14_internet_la_web_y_requests

October 15, 2020

1 Internet, la World Wide Web y el paquete `requests`.

Este capítulo trata sobre los concepto básicos sobre Internet y la "World Wide Web" desde un punto de vista técnico con la finalidad de poder tener los conocimientos necesarios para el desarrollo de aplicaciones y servicios web básica.

1.1 Breve introducción a Internet.

Internet es una red de comunicaciones global basada en la familia de protocolos de internet, también conocida como el conjunto de protocolos *TCP/IP*. A través de Internet y sus diversos protocolos es posible acceder a servicios tan diversos como:

- Correo electrónico.
- Mensajería instantánea y chats.
- Comercio electrónico.
- Telefonia.
- Transmisión de audio y video.
- Acceso a la web.

Los protocolos *TPC/IP* fueron desarrollados inicialmente por Vinton Cerf y Robert Kahn en la década de 1970 para el proyecto ARPANET, el cual posteriormente evolucionó en Internet.

Nota: En este capítulo se mencionarán brevemente los protocolos *IP*, *DNS*, *SSL/TLS* y *HTTPS*, mientras que el protocolo *HTTP* será el tema central del curso.

1.1.1 El modelo *OSI*.

El modelo *OSI*, por las siglas en inglés de "modelo de interconexión de sistemas abiertos" correspondiente al estándar *ISO/IEC 7498-1* es un modelo de referencia que facilita la comunicación e interacción entre diversos componentes de una red de comunicaciones de datos.

El modelo *OSI* consta de 7 capas:

1. Capa física.

- Capa de enlace.
- Capa de red.
- Capa de transporte.
- Capa de sesión.
- Capa de presentación.
- Capa de aplicación.

Cada uno de los protocolos de la familia *TCP/IP* es implementado en alguna de estas capas.

Para mayor referencia sobre los protocolos *TCP/IP* y su implementación en las capas del modelo *OSI* es posible consultar la siguiente liga.

https://en.wikipedia.org/wiki/List_of_network_protocols_(OSI_model)

1.2 Localizadores Uniformes de Recursos (*URL*).

Es posible acceder a un recurso disponble en Internet mediante los "Localizadores Uniformes de Recursos, o *URL* por sus siglas en inglés.

1.2.1 Estructura de una *URL*.

Una *URL* presenta la siguiente estructura general:

Para un recurso que requiere de autenticación:

```
<esquema>://<usuario>:<contraseña>@<anfitrión>:<puerto>/<ruta>?<parámetro 1>&<parámetro 2>&...&
```

Para un recurso que no requiere de autenticación:

```
<esquema>://<anfitrión>:<puerto>/<ruta>?<parámetro 1>&<parámetro 2>&...&<parámetro n>
```

Donde:

- `<protocolo>` es el protocolo que se utilizará para acceder al recurso.
  - Para *HTTP* se indica `http`.
  - Para *HTTPS* se indica `https`.
- `<usuario>` es el nombre del usuario con el que se accederá al recurso en el caso de que el acceso requiera de autenticación.
- `<contraseña>`es la contraseña del usuario con el que se accederá al recurso en el caso de que el acceso requiera de autenticación.
- `<anfitrión>` indica la dirección en la que se encuentra el sistema anfitrión (host) al que se quiere acceder.
- `<puerto>` es el puerto en el que el servicio está habilitado.
  - *HTTP* utiliza el puerto `80` por defecto.
  - *HTTPS* utiliza el puerto `443` por defecto.
- `<ruta>` es la ruta donde se encuentra localizado el recurso dentro del servidor.
- `<parámetro n>` es un par `<identificador>=<valor>` que da cierta información que podría ser procesada por el servidor.

Ejemplo:

- La siguiente *URL* realizará lo siguiente:
  - Realizará una conexión *HTTPS* en el puerto 443 con el sistema anfitrión localizado en `www.google.com`.
  - buscará al recurso localizado en la ruta `search`.
  - Enviará el parámetro `q=cloudevel`.

https://www.google.com/search?q=cloudevel

Al ingresar la *URL* desde un navegador, el resultado será la búsqueda del término `cloudevel` en Google.

