Clase 10 @October 27, 2022

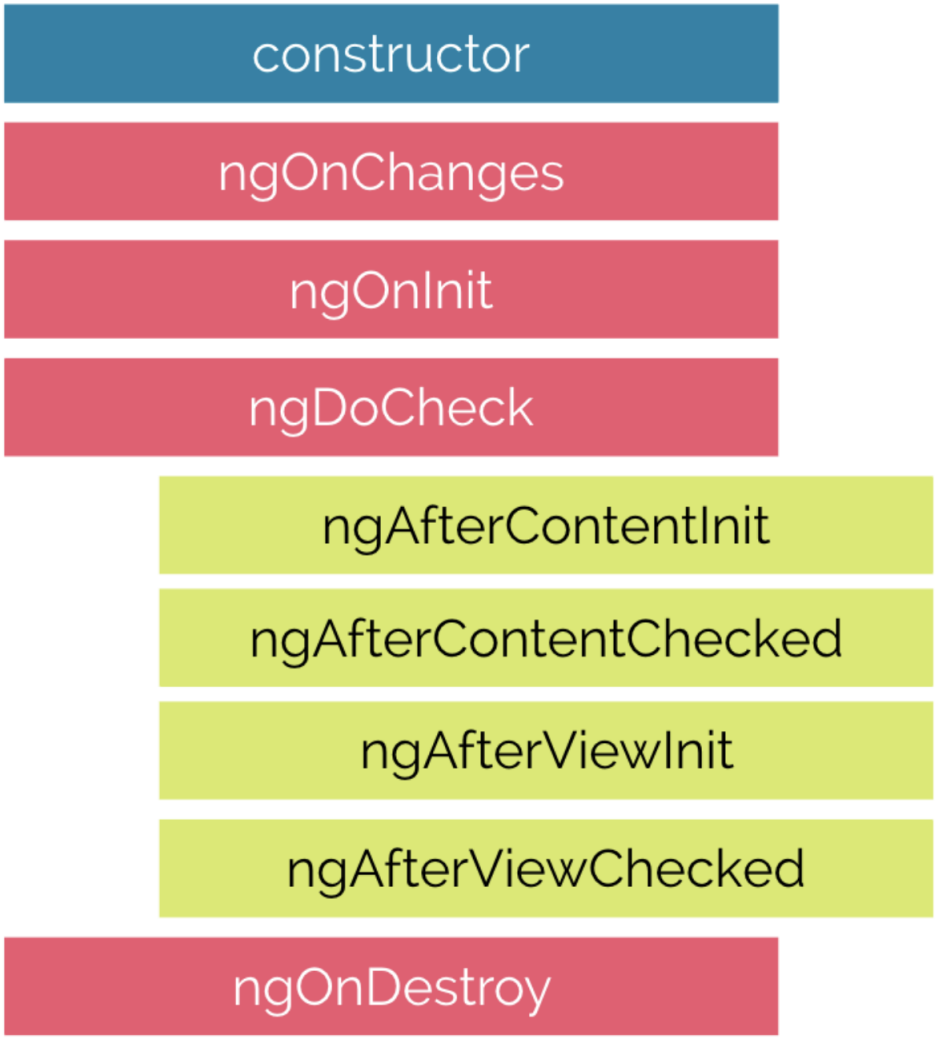
Esta clase veremos

- Ciclo de vida de los componentes: Hooks
- Introduccion a las interfaces
- Componentes anidados o Nested componentes
- Routing
- Practico 4 (incluyendo lo visto en clase 9 y clase 10)

Ciclo de vida de los componentes: Hooks

Para ayudar a trabajar con los componentes, Angular nos da acceso a los momentos clave de su vida a través de unos *callbacks* , los denominados ***lifecycle hooks***

Estos son:



Revisemos cada uno de los eventos:

