Artículo
David Reche · 1 jul, 2019 Lectura de 6 min

Crear un Backend con Node/Express y conectarlo a IRIS en menos que decimos 'Mississippi'

Hola a todos,

En este artículo voy a hacer un paso a paso desde cero para la creación de una simple API REST mediante Node/Express conectada a una instancia de InterSystems IRIS.

No voy a adentrarme en mucho detalle acerca de cómo trabajar con cualquiera de las tecnologías que menciono en el tutorial, pero dejaré enlaces a documentación adicional por si quieres aprender más. El objetivo es proporcionar una guía práctica de cómo configurar y conectar un backend node.js a InterSystems IRIS.

Antes de meterle mano a todo esto, verifiquemos que tenemos node.js corriendo en nuestra máquina. Para lo cual ejecutaremos este comando:

```
node --version
v8.12.0
```

La versión 8.12.0 (en este momento) es la actual versión LTS (Long Term support) de node.js.

En caso de tener que instalarlo utiliza el siguiente enlace https://nodejs.org.

Crear el directorio principal para el proyecto

Desde la línea de comandos, ve al directorio donde quieres crear una nueva carpeta para contener el proyecto y crea un nuevo directorio para el mismo, después cambiaremos a ese directorio.

```
mkdir amazingirisproject
cd amazingirisproject
```

Crear una aplicación Express

Bien, este paso será tan directo como el paso anterior. No olvides ejecutar los comandos desde el directorio creado anteriormente.

Dependiendo de la versión de node.js se instala el "Package Manager" (comando **npm**) y el "Package Runner" (comando **npx**). Con npm instalamos y actualizamos paquetes de forma sencilla y rápida. El comando npx nos permite ejecutar comandos sobre paquetes instalados.

Puedes probar a ejecutar este divertido comando:

```
npx cowsay hello
```

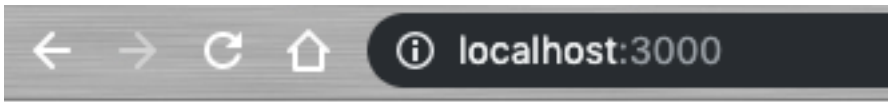
Usaré el "Express Application Generator" para crear rápidamente el esqueleto de una aplicación a la que llamaré "api". Usaremos el comando npx.

```
npx express-generator api
cd api
npm install
npm start
```

Veamos que hemos hecho con estos comandos:

1. He usado npx de npm para install express-generator globally.
2. He usado el express-generator para crear la aplicación de express que hemos llamado api.
3. Cambiamos al directorio de la API.
4. Instalamos todas las librerías de las que depende el proyecto.
5. Arrancamos la aplicación.

En tu navegador escribe la siguiente URL http://localhost:3000/. Si todo ha ido bien, Verás una página de bienvenida como esta:



¡Enhorabuena! Esto significa que tenemos una aplicación básica de Express corriendo en nuestra máquina local. ¿Fácil verdad?

Para parar la aplicación de Express, simplemente ejecuta Ctrl + c sobre la línea de comandos.

Instalar el adaptador de InterSystems IRIS

npm recupera todos los paquetes para ti excepto iris.node que debe copiarse manualmente. Así que deberemos copiar el fichero iris.node a la carpeta `/node_module/iris` (crear el directorio si no existe previamente).

El fichero iris.node se puede obtener desde WRC o también desde el directorio `/bin` de una instancia de InterSystems IRIS.

Hay que tener cuidado y chequear que el fichero del adaptador copiado sea de una versión compatible con la versión de nuestra instalación de node. En mi caso estoy usando la versión de node v8.12.0 así que necesitaré el fichero `iris800.node` (la idea es que la versión 8 de node coincide con el primer número del nombre del fichero del adaptador). Recuerda renombrar luego el fichero a `iris.node` al copiarlo a la carpeta.

Puedes encontrar más detalle sobre el adaptador de node de InterSystems IRIS y sus versiones aqui.

Configurar la conexión a la base de datos

Bien, podemos resolver esta cuestión de diferentes formas, pero hagámoslo simple. Vamos a crear un nuevo fichero db/database.js y escribiremos lo siguiente:

```
const iris = require('iris/iris');
const db = new iris.IRIS();
module.exports = db;
```

Con solo estas tres líneas estamos haciendo "público" al resto de componentes una referencia a una conexión de InterSystems IRIS.

