

Unidad 05. PROGRAMACIÓN BACKEND

Motores de Plantillas y Routers en Express

Objetivos de la clase

- Conocer la implementación Router en Express
- Implementar un sistema de recursos estáticos
- Comprender el uso de Middlewares
- Manejar Multer para carga de archivos.

Objetivos de la clase

- Comprender qué es un motor de plantillas.
- Diferencia entre un motor de plantillas y un framework de frontend
- Utilización de motor de plantillas Handlebars en Express.



Hands on lab

En esta instancia de la clase **se mostrará la implementación de express Router** con un ejercicio práctico

¿De qué manera?

El profesor demostrará cómo hacerlo y tú lo puedes ir replicando en tu computadora. Si surgen dudas las puedes compartir para resolverlas en conjunto de la mano de los tutores.

Tiempo estimado: **15 minutos**

Express Router

¿Cómo lo hacemos? **Se crearán dos routers: users y pets.**

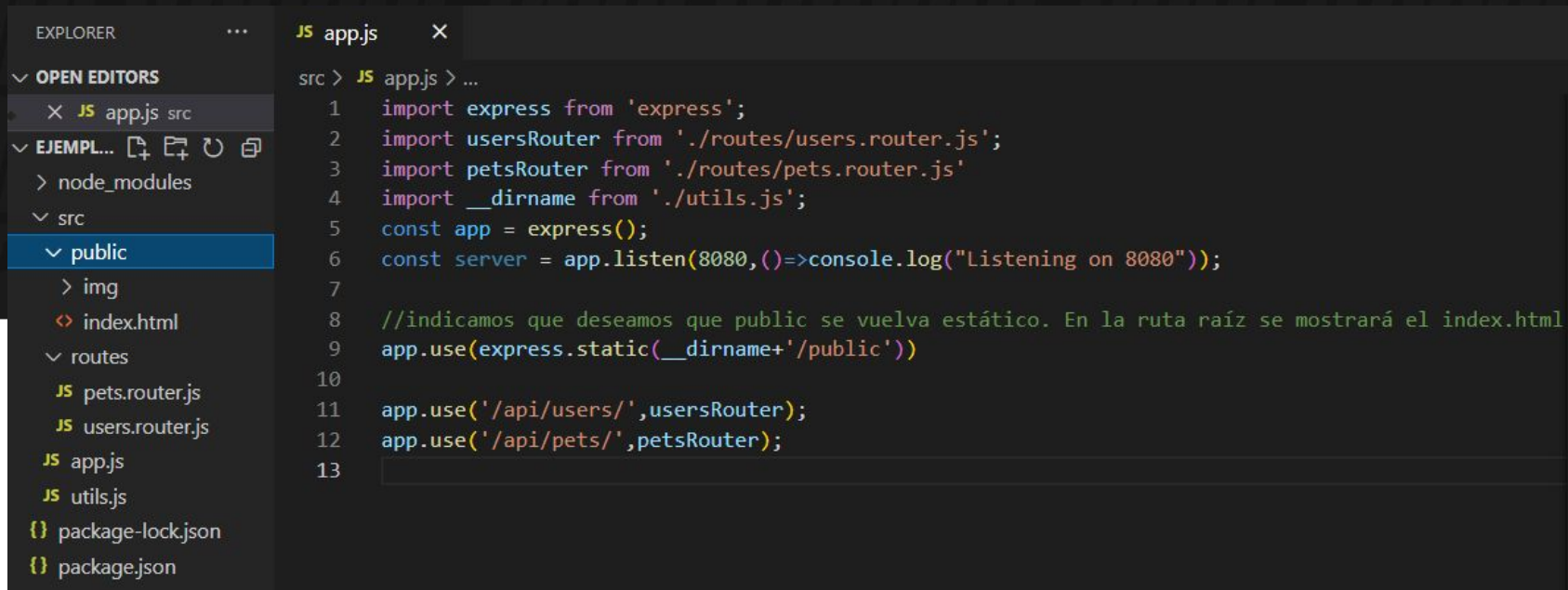
- ✓ El router de users debe tener la ruta principal /api/users
- ✓ El router de pets debe tener la ruta principal /api/pets
- ✓ Ambos deben tener, de manera interna, un array para almacenarlos.
- ✓ Ambos deben contar con un método get en su ruta raíz para poder obtener el arreglo.
- ✓ Ambos deben contar con un método POST en su ruta raíz para poder agregar un usuario o mascota según sea el router.
- ✓ Conectar los routers al archivo app.js para tener listo el apuntador al router.
- ✓ Probar funcionalidad con Postman.