- **ngOnChanges**: Este evento se ejecuta cada vez que se cambia un valor de un `input control` dentro de un componente. Se activa primero cuando se cambia el valor de una propiedad vinculada. Siempre recibe un `change data map` o mapa de datos de cambio, que contiene el valor actual y anterior de la propiedad vinculada envuelta en un `SimpleChange`
- **ngOnInit**: Se ejecuta una vez que Angular ha desplegado los `data-bound properties` (variables vinculadas a datos) o cuando el componente ha sido inicializado, una vez que `ngOnChanges` se haya ejecutado. Este evento es utilizado principalmente para inicializar la data en el componente.
- **ngDoCheck**: Se activa cada vez que se verifican las propiedades de entrada de un componente. Este método nos permite implementar nuestra propia lógica o

algoritmo de detección de cambios personalizado para cualquier componente.

- **ngAfterContentInit**: Se ejecuta cuando Angular realiza cualquier muestra de contenido dentro de las vistas de componentes y justo después de `ngDoCheck`. Actuando una vez que todas las vinculaciones del componente deban verificarse por primera vez. Está vinculado con las inicializaciones del componente hijo.
- **ngAfterContentChecked**: Se ejecuta cada vez que el contenido del componente ha sido verificado por el mecanismo de detección de cambios de Angular; se llama después del método `ngAfterContentInit`. Este también se invoca en cada ejecución posterior de `ngDoCheck` y está relacionado principalmente con las inicializaciones del componente hijo.
- **ngAfterViewInit:** Se ejecuta cuando la vista del componente se ha inicializado por completo. Este método se inicializa después de que Angular ha inicializado la vista del componente y las vistas secundarias. Se llama después de `ngAfterContentChecked`. Solo se aplica a los componentes.
- **ngAfterViewChecked:** Se ejecuta después del método `ngAfterViewInit` y cada vez que la vista del componente verifique cambios. También se ejecuta cuando se ha modificado cualquier enlace de las directivas secundarias. Por lo tanto, es muy útil cuando el componente espera algún valor que proviene de sus componentes secundarios.
- **ngOnDestroy:** Este método se ejecutará justo antes de que Angular destruya los componentes. Es muy útil para darse de baja de los observables y desconectar los `event handlers` para evitar `memory leaks` o fugas de memoria.

Interfaces de los hooks

Podemos definir los métodos directamente en la clase de componente, pero también podemos usar la ventaja de la interfaz, ya que cada uno de estos tiene una interfaz de TypeScript asociada.

```
import {
  Component,
  OnChanges,
  OnInit,
  DoCheck,
  AfterContentInit,
  AfterContentChecked,
```

```
  AfterViewInit,
  AfterViewChecked,
  OnDestroy,
} from '@angular/core';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent
  implements
  OnChanges,
  OnInit,
  DoCheck,
  AfterContentInit,
  AfterContentChecked,
  AfterViewInit,
  AfterViewChecked,
  OnDestroy {

  data = 10;
  constructor() {
    console.log(`new - data is ${this.data}`);
  }
  ngOnChanges() {
    console.log(`ngOnChanges - data is ${this.data}`);
  }
  ngOnInit() {
    console.log(`ngOnInit  - data is ${this.data}`);
  }
  ngDoCheck() {
    console.log('ngDoCheck');
  }
  ngAfterContentInit() {
    console.log('ngAfterContentInit');
  }
  ngAfterContentChecked() {
    console.log('ngAfterContentChecked');
  }
  ngAfterViewInit() {
    console.log('ngAfterViewInit');
  }
  ngAfterViewChecked() {
    console.log('ngAfterViewChecked');
  }
  ngOnDestroy() {
    console.log('ngOnDestroy');
  }
  addNumber(): void {
    this.data += 10;
  }
  deleteNumber(): void {
    this.data -= 10;
  }
}
```

Estas interfaces son simplemente contratos que obligan a cumplir una cierta estructura (por ejemplo, incluir el método `ngOnInit()`) a la clase que lo implementa.

No es obligatorio que usemos las interfaces asociadas a los *lifecycle hooks*. Angular ejecutará los *hooks* que implementemos, ya sea con, o sin las interfaces.

