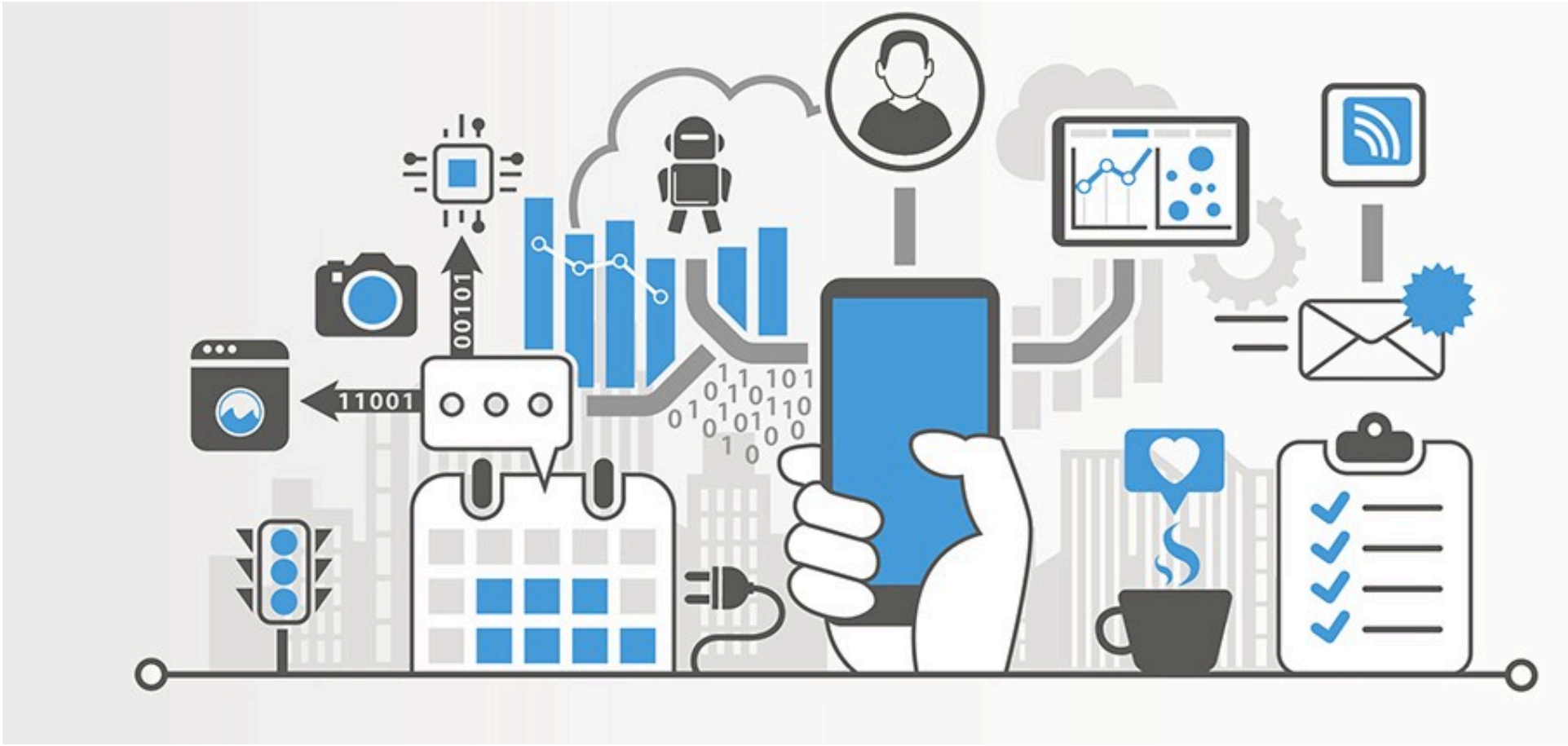
Programming for IoT Applications

Edoardo Patti

Lecture 5



Web Services introduction

WEB PROGRAMMING

Web development

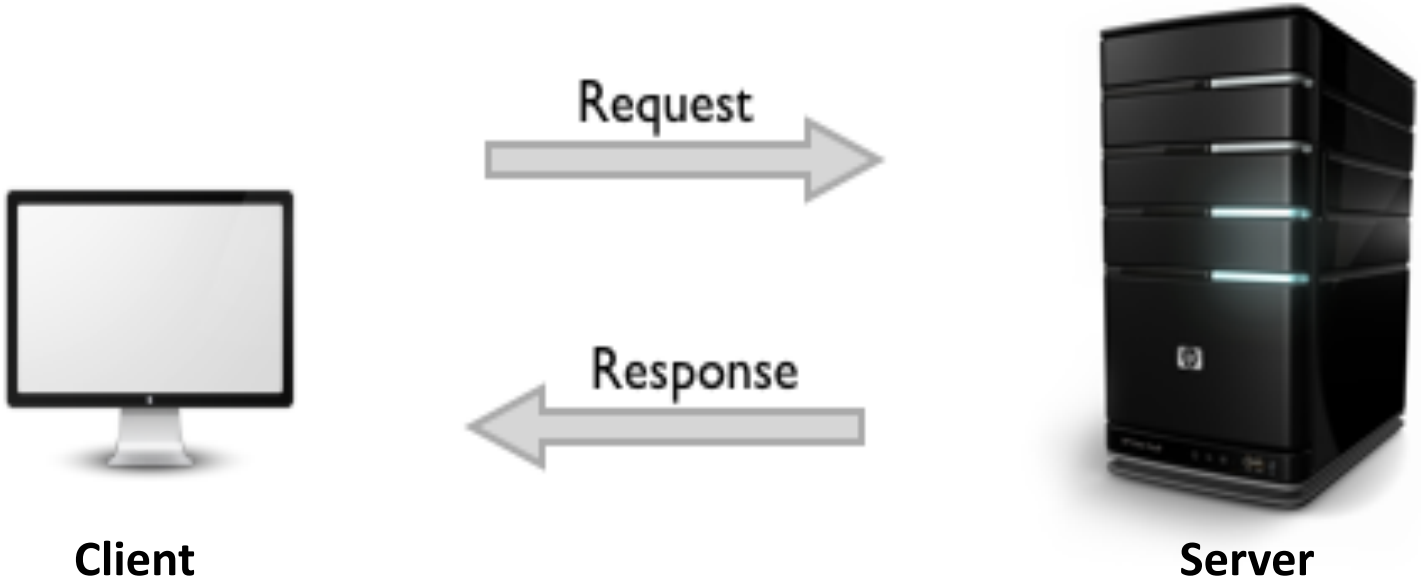
From Wikipedia, the free encyclopedia
(Redirected from Web programming)

Web development is the work involved in developing a web site for the Internet (World Wide Web) or an intranet (a private network).[1] Web development can range from developing a simple single static page of plain text to complex web-based internet applications (web apps), electronic businesses, and social network services. A more comprehensive list of tasks to which web development commonly refers, may include web engineering, web design, web content development, client liaison, client-side/server-side scripting, web server and network security configuration, and e-commerce development.


Two main communication paradigms:
• Request/Response
• Publish/Subscribe

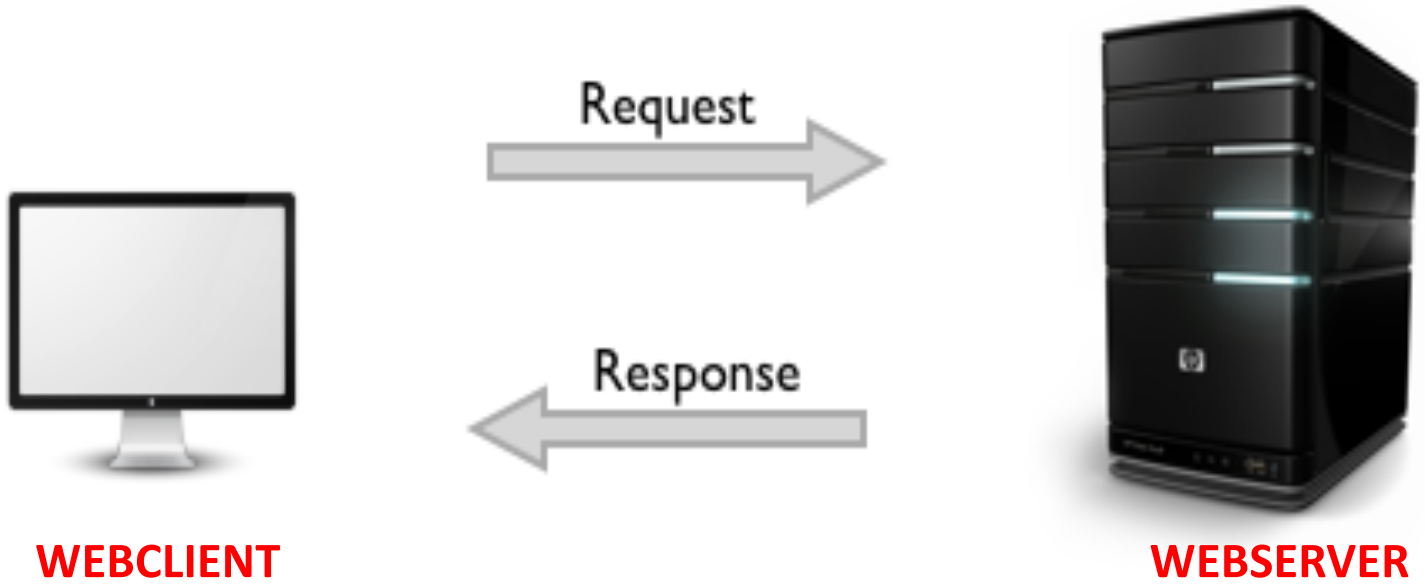
Communication paradigms

Request/Response is a **synchronous** communication paradigm. The client requests for some data and the server responds to the request.



Communication paradigms

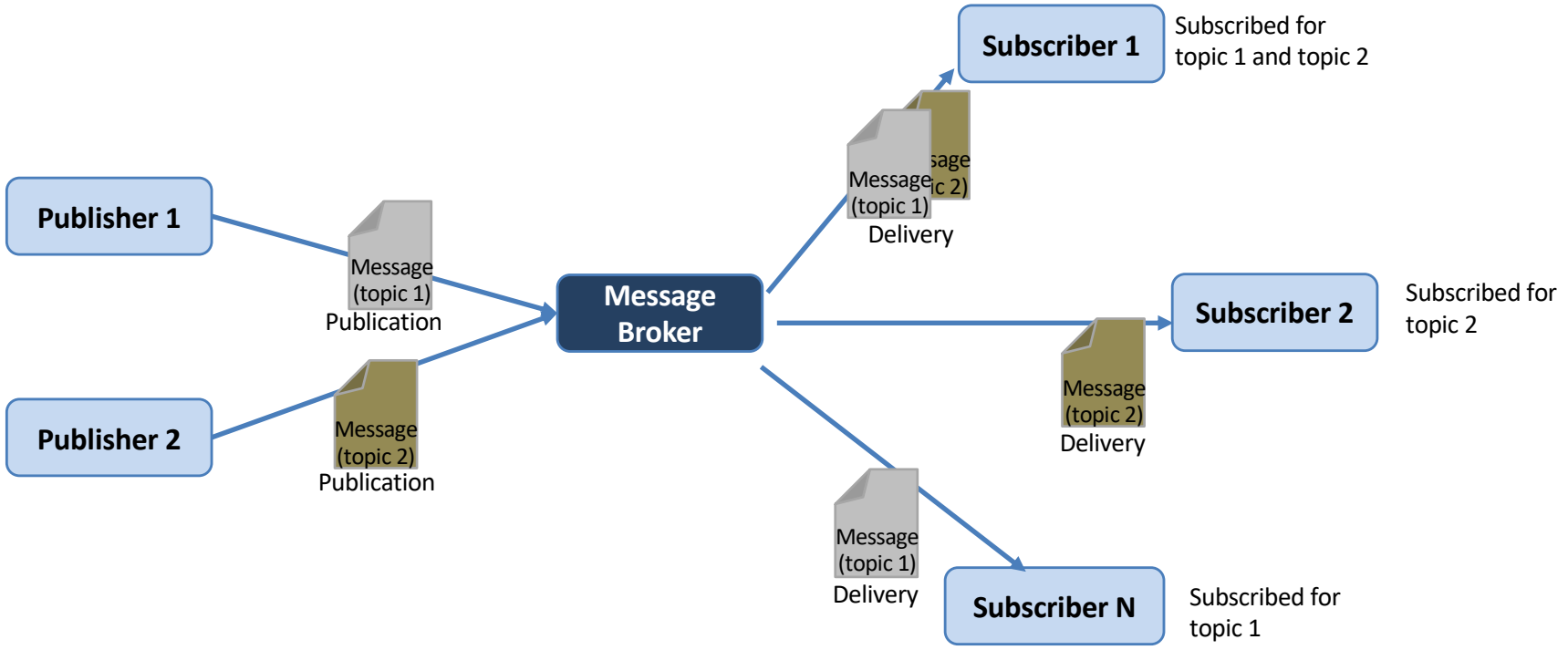
Request/Response is a **synchronous** communication paradigm. The client requests for some data and the server responds to the request.




