



Clase 47. Programación Backend

Deno: El futuro de NodeJS?



OBJETIVOS DE LA CLASE

- Introducir Deno, su configuración y utilidad.
- Crear códigos simples con Deno.
- Conocer y utilizar el módulo Denon.

CRONOGRAMA DEL CURSO

Clase 46



Introducción a
frameworks de desarrollo
backend - Parte II

Clase 47



Deno: El futuro de
Nodejs?

Clase 48



DENO

CODER HOUSE



¿De qué se trata?

- **Deno** es un entorno de ejecución de Javascript y TypeScript que permite ejecutar código en estos lenguajes fuera del contexto del navegador. Al igual que Node, está basado en el motor de ejecución de Javascript V8, el que viene incorporado en el navegador Chrome.
- Aunque principalmente se usa para programación del lado del servidor, creación de servicios web y programas de consola, con Deno podemos hacer todo tipo de programas.
- Además, tiene una configuración predeterminada muy potente para interpretar TypeScript. Esto permite escribir código TypeScript y ejecutar directamente sin necesidad de hacer ningún paso en particular. De este modo, en un proyecto podremos mezclar módulos escritos con Javascript y TypeScript sin problema alguno.



Características principales

- Seguro por defecto, sin acceso a archivos, red o entorno de trabajo, a menos que esto esté habilitado.
- Soporte para TypeScript.
- Se envía un solo ejecutable (*deno*).
- Cuenta con utilidades integradas como por ejemplo, un inspector de dependencias (*deno info*) y un formateo de código (*deno fmt*).
- Tiene un conjunto de módulos estándar previamente auditados los cuales están garantizados para trabajar con Deno.
- Si se quiere, los Scripts pueden ser agrupados en un solo archivo Javascript.

DENO vs NODE



Comparación con Node



Es más seguro: Cuando ejecutas un programa con Node éste tiene todos los permisos para hacer cualquier cosa en tu equipo. Con Deno el desarrollador es capaz de otorgar solamente los permisos que sean absolutamente necesarios.



No usa npm: Todas las dependencias las instala a través de la URL donde está la dependencia en sí, por lo que es capaz de funcionar sin depender de un repositorio central.



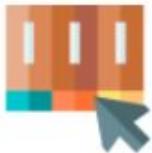
Usa los módulos ES6: En lugar de CommonJS como usa NodeJS, esto lo hace mucho más cercano al estándar de Javascript.



Comparación con Node



Funciona con promesas: Para gestionar los procesos asíncronos, Deno usa promesas en lugar de funciones callback, por lo que el código que se puede realizar es más legible. Además puedes usar await en cualquier punto del código, a cualquier nivel, sin necesidad de declarar una función async.



Ofrece más soporte a las API web: Dispone de manera predeterminada de librerías o APIs del navegador, como fetch, que no están disponibles en Node.



Comparación con Node - Resumen

DENO	NODE
Usa V8	Usa V8
Escrito en RUST y JavaScript	Escrito en C++ y JavaScript
Ejecuta en sandbox con acceso limitado: se requiere un comando explícito en el script	La cuestión del acceso se limita a los derechos de acceso específicos de un usuario que ejecuta el script
Módulos descentralizados - cargados desde URL	NPM
ES Module	CommonJS
La API y la biblioteca estándar aprovechan al máximo ES y Promise	La API y la biblioteca estándar están basadas en callbacks

INSTALACIÓN



Instalación



- Vamos a la página principal de Deno, <https://deno.land/>. Allí, además de tener la documentación oficial, vemos que podemos instalar Deno directamente por comandos de consola, según el sistema operativo que estemos usando.
- Para instalar Deno en Windows a través de PowerShell, el comando es:

```
$ iwr https://deno.land/x/install/install.ps1 -useb | iex
```

Deno - A secure runtime for JavaScript

deno.land

Installation

Deno ships as a single executable with no dependencies. You can install it using the installers below, or download a release binary from the [releases page](#).

Shell (Mac, Linux):

```
$ curl -fsSL https://deno.land/x/install/install.sh | sh
```

PowerShell (Windows):

```
$ iwr https://deno.land/x/install/install.ps1 -useb | iex
```

Homebrew (Mac):

```
$ brew install deno
```

Chocolatey (Windows):

```
$ choco install deno
```

Scoop (Windows):

```
$ scoop install deno
```



Instalación



- Ingresamos entonces el comando de instalación en nuestra consola Powershell.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\Users> iwr https://deno.land/x/install/install.ps1 -useb | iex
```

- Al comenzar la instalación nos debería aparecer algo como esto:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Escribiendo solicitud web
Escribiendo secuencia de solicitud... (Número de bytes escritos: 7650773)
```



Instalación



```
PS Windows PowerShell
PS C:\Users> iwr https://deno.land/x/install/install.ps1 -useb | iex
Deno was installed successfully to C:\Users\.deno\bin\deno.exe
Run 'deno --help' to get started

PS C:\Users> deno --version
deno 1.11.5 (release, x86_64-pc-windows-msvc)
v8 9.1.269.35
typescript 4.3.2

PS C:\Users> deno --help
deno 1.11.5
A Secure JavaScript and TypeScript runtime

Docs: https://deno.land/manual
Modules: https://deno.land/std/ https://deno.land/x/
Bugs: https://github.com/denoland/deno/issues

To start the REPL:
deno

To execute a script:
deno run https://deno.land/std/examples/welcome.ts

To evaluate code in the shell:
deno eval "console.log(30933 + 404)"

USAGE:
deno [OPTIONS] [SUBCOMMAND]

OPTIONS:
-h, --help
    Prints help information
-L, --log-level <log-level>
    Set log level [possible values: debug, info]
-q, --quiet
    Suppress diagnostic output
    By default, subcommands print human-readable diagnostic messages to stderr.
    If the flag is set, restrict these messages to errors.
--unstable
    Enable unstable features and APIs
-V, --version
    Prints version information
```

- Una vez que finaliza la instalación, podemos ver la versión instalada con el comando **deno --version**.
- Luego, con el comando **deno --help** nos salen los comandos y las funciones que podemos hacer en consola.



Instalación



The screenshot shows the Deno website on the left and a browser's developer tools console on the right. The website features the Deno logo, navigation links (Install, Manual, Blog, Runtime API, Standard Library, Third Party Modules), and a main heading "Deno" with the subtitle "A secure runtime for JavaScript and TypeScript." A blue button indicates the version "v1.11.5". The developer tools console displays a command-line interface with the following text:
```
> ["no", "de"].reverse()
< ▷(2) ["de", "no"]
> "node".split('').sort().join('')
< "deno"
>
```
The last two lines of the console output are highlighted with a red border.



Finalmente, como curiosidad, podemos ver que “node” es la inversa de “deno” en sílabas y además “deno” son las letras de “node” en orden alfabético.

