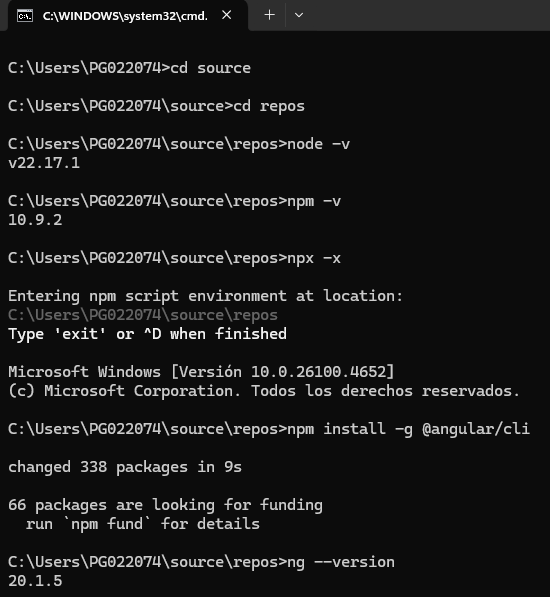
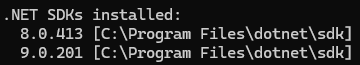
**ASPNET Core API Angular 01 y 02**

Comprobación de la instalación:





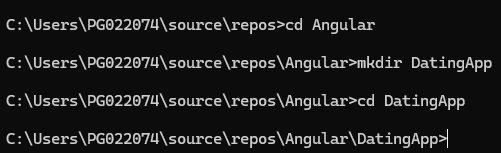


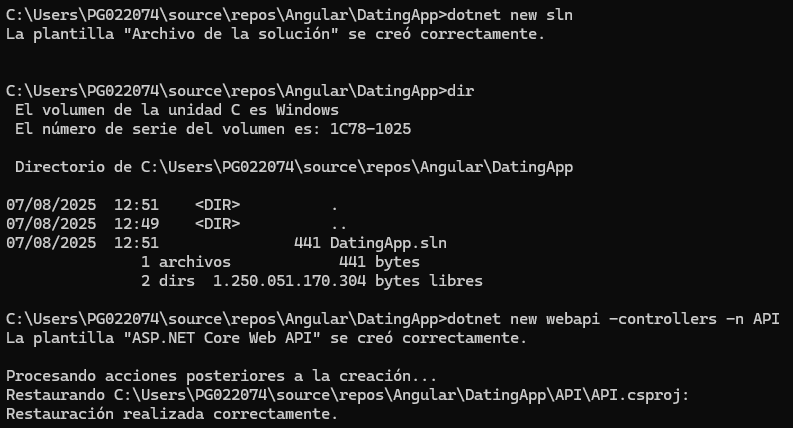


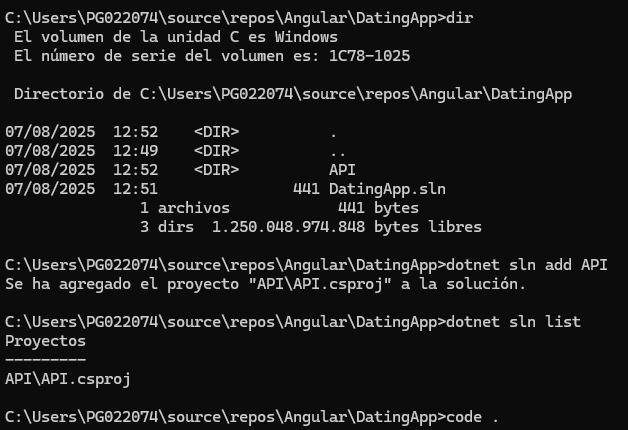




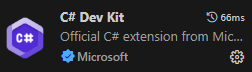
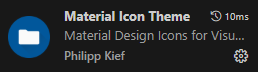
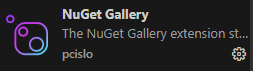
Crear el proyecto dentro de una carpeta de usuario porqué el proyecto necesita permisos para ejecutar permisos de usuario:





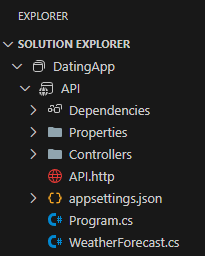
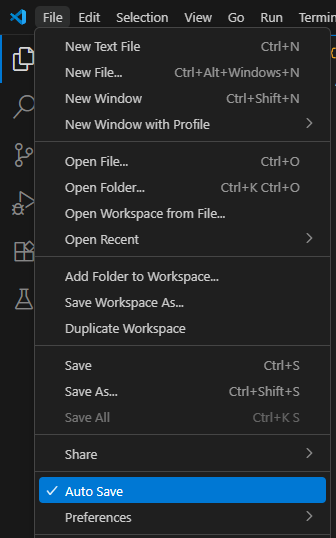


Instalar las extensiones:

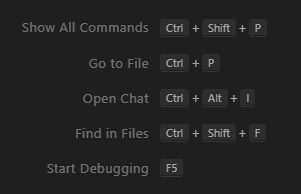
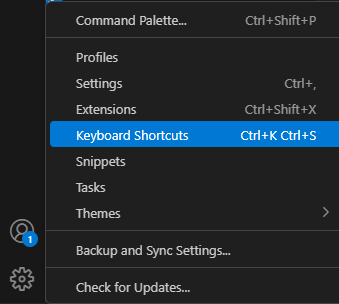
  

En VisualStudio Code, visualizar el explorador de la izquierda en:

SOLUTION EXPLORER y activamos el Auto Save

Combinaciones de teclas:

