



# Coding Bootcamp

## Sprint 2



# Temario



# Temario

- Express.
- HTTP
  - Introducción.
  - Request & Response.
  - Métodos: GET, POST, PUT, PATCH, DELETE.
- Express, rutas y HTTP.
- Responder un archivo HTML.
- Estructura de una URL.
- Parámetros de una ruta.



Express



# Express

- Es un **framework** para Node.js.
- Diseñado para construir aplicaciones web y APIs.
- Es muy popular. Y es open-source.
- Es **minimalista** → es **rápido**.
- Sirve como base para otros frameworks más “grandes” como [Sails.js](#) o [Adonis.js](#) (este último muy similar a Laravel para PHP).
- Documentación: <https://expressjs.com>.

Express

👉 Vamos a instalar Express como una **dependencia** en nuestro proyecto.



# Ejemplo 1



# Ejemplo 1 (1/3)

1. Crear un proyecto de Node (usando `npm init`).
2. **Instalar Express** como dependencia del proyecto: `npm install express`
3. Al terminar, el archivo `package.json` debería contener una nueva entrada `dependencies` con `express` dentro.

Algo así:

```
"dependencies": {  
  "express": "^4.17.1"  
}
```

*\* La versión puede ser distinta a la que aparece en esta diapositiva, Express está en constante actualización*

## Ejemplo 1 (2/3)

Podría tener otro nombre, por ejemplo: `server.js`.



Luego, crear un archivo `index.js` con el siguiente código:

```
const express = require("express");

const app = express(); // Crea una instancia de express.

app.get("/", (req, res) => res.send("!Hola Hack Academy!"));

app.listen(3000, () => console.log("!Servidor corriendo en el puerto 3000!"));
```

Aquí se está definiendo una función común y corriente, pero usando la sintaxis de **Arrow Functions**. Ver más [aquí](#).

Luego se deberá correr el comando `node index.js` para que el servidor quede “prendido” (ejecutándose).

Notar que si realiza un cambio en el archivo `index.js` se deberá “cortar” la ejecución anterior cerrando la terminal o con CTRL+C, y luego se deberá correr nuevamente el comando `node index.js`.





## Ejemplo 1 (3/3)

Es común crear, al lado de `index.js`, un archivo **routes.js** que contendrá lo que refiere a manejo de rutas (que previamente estaba en `index.js`).

`routes.js` deberá tener un *export*, el cual será una función que tan solo recibirá la instancia de `express` para poder crear los *handlers*.

Luego, en `index.js`, se deberá importar esa función y ejecutarla.

El código debería funcionar igual que antes, pero ahora las **responsabilidades** están mejor repartidas gracias a la modularización.



# HTTP

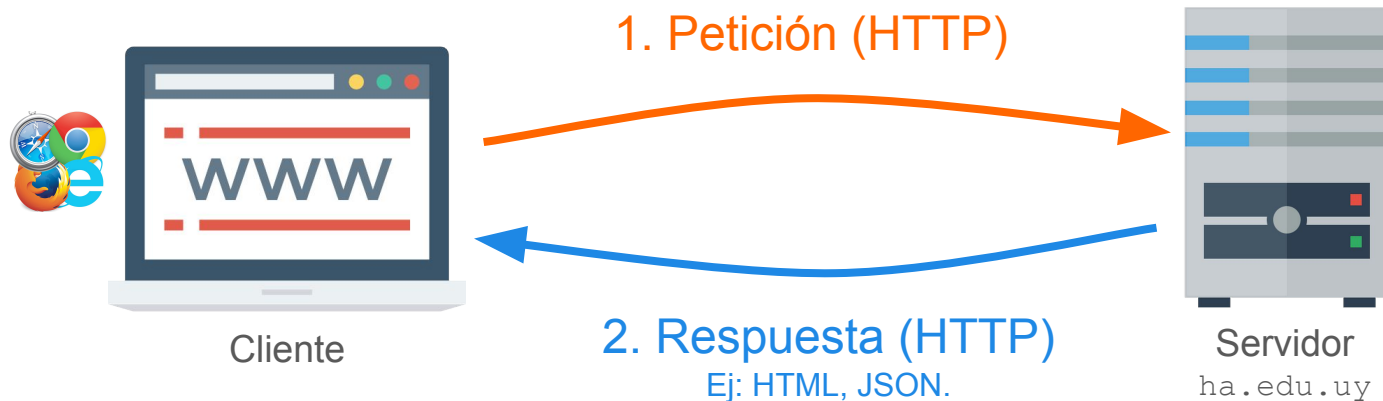
# Hypertext Transfer Protocol (HTTP) (1/3)



1. Cuando un usuario escribe una URL en su navegador y presiona `Enter`, lo que hace es realizarle una **petición** o **request** (HTTP) a un servidor web. Técnicamente esto se conoce como un request de tipo GET (lo veremos en breve).