Ejemplo archivos estáticos

- ✓ Se agregará una carpeta pública, la cual contendrá dentro una carpeta "img", donde guardaremos una imagen, desde el navegador se deberá acceder a este archivo a partir del servicio de archivos estáticos
- ✓ Crear una página index.html que contenga un mensaje de bienvenida y acceder desde la ruta raíz a éste.

Ejemplo sobre archivos estáticos en express



```
EXPLORER  ...

  OPEN EDITORS
    X JS app.js src
  EJEMPL...  [icon] [icon] [icon] [icon]
    > node_modules
    > src
    > public
    > img
    <> index.html
    > routes
      JS pets.router.js
      JS users.router.js
      JS app.js
      JS utils.js
    {} package-lock.json
    {} package.json

JS app.js  X
src > JS app.js > ...
1  import express from 'express';
2  import usersRouter from './routes/users.router.js';
3  import petsRouter from './routes/pets.router.js';
4  import __dirname from './utils.js';
5  const app = express();
6  const server = app.listen(8080, ()=>console.log("Listening on 8080"));
7
8  //indicamos que deseamos que public se vuelva estático. En la ruta raíz se mostrará el index.html
9  app.use(express.static(__dirname+'/public'))
10
11  app.use('/api/users/',usersRouter);
12  app.use('/api/pets/',petsRouter);
13
```



Carpeta public

Trabajaremos más a fondo el index que tenemos actualmente

Duración: 10/15 min



ACTIVIDAD EN CLASE

Carpeta public

Partiendo del ejemplo anterior, recrear la estructura con un index.html para poder visualizarse en la ruta raíz.

- ✓ En este archivo deberá haber un formulario donde podremos ingresar una mascota a partir del método POST. Dicho POST conectará al endpoint raíz del router pets
- ✓ Configurar el router pets para que pueda recibir el json por parte del formulario (recordar `express.json()` y `express.urlencoded({extended:true})`)
- ✓ Verificar con POSTMAN que la información llegue al servidor y se guarde correctamente.



Ejemplo en vivo

Instalación y configuración de MULTER

Dejamos las siguientes diapositivas con el paso a paso para la consulta de forma asincrónica.

1. Instalar multer

MULTER es una dependencia de terceros, de manera que, al igual que express, necesitaremos instalarlo dentro de nuestro package.json con el comando:

```
\src> npm install multer
```

Debemos corroborar que se encuentre dentro de las dependencias al mismo nivel que tenemos express instalado.

```
test : echo \ Error: no  
,  
"keywords": [],  
"author": "",  
"license": "ISC",  
"dependencies": {  
  "express": "^4.18.1",  
  "multer": "^1.4.5-lts.1"  
}
```

2. Configurar multer en el proyecto actual

Una vez que tenemos MULTER instalado, podemos importarlo en nuestro proyecto y configurarlo donde lo necesitemos (puede ser directamente en app, o bien se recomienda hacerlo en un archivo al mismo nivel de app llamado "utils")

Contar con un uploader externo a app.js, brindará más dinamismo al momento de utilizarlo, ya que podemos colocarlo en el router que necesitemos y no necesariamente instanciarlo a nivel general