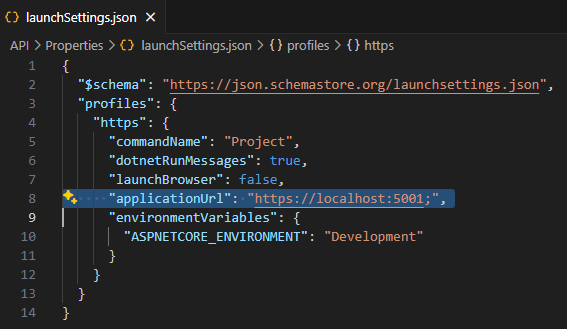
 

Abrir terminal: Ctrl + Ñ

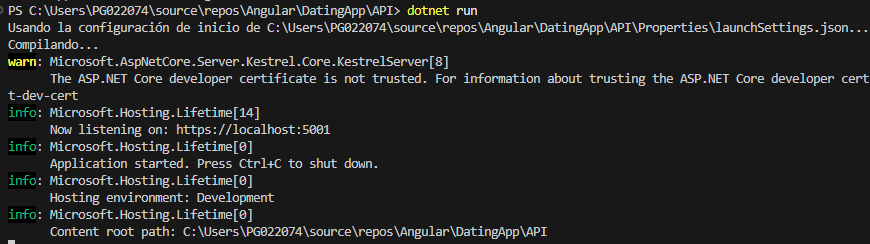
Ejecutar la aplicación:



Vemos que no se muestra nada, así que modificamos el archivo Properties/launchSettings.json

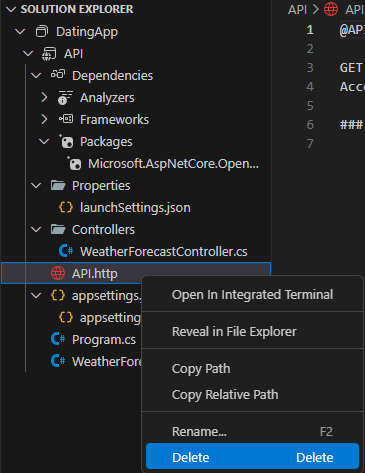


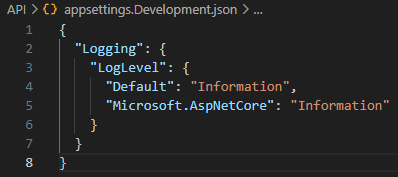
Volvemos a ejecutar la aplicación:



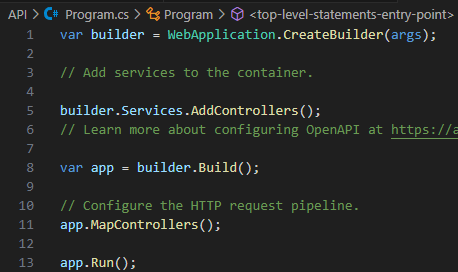
No muestra nada, pero ya está lista. Si vamos a la siguiente dirección, mostrará la clase que hay: <https://localhost:5001/WeatherForecast>

Eliminamos los siguientes archivos (API.http(:

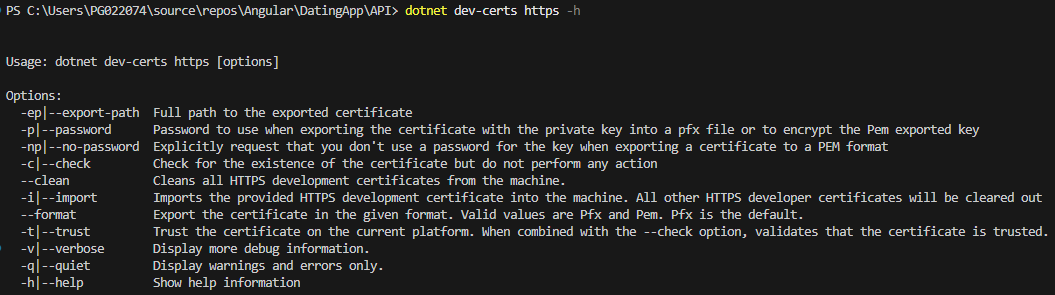


Cambiamos el Warning por el Information para recibir más información en el terminal.

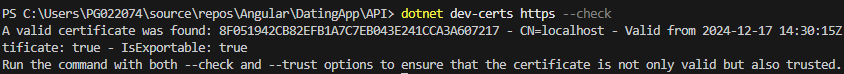
En este curso no se va a trata ciertas cosas; así que se quita:



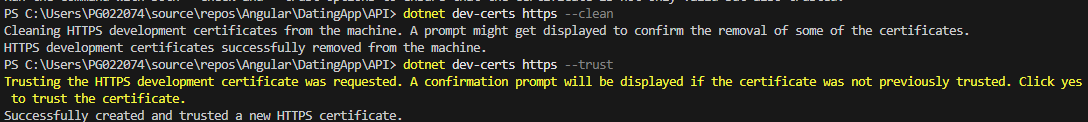
Ayuda por si no funciona algo y así sabemos el estado;



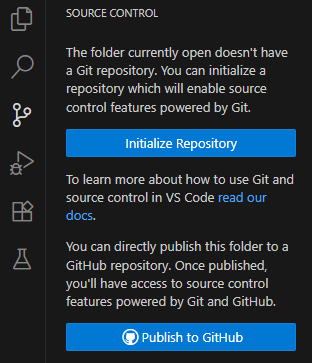
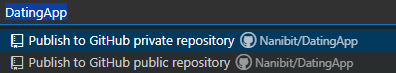
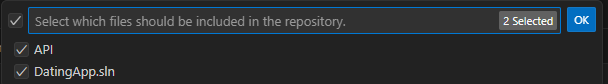
Situación actual con nuestros certificados:



Si da error, cambiar el check por clean y luego trust

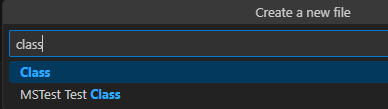


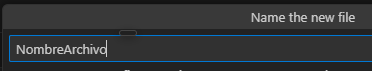
GitHub

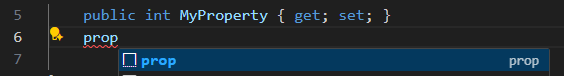


Crear una nueva clase C#:

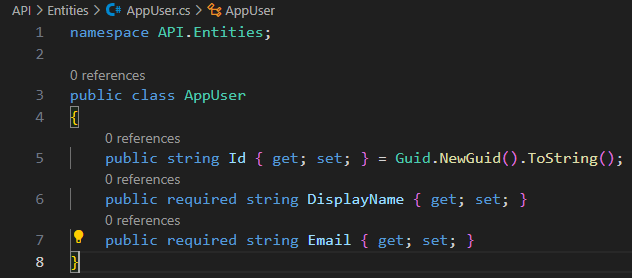


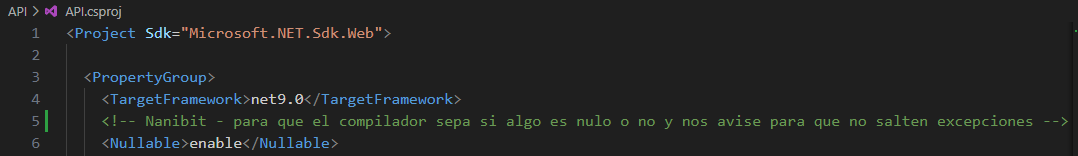




Si escribimos Propp y apretamos tabulador, nos escribe una propiedad auto con su get y set.