1.3 Acceso a un anfitrión (host).

Los sistemas anfitriones son equipos o sistemas de cómputo los cuales están distribuídos por todo el mundo y cuyos servicios pueden ser accedidos de dos maneras:

- Mediante su dirección *IP*.
- Mediante su nombre de dominio.

1.3.1 Direcciones *IP*.

Una dirección *IP* es un número único que identifica a un nodo de red de un servidor específico. Un servidor puede tener más de un nodo de red, por lo que cada nodo tendría una direccion *IP* diferente y única.

La "Internet Asigned Numbres Authority" (*IANA*) es la organización encargada de asignar y distribuir los segmentos de direcciones *IP* a las diversas organizaciones privadas y gubernamentales.

Direcciones *IP* versión 4 (*IPv4*). La especificación de los identificadores *IPv4* fue definida por la "Internet Engineering Task Force" (*IETF*) en el *RFC 6864*.

Las direcciones *IPv4* son las de uso más común. Corresponden a un número de 32 bits expresado por 4 segmentos de números que van de `0` a `255` (8 bits) separados por un punto `.`.

Ejemplo:

La dirección *IPv4* donde se encuentra el servidor de Cloudevel® es `142.4.217.131`.

Debido al auge de Internet, a partir del año 2011 la reserva de direcciones *IPv4* se ha agotado. Sin embargo, aún cuando las reservas están agotadas, todavía es posible reutilizar y administrar las direcciones *IPv4* existentes.

Direcciones *IP* versión 6 (*IPv6*). A partir de la necesidad de contar con un mayor número de direcciones *IP* disponibles, fue publicada la especificación de los identificadores *IPv6*, la cual fue definida en el *RFC 3513*.

Las direcciones *IPv6* son un número de 128 bits expresado por 8 segmentos de números hexadecimales que van de `0` a `FFFF` (`65535`), separados por dos puntos `:`. Los valores iguales a `0` pueden ser omitidos.

Ejemplo:

La dirección *IPv6* donde se encuentra el servidor de Cloudevel® es `fe80::a6bf:1ff:fe08:aee4`.

1.3.2 Las "intranets" y los rangos de direcciones *IP* privados..

Una intranet es una red privada basada en el conjunto de protocolos *TCP/IP*, en la que a cada interfaz de red de los sistemas locales se le asigna una dirección *IP* en un rango privado y sólo es posible acceder a Internet por medio de un ruteador.

Los rangos de direcciones *IP* privados son un conjunto de direcciones que pueden ser asignados a una intranet sin que existan colisiones con las direcciones *IP* públicas.

Los segmentos de red para direcciones *IP* privadas son:

- De `10.0.0.0` a `10.255.255.255`.

- De `172.16.0.0` a `172.31.255.255`.
- De `192.168.0.0` a `192.168.255.255`.

1.3.3 *DNS*, dominios y subdominios.

Aún cuando las direcciones *IP* pueden identificar con precisión a un host, no son fáciles de memorizar por parte de los seres humanos. Es por ello que fueron creados los "nombres de dominio" y el "sistema de nombres de dominio" o *DNS* por sus siglas en inglés.

La "Corporación de Internet para la Asignación de Nombres y Números" o *ICANN* por sus siglas en inglés, es la organización encargada en definir y asignar y distribuir los nombres de dominos válidos de Internet.

Gracias al *DNS* es posible relacionar uno o más nombres de dominio a una dirección *IP* utilizando una estructura como la siguiente.

```
<subdominio>.<nombre de dominio>
```

Donde:

- `<nombre de dominio>` es un nombre de domino válido que apunta a una dirección IP.
- `<subdominio>` es una extensión del nombre de dominio que puede apuntar a una dirección *IP* distinta a la del dominio. Los subdominios permiten ofrecer servicios diferenciados dentro de un mismo dominio.

Ejemplos:

- `google.com` es el dominio propiedad de la empresa Google.
- `www.google.com` apunta al servicio de búsqueda de Google.
- `docs.google.com` apunta al servicio de gestión de documentos de Google.
- `mail.google.com` apunta al servicio de correo electrónico de Google.

Los nombres de dominio pueden ser adquiridos mediante un "registrar" autorizado por *ICANN*, el cual por lo general también ofrece servicios de *DNS*.

Una vez adquirido un dominio es posible definir cualquier número de subdominios relacionados mediante el proveedor de servicios *DNS*.

