

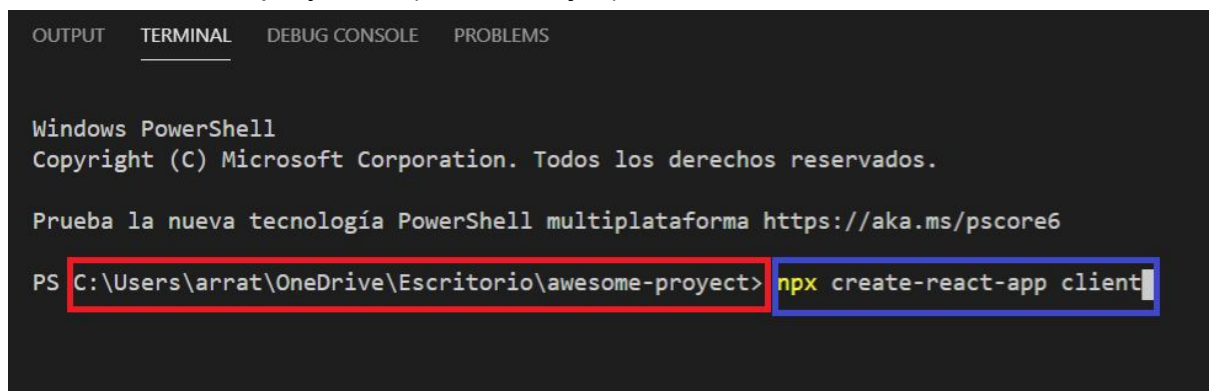
React y Express

En el módulo 2 vimos Express, que lo usamos como un gestor de rutas (o direcciones, como queráis llamarlo). Y esas llamadas a las diferentes direcciones del Express y sus Router las hacíamos desde las páginas HTML que teníamos incluídas en la carpeta public (desde los js asociados a esos HTML, más bien).

Pasemos a React ahora. React, como ya sabéis, es una forma diferente de trabajar con Javascript. Con React no renderizamos páginas de HTML, sino que creamos componentes (funcionales o de clase) que van a ir mostrándose en un único HTML, en función de qué componente queramos ver (gracias los Route y Link).

Nuestro trabajo ahora consistirá en unir Express con React, que sean capaces de comunicarse entre ellos. Cuando solo trabajamos con Express esto era fácil porque teníamos todo en el mismo proyecto (los html en la carpeta public). Pero ahora necesitamos dos puertos abiertos ya que son dos aplicaciones diferentes. Vamos a ello.

En primer lugar creamos una nueva carpeta llamada awesome-proyecto en el escritorio. Esta carpeta contendrá a su vez el proyecto api (Express) y el proyecto client (React). Una vez creada la carpeta, la abrimos con el Visual Studio Code y ejecutamos los dos comandos de las imagenes (cuadros azules) en la carpeta que contendrá ambos proyectos (cuadros rojos)



```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

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

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\arrat\OneDrive\Escritorio\awesome-proyecto> npx create-react-app client
```

Carpeta para la
instalación del proyecto
client (React)

Comando para la
creación del proyecto y
nombre que vamos a
darle a la carpeta (client)

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

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

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\arrat\OneDrive\Escritorio\awesome-proyect> npx express-generator api
```

Carpeta donde instalaremos
Express-Generator

Comando para instalar
Express generator en la
carpeta del nombre
indicado (api)

Un apunte: durante el módulo 2 nosotros montamos a mano todo el Express. Lo que hemos hecho es instalar Express-generator, que es un paquete que monta el esqueleto de una aplicación back o API. Después nosotros le daremos la forma que necesitemos en función de nuestro proyecto, pero este es el esqueleto que vamos a mantener siempre (fijaros en las carpetas y sus nombres, y en todo lo que contienen. Es así como organizábamos nosotros nuestro back de Express cuando lo hacíamos a mano).

Vale, cuando lo tengamos todo instalado, el siguiente paso es acceder a la carpeta api. En la terminal, escribimos:

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

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

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\arrat\OneDrive\Escritorio\awesome-proyect> cd api
```

Ruta que contiene
nuestros dos proyectos
(client y api)

Comando para acceder a
la carpeta que hemos
creado con Express-
generator (api)

Y una vez en esta carpeta (api), escribimos en la terminal

```
PS C:\Users\arrat\OneDrive\Escritorio\awesome-proyect\api> npm install
```

