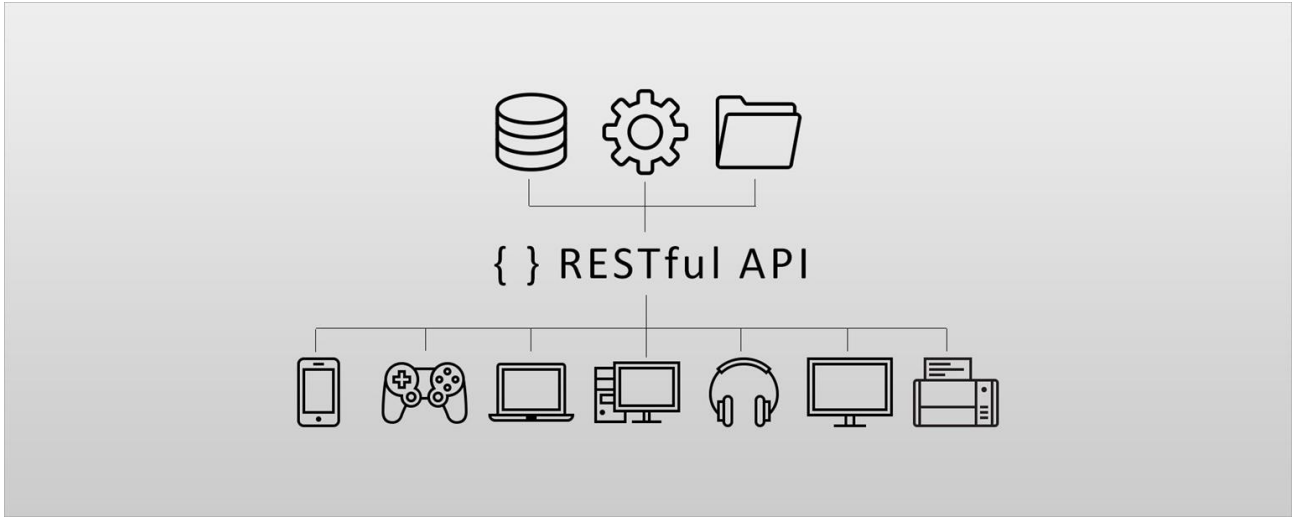
Express Avanzado

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Aplicaciones RESTful



Introducción

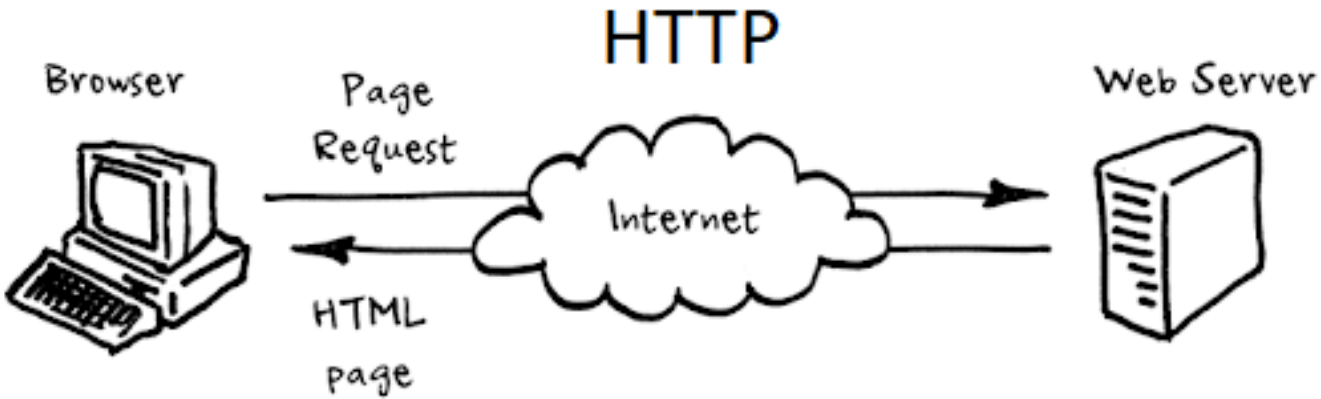
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

RESTless		RESTful	
VERB	HREF	VERB	URI
POST	/users/create	POST	/users
GET	/users/1	GET	/users/1
POST	/users/1/update	PUT	/users/1
????	/users/1/delete	DELETE	/users/1

Existen diversos tipos de aplicaciones: uno de los tipos más nombrados en la actualidad son las **Aplicaciones RESTFul**. Cuando hablamos de aplicaciones RESTful, nos referimos a aplicaciones que operan en forma de servicios web, respondiendo consultas a otros sistemas a través de internet. Dichas aplicaciones lo hacen respetando algunas reglas y convenciones que detallaremos a lo largo de esta clase

