



# Arrancamos a las 19:05

(Así esperamos a los alumnos que puedan tener algún inconveniente con Teams)



# Curso de Back-End con Node.js (Avanzado)

## Clase 01



# *¡Muchas gracias por habernos elegido!*

Esperamos que puedan **disfrutar** y **aprovechar** este curso al máximo, y ojalá logremos transmitirles la **pasión** que nosotros sentimos por estos temas.

Si en algún momento del curso ven algo que les gustaría **mejorar**, por favor háganoslo saber lo antes posible. Además, al finalizar tendrán una **encuesta** de satisfacción anónima para hacernos llegar sus comentarios.



# Presentarnos

(30 segundos cada uno)

*Nombre | Qué hago | Qué espero del curso*



# Pedidos

# Pedidos 🙏




- Respetar los **horarios** 🕒 de clase.
- ¡**Participen** mucho! Queremos oír la **voz** de todos. 🗣️
- Queremos que compartan su **pantalla** 🖥️. Queremos ver lo que hicieron, y si está mal, mejor. De los **errores** se aprende.
- La clase es mucho más **divertida** si todos participan. 🎉
- En la **interacción** con ustedes es donde podemos aportar más **valor**. Para tener una clase “unilateral” (dónde sólo habla el docente), probablemente les rinda más un curso de Udemy.



# Tips generales



# Tips generales

- ¡Hagan **muchas preguntas!** (no sientan vergüenza).
- **Googleen** mucho. También consulten [Stack Overflow](#).
- Intenten **ayudarse** entre ustedes. Enseñar es una gran forma de aprender.
- No copien/peguen código salvo que entiendan o sepan lo que están copiando. Copiar código no está mal, pero sean criteriosos.
-  Cuidado con las actualizaciones automáticas del sistema operativo. Sobre todo en Windows. Eviten actualizar su equipo en medio de la clase.





# Uso de Slack / Microsoft Teams



# Uso de Slack / Microsoft Teams (1/3)

Desde la fundación de Hack Academy en el año 2016, hemos usado [Slack](#) como la principal herramienta de comunicación con los alumnos.

A partir del año 2020, hemos empezado a incorporar, de forma paulatina, el uso de [Microsoft Teams](#), ya que permite manejar en una misma herramienta todo lo referente a **mensajería**, intercambio de **materiales** y **videollamadas**.

Independientemente de la herramienta utilizada, es importante respetar algunas buenas prácticas para hacer la comunicación más eficiente, las cuales se listan en la siguiente diapositiva.



# Uso de Slack / Microsoft Teams (2/3)

Buenas prácticas:

- Usar su **nombre completo** + **foto** de perfil.
- Escribir mensajes en los **canales** correspondientes.
- Compartir **código** con el formato adecuado (no como “texto plano”).
- Escribir las **consultas** en un **único mensaje**. Evitar escribir varios mensajes para una misma consulta. Ej: No escribir “*Hola*”, “*¿Cómo están?*”, “*Tengo una consulta*”, “*No logré hacer funcionar...*”, en 4 mensajes diferentes.
- Respetar los **hilos** (*threads*) de comunicación al responder un mensaje.



# Uso de Slack / Microsoft Teams (3/3)

Buenas prácticas (continuación):

- Evitar hacer **consultas** a los **docentes** vía **mensajes privados**. 🙏  
Por consultas administrativas, escribir a [hola@ha.dev](mailto:hola@ha.dev).
- No sientan vergüenza de usar el **canal de dudas**. Esto permite que cualquier persona pueda responder, ¡incluso otro alumno!
- Si bien Slack y Teams se pueden usar vía la web, para lograr una mejor experiencia, recomendamos descargar las **apps** para **desktop** y **mobile**.



# Requisitos previos



# Requisitos previos

- HTML, CSS y JavaScript:
- Novedades de ES6+. Ejemplos:
  - *Arrow Functions*.
  - `const` y `let`.
  - *Spread Operator*.
- Métodos de arrays como `map`, `filter` y `reduce`. También `for...of`.
- Node.js.
- Express.
- JSON.
- Nociones de bases de datos.
- Consola / Shell.
- Git y GitHub.

