

# Fundamentos de la Web

## Bloque III: Tecnologías de servidor web

### Tema 9.2: Aplicaciones web con Node.js

# Aplicaciones web con Node

- Las librería estándar de Node permiten crear aplicaciones web
- Pero son muy limitadas y se suele usar el paquete NPM **express**

express

<https://expressjs.com>

# Aplicaciones web con Express

## Mínima aplicación web

ejem1

app.js

```
import express from 'express';  
  
const app = express();  
  
app.get('/', (req, res) => {  
  res.send('Hello World!');  
});  
  
app.listen(3000, () => console.log('Listening on port 3000!'));
```

package.json

```
{  
  "name": "web-ejem1",  
  "version": "1.0.0",  
  "type": "module",  
  "dependencies": {  
    "express": "^4.17.1"  
  }  
}
```

# Aplicaciones web con Express

## Mínima aplicación web

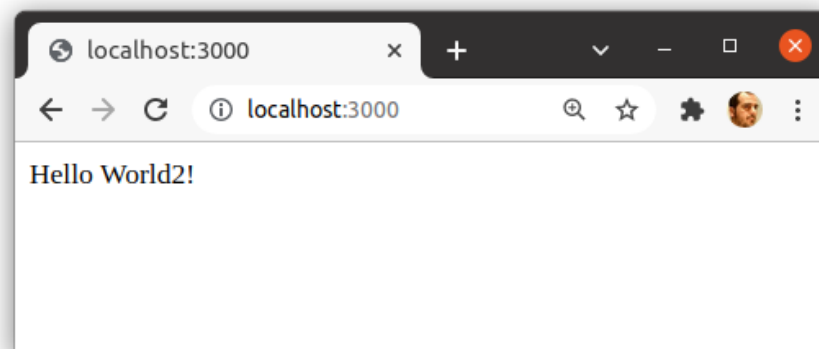
ejem1

- Instalar las librerías

```
$ npm install
```

- Ejecutar la aplicación

```
$ node app.js
```



<http://localhost:3000>

# Aplicaciones web con Express

## Mínima aplicación web

ejem1

- Por cada cambio de código es necesario parar y ejecutar de nuevo la aplicación
- La herramienta **nodemon** detecta automáticamente los cambios y reinicia el servicio

```
$ sudo npm install -g nodemon
```

```
$ nodemon app.js
```

# Templates y ficheros estáticos

- Express es una librería **muy configurable**
- Es necesario configurar de forma **explícita**:
  - Servir ficheros estáticos por http
  - Procesado del body en las peticiones
  - Tecnología de plantillas

# Templates y ficheros estáticos

- **Mustache en Node**

- Existen muchas otras tecnologías de plantillas
- Se puede usar Mustache

```
$ npm install --save mustache-express
```

- Existen diferencias entre Mustache para Node y para Java
  - P.e: No se puede usar `{{-index}}` en los bucles

<http://expressjs.com/en/guide/using-template-engines.html>

# Templates y ficheros estáticos

## Aplicación web básica con Mustache

ejem2

```
✓ web-ejem2
  > node_modules
  ✓ public
    🖼 mastercloudapps.png
  ✓ views
    🌀 index.mustache
  JS app.js
  JS dirname.js
  {} package-lock.json
  {} package.json
```

package.json

```
{
  "name": "web-ejem2",
  "version": "1.0.0",
  "type": "module",
  "dependencies": {
    "express": "^4.17.1",
    "mustache-express": "^1.3.0"
  }
}
```



# Templates y ficheros estáticos

ejem2

```
import express from 'express';
import mustacheExpress from 'mustache-express';
import bodyParser from 'body-parser';
import { __dirname } from './dirname.js';

const app = express();

app.set('views', __dirname + '/views');
app.set('view engine', 'mustache');
app.engine('mustache', mustacheExpress());

app.use(bodyParser.urlencoded({ extended: true }));

app.use(express.static(__dirname + '/public'));

app.get('/', (req, res) => {
  res.render('index', {
    name: "World"
  });
});

app.listen(3000, () => console.log('Listening on port 3000!'));
```

