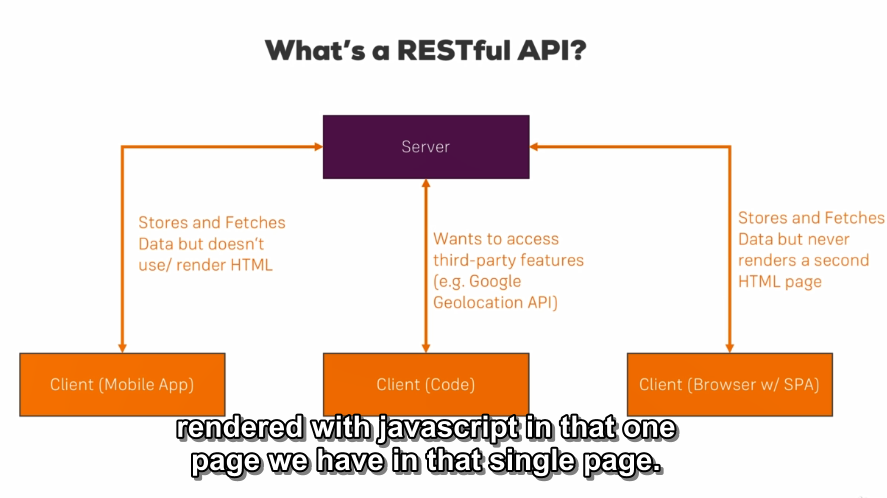
Node es un lenguaje en javascript que corre dentro del servidor.

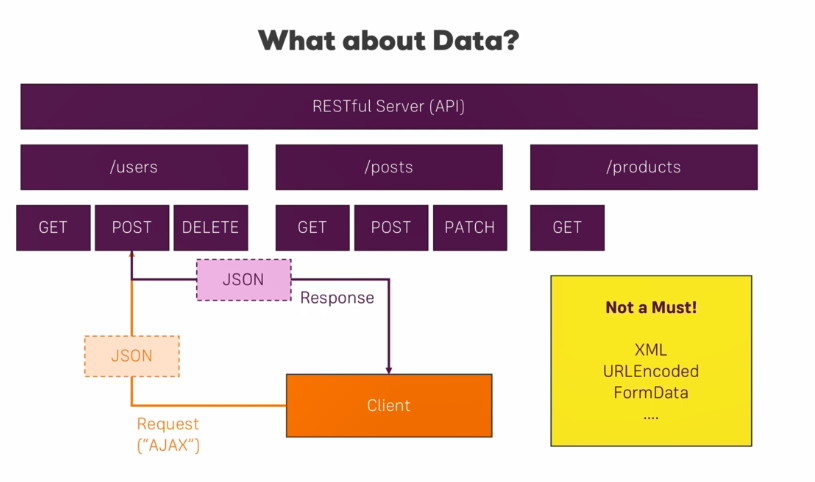
Conectando node y angular,

Siempre conectamos angular con un ng serve, pero ng serve corre node.

Existen dos “servidores” que utilizamos al manejar un proyecto con angular y node, uno que es el que retorna lo que necesita el SPA para correr, y el otro es el que tiene los servicios de autenticación, y lógica del negocio.

RESTFUL API  
Representation State Transfer  
Un api puede tener muchos clientes, una aplicación móvil, un backend, o una pagina web   


En nuestro servicio podemos definir los paths y encada uno los tipos de request que va a aceptar, ya sea solo de consulta o también para editar.



Para crear nuestro backend se puede hacer el un folder, proyecto totalmente diferente ya que es un servicio externo a nuestro código de angular, en este caso se creara en la misma carpeta.

Se crea una carpeta backend al nivel de src

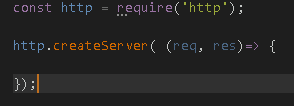
Para ejecutar el servidor se debe crear un archivo server.js, en el mismo nivel que src:

Para ejecutar un servidor se utiliza el comando node “nombre\_del\_servicio.js”





Es necesario importar paquetes de la forma de nodejs, en particular el http que permite crear un servidor, una vez declarado se accede a su función de createServer que recibe 2 parametros el request y el response.



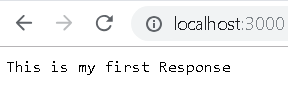
Dentro podemos acceder a las funciones de los parámetros por ejemplo end, el cual se ejecuta retornando el valor esperado.



Hay que crear una variable que aloje el servidor para luego acceder a el. Y al ejecutarlo se revisa el puerto default para ver si esta activo, en este caso se utiliza uno que podemos definir.