Conectarnos con la base de datos

Ahora, ya estamos preparados para decirle a nuestro servidor Express que abra una conexión cuando se inicie. Para ello vamos a editar el fichero bin/www y en cualquier lugar podemos escribir lo siguiente:

```
const db = require('../db/database.js');
const irisConfig = {
  ipaddress: '127.0.0.1',
  tcpport: 51773,
  username: 'superuser',
  password: '*********',
  namespace: 'USER'
};
```

En estas líneas de código estamos declarando dos objetos. El primer objeto es la referencia a la Base de Datos (desde el fichero que creamos en el paso anterior). El segundo objeto contiene la información necesaria para abrir la conexión (cambia lo necesario para ajustarse a tus necesidades, puerto, usuario, password, namespace, etc.).

Después de esto, modificaremos la función onListening de esta manera:

```
function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
  ? 'pipe ' + addr
  : 'port ' + addr.port;
  debug('Listening on ' + bind);
  db.open(irisConfig, (error, result) => {
  if (error) {
  debug(result);
  } else {
  console.log('IRIS connection opened!');
  }
  });
}
```

Lo que estamos haciendo es abrir la conexión a la base de datos una vez que el servidor Express esté levantado y preparado para escuchar peticiones.

Puedes levantar el servidor para probar y comprobar si vemos el mensaje "**IRIS connection opened!**" ... en ese caso, **bien hecho !!**

Puedes ir a por un café, te lo has ganado!! 

¡Vamos a por más!

¿Qué tal el café? ¿rico? listo, entonces ya es momento para divertirnos.

Estando el servidor en ejecución ¿que pasa si vas al navegador web y pones la siguiente URL http://localhost:3000/users/?
Pues..., vas a ver un mensaje "**respond with a resource**".  ¿Qué carajo es esto? y ¿de donde está viniendo?

Vale, te lo explico.

Express usa un sistema de **Routing** para decidir cómo el endpoint (URI) de una aplicación responde a solicitudes de clientes (más información) y entonces... bueno mejor vamos a echar un ojo al código... si abres app.js puedes ver una línea como esta:

```
var usersRouter = require('./routes/users');
```

y luego otra como esta:

```
app.use('/users', usersRouter);
```

Con estas líneas estamos diciéndole a Express que cada solicitud HTTP a la localización /users debe ser derivada al código en /routes/users.js. Ok, veamos que hay dentro de este fichero y fíjate en esta función:

```
/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});
```

Ahora lo entiendes ¿no? Finalmente no era tan complicado ¿verdad? ¿Qué te parece si utilizamos esta función y devolvemos alguna dato que recuperemos desde la instancia de Intersystems IRIS que tenemos conectada?

Añade esta línea al principio del fichero:

```
const db = require('../db/database');
```

Y reescribe el método router.get:

```
router.get('/', function(req, res, next) {
  db.get({global:'test'}, function (error, result) {
  if (error) {
  res.status(500).send(result);
  } else {
  res.status(200).send('global test='+result.data);
  }
  })
});
```

Este método ahora devolverá el valor almacenado dentro del InterSystems IRIS Global ^test.

¡¡Perfecto!! ya casi lo tenemos. Pongamos algo en este global abriendo una sesión de terminal en IRIS. Podemos hacer algo como esto:

```
root@iris:/# iris session IRIS
Node: iris, Instance: IRIS
Username: superuser
Password: ****
USER>set ^test="Hello from IRIS"
```

Tendrás que **detener y arrancar de nuevo** el servidor para que se utilice el nuevo código.

Usa el navegador para ir de nuevo a http://localhost:3000/users/. ¿Qué pasa ahora?

Sí!! Lo hiciste!! Hagamos el baile de la victoria!!

¿Divertido?

Bueno, está claro que este proyecto no hace mucho, pero puede ser el comienzo de una aplicación JavaScript Full Stack.

Si quieres continuar, dame una estrella o aporta comentarios y vamos a por el título de **Master Developer of Full Stack Cosmic Applications ever!!!**

#Principiante #Node.js #InterSystems IRIS