Unit 1 - What is an API

Defining Interface (0:04:50)

Defining API (0:08:49)

Remote APIs (0:13:59)

How the web works (0:18:17)

RESTful API Constraint Scavenger Hunt (0:23:21)

Unit 2 - Exploring APIs

⌨️ Exploring an API online (0:29:06)

⌨️ Using an API from the command line (0:45:01)

⌨️ You go Curl (0:56:20)

⌨️ Using tools to explore APIs (1:14:21)

⌨️ More tools for your API exploring toolbox (1:36:20)

⌨️ Using Helper Libraries (1:48:34)

Unit 3 - All Together Now

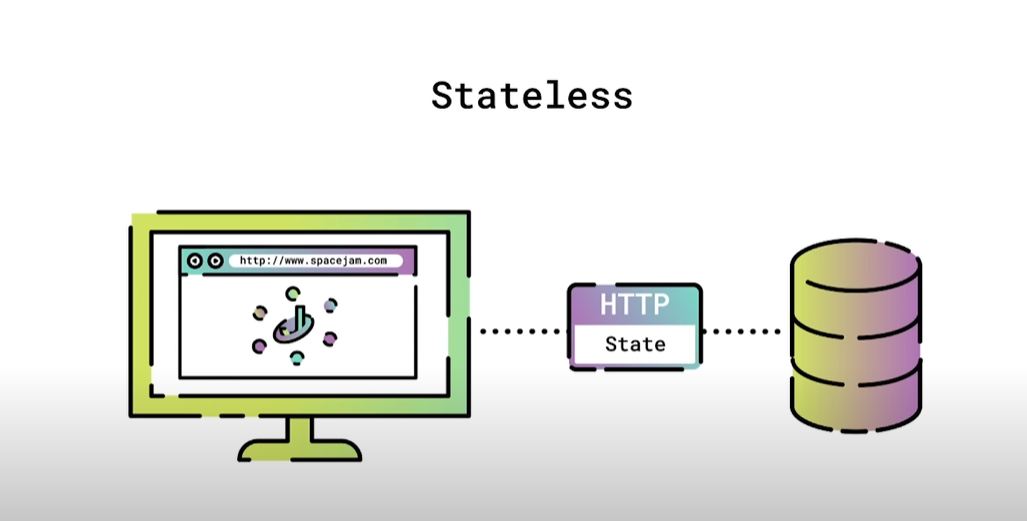
⌨️ Introducing the Project (2:15:30)

⌨️ Serverless (2:23:30)

⌨️ Writing a Server Side API (2:37:08)

⌨️ Fetching Results on the Client from our Server (2:56:54)

⌨️ Wrap Up (3:05:29)  
=================================================================  
what is https: hyper text protocol(protocol: rules and regulation we follow over the internet to do action like CRUD)  
  
HTTP: port 80 and not secure.  
  
HTTP Secure : **Port 443.** [HTTPS](https://www.techtarget.com/searchsoftwarequality/definition/HTTPS) is like HTTP but more [secure](https://www.techtarget.com/searchsecurity/tip/How-to-encrypt-and-secure-a-website-using-HTTPS). All HTTPS web traffic goes straight to port 443. Any network service that uses HTTPS for encryption, such as DNS over HTTPS, also connects directly to this port.

Https request header:  
**HTTP headers** let the client and the server pass additional information with an HTTP request or response. An HTTP header consists of its case-insensitive name followed by a colon (:), then by its value. [Whitespace](https://developer.mozilla.org/en-US/docs/Glossary/Whitespace) before the value is ignored.  
  


# API: Application programming interface(interface: points of communication between different components of an application or system, we only see the graphic but functionality we are not aware about) ## HTTP request methods/verb:

HTTP defines a set of **request methods** to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as *HTTP verbs*. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be [safe](https://developer.mozilla.org/en-US/docs/Glossary/Safe/HTTP), [idempotent](https://developer.mozilla.org/en-US/docs/Glossary/Idempotent), or [cacheable](https://developer.mozilla.org/en-US/docs/Glossary/Cacheable).

[GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET)

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

[HEAD](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD)

The HEAD method asks for a response identical to a GET request, but without the response body.

[POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST)

The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

[PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT)

The PUT method replaces all current representations of the target resource with the request payload.

[DELETE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE)

The DELETE method deletes the specified resource.

[CONNECT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/CONNECT)

The CONNECT method establishes a tunnel to the server identified by the target resource.

[OPTIONS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS)

The OPTIONS method describes the communication options for the target resource.

[TRACE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/TRACE)

The TRACE method performs a message loop-back test along the path to the target resource.

[PATCH](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH)

The PATCH method applies partial modifications to a resource.

================================  
What is REST ?

REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.