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Protocolo HTTP

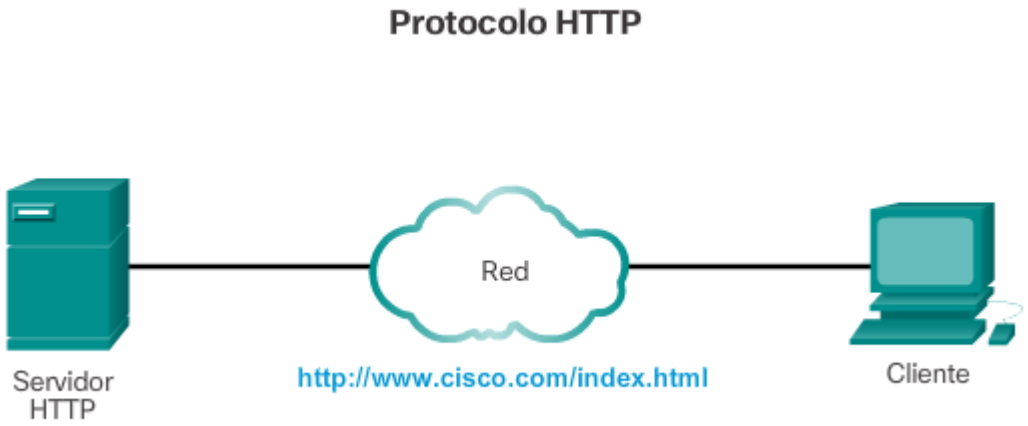


Repasemos el protocolo HTTP

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

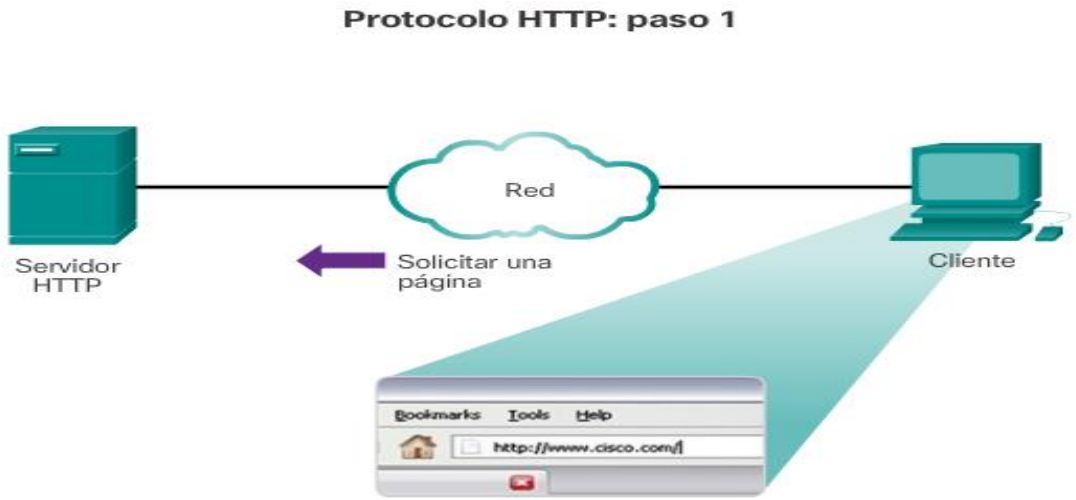
- **HTTP** *(Hypertext Transfer Protocol o Protocolo de Transferencia de HiperTexto)* es, como su nombre lo dice, un protocolo (conjunto de reglas y especificaciones) que se utiliza a la hora de **intercambiar datos a través de internet**.
- El protocolo se basa en un **esquema** de **petición-respuesta**.
- Existen **clientes** que realizan solicitudes de transmisión de datos, y un **servidor** que atiende la peticiones.
- HTTP establece varios tipos de peticiones, siendo las principales: *POST, GET, PUT, y DELETE*.