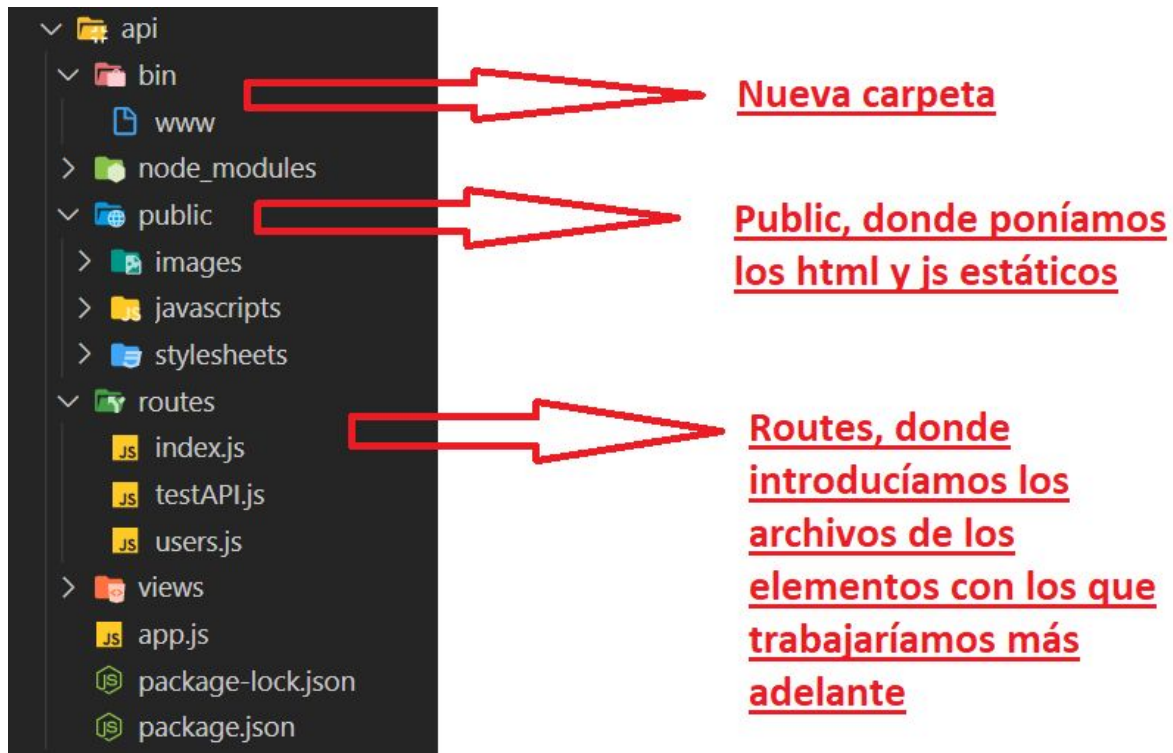
Ruta donde haremos el
npm install (fijaros que
estamos dentro de la
carpeta api)

comando para hacer
npm install

De esta forma tendremos (casi) todo instalado para que nuestra aplicación pueda funcionar. Ahora pasemos a ver las carpetas y sus archivos.

CARPETA API

Esta es la carpeta de nuestro Express. Aquí vemos una estructuración que nosotros ya habíamos realizado a mano a lo largo del módulo 2.



En la carpeta bin encontramos el archivo www. Dentro de ese archivo, en la línea 15 encontramos el puerto por defecto (3000) y lo cambiamos por el puerto 9000


```
10
11  /**
12   * Get port from environment and store in Express.
13   */
14
15  var port = normalizePort(process.env.PORT || '9000');
16  app.set('port', port);
17
18  /**
```

Esto lo hacemos porque nuestras dos aplicaciones van a estar levantadas al mismo tiempo y no pueden ocupar el mismo puerto. Por lo que el puerto 9000 se lo asignamos a la api y el cliente puede seguir ejecutándose en el puerto 3000. De esta forma, podemos arrancar ambas aplicaciones sin conflictos.