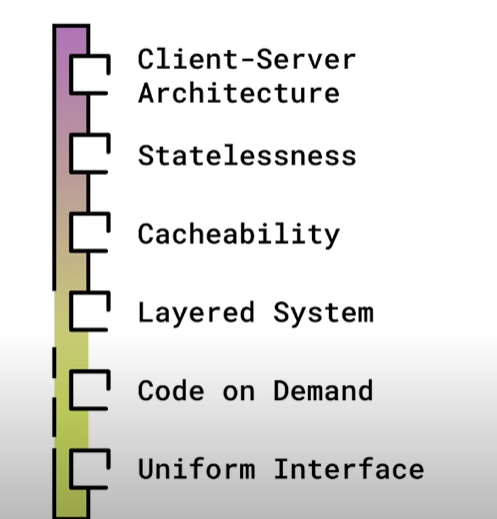
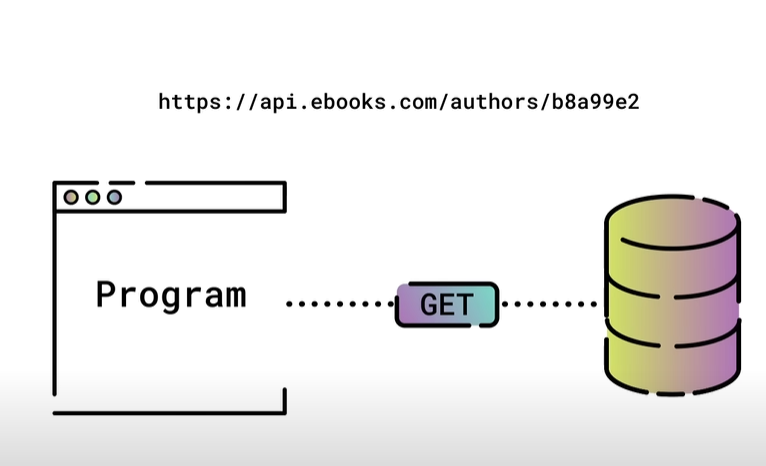
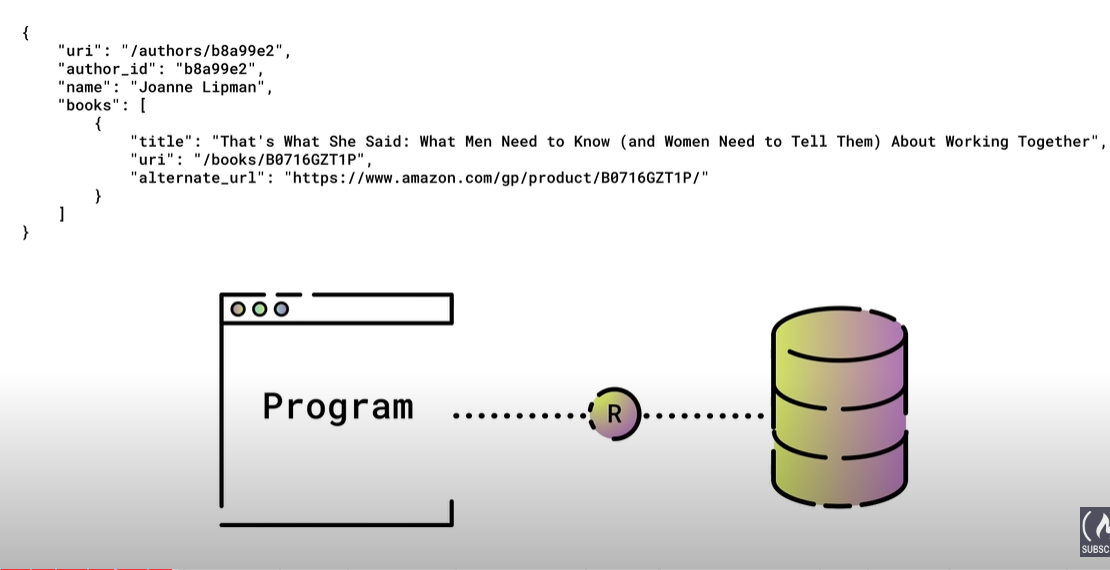
When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it’s language-agnostic, as well as readable by both humans and machines.

Something else to keep in mind: Headers and parameters are also important in the HTTP methods of a RESTful API HTTP request, as they contain important identifier information as to the request's metadata, authorization, uniform resource identifier (URI), caching, cookies, and more. There are request headers and response headers, each with their own HTTP connection information and status codes.

In order for an API to be considered RESTful, it has to conform to these criteria:

* A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
* [Stateless](https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless) client-server communication, meaning no client information is stored between get requests and each request is separate and unconnected.
* Cacheable data that streamlines client-server interactions.
* A uniform interface between components so that information is transferred in a standard form. This requires that:
  + resources requested are identifiable and separate from the representations sent to the client.
  + resources can be manipulated by the client via the representation they receive because the representation contains enough information to do so.
  + self-descriptive messages returned to the client have enough information to describe how the client should process it.
  + hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.
* A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involved the retrieval of requested information into hierarchies, invisible to the client.
* Code-on-demand (optional): the ability to send executable code from the server to the client when requested, extending client functionality.

Though the REST API has these criteria to conform to, it is still considered easier to use than a prescribed protocol like SOAP (Simple Object Access Protocol), which has specific requirements like XML messaging, and built-in security and transaction compliance that make it slower and heavier.

In contrast, REST is a set of guidelines that can be implemented as needed, making REST APIs faster and more lightweight, with increased scalablity—perfect for [Internet of Things (IoT)](https://www.redhat.com/en/topics/internet-of-things) and [mobile app development](https://www.redhat.com/en/topics/mobile).  
­==========================================  
following constrain should be there to be API RESTfull:  
  
you want to access something from server so you sent the GET request with URI:  
  
Than response come in Json format:  
  
JSON: java script object notation:  
  
=> We can perform the CRUD operation on server using following method:  