Pero aunque nos son obligatorios es una buena práctica que nos ayudará a prevenir errores (como por ejemplo no implementar un *hook* que realmente necesitamos) y a beneficiarte de las herramientas de tipado de TS.

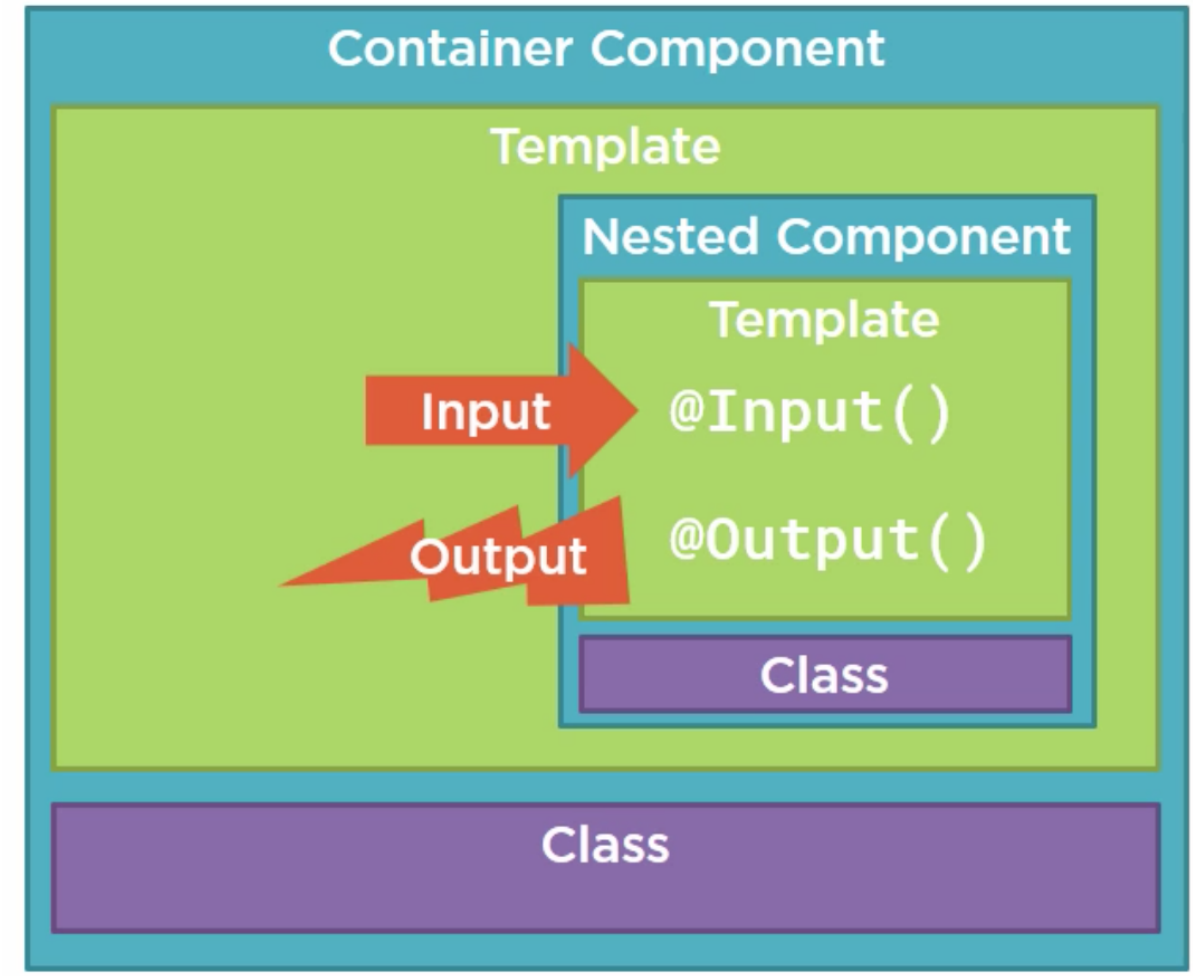
Componentes anidados o Nested components

Muchas veces nuestra aplicación va a tener features que son lo suficientemente complejas como para tratarlas de forma aislada, o donde existe una gran posibilidad de generar reuso en diferentes casos de uso, y por ende separarlas en componentes diferentes. Veremos ahora como hacer componentes anidados o `Nested Componentes` y como generar interacción entre el componente **contenedor** y el componente **anidado**. Estos componentes van a mandar datos de un lado para el otro, usando inputs y mandando outputs al componente contenedor.