SPOILER
ALERT!

Communication paradigms

Publish/subscribe is an **asynchronous** communication paradigm. It allows the development of loosely-coupled event-driven architectures. It removes the dependencies between producer and consumer of information.



Event-driven architecture

From Wikipedia, the free encyclopedia

Event-driven architecture (**EDA**) is a software architecture paradigm promoting the production, detection, consumption of, and reaction to events.


END

Webserver

- A webserver exposes services to clients

Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port

Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers

Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers
  - http://www.mywebsite.com:8080
  - http://192.168.1.34:8080

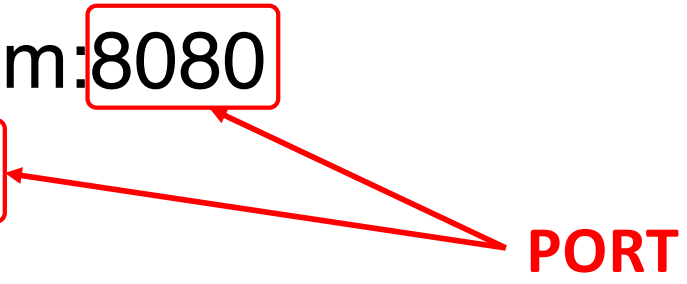
Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers
  - http://www.mywebsite.com:8080
  - http://192.168.1.34:8080

HOST

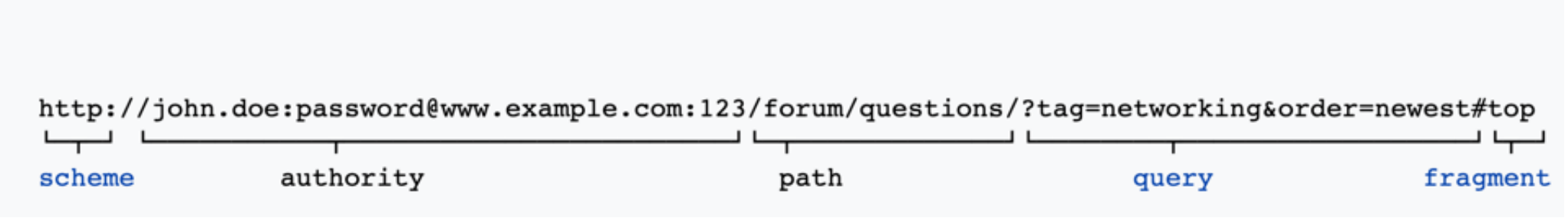
Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers
  - http://www.mywebsite.com:8080
  - http://192.168.1.34:8080



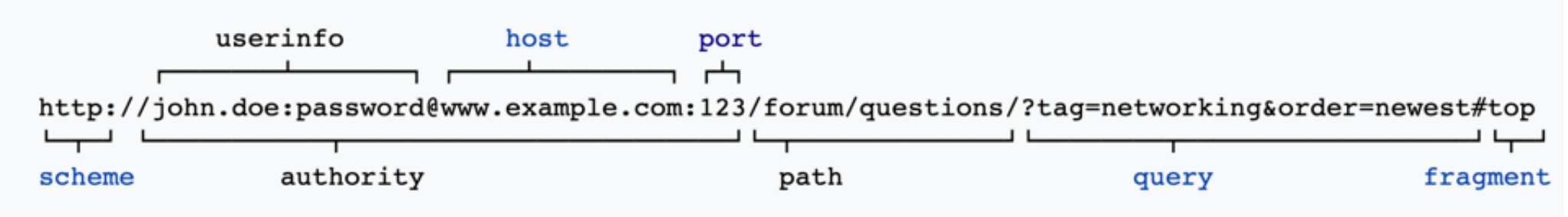
Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers



Webserver

- A webserver exposes services to clients
- A webserver can be reached using host:port
- A port is a communication endpoint identifing a specific process, application or a type of service running on server. It is 16-bit unsigned numbers



Webserver

A webserver must always be up and running ready to provide information

Webclient

- A webclient consumes services exposed by webserver

Webclient

- A webclient consumes services exposed by webserver
- A webclient starts the communication by indicating host:port

What is a Webservice?

The W3C (World Wide Web Consortium) defines a web service as *a software system designed to support interoperable machine-to-machine interaction over a network.*

What is a Webservice?

A web service is a **service** offered by an electronic device to another electronic device **communicating with each other via the World Wide Web**.

Web technology such as **HTTP is used for machine-to-machine communication**, transferring machine-readable file formats such as JSON and XML.

Hypertext Transfer Protocol

- HTTP is the most widespread protocol for communications between webserver and webclients

Hypertext Transfer Protocol

- HTTP is the most widespread protocol for communications between webserver and webclients
  - There are alternatives in the IoT worlds we will study, e.g. MQTT

Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia

(Redirected from Http)

The **Hypertext Transfer Protocol** (**HTTP**) is an application protocol for distributed, collaborative, hypermedia information systems.[1] HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.

Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia

(Redirected from Http)

Technical overview [edit]

HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the *client* and an application running on a computer hosting a website may be the *server*. The client submits an HTTP *request* message to the server. The server, which provides *resources* such as HTML files and other content, or performs other functions on behalf of the client, returns a *response* message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia

(Redirected from Http)

HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes *http* and *https*.

Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia

(Redirected from Http)

HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes *http* and *https*.

A **Uniform Resource Identifier** (**URI**) is a string of characters that unambiguously identifies a particular resource. To guarantee uniformity, all URIs follow a predefined set of syntax rules,[1] but also maintain extensibility through a separately defined hierarchical naming scheme (e.g. `http://`).

Such identification enables interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols. Schemes specifying a concrete syntax and associated protocols define each URI. The most common form of URI is the Uniform Resource Locator (URL), frequently referred to informally as a *web address*.

Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia

(Redirected from Http)

HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes *http* and *https*.

A **Uniform Resource Locator** (**URL**), colloquially termed a **web address**,[1] is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier (URI),[2][3] although many people use the two terms interchangeably.[4][a] URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database access (JDBC), and many other applications.

HTTP Requests

- HTTP is used by the client to make requests to the server
- HTTP makes different types of requests, among all:
  - **GET**
  - **POST**
  - **PUT**
  - **DELETE**

HTTP Requests

From Wikipedia, the free encyclopedia

(Redirected from Http)

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.

HTTP Requests

From Wikipedia, the free encyclopedia
(Redirected from Http)

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.

- A GET request is made each time the client (e.g. web browser) connects to a given URL
- For instance:
  - http://localhost:8080/ - is asking for the “/” URL in the localhost.
  - http://localhost:8080/hello - is asking for the “/hello” URL in the localhost

HTTP Requests

From Wikipedia, the free encyclopedia

(Redirected from Http)

POST

The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.[24]

HTTP Requests

From Wikipedia, the free encyclopedia

(Redirected from Http)

PUT

The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.[25]

HTTP Requests

From Wikipedia, the free encyclopedia
(Redirected from Http)

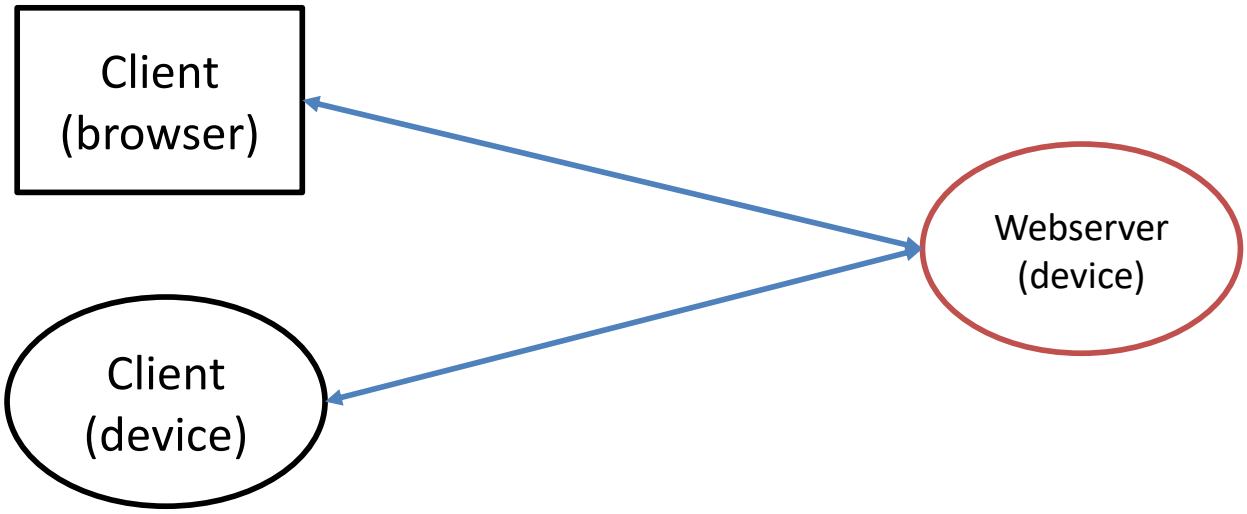
PUT

The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.[25]

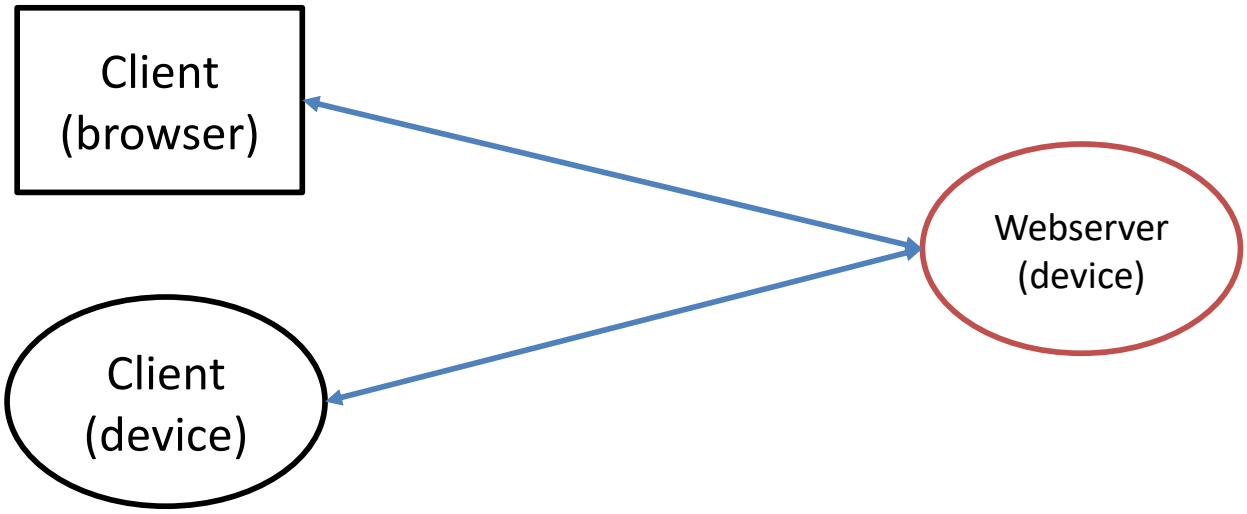
DELETE

The DELETE method deletes the specified resource.

