# MEMORIA DE PROYECTO EJ02 ANGULAR 12 – CONEXIÓN A API

### Objetivo P1-Construir tu API sobre un framework existente

* Se adjunta el proyecto para cumplir este punto.

### Objetivo P2-Examinar la relación entre una API y un SDK

* ¿Qué es un API?

Un API es un servicio o interfaz que permite el intercambio de información entre distintos dispositivos y software, cualquier sistema puede desarrollar un método de conexión http hacía un API existente y extraer o enviar información de cualquier tipo, siempre y cuando respetando la construcción del API.

Lo magnífico de las API es que no se necesita un software específico para acceder a ellas, desde cualquier programa que integre un protocolo http puede acceder a ellas.

Toda API tiene o debe tener una documentación que nos indica los métodos de conexión que soporta y las respuestas que nos devuelve al hacer cada petición.

* ¿Qué es un SDK?

Un SDK, Software Developmen Kit, no es más que un software o conjunto de paquetes de desarrollo que permite a los programadores crear aplicaciones diversas, usando código ya existente en librerías que estas incorporan o pueden ser integradas. Este software contiene un conjunto de herramientas que facilitan el desarrollo de aplicativos, en alunas ocasiones incluso ahorrando mucho trabajo y tiempo, como es el caso de Angular o JHipster, con ellas y unos sencillos pasos, podemos generar un proyecto base en pocos minutos del que partir para crear nuestra aplicación.

* Relación entre un API y un SDK

Un API no deja de ser una aplicación que debe desarrollarse con un SDK, y con un SDK podemos crear aplicaciones que se conecten a dichas API a través de protocolos http. Ambas son dos caras de la misma moneda, por un lado, el API es quien sirve o integra un conjunto de datos, ya sean usuarios, enlaces a multimedia, o cualquier dato que podamos necesitar, y con un SDK generamos, también, los programas que interactúan con estas API para enviar o recuperar información. Gracias a las API, podemos incluso dividir programas en varias aplicaciones que usa.

### Objetivo P3-Analizar una aplicación existente que pudiera ampliarse con una API

Aplicación:

Keep (Notas) de Google, obviamente seguro que está tiene conexiones a algún tipo de API siendo de Google, además, puedes compartir las notas con otras personas. Esta app te permite crear diferentes tipos de notas, listas, añadir enlaces, imágenes, darles color y agruparlas y establecerlas como prioritarias para que siempre permanezcan en lo alto de la tabla.

Mejora mediante API de posicionamiento:

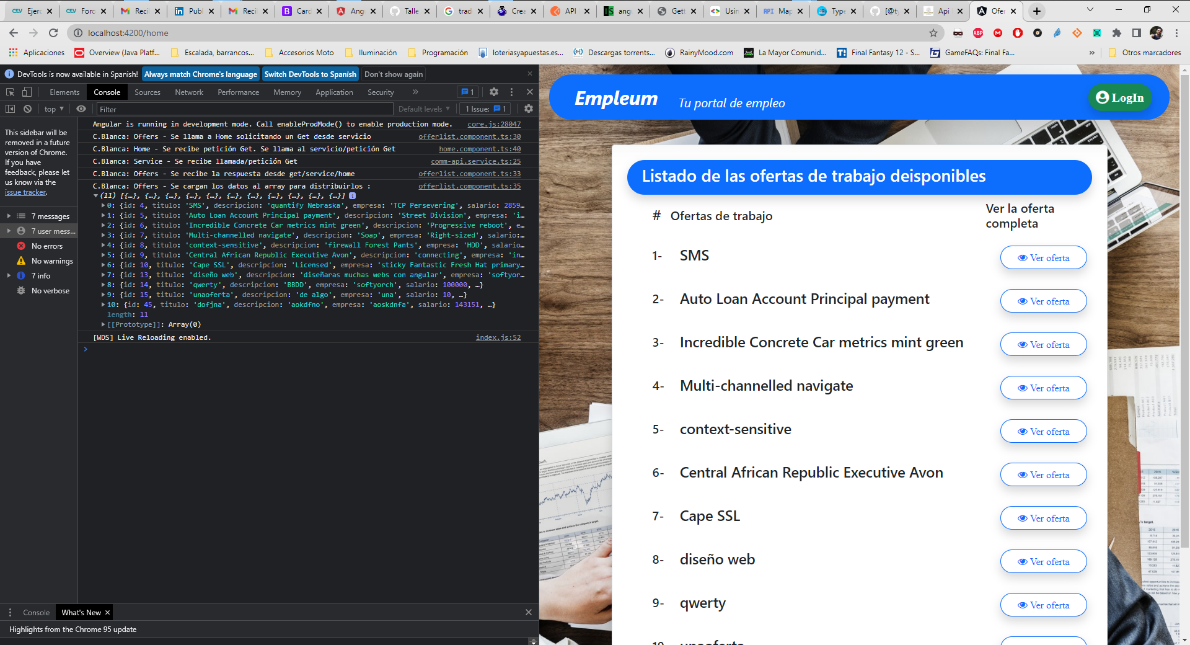
Si a esta app se le añadiese posicionamiento, se podría tener una opción que te permita ver ciertas anotaciones dependiendo de la posición actual, por ejemplo, si hacemos anotaciones en el trabajo y estas se registran como que han sido creadas en ese punto, cuando estemos en él, podríamos ver en las primeras posiciones las notas pertenecientes al trabajo, o filtrar solo dichas notas con tan solo tener el posicionamiento activado, y la autorización pertinente claro.