Hay dos formas de usar componentes anidados:

1. A traves de su directiva (como ya vimos en el index.html)
2. A traves de routing, indicandole a un componente que tiene que rutear a otro componente

Por ahora usaremos la opción 1. El criterio que utilizaremos para indicar si un componente es 'anidable' o no, es simplemente a partir de **evaluar si su template maneja una parte de una view más grande, y obviamente si tiene un selector y una forma de comunicación con su contenedor**.



Supongamos que nuestras Tareas de nuesta API tienen una popularidad asociada. Creemos ahora un nested component que lo que haga es mostrar estrellitas por cada tarea que tengamos. Lo que queremos hacer es que el mostrado de estrellitas sea un componente aparte, que maneje su propia interacción tanto de inputs de sus componentes contenedores, como de outputs hacia otros componentes.

1) Creamos los archivos para nuestro StarComponent

En primer lugar, creamos nuesto componente con el commando: `ng generate component Star`

2) Creamos el StarComponent

Dentro de `star.component.ts`, pegamos el siguiente código:

```
import { Component, OnChanges } from '@angular/core';

@Component({
  selector: 'app-star',
  templateUrl: './star.component.html',
```

```
  styleUrls: ['./star.component.css']
})
export class StarComponent implements OnChanges {

    rating: number = 4; //hardcodeamos un valor por defecto para ver algo
    starWidth: number;

    ngOnChanges():void {
        //86 es el width de nuestras estrellitas (ver el template)
        //como estamos implementando el OnChanges, cada vez que el valor de 'rating' c
ambie
        //esto se va a refrescar
        this.starWidth = this.rating * 86/5;
    }
}
```

3) Creamos el Template para nuestro StarComponent y sus estilos

Dentro de `star.component.html`, pegamos el siguiente código:

```
<div clas="crop"
    [style.width.px]="starWidht"
    [title]="rating">
    <div style="width:86px">
        <span class="glyphicon glyphicon-star" *ngIf="rating>=1"></span>
    <span class="glyphicon glyphicon-star" *ngIf="rating>=2"></span>
    <span class="glyphicon glyphicon-star" *ngIf="rating>=3"></span>
    <span class="glyphicon glyphicon-star" *ngIf="rating>=4"></span>
    <span class="glyphicon glyphicon-star" *ngIf="rating>=5"></span>
    </div>
</div>
```

A su vez, vamos a `star.component.css`, y pegamos el siguiente código:

```
.crop {
    overflow: hidden;
}

div {
    cursor: pointer;
}
```

4) Agregamos el StarComponent al AppModule si no se importo automaticamente.

Dentro de `app.module.ts`, reemplazamos lo que hay y pegamos el siguiente código:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { HomeworksListComponent } from './homeworks-list/homeworks-list.component';
import { HomeworksFilterPipe } from './homeworks-list/homeworks-filter.pipe';
import { HomeworksService } from './services/homeworks.service';
import { StarComponent } from './star/star.component'; //importamos el starcomponent

@NgModule({
  declarations: [
    AppComponent,
    HomeworksListComponent,
    HomeworksFilterPipe,
    StarComponent  //agregamos a las declarations
  ],
  imports: [
    FormsModule,
    BrowserModule
  ],
  providers: [
    HomeworksService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5) Usamos el StarComponent dentro de HomeworksListComponent

En el template de nuestro componente de listado de tareas, es decir dentro de ``homeworks-list.component.html`, agregamos: Dentro del header de la tabla, es decir, en el tag thead```, agregamos esta última celda:

```
<th>Rating</th>
```