Webservices and IoT

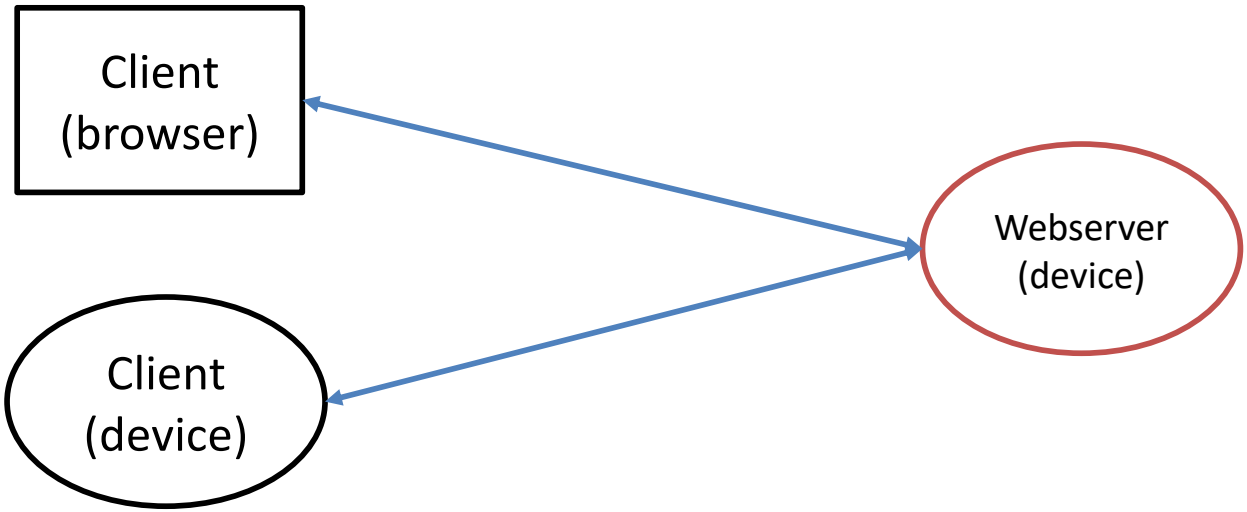


Webservices and IoT



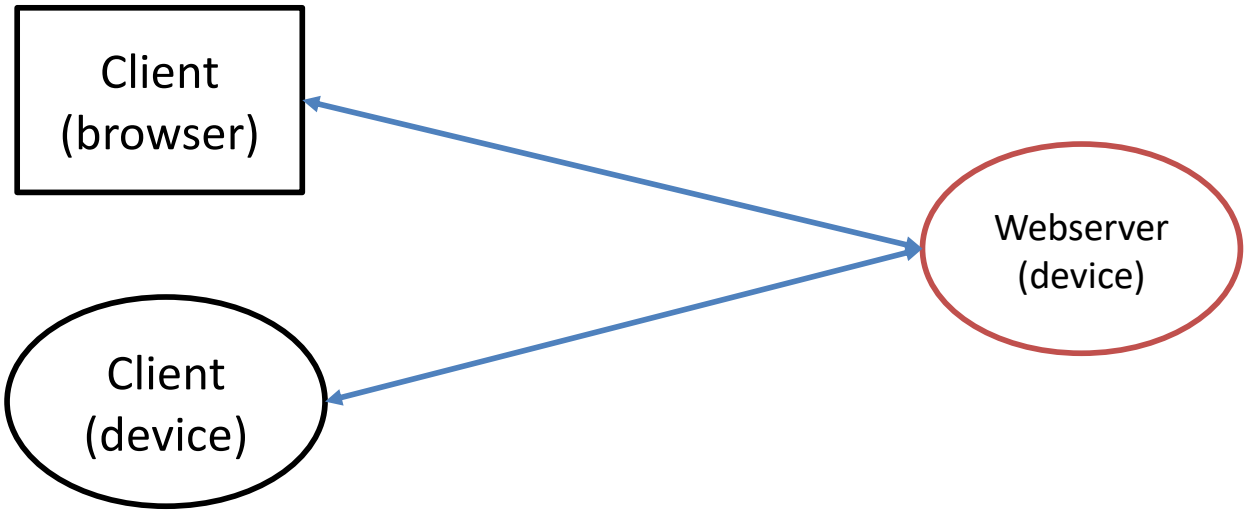
- A device will implement a webserver in a certain language using a web library
  - E.g. python with cherrypy, Java, etc…

Webservices and IoT



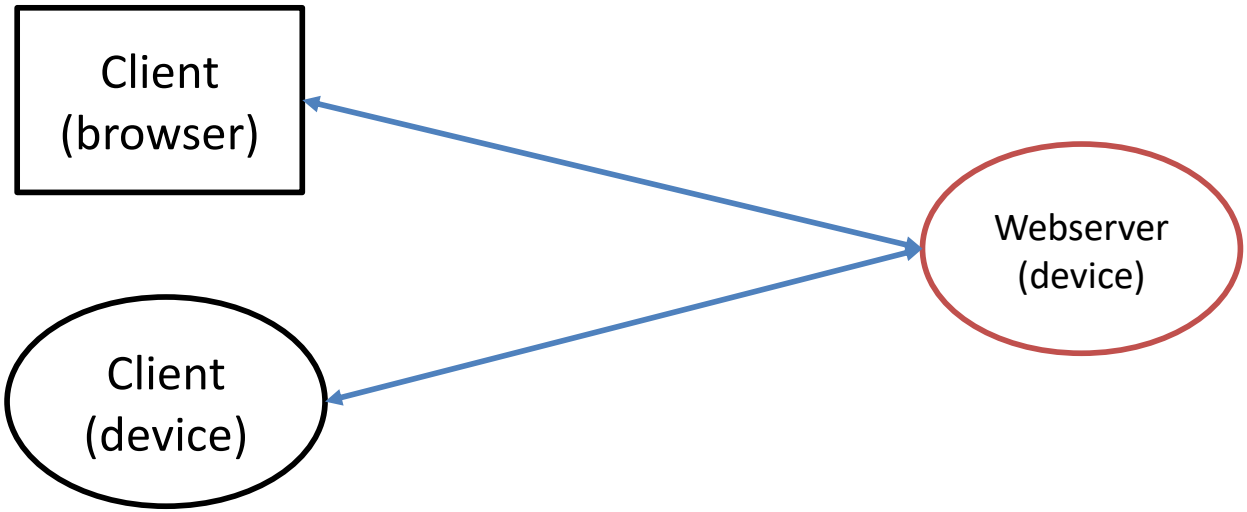
- A device will implement a webserver in a certain language using a web library
  - E.g. python with cherrypy, Java, etc…
- Through the webservices (e.g. using GET/POST methods) the device can be:
  - Configured
  - Send data/commands
  - Receive data/commands

Webservices and IoT



- A device will implement a webserver in a certain language using a web library
  - E.g. python with cherrypy, Java, etc…
- Through the webservices (e.g. using GET/POST methods) the device can be:
  - Configured
  - Send data/commands
  - Receive data/commands
- The client can be another device or a data collector node

Webservices and IoT



- A device will implement a webserver in a certain language using a web library
  - E.g. python with cherrypy, Java, etc…
- Through the webservices (e.g. using GET/POST methods) the device can be:
  - Configured
  - Send data/commands
  - Receive data/commands
- The client can be another device or a data collector node
- **Webserver should be lightweight**

HTML pages

- HTTP is a protocol for client-server communication
- The information exchanged by clients and servers can be based on different data format for visualization and content access, such as HTML pages
- HTML pages can also contain scripts to be executed (e.g. Javascripts)
- HTML pages can be associated with style specifications (e.g. CSS)

HTML pages

- HTML pages can provide input to and outputs from a web application
- HTML pages can contain information to trigger specific methods in the classes used by webservers as in the case of POST requests in a form:

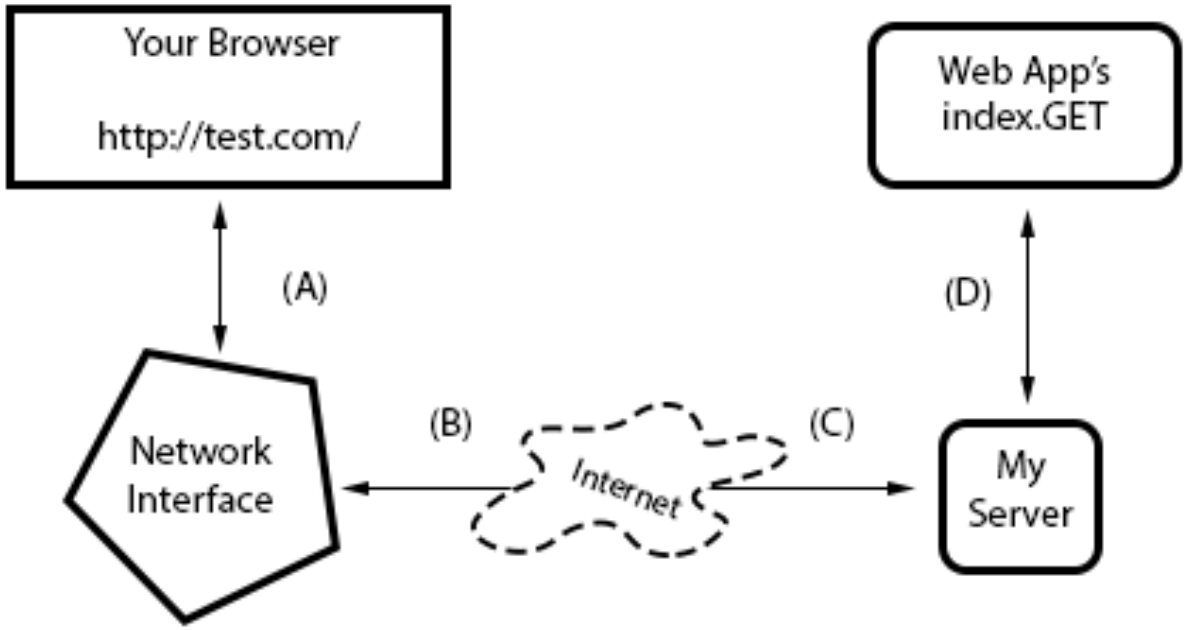
HTML pages

- HTML pages can provide input to and outputs from a web application
- HTML pages can contain information to trigger specific methods in the classes used by webservers as in the case of POST requests in a form:
  - E.g.: **<form** action="/reply" method="POST">

```
<html>
    <head>
        <title>Sample Web Form</title>
    </head>
<body>

<h1>Fill Out This Form</h1>

<form action=”/reply" method="POST">
    A Greeting: <input type="text" name="greet">
    <br/>
    Your Name: <input type="text" name="name">
    <br/>
    <input type="submit">
</form>

</body>
</html>
```