Dominios de nivel superior (*TLD*). Un nombre de dominio se compone de un nombre y un "dominio de nivel superior" o *TLD*, por sus siglas en inglés, de la siguiente forma.

```
<nombre>.<TLD>
```

Los TLD se utilizan primordialmente para: * Definir la actividad de una organilzación, como es el caso de: * `.com` para organizaciones comerciales. * `.edu` para organizaciones educativas. * `.net` para organizaciones especializadas en comunicaciones de red. * `.org` para organizaciones sin fines de lucro. * `.mil` para organizaciones militares. * Definir la región o país al que pertence la organización, como es el caso de: * `.mx` para México. * `.es` para España. * `.io` para la región del Oceano Índico. * `.lat` para Latinoamérica. * Definir el tipo de servicio que se ofrece, como es el caso de: * `.info` para servicios de información. * `.xxx` para servicios de entretenimiento para adultos. * `mobi` para aplicaciones móviles.

La gestión de cada *TLD* puede ser asignada a una organización en particular, la cual fija las reglas para poder adquirir un dominio con dicho *TLD*.

Los *TLD* pueden combinarse de tal forma que describan una actividad en una región.

Ejemplos:

- `google.com` es el dominio primario de Google.
- `google.com.mx` es el dominio de Google para México.
- `google.com.es` es el dominio de Google para España.

1.3.4 *localhost.*

Es común que al desarrollar una aplicación web o diseñar un sitio web se haga desde una máquina de escritorio la cual ofrecería un servicio web local para consulta interna.

En la mayoría de los sistemas, el nombre `localhost` permite conectarse a un servicio que corre en el mismo equipo sin necesidad de contar con una interfaz de red. Este nombre por lo general está relacionado con la dirección *IP* `127.0.0.1`.

Por seguridad, muchas herramientas de desarrollo de aplicaciones web están configuradas por defecto para conectarse exclusivamente a `localhost`.

Ejemplo:

En caso de estar viendo este contenido desde una notebook de Jupyter corriendo desde un equipo propio, es muy probable que el navegador apunte a `localhost:8888`.

1.4 Los protocolos *HTTP* y *HTTPS.*

1.4.1 El protocolo *HTTP.*

HTTP es una especificación que forma parte de la familia de protocolos de Internet y su nombre se refiere a las siglas en inglés de "Protocolo de Transferencia de Hipertexto".

Creado por Tim Berners-Lee en 1989, es el protocolo utilizado para acceder y publicar en la *World Wide Web*, o simplemente la web.

Actualmente el World Wide Web Consortium (*W3C*) es la entidad encargada, entre otras cosas, de publicar la especificación del protocolo *HTTP*. La versión más reciente es *HTTP/2*, pero la mayoría de los servidores utilizan la versión *HTTP/1.1*.

Para saber mas sobre *HTTP* se puede consultar el siguiente enlace:

https://developer.mozilla.org/es-ES/docs/Web/HTTP

Características de *HTTP.* *HTTP* está basado en una arquitectura cliente-servidor en la que se intercambian peticiones (requests) por parte del cliente y respuestas (responses) por parte del servidor y tiene las siguientes caracterísiticas.

- **Sin estado.** Es decir, que cada una de las transacciones request/response que se realizan no afectan al estado del cliente o del servidor, además de que cada transacción es totalmente independiente de el resto.
- **Independiente del contenido.** Aún cuando es muy común que un servidor *HTTP* entregue documentos en formato *HTML*, no existe restricción en el tipo de recurso al que se pueda acceder.

- **Sin conexión.** Una vez que la transacción petición/respuesta es terminada, la conexión entre cliente y servidor es destruida.

1.4.2 El protocolo *HTTPS*.

HTTPS es una variación de *HTTP* que permite cifrar las comunicaciones entre el cliente y el servidor por medio del protocolo *SSL/TLS*.

Organizaciones como Google y Mozilla dan preferencia a los sitios que tienen implementado *HTTPS* e incluso advierten que los sitios que no cuentan con dicha implementación son inseguros.

1.5 Conexiones *HTTP*.

Una conexion entre un cliente y un servidor web consiste en un intercambio de mensajes en ambas direcciones.