Si corremos el comando node server.js

  
De querer cambiar algo en el servidor hay que tumbarlo y correr nuevamente.

Añadiendo el Express Backend

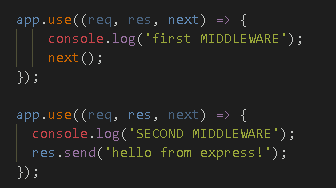


Luego dentro de la carpeta backend se agrega el archivo app.js

Dentro se importa express y asigna a una variable para acceder a sus propiedades,

Se ejecuta la función use, que recibe como parámetro: req, res, next

Podemos tener varias funciones y dependiendo de lo que contenga se ejecutaran todas o no.



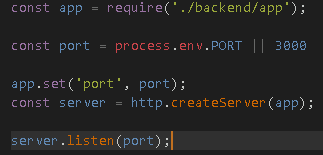
Y finalmente exportamos nuestras ejecuciones para poder ser improtadas con el servidor.

+

Dentro de nuestro server.js

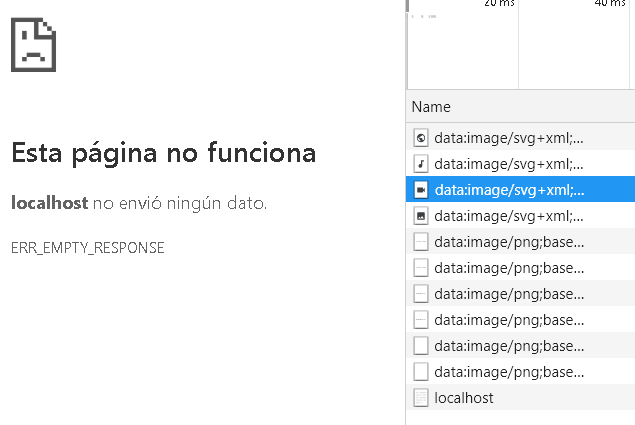


Ahora ya no ejecutamos el req y resp si no la variable que importamos al crear el servicio, también se define una constante con el puerto, para pasarla como parámetro al utilizar la función set del app, y luego der servidor.



Se ejecuta el servicio correctamente,

De no enviar una respuesta oseano poner res.send, este falla y sequeda en timing hasta que envie el timeout.



SE UTILIZÓ CODIGO PROVISTO POR EL CURSO PARA MEJORAR EL SERVER.JS, donde utiliza funciones para manejar errores y el puerto a utilizar.

Para no estar cerrando el servicio y reiniciándolo nuevamente se instala nodemon



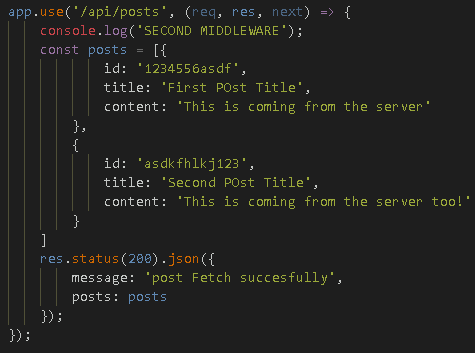
Luego de instalarla podemos correr el comando serve con nodemon, pero como no es una variable global hay que agregarla al package.json en los scripts



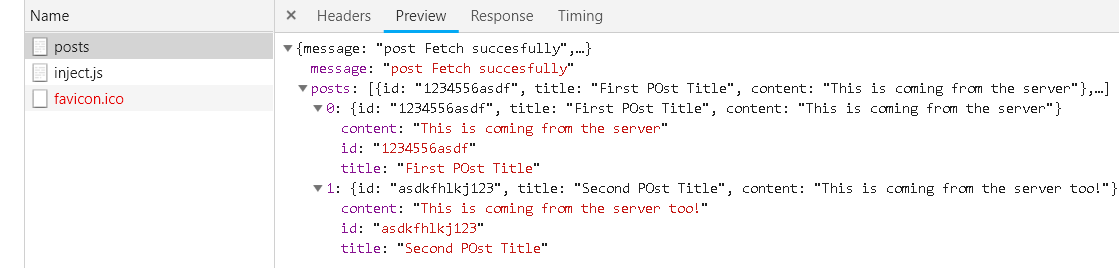


Ahora queremos traer los post desde nuestro backend para que nuestro servicio al hacer getPost traiga estos.

A nuestra función dentro de app.js, le agregamos otro parámetro, el path con el que podremos acceder a ella desde nuestro servicio de angular. Creamos un dummy object de post para el ejemplo, donde tendrá un campo ID que no esta definido dentro del modelo que manejamos en el frontend con angular, pero estos puede diferir entre back y front sin causar inconvenientes, luego se retorna nuestra respuesta, con un status definido para diferenciar los errores, luego ser parsea a json con un mensaje y finalmente nuestro objeto respuesta.