Etapas de comunicación HTTP



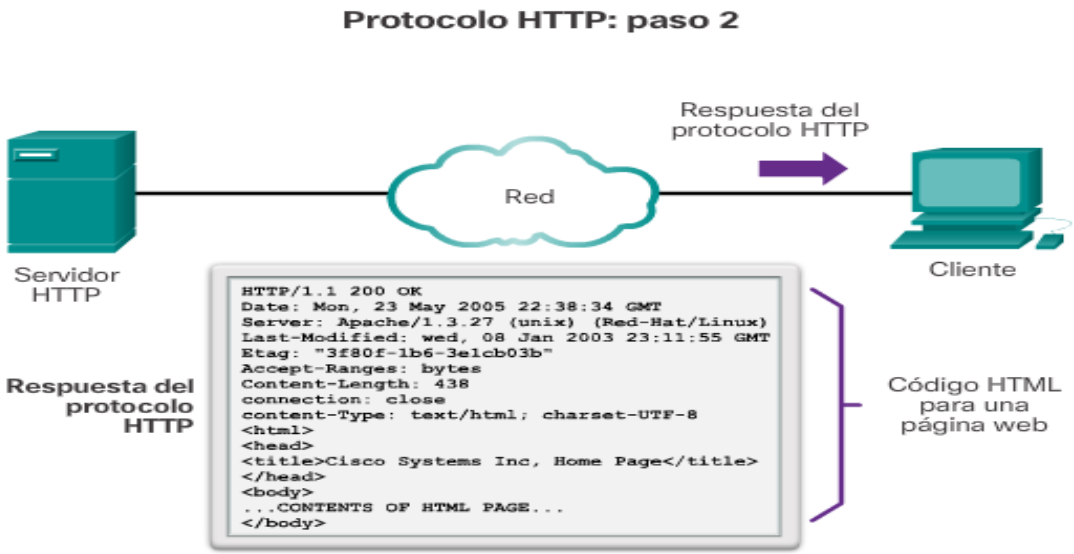
Etapas de comunicación HTTP

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES



El cliente inicia la solicitud de protocolo HTTP a un servidor.

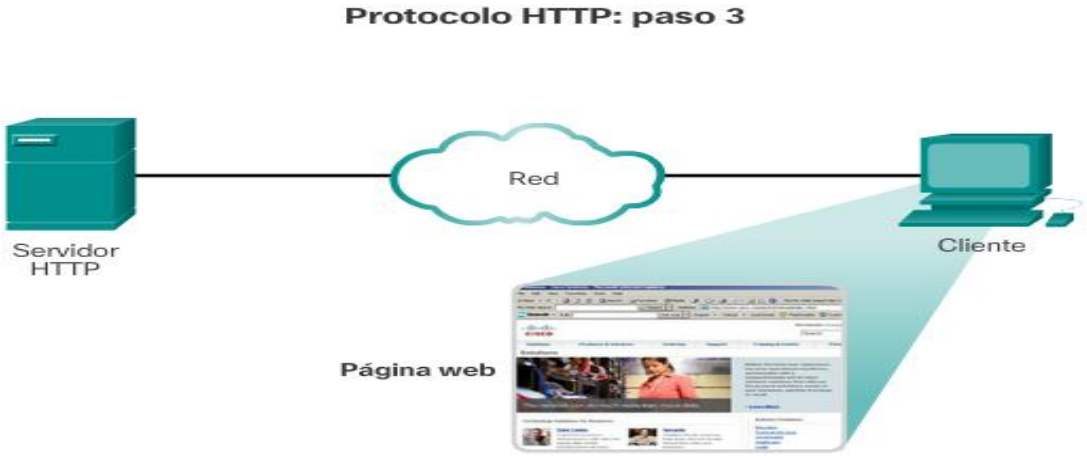
Etapas de comunicación HTTP



En respuesta a la solicitud, el servidor HTTP envía el código para una página web.

Etapas de comunicación HTTP

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES



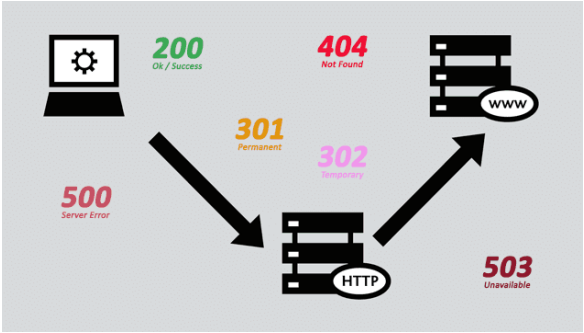
El navegador interpreta el código HTML y muestra una página web.

