Curso de Back-End con Node.js (Inicial)

Clase 04

Temario

Temario

- Protocolo HTTP:
  - Introducción.
  - Request & Response.
  - Métodos: GET, POST, PUT, PATCH, DELETE.
- Formularios HTML y métodos HTTP.
- Express – Parámetros de una ruta.
- Motores de templates.
  - Nunjucks.

REPASO

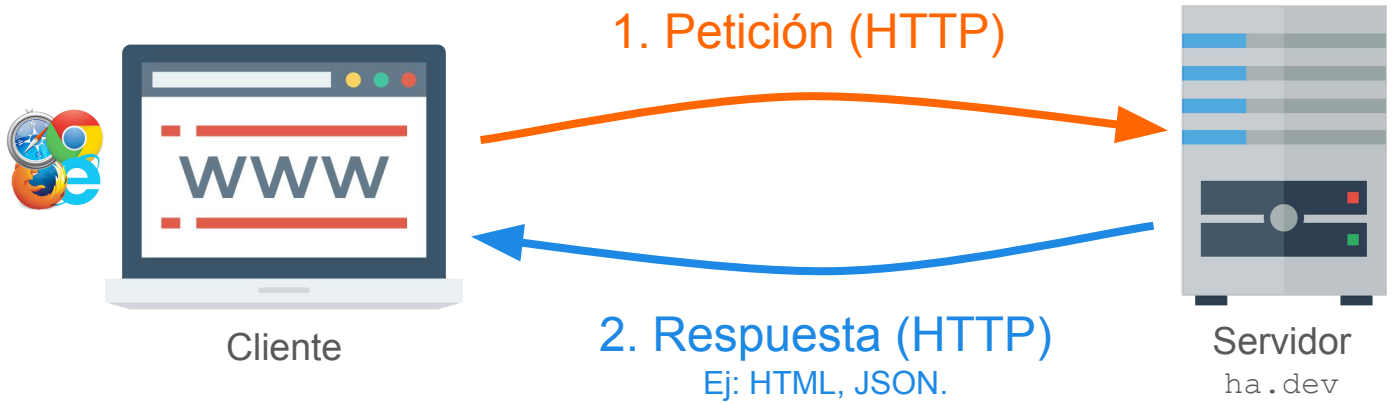
HTTP

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.dev”*.

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 o HTTPS.

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.dev/cursos/front-end.
  - 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.

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 y se está enviando un objeto JSON.

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.dev 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 – 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="/multiplicar" method="GET">
    <input type="text" name="num1" id="num1">
    <input type="text" name="num2" id="num2">
    <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 – 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`.

Ejercicio 1

Ejercicio 1 (1/2)

1. Crear un sitio web con Express que contenga las siguientes rutas:
   a. `[GET]` http://localhost:3000.
   b. `[GET]` http://localhost:3000/multiplicar.

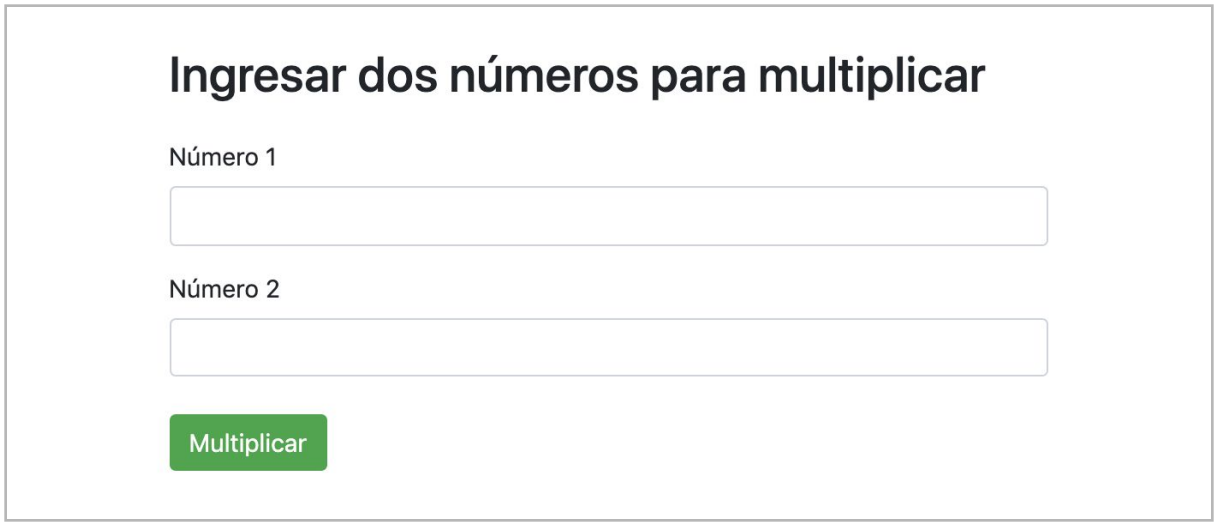
2. En la Home debe haber un formulario HTML que contenga dos campos:
   Número 1 y Número 2.

