and response, provides information about request status, and so on.

Data

REST API requests might include data for the POST, PUT, and other HTTP methods to work successfully.

Parameters

RESTful API requests can include parameters that give the server more details about what needs to be done. The following are some different types of parameters:

- Path parameters that specify URL details.
- Query parameters that request more information about the resource.
- Cookie parameters that authenticate clients quickly.

What are RESTful API authentication methods?

A RESTful web service must authenticate requests before it can send a response. Authentication is the process of verifying an identity. For example, you can prove your identity by showing an ID card or driver's license. Similarly, RESTful service clients must prove their identity to the server to establish trust.

RESTful API has four common authentication methods:

HTTP authentication

HTTP defines some authentication schemes that you can use directly when you are implementing REST API. The following are two of these schemes:

Basic authentication

In basic authentication, the client sends the user name and password in the request header. It encodes them with base64, which is an encoding technique that converts the pair into a set of 64 characters for safe transmission.

Bearer authentication

The term bearer authentication refers to the process of giving access control to the token bearer. The bearer token is typically an encrypted string of characters that the server generates in response to a login request. The client sends the token in the request headers to access resources.

API keys

API keys are another option for REST API authentication. In this approach, the server assigns a unique generated value to a first-time client. Whenever the client tries to access resources, it uses the unique API key to verify itself. API keys are less secure because the client has to transmit the key, which makes it vulnerable to network theft.

OAuth

OAuth combines passwords and tokens for highly secure login access to any system. The server first requests a password and then asks for an additional token to complete the authorization process. It can check the token at any time and also over time with a specific scope and longevity.

What does the RESTful API server response contain?

REST principles require the server response to contain the following main components:

Status line

The status line contains a three-digit status code that communicates request success or failure. For instance, 2XX codes indicate success, but 4XX and 5XX codes indicate errors. 3XX codes indicate URL redirection.

The following are some common status codes:

- 200: Generic success response
- 201: POST method success response
- 400: Incorrect request that the server cannot process
- 404: Resource not found

Message body

The response body contains the resource representation. The server selects an appropriate representation format based on what the request headers contain. Clients can request information in XML or JSON formats, which define how the data is written in plain text. For example, if the client requests the name and age of a person named John, the server returns a JSON representation as follows:

'{"name":"John", "age":30}'

Headers

The response also contains headers or metadata about the response. They give more context about the response and include information such as the server, encoding, date, and content type.