Códigos de Estado más comunes

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

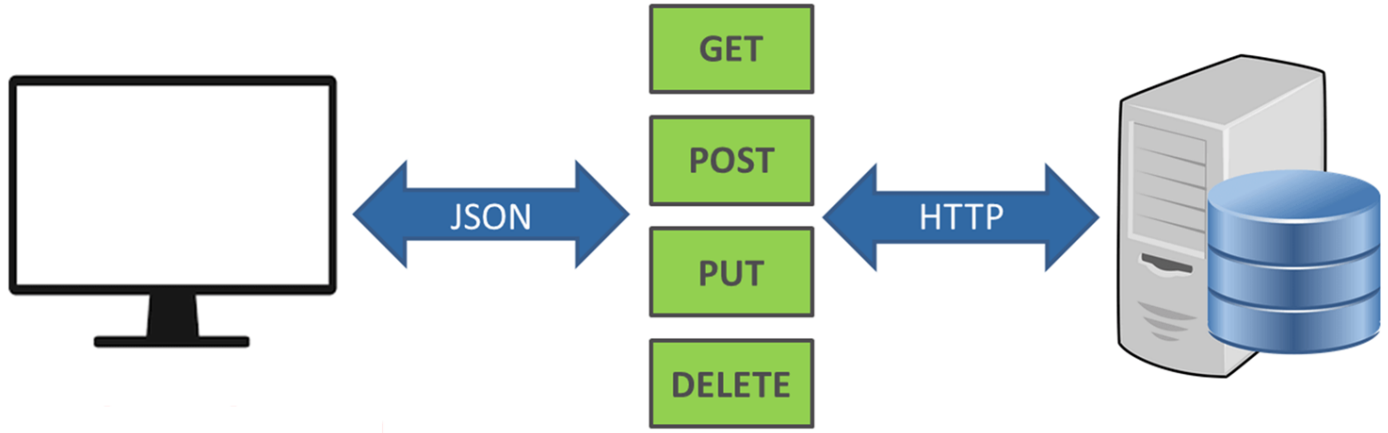
200	OK	Todo salió como lo esperado
400	Bad Request	La petición no cumple con lo esperado
404	Not Found	El recurso buscado no existe (URI inválido)
500	Internal Server Error	Error genérico del servidor al procesar una petición válida

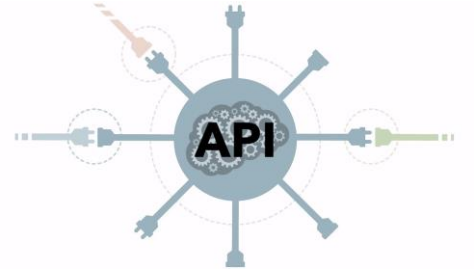
- **1xx**: Mensaje informativo.
- **2xx:** Exito
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No Content
- **3xx:** Redirección
  - 300 Multiple Choice
  - 301 Moved Permanently
  - 302 Found
  - 304 Not Modified
- **4xx:** Error del cliente
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- **5xx:** Error del servidor
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Conceptos de API, REST y API REST





¿Qué es una API?

- Una **API** es un conjunto de reglas y especificaciones que describen la manera en que un sistema puede comunicarse con otros.
- Definir una API en forma clara y explícita habilita y facilita el intercambio de mensajes entre sistemas.
- Permite la colaboración e interoperabilidad entre los sistemas desarrollados en distintas plataformas e incluso en distintos lenguajes.
- La **API** puede tener interfaz gráfica o ser de uso interno.
- La **API** tiene que estar acompañada con la documentación detallada que describa su operación y el formato de interacción con la misma.

{ REST }

¿Qué es REST?

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- **REST** viene del inglés “REpresentational State Transfer” (o en español: Transferencia de Estado Representacional).
- *Por Representación* nos referimos a un modelo o estructura con la que representamos algo.
- *Por Estado* de una representación, hablamos de los datos que contiene ese modelo estructura.
- Transferir un Estado de Representación implica el envío de datos (con una determinada estructura) entre dos partes.
- Los dos formatos más utilizados para este tipo de transferencias de datos son **XML y JSON.**

Formatos XML y JSON

- XML

```xml
<factura>
  <cliente>Gomez</cliente>
  <emisor>Perez S.A.</emisor>
  <tipo>A</tipo>
  <items>
    <item>Producto 1</item>
    <item>Producto 2</item>
    <item>Producto 3</item>
  </items>
</factura>
```

- JSON

```json
{
  "cliente":"Gomez",
  "emisor":"Perez S.A.",
  "tipo":"A",
  "items": [
    "Producto 1",
    "Producto 2",
    "Producto 3"
  ]
}
```

¿Qué es API REST?

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- Es un tipo de API que **no dispone de interfaz gráfica**.
- Se utiliza exclusivamente para **comunicación entre sistemas**, mediante el **protocolo HTTP**.
- Para que una API se considere REST, debe cumplir con las siguientes características:
  - *Arquitectura Cliente-Servidor sin estado*
  - *Cacheable*
  - *Operaciones comunes*
  - *Interfaz uniforme*
  - *Utilización de hipermedios*

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Características API REST

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Arquitectura Cliente-Servidor sin estado

- Cada mensaje HTTP contiene toda la información necesaria para comprender la petición.
- Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.
- Esta restricción mantiene al **cliente** y al **servidor débilmente acoplados**: el cliente no necesita conocer los detalles de implementación del servidor y el servidor se “despreocupa” de cómo son usados los datos que envía al cliente.