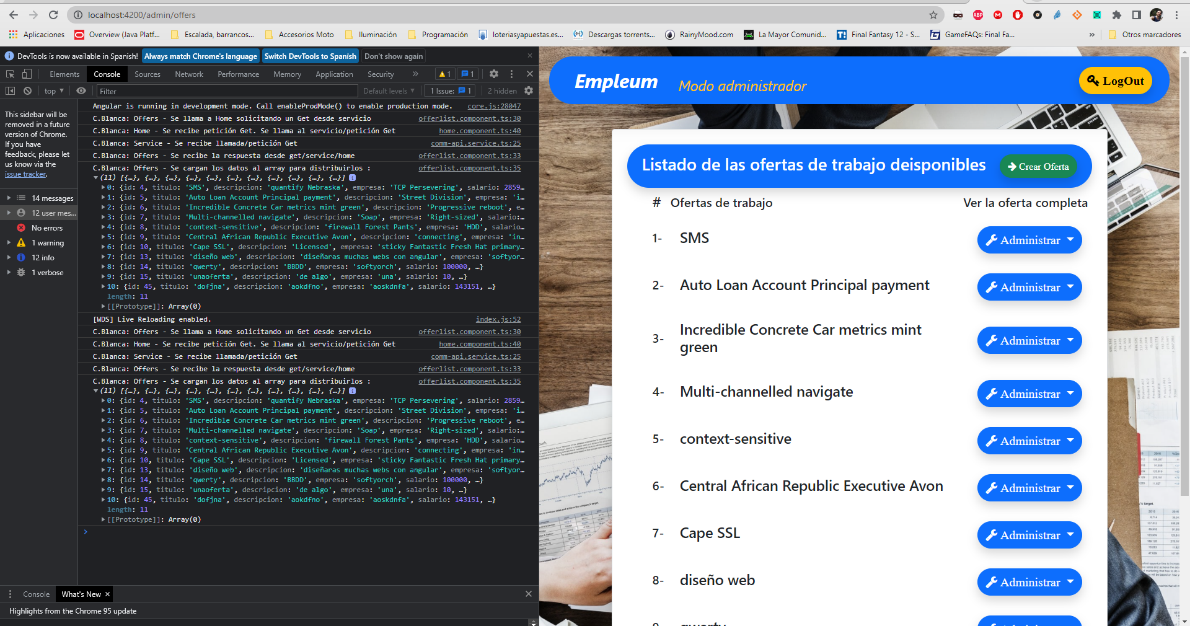
### Objetivo P4-Diseñar y realizar pruebas de "caja blanca" de la aplicación apuntando los resultados

