Angular – Ejemplo Lista de compras

<https://www.youtube.com/watch?v=tP42hCXf-Ro>

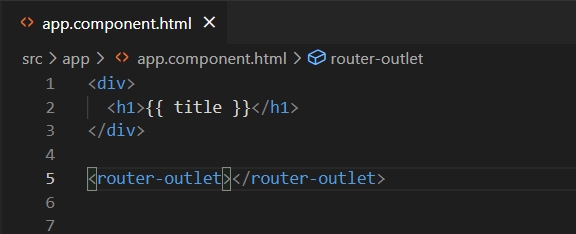
## Prueba de la aplicación

Como el ejercicio ya está completo, tiene la parte de carga de datos desde el backend por lo que para verlo fncionar hay que levantar el servidor backend desde backend/node server.ts.

Luego al iniciar la aplicación con ng serve se la vera funcionar. Para probarla sin el backend habrá que hacer un cambio de aquellos archivos q figuran como .old.

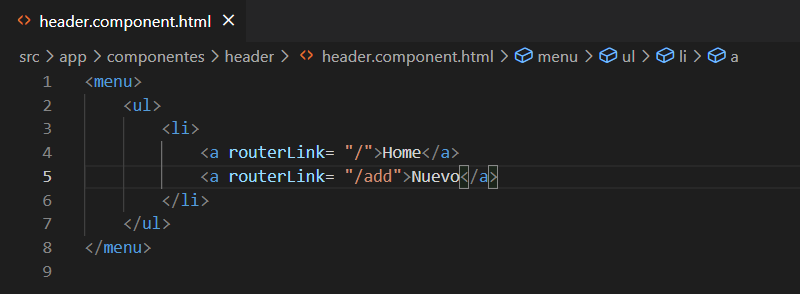
## Iniciando

Vamos a crear una aplicación que llamaremos Lista de compras. Provamos que la aplicación inicial levanta y luego porcedemos a limpiar el app.component.html para quitar lo que se agrega para que muestre la aplicación inicialmente. Solo dejamos algo así.



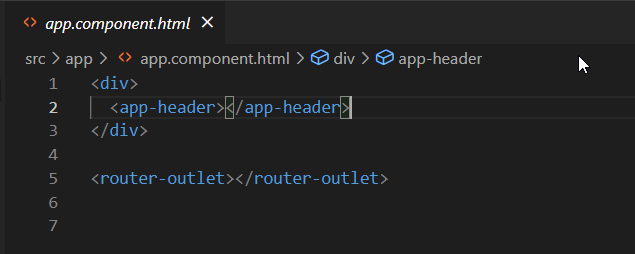
## El componente para navegar la pagina

Vamos a generar ahora un componente para el encabezado de la página (header) en el cual en principio imaginemos tener un navegador que nos permita navegar la pagina. En el archivo header.component.html haríamos algo del estilo



En la etiqueta <a></a> en vez de utilizar un href= a una URL, vamos a utilizar un routerLink= a una ruta de la aplicación.

El componente stendra el selector app-header, con lo cual en el html del archivo padre lo llamaremos con la etiqueta <app-header></app-header>



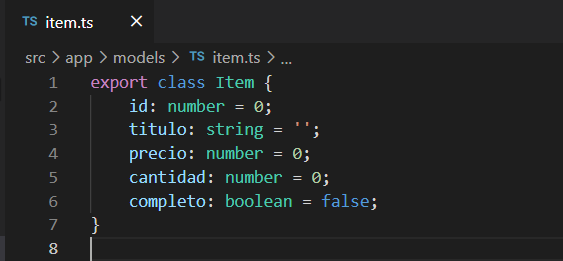
## El componente para los ítems de la lista de compras

Ahora vamos a pensar otro componente que será el encargado de mostrar y mantener los ítems de la compra lo llamaremos items. La etiqueta que nos creara para usarlo será <app-ietms> y mas adelante veremos como se implementara tanto su código como su html.

### El modelo de datos

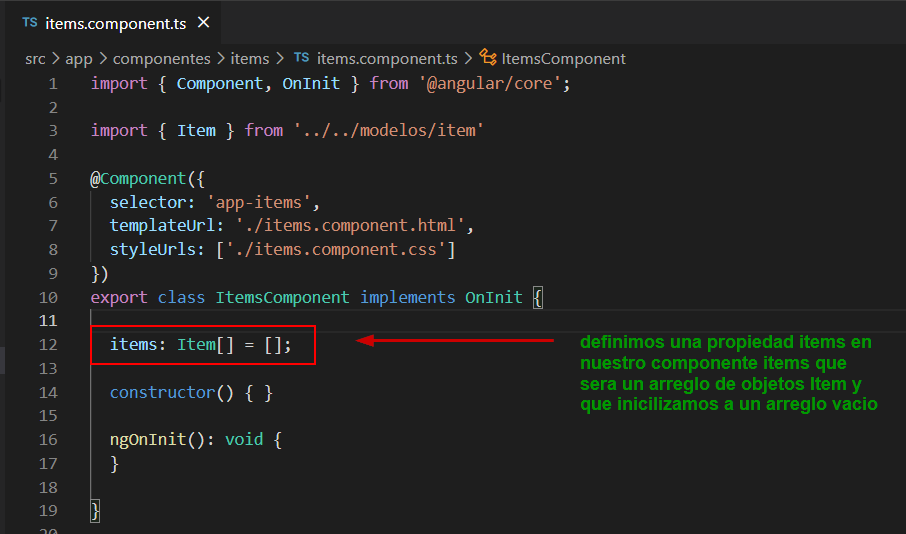
Pensemos que vamos a necesitar un modelo de datos que llamaremos Item, se definirá en una clase Item. Vamos a crear una carpeta Models donde vamos a mantener las clases referentes a los modelos de datos

que se requieran para la aplicación. Por el momento solo será el el correspondiente a la clase Item y estará en el archivo item.ts.

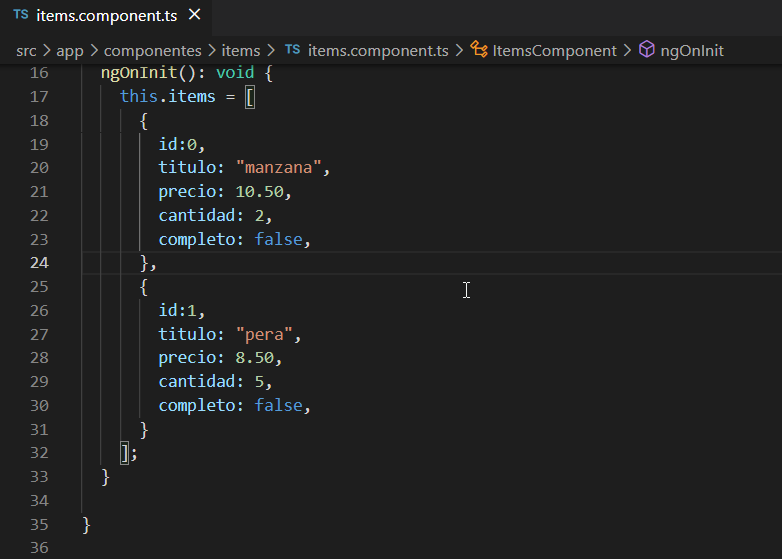


Debemos darle valor inicial a las propiedades del objeto ya que cuando creamos el proyecto angular le indicamos yes al tipado estricto.

Entonces ya teniendo el objeto Item definido, a éste lo vamos a utilizar en nuestro componente ítems. Dentro del componente vamos a definir una propiedad Items que es del tipo arreglo de objetos Item y será necesario inicializarlo a [] debido a que cuando creamos el proyecto Angular le indicamos que deseábamos tipado estricto.

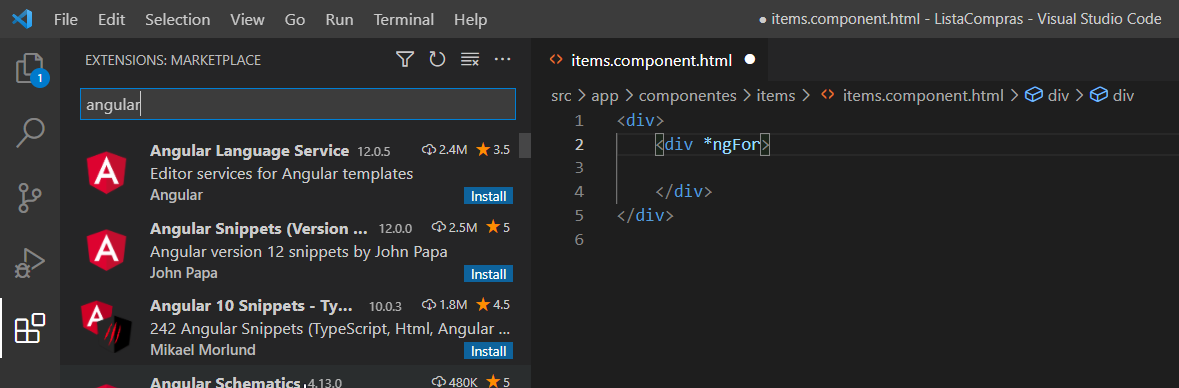


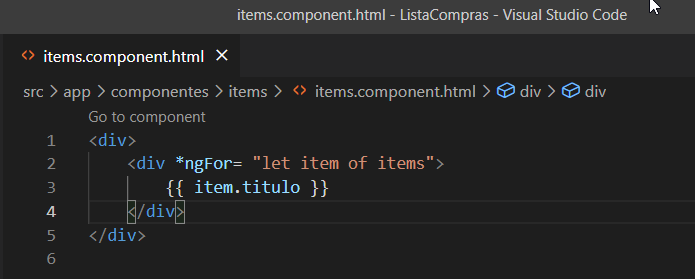
En el método ngOnInit() que es el método que se ejecuta cuando se crea el componente (o, más bien, cada vez que este se dibuja en la pantalla) voy a agregar algunos elementos al arreglo Items



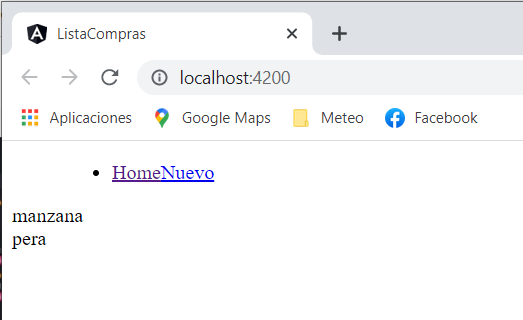
Ahora que tenemos algunos datos queremos mostrarlos y esto lo hacemos en el archivo html utilizando una directiva ngFor dentro de un bloque <div> que nos permitirá recorrer el arreglo de ítem.

Para facilitar el trabajo voy a agregar una extension al VSCode de Angular, el Angular Lenguaje Service que nos facilitara la escritura de código de Angular al reconocer la sentaxis de éste





La pagina me mostará el campo titulo de los ítems actuales en el arreglo.



## El componente para cada item de la lista de compras

Lo que voy a hacer ahora, en vez de mostrar los componentes de cada ítem, es crear un nuevo componente ítem para tal motivo. No se va a generar conflictos de nombre con nuestro modelo de dato ítems porque este está en la carpeta /modelos y el componente ítem en la carpeta /componentes y además se llamara item.component y la clase ItemComponent.

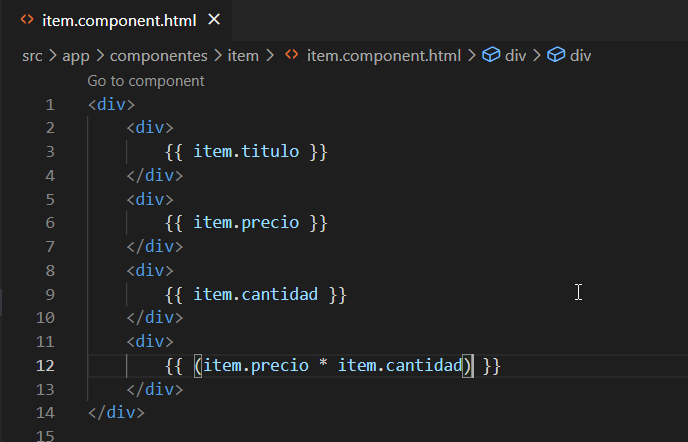
En el ítem.component.html vamos a armar algo que muestre todos los componentes del ítem, luego en el ítem.component.css vamos a darle un estilo. Obviamente a esta altura el componente no va a conocer al ítem, pero vamos a ver como lo pasamos desde el componente padre para poder trabajarlo en este.

### Uso de parametros o propiedades de entrada para un componente (decorador @input)

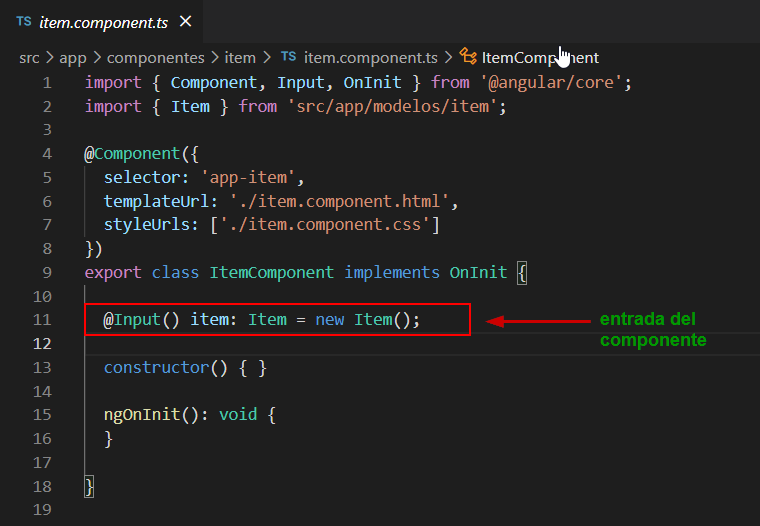
Hay varias formas de especificar que un componente esta recibiendo parámetros, tenemos por ejemplo tanto propiedades de entrada como propiedades de salida, vamos configurar una entrada para indicarle a este componente que va a recibir un objeto y para esto vamos a utilizar la notación de angular con un decorador

**@Input() nombre\_propiedad: tipo\_propiedad = valor\_inicial**

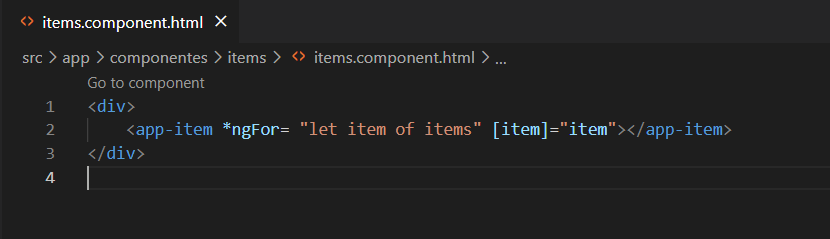
El item.component.html tendrá la siguiente forma



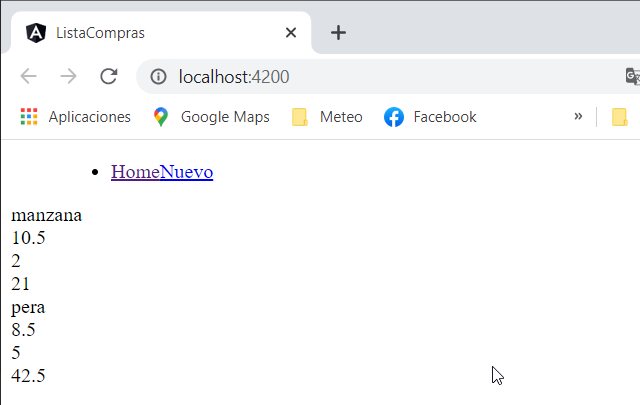
por otro lado en el ítem.component.ts allí tengo que definir la propiedad con el decorador @Input()



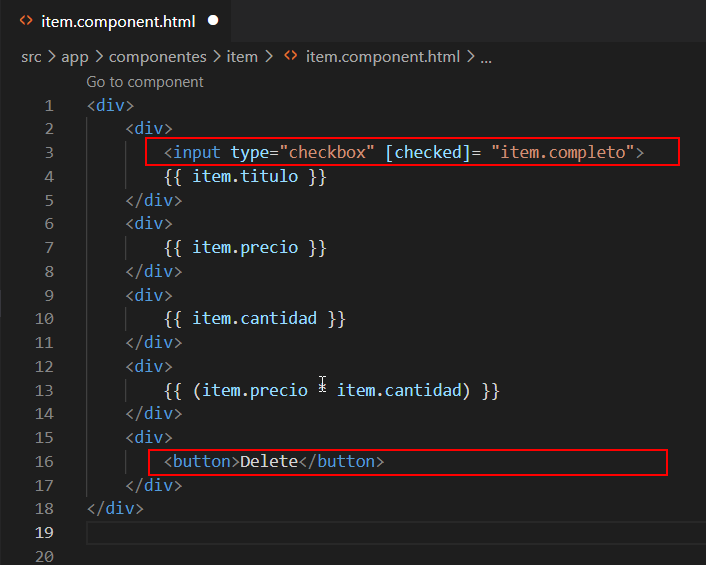
y en el componente padre vamos a reemplazar el bloque <div></div> por un bloque con la etiqueta del componente <app-ietm></app-ietm> mantenemos la directiva \*ngFor par arecorrer el arreglo de ítems y allí pasamos el argumento [ítem] al componente hijo



el resultado sería algo así



Vamos a agregarle a cada ítem un botón Eliminar y un checkbox que estará seteado o no de acuerdo a valor de completo. El primero lo logro agregando un bloque <div> al final con una etiqueta <button></button>, mientras que el checkbox lo hacemos agregando una etiqueta <input type=”checkbox”></input> dentro del primer bloque <div>. Como queremos que el esta casilla este relacionada dinámicamente con el valor booleano del valor completado del ítem, agregamos la propiedad checked y le asignamos el valor de completado. Se usa [] para la propiedad checked porque deseamos que tome el valor dinámicamente.

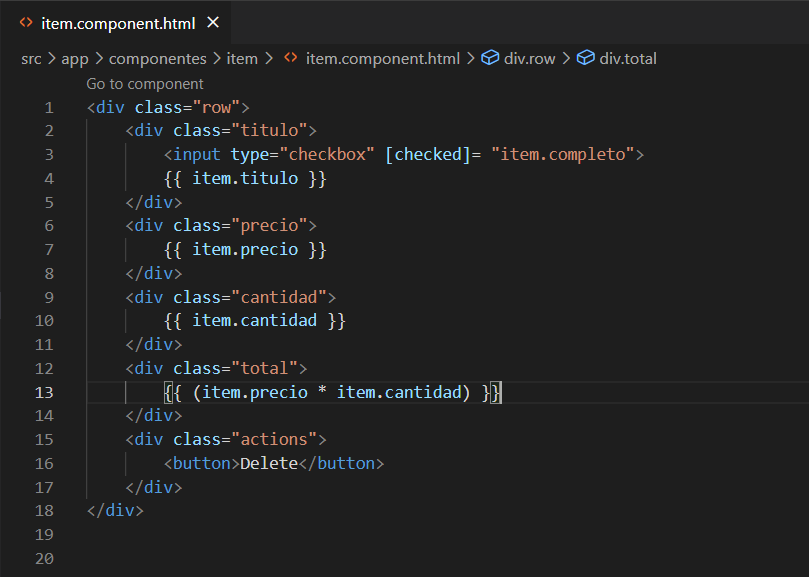


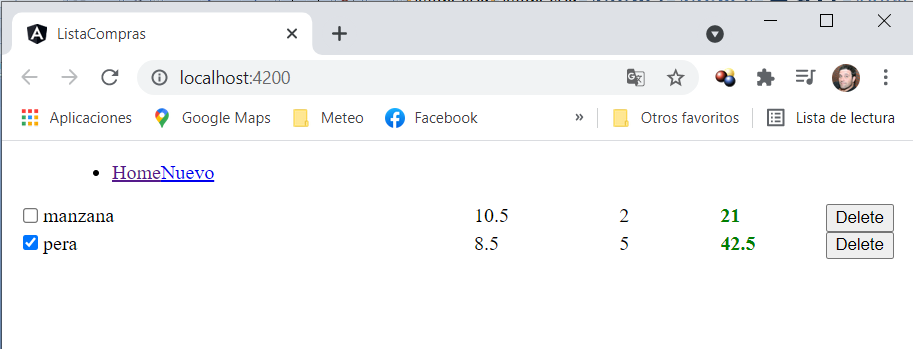
### Uso de estilos para el componente

Por otro lado en el ítem.component.css vamos a darle un estilo.

En el .html defino un nombre para la propied class para la etiqueta de cada componente, de manera de poder darles un estilo individual a cada uno de ellos en el .css.

Por ejemplo al bloque <div> mas externo de doy el nombre de class=”row” y en el archivo de estilos le doy el estilo a la fila dándoselo a .row. De la misma manera lo hago para cada componente del ítem



### Uso del operador pipe en Angular

Angular nos deja utilizar el operador pipe “|” que nos permite aplicar funciones a nuestros valores, por ejemplo

**{{ item.titulo | uppercase }}** 🡪 pasaría el titulo a mayúsculas

**{{ item.precio | currency }}** 🡪 agregaría el $ para darle un formato de moneda al valor

### Uso de eventos o propiedades de salida (decorador @output)

Vamos a ver con el botón eliminar el caso del uso de eventos o propiedades de salida de nuestro componente. Utilizando paréntesis () en la propiedad referente a un evento en nuestro componente y dándole como valor el nombre de un método que manejara el evento

**<button (click)= “onDelete(item)”>Delete<button>**



De esa manera estamos indicando que la propiedad click es un evento del component que responderá con el metodo onDelete() y le estamos pasando el elemento ítem correspondiente. Obviamente el método onDelet() que manejara dicho evento click debe esta definido en el componente en el ítem.component.ts y allí tendrá el código correspondiente que deberá ejecutarse.

Ahora bien, como nuestra fuente de datos esta en el componente padre ítems.component y no en el componente hijo ítem.component, voy a tener que llamar a un evento que elimine el ítem en el componente padre, es decir, existirá un método definido en el componente padre que deberá ser invocado desde el componente hijo y para esto utilizamos las propiedades de salida. En el componente hijo mediante el uso del decorador @output() en el hijo agregamos un parámetro o propiedad de salida

**@output() deleteItem: EventEmitter<Item> = new EventEmmiter()**

Nota: al usar EventEmitter se requiere importar este desde

**import { EventEmitter } from '@angular/core';**

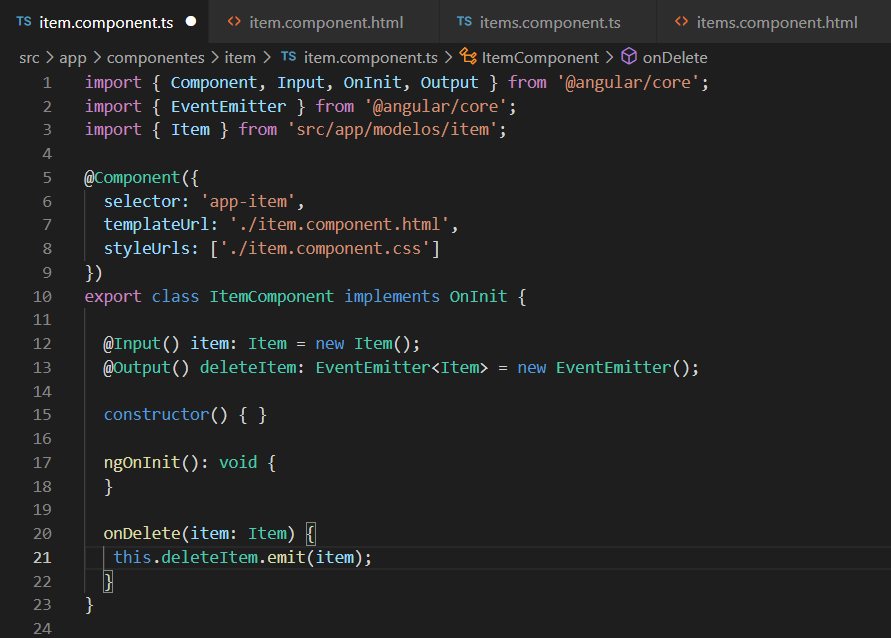
que significa esto, bueno, estamos definiendo en la clase del componente una propiedad de salida que tiene el nombre deleteItem y que es de tipo evento, pero al no ser un evento nativo del componente (sino que corresponde al padre) lo tengo que definir como de tipo EventEmitter, además al ser un tipo genérico debo indicarle que va a cargar o inyectar un tipo de dato especifico, en este caso algo del tipo Item y lo hago usando <Item>. Además, tengo que inicializarlo por lo ya mencionado anteriormente de que al crear el proyecto indicamos que debía solicitar el tipo, al indicarle el uso de tipado estricto.

Por otro lado, en la definición del metodo onDelete() que responde al evento click del componente, vamos a tener que hacer el llamado a esta propiedad recién definida

**onDelete(item: Item) {**

**this.deleteItem.emit(ítem);**

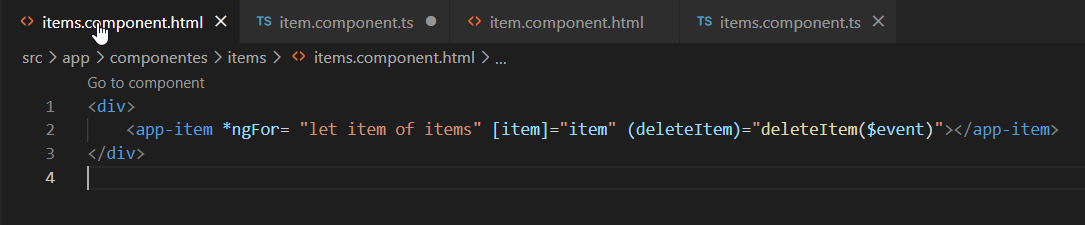
**}**



Lo que estamos indicando es que comience a emitir el ítem determinado, mediante emit(item) donde ítem será el ítem de botón seleccionado y la señal le llegara al componente padre ítems.component.

En el componente padre entonces en el ítems.component.html donde definimos el uso del componente ítem mediante la etiqueta <app-item></app-item> debemos agregar la propiedad de salida en la llamada… ya habíamos agregado la propiedad de entrada

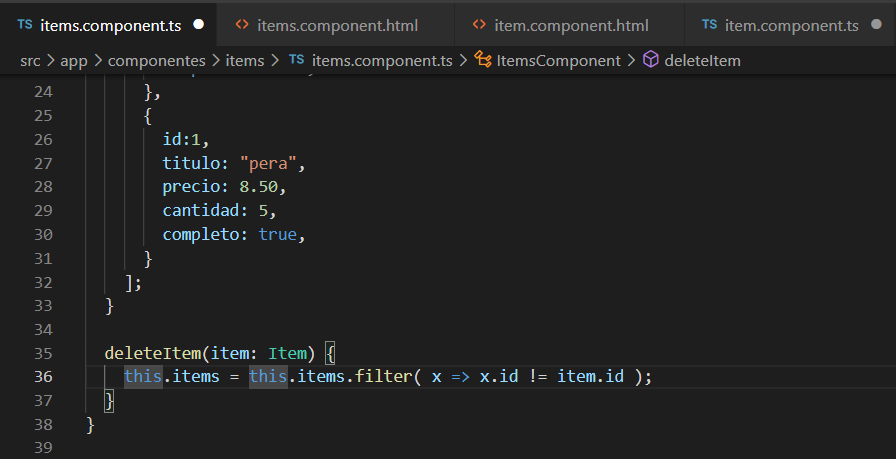
**<app-item \*ngFor=”let item of items” [ietm]=“ítem” (delteItem)=”deleteItem($event)”></app-item>**



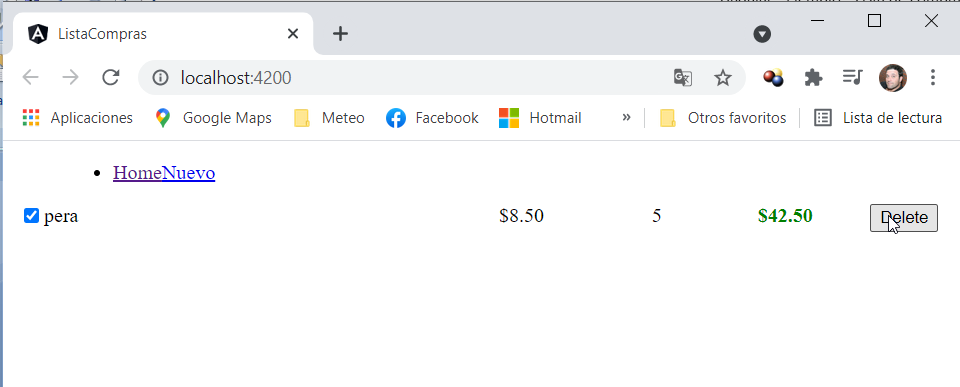
Estamos agregando un propiedad evento que invocara al metodo deleteItem del component y usamos una variable especial $event que es donde recibiremos (o contendrá) el ítem que nos indique el componente hijo que lo esta invocando.

Ahora veamos como se define el método deleteItem() del componente ítems en ítems.component.ts. Lo que vamos a hacer, lo haremos usando la función filter() del arreglos, vamos a asignar al arreglo ítems el resultado de aplicarle el filtro que obtiene todos los elementos del arreglo que sean direrentes al ítem especificado, resultando en un arreglo nuevo sin el elemento a eliminar.

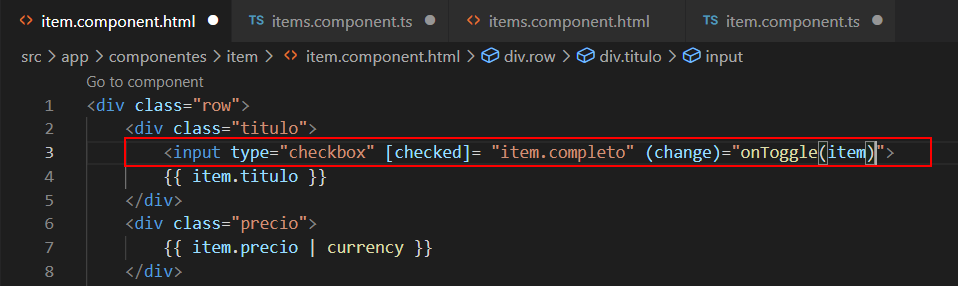
**this.items = this.items.filter( x => x.id != item.id)**



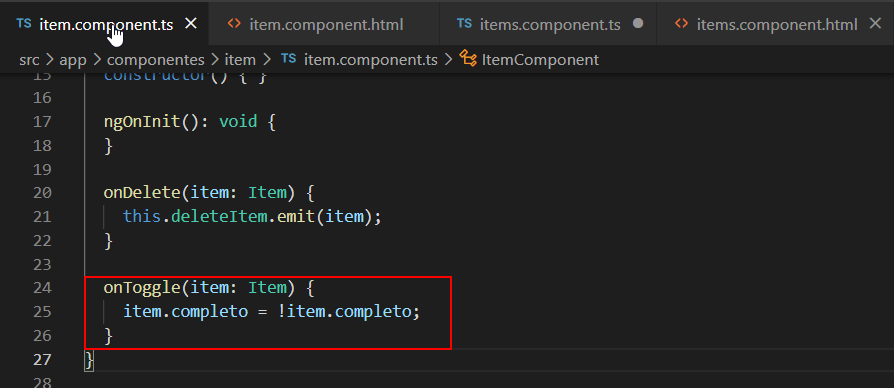
cuando le damos al boton Delete se elimina el item de la lista



Vamos ahora a trabajar con el evento ‘change’ del checkbox, al cual responderá el método onToggle() (conmutar) del componente. Dentro del ítem.component.html en la etiqueta <input type=”checkbox”></input> agregamos la propiedad correspondiente al evento (entre paréntesis) que y le damos el valor del nombre del método “onToggle()” del componente que manejara dicho evento.

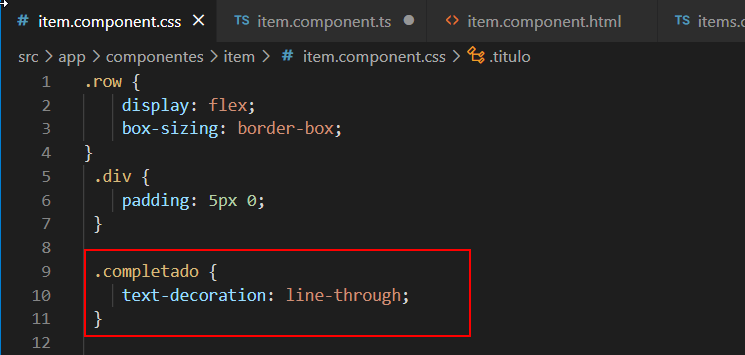


en el ítem.component.ts definimos el método onToggle() que recibe como argumento el ítem correspondiente, en este caso no será necesario trabajar con una propiedad de salida (de tipo evento) para enviar un evento con un método en el componente padre pues no se requiere trabajar sobre el arreglo se ítems sino que la modificación se lleva a cabo en el ítem mismo que es conocido a nivel del component hijo

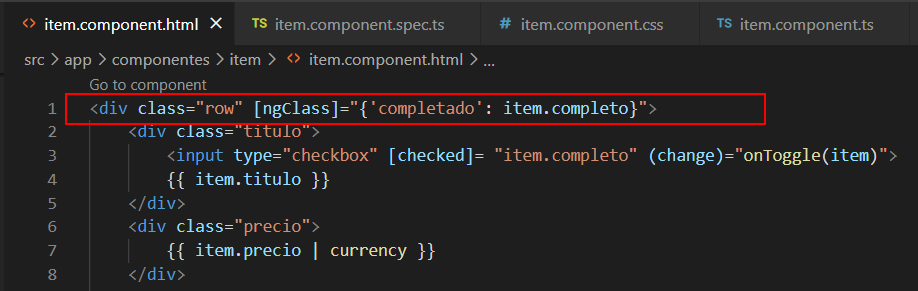


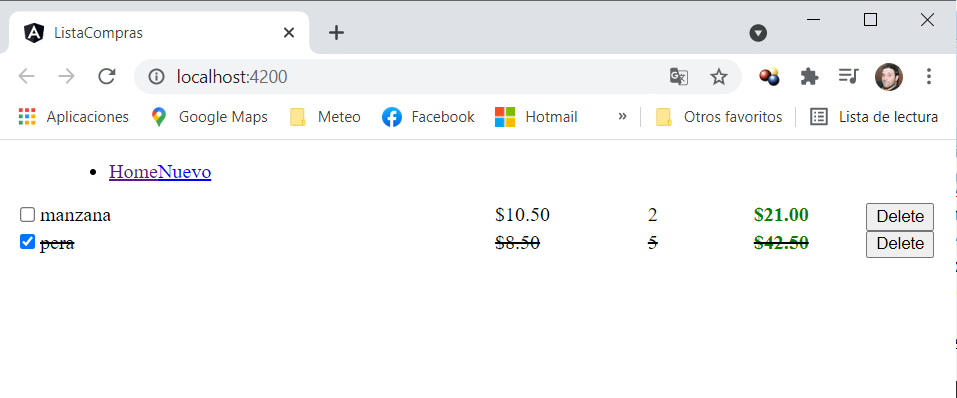
### Uso de condicional de clases de angular ngClass

Con esta directiva de Angular podemos establecer dinámicamente clases CSS en un elemento HTML mediante el enlace de datos de una expresión que representa todas las clases que se agregarán, dichas clases se aplicaran al elemento de acuerdo a la evaluación de una condición. Por ejemplo defino una clase .complatada en el ítem.component.css que tacha el texto del elemento html al que correponde la clase. Pero además, en el ítem.component.html en el elemento html que asocie la clase usando la propiedad [ngClass] de acuerdo a la evaluación de una expresión condicional dentro de un objeto {‘nombre\_clase’: expresión} este estilo se aplicara o no sobre el elemento



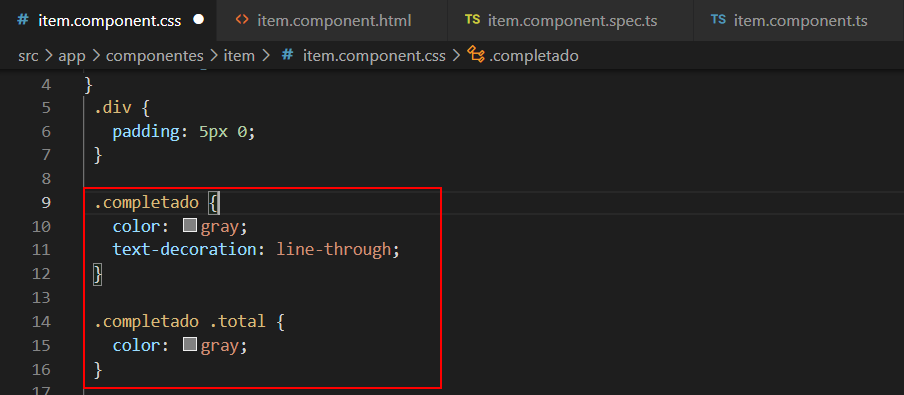
para este caso la expresión condicional será el valor que tenga el ítem.completado y de acuerdo a este aplicara o no el estilo en la clase.

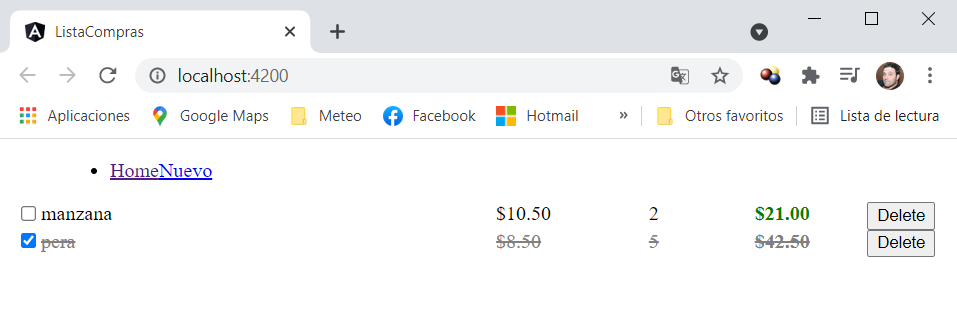




Supongamos que además de tachado lo quiero grisado al texto del ítem que cumple dichas condiciones, en ese caso tengo que agregar en la class del .completado el correpomndiente al gris para el texto

Pero además tengo que tener en cuenta a la clase .total cuyo texto lo tengo en verde por lo cual tengo que hacer lo siguiente





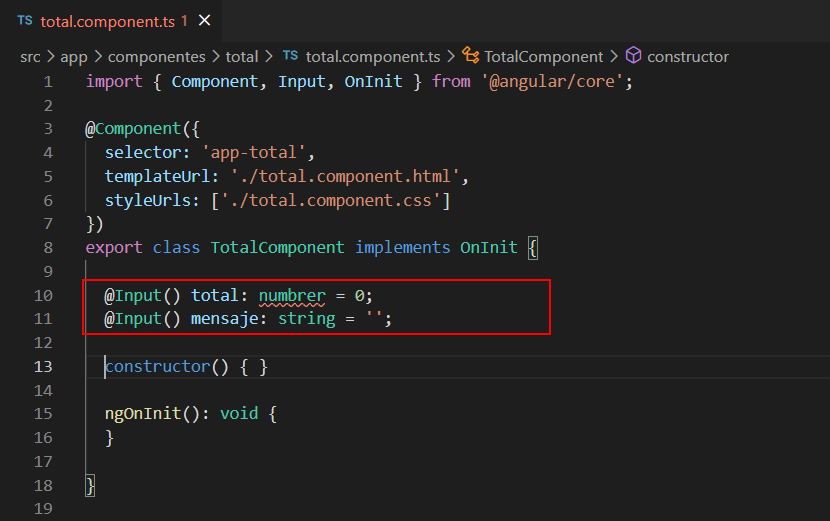
Con esto se ve que yo puedo determinar que estilo puedo aplicar a determinada clase CSS dependiendo de una expresión de condision que coloque en la definición de un objeto. (1:23:19)

### Comunicación entre componentes sin parentesco (servicios)

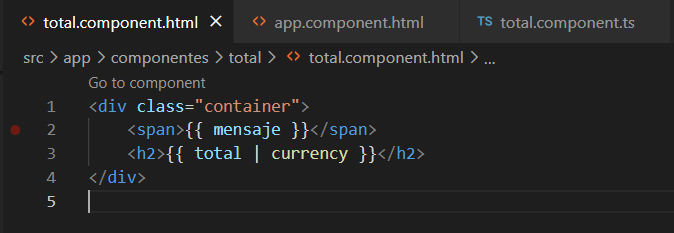
Asi como vinos como comunicar a través de parámetros de entrada o salida entre componentes padre e hijo. También se puede pasar entre componentes que no están relacionados de esa manera y esto se hace mediante el uso de servicios que ya veremos.

## El componente Total

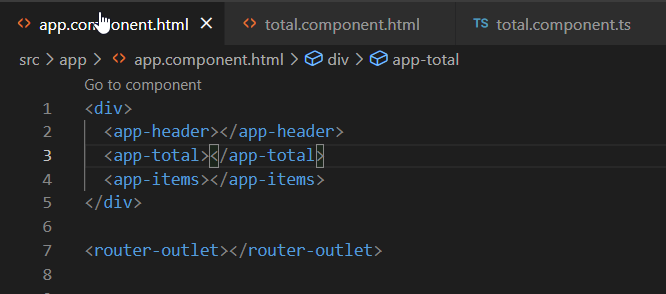
Vamos a crear un componente total que será el encargado de llevar el total de la compra. Vamos a definirle dos propiedades de entrada en el total.component.ts el total que llevara valor total de la compra y un mensaje

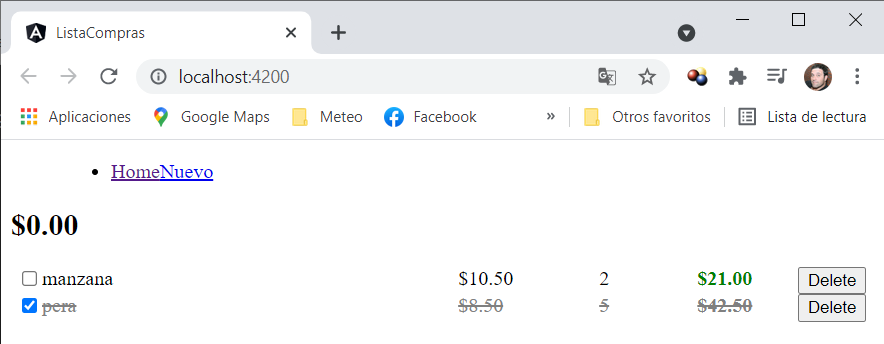


y en el total.component.html

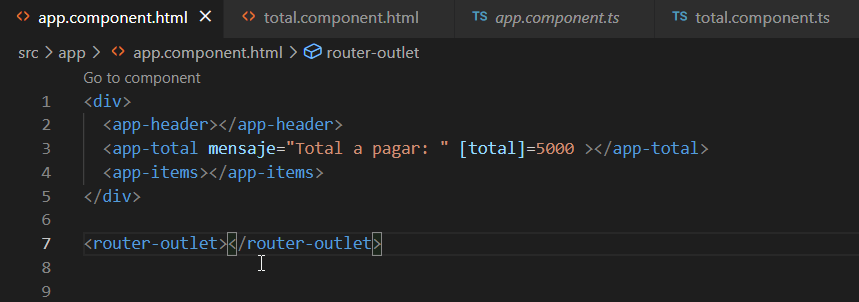


Agregamos la etiqueta <app-total></app-total> en app.component.html

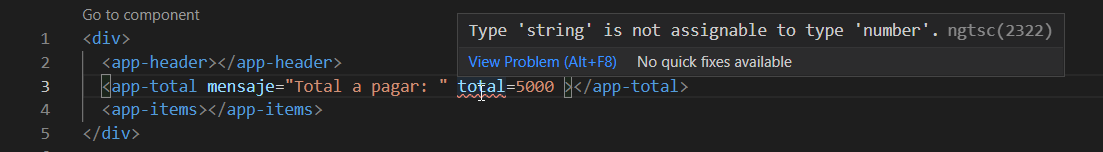




ahora vamos a pasarle valores desde el app.component.html

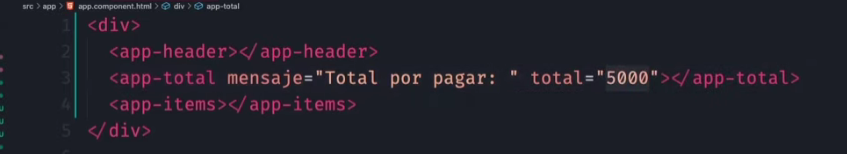


Aquí mensaje lo utilizaría como [mensaje] si fuese a pasarle el valor (dinámico) de una propiedad de la clase app.component (como por ejemplo title) pero en este caso le estoy pasando un string directamente. En cambio con la propiedad total, si la uso sin [], me da el error

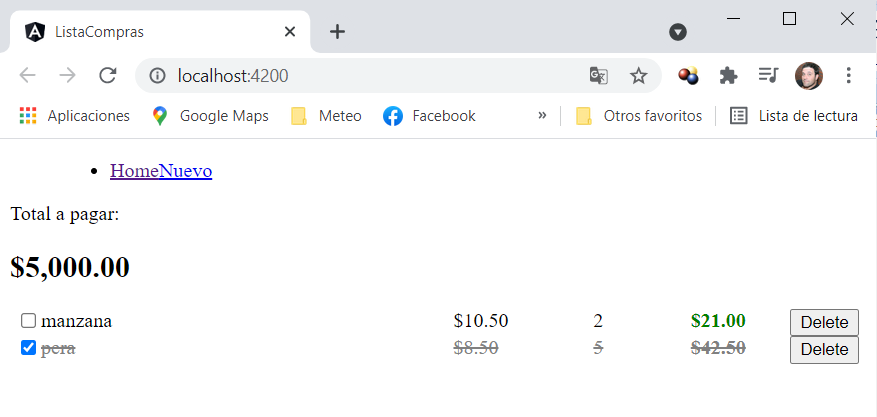


pero funciona si lo coloco como [total]

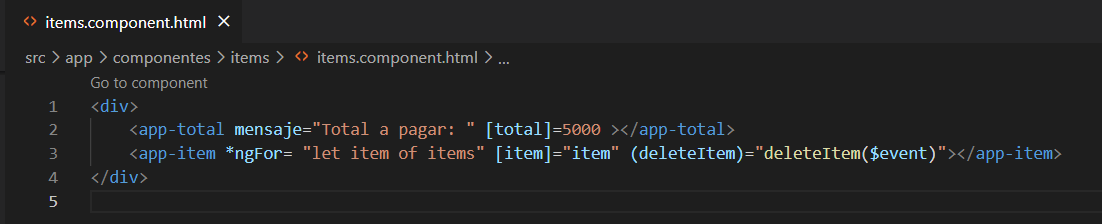
**Nota:** en el video, ambos parámetros los usa sin [] y los valores los pasa como string y no le dan error

****

parece funcionarle correctamente. En mi caso, ‘mensaje’ debo mencionarlo sin [] y a ‘total’ debo mencionarlo como [total] y el valor del total no pasarlo como string, sino como numero.



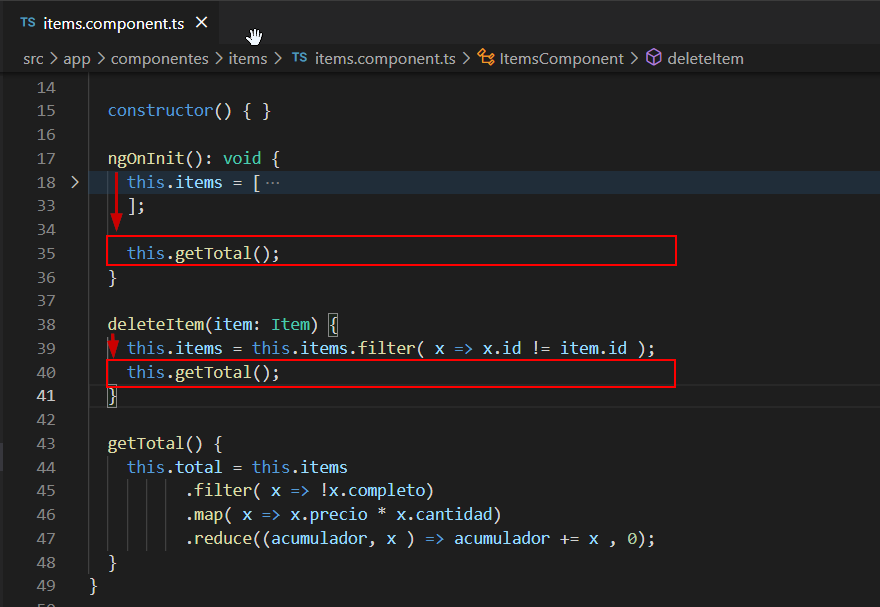
En realidad debemos sustituir el 5000 por el ‘total’ real de los datos, y la definición de ‘total’ estará en el componente ítems.component, por lo que convendría pasar la etiqueta de <app-total></app-total> a el html de este ultimo componente ítems.component, en vez de tenerla en el html del app.component



y el resultado será el mismo

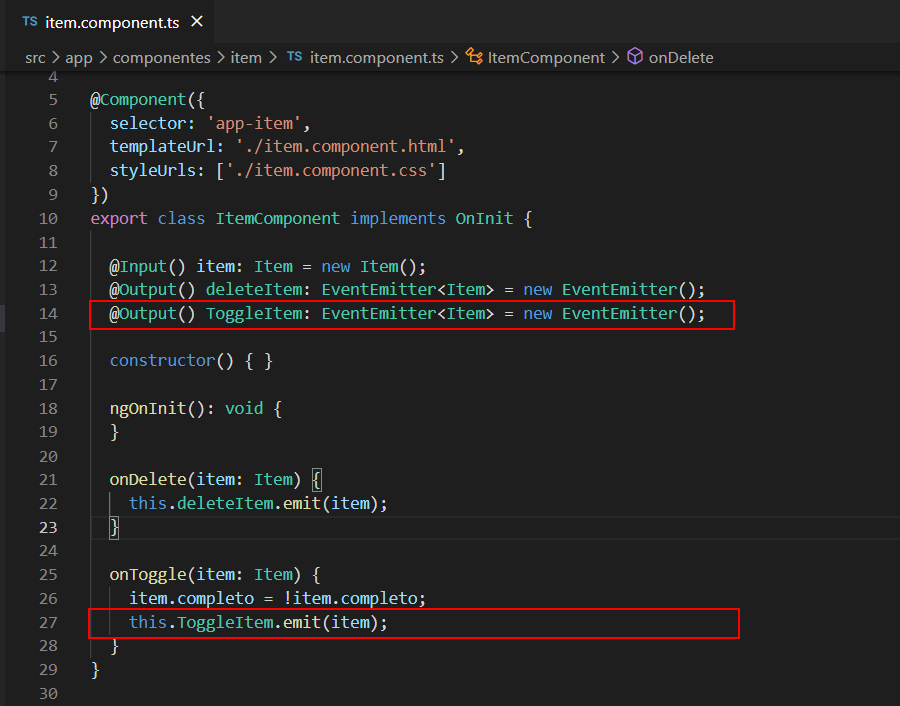
en el ítems.component.ts definimos la propiedad total de tipo number inicializada en 0.

Definimos además un método getTotal() que calcula el total de todos los ítems que poseen el estado completo en true y se lo asigna a la propiedad total. Hasta aquí todo bien pero el valor de total esta inicialmente en 0. En el método onInit() al final puedo llamar al método getTotal() y calculara el total, cuando el componente se inicializa, con todos aquellos ítems que inicialmente están chequeados en completado a true.

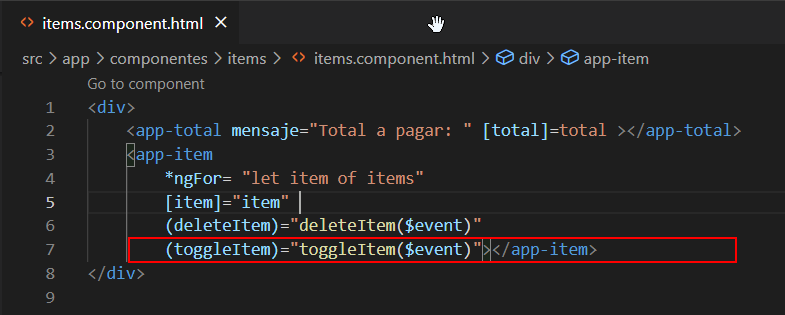


Pero tengo que tener en cuenta que si elimino un ítem determinado, es probable que el valor de total cambie, por lo cual tendría que llamar dentro del método deleteItem al método getTotal().

Por otro lado, cada vez que chequee o deschequee un ítem debo actualizar también el total y para esto tengo que llamar al método getTotal(), pero en el componente ítems.component no tengo un manejador para el evento checked del checkbox del componente ítem.component. Para solucionar esta parte tendre que enviar el evento desde el componente hijo al componente padre utilizando una nueva propiedad de salida @output en ítem.component para enviar el evento hacia el padre y definir un método

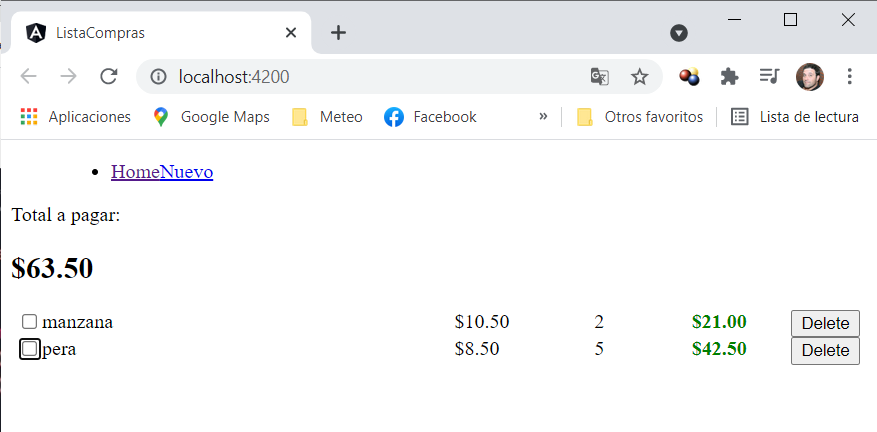


En el ítems.component.htm tengo que agregar la porpiedad toggleItem que es por la que recibo el evento



Y en el ítems.component.ts agrego el código para el evento toggleItem(ietm: Item) que llamara al método getTotal() y recalculara el total de la compra según el estado de los ítems del arreglo.

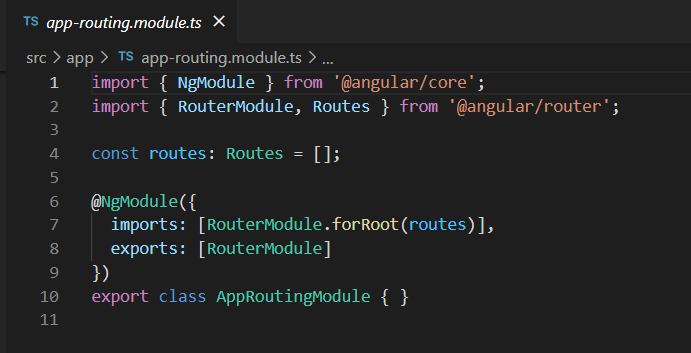
Con esto se completa por el momento el calculo dinamico del total de la compra.



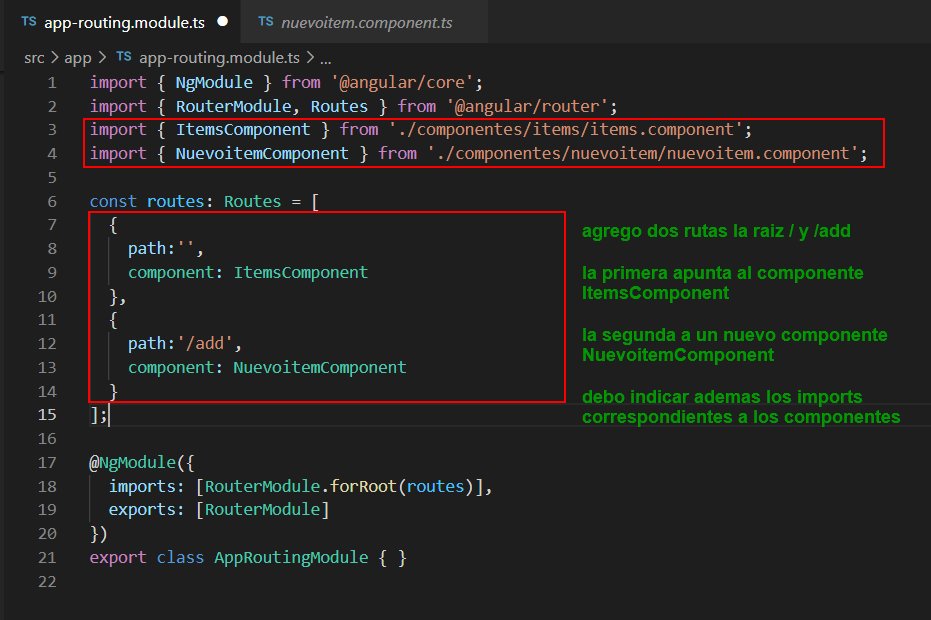
## Agregar nuevo Item (noción de ruteo)

Vamos a implementar la vista de agregar un Nuevo ítem, podríamos crear la vista allí mismo en la pagina que venimos viendo pero vamos a ver un poco el detalle de lo que es el Ruteo, para ver lo fácil que es crear rutas y cuando demos click en un ítem de la barra de navegación nos lleve a una nueva vista que en realidad es el mismo documento.

En principio vamos a ubicar el archivo de Ruteo de la aplicación que es el app-routing.module.ts y que pedimos se creara cuando creamos la aplicación. En este archivo es donde vamos a especificar nuestras rutas

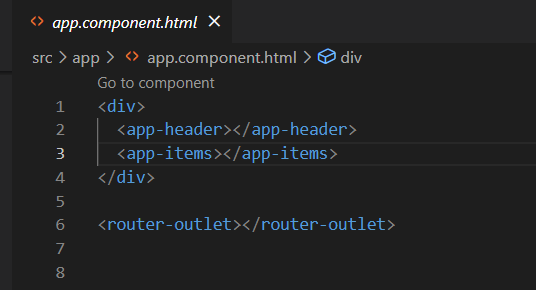


En principio lo vemos como lo muestra la figura anterior, pero dentro del arreglo routes se van a ir agregando las rutas de la siguiente manera

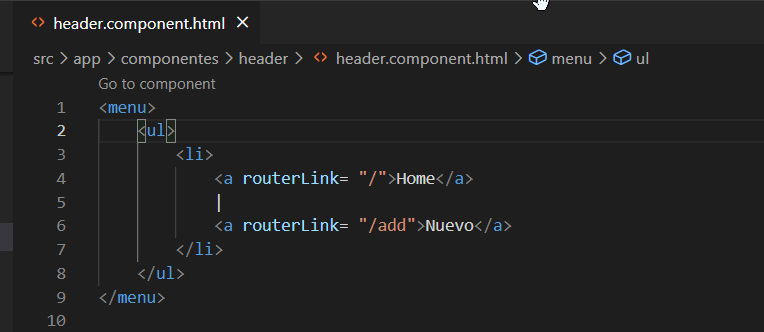


El componente itemsComponent es sobre el que estuve trabajando hasta ahora, el componente NuevoitemComponent es un componente que acabo de crear para tener la funcionalidad de agregar nuevos ítems a la lista de compras.

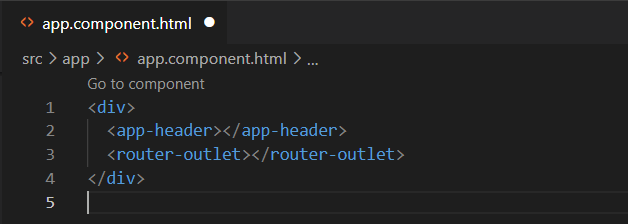
Volvamos al app-component.html allí donde tenemos las etiquetas de los componentes que muestra la aplicación inicialmente. En el archivo vemos en un bloque <div></div> la del componente header.component que es la que contiene la barra de navegación y además la del componente ítems.component que es la que muestra la ista de ítems de la compra.



y fuera del bloque <div></div> el etiqueta <router-outlet></router-outlet> que hasta ahora no tenia sentido porque no teníamos definida ninguna ruta, pero ahora que ya las definimos y teniendo en cuenta que en componente header.component ya habíamos definido algo como una barra de navegación donde a cada elemento lo definíamos con una etiqueta



Ahora las referencias a rutas van a funcionar pero debo modificar el app.component.html de la siguiente manera, pues inicialmente le indicaba la etiqueta <app-items></app-items> para que me mostrara el componente, ahora como están definidas las rutas y el home es la raíz ‘\’ la mostrara automáticamente (si lo dejara como antes mostratia al componente constanetemente).

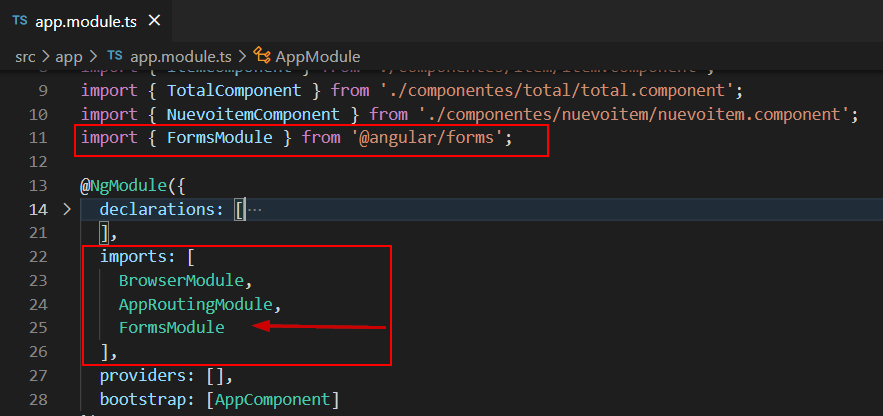


De esta manera el navegador <app-header></app-header> quedara fijo en la aplicación en cambio lo demás que va por rutas en el <router-outlet></router-outlet> cambiará dinámicamente.

### Componente Nuevoitem

En Nuevoitem.component.html vamos a definir un form en el que vamos a cargar los datos de cada ítem nuevo

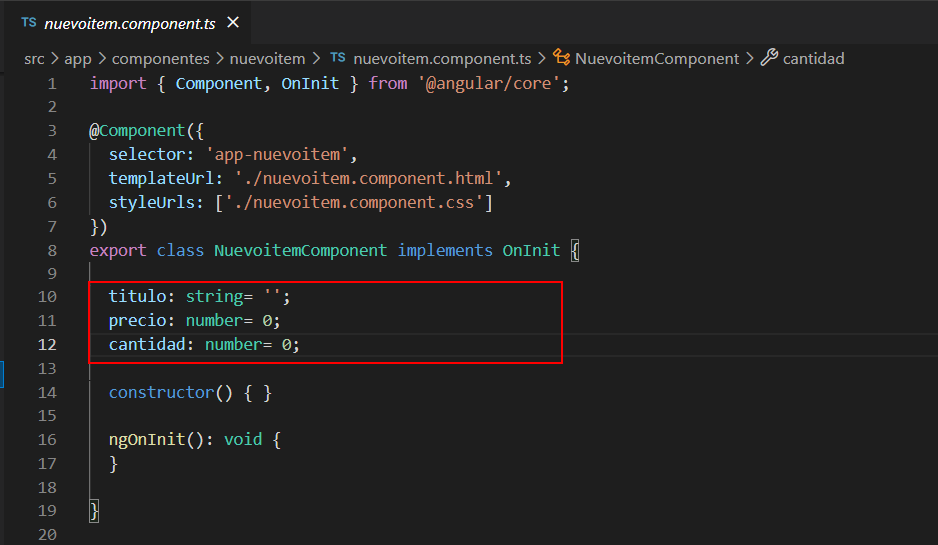
Como vamos a trabajar con formularios será necesario agregar en el bloque imports de app.module.ts el componente FormsModule de @angular/forms



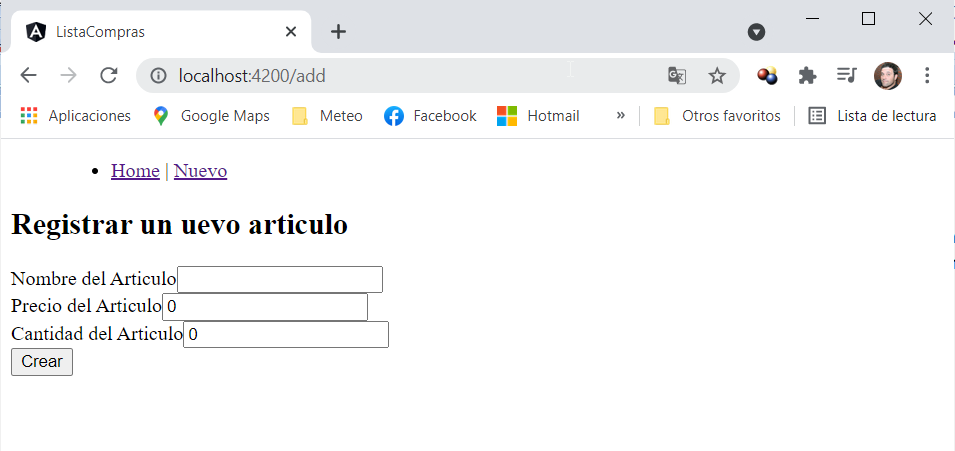
por otro lado en el Nuevoitem.component.html definimos el formulario en bloque de etiquetas <form></form> de la siguiente manera



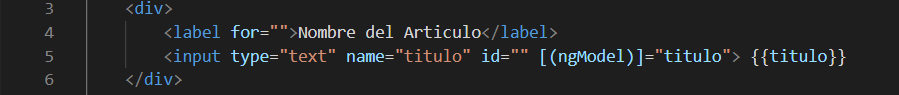
y en el Nuevoitem.component.ts definimos las propiedades o variables que estamos haciendo un binding en en cada input del formulario con una variable. Menciono vinculación doble porque cuando las edite en el formulario se deberán reflejar dichos cambios en las variables del componente que por otro lado deberán estar definidas como propiedades de la clase del componente Nuevoitem, de la siguiente manera

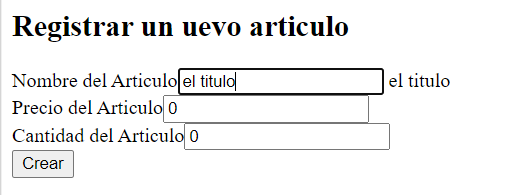


y esto ya nos iría mostrando algo asi

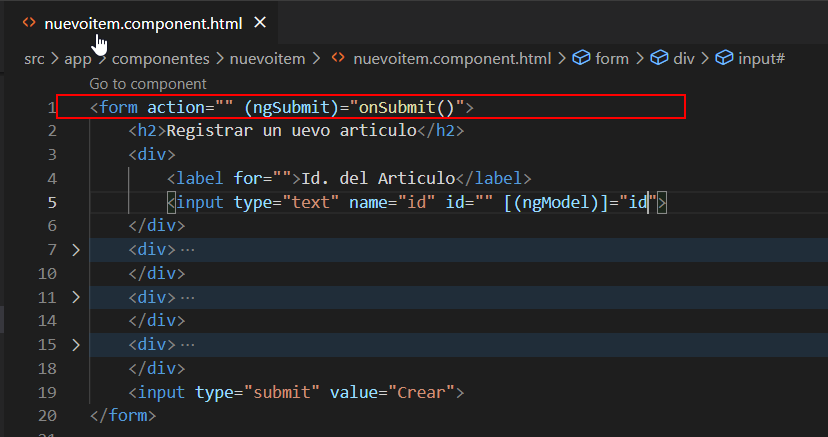


si quisiera comprobar que el binding de las variables funciona debiera de agregar como prueba a cada uno de los ítems algo asi

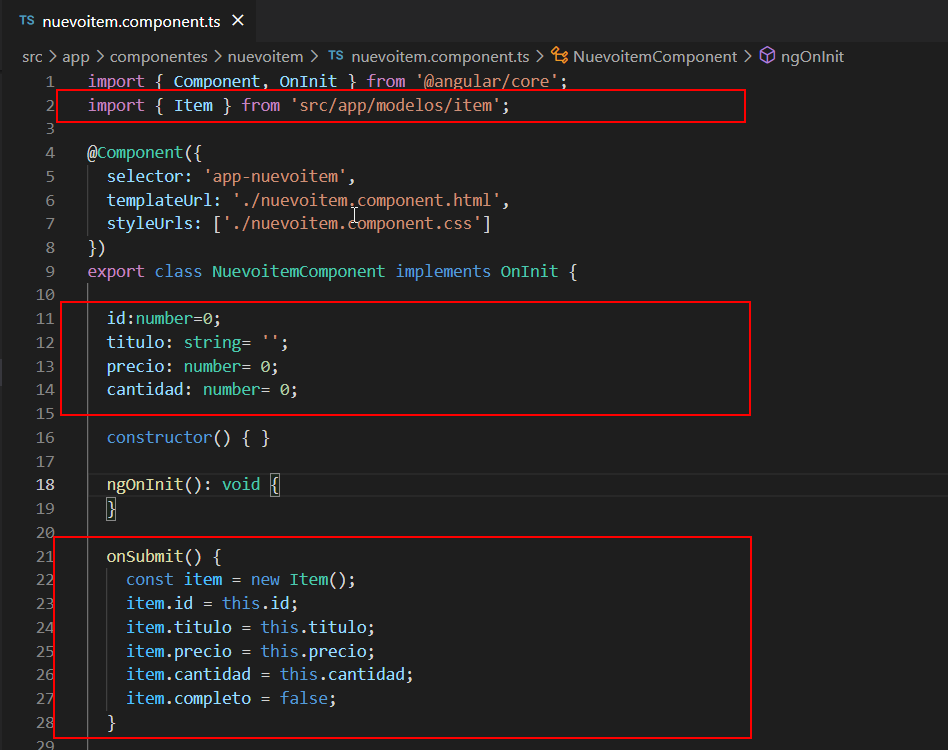




En este momento debiéramos de ver como agregar el nuevo ítem a los ítems de la compra. No vamos a grabar el ítem cuando se apriete el botón Crear, sino cuando se envie el formulario, es decir cuando se ejecute el evento submit()… para el formulario en Angular hay un evento ngSubmit que es el que vamos a utilizar en el formulario de la siguiente manera



Además agregue un input para el id de articulo que no lo había tenido en cuenta que también debo definir dentro de la clase del componente Nuevoitem además de agregar el código para el método onSubmit() que se encargara de manejar el evento ngSubmit del formulario Angular



Ya tenemos el objeto ítem cargado en el componente, el problema que se presenta aquí es que los ítems se manejan de manera centralizada en el componente ítems.component que no guarda ninguna relación con este componente Nuevoitem.component, no hay relación padre hijo para poder pasar el evento como una propiedad de salida como lo veníamos haciendo anteriormente en otros componentes. En este caso nos conectamos por rutas y el problema lo vamos a solucionar utilizando un servicio que nos permita acceder a la información desde cualquier parte de la aplicación web.

Anteriormente ya vimos los servicios para utilizar servicios html get, put, post y delete cuando trabajábamos con CRUD con datos de un servidor (backend).

Crear y utilizar un servicio para la comunicación entre componentes

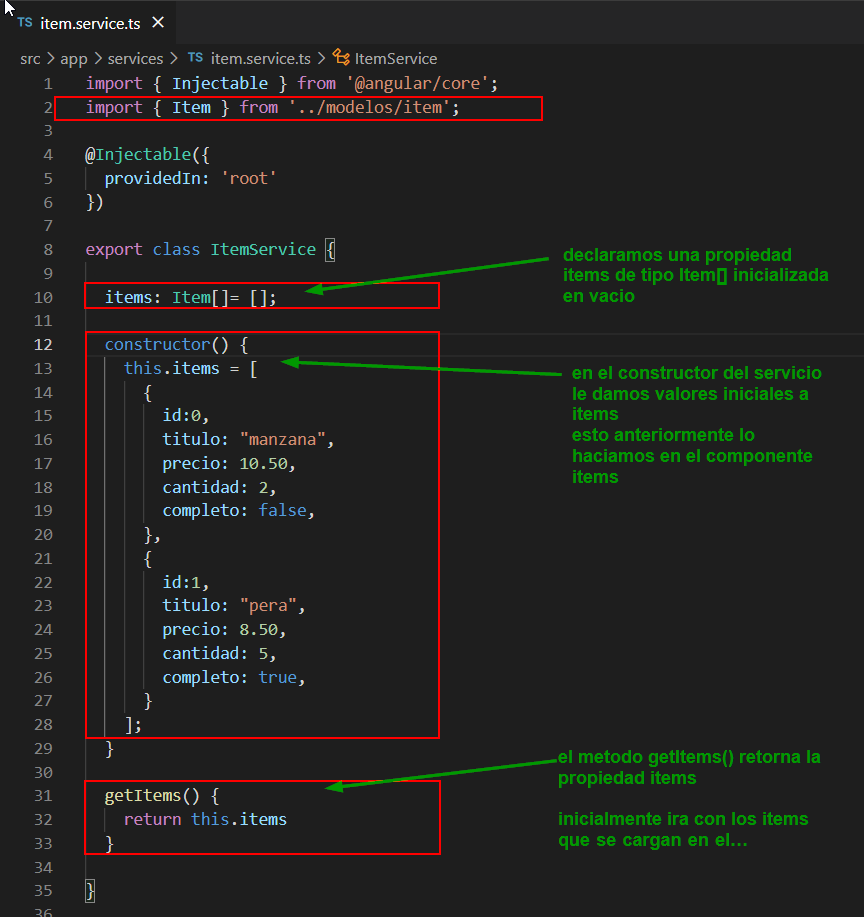
Para crear un servicio vamos a tener que apelar al comando de angular

**ng generate service services/item**

esto generara dos archivos en una carpeta services, uno para testing (por ahora no tiene relevancia) y el otro propio del servicio item.service.ts.

Un servicio en Angular nos sirve para acceder a información de forma general, sin importar si la información está en un componente u otro. Entonces lo que vamos a intentar aquí es hacer que la propiedad ítems (el arreglo de ítems de la compra) del componente ItemsComponent pueda ser accedido desde cualquier otro componente.

Cuando en su momento creamos el componente ItemsComponent, en el método ngOnInit() nos encargamos de darle unos valores iniciales al arreglo items[]. Esto ahora lo vamos a cambiar, es decir, los valores iniciales al arreglo se los vamos a dar a través de un servicio. En el servicio ItemService que creamos, definimos un método getItems() que retorna el arreglo inicializado (el que antes usábamos para inicializar directamente el componente ItemsComponent). Para esto ultimo creamos dentro del servicio una propiedad items: Item[] inicializada a vacio, en el constructor le insertamos el arreglo con los artículos iniciales y luego el método getItems() se encarga de retornar la propiedad items (en el inicio ira con los valores cargados en el constructor.



ya tenemos entonces nuestros datos que deberían ir a la propiedad items[] del componente ItemsComponent.

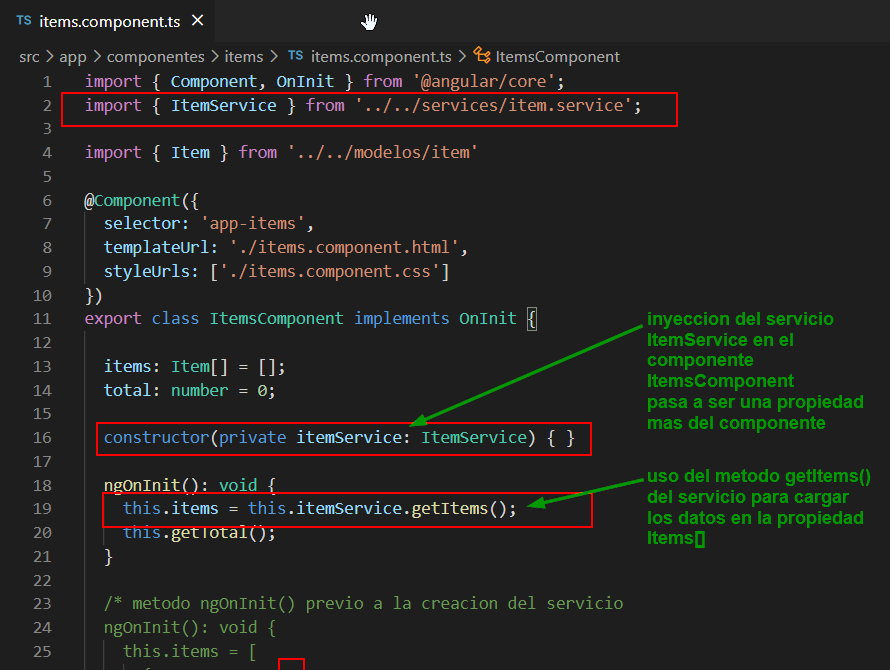
Ahora vamos a ver cómo acceder a ellos desde los componentes, en particular desde ItemsComponent que es el que los necesita para inicializar la propiedad. Es decir, como nos hacemos de ellos en el método ngOnInit() de ItemsComponent, de donde los quitamos.

Entonces, en el componente items.component en el momento que lo inicializamos en ngOnInit(), lo que queremos es que items[] se cargue, entonces allí debiéramos llamar al método getItems() del servicio… ¿pero como lo hacemos?

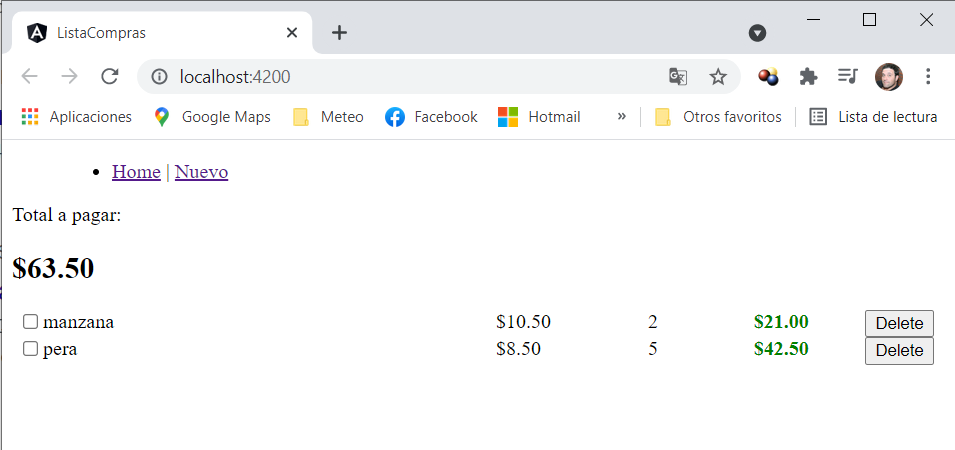
Lo que vamos a necesitar es una propiedad del tipo ItemsService dentro del componente ItemsComponent y esta la agregamos **inyectándola** en el constructor de la clase del componente de la siguiente manera

**constructor(private itemService: ItemService) { }**

para esto debimos importar ItemService desde ../../services/itemservive.ts (esto lo hace automáticamente cuando lo utilizamos en el código) y entonces ahora podemos llamar a getItem() en ngOnInit().

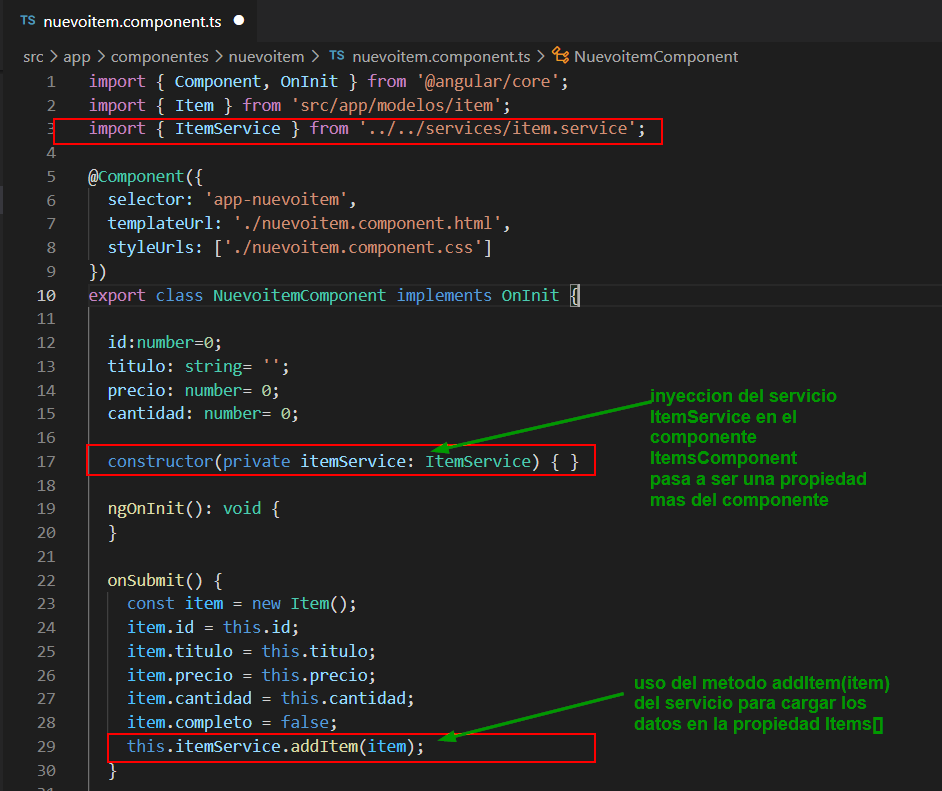


por lo menos hasta aquí verificamos que la aplicación funciona de la misma manera que lo hacia anteriormente



Vamos ahora a ver como tratamos el caso de adicionar un nuevo ítem, que era por donde habíamos comenzado a ver la opción de utilizar servicios.

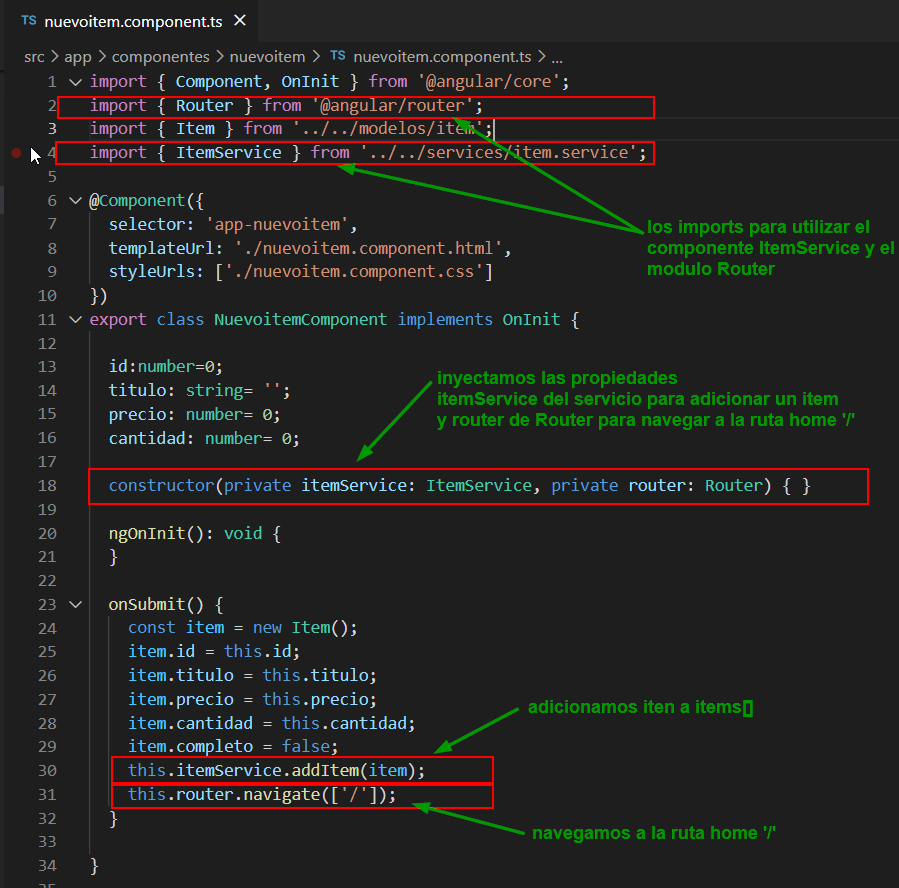
En principio lo que vamos a tener que hacer es definir un método addItem(item) en el servicio ItemService y cuando haga el submit() en el formulario del componente NuevoitemComponent tendré que llamar a éste método desde el servicio ItemService. Debemos inyectar itemService al componente NuevoitemComponent para que lo agregue como una propiedad del componente y poder utilizarlo. Esto ultimo se hace desde el constructor del componente

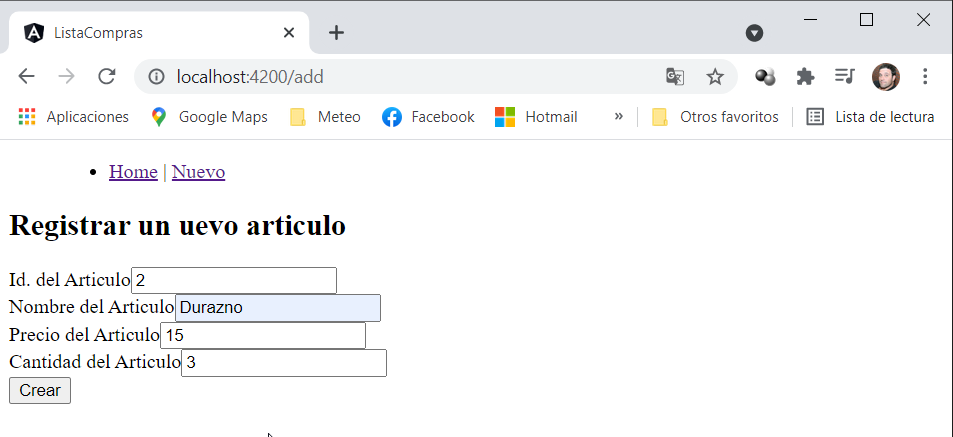


el método addItem(ietm) del servicio lo que debiera de hacer es insertar el nuevo item al principio del arreglo items[] (propiedad de ItemService), para esto usamos el método unshift(x) de los arrays.

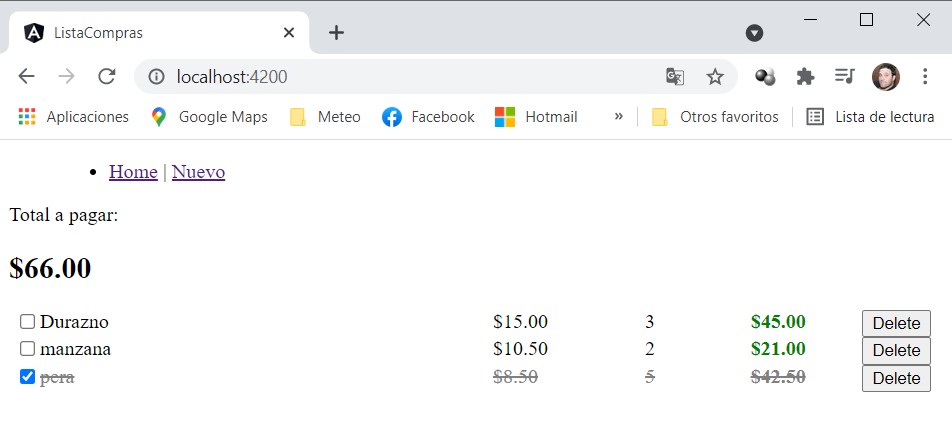
En el componente NuevoitemComponent en el evento submit() habíamos llegado hasta que cargar los datos de las entradas del formulario en un objeto Item, pero no podíamos pasarlo al componente ItemsComponent como un evento para cargarlo en su propiedad items[]. Para lo cual mencionamos utilizar servicios y creamos ItemService, la idea entonces es inyectar una propiedad itemService al componente NuevoitemComponent a través de su constructor u desde esta propiedad cargar el ítem utilizando su método addItem(Item).

Pero además vamos a hacer que el submit al finalizar navegue a home ‘\’ y la aplicación quede mostrando la lista de items. Esto ultimo lo podemos hacer utilizando el método navégate de un objeto Router y para poder utilizar el método necescitamos una propiedad router en el componente NuevoItemComponent que agregamos inyectándola desde el constructor como lo hicimos con ItemService





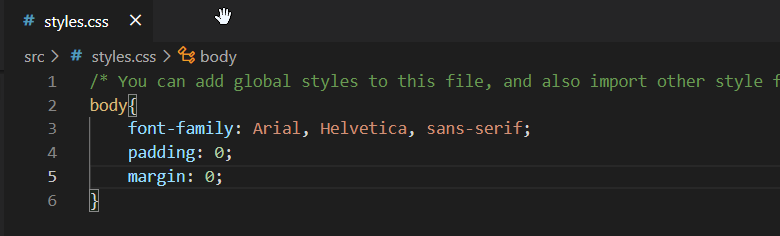
al dar Crear en el formulario, se ejecuta el método submit carga el ítem a items y navega a la pagina Home y me muestra el nuevo artículo en los items de la lista de compras. Esto ultimo es debido a que se ejecuta el evento ngOnInit() del componente ItemsComponent y se ste llama a getItems() del servicio, lo que hace que items[] del ItemsComponent se actualiza con items[] de itemService[].



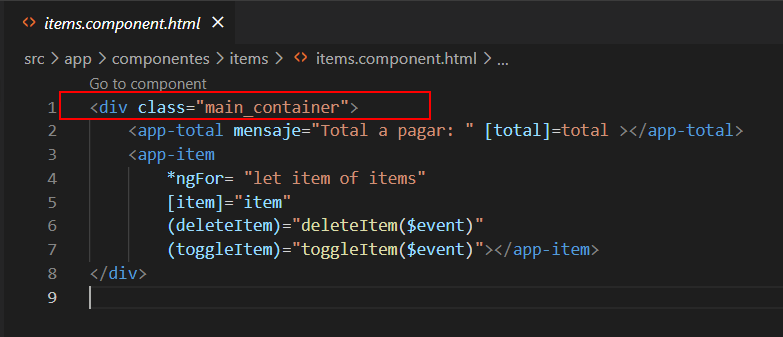
## Estilos a nuestra aplicación

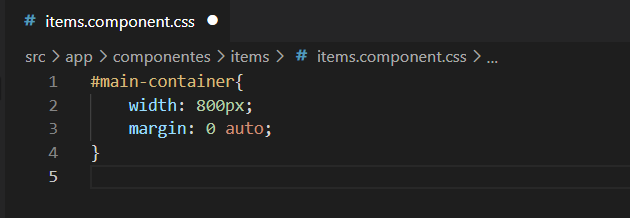
Con lo anterior a vemos funcionando la pagina aunque no con datos porvenientes de un servidor. Vamos ahora a darle un poco de estilos a los componentes que no lo poseen

Arrancamos en el styles.css para dar un estilo global a los fonts

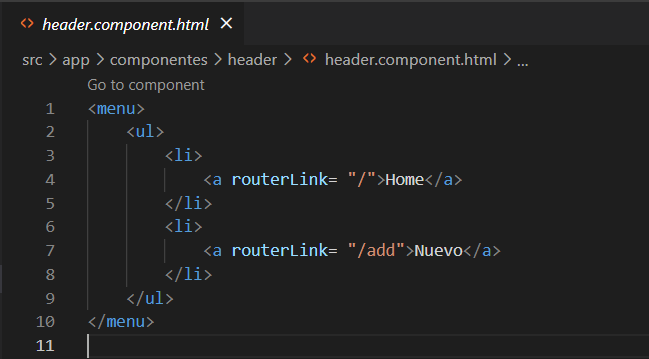


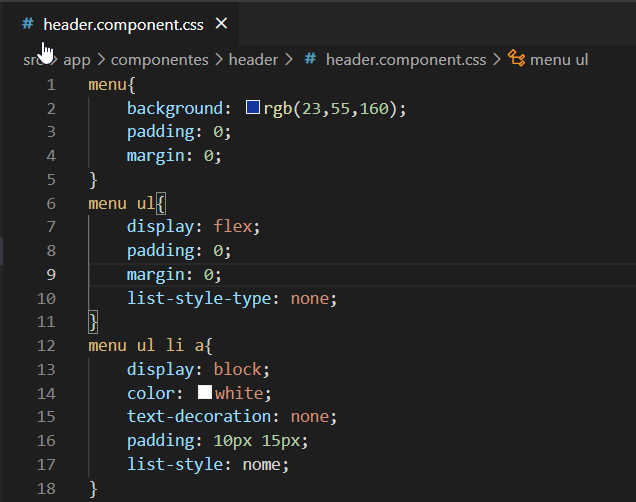
Luego comenzamos por el componente ItemsComponent al bloque <div> principal le agregamos class=”main-container” y en el archivo de estilos .css del componente



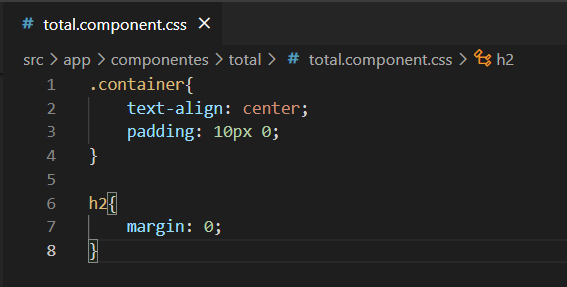


Para el componente HeaderComponent

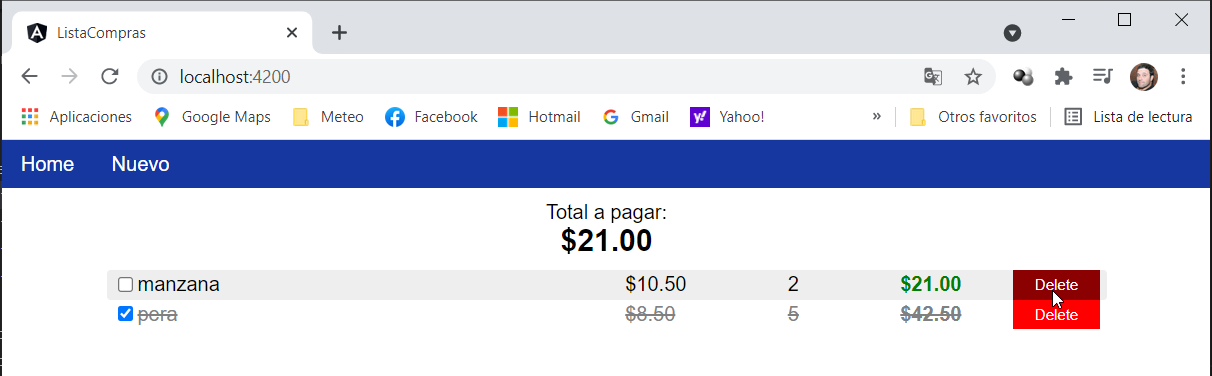




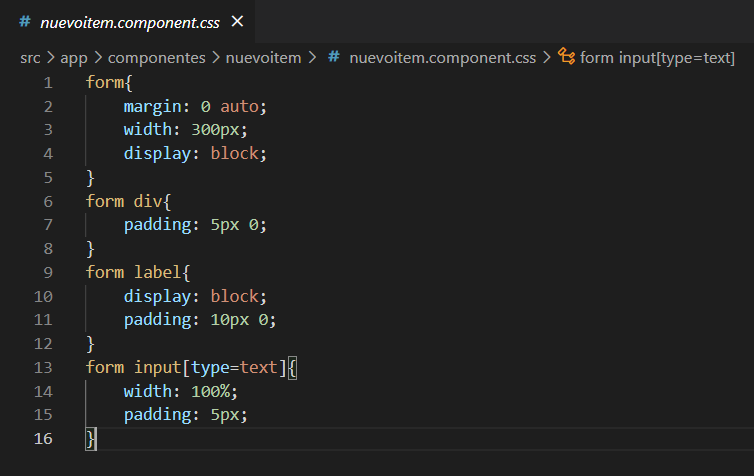
El componente TotalComponent podemos centrarlo, habíamos definifo una class=”container”



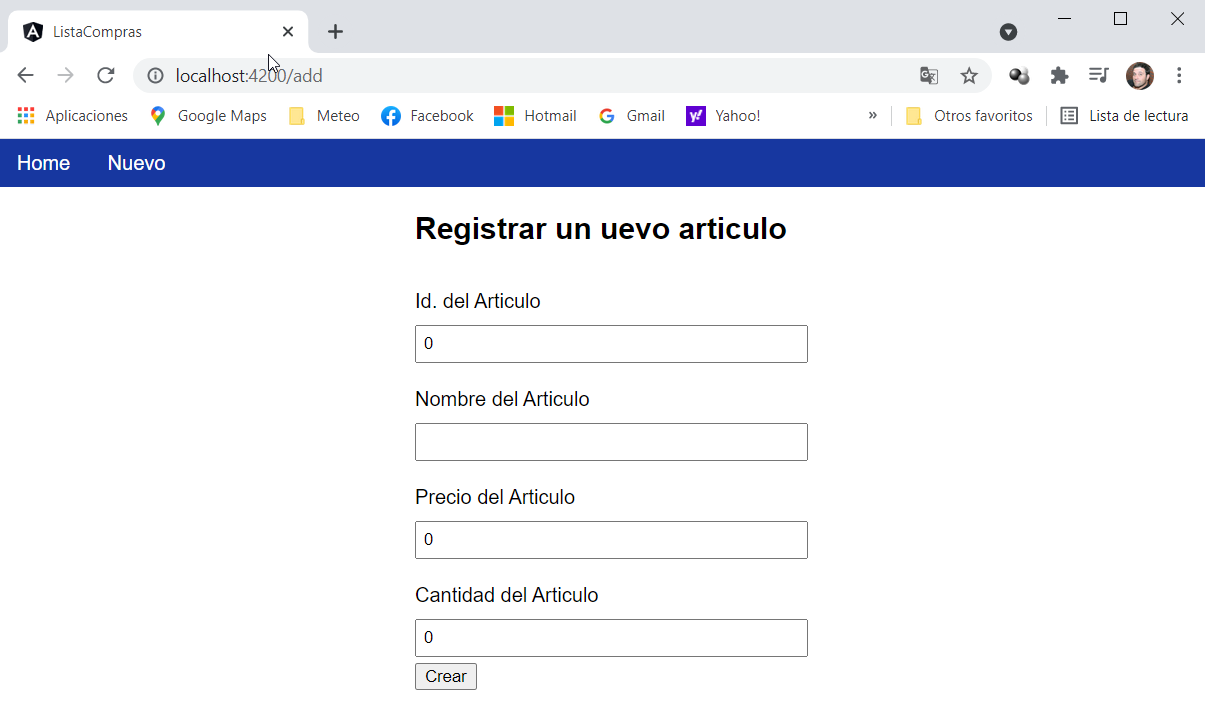
El componente ItemComponent podemos centrarlo, agregar un grisado a la fila sobre la que se encuentre el cursor del mouse y darle color al botón de eliminar. Finalmente lograríamos que se viera de la siguiente manera



El componente NuevoitemComponent tendrá el siguiente estilo



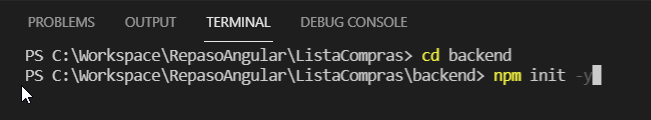
Y se mostrara de esta manera



## Solicitudes HTTP a servidor de datos externo Backend

Para poder trabajar con solicitudes http y mantener los datos de la lista de compras fuera del Frontend vamos a utilizar un servidor HTTP sencillo como Backend (server.js) y un archivo (data.json) con los datos de los articulos de la lista de compras.

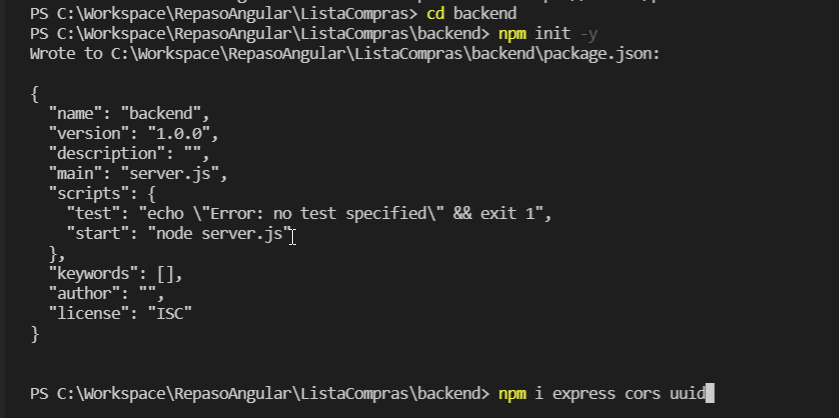
Dentro de nuestro poyecto vamos a crear una carpeta \Backend donde copiamos estos dos archivos y desde VS Code con Node.js vamos a crear un proyecto con el archivo .js, ejecutando desde una consola ubicados en la carpeta \Backend el siguiente comando

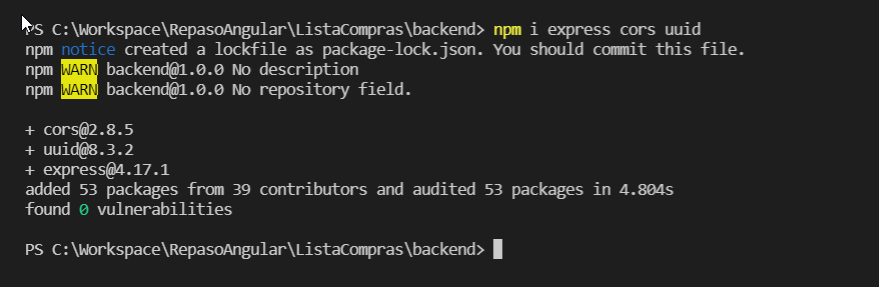


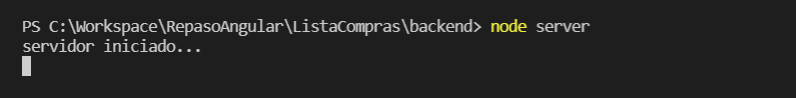
Este nos creara el proyecto para el servidor un archivo package.json que nos permiten compilar el proyecto y levantar el servidor. Por otro lado será necesario para compilar instalar las librerías que el servidor utiliza, estas son express, cors y uuid. Desde la teminal ejecutamos un comando npm install para instalar las tres librerías mendionadas.

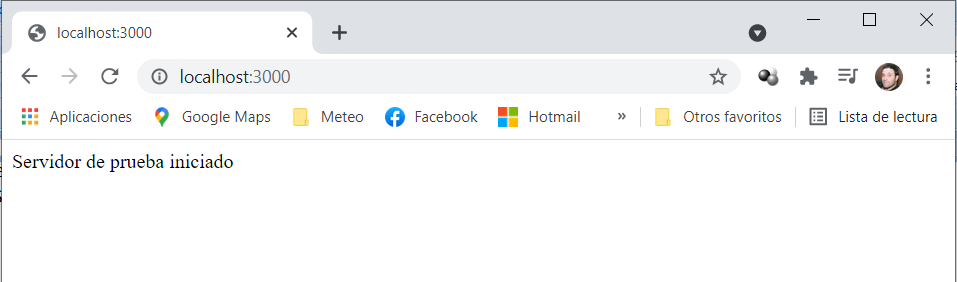
Una vez instaladas las librerías ya podemos compilar el servidor usando comando node serve y de no mediar inconvenientes por consola veremeos el mensaje que el servidor esta levantado.

Si vamos a un navegador u escribimos <http://localhost:3000> debieramos de ver un mensaje que indica que el servidor de prueba esta iniciado









Con esto ya estamos listos para hacer los cambios para que la aplicación se sirva de los datos desde el backend, mas que definirlos dentro de ella.

Vamos a hacer cambios en algunos archivos de manera que para conservear la versión hasta este punto, dejarremos una versión renombrada de los archivos que se miodifiquen

items.service.ts items.service.ts.old

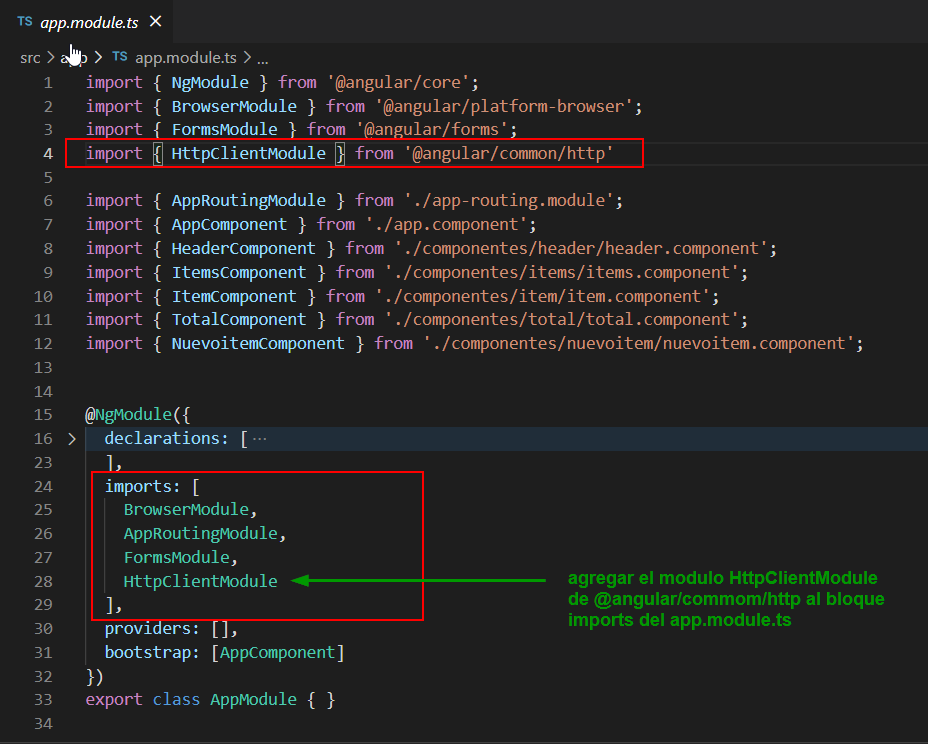
app.module.ts app.module.ts.old

nuevoitem.compnent.ts nuevoitem.compnent.ts.old

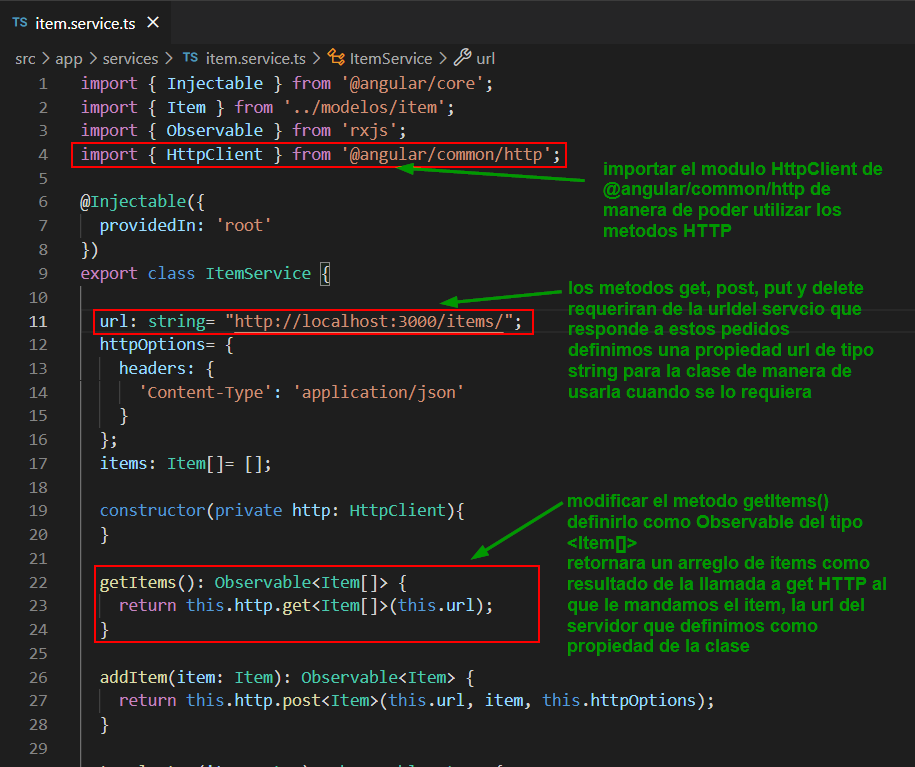
Entonces en el servicio ItemService que es donde estábamos centralizando la información. Alli el método getItem() retornaba la información, que estaba definida ahí mismo, ahora vamos a cambiar las cosas para que comience a requerirla al Backend. Tenemos que transformar estos métodos en funciones Observables (rxjs), manejo de flujo de eventos. Los eventos, como las solicitudes HTTP, entran en este concepto.

Para transformar estos métodos, entonces le decimos que ahora va a retornar un objeto Observable, en el caso de getItems que retornará un Observable del tipo arreglo de Items, es decir, indicar por el tipo de objetos por el cual está observando. Vamos a utilizar el HttpClientModule para el tema de los eventos HTTP y esto requiere:

* definir dicho modulo en el bloque Imports del app.module.ts e importarlo desde @angular/common/http

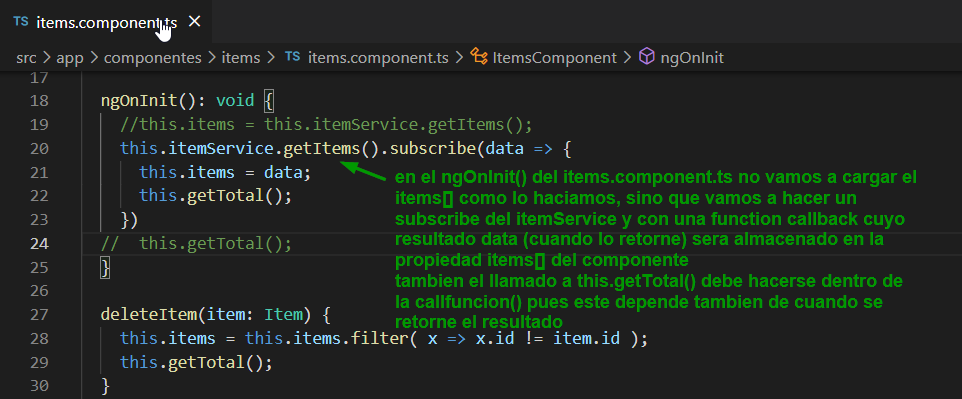


* modificar getItem() del servicio ItemService para observar el flujo http que vendrá del backend ante una solicitud

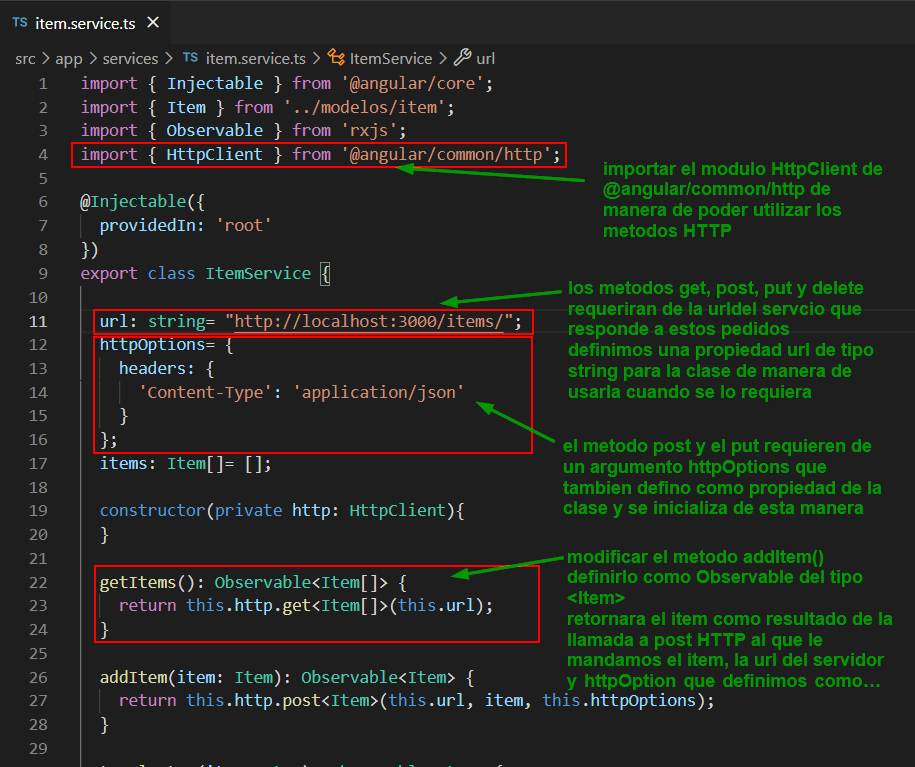


* el getItem() requerirá se le especifique una url desde la cual hará el método get de HTTP y para esto definimos una propiedad nueva url en la clase del servicio de tipo string inicializada con la ruta ‘http://localhost:3000/items”
* en items.componet.ts será el lugar donde me subscribiré al observable anterior para recibir esta información. En este subscribimos al observable el getItems() del itemService y pasamos como argumento una función callback que nos responderá con ‘data’ que lo asignaremos a la propiedad Items[] del componente items.

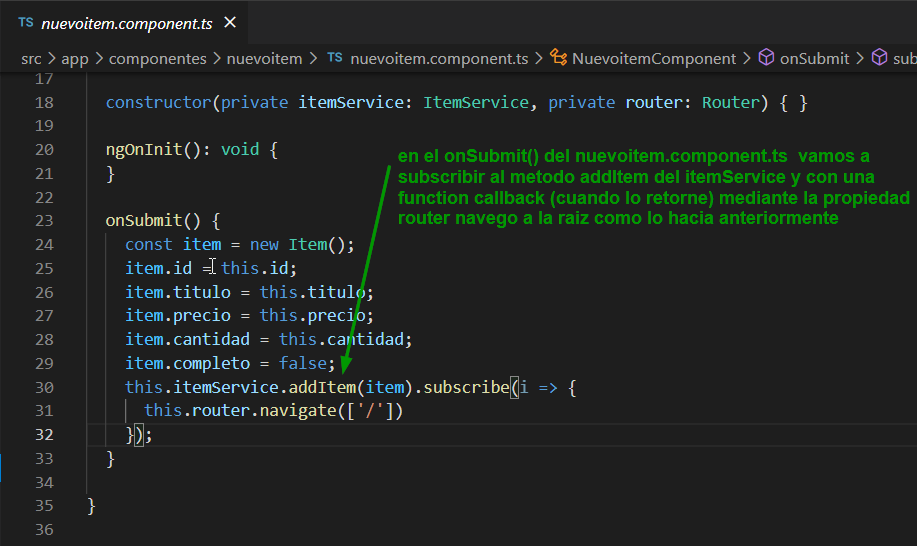
El problema que surge aquí es que la llamada a this.getTotal() seguro se ejecutara antes de la respuesta de la función callback entonces nos nos va a represntar el total. Va a haber que buscarle la vuelta



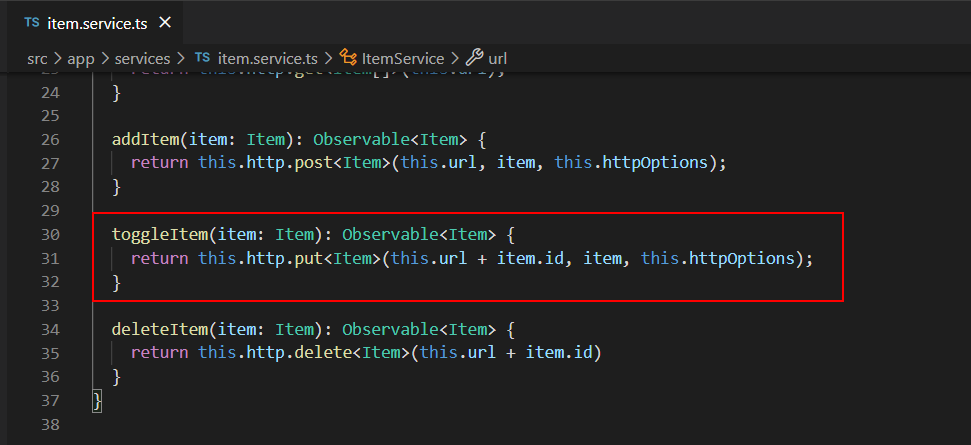
* Asi como se modificó getItems() en el servicio también hay que modificar addItem(ítem), de manera que también sea un observable y en este caso llamara al método post de HTTP habrá que pasarle además del ítem, la url del servicio y además requier de un paramentro httpOptions que definimos como propiedad en la clase del servicio y le damos el valor inicial que corresponde (se muesta en la siguiente imagen)



* En el componente NuevoitemComponent hay que hacer modificaciones a la llamada del método addItem() de manera que se subscriba al observable.Subscribe llama a una callback que cuando retorna indica navegar a la raíz.



* La actualización de un elemento en esta aplicación solo sucede sobre el valor de ‘completo’ de un Item, de acuerdo a si se marca en la casilla o se desmarca. Esta actualización está implemnetada en el componente ItemsComponenent, con el método toggleItem(ítem). Ahora, que utilizamos un servicio ItemService vamos a utilizar el método put de HTTP para actualizar un Item. Entonces va a requeriré definir un método toggleItem(ítem) en el servicio ItemService, este metodo será Observable y llamará al método put HTTP. El put requerirá se le pase la url del servicio, el id del ítem que se va a cambiar, el ítem modificado (para reemplazar) y el argumento httpOptions. Los argumento url u httpOption están definidos como propiedadesde la clase para utilizarlos

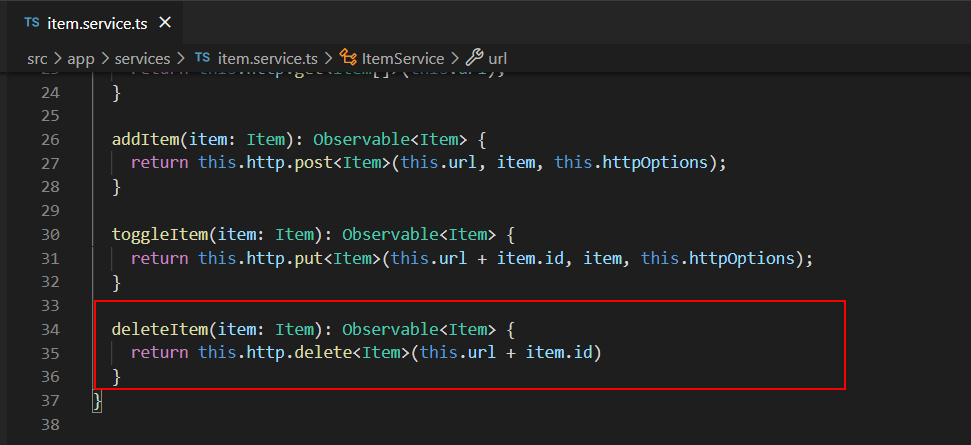


* Por otro lado en el componente ItemComponent debemos modificar el evento toggleItem para que suscriba al observable toggleItem del servicio ItemService

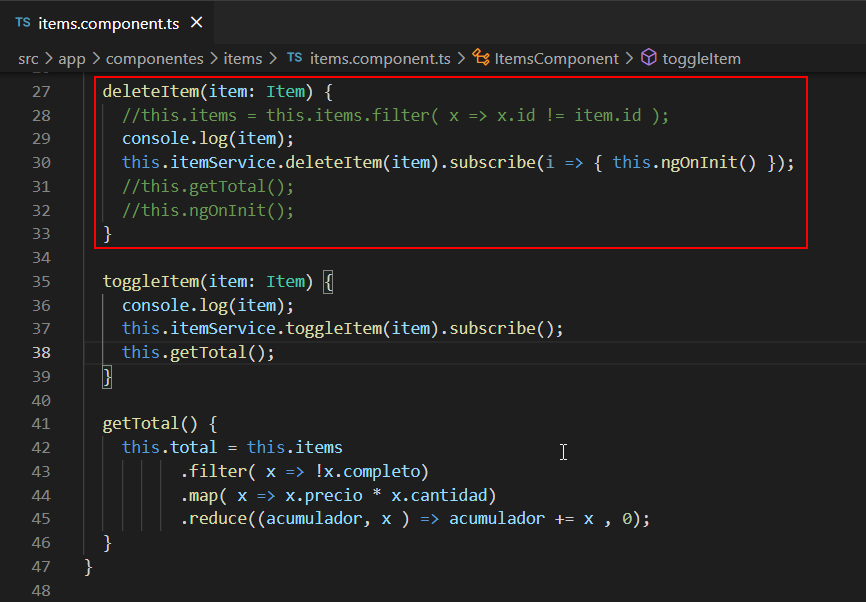


vamos a llamar al toggleItem(item) del this.itemservice haciendo que se subscriba. En este caso no requero utilizar una callback, aunque seria lo correcto llamar a this.getTotal() en una callback dentro del subscribe, porque como esta esrito getTotal() puede que se ejecute antes de que se lleve a cabo el cambio en el servidor… aunque en este caso mayormente no afecte.

* Queda el delete que hacemos del Item como resultado de darle al botón eliminar. En este caso desde el método deleteItem(ítem) del componente ItemsComponent vamos a llamar al deleteItem del servicio (cambia de cómo lo haciaos antes. Entonces es necesario definir un método deleteItem(item) Observable en el servicio ItemService que llamara al método delete HTTP del servidor. El delete requerirá se le pase la url del servicio y el id del ítem que se va a cambiar, con eso alcanza. el argumento url estn definido como propiedad la clase para utilizarlo



* Por otro lado en el componente ItemComponent debemos modificar el evento deleteItem para que suscriba al observable deleteItem del servicio ItemService



vamos a llamar al deleteItem(item) del this.itemservice haciendo que se subscriba. En este caso voy a utilizar una callback, que simplemente llamaría a this.ngOnInit() de manera que cuando el servidor responda la función callbak se ejecute ngOnInit, refljando los cambios en la pagina y al incluir esta ultima el getTotal() estaría todo resuelto.

En el video no lo hace asi, llama luego del subscribe() a this.getTotal() pero asi no me funciona, primero porque no me actualiza la pagina y además porque puede que calcule el total antes de que responda el servicio.