Las características de seguridad, duración y estado de estas conexiones son el resultado de una negociación entre el cliente y el servidor.

1.6 Mensajes *HTTP*.

Las comunicaciones entre el cliente y el servidor consisten en un serie de intercambios de mensajes.

- Los mensajes que envía un cliente a un servidor mediante una *URL* se conocen como "peticiones" (requests).
- Los mensajes que un servidor envía al cliente después de haber procesado la petición se conocen como "respuestas" (responses).

1.6.1 Estructura de un mensaje de *HTTP*.

Los mensajes de *HTTP* son conjuntos de datos trasnsmitidos en formato binario en codificación *ASCII*.

Tanto las peticiones como las respuestas tiene una estructura compuesta primordialmente por:

- Una línea de inicio, con la descripción general del mensaje.
- Una serie de encabezados (headers), los cuales corresponden a ciertos campos que definen las caracterśiticas del mensaje y que se encuentran definidos en la *RFC 4229*.
- Una línea en blanco.
- El cuerpo (body) del mensaje, el cual contienen los datos del mensaje.

Para conocer más con respecto a los mensajes *HTTP* es posible consultar el siguiente enlace:

https://developer.mozilla.org/es/docs/Web/HTTP/Messages

1.7 Métodos de *HTTP*.

El protocolo *HTTP* define métodos o "verbos", los cuales permiten realizar peticiones específicas entre un cliente y un servidor. Algunos de los métodos más utilizados son:

- `GET` se utiliza para obtener los datos de un recurso a partir de una *URL*. La información enviada mediante `GET` puede ser añadida a marcadores y puede ser regsitrada en las bitácoras del servidor.

- `HEAD` es similar al método `GET`, pero el servidor sólo proporcionará los encabezados respuesta y el mensaje de estado resultante.
- `POST` se utiliza para cear un recurso. Los datos enviados no son expuestos en la *URL* sino que son enviados dentro de la estructura de la petición.
- `PUT` se utiliza para sustituir un recurso existente y su estructura es similar a la de `POST`.
- `PATCH` es un método que se utiliza para modificar parcialmente un recurso.
- `DELETE` es un método que se utiliza para eliminar un recurso.

Existen algunos otros métodos como `OPTIONS`, `TRACE` y `CONNECT`, pero no están contemplados en el alcance de este taller. Puede consultar más al respecto puede acceder a https://developer.mozilla.org/es/docs/Web/HTTP/Methods.

1.7.1 Idempotencia.

Un método es 'idempotente' cuando no importan las veces que se envíe la misma petición, ésta dará el mismo resultado.

1.7.2 Seguridad.

Un método se considera seguro si no modifica los recursos a los que accede.

Método	Idempotente	Seguro
`GET`	Sí	Sí
`HEAD`	Sí	Sí
`DELETE`	Sí	No
`POST`	No	No
`PUT`	Sí	No
`PATCH`	No	No

1.7.3 Advertencia sobre los métodos seguros.

La seguridad de un método depende de su implementación y aún cuando se considera como una mala práctica, es posible que los métodos como `GET` sean capaces de modificar al recurso al que acceden.

1.8 Códigos de estado.

Los códigos de estado permiten informar al cliente sobre la manera en la que ha sido procesada la petición. Está conformado por un número entero de 3 dígitos. En caso de que la petición haya sido procesada exitosamente, el servidor regresa el número `200`.

1.8.1 Tipos por el número inicial:

- 1xx Información.
- 2xx Éxito.
- 3xx Redireccionamiento.
- 4xx Error del cliente.
- 5xx Error del servidor.

Puede consultar los mensajes de estado de *HTTP* en la siguiente liga: http://www.restapitutorial.com/httpstatuscodes.html

1.9 El paquete `requests`.

El paquete `requests` de Python permite iniciar una sesión con un servidor web, mediante funciones que emulan los métodos del protocolo *HTTP*, regresando un objeto de tipo `requests.models.Response`.

El paquete `requests` presenta funcionalidades avanzadas como autenticación, conexiones seguras, manejo de 'cookies', etc.