EJEMPLOS SIMPLES



Deno con TypeScript nativo



- Deno usa TypeScript como lenguaje por defecto por lo que no es necesario ningún tipo de configuración adicional.
- Un simple “hola mundo” en TypeScript es suficiente para comenzar a probar Deno:

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "EJEMPLOS" containing a file "d1.ts".
- Editor:** Displays the contents of "d1.ts":

```
TS d1.ts
1 const sayHelloTo = (name: string): string => {
2   return `Hello ${name} !`;
3 }
4
5 console.log(sayHelloTo("Coderhouse!"));
```

- Terminal:** Shows the output of running the code with Deno:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS> deno run d1.ts
Hello Coderhouse!
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS>
```

- Para ejecutar el código usamos el comando **deno run dt.ts**.

IMPORTACIÓN REMOTA DE DEPENDENCIA



Importación remota de dependencia



- Vamos a utilizar el módulo datetime de Deno. El cual vamos a importar de forma remota mediante la URL.
- El módulo es el siguiente:

The screenshot shows the Deno Standard Library interface. At the top, there's a navigation bar with links for Install, Manual, Blog, Runtime API, Standard Library, Third Party Modules, and a user icon. Below the navigation, a message says "To ensure a good user experience, we make use of functional cookies." with an "Ok" button. The main content area has a header "deno.land / std@0.100.0 / datetime". On the left, there's a file tree for the /datetime module, listing files like formatter.ts, mod.ts, README.md, test.ts, and tokenizer.ts. To the right of the file tree, there's a "Repository" section with a star icon and the version "0.100.0". Further down, there's a "Version Info" section with a timestamp "29/6/2021 12:47:22". At the bottom, there's a "Usage" section with a brief description: "Simple helper to help parse date strings into `Date`, with additional functions."



Importación remota de dependencia



- Luego, vamos al código y con “*import*” y la URL podemos importar la dependencia en nuestro archivo como vemos en la imagen. Recordar que Deno no admite *require*.

```
File Edit Selection View Go Run Terminal Help
d2.ts - ejemplosDenoTS - Visual Studio Code
EXPLORER    TS d2.ts  1 ×
OPEN EDITORS > ...
EJEMPLOSDENOTS
TS d1.ts
TS d2.ts  1
d2.ts > ...
1 import { parse } from "https://deno.land/std@0.100.0/datetime/mod.ts"; // Url REMOTA!
2
3 const myDate = parse(")
4
5 console.log(myDate); /
6
7
Follow link (ctrl + click)
An import path cannot end with a '.ts' extension. Consider importing
'https://deno.land/std@0.100.0/datetime/mod' instead. ts(2691)
View Problem (Alt+F8) No quick fixes available
```

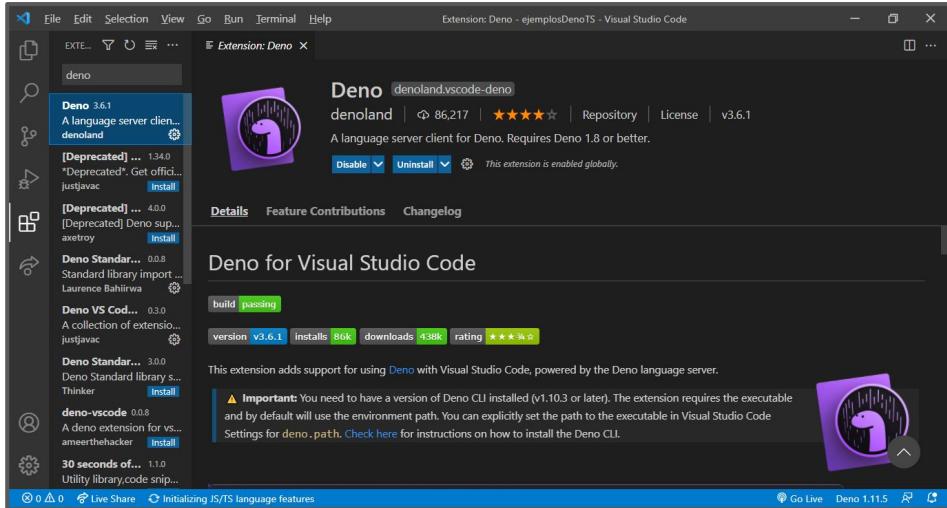
- Vemos que nos da un error en la URL. Vamos a ver por qué en la siguiente diapositiva.



Importación remota de dependencia



- Por un lado, debemos instalar una extensión en el Visual Studio Code para poder usar Deno sin que nos genere errores como este.
- La extensión a instalar se llama “Deno”.





Importación remota de dependencia



- Por otro lado, para que nos deje de aparecer el error, creamos una carpeta llamada **.vscode** y dentro de esta un archivo llamado **settings.json**.
- En este archivo, lo importante es definir “**deno.enable**” como **true** para que admita las extensiones *.ts* en las URLs de la importación remota de módulos de Deno.

```
1  "deno.enable": true,
2  "deno.suggest.imports.hosts": [
3    "https://deno.land"
4  ]
```



Importación remota de dependencia



- Ahora sí, vemos el código de nuestro ejemplo sin el error en la URL del *import*.
- La librería le cambia el formato a una fecha. En este caso, le cambiamos el formato e imprimimos en consola esa fecha modificada.

```
File Edit Selection View Go Run Terminal Help d2.ts - ejemplosDenoTS - Visual Studio Code
EXPLORER OPEN EDITORS EJEMPLOSDENOTS .vscode settings.json d1.ts d2.ts
d2.ts > ...
1 import { parse } from "https://deno.land/std@0.100.0/datetime/mod.ts"; // Url REMOTA!
2
3 const myDate = parse("04-07-2021", "dd-mm-yyyy");
4
5 console.log(myDate); // 2020-03-01T23:00:00.000Z
6
7
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE powershell + ^ x
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS> deno run d2.ts
Check file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/d2.ts
2021-07-04T03:07:00.000Z
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS>
```

Texto con color en Deno



Importación remota de dependencia



- Deno tiene una dependencia llamada **Colors**. Esta tiene diferentes funciones para generar texto de colores.
- Funciones de propiedad: son parte del módulo de colores, pero establecen la propiedad como negrita, cursiva, etc.
- Funciones de color de texto directo: reciben el nombre de un color y se pueden usar directamente para codificar un texto con colores.
- Funciones de color de fondo directo: establecen el fondo codificado por colores en lugar del texto.
- Funciones RGB: hay cuatro funciones de este tipo que pueden codificar con colores el texto o el fondo en la combinación de colores RGB dada. Estas funciones son flexibles ya que el color se puede controlar a través de la entrada RGB.a generar textos de colores.



Importación remota de dependencia



- Vemos el módulo Colors. Para importarlo podemos usar la URL "<https://deno.land/std/fmt/colors.ts>"

To ensure a good user experience, we make use of functional cookies.

Ok

 **Deno**
Standard Library

Install Manual Blog Runtime API Standard Library Third Party Modules 

[deno.land / std@0.100.0 / fmt / colors.ts](https://deno.land/std@0.100.0/fmt/colors.ts)

colors.ts [Raw](#) [Repository](#) [View Documentation](#)

```
1 // Copyright 2018-2021 the Deno authors. All rights reserved. MIT license.
2 // A module to print ANSI terminal colors. Inspired by chalk, kleur, and colors
3 // on npm.
4 //
5 // ...
6 // import { bgBlue, red, bold } from "https://deno.land/std/fmt/colors.ts";
7 // console.log(bgBlue(red(bold("Hello world!"))));
8 // ...
9 //
10 // This module supports `NO_COLOR` environmental variable disabling any coloring
11 // if `NO_COLOR` is set.
12 //
13 // This module is browser compatible.
14 //
15 const noColor = globalThis.Deno?.noColor ?? true;
16
17 interface Code {
18   open: string;
```

std
Deno standard library  [denoland/deno_std](#) ★ 1544 [0.100.0](#)

Version Info  29/6/2021 12:47:22

External Dependencies No external dependencies 



Importación remota de dependencia



- Podemos importar algunos de sus métodos, por ejemplo los nombres de los colores, o “bg + nombre color” que es para el color de fondo. Vemos en consola la ejecución de este ejemplo, como obtenemos los textos con color.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files: 'colors.ts' (selected), '.vscode', and 'settings.json'.
- Code Editor:** Displays the following TypeScript code:

```
1 import {  
2   red, green, bgYellow, bgBlack, bgWhite, bold  
3 } from "https://deno.land/std@0.100.0/fmt/colors.ts"  
4  
5 console.log(bgYellow(bold(red('Hello Deno!'))));  
6 console.log(bgWhite(bold(green('Hello Deno!'))));
```
- Terminal:** Shows the output of running the script:

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\2> deno run .\colors.ts  
Hello Deno!  
Hello Deno!
```
- Status Bar:** Shows the file path 'In 4, Col 1', line count 'Spaces: 4', encoding 'UTF-8', file type 'CRLF', language 'TypeScript', live preview status 'Go Live', version '4.3.5', Deno version '1.11.5', and Prettier status.