Ejemplo: configurando multer para poder utilizar un “uploader”

```
JS app.js JS utils.js ×
src > JS utils.js > ...
13 import multer from 'multer';|
14
15
16 //Antes de instanciar multer, debemos configurar dónde se almacenarán los archivos.
17 const storage = multer.diskStorage({
18   //destination hará referencia a la carpeta donde se va a guardar el archivo.
19   destination: function(req,file,cb){
20     cb(null,__dirname+'/public/img')//Especificamos la carpeta en este punto.
21   },
22   //filename hará referencia al nombre final que contendrá el archivo
23   filename: function(req,file,cb){
24     cb(null,file.originalname)//originalname indica que se conserve el nombre inicial
25   }
26 })
27
28 export const uploader = multer({storage});
```

3. Utilizar uploader

Una vez que nuestro uploader está listo para utilizarse, podemos importarlo en el router que necesitemos y colocarlo en la ruta donde lo necesitemos, recuerda que, al ser un middleware, éste va en medio de la ruta y de la función callback (req,res).

```
router.post('/',uploader.single('file'),(req,res)=>{
```

Podemos utilizar el uploader de dos formas principalmente:

- ✓ `uploader.single('nombre del campo')`: permitirá subir un único archivo, su resultado estará en **req.file**
- ✓ `uploader.array('nombre de campos')`: permitirá subir múltiples archivos, su resultado estará en **req.files**

Ejemplo implementación MULTER a partir de su uploader

JS users.router.js X

src > routes > JS users.router.js > default

```
1  import {Router} from 'express';
2  import { uploader } from '../utils.js'; //importamos el uploader previamente configurado
3  const router = Router();
4
5  let users = [];
6  router.get('/', (req, res) => {
7    res.send({status: "success", payload: users})
8  })
9
10 router.post('/', uploader.single('file'), (req, res) => {
11   if(!req.file) { //Si no existe req.file, significa que hubo un error al subir el archivo
12     //queda en tus manos decidir si puede continuar con el proceso o no.
13     return res.status(400).send({status: "error", error: "No se pudo guardar la imagen"})
14   }
15   console.log(req.file); //Revisar los campos que vienen en req.file por parte de multer
16   //El resto del cuerpo del usuario a agregar vivirá en req.body, como es usual.
17   let user = req.body;
18   user.profile = req.file.path; //agregamos al usuario la ruta de su respectiva imagen.
19   users.push(user);
20   res.send({status: "success", message: "User created"})
21 })
22 export default router;
```



Express + multer

Configuraremos el formulario previo para poder cargar una imagen

Duración: **10/15 min**



ACTIVIDAD EN CLASE

Express + multer

Basado en el formulario para ingresar una mascota al sistema:

- ✓ Configurar el formulario para añadir un campo `input type="file" name="file"` para que la mascota a agregar pueda tener una "imagen representativa".
- ✓ El nombre del archivo guardado se formará con el nombre original anteponiéndole un timestamp (`Date.now()`) seguido con un guión. Ej: `1610894554093-clase1.zip`.
- ✓ Corroborar que la imagen se guarde correctamente. Guardar la ruta del archivo guardado en un campo "thumbnail".



Break

¡10 minutos y volvemos!

Ejemplo de una plantilla de carrito de compra

index.html X

src > public > index.html > html > body > br

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10  <h1>¡Hola, {{nombre}}!</h1>
11  <p>Estuvimos procesando tu carrito de compra y el monto final será de {{precio}}</p>
12  <br>
13  <p>El pedido será entregado a {{dirección}}</p>
14  <a href="http://linkfalso.com">Cambiar dirección</a>
15  <p>Además el precio de envío será {{costo_envio}} ya que su membresia es {{tipo_membresia}}</p>
16 </body>
17 </html>
```

Instalación y configuración de Handlebars en Express

APROXIMACIÓN AL PROCESO

1. Tener listo tu proyecto express

Antes de pensar en Handlebars tenemos que comenzar a plantear nuestro proyecto de Express. Debemos pensar en la estructura de nuestras carpetas y en dónde colocaremos las plantillas.

¿Será mi aplicación completamente hecha de plantillas, o generaré una plantilla para casos especiales? En este curso trabajaremos sobre un proyecto pensado completamente en plantillas.

La estructura inicial deberá ser como lo indica la imagen