Cacheable

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- Debe admitir un **sistema de almacenamiento en caché**.
- La infraestructura de red debe soportar una caché **de varios niveles**.
- Este almacenamiento **evita repetir** varias **conexiones** entre el servidor y el cliente, en casos en que peticiones idénticas fueran a generar la misma respuesta.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Operaciones comunes

- Todos los recursos detrás de nuestra API deben poder ser **consumidos** mediante **peticiones HTTP**, preferentemente sus principales (POST, GET, PUT y DELETE).
- Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (en inglés: Create, Read, Update, Delete, en español: Alta, Lectura, Modificación, y Baja).
- Al tratarse de peticiones HTTP, éstas deberán **devolver** con sus respuestas los correspondientes **códigos de estado**, informando el resultado de las mismas.

Interfaz uniforme

- En un sistema REST, cada acción (más correctamente, cada recurso) debe contar con una URI (Uniform Resource Identifier), un identificador único.
- Ésta nos facilita el acceso a la información, tanto para consultarla, como para modificarla o eliminarla, pero también para compartir su ubicación exacta a terceros.

Utilización de hipermedios

- Cada vez que se hace una petición al servidor y este devuelve una respuesta, parte de la información devuelta pueden ser también hipervínculos de navegación asociada a otros recursos del cliente.
- Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

Principios

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- **Una aplicación RESTful** requiere un enfoque de diseño distinto a la forma típica de pensar en un sistema: lo **contrario a RPC**
- **RPC** (*Remote Procedure Calls*, llamadas a procedimientos remotos) basa su funcionamiento en las operaciones que puede realizar el sistema (acciones, usualmente verbos). *Ej: getUsuario()*
- En **REST**, por el contrario, el **énfasis** se pone **en los recursos** (usualmente sustantivos), especialmente en los nombres que se le asigna a cada tipo de recurso. *Ej. Usuarios.*
- Cada funcionalidad relacionada con este recurso tendría sus propios identificadores y peticiones en HTTP.

Por ejemplo...

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- **Listar usuarios:** Petición HTTP de tipo GET a la URL: http://servicio/api/usuarios

- **Agregar usuario:** Petición HTTP de tipo POST a la URL: http://servicio/api/usuarios (Agregando a la petición el registro correspondiente con los datos del nuevo usuario)

- **Obtener al usuario 1:** En caso de querer acceder a un elemento en particular dentro de un recurso, se lo puede hacer fácilmente si se conoce su identificador (URI): Petición HTTP de tipo GET a la URL: http://servicio/api/usuarios/1

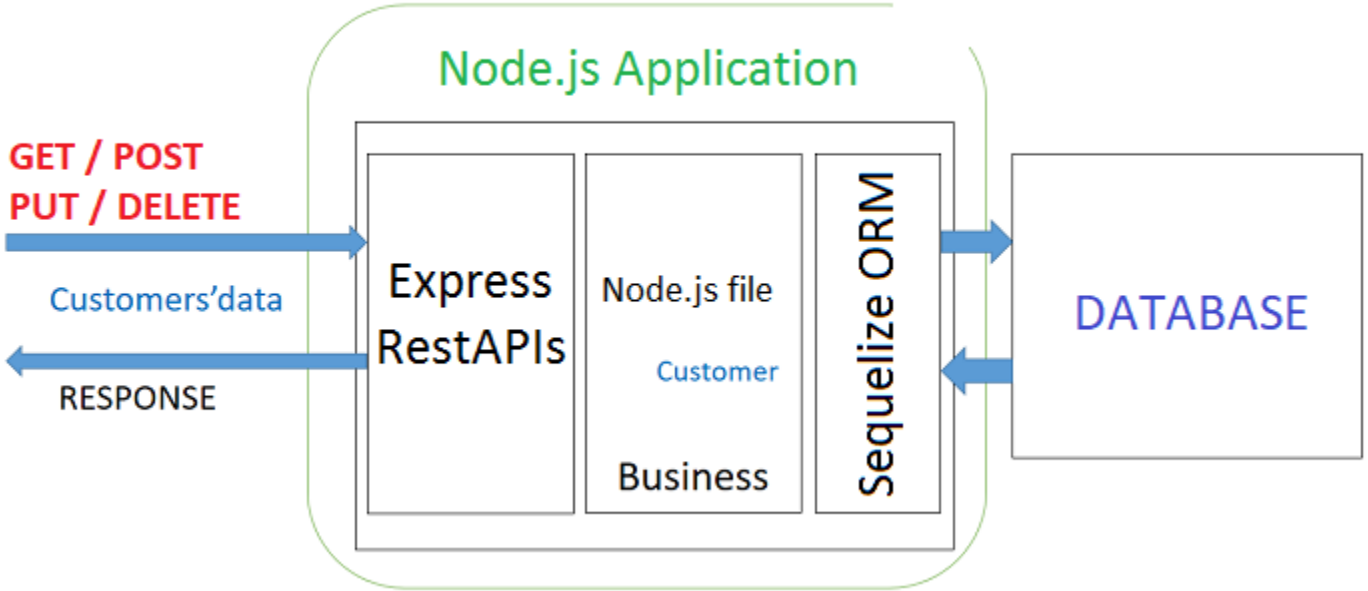
Por ejemplo...