OBJETO GLOBAL DENO



Objeto global Deno

- El objeto llamado Deno es el objeto global de este entorno de ejecución. Es similar al objeto process de Node.
- En este ejemplo, simplemente imprimimos en consola este objeto.

```
File Edit Selection View Go Run Terminal Help d3.ts - ejemplosDenoTS - Visual Studio Code
EXPLORER OPEN EDITORS EJEMPLOSDENOTS .vscode settings.json d1.ts d2.ts d3.ts
TS d3.ts
1 console.log(Deno);
2
3 /* { Buffer, readAll, readAllSync, writeAll, writeAllSync, build, chmodSync, chmod, chownSync, chdir, inspect, copyFileSync, copyFile, cwd, applySourceMap, ErrorKind, DenoError, File,
4
5
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
powershell + ^ x
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS> deno run d3.ts
Check file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/d3.ts
{
  core: {
    opcall: [Function],
    setMacrotaskCallback: [Function],
    evalContext: [Function],
    encode: [Function],
    decode: [Function],
    serialize: [Function],
    deserialize: [Function],
    getPromiseDetails: [Function],
    getProxyDetails: [Function],
    memoryUsage: [Function],
    opAsync: [Function: opAsync],
    opSync: [Function: opSync],
    ops: [Function: ops],
    close: [Function: close],
    print: [Function: print],
  }
}
Ln 5, Col 1  Spaces: 4  UTF-8  CRLF  TypeScript  Go Live
```

CODER HOUSE



Deno.args

- Esta es la forma que tiene Deno de levantar parámetros de la línea de comandos. Similar a process.argv, con Deno.args podemos levantar los parámetros.

```
// El objeto Deno tiene los argumentos usados en la ejecucion
// justo como process.argv
console.log(Deno.args)
```

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\denoPlayground-main> deno run .\denoArgs.ts 1 2 3 4
[ "1", "2", "3", "4" ]
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\denoPlayground-main>
```



Deno.env



- Esta es la forma que tiene Deno de levantar variables de entorno de la línea de comandos. Similar a process.env. Requiere adicionalmente ejecutar el programa con el flag: **--allow-env**. Deno.env nos devuelve un diccionario con un par clave-valor para cada variable de entorno cargada.

Ejemplo:

```
const port = Number(Deno.env.get("PORT")) || 8080;  
console.log(port)
```

MANEJO DE ARCHIVOS EN DENO



Deno - Manejo de archivos

- Escribir texto en un archivo con Deno es muy sencillo. Simplemente usamos la función writeTextFile del objeto Deno:

```
await Deno.writeTextFile("test.txt", "Hola deno facil!");
```

- Cuando lo ejecutamos, nos da un error ya que para usar Deno.writeFile necesitamos permisos (Deno trabaja de esta forma).

```
$ deno run 2-ejemploWriteTextFile.ts  
  
error: Uncaught (in promise) PermissionDenied: Requires write access to "test.txt", run again with the --allow-write flag  
await Deno.writeTextFile("test.txt", "Hola deno facil!");  
^  
at async open (deno:runtime/js/40_files.js:51:17)  
at async writeFile (deno:runtime/js/40_write_file.js:64:18)  
at async file:///clase-24-deno/ejemplosClase/2-ejemploWriteTextFile.ts:9:1
```

- Para tener entonces el permiso para ejecutar ese proceso, ejecutamos el código con el flag: **--allow-write**. Con este se ejecuta correctamente.



Objeto global Deno - procesos



- Vemos en el código que estamos usando `await` sin estar dentro de un bloque que sea `async`. Esto como mencionamos es posible en Deno, y lo es gracias a **Top-level await**.

The screenshot shows a Visual Studio Code interface with the following details:

- File Path:** d4.ts - ejemplosDenoTS - Visual Studio Code
- Code Content:**

```
const encoder = new TextEncoder();
const data = encoder.encode('Hello Deno!');
await Deno.writeFile('test.txt', data);
```
- Editor Status:** Line 4, Col 1, Spaces: 4, UTF-8, CRLF, TypeScript, Go Live, 4.3.2, Deno 1.11.5, Prettier
- Bottom Bar:** Live Share
- Background Information:** A blue banner from deno.com explains "Top-level await" as enabling the use of `await` outside of `async` functions, acting like a big `async` function.

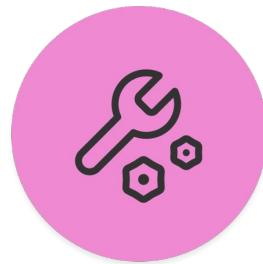


Deno - Manejo de archivos

- Leer texto de un archivo con Deno también es muy sencillo. Simplemente usamos la función `readTextFile` del objeto Deno:

```
const text = await Deno.readTextFile("test.txt");  
console.log(text)
```

- Para que el código no lance un error de permisos, ejecutamos el código con el flag: `--allow-read`.



PROYECTO EN DENO

Crea un proyecto en Deno configurando localmente.

Tiempo: 10 minutos

PROYECTO EN DENO

Desafío
generico



Tiempo: 10 minutos

Crear un proyecto en Deno, configurando localmente Visual Studio Code para que interprete la sintaxis y características de la plataforma.

- Esta aplicación recibirá una cantidad ilimitada de parámetros numéricos y deberá determinar el valor mínimo, el máximo y el promedio entre todos ellos.
- Los resultados se almacenarán en un archivo llamado resultados.dat, sin utilizar librerías externas, respetando el siguiente formato:

Números: 4,5,33,7,94,56,.....

Mínimo: x

Máximo: X

Promedio: Y

- En la consola se representarán los datos de la misma forma pero agregando además color al texto: la palabra mínimo y su valor se imprimirán en amarillo, el máximo en rojo y el promedio en verde, todos con fondo blanco.

CODER HOUSE

SERVIDOR EN DENO



Servidor en Deno

- Para poder crear un servidor, necesitamos el módulo **http** de Deno.

To ensure a good user experience, we make use of functional cookies. [Ok](#)

Deno
Standard Library

Install Manual Blog Runtime API Standard Library Third Party Modules [?](#)

deno.land / std@0.100.0 / http

Repository

/http	
testdata	10 KB
_io_test.ts	14 KB
_io.ts	11 KB
_mock_conn.ts	669 B
bench.ts	506 B
cookie_test.ts	7 KB
cookie.ts	6 KB
file_server_test.ts	14 KB
file_server.ts	12 KB
http_status.ts	6 KB
mod.ts	167 B

std
Deno standard library

denoland/deno_std
★ 1533
0.100.0

Version Info
29/6/2021 12:47:22



Servidor en Deno

- Tenemos entonces el código del servidor.
- Vemos que el error en la URL de la dependencia que nos dice es que no esta no se encuentra en caché (las dependencias se instalan una vez que se ejecuta el código por primera vez).

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** d5.ts - ejemplosDenoTS - Visual Studio Code.
- Explorer View:** Shows a tree structure with a folder "EJEMPLOSDENOTS" containing ".vscode", "settings.json", "d1.ts", "d2.ts", "d3.ts", "d4.ts", "d5.ts" (which is currently selected), and "test.txt".
- Code Editor:** Displays the following TypeScript code:

```
1 import { serve } from "https://deno.land/std/http/mod.ts";
2 const s = serve({ port
3 for await (const req o
4 | req.respond({ body:
5 }
```

A tooltip is visible over the URL "https://deno.land/std/http/mod.ts", stating "Follow link (ctrl + click)" and "Uncached or missing remote URL: https://deno.land/std/http/mod.ts". It also includes "View Problem (Alt+F8)" and "Quick Fix... (Ctrl+.)".



Servidor en Deno



- Ejecutamos el código, y vemos que se está descargando la dependencia de http.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the "EJEMPLOSDENOS" folder: .vscode, d1.ts, d2.ts, d3.ts, d4.ts, and d5.ts (the active file).
- Terminal:** Running a Windows PowerShell terminal with the command `deno run d5.ts`. The output shows Deno downloading various modules from `https://deno.land/std/http/mod.ts` and its dependencies.
- Code Editor:** Displays the content of d5.ts, which imports `serve` from `https://deno.land/std/http/mod.ts` and creates a server at port 8000.
- Status Bar:** Shows the file has 1 change, is live shared, and includes standard VS Code status icons.

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS> deno run d5.ts
Download https://deno.land/std/http/mod.ts
Warning Implicitly using latest version (0.100.0) for https://deno.land/std/http/mod.ts
Download https://deno.land/std@0.100.0/http/mod.ts
Download https://deno.land/std@0.100.0/http/cookie.ts
Download https://deno.land/std@0.100.0/http/http_status.ts
Download https://deno.land/std@0.100.0/http/server.ts
Download https://deno.land/std@0.100.0/async/mod.ts
Download https://deno.land/std@0.100.0/io/bufio.ts
Download https://deno.land/std@0.100.0/http/_io.ts
Download https://deno.land/std@0.100.0/_util/assert.ts
Download https://deno.land/std@0.100.0/bytes/bytes_list.ts
Download https://deno.land/std@0.100.0/io/util.ts
Download https://deno.land/std@0.100.0/bytes/mod.ts
Download https://deno.land/std@0.100.0/textproto/mod.ts
Download https://deno.land/std@0.100.0/async/deferred.ts
```



Servidor en Deno



- Sin embargo, la ejecución nos da un error ya que no tenemos el permiso nuevamente.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files d1.ts, d2.ts, d3.ts, d4.ts, and d5.ts.
- Editor:** The d5.ts file is open, containing the following code:

```
1 import { serve } from "https://deno.land/std/http/mod.ts";
2 const s = serve({ port: 8000 });
3 for await (const req of s) {
4   req.respond({ body: "Hello Deno server!" });
5 }
```
- Terminal:** Shows the command `deno run d5.ts` being executed, resulting in the following output:

```
Download https://deno.land/std@0.100.0/async/tee.ts
Download https://deno.land/std@0.100.0/async/pool.ts
Download https://deno.land/std@0.100.0/io/buffer.ts
Download https://deno.land/std@0.100.0/testing/asserts.ts
Download https://deno.land/std@0.100.0/testing/_diff.ts
Download https://deno.land/std@0.100.0/fmt/colors.ts
Download https://deno.land/std@0.100.0/io/types.ts
Check file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/d5.ts
error: Uncaught PermissionDenied: Requires net access to "0.0.0.0:8000", run again with the --allow-net flag
const listener = Deno.listen(addr);
^
at deno:core/core.js:86:46
at unwrapOpResult (deno:core/core.js:106:13)
at Object.opSync (deno:core/core.js:120:12)
at opListen (deno:runtime/js/30_net.js:18:17)
at Object.listen (deno:runtime/js/30_net.js:184:17)
at serve (https://deno.land/std@0.100.0/http/server.ts:303:25)
at file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/d5.ts:2:11
```
- Status Bar:** Shows the file path `PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS>`, the current file `d5.ts`, and the status bar with various developer tools.



Servidor en Deno



- Cuando ejecutamos el código con los permisos (`--allow-net`) nos sale el cartel para permitir el acceso.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files like `d1.ts`, `d2.ts`, `d3.ts`, `d4.ts`, and `d5.ts`.
- Terminal:** Displays the command `deno run --allow-net d5.ts` being run.
- Message Bar:** Shows an alert from Windows Defender: "Firewall de Windows Defender bloqueó algunas características de esta aplicación".
- Windows Firewall Dialog:** A modal window titled "Firewall de Windows Defender bloqueó algunas características de esta aplicación" with the following details:
 - Nombre: Deno: A secure runtime for JavaScript and TypeScript
 - Editor: Desconocido
 - Ruta de acceso: C:\users\educionit\deno\bin\deno.exe

Checkboxes:
Permitir que Deno: A secure runtime for JavaScript and TypeScript se comunique en
 Redes privadas, como las domésticas o del trabajo
 Redes públicas, como las aeropuertos y cafeterías (no se recomienda porque estas redes públicas suelen tener poca seguridad o carecer de ella)
- Status Bar:** Shows the command `deno run --allow-net d5.ts` again with a red arrow pointing to it.



Servidor en Deno



- Finalmente, si vamos al navegador en el puerto 8080 vemos que funciona nuestro servidor.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the "EJEMPLOSDENOTS" folder: .vscode, settings.json, d1.ts, d2.ts, d3.ts, d4.ts, and d5.ts (the active file).
- Editor:** Displays the content of d5.ts:

```
import { serve } from "https://deno.land/std@0.100.0/http/mod.ts";
const s = serve({ port: 8000 });
for await (const req of s) {
  req.respond({ body: "Hello Deno server!" });
}
```
- Terminal:** Shows the command "deno run --allow-net d5.ts" being run in the background, with output indicating the server is running at "localhost:8000".
- Browser Preview:** A browser window shows the response "Hello Deno server!".
- Status Bar:** Shows the file path "C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenots/d5.ts", line 5, column 2, and other status information like "Live Share".

TESTS EN DENO



Tests en Deno



- Vamos a usar ahora la dependencia **testing** de Deno para realizar test en nuestro código.

To ensure a good user experience, we make use of functional cookies. [Ok](#)

Deno
Standard Library

deno.land / std@0.100.0 / testing

/testing [Repository](#)

Search files...