Básicamente el usuario le dice al servidor: *“Dame lo que haya en esta URL: `https://ha.edu.uy`”*.

# Hypertext Transfer Protocol (HTTP) (2/3)



2. Al recibir la petición, una aplicación en el servidor la analiza, la procesa y responde al cliente con una **respuesta** o **response** que suele ser en formato HTML (pero puede ser JSON, JavaScript, CSS, una imagen o texto plano). Los posibles formatos se pueden consultar [aquí](#).

El servidor jamás retorna código PHP, Java, Ruby, Python, C# u otro código de Back-End. Recordar que el navegador sólo "entiende" cosas como HTML, CSS, JavaScript e imágenes.



# Hypertext Transfer Protocol (HTTP) (3/3)

La comunicación anterior, entre el cliente y el servidor, se realiza utilizando un **protocolo** llamado **HTTP**.

Un protocolo es un **conjunto de reglas** que indican cómo debe ser la comunicación entre dos equipos, estableciendo la forma de identificación de los equipos en la red, la forma de transmisión de los datos y la forma en que la información debe procesarse.

La comunicación con un sitio web puede realizarse usando el protocolo HTTP (sin encriptar) o el HTTPS (encriptado).

Otros protocolos que tal vez conocen son: FTP, SMTP, IMAP, POP, SSH, TCP, UDP, etc.



# HTTP – Request



# HTTP – Request (1/3)

Cuando se realiza una llamada (*request*) HTTP es necesario especificar:

- Una URL.
  - Ejemplo: <https://ha.edu.uy/cursos/back-end-nodejs>.
  - La URL contiene datos como el protocolo, dominio y recurso que se quiere acceder.
- Un método.
  - Ejemplos: GET, POST, PUT, DELETE.
- Headers.
  - Para enviar datos adicionales en la llamada, por ejemplo, una contraseña o API Token.
- Un body (opcional).
  - Para enviar contenido junto con la llamada, por ejemplo, un objeto JSON.



# HTTP – Request (2/3)

Ejemplo de un *request* de tipo **GET**. En este caso se está llamado a la URL <http://examplecat.com/cat.png>.

method  
(usually GET or POST) → GET /cat.png HTTP/1.1

resource being requested →

HTTP version →

headers { Host: examplecat.com ← domain being requested  
User-Agent: Mozilla...  
Cookie: .....





# HTTP – Request (3/3)

Ejemplo de un *request* de tipo **POST**. En este caso se está llamado a la URL [http://examplecat.com/add\\_cat](http://examplecat.com/add_cat) y se está enviando un objeto JSON.

*method* → **POST /add\_cat HTTP/1.1**

*headers* { **Host: examplecat.com**  
**Content-Type: application/json** ← *content type of body*  
**Content-Length: 20**

**{"name": "mr darcy"}** ← *request body: the JSON we're sending to the server*



# HTTP – Response



# HTTP – Response (1/2)

Toda llamada (*request*) HTTP recibe una respuesta (*response*) HTTP, que debe contener:

- Un código de estado (*status code*).
  - Ejemplos: 200 (OK), 404 (Not Found), 500 (Internal Server Error). [Ver más](#).
- Headers.
  - Para enviar datos adicionales.
- Un body.
  - Para enviar contenido. Ej: HTML, una imagen, JSON, texto plano.

👉 Ej: Cuando se hace un *request* a <https://ha.edu.uy> se obtiene un *response* 200 que contiene HTML. Sugerimos que investiguen esto en la pestaña Network de los Developer Tools de Chrome.



# HTTP – Response (2/2)

Ejemplo de una *response* (respuesta) HTTP que retorna un texto plano.