[ ]:
```
!pip install requests
```

[ ]:
```
import requests
```

1.9.1 Funciones de métodos de `requests`.

El paquete `requests` contiene varias funciones que pueden enviar peticiones a una *URL* utilizando un método en particular de la siguiente manera:

`<funcion de método>('<URL>', <kwargs>)`

Donde:

- `<URL>` es la *URL* de un recurso por el cual se hará la petición.
- `<kwargs>` son una serie argumentos que se ingresan en el formato `<nombre>=<valor>`.

Estas funciones pueden ser:

- `requests.get()`, la cual accederá a una *URL* utilizando el método `GET`.
- `requests.head()`, la cual accederá a una *URL* utilizando el método `HEAD`.
- `requests.post()`, la cual accederá a una *URL* utilizando el método `POST`.
- `requests.put()`, la cual accederá a una *URL* utilizando el método `PUT`.
- `requests.patch()`, la cual accederá a una *URL* utilizando el método `PATCH`.
- `requests.delete()`, la cual accederá a una *URL* utilizando el método `DELETE`.
- `requests.connect()`, la cual accederá a una *URL* utilizando el método `CONNECT`.
- `requests.options()`, la cual accederá a una *URL* utilizando el método `OPTIONS`.
- `requests.trace()`, la cual accederá a una *URL* utilizando el método `TRACE`.

Del mismo modo, al recibir la respuesta del servidor, la función regresará a un objeto de tipo `requests.models.Response`.

Argumentos que pueden ser ingresados a las funciones de métodos.

- `params` al que se le asigna un objeto de tipo `dict` con pares correspondientes a los parámetros que se le añaden a una *URL*.
- `header` al que se le asigna un objeto de tipo `dict` con pares correspondientes a campos de encabezados, los cuales sustituirían a los campos de encabezaddos que usa `requests` por defecto.
- `data` al que se le asigna un objeto que corresponde a los datos que serán enviados al servidor. Los datos son enviados en formato binario.

- `json` al que se le asigna un objeto de tipo `dict` cuyos datos serán enviados en formato *JSON*.
- `auth` se utiliza para conexiones que requieren de autenticación simple y al que se le asigna un objeto de tipo `tuple` con un par de datos `(<usuario>, <contraseña>)`.

Ejemplo:

- Se utilizará la función `requests.get()` en la *URL* https://coder.mx para abir una conexión enviando una petición `GET` y el objeto resultante será nombrado `sitio`.

```
[ ]: sitio = requests.get("https://coder.mx")
```

- El objeto sitio es una instancia de la clase `requests.models.Response`.

```
[ ]: type(sitio)
```

1.10 Atributos y métodos de los objetos `requests.models.Response`.

Los objetos de tipo `requests.models.Response` contienen varios atributos y métodos que permiten acceder a diversos datos de la respuesta obtenida a partir de uan petición.

1.10.1 Atributos.

- El atributo `headers`, el cual es un objeto de tipo `dict` conteniendo los encabezados de la respuesta del servidor. Cada clave del objeto corresponde al identificador de un atributo y los valores correspondientes a la calve siempre son objetos de tipo `str`. Algunas de dichas claves son:
  - La clave `'Date'`, el cual contiene la fecha y hora en la que se envió la respuesta.
  - La clave `'Server'`, el cual contiene la descripción del servidor.
  - La clave `'Last-Modified'`, el cual contiene la fecha de la última modificación del recurso.
  - La clave `'Content-Encoding'`, el cual contiene la codificación del contenido.
  - La clave `'Content-Length'`, el cual contiene el tamaño en bytes del contenido.
  - La clave `'Content-Type'`, el cual describe el tipo MIME del contenido.
- El atributo `status_code`, el cual corresponde al número de código de estado que regresa el servidor.
- El atributo `reason`, el cual es un objeto de tipo`str` con un mensaje relacionado al código de estado.
- El atributo `url`, el cual es un objeto de tipo `str` que contiene a la *URL* a la que se le hizo la peitción.
- El atributo `content`, el cual es un objeto de tipo `bytes` con los datos enviados por el servidor.
- El atributo `encoding`, el cual es un objeto de tipo `str` que describe la codificación usada por el servidor.
- El atributo `text`, el cual es un objeto de tipo `str` que contiene una representación del contenido, en caso de que el tipo de contenido sea texto.