Configuración de  
**Mustache**

Configuración del  
analizador del  
body

Configuración de  
carpeta pública

# Plantillas y archivos estáticos

ejem2

- **Rutas relativas**

- Con CommonJS la variable global `__dirname` apunta a la carpeta del fichero `.js`
- En ESM no existe esa variable global, pero su valor se puede obtener usando la API estándar

`dirname.js`

```
import { fileURLToPath } from 'url';
import { dirname } from 'path';

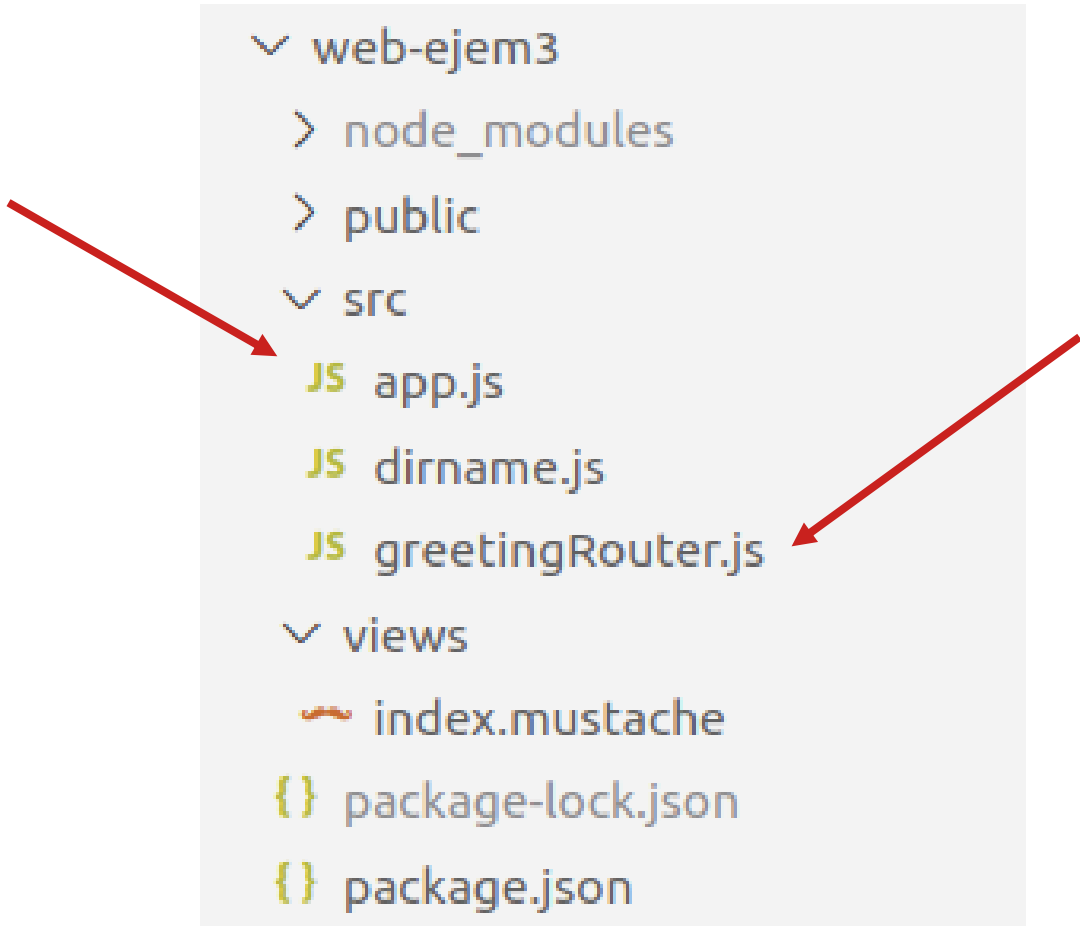
export const __dirname = dirname(fileURLToPath(import.meta.url));
```