Y tiene que ser pública porqué EntityFramework está fuera de nuestra clase y necesitará tener acceso.

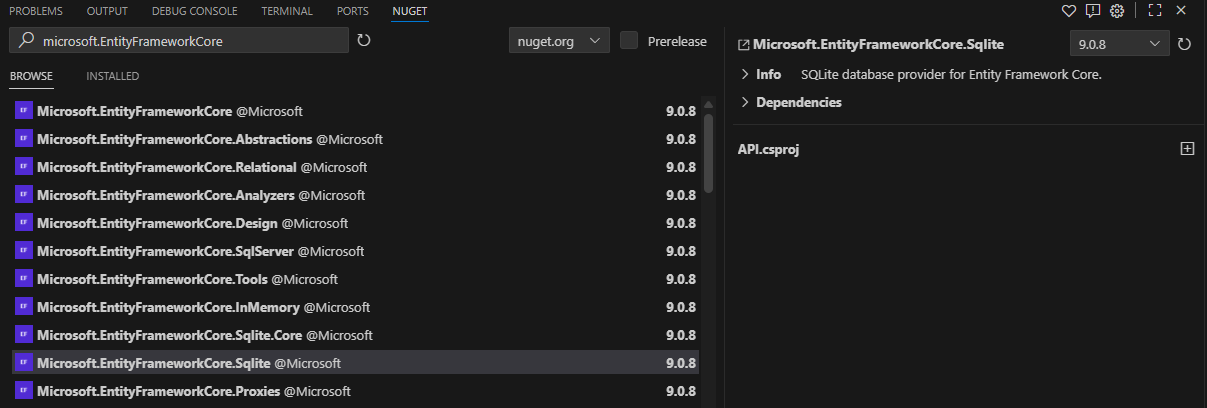




Instalar EntityFramework:

Vamos al terminal, en la parte de NUGET, desmarcamos Prerelease y escribimos Microsoft.EntityFrameworkCore

Escogemos el Design y apretamos el símbolo + de la derecha y también instalamos el Sqlite (sin el core)



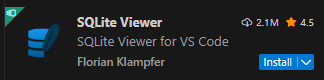
En API/API.csproj



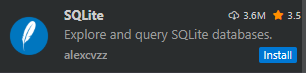
Se ha borrado la línea del código de OpenAI. Para actualizar el proyecto y actualice los paquetes, porqué lo hemos cambiado manualmente, se tiene que escribir en la línea de comandos:



Instalar las siguientes extensiones:



Viewer: Nos permitirá hacer clic en el archivo base de datos y ver lo que hay dentro de la bd.



Nos permitirá consultar la bd, luego lo podremos desinstalar.

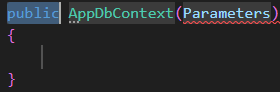
Creamos una carpeta Data y dentro un archivo del tipo clase con el nombre AppDbContext



Escribimos ctor y luego tabulador:



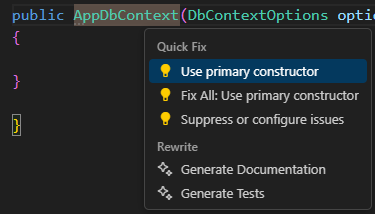
Se nos convierte para especificar las opciones:

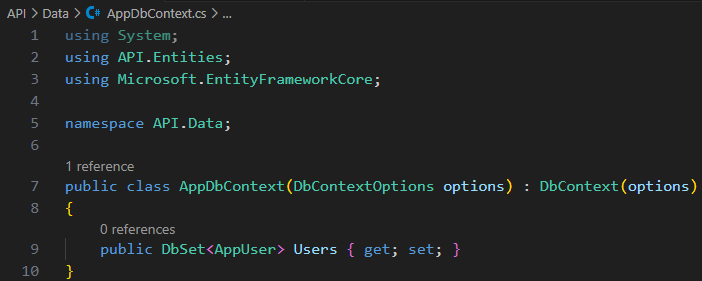


Forma antigua de hacerlo:

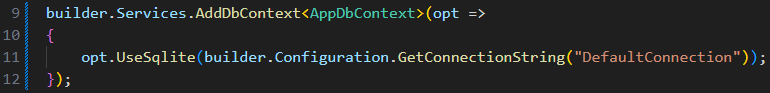


Si señalamos la palabra AppDbContext y presionamos Ctrl + . (punto) se abrirá el menú rápido:





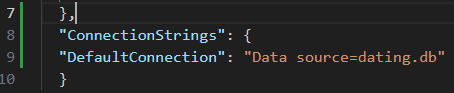
En el archivo Program.cs, añadir el servicio y la cadena de conexión:



Video 13 – 59:30

Crear la cadena de conexión para decirle a nuestra aplicación donde se encuentra la base de datos:

En el archivo appsettings.Development.json, cuidado que no detecta ciertos errores:

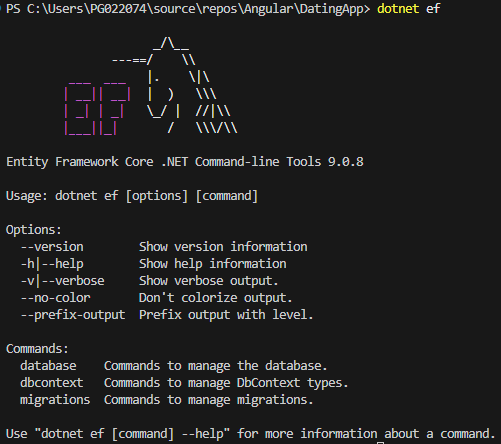


Creación de una migración para recuperar el esquema de la base de datos:



Debe salir vacío porqué aún no hay ninguna instalada. Para hacer la instalación:



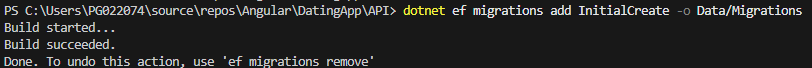






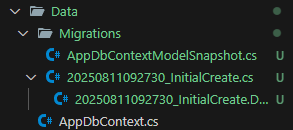
Asegurarnos que estamos dentro de la carpeta del proyecto para hacer la migración:

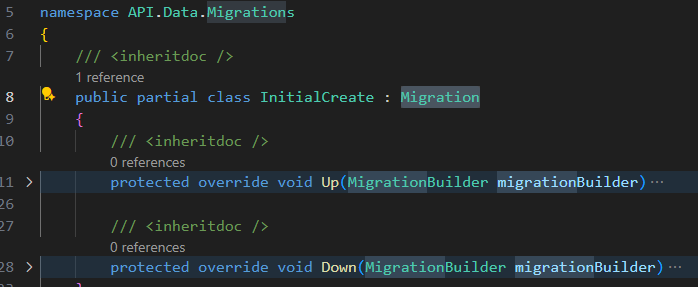




\*asegurarnos de que para hacer una migración, nuestro servido API no está en ejecución:

Ahora nos ha creado:



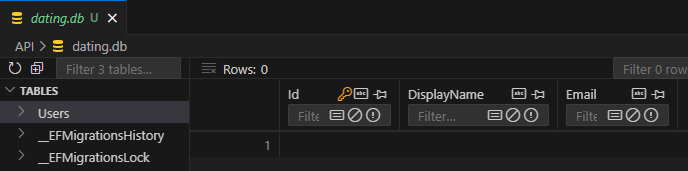


Video 14 – 1:05

Creación de la base de datos basada en la migración (con el comando update):

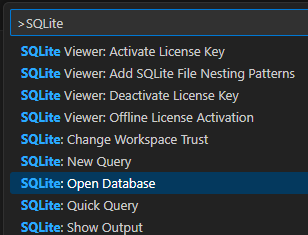


 con la extensión SQLite

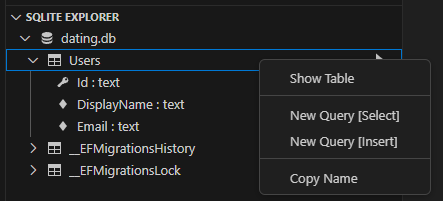


Abrir nuestra Base de datos:

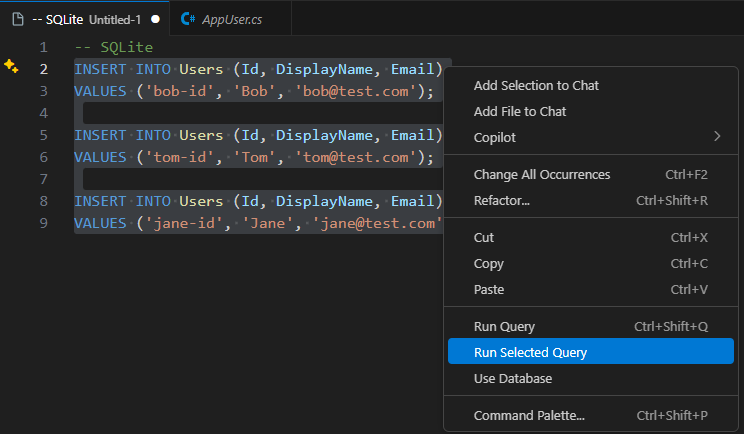


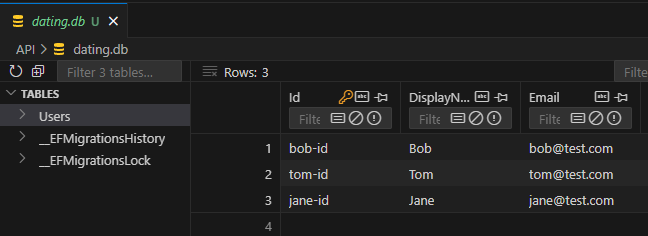


Con esto ya estamos conectados a la Base de Datos.

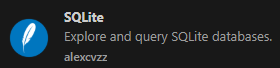


Botón derecho en Users / New Query Insert

Y para ejecutar la Query, seleccionar el código / Run Selected Query



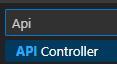
\*Se puede eliminar la extensión SQLite ya que solo sirvió para insertar los datos.

 Desinstalar y reiniciar extensiones

Creación de controlador API

Video 15 – 1:44

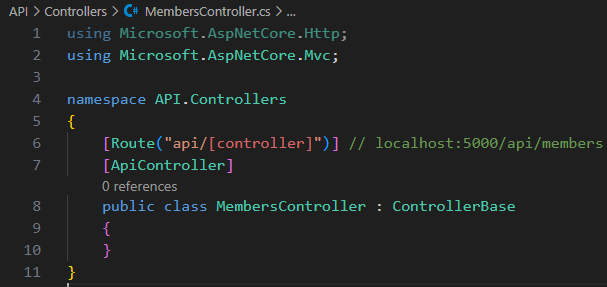
Para consultar datos en la Base de datos creamos un controlador API con puntos finales para recibir una solicitud HTTP y devolver una respuesta HTTP.

\*Members en plural, Controller en singular (para que luego coja bien las rutas)

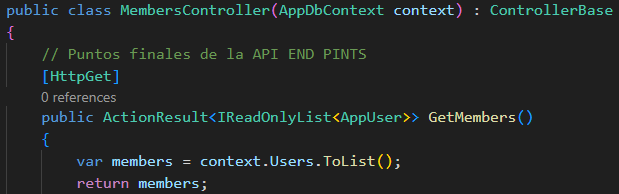
Se reemplaza con la primera parte del nombre del controlador. Así que solo funciona si está bien escrito.