1.10.2 Métodos.

- El método `close()`. Este método cierra la sesión entre el cliente y el servidor.
- El método `json()` puede convertir en un objeto tipo `dict` al contenido de una respuesta cuando el atributo `Content-Type` sea de tipo `'application/json'`.

Ejemplos:

- La siguiente celda deplegará el contenido del atributo `sitios.headers`.

```
[ ]: sitio.headers
```

- La siguiente celda deplegará el contenido de la clave `'Content-Type'` del atributo `sitios.headers`.

```
[ ]: sitio.headers['Content-Type']
```

- La siguiente celda deplegará el contenido del atributo `sitio.status_code`.

```
[ ]: sitio.status_code
```

- La siguiente celda deplegará el contenido del atributo `sitio.reason`.

```
[ ]: sitio.reason
```

- La siguiente celda deplegará el contenido del atributo `sitio.content`.

```
[ ]: sitio.content
```

```
[ ]:
```

- La siguiente celda deplegará el contenido del atributo `sitio.url`.

```
[ ]: sitio.url
```

- La siguiente celda deplegará el contenido del atributo `sitio.encoding`.

```
[ ]: sitio.encoding
```

- La siguiente celda deplegará el contenido del atributo `sitio.text`.

```
[ ]: print(sitio.text)
```

- La siguiente celda cerrará la sesión mediante el método `sitio.close()`.

```
[ ]: sitio.close()
```

1.10.3 Uso de la declaración `with` para objetos `requests.models.Response`.

En vista de que los objetos instanciado de `requests.models.Response` cuentan con un método `close()` es posible utilizar la declaración `with`.

Ejemplo:

Se utilizará la función `requests.get()` para acceder a la *URL* https://cloduevel.com enviando una petición con el método `GET`. Se desplegarán los siguientes datos guardados en el objeto resultante.

- El mensaje de estado resultante.

- Los encabezados del mensaje de respuesta.

[ ]:
```
with requests.get("http://cloudevel.com") as sitio:
    print(sitio.status_code)
    print(sitio.headers)
```

1.11 Ejemplos ilustrativos.

El sitio https://httpbin.org/ ofrece un servicio web que permite experimentar con diveras peticiones *HTTP* de forma segura.

Ejemplo:

- Se utilizará `requests.get()` para obtener un recurso que corresponde a una imagen loacalizada en la *URL* https://httpbin.org/image/png.

[ ]:
```
resultado = requests.get("https://httpbin.org/image/png")
```

- La siguiente celda desplegará el contenido en formato de bytes que fue enviado como respuesta.

[ ]:
```
print(resultado.content)
```

- La siguiente celda desplegará los encabezados de la respuesta.

[ ]:
```
resultado.headers
```

- La clave `'Content-Type'` del atributo `resultado.headers` indica que el contenido corresponde a una imagen en formato *png*.

[ ]:
```
resultado.headers['Content-Type']
```

- Para desplegar una imagen se utilizará el módulo `IPython.Display.Image` de *IPython*.

[ ]:
```
from IPython.display import Image
```

[ ]:
```
Image(resultado.content)
```

- A continuación se cerrará la sesión.

[ ]:
```
resultado.close()
```

Ejemplo:

- A continuación se utilizará la función `requests.post` para enviar una carga de datos a la *URL* "https://httpbin.org/post, la cual es capaz de procesar peticiones utilizando el método `POST` de *HTTP*.
- La carga de datos se se enviará en formato *JSON*.
- Al objeto resultante se le dará el nombre `respuesta`.

[ ]:
```
respuesta = requests.post("https://httpbin.org/post", json = {"saludo": "Hola"})
```

- La clave `'Content-Type'` del atributo `resultado.headers` indica que el contenido fue enviado usando el formato *JSON*.

[ ]:
```
respuesta.headers['Content-Type']
```

- A continuación se desplegará el contenido del atributo `respuesta.content`.

[ ]:
```
respuesta.content
```

- En vista de que el encabezado de la respuesta indica que el atributo`Content-Type` es `'application/json'`, es posible ejecutar el método `respuesta.json()` el cual regresará un objeto de tipo `dict` al que se le dará el nombre `respuesta_json`.

[ ]:
```
respuesta_json = respuesta.json()
```

[ ]:
```
respuesta_json
```

- A continuación se cerrará la sesión.

[ ]:
```
respuesta.close()
```