# Separación en módulos

- Se recomienda dejar app.js para configurar el servidor
- La gestión de peticiones se define en otro módulo en un **Router**
- No se usa inyección de dependencias, se usan referencias entre módulos con **import**
- Los módulos se mueven a la carpeta **/src**

# Separación en módulos

ejem3



```

✓ web-ejem3
  > node_modules
  > public
  ✓ src
    JS app.js
    JS dirname.js
    JS greetingRouter.js
  ✓ views
    📄 index.mustache
  {} package-lock.json
  {} package.json
```

# Separación en módulos

ejem3

src/app.js

```
import express from 'express';
import mustacheExpress from 'mustache-express';
import bodyParser from 'body-parser';
import { __dirname } from './dirname.js';
import greetingRouter from './greetingRouter.js';

const app = express();

app.set('views', __dirname + '/../views');
app.set('view engine', 'mustache');
app.engine('mustache', mustacheExpress());

app.use(bodyParser.urlencoded({ extended: true }));

app.use(express.static(__dirname + '/../public'));

app.use('/', greetingRouter);

app.listen(3000, () => console.log('Listening on port 3000!'));
```

src/greetingRouter.js

```
import express from 'express';

const router = express.Router();

router.get('/', (req, res) => {

  res.render('index', {
    name: "World"
  });

});

export default router;
```

# Proceso de formularios y enlaces

- Acceso a los datos en el servidor
  - Los valores se almacenan en **req.query** (formulario GET) o **req.body** (formulario POST)

public/index.html

```
<form action="greeting">
  <p>Saludar a :</p>
  <input type='text' name='userName' />
  <input type='submit' value='Enviar' />
</form>
```

```
router.get('/greeting', (req, res) => {
  res.render('greeting', {
    name: req.query.userName
  });
});
```

Parámetro con el valor  
del campo de texto del  
formulario

# Proceso de formularios y enlaces

- Procesar diferentes métodos http

```
app.METHOD(PATH, HANDLER)
```

- Donde:
  - **app** es una instancia de express. También puede ser un router
  - **METHOD** es el método de solicitud HTTP
  - **PATH** es la ruta de acceso en el servidor
  - **HANDLER** es la función que se ejecuta cuando se accede a la ruta

# Ejercicio 1 – Tablón de mensajes

- Crear una aplicación web para gestionar un tablón de anuncios con varias páginas
- La página principal muestra los anuncios existentes (sólo nombre y asunto) y un enlace para insertar un nuevo anuncio
- Si pulsamos en la cabecera de un anuncio se navegará a una página nueva que muestre el contenido completo del anuncio