Saber inglés  es una gran ventaja.



# Contenido del curso



# Contenido del curso

## Herramientas / Tecnologías:

- JavaScript (ES6+).
- Node.js.
- Express.
- MongoDB.
- MongoDB Compass.
- JWT.

## Conceptos:

- APIs (REST).
- Autenticación en APIs.
- Bases de datos No Relacionales.
- WebSockets.





# Repaso

En la clase de hoy repasaremos algunos temas importantes y necesarios para realizar el curso de Node.js Avanzado. Así nos aseguramos de que todos estemos en la “misma página”.



# Repaso de JavaScript (ES6+)

¿Qué es cada uno de estos conceptos? – 👉 Repasemos entre todos.

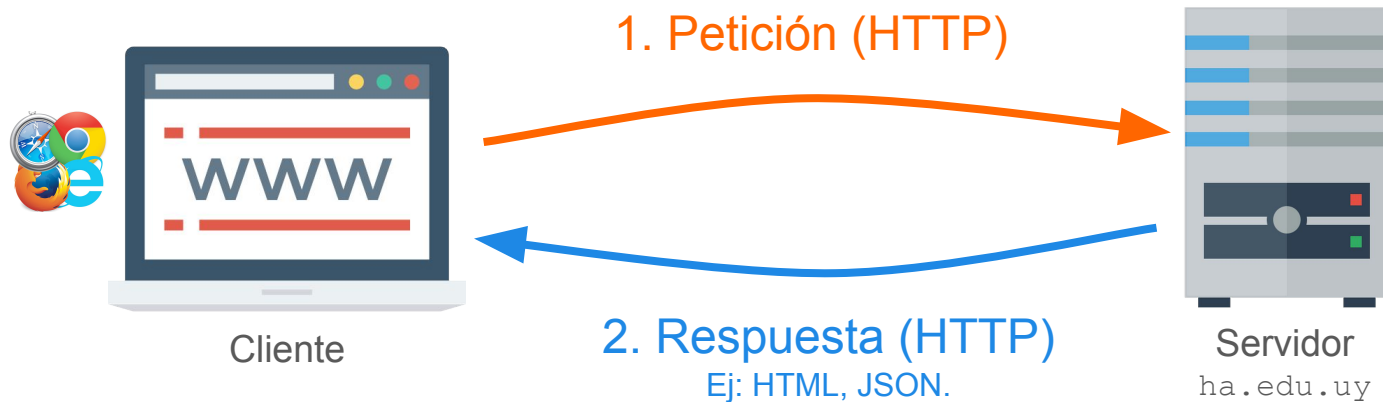
- `var`, `let` y `const` – ¿cuándo usar cada uno?
- Arrow functions – ¿qué diferencia tienen con las funciones “tradicionales”?
- `map`, `reduce`, `filter`, `some`, `every`, `indexOf` – ¿cuándo se usan?
- Spread Operator.
- Objetos y Clases.



# HTTP



# Hypertext Transfer Protocol (HTTP)



La comunicación entre un cliente y un servidor web, se realiza utilizando un **protocolo** llamado **HTTP**. A la primera llamada se le denomina petición (*request*) y a la segunda respuesta (*response*).



# HTTP – Request

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

- Una URL.
  - Ejemplo: <https://ha.dev/cursos/back-end-nodejs-avanzado>.
  - 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 – Response

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 del navegador.



# ¿Qué es Node.js?



# ¿Qué es Node.js? (1/3)



*“Es un runtime que permite ejecutar  
**JavaScript** fuera de los navegadores”.*





# ¿Qué es Node.js? (3/3)

Node.js es básicamente un **runtime** (construido sobre el motor de JavaScript V8 de Google Chrome) que permite **ejecutar JavaScript** fuera de los navegadores.

Es una herramienta que permite construir **servidores web** y aplicaciones de red **escalables y asincrónicas**.