- **Modificar al usuario 1:** Para actualizar un dato del usuario, un cliente REST podría primero descargar el registro anterior usando GET. El cliente después modificaría el objeto para ese dato, y lo enviaría al servidor utilizando una petición HTTP de tipo PUT.

- **Obtener usuarios con domicilio en CABA:** Si en cambio es necesario realizar una búsqueda por algún criterio, se pueden enviar parámetros en una petición HTTP. Éstos se pueden añadir al final de la misma con la siguiente sintaxis:
  Petición HTTP de tipo GET a la URL:
  http://servicio/api/usuarios?domicilio=CABA

Manejo de peticiones HTTP con Express

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES



Express: atención de peticiones

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- Para definir cómo se debe manejar cada tipo de petición usaremos los métodos nombrados de acuerdo al tipo de petición que manejan: ***get(), post(), delete(), y put()***.
- Todos reciben como primer argumento la ruta que van a estar escuchando, y *solo manejarán* peticiones que coincidan en ruta y en tipo. Luego, el segundo argumento será el callback con que se manejará la petición.
- Está tendrá dos parámetros: el primero con la petición (request) en sí y el segundo con la respuesta (response) que espera devolver.

Ejemplo de petición GET (Pedir)

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Cada tipo de petición puede tener diferentes características. Por ejemplo, algunas peticiones **no requieren el envío de ningún dato extra** en particular para obtener el recurso buscado. Este es el caso de la petición GET. Como respuesta a la petición, **devolverá** el **resultado** deseado en **forma de objeto**.

```
app.get('/api/mensajes', (req, res) => {
    console.log('request recibido')

    // acá debería obtener todos los recursos de tipo ‘mensaje’

    res.json({ msg: 'Hola mundo!'})
})
```

Ejemplo de petición GET con parámetros de búsqueda

Las **peticiones** pueden **incorporar detalles** sobre la búsqueda _____ que se quiere realizar.

- Estos parámetros se agregan al final de la URL, mediante un signo de interrogación ‘?’ y enumerando pares ‘clave=valor’ separados por un ampersand ‘&’ si hay más de uno.
- Al recibirlos, los mismos se encontrarán en el objeto ‘query’ dentro del objeto petición (req).

```
app.get('/api/mensajes', (req, res) => {
    console.log('GET request recibido')

    if (Object.entries(req.query).length > 0) {
        res.json({
            result: 'get with query params: ok',
            query: req.query
        })
    } else {
        res.json({
            result: 'get all: ok'
        })
    }
})
```

Ejemplo de petición GET con identificador

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

En caso de que se quiera acceder a un recurso en particular ya conocido, es necesario **enviar un identificador unívoco** en la URL.

- Para enviar este tipo de parámetros, el mismo se escribirá luego del nombre del recurso (en la URL), separado por una barra.
  Por ejemplo: ***http://miservidor.com/api/mensajes/1***

  (En este ejemplo estamos queriendo acceder al mensaje nro 1 de nuestros recursos.)

Ejemplo de petición GET con identificador

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- Para acceder al campo identificador desde el lado del servidor, Express utiliza una sintaxis que permite indicar anteponiendo **‘dos puntos’** antes del nombre del campo identificador, al especificar la ruta escuchada. Luego, para acceder al valor del mismo, se hará a través del **campo ‘params’** del objeto petición (req) recibido en el callback.

```
app.get('/api/mensajes/:id', (req, res) => {
    console.log('GET request recibido')

    // acá debería hallar y devolver el recurso con id == req.params.id

    res.json(elRecursoBuscado)
})
```

Ejemplo de petición POST (Enviar)

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Algunas peticiones requieren el **envío** de algún **dato** desde el **cliente** _ **hacia** el **servidor**. Por ejemplo, al crear un nuevo registro. Este es el caso de la petición **POST**. Para acceder al cuerpo del mensaje, incluído en la petición, lo haremos a través del campo ‘body’ del objeto petición recibido en el callback. En este caso, estamos devolviendo como respuesta el mismo registro que se envió en la petición.

```
app.post('/api/mensajes', (req, res) => {
    console.log('POST request recibido')

    // acá debería crear y guardar un nuevo recurso
    // const mensaje = req.body
```

Ejemplo de petición PUT (Actualizar)

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

También es posible mezclar varios mecanismos de pasaje de datos/parámetros, como es el caso de las peticiones de tipo PUT, en las que se desea actualizar un registro con uno nuevo.

