Taller Práctico: Instalación de librerías y paquetes con NPM

Julian F. Latorre
Bootcamp Desarrollo Web Full Stack

Introducción

Este taller práctico está diseñado para reforzar los conceptos aprendidos sobre la instalación de librerías y paquetes con NPM. A través de una serie de ejercicios, aprenderás a crear un proyecto, instalar dependencias, y utilizar scripts NPM para automatizar tareas comunes.

1. Configuración inicial

Prerequisito

Asegúrate de tener Node.js y NPM instalados en tu sistema. Puedes verificarlo ejecutando los siguientes comandos en tu terminal:

```
node --version
npm --version
```

1.1. Ejercicio 1: Creación de un nuevo proyecto

1. Crea un nuevo directorio para tu proyecto y navega a él:

   ```
   mkdir mi-proyecto-npm
   cd mi-proyecto-npm
   ```

2. Inicializa un nuevo proyecto npm:

   ```
   npm init -y
   ```

3. Abre el archivo `package.json` generado y familiarízate con su estructura.

2. Instalación de dependencias

2.1. Ejercicio 2: Instalación de una librería de utilidades

1. Instala la librería `lodash` como una dependencia de producción:

```
npm install lodash
```

2. Verifica que `lodash` se haya añadido a las dependencias en `package.json`.

3. Crea un archivo `index.js` en la raíz de tu proyecto con el siguiente contenido:

```
const _ = require('lodash');

const numbers = [1, 2, 3, 4, 5];
console.log('Suma:', _.sum(numbers));
console.log('Promedio:', _.mean(numbers));
```

4. Ejecuta el script con Node.js:

```
node index.js
```

2.2. Ejercicio 3: Instalación de una dependencia de desarrollo

1. Instala `jest` como una dependencia de desarrollo:

```
npm install --save-dev jest
```

2. Verifica que `jest` se haya añadido a las devDependencies en `package.json`.

3. Crea un archivo `math.js` con una función simple:

```
function sum(a, b) {
  return a + b;
}

module.exports = sum;
```

4. Crea un archivo de prueba `math.test.js`:

```
const sum = require('./math');

test('suma 1 + 2 para obtener 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

5. Modifica el script de prueba en `package.json`:

```
"scripts": {
  "test": "jest"
}
```

6. Ejecuta las pruebas:

```
npm test
```

3. Gestión de dependencias

3.1. Ejercicio 4: Actualización de dependencias

1. Verifica si hay actualizaciones disponibles para tus dependencias:

```
npm outdated
```

2. Actualiza todas las dependencias a sus últimas versiones compatibles:

```
npm update
```

3. Verifica los cambios en `package.json` y `package-lock.json`.

3.2. Ejercicio 5: Desinstalación de una dependencia

1. Desinstala la librería `lodash`:

```
npm uninstall lodash
```

2. Verifica que `lodash` se haya removido de las dependencias en `package.json`.

3. Modifica `index.js` para que funcione sin `lodash`:

```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((acc, curr) => acc + curr, 0);
const average = sum / numbers.length;
console.log('Suma:', sum);
console.log('Promedio:', average);
```

4. Ejecuta el script nuevamente para verificar que funciona correctamente:

```
node index.js
```

4. Scripts personalizados

4.1. Ejercicio 6: Creación de scripts personalizados

1. Modifica la sección `scripts` en `package.json` para incluir los siguientes scripts:

```
"scripts": {
  "test": "jest",
  "start": "node index.js",
  "lint": "echo 'Ejecutando linter...'",
  "build": "echo 'Construyendo proyecto...'"
}
```

2. Ejecuta cada uno de los scripts utilizando `npm run`:

```
npm run start
npm run lint
npm run build
```

5. Seguridad

5.1. Ejercicio 7: Auditoría de seguridad

1. Ejecuta una auditoría de seguridad en tu proyecto:

```
npm audit
```

2. Si se encuentran vulnerabilidades, intenta corregirlas automáticamente:

```
npm audit fix
```

3. Si algunas vulnerabilidades no se pueden corregir automáticamente, investiga cómo resolverlas manualmente.

6. Proyecto final

6.1. Ejercicio 8: Creación de una aplicación web simple

En este ejercicio final, crearás una aplicación web simple utilizando Express.js y algunas otras dependencias.

1. Instala Express.js y otras dependencias necesarias:

```
npm install express body-parser cors
npm install --save-dev nodemon
```

2. Crea un archivo `server.js` con el siguiente contenido:

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();
const port = 3000;

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

app.get('/', (req, res) => {
  res.json({ message: '  Bienvenido   a mi aplicación!' });
});

app.listen(port, () => {
  console.log(`Servidor corriendo en http://localhost:${
port}`);
});
```

3. Modifica la sección `scripts` en `package.json` para incluir un script de desarrollo:

```
"scripts": {
  "test": "jest",
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

4. Inicia el servidor en modo de desarrollo:

```
npm run dev
```

5. Abre un navegador y visita `http://localhost:3000` para ver el mensaje de bienvenida.

6. Experimenta añadiendo más rutas y funcionalidades a tu aplicación.

7. Conclusión

En este taller práctico, has aprendido a:

- Crear un nuevo proyecto npm
- Instalar y desinstalar dependencias
- Diferenciar entre dependencias de producción y desarrollo
- Actualizar paquetes

- Crear y utilizar scripts npm personalizados
- Realizar auditorías de seguridad
- Crear una aplicación web simple utilizando npm para gestionar dependencias

Estos conocimientos te servirán como base para desarrollar proyectos más complejos y gestionar eficientemente las dependencias de tus aplicaciones web.

Consejo final

Recuerda siempre consultar la documentación oficial de NPM (`https://docs.npmjs.com/`) y de los paquetes que utilices para mantenerte actualizado sobre las mejores prácticas y nuevas funcionalidades.