Está escrito en C++ (mayormente) y JavaScript.

---

👉 Recomendamos descargar la versión **LTS** de <https://nodejs.org> o instalarlo vía [Homebrew](#) (Mac) o similar.



## ¿Qué es Node.js? (3/3)

*“Node.js es una infraestructura **no-bloqueante** y **orientada a eventos** para desarrollar programas con **alta concurrencia**.”*

Los programadores que utilicen Node.js pueden despreocuparse de la ocurrencia de *deadlocks*.



# 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

👉 Express se instalará como una **dependencia** en nuestro proyecto, usando el comando `npm install express`.



# Express – Rutas

La sintaxis genérica de una **ruta** es:

En archivo `index.js` o `server.js`

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

- **APP** es una instancia de Express.
- **METHOD** es un método HTTP (en minúscula). Ej: `get` o `post`.
- **PATH** es una ruta  $\approx$  la porción de la URL seguida del dominio. Ej: `“/productos”`.
- **HANDLER** es la función que se ejecuta cada vez que se recibe un *request* en la ruta especificada. También se le dice **callback**.

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



# MVC

# MVC (1/4)

Los [patrones de diseño](#) son soluciones reusables a problemas comunes y recurrentes que se dan en software.

REPASO



MVC es un **patrón de diseño arquitectónico** usado por muchos frameworks de desarrollo web (como [AdonisJS](#) en Node.js, [Laravel](#) en PHP y [Rails](#) en Ruby).

MVC propone dividir una aplicación en **3 grandes componentes** con responsabilidades bien definidas con el objetivo de fomentar el reuso de código y permitir el desarrollo en paralelo (se puede trabajar en simultáneo en cada componente).

Los componentes propuestos por MVC son:

- **Modelos.**
- **Vistas.**
- **Controladores.**



# MVC (2/4)

## Modelo:

- Es el componente central del patrón MVC.
- Expresa el comportamiento de la aplicación en términos del problema de negocio, de forma **independiente a la interfaz** (no importa si es una web, *desktop app* o *mobile app*).
- Gestiona los datos y **reglas del negocio**.
- En general, interactúa con una base de datos.
- Suele “modelar” una **entidad** del problema.  
Ej: el modelo Usuario, el modelo Proveedor, el modelo Empleado, el modelo Producto, etc.
- En general, cada modelo se coloca en un archivo con el nombre del modelo (ej: `User.js`) dentro de un directorio llamado `/models`.