Se realizan dos pruebas de caja blanca:

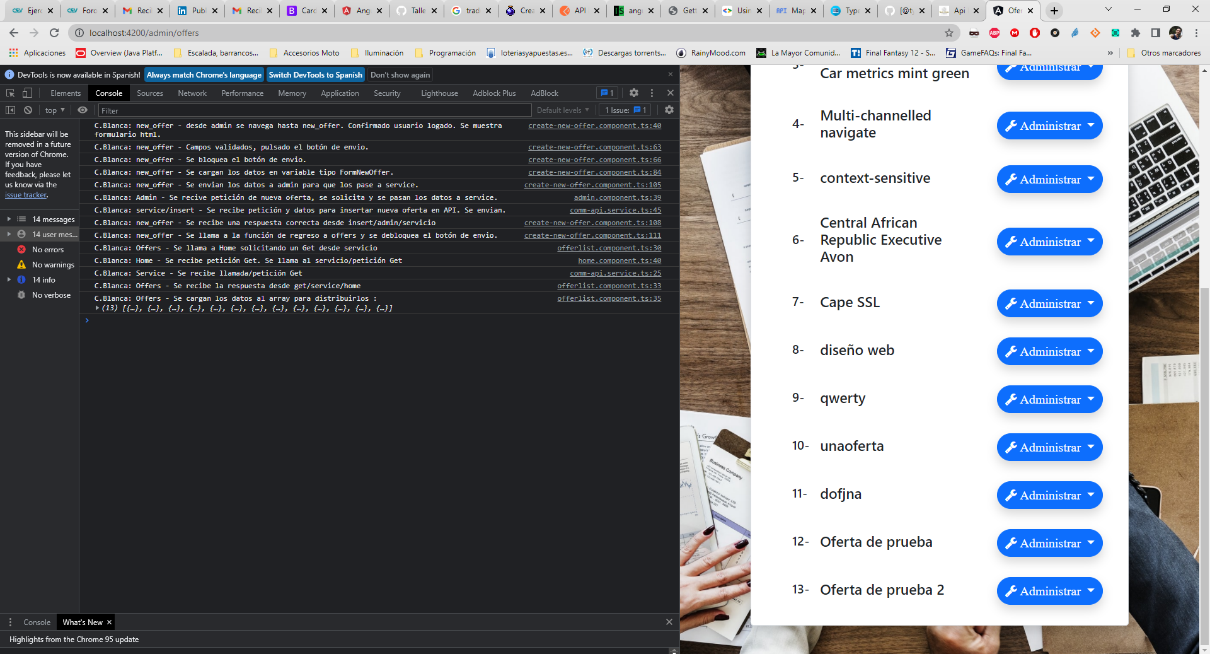
* 1. Recepción de datos desde el API con la petición GET:
* Se debe producir de forma automática al cargar la web, o acceder a ../offers:
  1. Se carga offers de forma automática al entrar a la web… OK
  2. Se produce la llamada desde offers a home para solicitar una llamada a service… OK
  3. Home llama a la petición Get de service… OK
  4. Service recibe la llamada, ejecuta la petición a la URL asignada… OK
  5. Service devuelve el resultado hacia arriba, a home y este a offers… OK
  6. Offers recibe la respuesta correctamente… OK
  7. Los datos aparecen en el documento html… OK

\*Se realiza la misma prueba accediendo desde Admin, se registran los mismos resultados, pues la llamada en ambos casos, ya sea desde home sin Super User o con SuperUser, la llamada a get se solicita a través de home.

Recepción de datos satisfactoria. *Resultado esperado y satisfactorio*.