- Se debe proveer el identificador del registro a reemplazar y el dato con el que se lo quiere sobreescribir.

```
app.put('/api/mensajes-json/:id', (req, res) => {
    console.log('PUT request recibido')

    // acá debo hallar al recurso con id == req.params.id
    // y luego reemplazarlo con el registro recibido en req.body

    res.json({
        result: 'ok',
        id: req.params.id,
        nuevo: req.body
    })
})
```

Ejemplo de petición DELETE (Borrar)

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Si quisiéramos **eliminar** un recurso, debemos **identificar unívocamente** sobre cuál de todos los disponibles se desea realizar la operación

```
app.delete('/api/mensajes/:id', (req, res) => {
    console.log('DELETE request recibido')

    // acá debería eliminar el recurso con id == req.params.id

    res.json({
        result: 'ok',
        id: req.params.id
    })
})
```

Configuración extra

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Para que nuestro servidor express pueda interpretar en forma automática mensajes de tipo **JSON** en formato **urlencoded** al recibirlos, debemos indicarlo en forma explícita, agregando las siguiente líneas luego de crearlo.

```
app.use(express.json())
app.use(express.urlencoded({extended: true}))
```

Aclaración: *{extended:true} precisa que el objeto req.body contendrá valores de cualquier tipo en lugar de solo cadenas.*
¡Sin esta línea, el servidor no sabrá cómo interpretar los objetos recibidos!

POSTMAN

API Testing and Automation

Postman

PUBLISH

Onboard developers to your API faster with Postman collections and documentation

DESIGN & MOCK

Design in Postman & use Postman's mock service

MONITOR

Create automated tests to monitor APIs for uptime, responsiveness, and correctness

DEBUG

Test APIs, examine responses, add tests and scripts

DOCUMENT

Create beautiful web-viewable documentation

AUTOMATED TESTING

Run automated tests using the Postman collection runner

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

¿Qué es Postman?

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Postman nace como una herramienta que principalmente nos permite crear peticiones sobre APIs de una forma muy sencilla y de esta manera, probar las APIs.

- El usuario de Postman puede ser un desarrollador que esté comprobando el funcionamiento de una API para desarrollar sobre ella o un operador que esté realizando tareas de monitorización **sobre una API.**

- **Instalación:** https://www.postman.com/downloads/

Download Postman | Try Postma

postman.com/downloads/

POSTMAN Product Use Cases Pricing Enterprise Explore Learning Center Sign In Sign Up

Download Postman

Download the app to quickly get started using the Postman API Platform. Or, if you prefer a browser experience, you can try the new web version of Postman.

The Postman app

The ever-improving Postman app (a new release every two weeks) gives you a full-featured Postman experience.

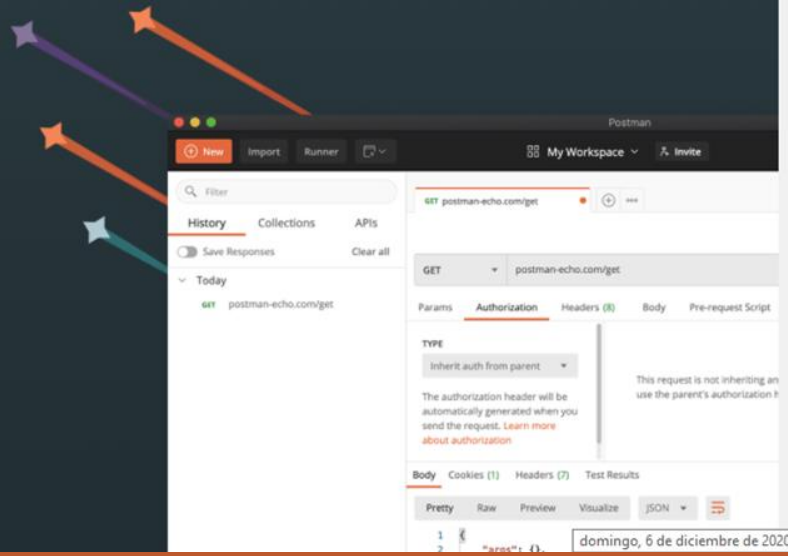
Download the App

By downloading and using Postman, I agree to the Privacy Policy and Terms.