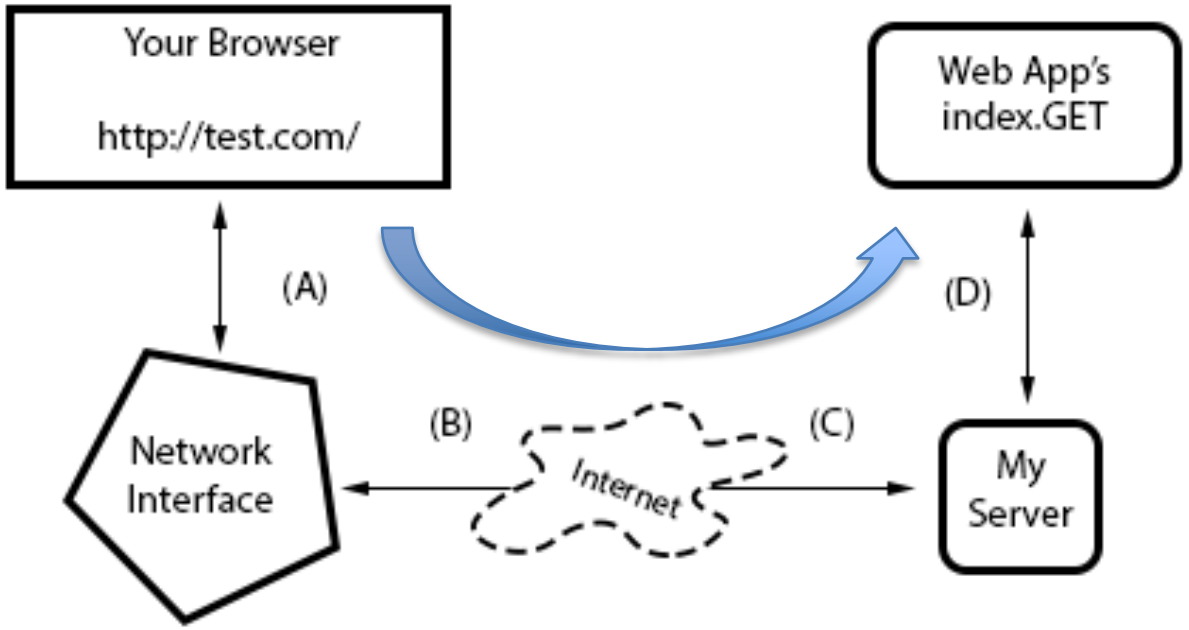
How the web works

- GET method is used (for instance) to retrieve a web page



How the web works

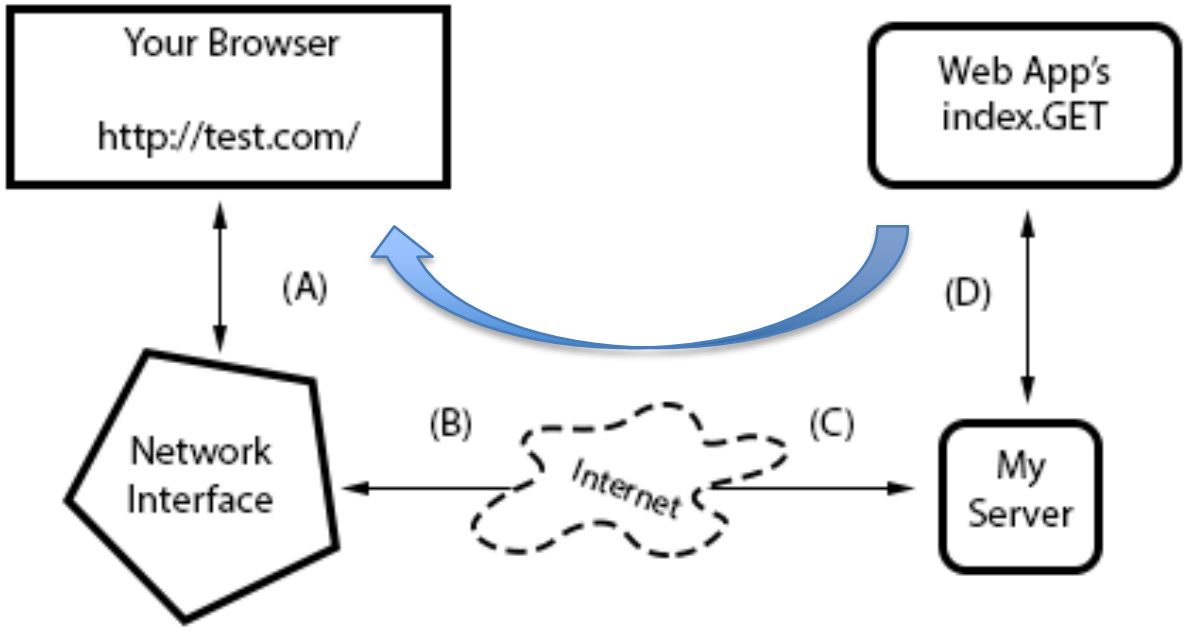
- GET method is used (for instance) to retrieve a web page



GET request path: A -> B -> C -> D

How the web works

- GET method is used (for instance) to retrieve a web page



GET request path: A -> B -> C -> D
HTML file path: D -> C -> B -> A

Form with HTML files

The HTML template (hello_form.html) and the web page to be returned (page.html) are located in ./hello

```
import cherrypy

class Generator(object):

          def hello (self, *uri, ** params):
                    return
open('./hello/hello_form.html','r').read()

          hello.exposed = True

          def reply (self, *uri, ** params):
                    return
open('./hello/page.html','r').read()

          reply.exposed = True

if __name__           == '__main__':
          cherrypy.tree.mount (Generator(), '/')
          cherrypy.engine.start()
          cherrypy.engine.block()
```

hello_form.html

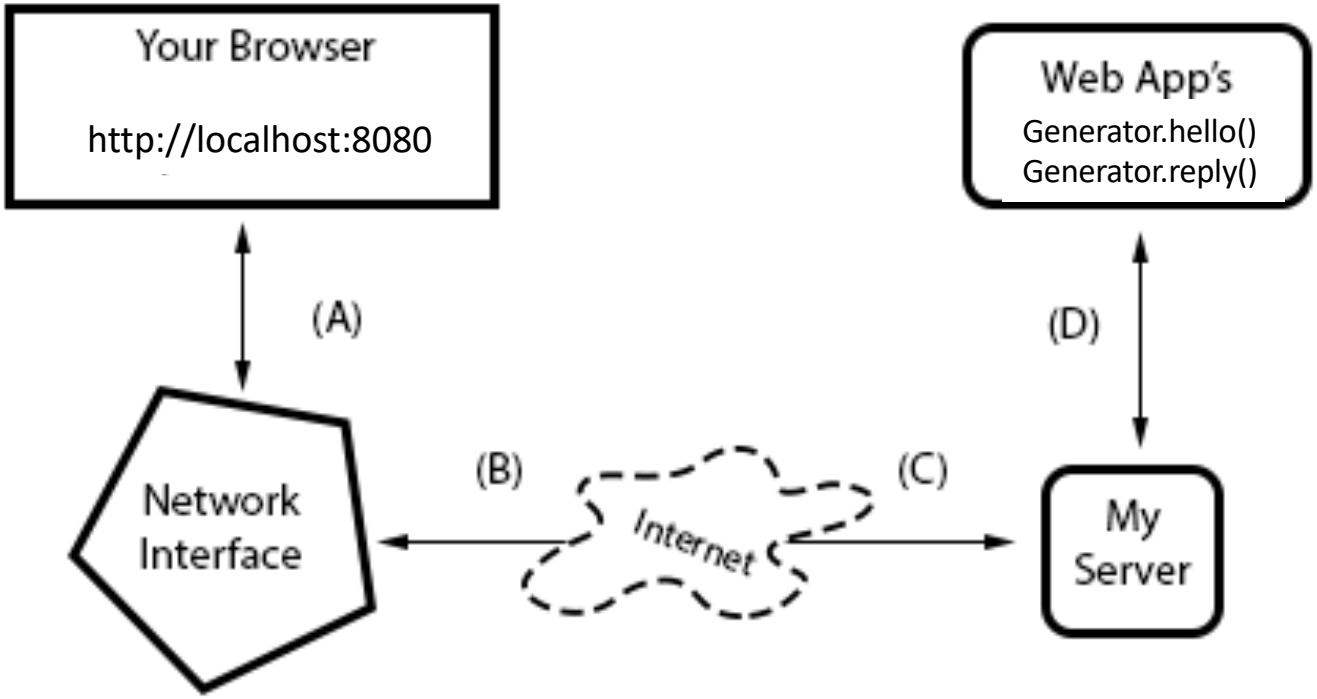
```
<html>
    <head>
        <title>Sample Web Form</title>
    </head>
<body>

<h1>Fill Out This Form</h1>

<form action=”/reply" method="POST">
    A Greeting: <input type="text" name="greet">
    <br/>
    Your Name: <input type="text" name="name">
    <br/>
    <input type="submit">
</form>

</body>
</html>
```

page.html

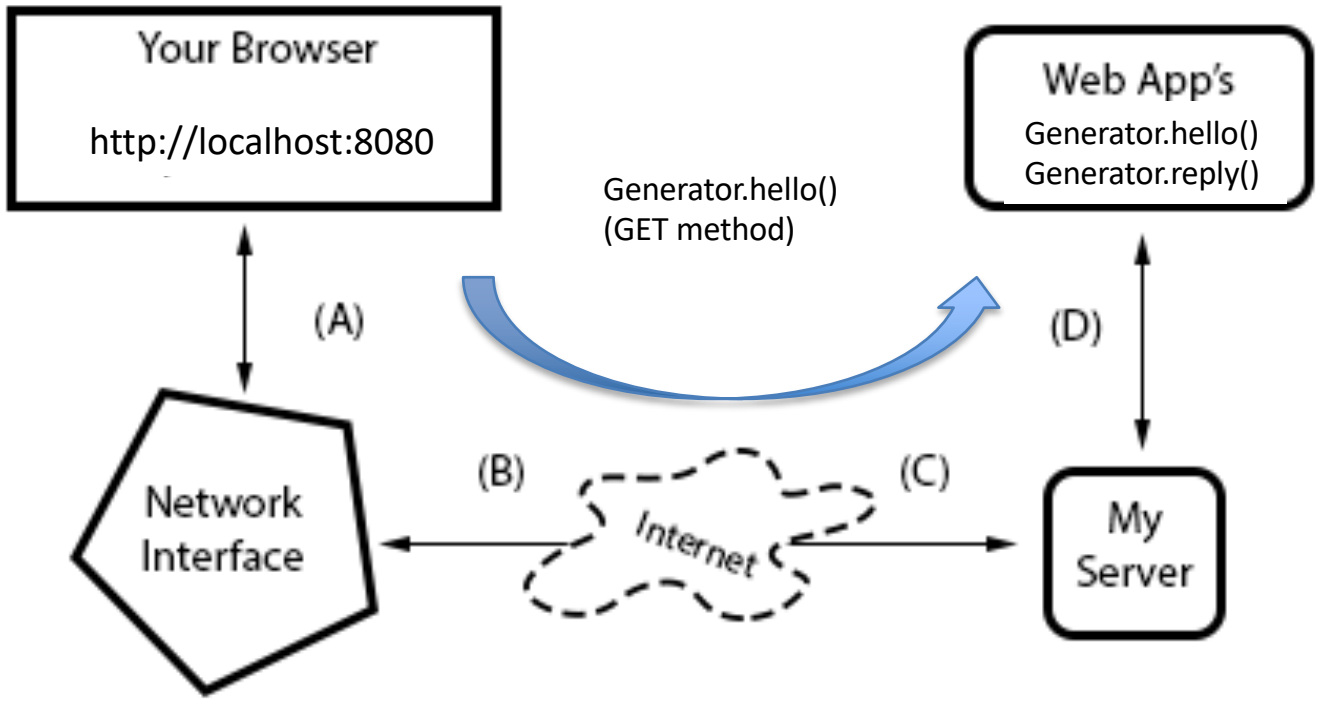
```
<html>
    <head>
        <title>Sample Web Form</title>
    </head>
<body>

<h1>FILLED thank you</h1>

</body>
</html>
```