_diff_test.ts	3 KB
_diff.ts	9 KB
asserts_test.ts	22 KB
asserts.ts	18 KB
bench_example.ts	720 B
bench_test.ts	11 KB
bench.ts	11 KB
README.md	7 KB

std
Deno standard library
[denoland/deno_std](#)
★ 1533
0.100.0

Version Info
29/6/2021 12:47:22

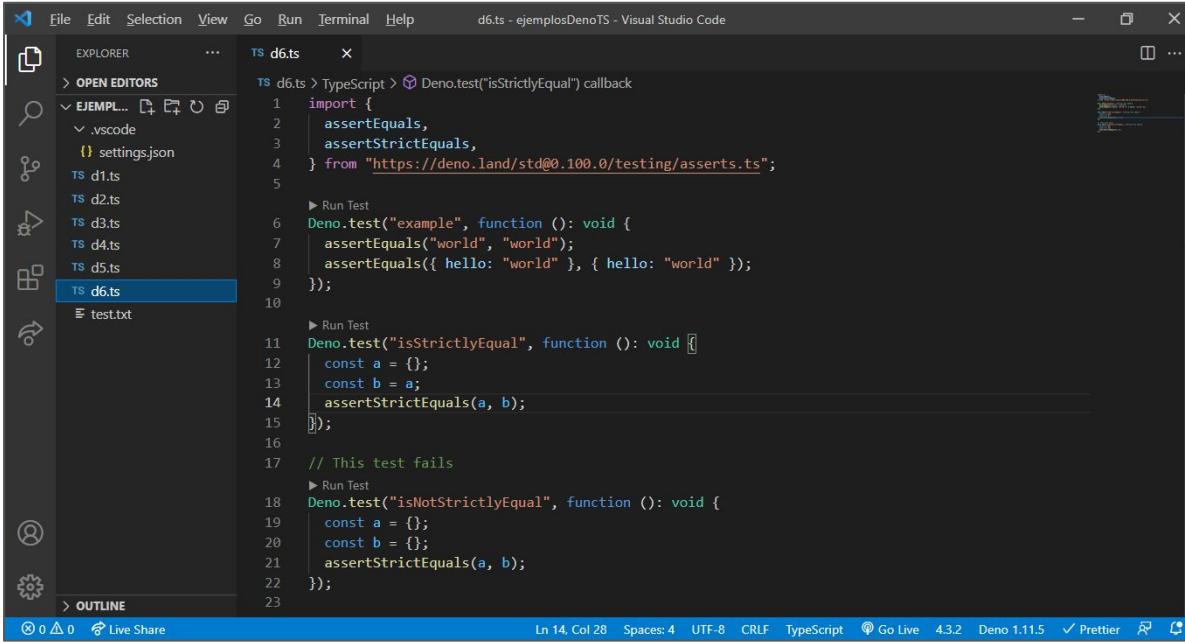
README.md [Raw](#) [Repository](#)



Tests en Deno



- Vemos en el código que directamente usamos `Deno.test()` y Deno va a testear la función que le pongamos dentro.



The screenshot shows a Visual Studio Code interface with a dark theme. The left sidebar has icons for Explorer, Search, Open Editors, and Outline. The Explorer panel shows a folder named 'EJEMP...' containing files: .vscode, settings.json, d1.ts, d2.ts, d3.ts, d4.ts, d5.ts, and d6.ts. The 'd6.ts' file is selected. The main editor area displays the following TypeScript code:

```
TS d6.ts > TypeScript > Deno.test("isStrictEqual") callback
1 import {
2     assertEquals,
3     assertStrictEquals,
4 } from "https://deno.land/std@0.100.0/testing/asserts.ts";
5
6 ▶ Run Test
7 Deno.test("example", function (): void {
8     assertEquals("world", "world");
9     assertEquals({ hello: "world" }, { hello: "world" });
10 });
11 ▶ Run Test
12 Deno.test("isStrictEqual", function (): void {
13     const a = {};
14     const b = a;
15     assertStrictEquals(a, b);
16 });
17 // This test fails
18 ▶ Run Test
19 Deno.test("isNotStrictEqual", function (): void {
20     const a = {};
21     const b = {};
22     assertStrictEquals(a, b);
23 });

Ln 14, Col 28   Spaces: 4   UTF-8   CRLF   TypeScript   Go Live   4.3.2   Deno 1.11.5   Prettier
```



Tests en Deno



- Para correr los test, directamente clickeamos en “Run test” encima del código de cada uno de ellos.
 - Ejecutamos el primero y vemos que pasa correctamente.



File Edit Selection View Go Run Terminal Help d6.ts - ejemplosDenoTS - Visual Studio Code

EXPLORER OPEN EDITORS EJEMPLOSDENOTS .vscode d1.ts d2.ts d3.ts d4.ts d5.ts d6.ts test.txt

TS d6.ts x

TS d6.ts > TypeScript > Deno.test("isStrictEqual") callback

```
1 import {  
2     assertEquals,  
3     assertStrictEquals,  
4 } from "https://deno.land/std@0.100.0/testing/asserts.ts";  
5  
6 ► Run Test  
7 Deno.test("example", function (): void {  
8     assertEquals("world", "world");  
9     assertEquals({ hello: "world" }, { hello: "world" });  
10});  
11  
12 ► Run Test  
13 Deno.test("isStrictEqual", function (): void {  
14     const a = {};  
15     const b = a;
```

TUTORIALS WORKSPACE DOCUMENTS RECENT FILES

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

soBackend\Clase47\ejemplosDenoTS\d6.ts <

running 1 test from file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/d6.ts

test example ... ok (7ms)

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 2 filtered out (36ms)

Terminal will be reused by tasks, press any key to close it.