# Ejercicio 1 – Tablón de mensajes

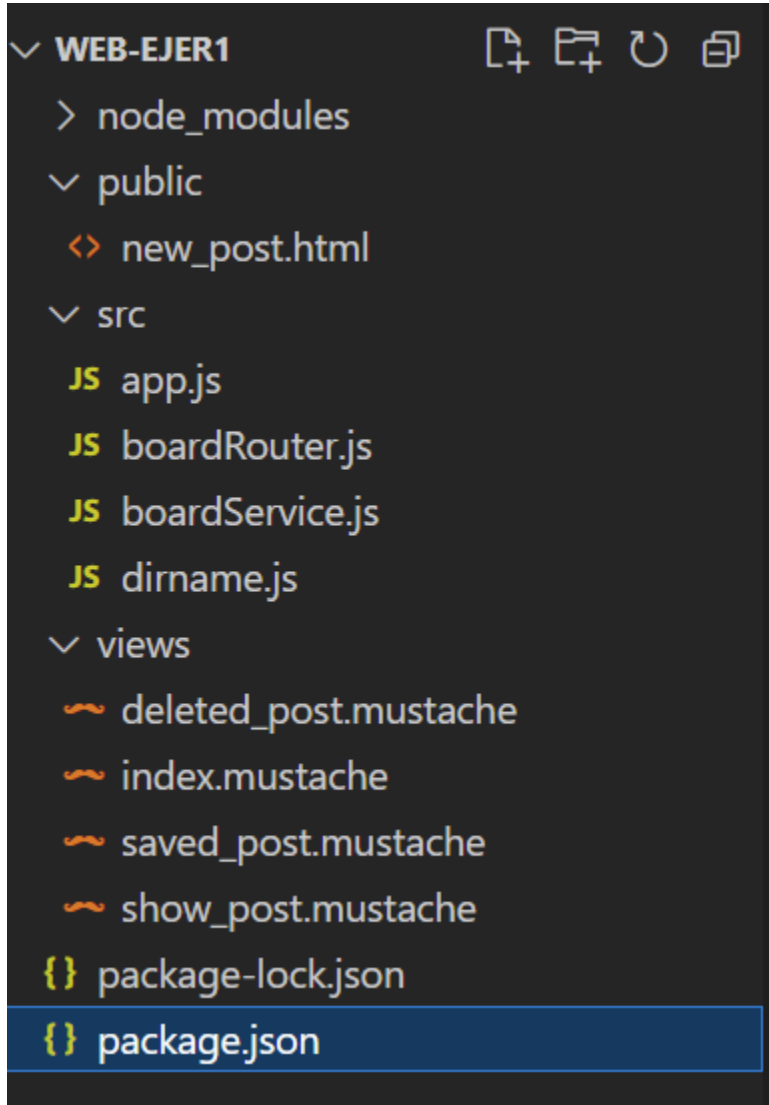
- Si se pulsa el enlace para añadir el anuncio se navegará a una nueva página que contenga un formulario
- Al enviar el formulario se guardará el nuevo anuncio y se mostrará una página indicando que se ha insertado correctamente y un enlace para volver
- En la página del anuncio se podrá borrar

# Ejercicio 1 – Tablón de mensajes

- **Implementación**

- Se usará un único router con varias funciones (cada una atendiendo una URL diferente)
- Se creará un módulo independiente para la gestión de los posts
- Cada post tendrá asociado un id generado con un contador

# Ejercicio 1 – Tablón de mensajes



## Estructura del proyecto:

- public: Carpeta con páginas accesibles para todo el mundo
- src: Código fuente del proyecto, con la funcionalidad en sí
- views: Carpeta con las páginas devueltas por los fuentes. Las plantillas con mustache siempre deberán estar en esta carpeta.

# Ejercicio 1 – Tablón de mensajes



The image shows a code editor with a `package.json` file. The file content is as follows:

```

{} package.json > ...
1  {
2    "name": "web-ejem1",
3    "version": "1.0.0",
4    "type": "module",
5    "dependencies": {
6      "express": "^4.17.1",
7      "mustache-express": "^1.3.0"
8    }
9  }

```

Two green arrows point from text boxes to the code:

- An arrow points from the box "Tipo de proyecto" to the line `"type": "module",`.
- An arrow points from the box "Dependencias" to the line `"mustache-express": "^1.3.0"`.

Sobre un proyecto ya creado es suficiente con:

```
$ npm install
```

```
$ node /src/app.js
```

# Ejercicio 1 – Tablón de mensajes

public/new\_post.html

```
<html>
<head>
  <meta charset="UTF-8"/>
</head>
<body>
  <form action="post/new" method="post">
    <p>User: </p>
    <input type='text' name='user' />
    <p>Title:</p>
    <input type='text' name='title' />
    <p>Text:</p>
    <textarea name='text' rows=5 cols=40></textarea>
    <input type='submit' value='Save' />
  </form>
</body>
</html>
```

URL donde se recogen los datos

POST, dado que se envía información

# Ejercicio 1 – Tablón de mensajes

src/app.js

```
import express from 'express';
import mustacheExpress from 'mustache-express';
import bodyParser from 'body-parser';
import { __dirname } from './dirname.js';
import boardRouter from './boardRouter.js';
```

```
const app = express();
```

Configuración de mustache

```
app.set('views', __dirname + '/../views');
app.set('view engine', 'mustache');
app.engine('mustache', mustacheExpress());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static(__dirname + '/../public'));
```

```
app.use('/', boardRouter);
```