Dentro del body de la tabla, es decir, en el tag `tbody`, y dentro del `*ngFor` sobre las rows, agregamos esta última celda:

```
<td>
   <app-star></app-star>
</td>
```

7) Definimos la Input Property Rating en StarComponent

Para ello precisamos importar Input:

El decorador Input en un componente secundario o directiva significa que la propiedad puede recibir su valor de su componente principal, permite que los datos fluyan del padre al hijo.

```
import { Component, OnChanges, Input } from '@angular/core';
```

Y luego agregar el decorador a la property `rating`:

```
@Input() rating: number;
```

6) Usando input properties

Vemos que se muestran 5 estrellas y no 4 como habíamos hardcodeado. Ni que tampoco se modifica el valor en el OnChanges (esto es porque el OnChanges se cambia cuando alguna input property de un componente se refresca). Veamos esto:

Si un componente anidado quiere recibir inputs de su componente contenedor, debe exponer properties a partir del decorador `@Input`. En consecuencia, cada property deberamos decorarla con tal decorador. Luego, el componente contenedor, deberá encargarse de setearle dicha property al componente anidado a partir de property binding en el template con paréntesis rectos:

```
<td>
    <app-star [rating]='aHomework.rating'></app-star>
</td>
```

Para hacer lo de arriba precisamos:

- Ir a `app/models/homework.ts` y agregar la property rating
- Ir a `app/homeworks-list/homeworks-list.component.ts` y agregar un valor de rating en las tareas que tengamos creadas. Esto no se describe aquí pero se puede observar en el código fuente. Lo que haremos es ir al HTML del template y cambiar lo que teníamos antes por lo que acabamos de ver arriba.

8) Levantando eventos desde un componente anidado

Si queremos que nuestro StarComponent se comunique con su contenedor HomeworksListComponent, debemos usar eventos. Para ello tenemos que usar, para definirlos, el decorator `@Output`, el cual debemos aplicar sobre la property del componente anidado que queremos usar (es importante notar que el tipo de dicha property debe ser un `EventEmitter`, la única forma de pasar datos a su contenedor). A su vez, dicha clase se basa en generics para especificar el tipo de datos de lo que queremos pasar al componente contenedor.

La forma que usaremos para activar el evento, es a partir del método click sobre las estrellas; una vez que se haga el click, se activará la lógica que definamos del evento.

La sintaxis para activar al evento desde nuestros componentes anidados es: onClick() { this.nombreDelEvento.emit('parametro a pasar al contenedor'); } Siendo `emit` lo que usamos para levantar el evento al contenedor. Vayamos al código: Lo primero que hacemos es importar `EventEmitter` y `Output` en nuestro `StarComponent`:

El decorador `@Output()` en un componente hijo permite que los datos fluyan del hijo al padre.

```
import { Component, OnChanges, Input, Output, EventEmitter } from '@angular/core';
```

Luego definimos la property del evento en nuestra clase del StarComponent:

new EventEmitter<string> Le dice a Angular que cree un nuevo emisor de eventos y que los datos que emite sean de tipo cadena.

```
@Output() ratingClicked: EventEmitter<string> = new EventEmitter<string>();
```

Luego hacemos el binding del evento en el template del evento:

```
<div class="crop"
    [style.width.px]="starWidth"
    title={{starWidth}}
    (click)='onClick()'>
    <div style="width: 86px">
      <span class="glyphicon glyphicon-star" *ngIf="rating>=1"></span>
      <span class="glyphicon glyphicon-star" *ngIf="rating>=2"></span>
      <span class="glyphicon glyphicon-star" *ngIf="rating>=3"></span>
      <span class="glyphicon glyphicon-star" *ngIf="rating>=4"></span>
      <span class="glyphicon glyphicon-star" *ngIf="rating>=5"></span>
```

```
    </div>
</div>
```

Y ahora volvemos al StarComponent y agregamos la funcion onClick que acabamos de declarar en nuestro template:

```
onClick(): void {
    this.ratingClicked.emit(`El raiting es ${this.rating}!`);
}
```