http://localhost:8080/hello

How POST works



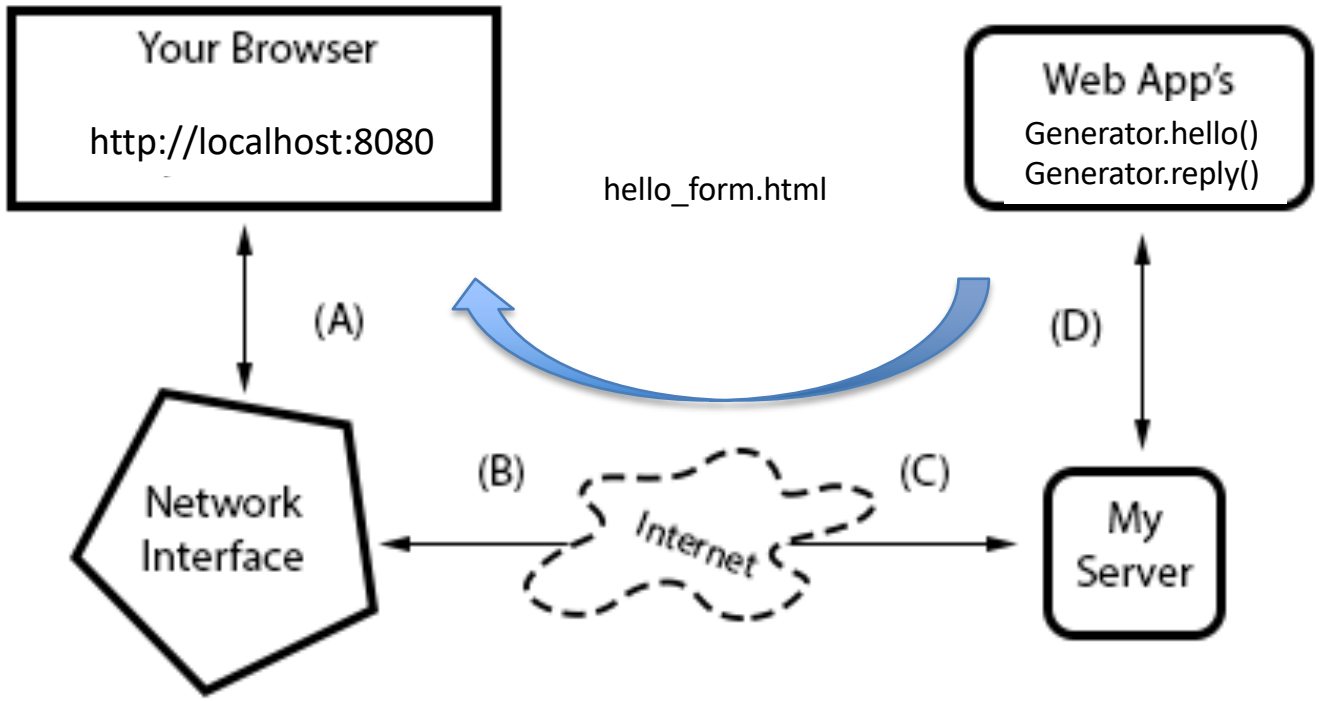
How POST works



Execution:
Generator.hello

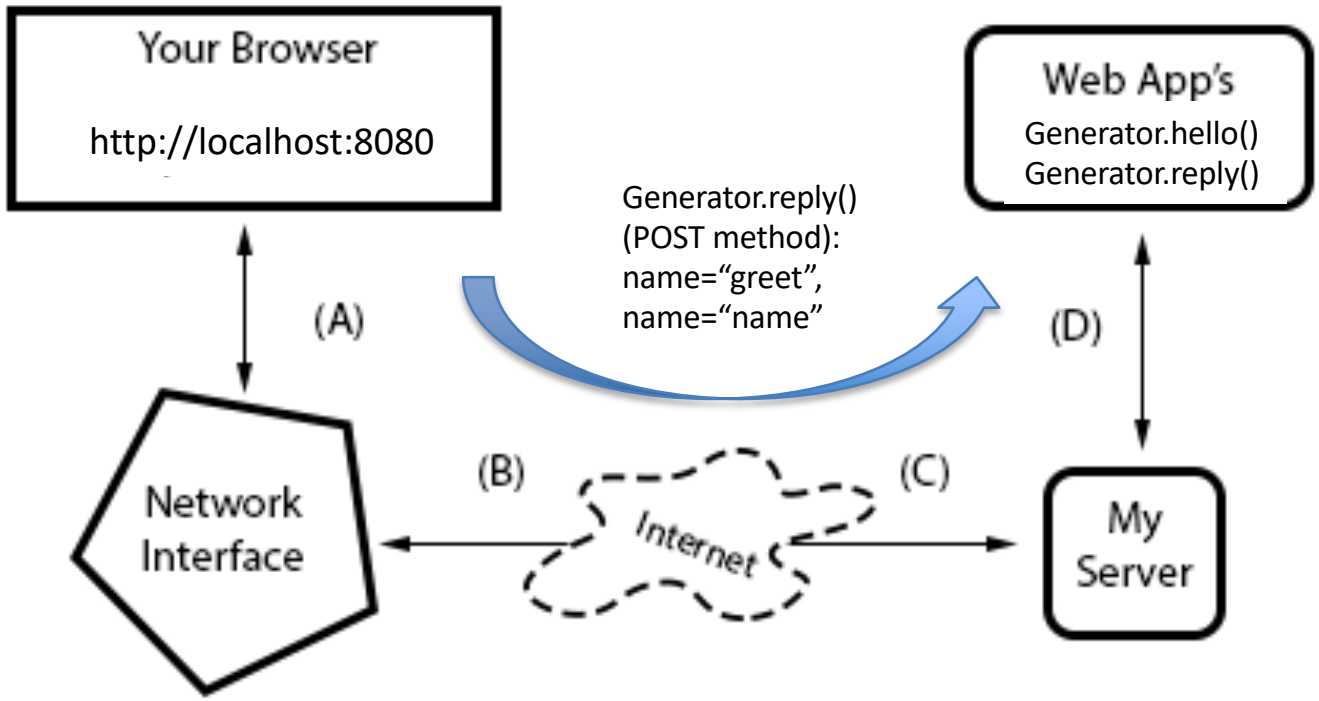
1. The browser first hits the web application at /hello but it sends a GET, so our Generator.hello() method runs and returns the hello_form.html

How POST works



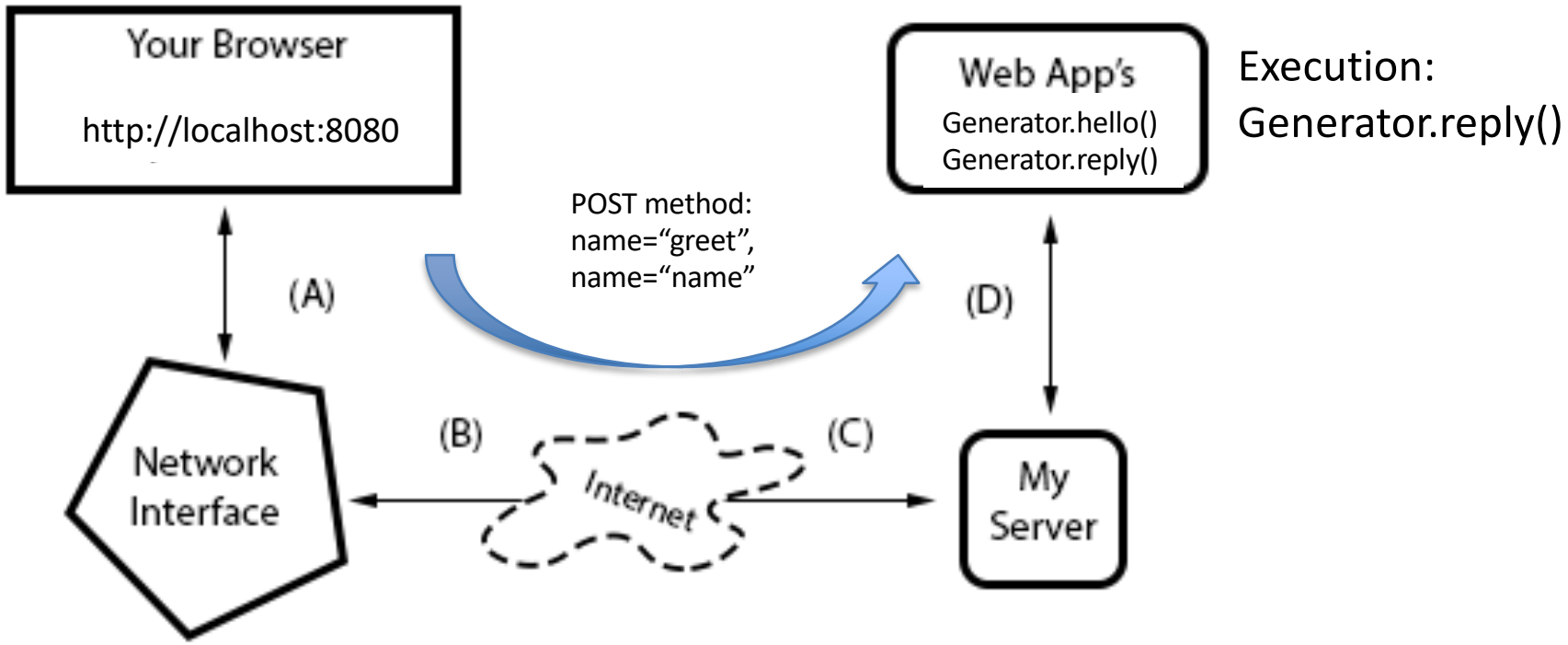
1. The browser first hits the web application at /hello but it sends a GET, so our Generator.hello() method runs and returns the hello_form.html