* 1. Creación de una nueva oferta enviando los nuevos datos a la API y recibiéndolos de nuevo en la lista actualizada:
* Una vez producida la entrada de usuario a través del login de forma satisfactoria, aparece la lista de ofertas, aparece un nuevo desplegable y un botón “crear oferta” antes no visibles:
  1. Se accede desde el botón en la cabecera del listado a new\_offer… OK
  2. Se solicita a admin la navegación a new\_offer… OK
  3. Se llega a new\_offer y se muestra el formulario… OK
  4. Se rellenan los campos correctamente para que el formulario nos permita continuar… OK
  5. Se pulsa a crear y se desactiva el botón… OK
  6. Se envía la petición y los datos a admin para que este se los pase al servicio… OK
  7. El servicio recibe la solicitud y los datos en la función insert y procede al envío al API… OK
  8. Se recibe una respuesta correcta en new\_offer desde insert/service/admin… OK
  9. Se navega automáticamente a la lista de ofertas… OK
  10. Se visualiza la nueva oferta en la lista… OK

 Recepción de datos satisfactoria sobre los introducidos desde nueva offer. *Resultado esperado y satisfactorio*.

### Objetivo M1-Diseña una aplicación que use una API para un propósito específico

Aplicación Ofertas de Trabajo.

Software empleado:

Angular 12: <https://angular.io/>

Bootstrap: <https://getbootstrap.com/>

Font-awesome: <https://fontawesome.com/>

Sweetalert2: <https://sweetalert2.github.io/>

Google Maps: En proceso…

Antecedentes:

Se requiere por parte del Cliente (CEV) una aplicación que permita la conexión a un API ya construida, la cual tiene en sus servideres, para observar ofertas de empleo para sus alumnos. Dicha oferta permitirá acceder sin credenciales a cualquier persona que visite la web en la que se aloje y ver las ofertas disponibles por listado y específicamente cada una de ellas.

Además, contará con un servicio Administrador, para que la persona o personas autorizadas puedan añadir y borrar ofertas de dicha aplicación.

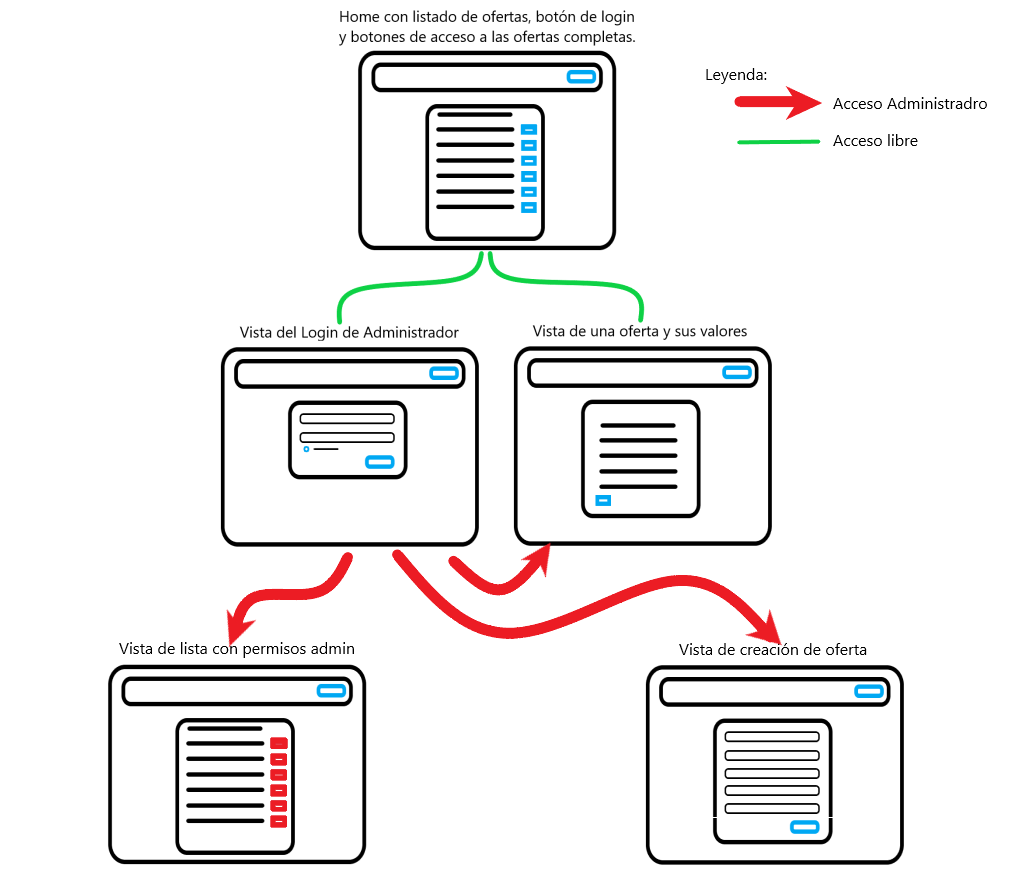
La aplicación, por tanto, deberá mostrar 5 vistas distintas:

1. Una vista Home, la cual mostrará automáticamente la lista de ofertas que contenga el servidor API con tan solo su encabezado y un botón de acceso.
2. Una vista Oferta, en ella se visualizará la oferta en la que se haya pulsado el botón ver con todos sus detalles.
3. Una vista Login, que permitirá acceder solo al usuario/s administrador.
4. Una vista con la lista inicial, pero esta vez con un botón de creación de nuevas ofertas y un botón en cada oferta de la lista para eliminarla.
5. Por último, tendrá una vista para la creación de las nuevas ofertas con su formulario correspondiente.

La aplicación podrá navegar desde Home hacia oferta o hacia login, y desde login se podrá navegar a la lista de ofertas y a nueva oferta.

API REST:

El cliente proporciona los datos de acceso a su API en la solicitud del proyecto.

Mokup del proyecto:

Modelado de la app:

Se diseña un sistema MVC (Nota, o al menos eso pretendía)

***Service***, Comm\_Api.service:

* Se genera un servicio que recoge las peticiones API del cliente y las envía al servidor, recibiendo las respuestas oportunas.

Este servicio solo conecta con dos pages, Home.component y admin.component

***Pages***:

* Home, page pública la cual tiene acceso cualquier persona que acceda a la web, esta tiene acceso al servicio, a la lista de ofertas la cual se muestra de forma predeterminada al ver esta vista, a la vista de una sola oferta, a la que solo tendrá acceso siendo solicitada desde la lista de ofertas, y a login a través de admin.
* Admin, a ella se accede desde home, permitiendo solo desde esta ruta la vista de login. Una vez pasado el login, se visualiza la vista de lista de ofertas, de forma predeterminada, desde la lista de ofertas se pude solicitar a admin que acceda a nueva oferta y visualizar oferta completa.

Ambas gestionan el flujo de información desde los componentes al servicio y viceversa.

***Component***:

* ***Offertlist***: Lista de ofertas, se visualiza la lista d ofertas con la información facilitada por Home/servicio
* ***Offer***: Oferta completa, se visualiza la oferta completa con la información facilitada por Home/servicio
* ***Logincard***: Login, da acceso al usuario/s administrador para acceder a la parte restringida de la aplicación y comunica los datos de acceso con admin/server.
* ***Offerlist***: Lista (Admin) de ofertas, es la misma vista que Lista de ofertas, pero con permisos de administrador, se visualiza la lista de ofertas con la información obtenida de Home/servicio, y al poseer permisos de administrador, le permite visualizar, crear y borrar ofertas.
* ***Create\_new\_ofer***: Nueva oferta, solicita a admin/servico la inserción de una nueva oferta con los datos que el usuario administrador facilite a través del formulario.
* ***Navbar***: Barra de navegación (casi más decorativa), da acceso al botón de login/logout y se muestra en todas las vistas.

Ninguna de ellas tiene acceso directo a servicio, solo comunican lo que necesitan en cada momento a Home o Admin, tampoco pueden navegar, para ello solicitan navegación a sus respectivas pages.

Cada componente y page dispone de su pareja html y css donde se diseñan los objetos visualmente.