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Content-Length: 33
Content-Type: text/plain; charset=UTF-8
Date: Mon, 09 Sep 2019 01:57:35 GMT
Etag: "ac5affa59f554a1440043537ae973790-ssl"
Strict-Transport-Security: max-age=31536000
Age: 0
Server: Netlify

\      /\
 )    ( ' )
(    /  )
 \(__) |
```

*status*

*status code*

*headers*

*cat! ☺*

*body*



# HTTP – Métodos



# HTTP – Request: Métodos HTTP

MDN: “*HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Cada uno de ellos implementan una semántica diferente*”.

A continuación se hará un resumen de los más importantes, de aquellos que usaremos en el curso. Por más información pueden ingresar a [MDN](#).

👉 Es interesante notar que un método HTTP no es más que un pequeño texto que se incluye en un *request* HTTP como, por ejemplo, “GET” y “POST”.



# HTTP – Request: Método GET

El método GET se utiliza para **solicitar** una **representación** de un **recurso** específico.

En este curso, la representación que obtendremos del servidor generalmente será en formato JSON. Los *requests* que usan el método GET sólo deben **obtener** datos.

Algunos *requests* con este método incluyen un *body*, para incluir ciertos parámetros que no están presentes en la URL a la cual se apunta la petición.

👉 Ej: hacemos un *request* GET para **obtener** datos de un artículo de un Blog.



# HTTP – Request: Método POST

El método POST se utiliza para **crear una entidad**, en general, nueva de un recurso, **causando un cambio** en el estado o efectos secundarios en el servidor.

Generalmente este cambio constituye una inserción en la base de datos.

Este *request* generalmente incluye un *body*, en el cual van los datos que constituyen la entidad.

👉 Ej: hacemos un *request* POST para crear un artículo nuevo en un Blog.





# HTTP – Request: Método PUT

El método PUT se usa para **reemplazar completamente una entidad existente** en el servidor con la del *body* del *request*, **causando un cambio** en el estado o efectos secundarios en el servidor. Generalmente este cambio implica una interacción en la base de datos.

Este *request* generalmente incluye un *body*, en el cual van los datos que constituyen la entidad.

👉 Ej: hacemos un *request* PUT para reemplazar un artículo nuevo en un Blog.



# HTTP – Request: Método PATCH

El método PATCH se usa para **alterar parcialmente una entidad existente** en el servidor con los datos incluídos en el *body* del *request*, **causando un cambio** en el estado o efectos secundarios en el servidor. Generalmente este cambio implica una interacción en la base de datos.

👉 Ej: hacemos un *request* PATCH para editar un artículo en un Blog.



# HTTP – Request: Método DELETE

El método DELETE **elimina una entidad existente** en el servidor.

Este *request* no incluye un *body*.

👉 Ej: hacemos un request DELETE para eliminar un artículo en un Blog.

# CRUD

Create



POST

Read



GET

Update



PUT, PATCH

Delete



DELETE



HTTP  
Methods  
(Verbs)



# Formularios HTML y métodos HTTP



# Formularios HTML y métodos HTTP

Es interesante notar que desde un formulario HTML sólo se pueden hacer requests de tipo GET y POST.

```
<form action="/contacto" method="POST">  
  <input type="text" name="nombre" id="nombre">  
  <textarea name="comentario" id="comentario"></textarea>  
  <button type="submit">Enviar datos</button>  
</form>
```

Además, desde la barra de direcciones de un navegador sólo es posible hacer llamadas de tipo GET. En breve veremos cómo hacer llamadas con los otros métodos.



# Express, rutas y HTTP



# Express, rutas y HTTP (1/3)

¿Recuerdan este código? Veamos qué es cada componente.