3. El formulario debe tener un botón llamado “Multiplicar” y hacer click en el mismo se debe llamar a la ruta `/multiplicar`, a la cual se le deben pasar los números por el query string.
   Ejemplo: http://localhost:3000/multiplicar?num1=5&num2=6.

4. Al acceder a dicha URL debería aparecer el texto: “El resultado es 30”. Es página no debe ser un HTML.

Ejercicio 1 (2/2)

Ejemplo visual:



Motores de Templates

Motores de Templates

Anteriormente vimos que podíamos enviar un archivo HTML como respuesta a una llamada HTTP.

En Express vimos que podemos hacerlo utilizando el método `res.sendFile` y pasándole un archivo HTML (ej: `home.html`).

Sin embargo, los archivos HTML son estáticos. Es decir, tendríamos que tener un archivo HTML distinto para cada página o para cada tipo de respuesta que quisiéramos enviar. Además, tendríamos que repetir código en cada página (ej: para el cabezal o para el pie de página). Esto es muy poco conveniente.

Por suerte, en casi todos los lenguajes de programación existen Motores de Templates o *Template Engines* que facilitan mucho esta tarea.

Nunjucks

Nunjucks

- Es un motor de templates para JavaScript. Es independiente de Node.js y de Express.
- Permite generar archivos HTML de forma dinámica, a partir de código JavaScript (aunque con una sintaxis particular), y aprovechando todas las ventajas de un lenguaje de programación.
- En general vamos a trabajar con archivos con extensión `.njk`, aunque podrían ser `.html`, y los colocaremos dentro de una carpeta llamada `views`.
- Documentación: https://mozilla.github.io/nunjucks.
- Otras alternativas pueden ser Pug (antes llamado Jade), EJS, Mustache o Edge. Nota: Cada motor de templates tiene su propia sintaxis y particularidades.

Nunjucks – Instalación (1/2)

Para instalar Nunjucks en un proyecto de Node.js se debe correr el comando:

```
npm i nunjucks
```

Nunjucks – Instalación (2/2)

Si se está usando Express, es necesario indicarle a la `app` (la instancia de Express) que usaremos Nunjucks. Esto se hace en `index.js` (o el archivo principal que se esté usando):

```
nunjucks.configure("views", {
  autoescape: true,
  express: app,
});

app.set("view engine", "njk");
```

Nunjucks – Uso (1/2)

Ejemplo de uso:

Ej: `index.js`

```
app.get("/", function (req, res) {
    res.render("home", { title: "Hack Academy" });
});
```

El método `render` permite llamar a un archivo (`home.njk`) que está dentro de la carpeta `views` del proyecto. Notar que no es necesario especificar la extensión del archivo ya que anteriormente se le dijo a Express que se usará Nunjucks como motor de templates.

Como segundo parámetro se pasa un objeto con datos que se le quieren pasar a la vista `home`.

Nunjucks – Uso (2/2)

Veamos ahora el archivo `home.njk`. Su contenido es muy similar al que podría contener un archivo HTML, pero con algunas características especiales:

Ej: `home.njk`

```
    ...
    <body>
        <h1>{{ title }}</h1>
    </body>
</html>
```

La sintaxis `{{ … }}` es propia de Nunjucks. En tiempo de ejecución, Nunjucks sustituye esa “etiqueta” por el contenido de la variable `title`.

Por mayor detalles entrar a https://mozilla.github.io/nunjucks/templating.html.

Nunjucks – Include

Nunjucks – Include (1/2)

Con Nunjucks es posible que un archivo `.njk` incluya a otro archivo `.njk`.

Por ejemplo, el archivo `views/home.njk` podría incluir el archivo `views/partials/header.njk`:

Ej: `home.njk`

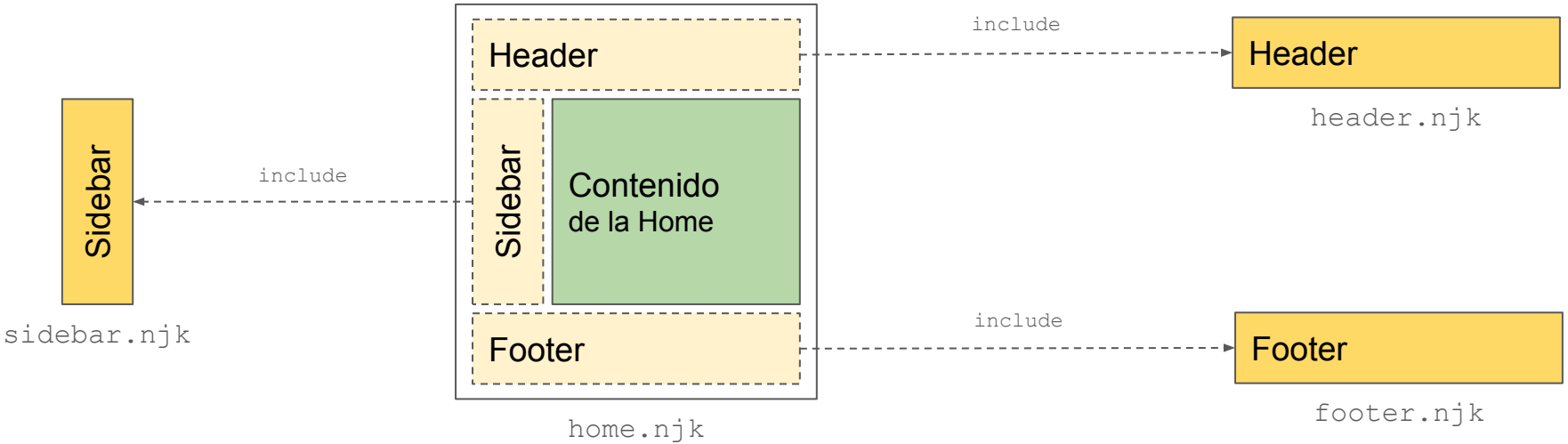
```
    ...
    <body>
        {% include "partials/header" %}
    </body>
</html>
```

El uso de la carpeta `partials` no es obligatorio, pero suele ser una buena práctica.

Esto es útil para no repetir código, ya que el cabezal del sitio lo podemos escribir una sóla vez y luego incluirlo desde varias otras páginas.

Nunjucks – Include (2/2)

La solución anterior se podría resumir en el siguiente diagrama.



Nunjucks – Herencia

Nunjucks – Herencia (1/2)

La directiva `extends` permite definir un archivo `master` (padre) que contenga la estructura básica de una página y luego definir páginas *secundarias* (o *hijas*) que extiendan o *hereden* de `master`.

En `master.njk`

```
<html>
    <head>
        <title>Empresa ACME</title>
    </head>
    <body>
        {% block contenido %}{% endblock %}
        <footer>ACME - Copyright © 2021</footer>
    </body>
</html>
```

En `contacto.njk`

```
{% extends "master" %}

{% block contenido %}
    <h1>Página de Contacto</h1>
{% endblock %}
```

Nota: El nombre `master` es arbitrario, pero es común llamarle de esa forma a la página que contiene la estructura básica del sitio.

Nunjucks – Herencia (2/2)

La directiva `block` permite definir zonas del archivo padre que luego serán sustituidas por código definido en el archivo hijo:

En `master.njk`

```
<html>
    <head>
        <title>Empresa ACME</title>
    </head>
    <body>
        {% block contenido %}{% endblock %}
        <footer>ACME - Copyright © 2021</footer>
    </body>
</html>
```

En `contacto.njk`

```
{% extends "master" %}

{% block contenido %}
    <h1>Página de Contacto</h1>
{% endblock %}
```

Nota: El nombre `master` es arbitrario, pero es común llamarle de esa forma a la página que contiene la estructura básica del sitio.

Ejercicio 2

Ejercicio 2

1. Crear una carpeta llamada `ejercicio_nunjucks`.
2. Inicializar un proyecto con el comando `npm init -y`.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. Instalar los módulos `express` y `nunjucks`. Tener nodemon instalado de forma global.
6. Crear las siguientes rutas:
   - [GET] http://localhost:3000.
   - [GET] http://localhost:3000/productos.
   - [GET] http://localhost:3000/sobre-nosotros.
   - [GET] http://localhost:3000/contacto.

Por un tema de orden:

⚠ Crear las rutas en un archivo llamado `routes.js`.

⚠ Crear los *handlers* de dichas rutas en un archivo aparte llamado `pagesController.js` Más adelante veremos por qué la elección de este nombre *"Controller"*.

Ejercicio 2 (cont)

7. Crear un cabezal (ej: usando el componente navbar de Bootstrap) que aparezca en todas las páginas y que contenga links entre ellas. El código del cabezal debe escribirse sólo una vez, para evitar código repetido y lograr un sitio más mantenible.
8. Levantar el servidor en el puerto 3000 y verificar que el sitio sea correctamente navegable.
9. En la Home (debajo del cabezal) debe haber un texto que diga:
    - “Hoy es un día de semana”, si estamos entre Lu y Vi.
    - “Hoy es fin de semana”, si estamos en Sábado o Domingo.

Ejercicio 2 (cont)

10. A la vista de productos, pasarle un array de strings con nombres de productos (ej: "Notebook", "Impresora", "Monitor", etc), el cual se deberá mostrar en una lista `<ul>`.