Otra de las cosas que vamos a necesitar hacer es usar CORS para que puedan comunicarse entre ellas y una no cape o prohíba las llamadas de la otra. Por lo que realizamos un

npm install cors

en el proyecto api. Una vez lo tengamos instalado, tenemos que importarlo en el archivo app.js de la carpeta api para poder usarlo y declarar su uso, de la siguiente manera:

```
api >  app.js > ...
1  var createError = require('http-errors');
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6  var cors = require('cors'); Importamos el paquete de
                                CORS
7
8  var indexRouter = require('./routes/index');
9  var usersRouter = require('./routes/users');
10 var testAPIRouter = require('./routes/testAPI');
11
12 var app = express();
13
14 // view engine setup
15 app.set('views', path.join(__dirname, 'views'));
16 app.set('view engine', 'jade');
17 app.use(cors()); Lo declaramos para su uso
```


Por último, vamos a echar un vistazo al archivo users de dentro de la carpeta Routes

```
api > routes > JS users.js > ...
1  var express = require('express');
2  var router = express.Router();
3
4  /* GET users listing. */
5  router.get('/', function(req, res, next) {
6    res.send('respond with a resource');
7  });
8
9  module.exports = router;
10 |
```

Si recordáis cómo funcionaba el Router, se dedica a redirigir llamadas en función de la url que llega (<https://localhost:3000/users/loQueSea> nos hará acceder al Router de Users y <https://localhost:3000/libros/loQueSea> al Router de Libros). A esta llamada router.get de ejemplo que nos da Express-generator es a donde vamos a realizar una llamada desde nuestro proyecto client (React). Pasemos a él.

CARPETA CLIENT

Esta carpeta es nuestra carpeta de React. Esto es lo que verá el usuario, con lo que interactuará y donde nosotros estableceremos las diferentes conexiones con nuestra api mediante llamadas fetch. Lo primero que haremos será importar en el archivo App.js dos hooks: useState y useEffect.

Tras eso, declararemos un hook de useState con valor inicial "" (comillas vacías) y nombre de variable apiResponse.

El siguiente paso será declarar un useEffect que se ejecute en cuanto se cargue el componente ([] al final del useEffect) y en el cual llamaremos a la función que contiene el fetch a la ruta de nuestra API, apuntando a Users. Tranquilos, en la foto se ve mejor

```
3 | import {useState, useEffect} from 'react';
4 |
5 | function App() {
6 |
7 |   let [apiResponse, setApiResponse] = useState("");
8 |
9 |   function callAPI(){
10 |     fetch('http://localhost:9000/users')
11 |       .then(res => res.text())
12 |       .then(res => setApiResponse(res))
13 |   }
14 |
15 |   useEffect(()=>{
16 |     callAPI();
17 |   },[])
```

Importamos los hooks

Establecemos un hook con valor empty string ("") y nombre de variable apiResponse

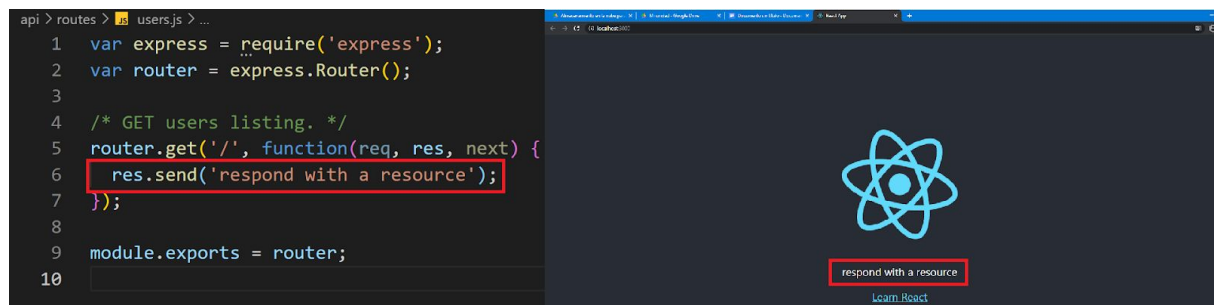
Declaramos la función que contendrá el fetch a nuestra API

UseEffect que se ejecutará en cuanto el componente App se renderize (con la función del fetch incluida dentro

En el return de la propia App escribiremos la variable a la que seteamos el valor de la respuesta de la api (línea 24 de la foto siguiente)

```
19 |   return (
20 |     <div className="App">
21 |       <header className="App-header">
22 |         <img src={logo} className="App-logo" alt="logo" />
23 |         <p>
24 |           {apiResponse}
25 |         </p>
26 |       <a
```

Por último ejecutamos ambos programas (npm start en las terminales de las carpetas api y client) y comprobamos que lo que nos devuelve el navegador (client) es la misma frase que el Router de Users tiene en el res.send (api)



Ya está, ya hemos hecho que nuestro client (React) se comunice con nuestra api (Express).