```
app.get("/saludo", (req, res) => res.send("¡Hola Hack Academy!"));
```

- `app` es una instancia de Express.
- `get` es un método de Express, cuyo nombre coincide con el **método HTTP** GET. Análogamente, se pueden usar los métodos `.post`, `.put`, `.patch`, `.delete`, para aceptar distintos tipos de requests en la aplicación.
- `"/saludo"` (primer parámetro de la función `get`) es una **ruta** (*route*). También se le puede decir *path*.
- El segundo parámetro de la función `get` es también una función (*handler* o *callback*) que se ejecuta cada vez que se reciba un *request* de tipo GET en la ruta especificada.





# Express, rutas y HTTP (2/3)

La sintaxis genérica es:

```
APP.METHOD(PATH, HANDLER);
```

- `APP` es una instancia de Express.
- `METHOD` es un método HTTP (en minúscula).
- `PATH` es una ruta  $\approx$  la porción de la URL seguida del dominio.
- `HANDLER` es la función que se ejecuta cada vez que se reciba un request en la ruta especificada. También se le dice *callback*.

👉 Por detalles, ver la documentación oficial sobre [Routing](#).



# Express, rutas y HTTP (3/3)

La función *handler* (o *callback*) puede recibir como parámetros dos objetos a los cuales se les llama `request` y `response`, o simplemente `req` y `res`.

```
app.get("/saludo", function(req, res) {  
    res.send("¡Hola Mundo!");  
});
```

- `req` es un objeto que contiene información sobre el *request* (petición).
- `res` es un objeto que permite retornar una respuesta, usando el método `send`.



# Responder un archivo HTML



# Responder un archivo HTML

Con Express podemos responder un *request* con un archivo HTML:

```
app.get("/productos", (req, res) => {  
    res.sendFile(__dirname + "/productos.html");  
});
```

`__dirname` es una variable que siempre está presente en Node y lo que contiene es el *path* absoluto del directorio donde está ubicado el archivo que está llamado a dicha variable.

⚠ Notar que no es posible escribir `res.sendFile("./productos.html")` ya que daría error.  
[Más info aquí.](#)



# Estructura de una URL

# Estructura de una URL (1/2)

`http://ha.edu.uy:80/productos`

The diagram shows the URL `http://ha.edu.uy:80/productos` with four colored brackets underneath it, each pointing to a specific part of the URL. Below each bracket is a label: 'Protocolo' (blue) under 'http', 'Dominio' (orange) under 'ha.edu.uy', 'Puerto' (dark blue) under ':80', and 'Ruta (path)' (pink) under '/productos'.


Protocolo      Dominio      Puerto      Ruta (path)

- **Protocolo:** Conjunto de reglas que indican cómo debe ser la comunicación entre dos equipos, estableciendo la forma de identificación de los equipos en la red, la forma de transmisión de los datos y la forma en que la información debe procesarse. Por defecto se usa el protocolo `HTTP`, pero se podría usar `HTTPS`.
- **Dominio:** Dirección (string) que permite acceder de forma amigable a la dirección IP donde se encuentra el servidor. Ej: `ha.edu.uy` apunta a la IP `162.243.205.105`.
- **Puerto:** Interfaz de comunicación en un sistema operativo. No es hardware, es un concepto lógico (software). Por defecto los servidores web se encuentran “escuchando” en el puerto 80 para `HTTP` y el puerto 443 para `HTTPS`.
- **Ruta:** El recurso que se le está pidiendo al servidor. Podría ser un archivo como `avion.jpg`.



## Estructura de una URL (2/2)

`http://ha.edu.uy:80/productos?moneda=USD`



Query String

- **Query String:** En la URL también se puede agregar datos adicionales que serán enviados al servidor a través de un conjunto de parámetros llamado *query string*.
  - El símbolo de pregunta ? marca el comienzo del query string.
  - En este ejemplo, el query string contiene un sólo parámetro llamado `moneda` con el valor `USD`. En este caso, esta información adicional es para avisarle al servidor que muestre los precios de los productos en dólares.
  - Se pueden enviar varios parámetros, separándolos con un ampersand &.



# Parámetros de una ruta





# Parámetros de una ruta

Express permite crear rutas “dinámicas” compuestas por distintos parámetros.

```
app.get("/usuarios/:userId", (req, res) => {  
    res.send("El id del usuario es: " + req.params.userId);  
});
```

Esto significa que las siguientes rutas cumplen el caso anterior y, por lo tanto, para cualquiera de ellas se llamará al mismo handler.

- `http://localhost:3000/usuarios/2`
- `http://localhost:3000/usuarios/67`
- `http://localhost:3000/usuarios/43?color=red`
- `http://localhost:3000/usuarios/maria`

A través del objeto `params` se accede a los parámetros de la ruta (path).

Para acceder a parámetros que vienen por *query string*, se utiliza el objeto `query`.