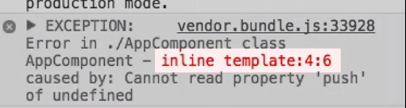
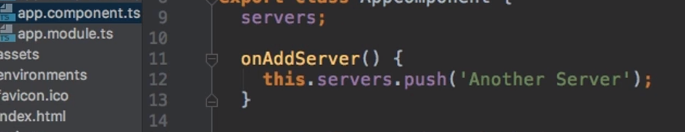
**Debugging:**

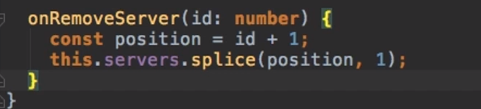
browser F12 – konzol developer tools



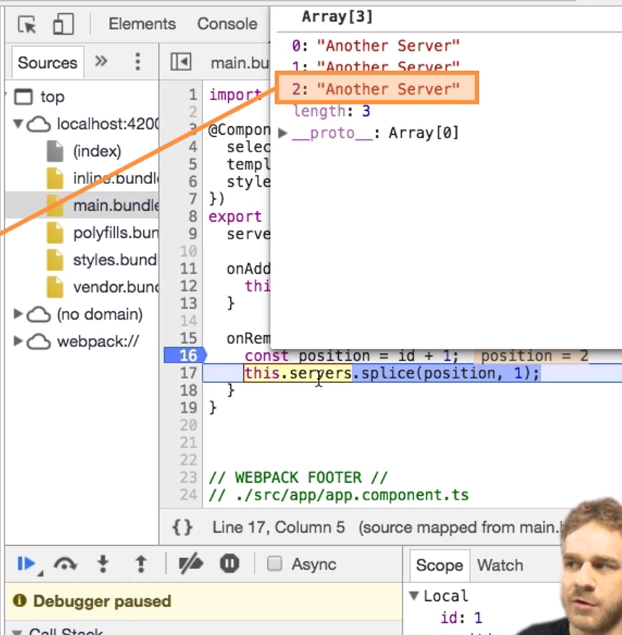




deklaráltuk a tömb változót, de típust nem adtunk neki ezért nem tudja belepusholni a gomb az értéket



ha itemre kattintunk törli

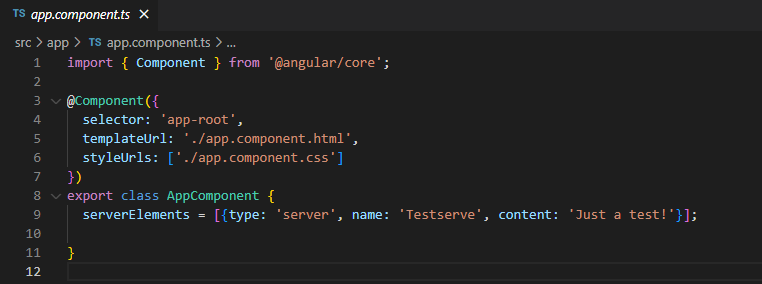


alul lehet a nyilakkal lépkedni, debugolni

**Components, Databinding:**



cli komponens készítése nem működött



teszt adatok itt vannak



komponens megjelenítése, attributumos kötéssel kapja meg az adatokat a másik komponens

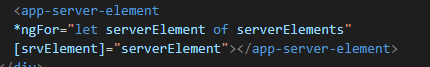


itt az input-tal fogja felnyalni és itt az osztály struktúra

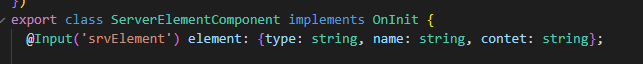


itt a komponens html-je

Alias:



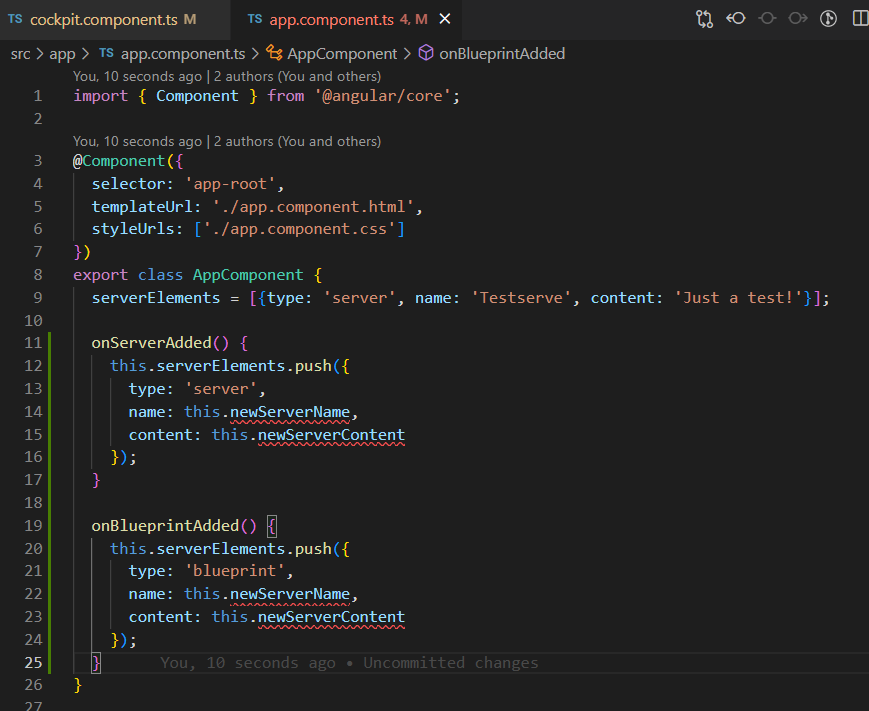
srvElement-et nem ismeri még



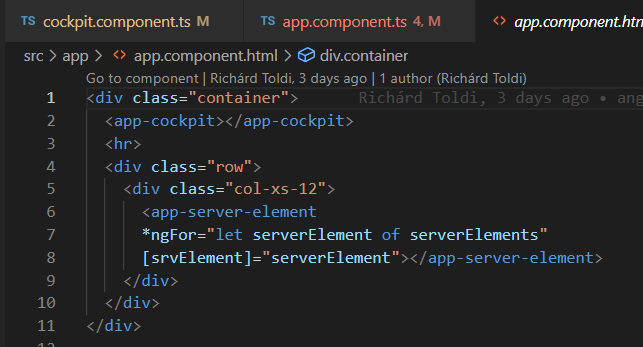
de így már felismeri

**Custom Event binding:**

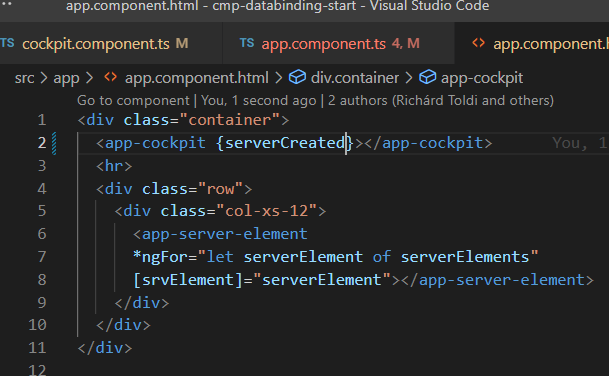
We want to inform the parent component that a new server was created by a child component.

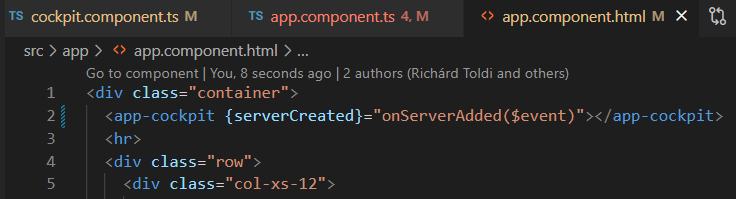


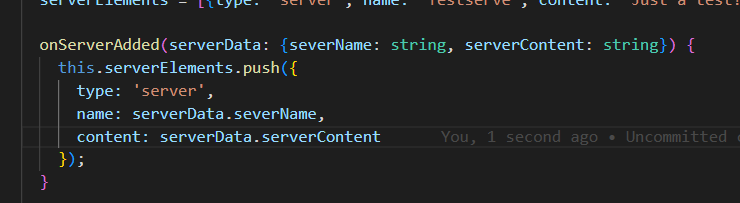
komponensből a metódusokat áthoztuk a fő ts file-ba



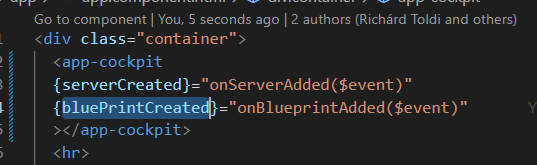
Eredeti állapot



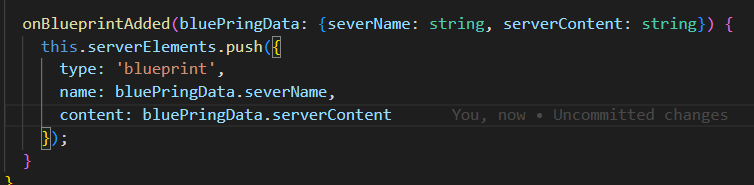




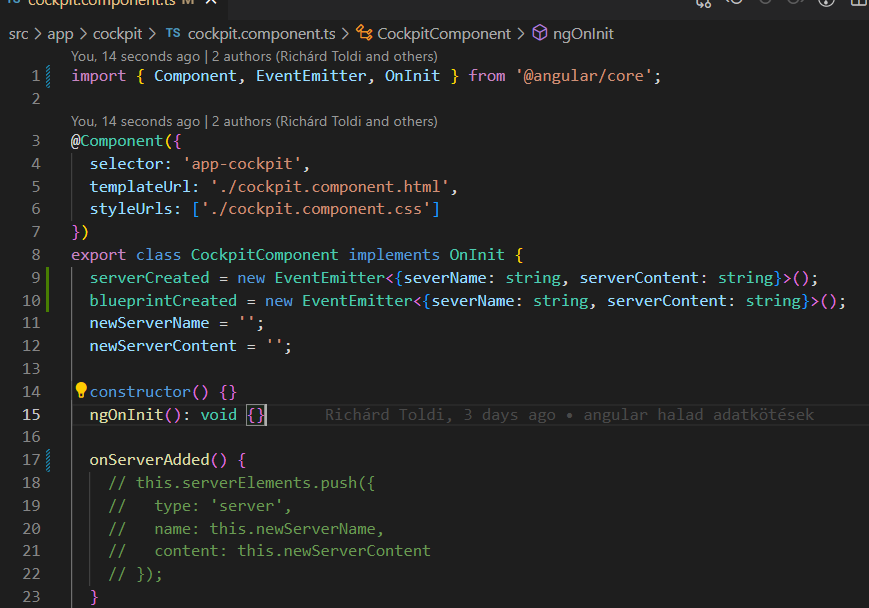
metódust átalakítottuk



event neve a kijelölt rész



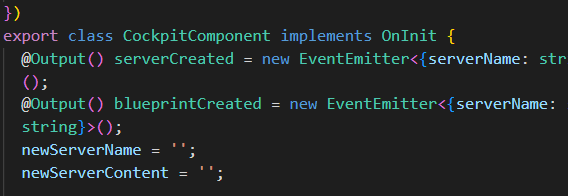
másik metódust is átírtuk



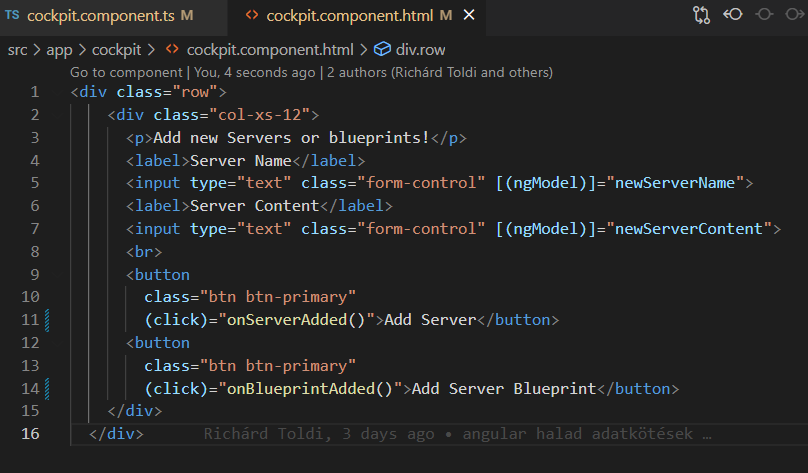
Eventeket itt is létrehoztuk

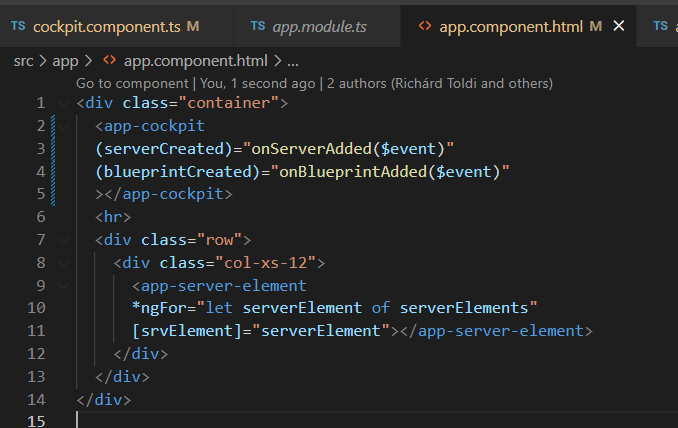


metódusokat az eventnek megfelelően így kellet átírni



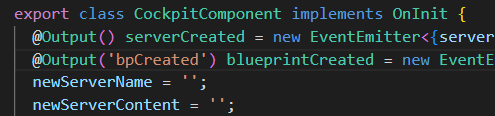
kell, mert most nem külső komponenstől kapjuk az adatokat, hanem külső komponensnek adjuk át az adatokat



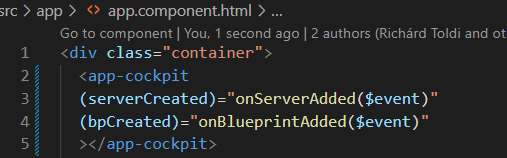


sima zárójel kell az attributomos kötéshez!!!

és már megy is mint eredetileg



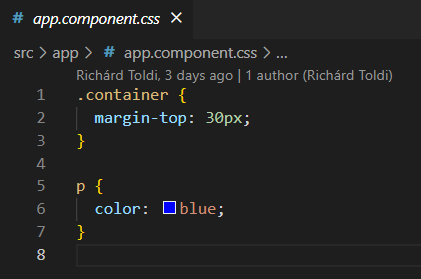
Alias az output esetén is működik

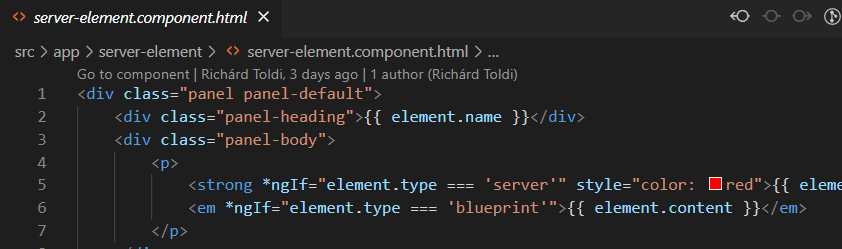


itt is át kell akkor írni, hogy ne az event nevét figyelje, hanem az aliast

**View Encapsulation:**

Minden komponensnek megvan a saját css file-ja és csak a komponensre érvényes a css beállítás. Ha a fő app komponensbe be is van állítva, hogy a p tag kék legyen.

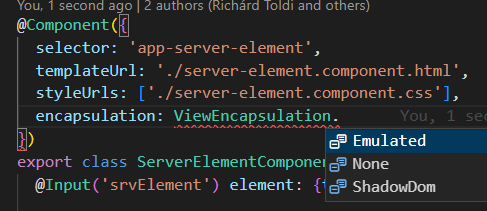




a server komponens p tegje nem lesz kék, mert az külön komponens csak!

Ha azt akarjuk, hogy a komponensre is érvényes legyen akkor a css-be át kell hozni ami kapcsolva van. Ez direkt így lett kialakítva, hogy ne keveredjenek a css-eket össze a komponensek között.

**Style encapsulation**



Emulated a default beállítás



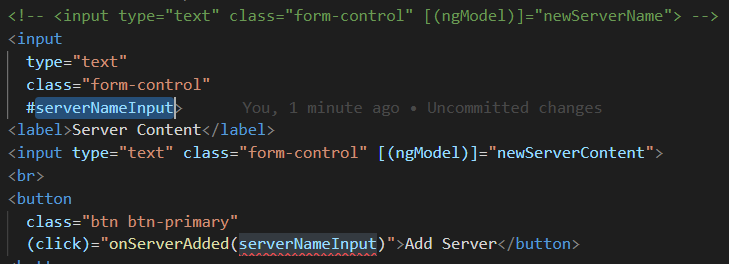
ezzel a beállítással a komponens css-je globális lesz, minden amit belövök a komponensnél máshol is az lesz



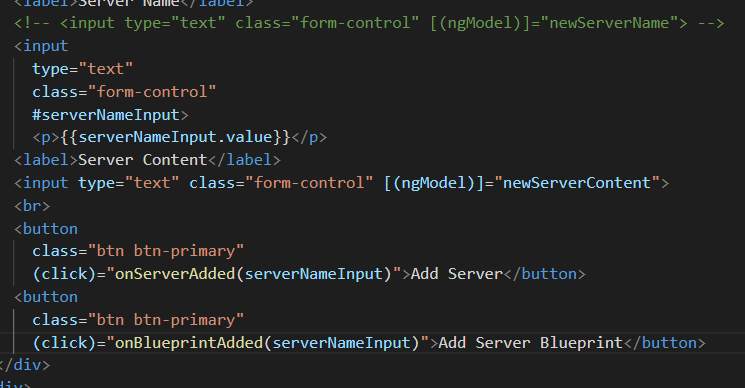
olyan mintha emulated lenne beállítva

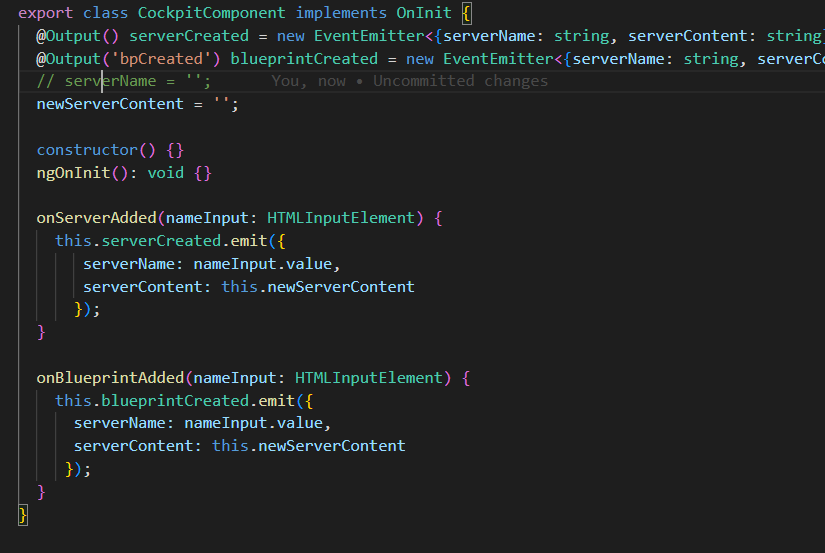
**Local reference:**

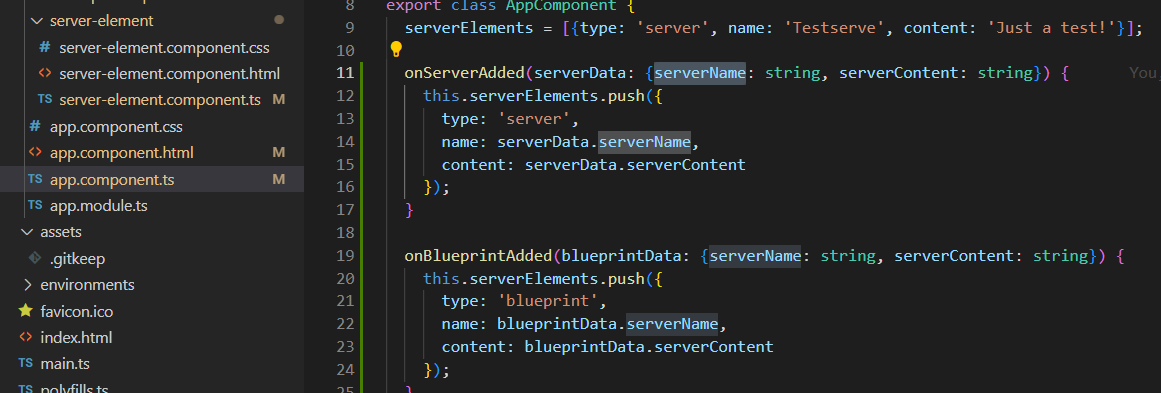
Bárhol használható a template-n belül, de csak ott lehet.



így adható át a typescript kódnak, hogy a metódusnak átadjuk paraméterként







Fontos, hogy az elnevezések egyezzenek, serverName, első r-t kihagytam ezért az adatkötés nem volt jó…

**@ViewChild() in Angular 8+**

In **Angular 8+**, the @ViewChild() syntax which you'll see in the next lecture needs to be changed slightly:

Instead of:

1. @ViewChild('serverContentInput') serverContentInput: ElementRef;

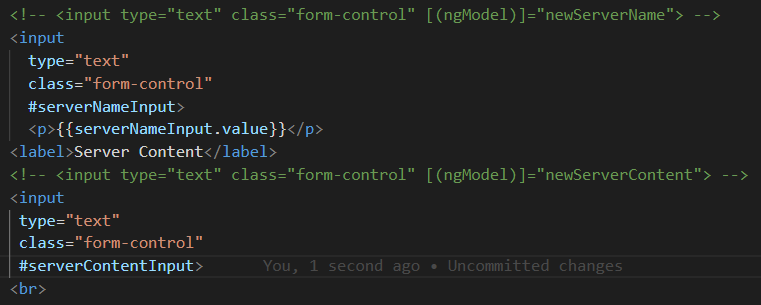
use

1. @ViewChild('serverContentInput', {static: true}) serverContentInput: ElementRef;

The same change (add { static: true } as a second argument) needs to be applied to ALL usages of @ViewChild() (and also @ContentChild() which you'll learn about later) IF you plan on accessing the selected element inside of ngOnInit().

If you DON'T access the selected element in ngOnInit (but anywhere else in your component), set static: false instead!

If you're using Angular 9+, you only need to add { static: true } (if needed) but not { static: false }.

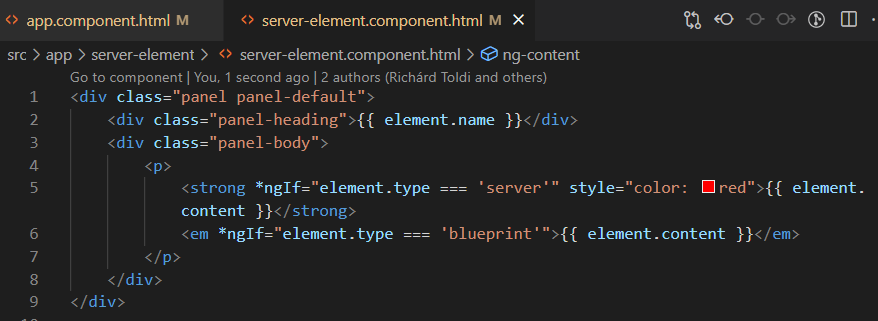




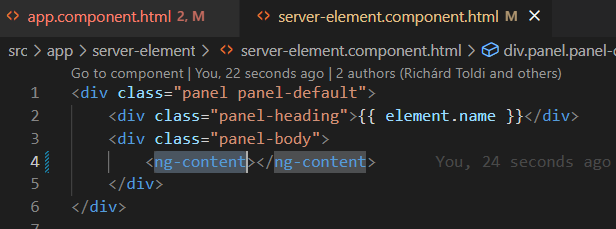
A komponens typescriptjét is átírtuk a 2 metódust. Ez a viewChild egy ElementRef-et ad nekünk aminek az értékét a .nativeElement.value-val tudjuk elérni. Ez egy angular cucc szóval az ng/core-ba importálnunk kell.

Ez a megoldás nem annyira közkedvelt! Mert a DOM-on keresztül megy.

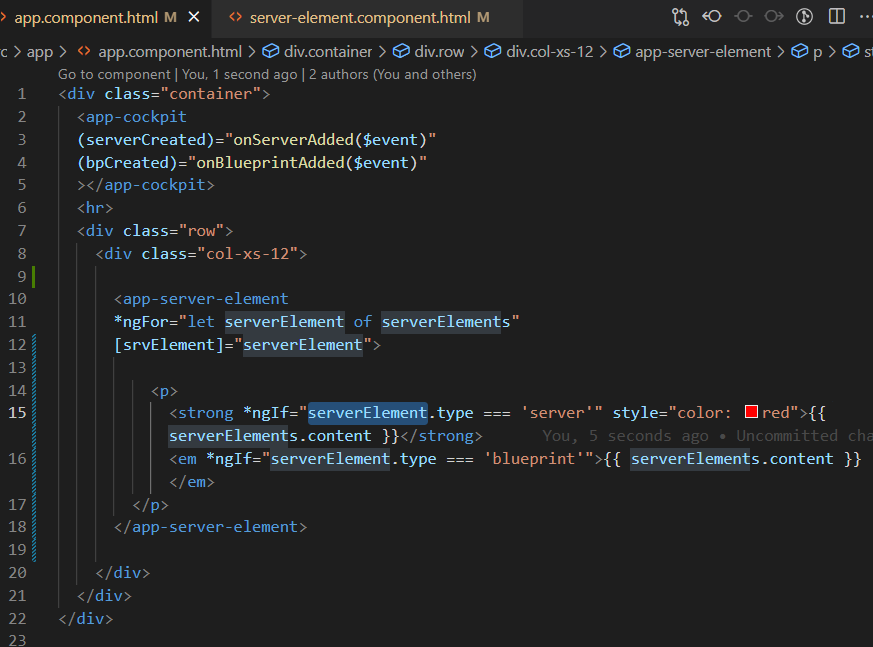
**ng-content:**

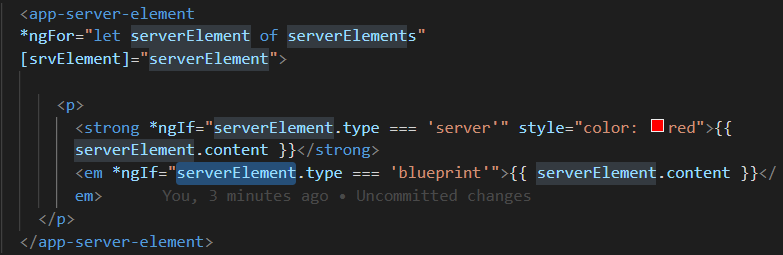
****

eredeti állapot p tegek közti részt raktuk át az app component-be



ez fogja behúzni a content-et amit talál egy nyitó és egy záró teg között





Elírás volt a .content-nél nem kell a többesszám!!!

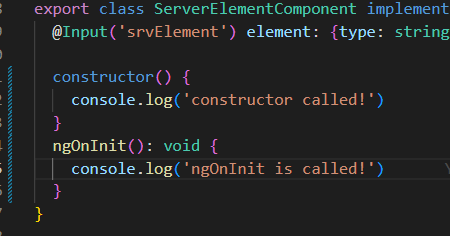
Kinézeter ugyanaz, de igazából a kontent ki van vetítve a komponensre, nincs benne.

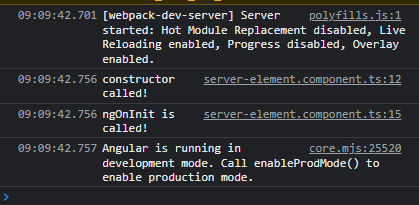
**Component life cycle:**

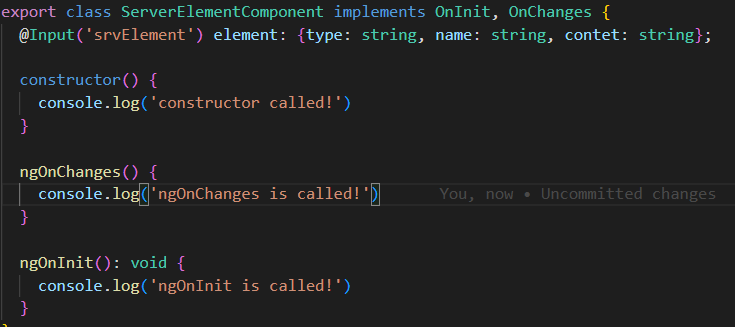
ngOnInit lifecycle hook

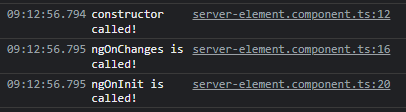
**Elmélet:**

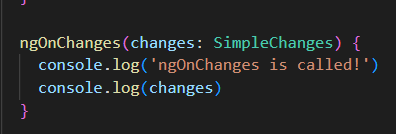
1. fázis **ngOnChanges** -> called after a bound input property changes
2. fázis **ngOnInit** -> called once the component is initialized
3. fázis **ngDoCheck** -> called during every change detection run
4. fázis **ngAfterContentInit** -> called after content (ng-content) has been projected into view
5. fázis **ngAfterContentChecked** -> called every time the projected content has been checked
6. fázis **ngAfterViewInit** -> called after the component’s view (and child views) has been initialized
7. fázis **ngAfterViewChecked** -> called every time the view (and child views) have been checked
8. fázis **ngOnDestroy** -> called once the component is about to be destroyed

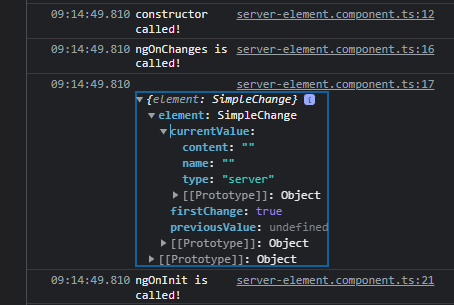


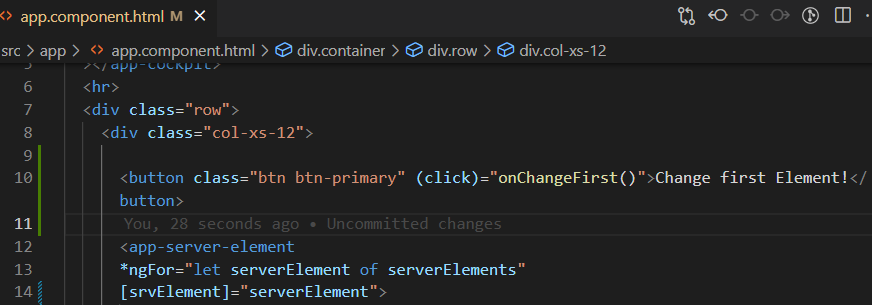


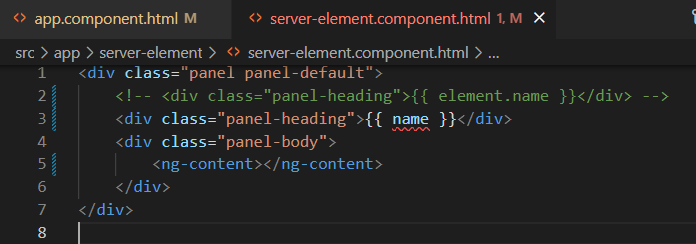


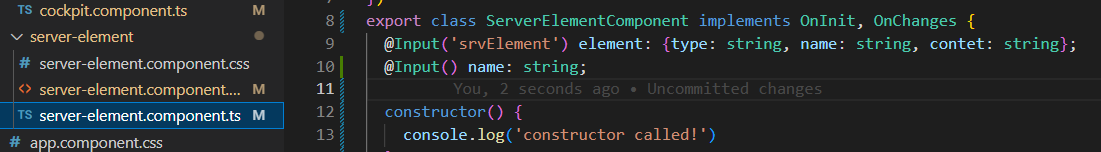


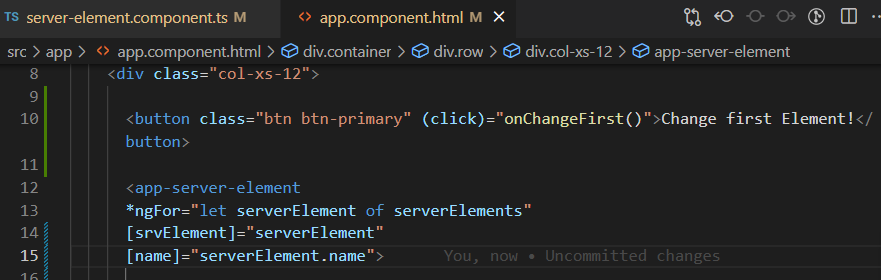


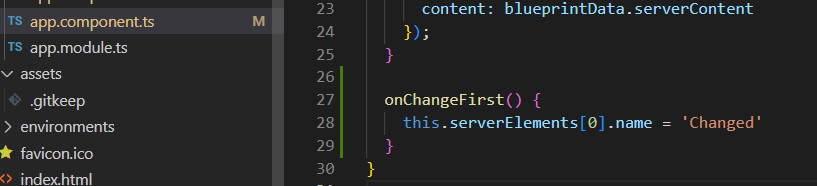


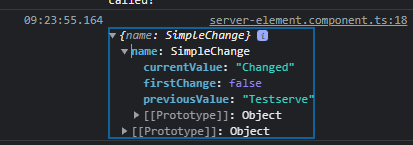






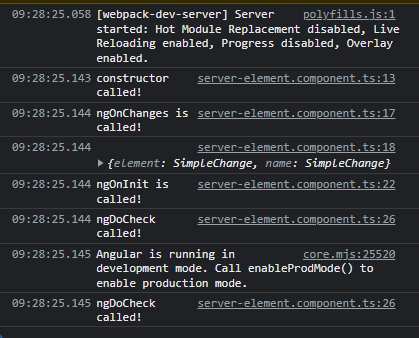




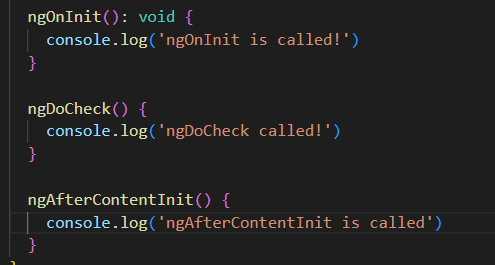


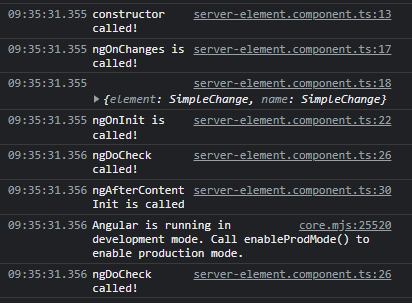
Ez akkor lehet jó, ha valami változásra akarunk reagálni.

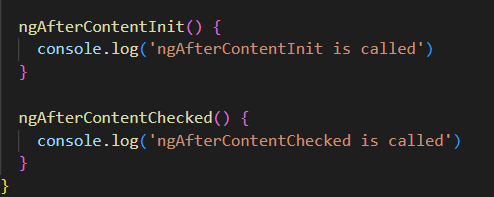


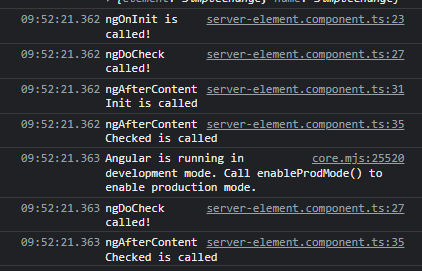


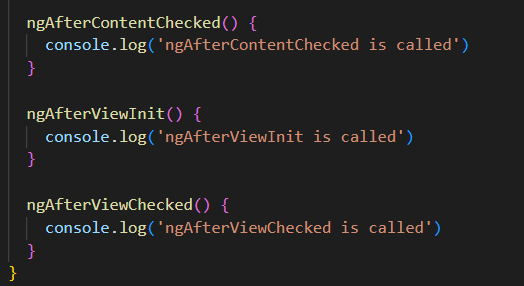
ngDoCheck meghívódik, mindig amikor az angulár ellenörzi a változásokat

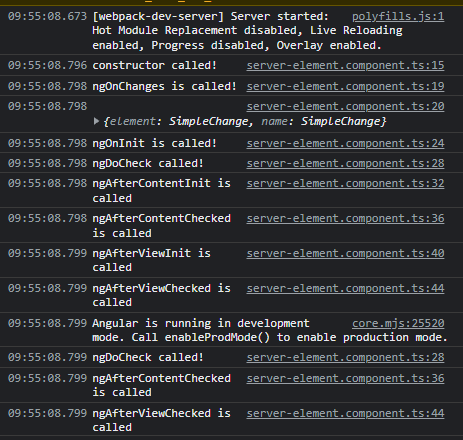


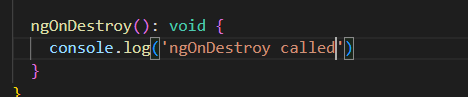


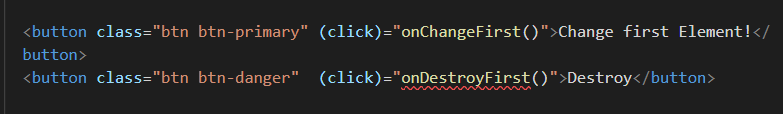


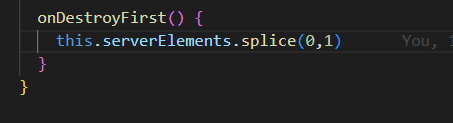


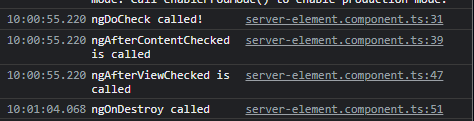


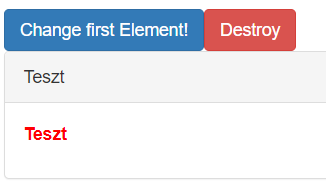


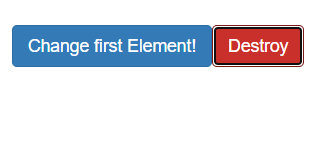






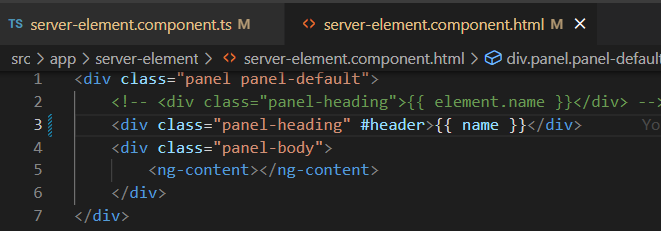


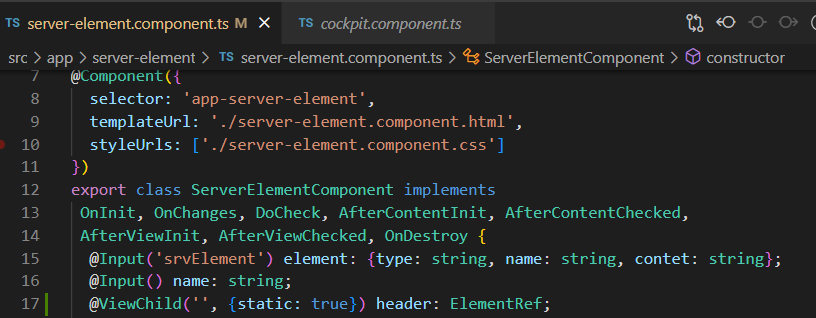


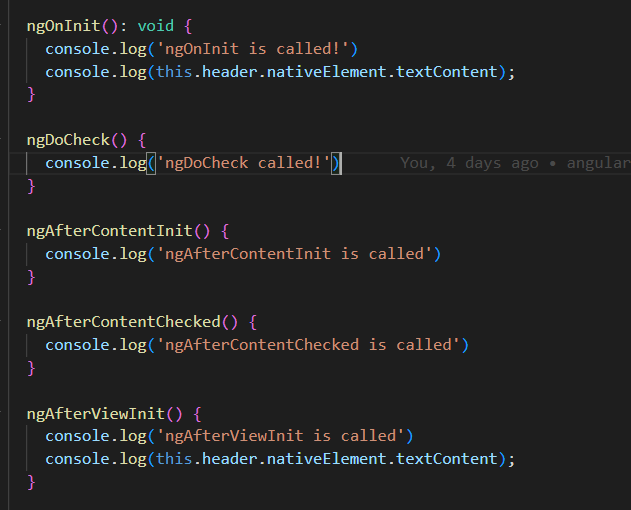


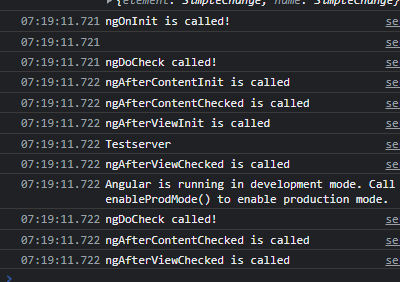
Törli az első elemet mindig, eltávolítottuk a tömbből.

**Lifecycle hooks and template access:**

****





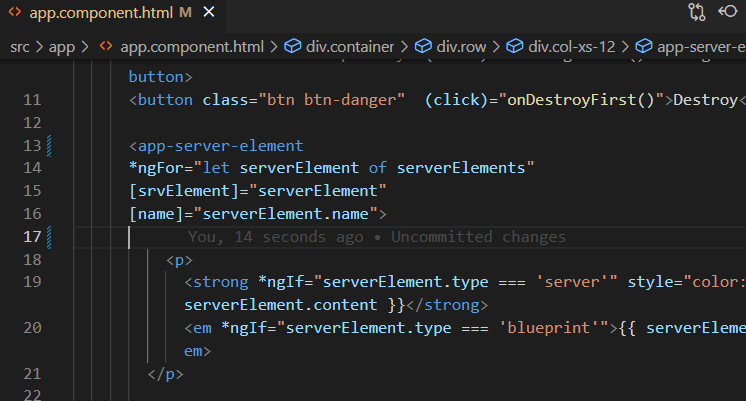


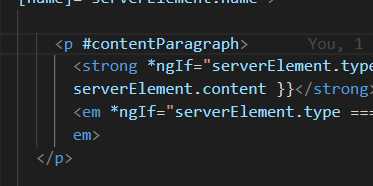
ViewInit után látja az adatot, de előtte nem!

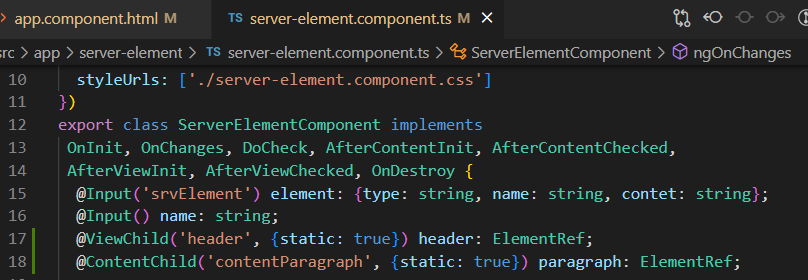
ViewInit után hozzáfér a template-hez előtte nem!

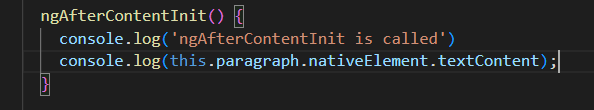
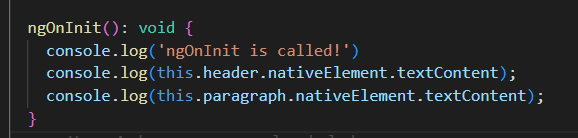
**ContentChild:**

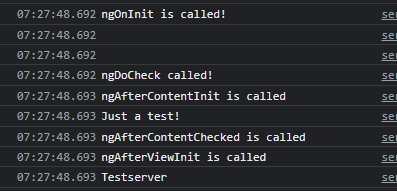
Ugyanúgy kell használni, mint a ViewChild-et, kell a static….











csak a contentinit után fér hozzá az adathoz