How POST works



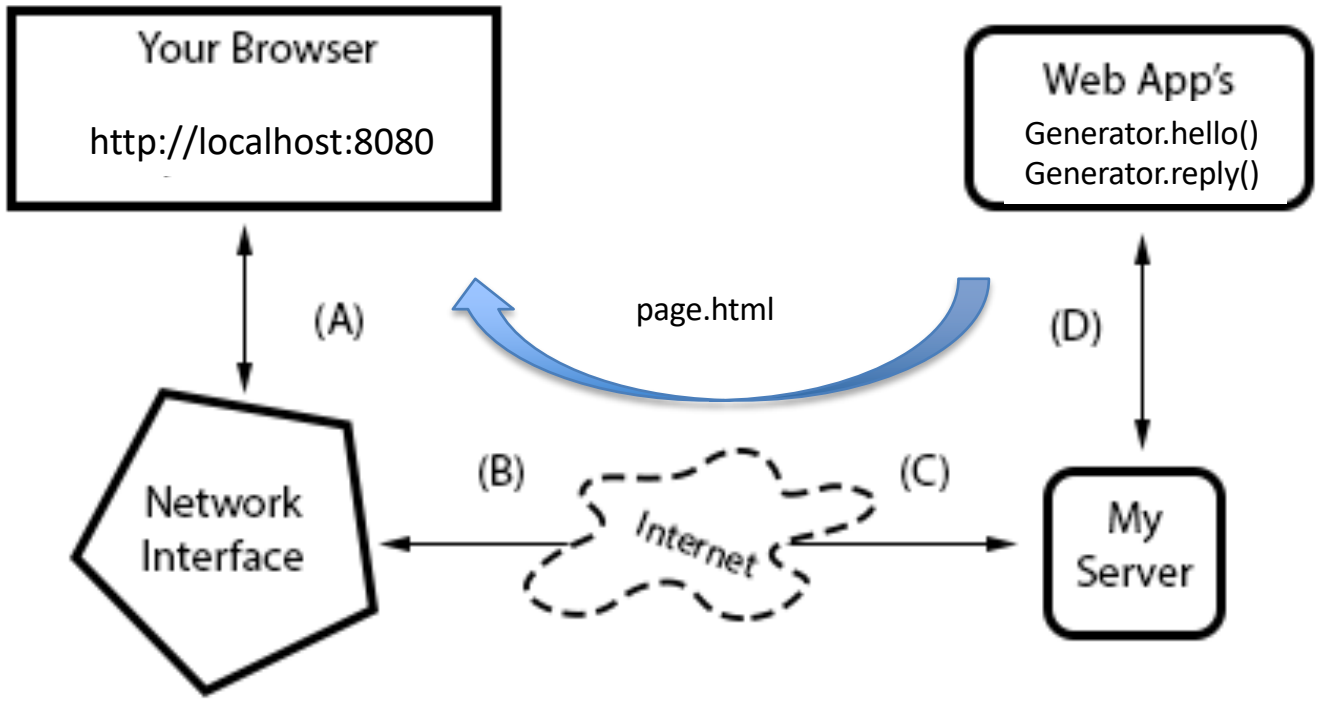
1. The browser first hits the web application at /hello but it sends a GET, so our Generator.hello() method runs and returns the hello_form.html
2. You fill out the form in the browser, and the browser does what the <form> says and sends the data as a POST.

How POST works



1. The browser first hits the web application at /hello but it sends a GET, so our Generator.hello() method runs and returns the hello_form.html
2. You fill out the form in the browser, and the browser does what the <form> says and sends the data as a POST.
3. The web application then runs the Generator.reply() method rather than the Generator.hello() to handle this request.

How POST works



1. The browser first hits the web application at /hello but it sends a GET, so our Generator.hello() method runs and returns the hello_form.html
2. You fill out the form in the browser, and the browser does what the <form> says and sends the data as a POST.
3. The web application then runs the Generator.reply() method rather than the Generator.hello() to handle this request.
4. Generator.reply() method analyses the POST request and responses with page.html

What is a Web **resource**? (1)

- A **resource** is any information that can be named, such as, a document, an image, etc..

What is a Web **resource**? (1)

- A **resource** is any information that can be named, such as, a document, an image, etc..
- **A resource has a name and an address represented by a URI**

What is a Web **resource**? (1)

- A **resource** is any information that can be named, such as, a document, an image, etc..
- **A resource has a name and an address represented by a URI**
- A resource is something that can be **stored on a computer and represented as a stream of bits** (e.g. a document, a row in a database, or the result of running an algorithm)

What is a Web **resource**? (2)

- A **representation** is defined as **a sequence of bytes, plus representation meta-data to describe those bytes**.
  - The client receives a representation when it requests a resource or sends one when it wishes to update a resource.

What is a Web **resource**? (3)

- What makes a resource a resource? **It has to have at least one URI.**

What is a Web **resource**? (3)

- What makes a resource a resource? **It has to have at least one URI.**
  - The **URI is the name and address of a resource**.
  - **URIs uniquely identify a resource**, regardless of its type and representation.
  - **If a piece of information does not have a URI, it is not a resource** and it is not really on the Web

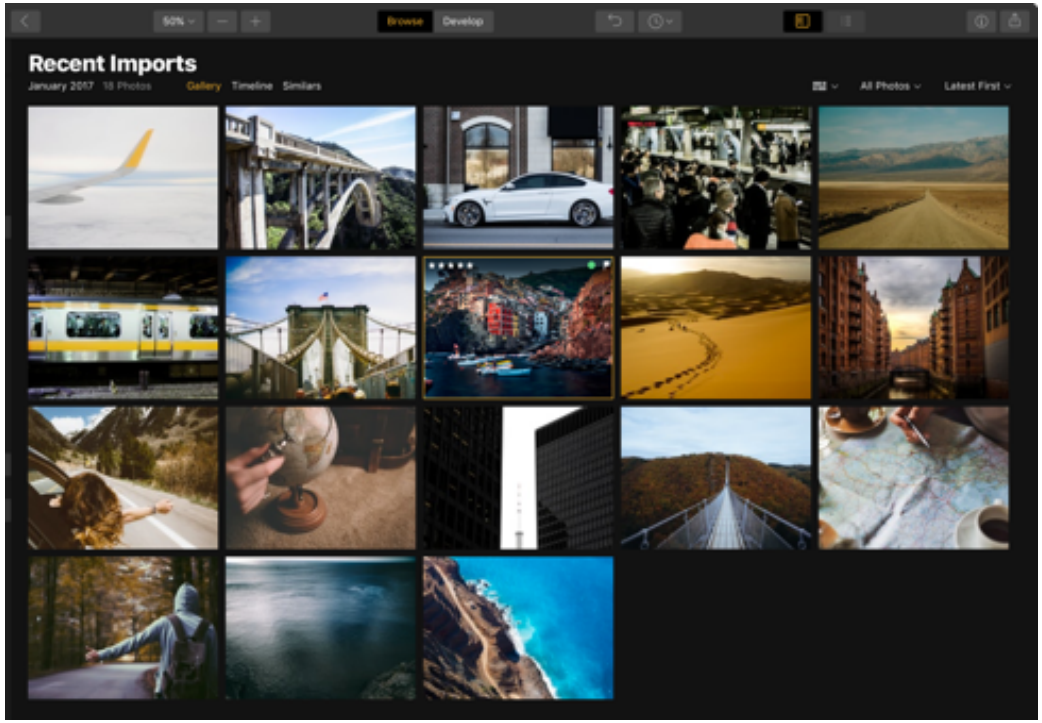
Addressability

A web application is **addressable** if it exposes the interesting aspects of its data set as resources.

Since resources are exposed through URIs, an addressable application exposes a URI for every piece of information it might conceivably serve

Example

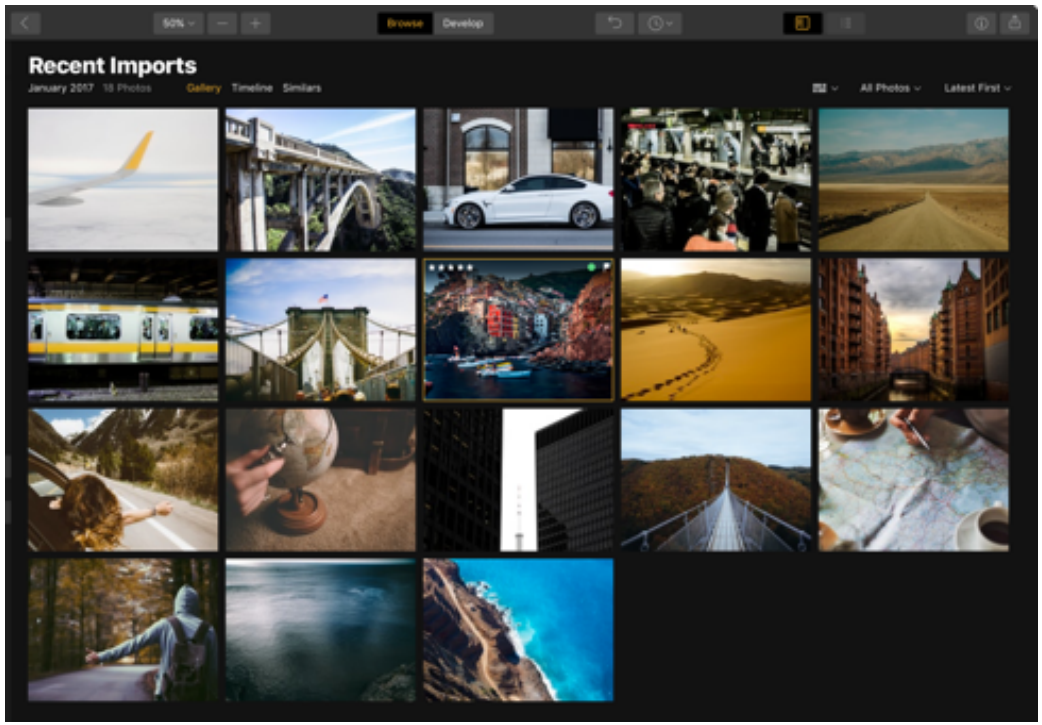
Given an addressable application to manage photo albums hosted at http://www.example.com and the photo album ‘2008Holiday’



Example

Given an addressable application to manage photo albums hosted at http://www.example.com and the photo album ‘2008Holiday’

A possible URI for the album “2008Holiday” would be http://www.example.com/2008Holiday



Example

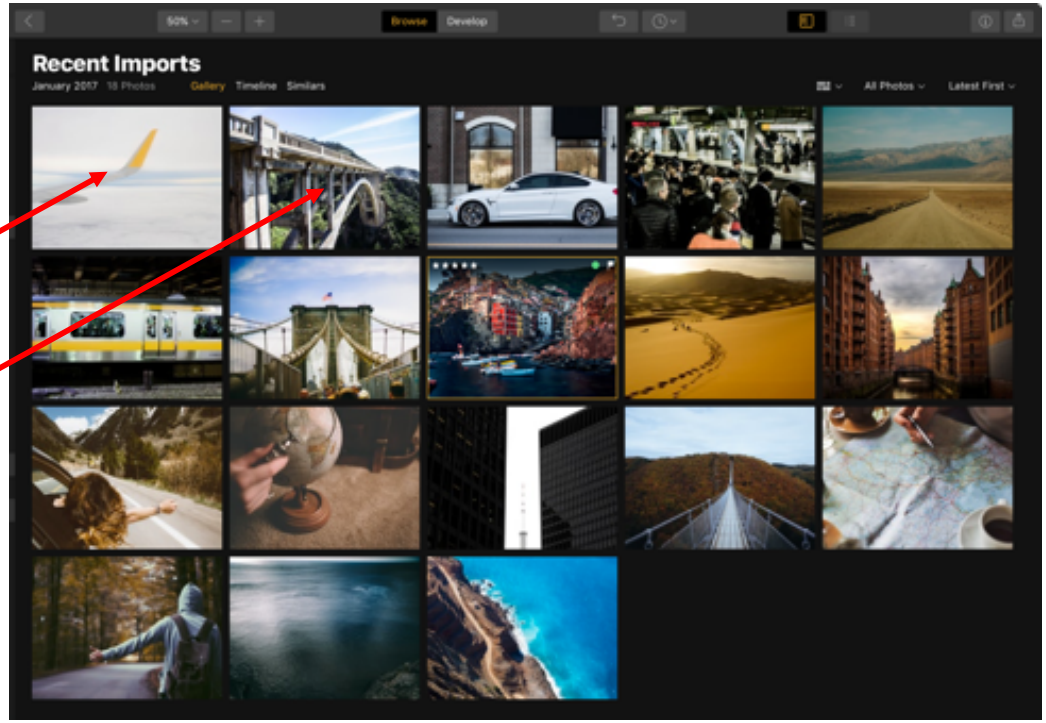
Given an addressable application to manage photo albums hosted at http://www.example.com and the photo album ‘2008Holiday’