```
> node_modules
└─ src
   ├── public
   ├── routes
   ├── JS app.js
   ├── JS utils.js
   ├── {} package-lock.json
   └── {} package.json
```



APROXIMACIÓN AL PROCESO

2. Agregamos una carpeta Views

En esta carpeta agregaremos todas las plantillas que queramos utilizar, podemos entender entonces que aquí colocaremos las páginas a utilizar en el proyecto.

Sin embargo, necesitamos un marco inicial para colocar las plantillas, para ello utilizaremos un “layout”. Crearemos una carpeta layouts dentro de views, y dentro colocaremos un “main.handlebars”, haciendo referencia a que ese es el marco principal.

Luego, fuera de layouts, pero dentro de views, agregaremos un index (página inicial)

▼ EJEMPLO HANDLEBARS

> node_modules

▼ src

> public

> routes

▼ views

▼ layouts

🔥 main.handlebars

🔥 index.handlebars

JS app.js

JS utils.js

{ } package-lock.json

{ } package.json



APROXIMACIÓN AL PROCESO

3. Configurar main.handlebars

main.handlebars X

src > views > layouts > main.handlebars > ...

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <script src="//cdn.jsdelivr.net/npm/sweetalert2@11"></script>
8   <title>{{{title}}}</title>
9 </head>
10 <body>
11   {{{body}}}
12 </body>
13 </html>
```

Nota que es la estructura de un html cualquiera, sin embargo, esta vez dentro de body colocamos con `{{{}}}` el nombre body (única vez que encerraremos en tres llaves)

Ahora, cada vista de views que vayamos a renderizar, se renderizará dentro de este body. De esta manera, no tenemos que escribir una estructura html completa por cada vista que queramos trabajar.



APROXIMACIÓN AL PROCESO

4. Saludar desde index.handlebars

Nota que, como la vista index estará dentro del main layout, ya no es necesario crear una página para poder mostrar la vista.

Generaremos un saludo, indicando que al entrar a dicha ruta se mostrará el nombre. Nota cómo dejamos **name** entre las dos llaves, haciendo referencia a que es una variable reemplazable al momento de hacer el preprocesamiento.

Podemos colocar cuantas variables necesitemos al momento de renderizar.

index.handlebars X

src > views > index.handlebars > div > h1

```
1 <div>
2   <h1>¡Hola, {{name}}! ¡Qué gusto verte!</h1>
3 </div>
```




APROXIMACIÓN AL PROCESO

5. Instalar el motor de handlebars

Tenemos la estructura, las plantillas, ¡todo muy bonito! Sin embargo, no hemos configurado nada en el servidor para que realmente podamos leer y trabajar con las plantillas.

Para ello, levantaremos nuestro servidor y configuraremos el motor de plantillas interno, esto lo conseguimos instalando con npm:

```
> npm install express express-handlebars
```