# MVC (3/4)

## Vista:

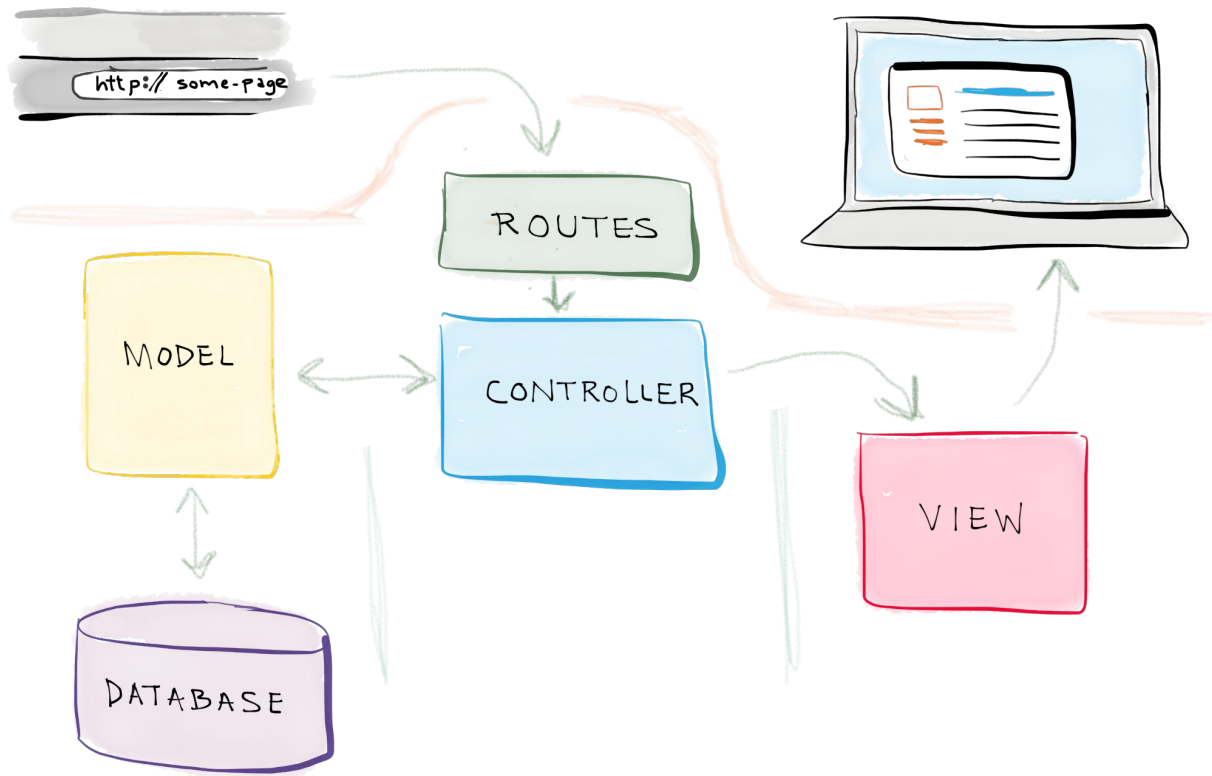
- Se encarga de presentar los datos del modelo. Es la **interfaz visual** (UI).
- Puede ser una web en HTML, una *mobile* o *desktop app* o incluso la pantalla de un cajero automático.
- En el caso de una web app “tradicional” (*server-side rendering*), las vistas se suelen colocar en un directorio llamado `/views`.

## Controlador:

- Es un **intermediario** entre la vista y el modelo.
- Recibe un *request* de un navegante a través de una URL. Es el **handler** de las rutas.
- Se comunica con los modelos para obtener la información necesaria, la organiza y se la entrega a la vista.
- En general, los controladores se colocan en un directorio llamado `/controllers`.

# MVC (4/4)

Diagrama:





# Ejercicio de Repaso



# Ejercicio de Repaso

El objetivo de esta actividad es hacer un ejercicio entre toda la clase para repasar algunos temas de Express y otros módulos.

Crear un sitio web que contenga las siguientes rutas:

- [GET] <http://localhost:3000> → Mostrar un HTML básico con un `h1` que diga “Home”.
- [GET] <http://localhost:3000/fecha> → Mostrar un texto diga “Hoy es 17 de marzo de 2021 y son las 21:56:57 (martes)” con los datos correspondientes a la fecha actual. Usar [moment](#) o [date-fns](#).
- [GET] <http://localhost:3000/multiplicar> → Ver siguiente diapositiva.
- [POST] <http://localhost:3000/multiplicar> → Ver siguiente diapositiva.

⚠ Por un tema de orden: crear un archivo llamado `routes.js` para las rutas y una carpeta `controllers` para alojar los controladores.



# Ejercicio de Repaso (cont)

Referencia visual:

### Ingresar dos números para multiplicar

Número 1

Número 2

**Multiplicar**

Al hacer click sobre el botón se debe llamar a la ruta `/multiplicar` vía POST y en la pantalla deberá aparecer el texto: “El resultado es XXXX”.



# Ejercicio de Repaso (cont)

## EXTRA:

Modificar el ejercicio de “multiplicar” de tal manera de que en lugar de ver el resultado en una nueva página, el mismo se vea debajo del formulario, luego de hacer click en el botón. No se deberá recargar la página al presionar el botón.

Se deberá seguir utilizando la URL <http://localhost:3000/multiplicar> para realizar el cálculo (la multiplicación), por lo que será necesario hacer una llamada **AJAX** desde la página del formulario y recibir el resultado a mostrar.