A possible URI for the album “2008Holiday” would be http://www.example.com/2008Holiday

The photo album representation would include the URIs of the photos (resources) that belong to it

http://www.example.com/2008Holiday/1.jpeg

http://www.example.com/2008Holiday/2.jpeg

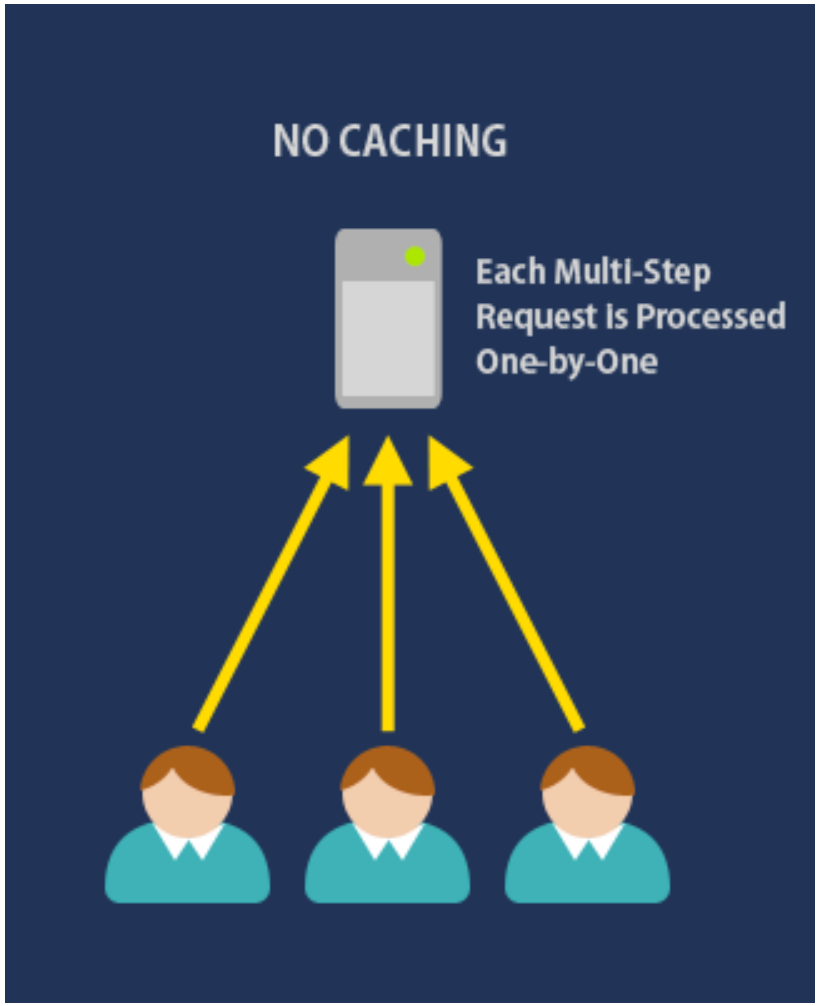


Cache (computing)

From Wikipedia, the free encyclopedia

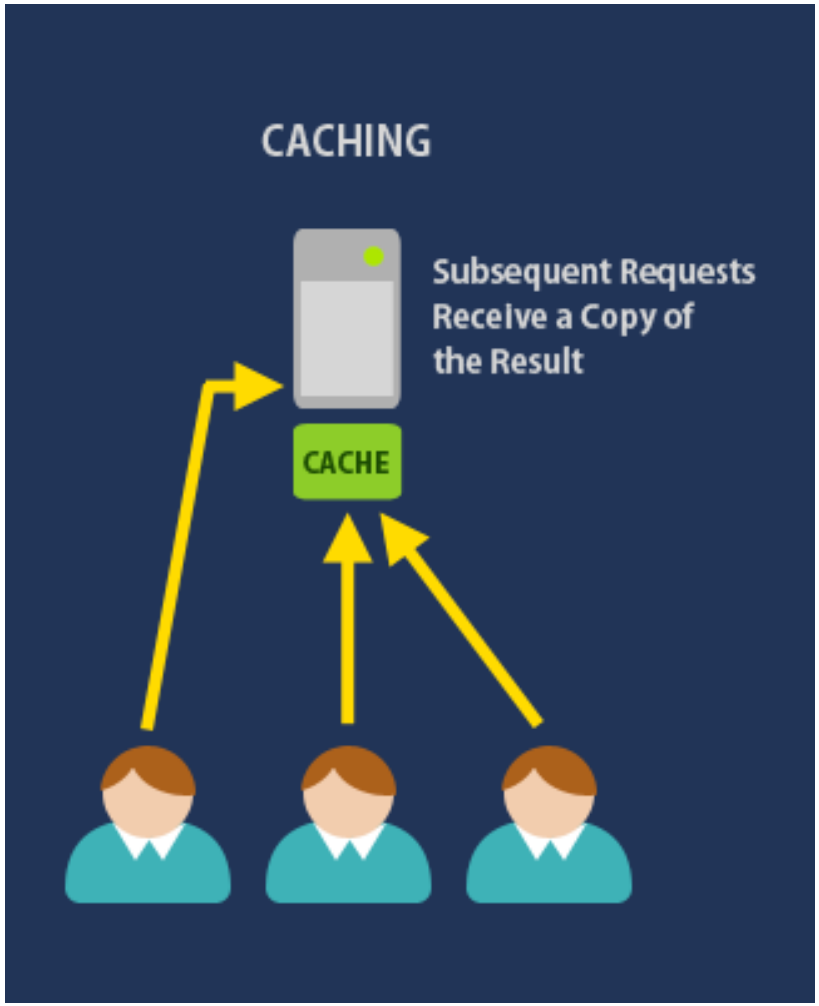
In computing, a **cache** (/kæʃ/ (listen) *kash*,[1] or /ˈkeɪʃ/ *kaysh* in AuE[2]) is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere. A *cache hit* occurs when the requested data can be found in a cache, while a *cache miss* occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests that can be served from the cache, the faster the system performs.

Cacheability



Cacheability

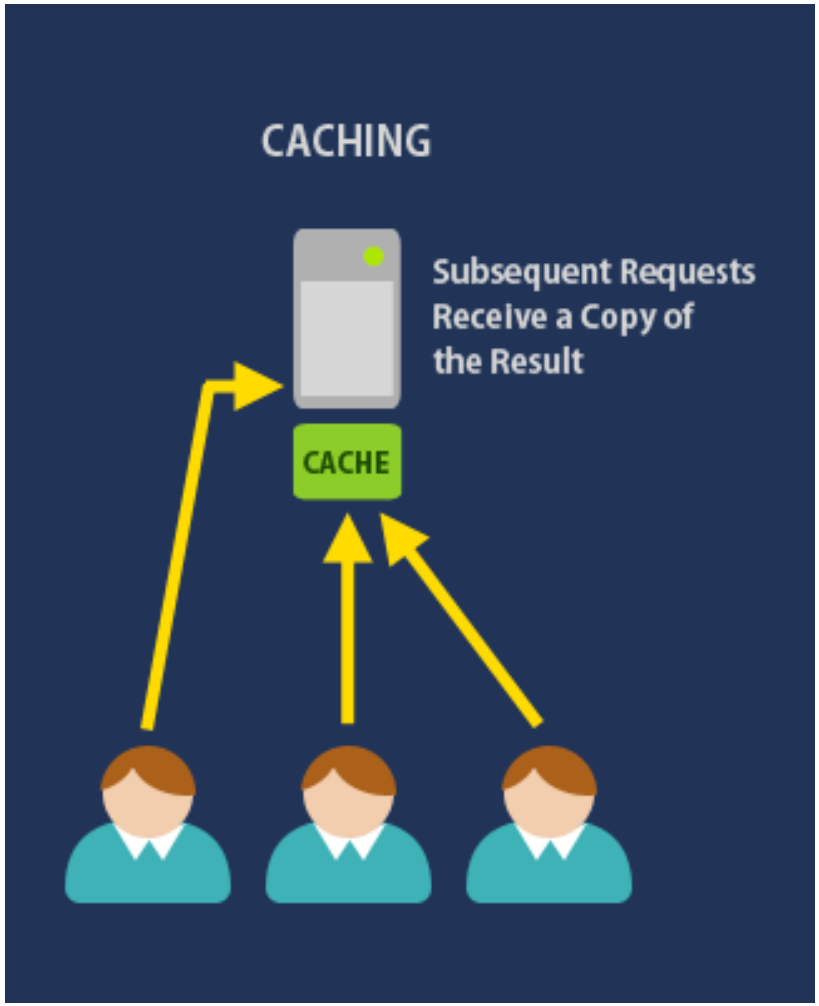
To save bandwidth, you can set up an **HTTP proxy cache** on your local network



- The first time someone requests http://www. example.com/2008Holiday the cache will save a local copy of the document
- The next time someone hits that URI, the cache might serve the saved copy instead of downloading it again

Cacheability

To save bandwidth, you can set up an **HTTP proxy cache** on your local network



- The first time someone requests http://www. example.com/2008Holiday the cache will save a local copy of the document
- The next time someone hits that URI, the cache might serve the saved copy instead of downloading it again

These feature can be enabled only if every resource has a unique identifying string i.e. URI


Cacheability
Gimme all your cache!!

HTTP session state

- HTTP is a **stateless protocol**.

HTTP session state

- HTTP is a **stateless protocol**.

From Wikipedia

In computing, a **stateless protocol** is a communications protocol in which no session information is retained by the receiver, usually a server. Relevant session data is sent to the receiver by the client in such a way that every packet of information transferred can be understood in isolation, without context information from previous packets in the session. This property of stateless protocols makes them ideal in high volume applications, increasing performance by removing server load caused by retention of session information.

HTTP session state

- HTTP is a **stateless protocol**.

From Wikipedia

In computing, a **stateless protocol** is a communications protocol in which no session information is retained by the receiver, usually a server. Relevant session data is sent to the receiver by the client in such a way that every packet of information transferred can be understood in isolation, without context information from previous packets in the session. This property of stateless protocols makes them ideal in high volume applications, increasing performance by removing server load caused by retention of session information.

A stateless protocol does not require the server to retain session information or status about each communicating partner for the duration of multiple requests. In contrast, a protocol that requires keeping of the internal state on the server is known as a stateful protocol.

HTTP session state

- HTTP is a **stateless protocol**.

From Wikipedia

In computing, a **stateless protocol** is a communications protocol in which no session information is retained by the receiver, usually a server. Relevant session data is sent to the receiver by the client in such a way that every packet of information transferred can be understood in isolation, without context information from previous packets in the session. This property of stateless protocols makes them ideal in high volume applications, increasing performance by removing server load caused by retention of session information.