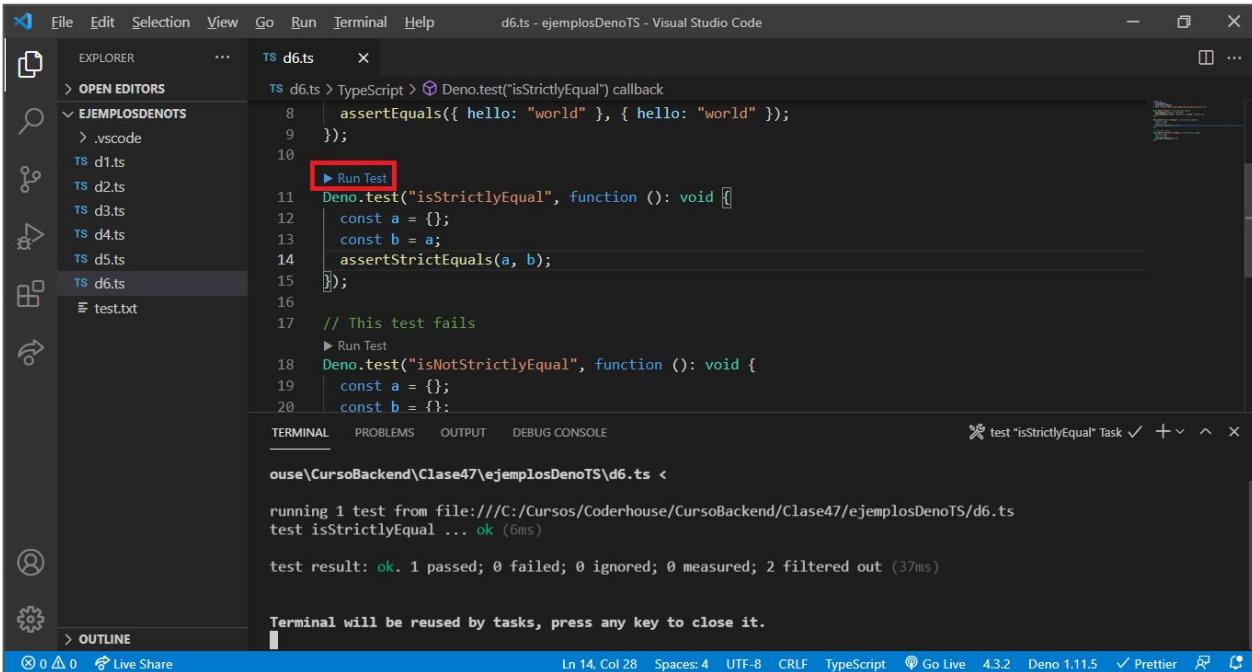
CODER HOUSE



Tests en Deno



- Ejecutamos el siguiente test, para chequear la igualdad estricta, y vemos que también pasa correctamente.



```
File Edit Selection View Go Run Terminal Help
d6.ts - ejemplosDenoTS - Visual Studio Code
EXPLORER EJEMPLOSDENOTS .vscode d1.ts d2.ts d3.ts d4.ts d5.ts d6.ts test.txt
TS d6.ts > TypeScript > Deno.test("isStrictEqual") callback
8 | assertEquals({ hello: "world" }, { hello: "world" });
9 |
10 |
11 | ► Run Test
12 | Deno.test("isStrictEqual", function (): void {
13 |   const a = {};
14 |   const b = a;
15 |   assertStrictEquals(a, b);
16 |
17 | // This test fails
18 | ► Run Test
19 | Deno.test("isNotStrictEqual", function (): void {
20 |   const a = {};
21 |   const b = {};
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
use\CursoBackend\Clase47\ejemplosDenoTS\d6.ts <
running 1 test from file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/d6.ts
test isStrictEqual .... ok (6ms)

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 2 filtered out (37ms)

Terminal will be reused by tasks, press any key to close it.
```

Ln 14, Col 28 Spaces: 4 UTF-8 CRLF TypeScript Go Live 4.3.2 Deno 1.11.5 Prettier



Tests en Deno



- Finalmente, ejecutamos el último test y el mismo falla. Lo hicimos a propósito en este caso, para ver un caso de falla de test.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files d1.ts, d2.ts, d3.ts, d4.ts, d5.ts, d6.ts, and test.txt.
- Editor:** The d6.ts file is open, containing the following code:

```
15  });
16
17 // This test fails
18 Deno.test("isNotStrictEqual", function (): void {
19   const a = {};
20   const b = {};
21   assertStrictEquals(a, b);
22 });
23
```
- Terminal:** Shows the output of running the test:

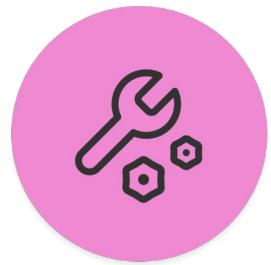
```
at exitSanitizer (deno:runtime/js/40_testing.js:85:15)
at runTest (deno:runtime/js/40_testing.js:199:13)
at Object.runTests (deno:runtime/js/40_testing.js:244:13)
at file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/$deno$test.js:1:27

failures:

isNotStrictEqual

test result: FAILED. 0 passed; 1 failed; 0 ignored; 0 measured; 2 filtered out (44ms)

The terminal process "C:\Users\EducacionIT\.deno\bin\deno.EXE 'test', '--allow-all', '--filter', 'isNotStrictEqual', 'c:\Cursos\Coderhouse\CursoBackend\Clase47/ejemplosDenoTS/d6.ts'" failed to launch (exit code: 1).
```
- Status Bar:** Shows the terminal will be reused by tasks, and the status bar at the bottom indicates: Ln 14, Col 28, Spaces: 4, UTF-8, CRLF, TypeScript 4.3.2, Deno 1.11.5, Prettier.



SERVIDOR EN DENO

Tiempo: 10 minutos

SERVIDOR EN DENO

Desafío
generico



Tiempo: 10 minutos

Realizar un servidor en Deno que reciba por query params una frase y responda otra frase con las palabras invertidas.

Ejemplo del formato de query:

`http://localhost:8080?frase=Hola servidor Deno!`

Respuesta del servidor:

Deno! servidor Hola

- Utilizar la librería http de Deno.



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

MÁS EJEMPLOS USANDO DENO



Hola mundo

- Vemos en este simple ejemplo, que para ejecutarlo, no necesitamos permisos de ningún tipo, simplemente podemos usar el comando **deno run <name.ts>**.

The screenshot shows a Visual Studio Code interface. On the left is the Explorer sidebar with a tree view showing a folder named 'EJEMPLOSDENOTS' containing a '.vscode' folder and a 'settings.json' file. Below that is a 'helloworld.ts' file. The main editor area contains the following TypeScript code:

```
const message = 'Hola mundo'
console.log(message);
```

At the bottom, the terminal window is open in PowerShell and displays the output of running the script:

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\ejemplosDenoTS> deno run helloworld.ts
Check file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/ejemplosDenoTS/helloworld.ts
Hola mundo
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\ejemplosDenoTS>
```

The status bar at the bottom of the code editor shows: Line 1, Col 1 | Spaces: 4 | UTF-8 | CRLF | TypeScript | Go Live | 4.3.2 | Deno 1.11.5 | Prettier.



Usando *fetch*



- Por otro lado, si usamos *fetch*, para por ejemplo consumir un endpoint de una API, sí debemos especificarle el permiso de red para que funcione. En este caso, el comando para ejecutarlo con permiso de red es: `deno run --allow-net <name.ts>`.

```
File Edit Selection View Go Run Terminal Help
do-fetch.ts - ejemplosDenoTS - Visual Studio Code
EXPLORER OPEN EDITORS
EJEMPLOSDENOTS
  .vscode
    settings.json
  do-fetch.ts
  helloworld.ts
TS do-fetch.ts > ...
1 const response = await fetch('https://jsonplaceholder.typicode.com/todos/1')
2 const json = await response.json();
3 console.log(json);

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
powershell + ^ x
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\ejemplosDenoTS> deno run .\do-fetch.ts
Check file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/ejemplosDenoTS/do-fetch.ts
error: Uncaught PermissionDenied: Requires net access to "jsonplaceholder.typicode.com", run again with the --allow-net flag
const response = await fetch('https://jsonplaceholder.typicode.com/todos/1')

at deno:core/core.js:86:46
at unwrapOpResult (deno:core/core.js:106:13)
at Object.opSync (deno:core/core.js:120:12)
at opFetch (deno:extensions/fetch/26_fetch.js:43:17)
at mainFetch (deno:extensions/fetch/26_fetch.js:170:61)
at deno:extensions/fetch/26_fetch.js:395:7
at new Promise (<anonymous>)
at fetch (deno:extensions/fetch/26_fetch.js:357:15)
at file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/ejemplosDenoTS/do-fetch.ts:1:24
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\ejemplosDenoTS> deno run --allow-net .\do-fetch.ts
{ userId: 1, id: 1, title: "delectus aut autem", completed: false }
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\ejemplosDenoTS>
```

Live Share

Ln 3, Col 19 Spaces: 4 UFT-8 CRLF TypeScript Go Live 4.3.2 Deno 1.11.5 Prettier

CODER HOUSE

DENON

CODER HOUSE



¿De qué se trata?

- **Deno** es el reemplazo Deno para Nodemon que proporciona una experiencia llena de funciones, altamente configurable y fácil de usar.
- No requiere ningún cambio adicional en nuestro código o método de desarrollo.
- Reinicia automáticamente el servidor de nuestros proyectos Deno.
- Es un reemplazo directo para el ejecutable deno.



¿De qué se trata?

- Tiene amplias opciones de configuración con soporte de scripts.
- Tiene observador de archivos configurable con soporte para eventos del sistema de archivos y caminata de directorio.
- Ignora archivos o directorios específicos con patrones globales.
- No se limita a proyectos deno y tiene una potente configuración de script.



Instalación y uso

- Para instalarlo simplemente ponemos en consola el comando:

```
deno install -qAf --unstable https://deno.land/x/denon/denon.ts.
```

- Vemos por el comando, que este es un paquete inestable aún.
- Para iniciar nuestros proyectos usando Denon, simplemente ejecutamos los mismos comandos que vimos antes, pero en lugar de empezarlos con “*deno*” empiezan con “*denon*”. Por ejemplo: **denon run <name.ts>**.
- También le podemos pasar las flags, como permisos y todo lo que vimos en las diapositivas anteriores, con los mismos comandos.
Ejemplo: **denon run --allow-read <name.ts>**.



Configuración

- Denon está diseñado para ser simple pero también extremadamente configurable para adaptarse a las necesidades de nuestro proyecto. Es compatible con json y yaml para el archivo de configuración. Las opciones de configuración en yaml son las mismas que json, lo que lo hace compatible.
- Para crear una configuración básica, en la raíz de nuestro proyecto ejecutamos el comando: **denon --init**. Este comando crea un archivo básico llamado scripts.json:

```
{  
    // optional but highly recommended  
    "$schema": "https://deno.land/x/denon/schema.json",  
  
    "scripts": {  
        "start": {  
            "cmd": "deno run app.ts",  
            "desc": "run my app.ts file"  
        }  
    }  
}
```



Servidor usando Denon

- Vamos al código, y creamos un servidor, en Deno, como ya hicimos antes.
- Como respuesta el servidor muestra en pantalla un texto HTML.

```
TS server.ts  X
TS server.ts
  6  /** Create Server */
  7  const server = serve({
  8    port: PORT
  9  );
 10
 11 console.log("http://localhost:" + PORT);
 12 for await (const req of server) {
 13   req.respond({
 14     status: 200,
 15     headers: new Headers({
 16       "content-type": "text/html",
 17     }),
 18     body: "<h2>Hola seguidores de Coderhouse!!!</h2>"
 19   });
 20 }
```



Servidor usando Denon



- Recordemos que para ejecutar nuestro servidor debemos anteponer el flag **--allow-net**, y si queremos tomar el puerto desde las variables de entorno, debemos usar también el flag **--allow-env**.

The screenshot shows a Visual Studio Code interface with a file named `server.ts` open. The code defines a Deno server that logs its port and responds with "Hola seguidores de Coderhouse". A terminal window below shows the command `deno run server.ts` being run, which fails due to a permission denied error because it lacks net access.

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\3\Ejemplo1> deno run server.ts
error: Uncaught PermissionDenied: Requires net access to "0.0.0.0:3000", run again with the --allow-net flag
const listener = Deno.listen(addr);
```

A Windows Firewall alert dialog is visible, prompting the user to allow Deno's JavaScript and TypeScript traffic through the firewall. A red arrow points to the "Permitir acceso" (Allow access) button.

At the bottom of the terminal, the command `--allow-net` is shown being added to the run command.

```
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\3\Ejemplo1> deno run --allow-net server.ts
```



Servidor usando Denon



```
File Edit Selection View Go Run Terminal Help
server.ts - Ejemplo1 - Visual Studio Code

EXPLORER TS server.ts x
OPEN EDITORS TS server.ts > TypeScript > Headers
EJEMPLO1 > .vscode README.md TS server.ts

11 console.log("http://localhost:" + PORT);
12 for await (const req of server) {
13   req.respond({
14     status: 200,
15     headers: new Headers([
16       ["content-type": "text/html",
17     ]),
18     body: "<h2>Hola seguidores de Coderhouse</h2>"
19   });
20 }

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\3\Ejemplo1> deno install -qAf --unstable https://deno.land/x/denon/denon.ts
Warning: Implicitly using latest version (2.4.8) for https://deno.land/x/denon/denon.ts
✓ Successfully installed denon
C:\Users\EducacionIT\.deno\bin\denon.cmd
C:\Users\EducacionIT\.deno\bin\denon (shell)
PS C:\Cursos\Coderhouse\CursoBackend\Clase47\ejemplosDenoTS\3\Ejemplo1> denon run --allow-net server.ts
[*] [main] v2.4.7
[*] [daem] watching path(s): **/*.*
[*] [daem] watching extensions: ts,tsx,js,json
[!] [#[0]] starting "deno run --allow-net server.ts"
Check file:///C:/Cursos/Coderhouse/CursoBackend/Clase47/ejemplosDenoTS/3/Ejemplo1/server.ts
http://localhost:3000
```

- Luego, si queremos ejecutar este mismo servidor pero con Denon, instalamos primero Denon como vimos en las diapositivas anteriores.
- Una vez instalado, para ejecutar el servidor usamos el mismo comando que antes pero con “denon” en lugar de “deno”: **denon run --allow-net server.ts.**

CODER HOUSE



Vemos que el servidor queda prendido en modo escucha, como con Nodemon. Si hacemos algún cambio, este se reinicia de forma automática reflejando estos cambios.



Servidor usando Denon

- Finalmente, si vamos al navegador, en el puerto correspondiente, vemos la respuesta que configuramos en el servidor (el texto en HTML).



DEPENDENCIA SERVEST



Dependencia Servest

- Es similar a http, y se usa para servidores.
- Tenemos una pequeña comparación entre dos servidores, uno con http y el otro con servest. Son bastante similares, aunque tienen algunas diferencias.

std_http.ts

```
import { serve } from "https://deno.land/std/http/mod.ts";

const server = serve({ port: 8888 });
for await (const req of server) {
    await req.respond({
        status: 200,
        headers: new Headers({
            "content-type": "text/plain",
        }),
        body: "hello deno!",
    });
}
```

servest.ts

```
import { createApp } from "https://deno.land/x/servest@v1.3.1/mod.ts";
const app = createApp();
app.handle("/", async (req) => {
    await req.respond({
        status: 200,
        headers: new Headers({
            "content-type": "text/plain",
        }),
        body: "hello deno!",
    });
});
app.listen({ port: 8888 });
```



Servidor Servest con React y Denon



```
serverReact.tsx > ...
1 // @deno-types="https://deno.land/x/servest@v1.3.1/types/react/index.d.ts"
2 import React from "https://dev.jspm.io/react/index.js";
3 // @deno-types="https://deno.land/x/servest@v1.3.1/types/react-dom/server/index.d.ts"
4 import ReactDOMServer from "https://dev.jspm.io/react-dom/server.js";
5 import { createApp } from "https://deno.land/x/servest@v1.3.1/mod.ts";
6
7 const app = createApp();
8 let visitas:number = 0
9
10 app.handle("/", async (req) => {
11   await req.respond({
12     status: 200,
13     headers: new Headers({
14       "content-type": "text/html; charset=UTF-8",
15     }),
16     body: ReactDOMServer.renderToString(
17       <html>
18         <head>
19           <meta charset="utf-8" />
20           <title>servest</title>
21         </head>
22         <body>
23           <h1 style={{color:'blue'}}>Hello Servest con React!</h1>
24           <h2 style={{color:'brown'}}>Visitas: {++visitas}</h2>
25           <h3 style={{color:'purple'}}>FyH: {new Date().toLocaleString()}</h3>
26         </body>
27       </html>
28     ),
29   );
30 });
31 app.listen({ port: 8899 });


```

- Tenemos ahora otro servidor en el cual usamos Deno con React.
- La dependencia de Deno que usamos ahora es **servest** en lugar de *http* para crear el servidor.
- Vemos que en el body de la respuesta de la ruta “/” que configuramos en el servidor, tenemos un componente de React que se renderiza para mostrarse en el navegador.
- Al tener React adentro, la extensión del archivo del servidor es *tsx* (TypeScript extendido).