### Objetivo M2-Desarrolla una aplicación que use una API

Se ha desarrollado una aplicación que permite ver, crear y borrar ofertas. Para ello existen dos niveles de acceso, uno totalmente público, el cual permite, sin ningún tipo de login, visualizar una lista de ofertas con una información mínima y observar cada una por separado con todos los datos que contengan. Por otro lado, existe el usuario administrador, el cual una vez logado tiene acceso al mismo sistema que el usuario sin permisos, pero con la particularidad de que puede eliminar las ofertas y crear nuevas. Se está trabajando en una mejora para la edición de estas. Esta aplicación se conecta a un servicio API creado con Jhipster ex profeso para ella, la cual nos permite logarnos como usuario, ver la lista de ofertas, filtrarlas por su identificador, y borrarlas.

Este API REST nos da los permisos adecuados mediante un token de seguridad, el cual devolvemos en las cabeceras de las peticiones que lo requieren.

Se adjunta carpeta la del proyecto de la aplicación.

### Objetivo M3-Realizar pruebas de "caja negra" apuntando los resultados

Pruebas de caja negra, no se tiene en cuenta nada de la programación y se tantea dos opciones de las que dispone la aplicación:

1. Login:

Se accede a la web, se pulsa en el botón de login, se pulsa login sin introducir datos…Fail

* + Ha devuelto un error 400.

Se realiza la misma prueba, pero esta vez se coloca un usuario al azar sin password…Fail

* + Vuelve a fallar, devuelve un error 400 de nuevo.
* Se insiste en esta prueba hasta obtener un resultado 401, unAtuthorize, con lo cual sabemos que ahora si accedemos al servicio, pero el usuario o la contraseña no son correctos.
* Hasta solucionar el error 400 se ha ido sumando carácter a carácter en el usuario y la contraseña hasta descubrir que necesitan un mínimo de caracteres por parte del servicio API para funcionar.

Se introduce el usuario/contraseña correctos. Esta vez se pulsa todas las veces que se puede el botón de login antes de que envíe la vista a la page de Admin…fail

* El servidor acepta o parece aceptar tantas peticiones de login como queramos.

Esta vez se envían los datos conocidos de usuario/contraseña, se pulsa el botón, se accede a la parte de administrador…Ok

Resultado esperado.

1. Borrar Oferta:

Accedemos a la web, accedemos con los datos conocidos de usuario/contraseña, pulsamos sobre borrar oferta, pero no se obtiene ninguna verificación para que el usuario pueda asegurar el borrado de la oferta, la oferta desaparece del listado…OK

* Aunque la respuesta es la esperada, se observa una deficiencia en la acción de borrar, pues al no confirmar el borrado puede provocar un error por parte del usuario al borrar accidentalmente la oferta seleccionada.

Se repite la prueba anterior hasta llegar al punto de borrado, se hace click sobre el botón todas las veces que se puede hasta que desaparece la oferta provocando un error al no encontrar la oferta en el servidor una vez borrada con la primera solicitud…fail

* El servidor acepta tantas peticiones que seamos posibles de enviar haciendo click sobre el botón de borrado.

### Objetivo M4-Actualiza la aplicación a partir de lo obtenido en los resultados

Dados los problemas descubiertos con las pruebas de caja blanca y caja negra, se implementan las siguientes soluciones.

* Problema error 400 al hacer login:

Se validan los caracteres introducidos por el usuario siendo el mínimo de 4 para ambos casos, usuario y contraseña.

Para evitar el problema del envío múltiple de login, se implementa un bloque en html junto con un icono spinner que simboliza que la aplicación está trabajando.

Este último se implementa en todos los botones que dan acceso a envíos o solicitudes de información a la API.

* Error de no verificación:

Se subsana este error añadiendo un popup que informa al usuario de la selección para eliminar el objeto y la necesidad de verificar dicha eliminación mediante una pregunta, también dispone de un botón de cancelación.