A stateless protocol does not require the server to retain session information or status about each communicating partner for the duration of multiple requests. In contrast, a protocol that requires keeping of the internal state on the server is known as a stateful protocol.

The stateless design simplifies the server design because there is no need to dynamically allocate storage to deal with conversations in progress. If a client session dies in mid-transaction, no part of the system needs to be responsible for cleaning up the present state of the server. A disadvantage of statelessness is that it may be necessary to include additional information in every request, and this extra information will need to be interpreted by the server.

HTTP session state

- HTTP is a **stateless protocol**.
- A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests.

HTTP session state

- HTTP is a **stateless protocol**.
- A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests.
- Every HTTP request happens in **complete isolation**

HTTP session state

- HTTP is a **stateless protocol**.
- A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests.
- Every HTTP request happens in **complete isolation**
- Some web applications implement states or server side sessions using for instance HTTP **cookies** or hidden variables within web forms.

HTTP session state

- HTTP is a **stateless protocol**.
- A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests.
- Every HTTP request happens in **complete isolation**
- Some web applications implement states or server side sessions using for instance HTTP **cookies** or hidden variables within web forms.

From Wikipedia

An **HTTP cookie** (also called **web cookie**, **Internet cookie**, **browser cookie**, or simply **cookie**) is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing. Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited in the past).

HTTP session state

- HTTP is a **stateless protocol**.
- A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests.
- Every HTTP request happens in **complete isolation**
- Some web applications implement states or server side sessions using for instance HTTP **cookies** or hidden variables within web forms.
  - When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request

HTTP session state

- HTTP is a **stateless protocol**.
- A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests.
- Every HTTP request happens in **complete isolation**
- Some web applications implement states or server side sessions using for instance HTTP **cookies** or hidden variables within web forms.
  - When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request
- **The server never relies on information from previous requests**
  - If that information was important, the client would have sent it again in this request

Why is HTTP stateless? (1)

- **State** would make individual HTTP requests simpler, but it would make the **HTTP protocol much more complicated**
- To eliminate state from a protocol is to **eliminate** a lot of **failure conditions**
- **The server never has to worry about the client timing out**, because no interaction lasts longer than a single request
- The server never loses track of “where” each client is in the application, because **the client sends all necessary information with each request**
- The client never ends up performing an action in the wrong “working directory” due to the server keeping some state around without telling the client

Why is HTTP stateless? (2)

- Statelessness also brings **new features**:
  - It is **easier to distribute** a stateless application **across load-balanced servers**
  - Since **two requests** do not depend on each other, they can be **handled by two different servers** that never coordinate with each other
  - **Scaling up** is as simple as plugging more servers into the load balancer
  - A stateless application is also **easy to cache**: a piece of software can decide whether or not to cache the result of an HTTP request just by looking at that one request
  - The state from a previous request will not affect the cacheability of this one

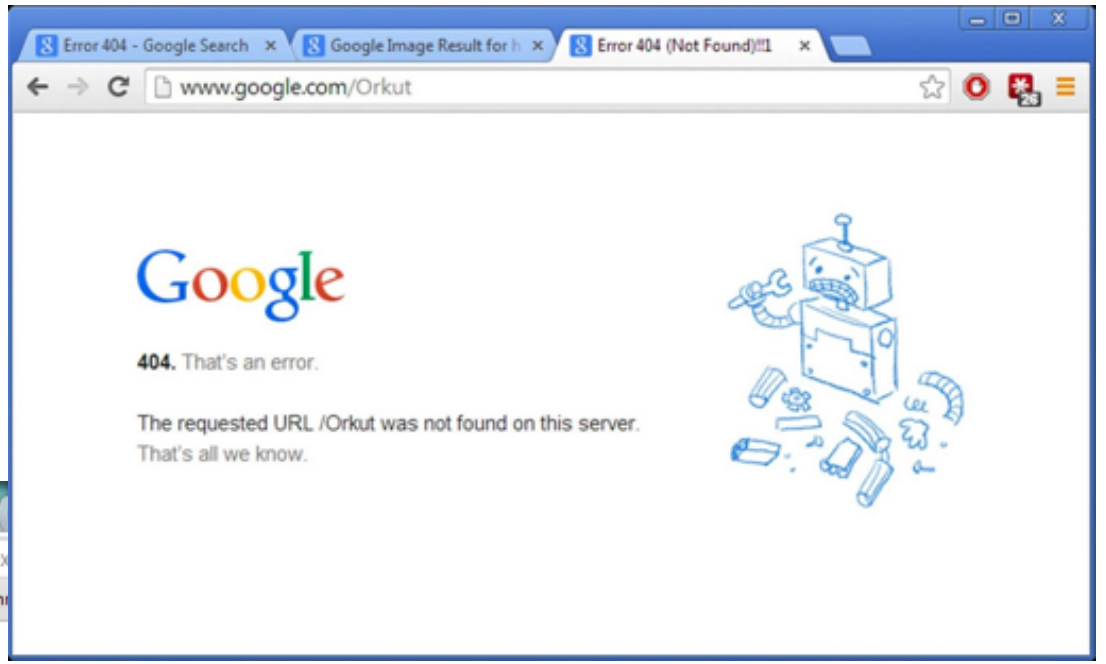
Why is HTTP stateless? (3)

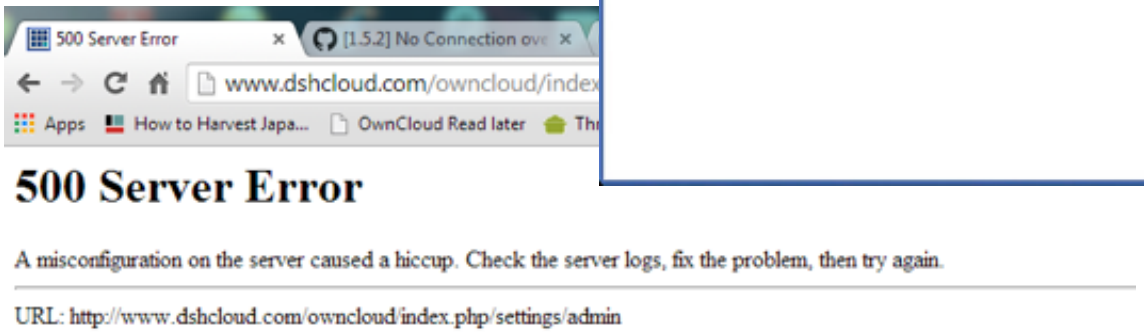
- The client benefits from statelessness as well:
  - A client can get and process information from a URI at any time
  - A URI that works when you are hours deep into an HTTP session will work the same way as the first URI sent in a new session

HTTP status codes

HTTP provides status codes for defining the outcome of the communication. They are organized in five classes:

- **1xx** – Informational
- **2xx** – Successful
- **3xx** – Redirection
- **4xx** – Client Error
- **5xx** – Server Error





HTTP status codes: 2xx – Sucessful

This class indicates that the client's request was successfully received, understood, and accepted.

Code	Name	Description
200	OK	The request has succeeded.
201	Created	The request has been fulfilled and resulted in a new resource being created.
202	Accepted	The request has been accepted for processing, but the processing has not been completed.
…	…	…

HTTP status codes: 4xx – Client Error

This class is intended for cases in which the client seems to have erred. The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Code	Name	Description
400	Bad Request	The request could not be understood by the server due to malformed syntax.
401	Unauthorized	The request requires user authentication.
403	Forbidden	The server understood the request, but is refusing to fulfill it.
404	Not Found	The server has not found anything matching the Request-URI.
…	…	…

HTTP status codes: 5xx – Server Error

This class indicates cases in which the server is aware that it has erred or is incapable of performing the request. The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Code	Name	Description
500	Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
501	Not Implemented	The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.
503	Service Unavailable	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.
…	…	…


REST Web Services

Representational state transfer

From Wikipedia, the free encyclopedia

Representational state transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

Representational state transfer

From Wikipedia, the free encyclopedia

Representational state transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

In a RESTful Web service, requests made to a resource's URI will elicit a response with a payload formatted in HTML, XML, JSON, or some other format. The response can confirm that some alteration has been made to the stored resource, and the response can provide hypertext links to other related resources or collections of resources. When HTTP is used, as is most common, the operations (HTTP methods) available are GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS

Representational state transfer

From Wikipedia, the free encyclopedia

Representational state transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

In a RESTful Web service, requests made to a resource's URI will elicit a response with a payload formatted in HTML, XML, JSON, or some other format. The response can confirm that some alteration has been made to the stored resource, and the response can provide hypertext links to other related resources or collections of resources. When HTTP is used, as is most common, the operations (HTTP methods) available are GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS

By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow by reusing components that can be managed and updated without affecting the system as a whole, even while it is running.

REST Web Services

Roy Fielding defined REST in his 2000 PhD dissertation *"Architectural Styles and the Design of Network-based Software Architectures"*. He developed the REST architectural style in parallel with HTTP 1.1

REST Web Services

Representational state transfer *(REST) or* ***RESTful*** *web services is a way of providing interoperability between computer systems on the Internet following a* ***Client-server approach***

REST Web Services

Representational state transfer *(REST) or* ***RESTful*** *web services is a way of providing interoperability between computer systems on the Internet following a* ***Client-server approach***

REST-compliant Web services allow requesting systems to ***access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations****.*

REST properties (1)

- **Scalability** allowing the support of large numbers of components and interactions among components

REST properties (1)

- **Scalability** allowing the support of large numbers of components and interactions among components
- **Simplicity** of a uniform interface

REST properties (1)

- **Scalability** allowing the support of large numbers of components and interactions among components
- **Simplicity** of a uniform interface
- **Modifiability** of components to meet changing needs (even while the application is running)

REST properties (1)

- **Scalability** allowing the support of large numbers of components and interactions among components
- **Simplicity** of a uniform interface
- **Modifiability** of components to meet changing needs (even while the application is running)
- **Visibility** of communication between components by service agents

REST properties (1)

- **Scalability** allowing the support of large numbers of components and interactions among components
- **Simplicity** of a uniform interface
- **Modifiability** of components to meet changing needs (even while the application is running)
- **Visibility** of communication between components by service agents
- **Portability** of components by **moving program code with the data**

REST properties (2)

- **Statelessness**

REST properties (2)

- **Statelessness**
- **Cacheability**
  - Responses must define themselves as cacheable or not to prevent clients from getting stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance

REST properties (2)

- **Statelessness**
- **Cacheability**
  - Responses must define themselves as cacheable or not to prevent clients from getting stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance
- **Layered system**
  - If a proxy or load balancer is placed between the client and server, it would not affect their communications. Intermediary servers can **improve system scalability** by enabling load balancing and by providing shared caches.

REST properties (2)

- **Statelessness**
- **Cacheability**
  - Responses must define themselves as cacheable or not to prevent clients from getting stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance
- **Layered system**
  - If a proxy or load balancer is placed between the client and server, it would not affect their communications. Intermediary servers can **improve system scalability** by enabling load balancing and by providing shared caches.
- **Uniform interface**
  - It simplifies and decouples the architecture, which **enables each part to evolve independently**.

REST – Methods

- **HTTP Methods are a key corner stone in REST**
- They define the actions to be taken with a URL
- Proper **RESTful** webservices **expose all CRUD methods** (Create, Read, Update, Delete)

HTTP	REST - CRUD
POST	Create
GET	Read
PUT	Update or Modify or Replace
DELETE	Delete