Asociar URL – Router

```
app.listen(3000, () => console.log('Listening on port 3000!'));
```

# Ejercicio 1 – Tablón de mensajes

src/boardRouter.js

```
import express from 'express';
import * as boardService from './boardService.js';
const router = express.Router();

router.get('/', (req, res) => {
  res.render('index', {
    posts: boardService.getPosts()
  });
});

router.post('/post/new', (req, res) => {
  let { user, title, text } = req.body;
  boardService.addPost({ user, title, text });
  res.render('saved_post');
});
```

URL – Página de inicio

Página a renderizar

URL asociada en app.js

Página a renderizar

# Ejercicio 1 – Tablón de mensajes

src/boardRouter.js

```
router.get('/post/:id', (req, res) => {
  let post = boardService.getPost(req.params.id);
  res.render('show_post', { post });
});
```

Método para obtener un elemento al completo

```
router.get('/post/:id/delete', (req, res) => {
  boardService.deletePost(req.params.id);
  res.render('deleted_post');
});
```

Método para borrar un elemento

```
export default router;
```

Hay que tener en cuenta  
como pasa :id como  
parametro



# Ejercicio 1 – Tablón de mensajes

src/boardService.js

```
const posts = new Map();
let nextId = 0;

addPost({
  user: "Pepe", title: "Vendo moto", text: "Barata, barata" });
addPost({
  user: "Juan", title: "Compro coche", text: "Pago bien" });
}
```

- Se utiliza un mapa en lugar de un lista dado que un id debe identificar de manera univocal un element
- El incremento del id, lo realizamos nosotros
- Poblamos el mapa que actúa como BBDD (aunque no hay persistencia de datos)

# Ejercicio 1 – Tablón de mensajes

src/boardService.js

```
export function addPost(post) {
  let id = nextId++;
  post.id = id.toString();
  posts.set(post.id, post);
}

export function deletePost(id){
  posts.delete(id);
}

export function getPosts(){
  return [...posts.values()];
}

export function getPost(id){
  return posts.get(id);
}
```

- Se incrementa en 1 el id para el siguiente elemento, se guarda en el mapa.
- Se elimina un elemento del mapa
- Se obtiene todos los elementos del mapa.
- Se obtiene un elemento concreto del mapa (id)

# Ejercicio 1 – Tablón de mensajes

src/dirname.js

```
import { fileURLToPath } from 'url';
import { dirname } from 'path';

export const __dirname =
  dirname(fileURLToPath(import.meta.url));
```

views/deleted\_post.mustache

```
<html>
<body>
  <p>Post has been deleted.</p>
  <a href="/">Back</a>
</body>
</html>
```

views/saved\_post.mustache

```
<html>
<body>
  <p>Post has been deleted.</p>
  <a href="/">Back</a>
</body>
</html>
```


# Ejercicio 1 – Tablón de mensajes

view/show\_post.mustache

```
<html>
<body>
  <p>User: {{post.user}}</p>
  <p>Title: {{post.title}}</p>
  <p>Text: {{post.text}}</p>
  <a href="/post/{{post.id}}/delete">Delete post</a>

  <br>

  <a href="/">Back</a>
</body>
</html>
```


 {{var}} -> indica que hay una variable

{{var.campo}} -> indica campo de una variable

- Hay que tener en cuenta que post es lo que se manda al pedir renderizar la página (posts en este caso)

# Ejercicio 1 – Tablón de mensajes

view/index.mustache

```
<html>
<body>
  <h1>Posts</h1>
  {{#posts}}
    <a href="post/{{id}}">{{user}} {{title}}</a><br>
  {{/posts}}

  {{^posts}}
    <p>No posts yet.</p>
  {{/posts}}

  <br>
  <a href="new_post.h"
</body>
</html>
```

Región en mustache.  
Nos permite explorar  
todos los elementos de  
un array

Campos de la lista de  
elementos pasados por  
parámetros

Región condicional en  
mustache. En este  
caso, si no hay  
elementos en posts, se  
muestra el mensaje