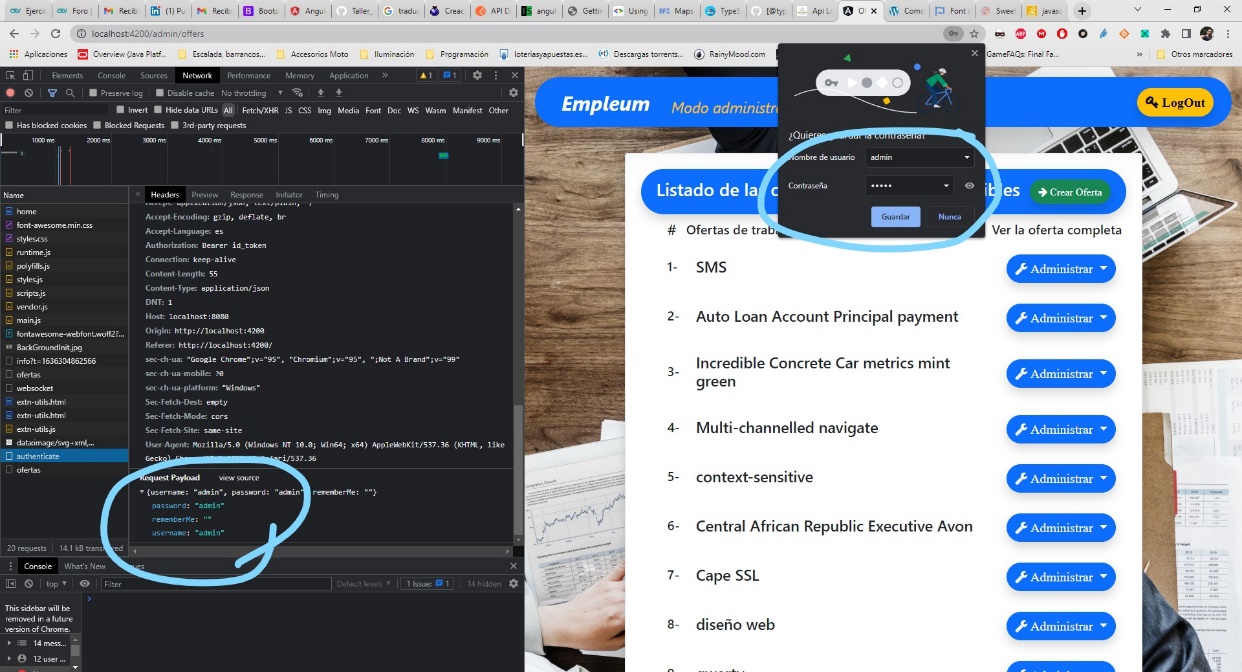
El problema de múltiples clicks ya está solventado con la acción anterior.

### Objetivo D1-Construye una aplicación utilizando múltiples APIs siguiendo el diseño de M1

### Objetivo D2-Evaluar cuestiones de seguridad API

La API con la que trabajamos no es muy segura por varios motivos, primero y como se ha demostrado con las pruebas realizadas, se pude solicitar al servidor tantas peticiones como se quiera, por lo que sería fácil para alguien que quiera realizar una acción dañina contra este, crear una aplicación que ejecute bucles infinitos con peticione al servidor hasta que este sature y caiga.

Otro de los inconvenientes a primera vista es que el servidor no es https, con lo cual, cualquiera que pueda interceptar la información será capaz de ver todo aquello que se transmita de un punto a otro.

 También se muestra una vulnerabilidad al estudiar la consola del navegador, en ella podemos encontrar tanto el token como el usuario y contraseña, que en teoría deberían ser ocultos o encriptados.

Otra buena práctica para la seguridad de las API es el uso de una puerta de enlace, permite autenticar el tráfico y controlar el uso del API.

### Objetivo D3-Crea el diseño de una aplicación que utilice varias APIS justificando tus elecciones

### Objetivo D4-Evaluar críticamente la API que has utilizado. Realiza un informe evaluando la seguridad de los datos de tu aplicación

Aunque el objetivo D2 ya evaluamos la seguridad del API, en definitiva, podemos decir que esta no es nada segura, pues posee demasiadas vulnerabilidades para un uso adecuado, no pudiendo almacenar datos sensibles y siendo victima en potencia de quien quiera dañar el servicio u obtener datos de este sin apenas complicación.

Para el uso adecuado del software a entregar, se debería poseer por parte del cliente, un API con mayo poder de seguridad.