APROXIMACIÓN AL PROCESO

6. Configurando handlebars en nuestro motor express

```
JS app.js x
src > JS app.js > ...
 1 import express from 'express';
 2 import handlebars from 'express-handlebars';
 3 import __dirname from './utils.js';
 4 const app = express();
 5 //Inicializamos el motor indicando con app.engine('qué motor utilizaremos',el motor instanciado)
 6 app.engine('handlebars', handlebars.engine());
 7 /**
 8  * luego, con app.set('views',ruta) Indicamos en qué parte del proyecto estarán las vistas
 9  * Recuerda utilizar rutas absolutas para evitar asuntos de ruteo relativo.
10  */
11 app.set('views',__dirname+'/views');
12 /**
13  * Finalmente, con app.set('view engine','handlebars') indicamos que, el motor que ya
14  * inicializamos arriba, es el que queremos utilizar. Es importante para saber que,
15  * cuando digamos al servidor que renderice, sepa que tiene que hacerlo con el motor de
16  * handlebars.
17  */
18 app.set('view engine','handlebars');
19
20 //Seteamos de manera estática nuestra carpeta public
21 app.use(express.static(__dirname+'/public'))
22 |
23 const server = app.listen(8080,()=>console.log("Listening on PORT 8080"));
24
```



APROXIMACIÓN AL PROCESO

6.5. Archivo utils.js para poder exportar el `__dirname`

(sólo válido si trabajamos con `type: module`)

```
JS app.js JS utils.js ×
src > JS utils.js > [e] default
1  import {fileURLToPath} from 'url';
2  import { dirname } from 'path';
3  const __filename = fileURLToPath(import.meta.url);
4  const __dirname = dirname(__filename);
5
6  export default __dirname;
```



APROXIMACIÓN AL PROCESO

Creando método GET que renderiza la pantalla

```
EXPLORER  ...  JS app.js  X
├── OPEN EDITORS
│   └── JS app.js src
├── EJEMPLO HANDLEBARS
│   ├── node_modules
│   ├── src
│   │   ├── public
│   │   ├── routes
│   │   └── views
│   │       ├── layouts
│   │       └── index.handlebars
│   ├── JS app.js
│   ├── JS utils.js
│   ├── package-lock.json
│   └── package.json
└── TIMELINE

src > JS app.js > app.get('/') callback > testUser
9      * Recuerda utilizar rutas absolutas para evitar asuntos de ruteo relativo.
10     */
11     app.set('views', __dirname + '/views');
12     /**
13     * Finalmente, con app.set('view engine', 'handlebars') indicamos que, el motor que ya
14     * inicializamos arriba, es el que queremos utilizar. Es importante para saber que,
15     * cuando digamos al servidor que renderice, sepa que tiene que hacerlo con el motor de
16     * handlebars.
17     */
18     app.set('view engine', 'handlebars');
19
20     //Seteamos de manera estática nuestra carpeta public
21     app.use(express.static(__dirname + '/public'))
22
23     app.get('/', (req, res) => {
24         //Usaremos un objeto de prueba, que es el que meteremos a la plantilla para sustituir campos
25         let testUser = {
26             name: "Hilda",
27             last_name: "Martinez"
28         }
29         //res.render es nuestro nuevo método para renderizar plantillas, y se compone de:
30         //(nombre de la plantilla, objeto para reemplazar campos)
31         res.render('index', testUser);
32     })
33     const server = app.listen(8080, () => console.log("Listening on PORT 8080"));
34
```



APROXIMACIÓN AL PROCESO

```
views > index.handlebars > div > h1  
<div>  
  <h1>¡Hola, {{name}}! ¡Qué gusto verte!</h1>  
</div>
```

+

```
let testUser = {  
  name: "Hilda",  
  last_name: "Martinez"  
}  
  
//res.render es nuestro nuevo  
//(nombre de la plantilla,objeto de datos)  
res.render('index',testUser);
```

=

¡Hola, Hilda! ¡Qué gusto verte!

¡Voilà! ¡Ahora la plantilla reemplazó el nombre para dar la bienvenida a un usuario específico!



Datos personales

Es tu turno de poner a prueba el dinamismo de handlebars

Duración: 10/15 min



ACTIVIDAD EN CLASE

Datos personales

Basándonos en el ejemplo anterior, desarrollar una vista web que permita mostrar los datos personales de múltiples usuarios.