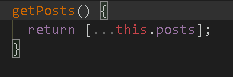


Si accedemos a este path desde el navegador:



Regresando a angular utilizando el http client

Desde el servicio nuestra función original solo regresa post desde el local.

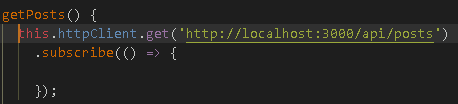


Importamos en el app.module el HttpClientModule

Y luego en el servicio lo inyectamos dentro del constructor para poder hacer llamado de sus funciones pero ahora como HttpClient

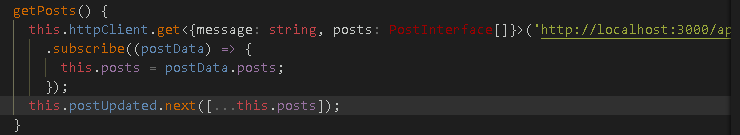


Luego hacemos el llamado desde la función y nos suscribumos a la respuesta

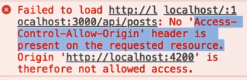


Modificamos el modelo local del post para poder especificar el tipo de respuesta del get y asignamos nuestro arreglo post local con su nuevo valor que es la respuesta que trael el servicio.

y luego asignando una copia al postUpdatepara que se escuche en toda la aplicación.

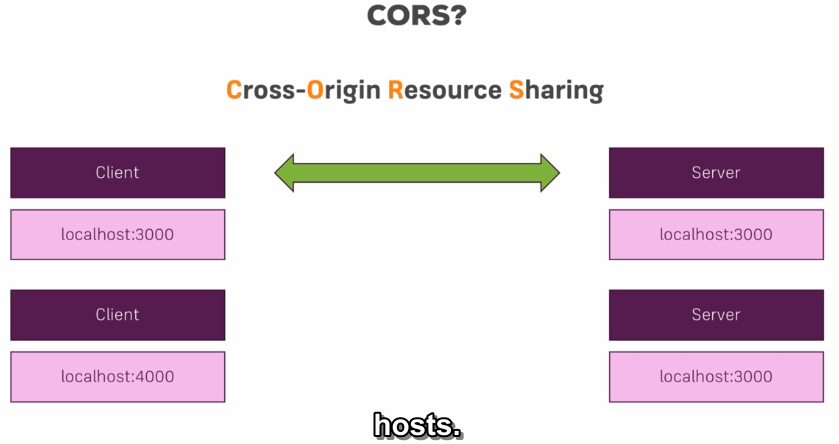


Ahora nos lanza un error



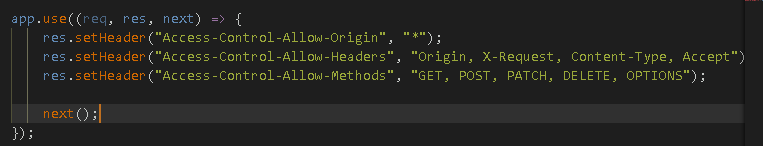


El servidor del backend esta corriendo en el puerto 3000 mientras que el de angular esta corriendo en el 4200, lo que trae un conflicto.

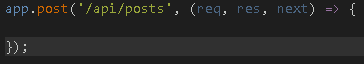


Es necesario añadir un intermediario en el backend, desde el app.Js

Se añaden los headers para dar los permisos necesarios a las aplicaciones externas de consultar, se puede manejarlas direcciones de las paginas que consultan y los métodos que pueden realizar.



Añadiendo datos al backend desde el cliente  
Se crea la función:



Luego para leer lo que nos mande desdeel forntend hay que instalar un paquete que se maneja con express:



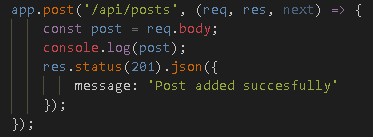
Para luego improtarlo en el app.js



Y declarar su funcionalidad antes de los llamados



Ahora desde la función que será llamada para añadir, podemos acceder a la propiedad body de res, gracias a la implementación de bodyParser. Y luego enviar una respuesta con código 200, por ejemplo, para que reciba el cliente.



Ahora nuevamente hay que conectar el cliente con el backend para acceder a esta función.



Al existir la misma ruta se diferencia en el backen por la petición de post y de get aun teniendo el mismo url.

En la próxima sección se maneja la base de datos con mongo.