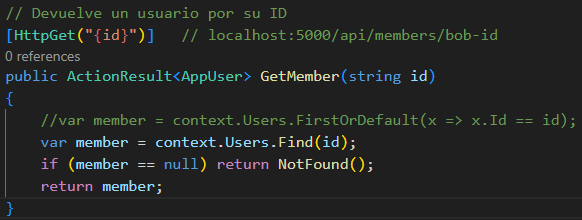


Una solicitud API devuelve una respuesta API en formato JSON

En una solicitud HTTP, entra, se dirige a este controlador en esta URL localhost:5000/api/membres y si es una petición HTTP get, ejecuta el código de dentro.



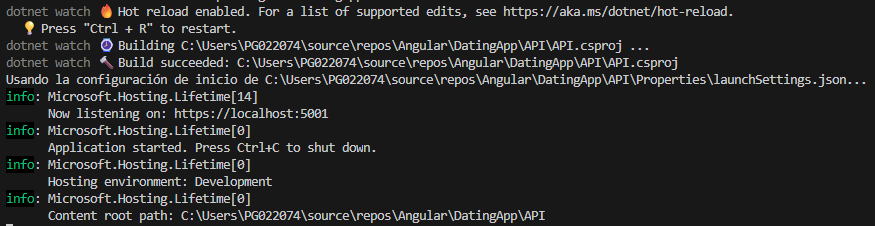
El código devuelve una lista de miembros



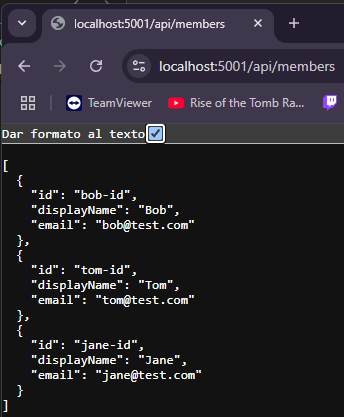
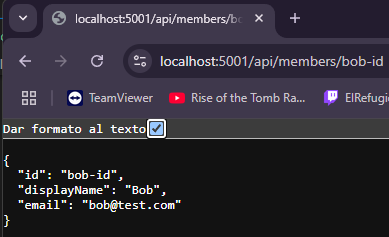
Iniciar la aplicación para ver si funciona:



Va a reconocer si hemos hecho algún cambio y luego intentar reiniciar la aplicación o utilizar la carga en caliente (Hot Reload). Si algo no funciona reiniciamos el servicio



Abrimos el navegador. Control + click:

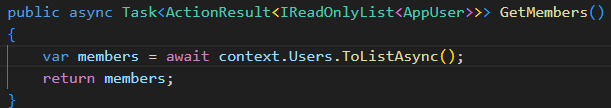
 Si ponemos /bob-id sólo nos muestra el id en concreto. Funciona!

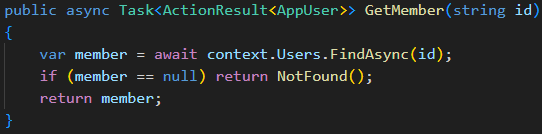
Creando el código asíncrono

Video 16 – 1:24

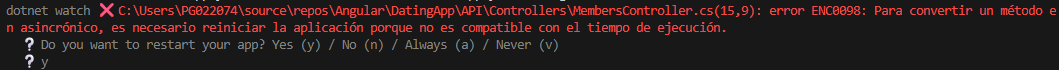
Código síncrono: mientras esta petición va a la base de datos, el hilo en el que entra la petición se bloquea hasta que se devuelven los datos y finaliza la petición.

Lo vamos a pasar a asíncrono: En lugar de que la petición se bloquee cuando entra en este API controlador y mientras esperamos a que ocurra algo, la acción de salir a nuestra base de datos se delega a otro hilo y ese hilo puede dedicarse a consultar la base de datos, y si se trata de consulta de base de datos de larga duración, no va a afectar a nuestro controlador API.





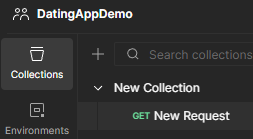
Reiniciamos el servicio con la tecla y



Para ver el resultado en Postman, el servicio debe estar activo:



Creamos un nuevo workplace DatingAppDemo y en Collections, una nueva Requast:





Podemos importar el archivo de postman que se adjunta.

<https://drive.google.com/file/d/1iLXIcES0b5bfqZZRBI_lvqNWn-4q7enn/view?usp=drive_link>



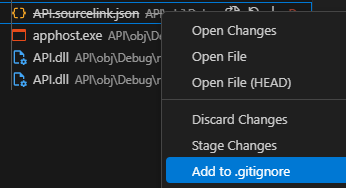
Git y GitHub

Video 17 - 1:33



Con esto ignora todos los archivos que no escribimos nosotros. Podemo verlos dentro del archivo .gitignore

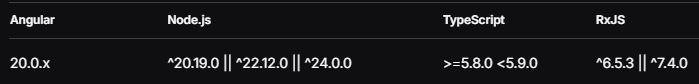
También podemos hacerlo individualmente:



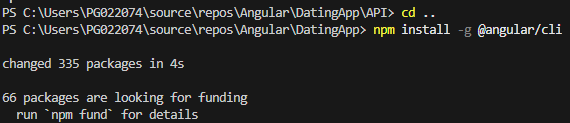
**ASPNET Core API Angular 03**

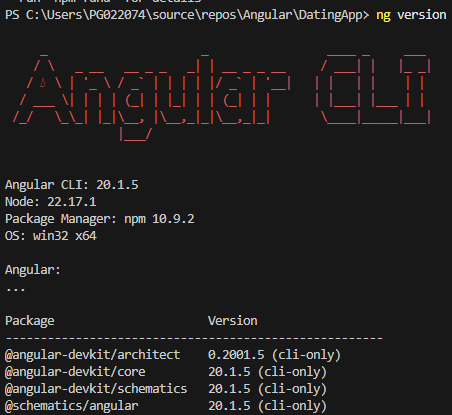
Video 20 – 02:00

Comprobar e instalar Angular



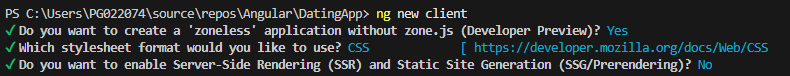




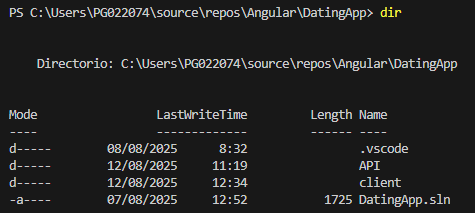


Crear proyecto Angular:

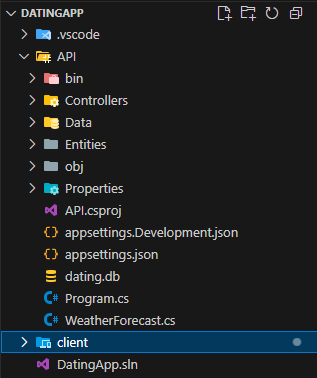
Nos aseguramos de que estamos en la carpeta donde se encuentra la solución:



Si listamos, vemos que se ha creado una carpeta nueva llamada client:



En la columna de la izquierda, en SOLUTION EXPLORER, no vamos a ver la carpeta, tenemos que ir a la vista del explorador de archivos con el nombre de la aplicación (DattingApp):

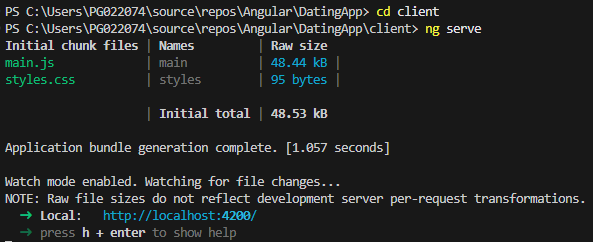


Whats in the Angular ptoject

Video 21 – 13:34

Ejecutar el proyecto Angular:

Debemos estar en la carpeta cliente:



Así se llaman los componentes, con la etiqueta selector en el archivo \*.ts:



Para mostrar el componente, en el html se escriben las etiquetas:



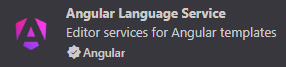
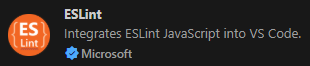
Los archivos spect.ts se utilizan para hacer pruebas, en este caso los eliminaremos.



Adding extensions to Vs Code for Angular

24:10

Instalar la extensión:

La paleta de comandos(Control + Shift + P): (escribiendo > arriba en el buscador)

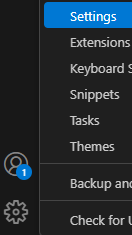
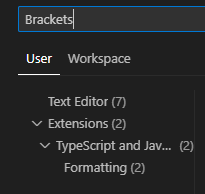
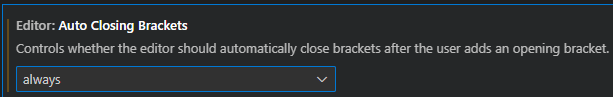


Con esto se recarga la ventana por si creemos que puede haber algún problema.

Para activar el cierre de los corchetes en html, es un ajuste que se tiene que configurar:

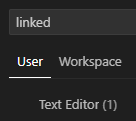
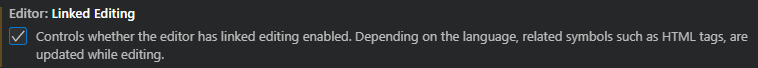


Herramientas / Settings:

Con esto también tenemos ayuda al escribir dentro de los corchetes.

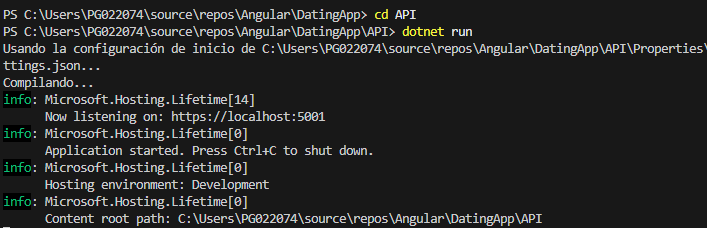
Si cambiamos el primer nombre de la etiqueta de la obertura, no se actualiza la etiqueta de cierre. Para que lo modifique volvemos al menú anterior y buscamos linked:

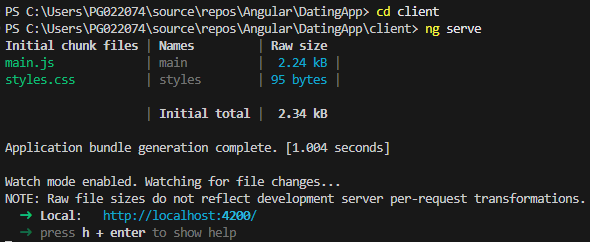
Making HTTP request in Angular

Video 23 – 28:28

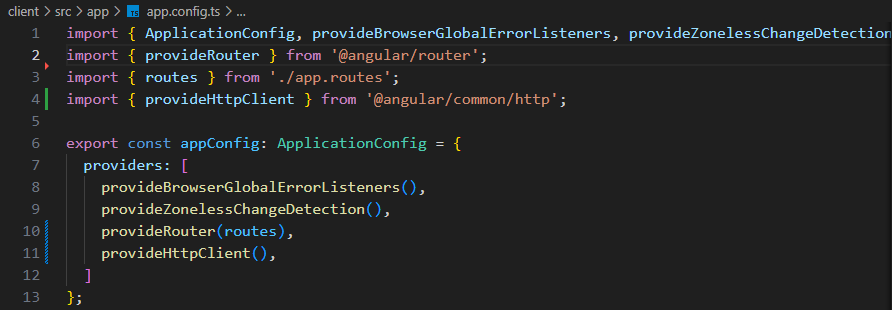
Ejecutar proyecto c#:



Ejecutar proyecto Angular:



Añadimos el servicio del proveedor de HTTP:



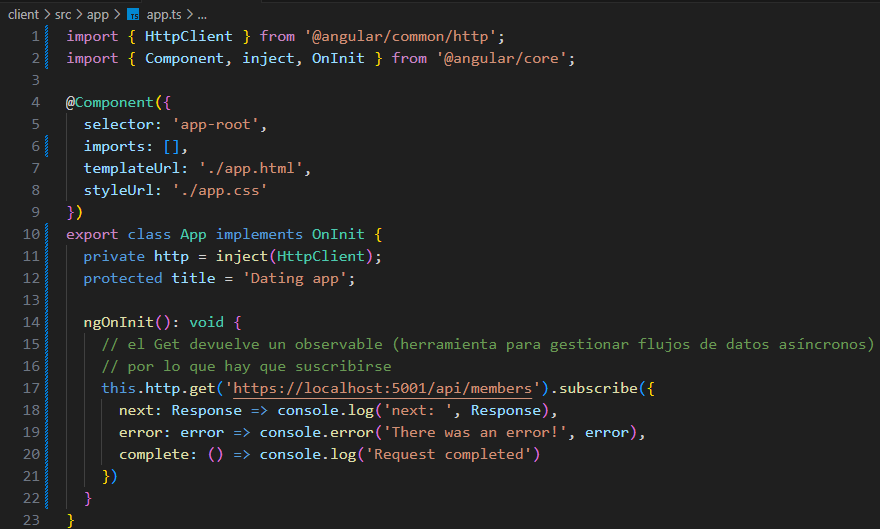
Después inyectamos el cliente HTTP en el constructor del archivo TS



Pero es mejor ponerlo de la siguiente forma en vez de en el constructor:

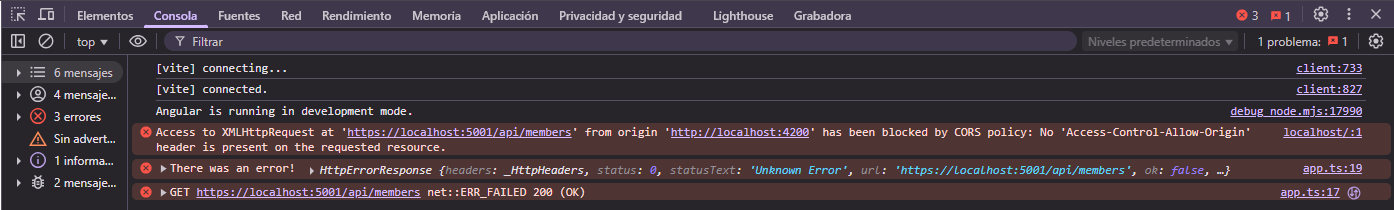


En app.ts:



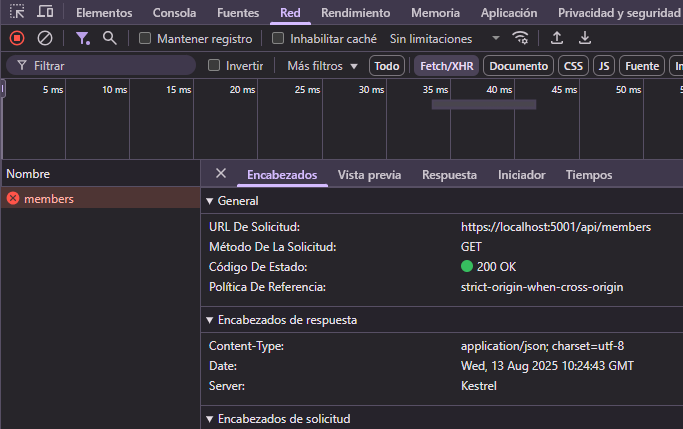
Comprobamos el funcionamiento

En el navegador presionamos la tecla F12:

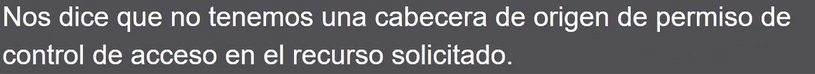


No da el error: Access to XMLHttpRequest at 'https://localhost:5001/api/members' from origin 'http://localhost:4200' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

Si vamos a Red, Fetch/XHR y members:



Tenemos que modificar algo en la aplicación API para que no muestre el error: has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource

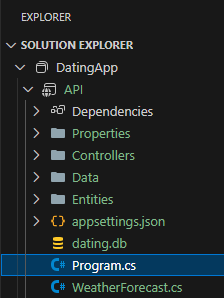


Así que vamos a habilitar los núcleos para que podamos ver los datos en la respuesta del servidor.

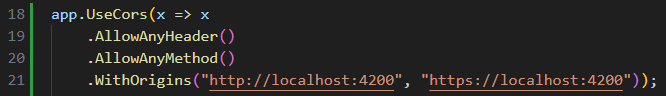
Adding CORS configuration to the API

Video 24 – 38:45

El error es de política de compartición de recursos entre orígenes, ya que Angular viene de localhost 4200 y el servidor API viene de localhost 5001. Añadiremos en el servidor API un encabezado.





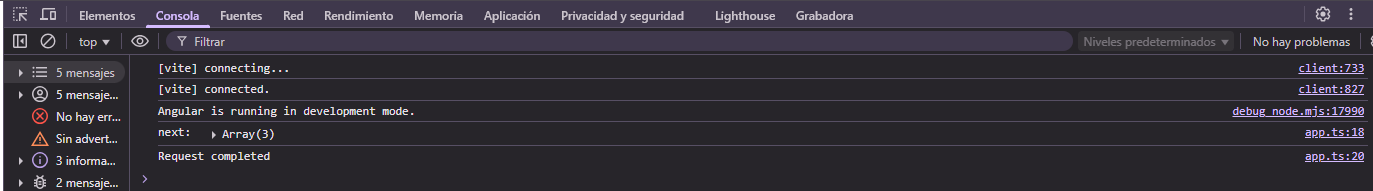


Reiniciar el servicio NET:

Control + C



Ya podemos ir al navegador y vemos que no hay error

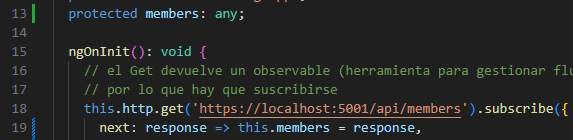


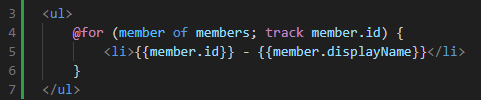
Las subscripciones html se completan, así que no hace falta darse de baja de la suscripción.