- ✓ Utilizar la misma estructura mostrada por el profesor, para poder levantar un servidor que utilice handlebars como motor de plantillas.
- ✓ Configurar la plantilla para que muestre los siguientes datos: nombre, apellido, edad, correo, teléfono.
- ✓ Crear un array "users" que cuente con 5 usuarios de tipo objeto, cada uno con los datos mencionados arriba.
- ✓ Al llamar al método get '/', generar un número random para elegir a alguno de los usuarios y mostrar el usuario seleccionado al azar en la plantilla.
- ✓ Observar los diferentes resultados en el navegador.

Estructuras en Handlebars

No solo se trata de reemplazar

Si bien las plantillas permiten “reemplazar” una marca por un dato dinámico, Handlebars nos proporciona también la capacidad para poder realizar estructuras de decisión o de repetición.

Cuando utilicemos una estructura, lo haremos con el símbolo # dentro de las dos llaves, como:

{{#if condición}}

No admite expresiones, sólo booleanos.

Asimismo, podemos realizar ciclos también utilizando la palabra reservada para Handlebars “each”.

Como se mencionó, necesitaremos colocar el hashtag para poder ejecutar un ciclo:

{{#each elemento iterable}}



Ejemplo de estructuras con Handlebars

- ✓ Agregar un arreglo con nombre "food", el cual contendrá 5 objetos con los datos: *name*, *price*
- ✓ Con base en la estructura ya desarrollada. Agregar al usuario de prueba un campo "role" el cual podrá ser "admin" o "user"
- ✓ Modificar la plantilla para que, si el usuario es de rol admin, pueda ver la lista de alimentos, si es usuario, sólo verá la bienvenida

Ejemplo: configurando método para hacerlo más complejo

```
let food = [
  {name:"Hamburguesa", price:"100"},
  {name:"Banana",price:"20"},
  {name:"Soda",price:"40"},
  {name:"Ensalada",price:"120"},
  {name:"Pizza",price:"150"}
]

app.get('/',(req,res)=>{
  //Usaremos un objeto de prueba, que es el que meteremos a la plantilla para sustituir campos
  let testUser = {
    name:"Hilda",
    last_name:"Martinez",
    role:"user"
  }

  //res.render es nuestro nuevo método para renderizar plantillas, y se compone de:
  //(nombre de la plantilla,objeto para reemplazar campos)
  res.render('index',{
    user:testUser,
    isAdmin:testUser.role==="admin",
    food
  });
})
```

Ejemplo: configurando index para poder utilizar if/else y ciclo each

index.handlebars X

src > views > index.handlebars > div

```
1 <div>
2   <h1>¡Hola, {{user.name}}! ¡Qué gusto verte!</h1>
3   {{#if isAdmin}}
4     {{#each food}}
5       <div>
6         <p>{{this.name}}</p>
7         <p>{{this.price}}</p>
8       </div>
9     {{/each}}
10  {{else}}
11    <p>No tienes permisos para ver la lista de alimentos</p>
12  {{/if}}
13
14 </div>
```

Ejemplo: Hilda rol = user

¡Hola, Hilda! ¡Qué gusto verte!

No tienes permisos para ver la lista de alimentos

Ejemplo: Hilda rol = admin

¡Hola, Hilda! ¡Qué gusto verte!

Hamburguesa

100

Banana

20

Soda

40

Ensalada

120

Pizza

150



Handlebars con express

Aplicando lo visto en una nueva plantilla

Duración: 10/15 min



ACTIVIDAD EN CLASE

Handlebars con express

Realizar un formulario en una nueva plantilla.

- ✓ Se creará un archivo “register.handlebars” como nueva plantilla, donde se colocará un form
- ✓ Dicho form debe servir para registrar un usuario, por lo que contará con nombre, correo, y contraseña
- ✓ Enviar los datos a una ruta POST ‘/user’, y guardar el usuario en un arreglo.
Confirmar que el guardado se realice exitosamente.

Muchas gracias.

#DemocratizandoLaEducación