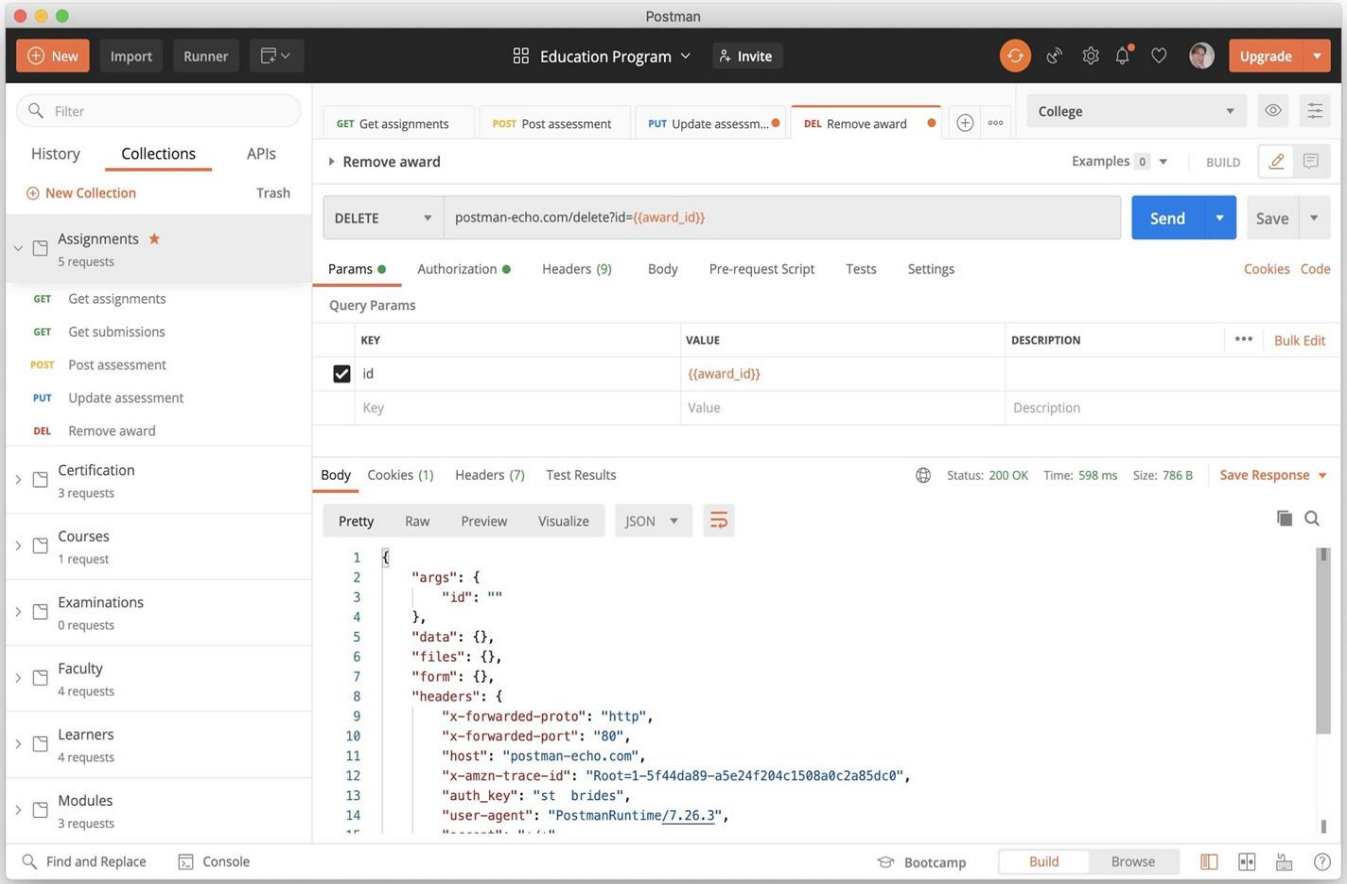
Version 7.36.0 | Release Notes | Product Roadmap

Not your OS? *Download for Mac (macOS) or Linux (x64)*

Postman on the web



domingo, 6 de diciembre de 2020



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

ARRAY DE PRODUCTOS

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Realizar un proyecto de servidor basado en node.js que permita listar e incorporar ítems dentro de un array de productos en memoria.

Cada producto estará representado por un objeto con el siguiente formato

```
{
    title: (nombre del producto),
    price: (precio),
    thumbnail: (url al logo o foto del producto)
}
```

- Implementar las rutas get y post en conjunto con las funciones necesarias (utilizar clases y un módulo propio).

Cada ítem almacenado dispondrá de un id proporcionado por el backend, que se irá incrementando a medida de que se incorporen productos. Ese id será utilizado para identificar un producto que ve a ser listado en forma individual.

Las rutas propuestas serían las siguientes:

A. Listar en forma total (get) : '/api/productos/listar' -> devuelve array de productos

B. Listar en forma individual (get) (por id): '/api/productos/listar/:id' -> devuelve producto listado

C. Almacenar un producto (post) : '/api/productos/guardar/' -> devuelve producto incorporado

- Para el caso de que se liste en forma individual un producto que no exista, se devolverá el objeto: {error : 'producto no encontrado'}

En caso de no haber productos en el listado total, se retornará el objeto: {error : 'no hay productos cargados'}

Las respuestas del servidor serán en formato JSON. La funcionalidad será probada a través de Postman.

Aclaración:

El servidor debe estar basado en express y debe implementar los mensajes de conexión al puerto 8080 y en caso de error, representar la descripción del mismo.