Finalmente, lo que haremos es agregar la referencia al evento en el template:

para vincular la rating propiedad del elemento secundario a la aHomework.rating propiedad del elemento principal.

```
<app-star [rating]='aHomework.rating'
    (ratingClicked)='onRatingClicked($event)'>
</app-star>
```

Y dentro del código del HomeworksListComponent definimos el callback que queremos que se ejecute cuando se haga click en nuestras estrellas:

message es el string que lanza el emit del evento definido antes

```
onRatingClicked(message:string):void {
    this.pageTitle = 'HomeworksList ' + message;
}
```

Routing

1. Creamos el menu

ng generate component menu

html

```
<div class="panel panel-primary">
  <div class="panel-heading">
    {{pageTitle}}
```

```
  </div>
  <td>
    <a [routerLink]="['/homeworks']"> {{"Homeworks list" | uppercase}} </a>
  </td>
</div>
<router-outlet></router-outlet>
```

component

```
pageTitle : string = 'Homework app';
```

homeworks module

```
const rootes:Routes = [
  { path: 'homeworks', component: HomeworksListComponent}]

@NgModule({
  declarations: [
    HomeworksListComponent,
    HomeworksFilterPipe,
    StarComponent,
    HomeworkDetailComponent,
    MenuComponent,
    ],
    exports: [HomeworksListComponent,
      RouterModule, MenuComponent],
    imports: [
      CommonModule,
      FormsModule,
      HttpClientModule,
      RouterModule.
      forRoot(rootes, {onSameUrlNavigation:'reload'})
    ],
  providers:[HomeworksService]

}
)
export class HomeworksModule {

}
```

app module html

```
<app-menu></app-menu>
```

2. Creamos un HomeworkDetailComponent:

ng generate component homeworkDetail

`homework-detail.component.html`:

```
<!--Tabla de tareas -->
  <div class='table-responsive'>
    <table class='table'>
      <!--Cabezal de la tabla -->
      <thead>
      <tr>
        <th>Id</th>
        <th>Description</th>
        <th>DueDate</th>
        <th>Score</th>
        <th>Rating</th>
        <th>
        </th>
      </tr>
      </thead>
      <!--Cuerpo de la tabla-->
      <tbody>
        <td>{{aHomework?.id}}</td>
        <td>{{aHomework?.description | uppercase}}</td>
        <td>{{aHomework?.dueDate}}</td>
        <td>{{aHomework?.score}}</td>
        <td>
          <div>
                <table>
                  <thead>
                  <tr>
                    <th>Problem</th>
                    <th>Score</th>
                  </tr>
                  </thead>
                  <tbody>
                  <tr *ngFor='let aExercise of aHomework?.exercises'>
                    <td>{{aExercise.problem}}</td>
                    <td>{{aExercise.score}}</td>
                  </tr>
                  </tbody>
                </table>
            </div>
        </td>

      </tbody>
    </table>
</div>
```

```
  <div class='panel-footer'>
    <a class='btn btn-default' (click)="onBack()" style="width:80px">
      <i class='glyphicon glyphicon-chevron-left' ></i> Back
    </a>
  </div>
</div>
```

`homework-detail.component.ts`:

```
import { Component, OnInit } from '@angular/core';
import { Homework } from '../models/Homework';

@Component({
  selector: 'app-homework-detail',
  templateUrl: './homework-detail.component.html',
  styleUrls: ['./homework-detail.component.css']
})
export class HomeworkDetailComponent implements OnInit {
  pageTitle : string = 'Homework Detail';
aHomework : Homework | undefined = new Homework("2", "Otra tarea", 0, new Date(), [],
4);

  constructor() { }

  ngOnInit() {
  }
}
```

Y a su vez agregamos este componente en el HomeworksModule, primero haciendo el import y luego agregando `HomeworkDetailComponent` en el array de declarations:

```
import { HomeworkDetailComponent } from './homework-detail/homework-detail.component';
```

```
declarations: [
  AppComponent,
  HomeworksListComponent,
  HomeworksFilterPipe,
  StarComponent,
  WelcomeComponent,
  HomeworkDetailComponent
],
```

1. Seteamos el path en app.module.ts (AppModule)

En este caso el path sería: *homeworks/id*, indicando que ruta a un componente **HomeworkDetailComponent**. A su vez, le pasamos el parámetro id,

con una barra y un dos puntos adelante (/:id). Si quisiéramos más parámetros, repetimos esto.

```
RouterModule.forRoot([{ path: 'homeworks/:id', component: HomeworkDetailComponent }]
```

2. Ruteamos al path

Agregamos en ***HomeworkDetailComponent***

```
@Input() routerLink: string | any[] ="";
```

Luego en el HTML de nuestro ***HomeworkDetailComponent***, ponemos un link (ancla) sobre el nombre, de manera de que cada vez que se haga click sobre el mismo, dicha ruta se resuelva y se le pase el parámetro asociado.

```
  <td><a [routerLink]="['/homeworks', aHomework.id]"> {{aHomework.id | uppercase}} </a
></td>
```

3. Leemos los parámetros de la ruta en el HomeworkDetailComponent

Leemos los parámetros de la ruta, usando el service ActivatedRoute de '@angular/router'.

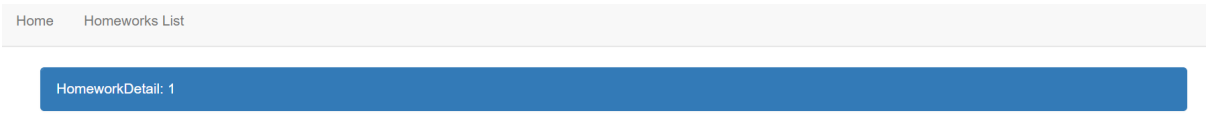
Lo inyectamos en nuestro componente para que use este servicio (el provider ya viene resuelto por el RouterModule que usamos la clase anterior):

```
constructor(private _currentRoute: ActivatedRoute, private serviceHomework:HomeworksSe
rvice) { }
```

1. Agarramos el parámetro de la ruta y lo ponemos en una variable privada, dicha lógica lo haremos en el OnInit (hay que implementar OnInit).

```
ngOnInit() : void {
  // let (es parte de ES2015) y define una variable que vive en este scope
  // usamos el nombre del parámetro que uamos en la configuración de la ruta y lo obte
nemos
  let id =+ this._currentRoute.snapshot.params['id'];
```

```
  // definimos el string con interpolacion
  this.pageTitle +=  `: ${id}`;
this.aHomework= this.serviceHomework.getHomeworks().find(x=>x.id==id.toString())
}
```



Routing a través de código

Haremos Routing en código en lugar de hacerlo con la directiva **RouterLink** que hemos venido usando en el template.

Por ejemplo: un botón de Save que tiene que ejecutar cierto código una vez que se llenen campos, y recién ahí rutear (si todo salió satisfactoriamente). Para routear con código, usaremos el Router Service.

Recordemos, cada vez que inyectemos un servicio en una clase, tenemos que preguntarnos “registramos este servicio en el Angular Injector?”. En este caso, el provider ya viene dad por el RouterModule, por lo que no tenemos que hacerlo

1. Importamos el Router Service (HomeworkDetailComponent)

En HomeworkDetailComponent:

```
import { ActivatedRoute, Router } from '@angular/router';
```

2. Inyectamos el servicio en la clase a través del constructor:

```
    constructor(private _currentRoute: ActivatedRoute, private _router : Router) {  }
```

3. Creamos una función que rutee a cierto path:

```
  onBack(): void {
       this._router.navigate(['/homeworks']); //En caso de que necesite parametros los
paso como otros argumentos
  }
```

4. En el template de HomeworkDetail (HTML), creamos un botón para ir para atrás:

```
<div class='panel-footer'>
  <a class='btn btn-default' (click)="onBack()" style="width:80px">
      <i class='glyphicon glyphicon-chevron-left' ></i> Back
  </a>
</div>
```