Displaying the fetched users in the browser

Video 25 – 45:21

Usaremos la propiedad Protected cuando queramos que esté disponible para la propia clase y sus plantillas.





Puede que no se refleje la lista, porqué cuando creamos la aplicación angular, la creamos usando listas de zonas

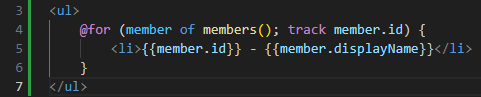


Señales en angular:

Para que el navegador pueda detectar los cambios y mostrarlos, se tienen que utilizar las señales, entonces no se puede utilizar el código anterior, sinó el siguiente que se muestra:

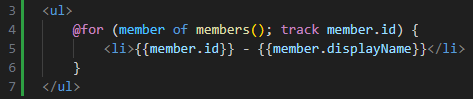






Pero a partir de ahora lo aremos de la siguiente manera:





Adding Tailwind and DaisyUI for styling

Video 26 – 57:17

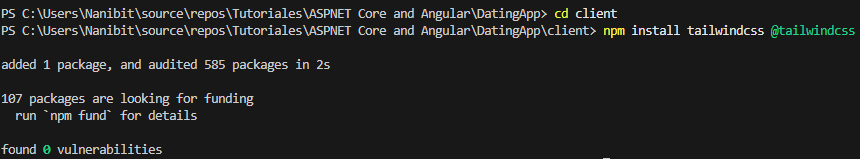
Tailwind proporciona estilos CSS par anuestro HTML, però no proporciona Componentes (botones, formularios...) así que vamos a utilitzar Daisy UI que es un pluguin CSS de Tailwind que proporciona Componentes sin escribir mucho CSS.

Get Started / Installation / Framework Guides / Angular

Seguimos la instalación, però seguramente no necesitemos el postCSS así que esa parte no la vamos a instal·lar.

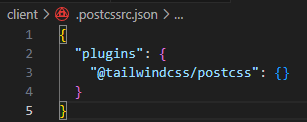
Tenemos que asegurarnos que estamos en la carpeta client, la de Angular.

npm install tailwindcss @tailwindcss/postcss postcss --force

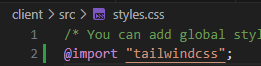




Creamos archivo en carpeta cliente:



En el fichero de estilos globales importamos:



PARAMOS EL SERVIDOR Y VOLVEMOS A ARRANCARLO:



Haciendo esto, las etiquetes de estilo html como el h1, h2, ya no funcionan, tenemos que utilitzar etiquetes tailwind.

Vamos a la web de DaisyUI <https://daisyui.com/>

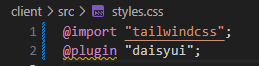
How to use / Install

Instalar la depndencia:





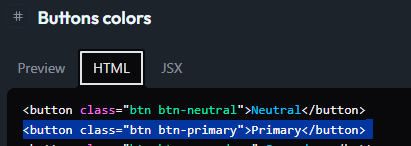
Añadir el pluguin dentro del CSS:



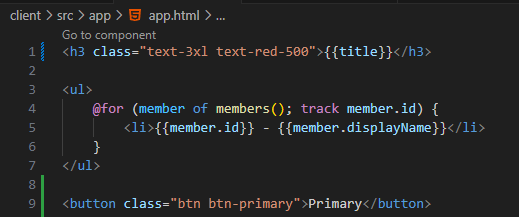
\*Si funciona ignoramos la advertència

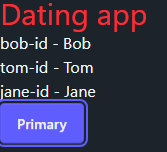
Instalación de un botón:

Copiamos el código HTML de la web de Daisy:



Pegamos el código en el HTML

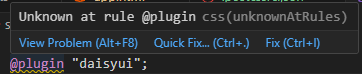




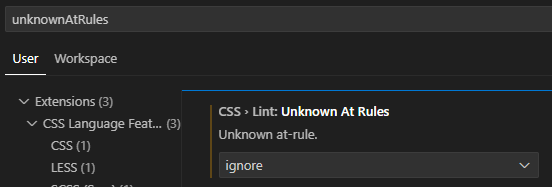
Using the new styling tools in our app

Video 27: 1:07

Para quitar la alerta:

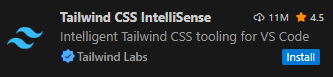


Vamos a Herramientas / Settings, buscamos unknownAtRules y escogemos Ignore:

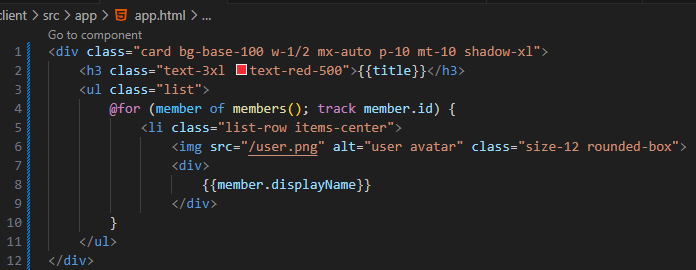


Al empezar escribir en el HTML la classe font- no obtenemos ninguna ayuda, así que vamos a intalar una extensión para que nos ayude:

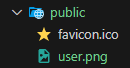




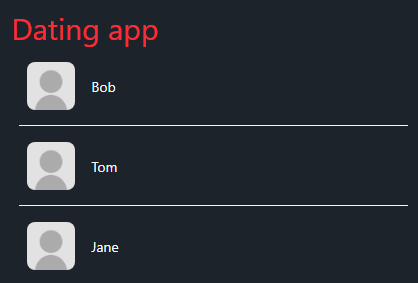
Modificamos el archivo HTML



Todo lo que dejemos dentro de la carpeta publi, no hará falta especificar en la ruta que està en la carpeta públic.



Reiniciamos Servicio del ng serve



Running the Angular app over https

Vídeo 28: 1:15