Servidor Servest con React y Denon



serverReact.tsx >

```
1 // @deno-types="https://deno.land/x/servest@v1.3.1/types/react/index.d.ts"
2 import React from "https://dev.jspm.io/react/index.js";
3 // @deno-types="https://deno.land/x/servest@v1.3.1/types/react-dom/server/index.d.ts"
4 import ReactDOMServer from "https://dev.jspm.io/react-dom/server.js";
5 import { createApp } from "https://deno.land/x/servest@v1.3.1/mod.ts";
6
7 const app = createApp();
8 let visitas: number = 0
9
10 app.handle("/", async (req) => {
11   await req.respond({
12     status: 200,
13     headers: new Headers({
14       "content-type": "text/html; charset=UTF-8",
15     }),
16     body: ReactDOMServer.renderToString(
17       <html>
18         <head>
19           <meta charset="utf-8" />
20           <title>servest</title>
21         </head>
22         <body>
23           <h1 style={{color:'blue'}}>Hello Servest con React!</h1>
24           <h2 style={{color:'brown'}}>Visitas: {++visitas}</h2>
25           <h3 style={{color:'purple'}}>FyH: {new Date().toLocaleString()}</h3>
26         </body>
27       </html>
28     ),
29   );
30 });
31 app.listen({ port: 8899 });
```

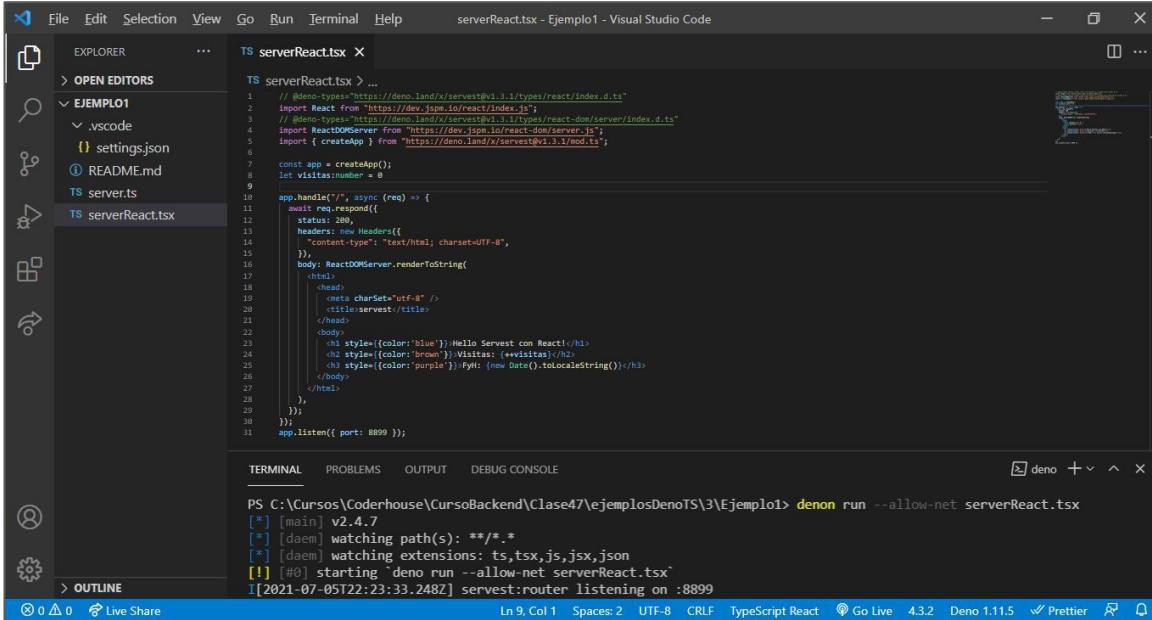
- El código de las líneas 1 y 3, que parecen comentarios, no lo son.
- Es una notación de Deno para indicar que este Typescript de la línea 1 por ejemplo, es el utilizado en la dependencia Javascript de la línea 2. Y lo mismo con las líneas 3 y 4.
- La doble barra del principio es la notación que necesita @deno-types cuando su uso impacta en la siguiente declaración de importación de módulo.



Servidor con React y Denon



- Ejecutamos entonces el servidor con Denon, para que quede en modo escucha. Lo hacemos como siempre, agregando los permisos de red:
--allow-net.



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder named "EJEMPLO1" containing ".vscode", "settings.json", "README.md", "server.ts", and "serverReact.tsx".
- Editor:** The "serverReact.tsx" file is open, displaying the following code:

```
// @deno-types="https://deno.land/x/servert@v1.3.1/types/react/index.d.ts"
import React from "https://dev.jspm.io/react/index.js";
import ReactDOMServer from "https://dev.jspm.io/react-dom/server/index.d.ts";
import { createApp } from "https://deno.land/x/servert@v1.3.1/mod.ts";

const app = createApp();
let visitas:number = 0;

app.handle("/", async (req) => {
    await res.respond({
        status: 200,
        headers: new Headers([
            { "content-type": "text/html; charset=UTF-8" },
        ]),
        body: ReactDOMServer.renderToString(
            <html>
                <head>
                    <meta charset="utf-8" />
                    <title>servert</title>
                </head>
                <body>
                    <h1 style={{color:'blue'}}>Hello Servet con React!</h1>
                    <h2 style={{color:'brown'}}>Visitas: <+visitas></h2>
                    <h3 style={{color:'purple'}}>Pyth: [new Date().toLocaleString()]</h3>
                </body>
            </html>
        ),
    ));
});

app.listen({ port: 8899 });

```

- Terminal:** Shows the command being run: `denon run --allow-net serverReact.tsx`. The output shows Denon starting and listening on port 8899.

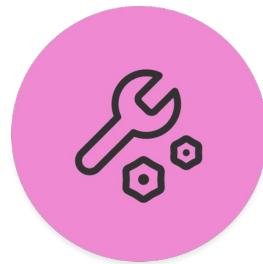


Servidor con React y Denon



- Finalmente, si vamos al navegador, vemos el componente de React, renderizado como HTML para poder ser mostrado en el navegador.
- Es simplemente un contador de visitas, y se muestra también la fecha actual.





SERVIDOR DENO CON HTTP

Tiempo: 10 minutos



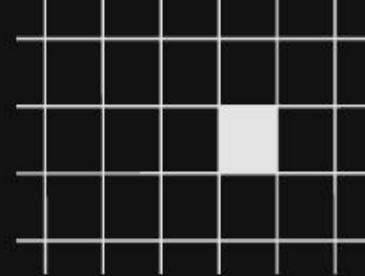
SERVIDOR DENO CON HTTP

Tiempo: 10 minutos

1. Modificar el servidor del desafío anterior (conservando la misma funcionalidad) para que utilice el módulo http servest generando la vista con React render.
 - El servidor deberá tener extensión tsx para el correcto funcionamiento de la sintaxis de vista de React en Typescript.
2. Utilizar denon para que, ante un cambio de código, el servidor de reinicie automáticamente.

*¿*PREGUNTAS?

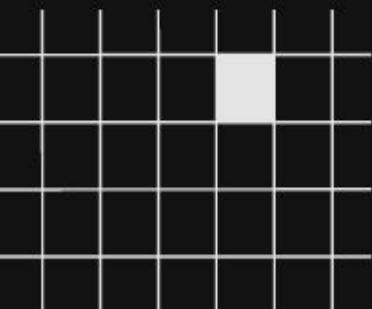




¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Entorno de ejecución Deno.
- Dependencia Denon.
- Ejemplos simples usando Deno y Denon.





OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN