Angular Developer 7

lifecycle hooks | http

Tasks

online server | online docs | server sources

Reading and displaying highscores

1. Read current highscores (GET /scores)
   - reading as text/html - default
   - reading as json - add *accept: application/json* header
2. Display highscores (component):
   - List with entries (name - score pairs)
   - show only top 10 entries
3. List sorting
   - allow sorting by: score asc/desc

Authentication input

(Intro page form)

1. Add token input (student ID)
   - remove email from the form
   - add token input field (text entry, just required, no special validations)
2. Upon form submission validate entered token (POST /check-token)

My score

1. On game finished
   - submit player score and name (POST /scores)
   - sign with auth token (auth-token header)
2. Display my scores list (component):
   - filter data (only my entries)
   - sorting by score asc/desc
3. Update score lists every 30 seconds

COMPONENT LIFECYCLE

Life of a component

1. Beginning - component creation
2. Work work work
3. End - destruction

LIFECYCLE HOOKS

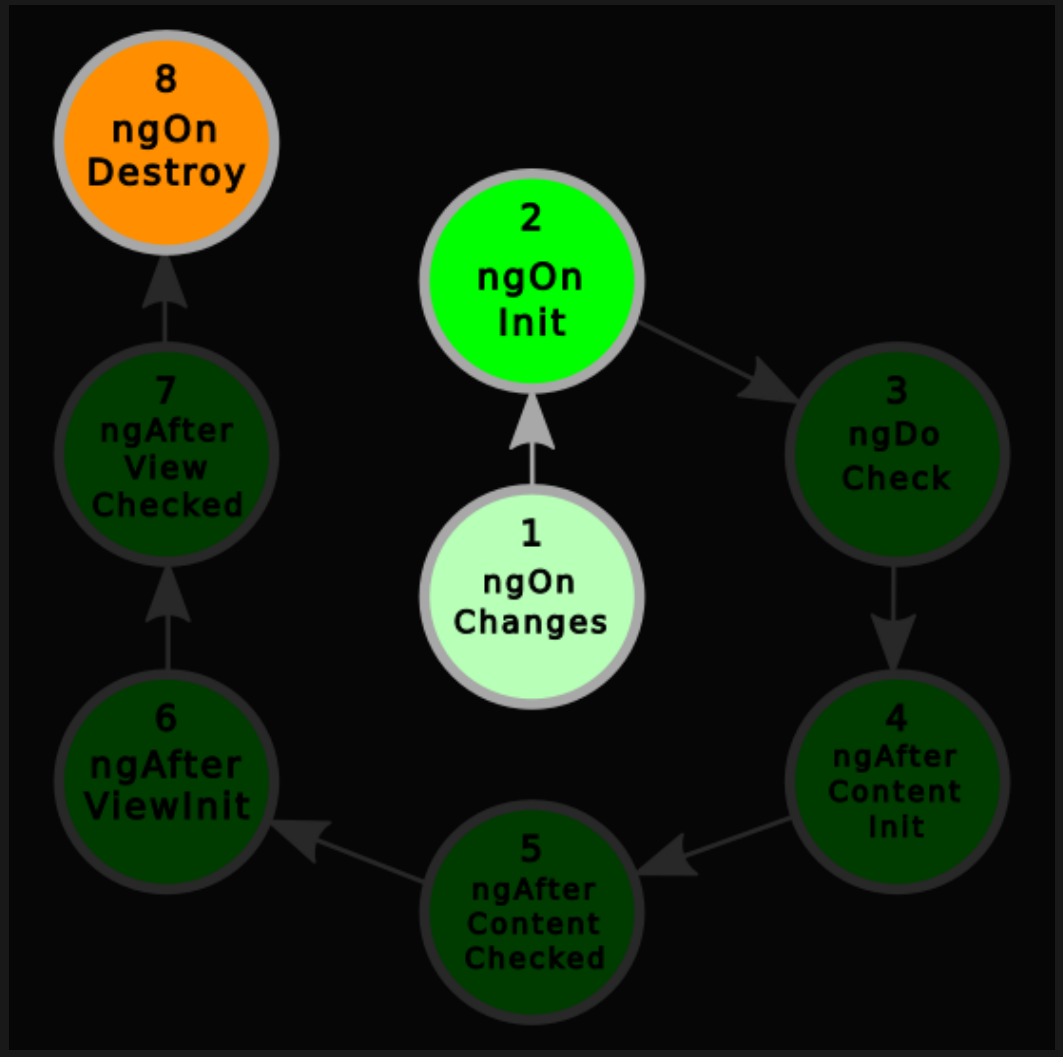
What is that?

Events from component life

What for?

Methods to execute logic when key events happen

From creation to destruction



no WORK loop here

- ngOnInit
- ngOnChanges
- ngOnDestroy

Constructor

- Js/Es6 and Ts feature
- ABSOLUTELY the first thing fired
- DI list of dependencies

ngOnInit

- Fired when component is ready
  - Inputs are connected
  - Data is displayed
- Called just once
- Place for our logic to kick in

ngOnChanges

- Fired ONLY in components with @Inputs()
- Executed when any of the @Inputs changes
- Receive special 'changes' param

ngOnDestroy

- Fired when component is to be destroyed
- To perform some cleanup logic

How to use

```
// sum.component.ts
import { OnInit } from '@angular/core';
export class SumComponent implements OnInit {
    @Input() a;
    @Input() b;

    public sum;

    public ngOnInit() {
      this.sum = this.a + this.b;
    }
}
```

Constructor vs ngOnInit

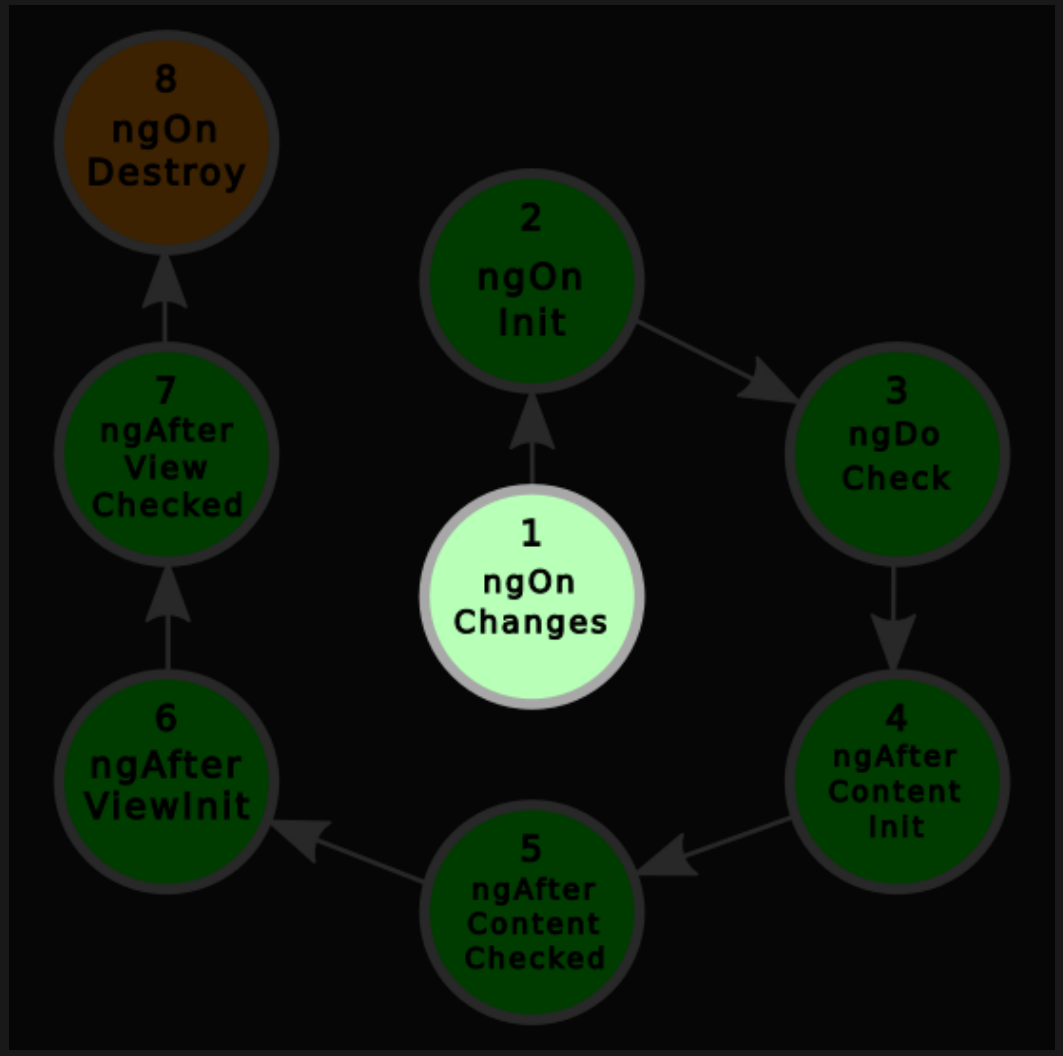
```
1  export class SumComponent implements OnInit {
2      @Input() a;
3      @Input() b;
4
5      public sum;
6
7      constructor() {
8          this.doTheMath('Constructor');
9      }
10
11     ngOnInit(): void {
12         this.doTheMath('ngOnInit');
13     }
14
15     public doTheMath(event): void {
```

Result

```
1 <!-- app.component.html -->
2 <app-sum [a]="0" [b]="6"></app-sum>
```

```
---Constructor--- summing: undefined + undefined = NaN
---ngOnInit--- summing: 0 + 6 = 6
```

Work / change detection loop



ngOnChanges

```
1  export class SumComponent implements
2                              OnInit, OnChanges, OnDestroy {
3      @Input() a;
4      @Input() b;
5
6      public sum;
7
8      constructor() {
9          this.doTheMath('Constructor');
10     }
11
12     ngOnInit(): void {
13         this.doTheMath('ngOnInit');
14     }
15
```

Result

```
---Constructor--- summing: undefined + undefined = NaN
---ngOnChanges--- summing: 0 + 6 = 6
{ "a":{"currentValue":0,"firstChange":true},
  "b":{"currentValue":6,"firstChange":true} }
---ngOnInit--- summing: 0 + 6 = 6
---ngOnChanges--- summing: 1 + 6 = 7
{ "a":{
    "previousValue":0,
    "currentValue":1,
    "firstChange":false
  }
}
---ngOnDestroy--- summing: 1 + 6 = 7
```

Summary

(most commonly used hooks)

Hook	Purpose
ngOnInit	Put your initialization logic
ngOnChanges	React on input changes
ngOnDestroy	Cleanup

COMMUNICATION WITH SERVER(S)

Http protocol

- Download data
- Upload data

Http request types

- GET
- POST
- PATCH
- PUT
- DELETE
- OPTIONS

Http requests url params

Way to pass information in URL

Information about resource

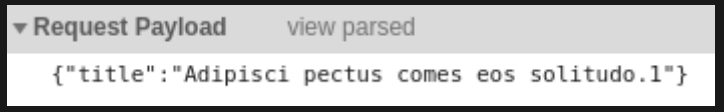
```
https://shop.com/products?order=asc

https://shop.com/products?order=asc&min_price=100
```

Http requests headers

- Metadata
- Request configuration
- Example: content type, auth data

Http request body

- Actual resource



REST

Representational state transfer

REST - architecture

- Client - server
- Stateless
- Cacheable
- Layerable

REST methods

- GET - reading data
- POST - creating data
- PATCH / PUT - updating data
- DELETE - deleting data
- OPTIONS - special automatic type

Rest url pattern

Address of the server		Name of the resource collection		Optional param to identify the resource
<api-url>	/	<resource>	/	[<id>]
https://gorest.co.in/public-api	/	todos	/	1

Method	Endpoint	Action
GET	https://gorest.co.in/public-api/todos	Read/list all todos
POST	https://gorest.co.in/public-api/todos	Create new todos
PATCH	https://gorest.co.in/public-api/todos/1	Update todo
DELETE	https://gorest.co.in/public-api/todos/1	Delete todo

HTTP IN ANGULAR

HttpClient service

from HttpClientModule

Lets organize things a little

```
$ ng generate service todos
```

```
// app.component.ts
import { Component } from '@angular/core';
import { TodosService } from './todos.service';

@Component(...)
export class AppComponent {
    constructor(private _todos: TodosService) {

    }
}
```

(Injecting created service)

Now back to http...

Import module

```
1  // app.module.ts
2  import { HttpClientModule } from '@angular/common/http';
3  @NgModule({
4    ...
5    imports: [
6        BrowserModule,
7        HttpClientModule
8    ],
9    ...
10 })
11 export class AppModule { }
```

Inject service to your ~~component~~ service

```
1  // todos.service.ts
2  import { HttpClient } from '@angular/common/http';
3  @Injectable({
4      providedIn: 'root'
5  })
6  export class TodosService {
7      constructor(private _http: HttpClient) {
8
9      }
10 }
```

Start making http calls

```
1  // todos.service.ts
2  import { HttpClient } from '@angular/common/http';
3  @Injectable({
4      providedIn: 'root'
5  })
6  export class TodosService {
7      constructor(private _http: HttpClient) { }
8
9      load(): Observable<ToDoResponse> {
10       const URL = 'https://gorest.co.in/public-api/todos';
11       return this._http.get<ToDoResponse>(URL);
12     }
13 }
```

Are we actually making a call?

(downloading data)

NO! HttpClient methods return observables that are lazy

PROMISE (ES6)	vs	OBSERVABLE (RXJS)
Dealing with async tasks		
Short living: one event		Short and long living: 0+ events
Not cancellable		Cancellable
Not retryable		Easy to retry
Fire immediately - EAGER		Wait for subscriber - LAZY

```
// app.component.ts
@Component(...)
export class AppComponent {
    public data = [];
    constructor(private _todos: TodosService) {
      this._todos.load().subscribe((result) => {
          this.data = result['data'];
      });
    }
}
```

Summary

- Import module

```
@NgModule({
  imports: [
      HttpClientModule
  ],
})
```

- Inject where you want to use it

```
export class TodosService {
    constructor(private _http: HttpClient) { }
}
```

- Prepare request and fire by subscribing

```
export class TodosService {
  load() {
    return this._http.get('https://url.to')
      .subscribe((result) => {
        console.log(result);
      });
  }
```

HttpClient methods

and their configuration

get()

```
load(): Observable<ToDoResponse> {
  const URL = 'https://gorest.co.in/public-api/todos';
  const Options = { ... };
  return this._http.get<ToDoResponse>(URL, Options);
}
```

Request options

```
get<T>(url: string, options?: {
    headers?: HttpHeaders | {
        [header: string]: string | string[];
    };
    observe?: 'body';
    params?: HttpParams | {
        [param: string]: string | string[];
    };
    reportProgress?: boolean;
    responseType?: 'json';
    withCredentials?: boolean;
}): Observable<T>;
```

Passing url params

```
load(): Observable<ToDoResponse> {
  const URL = 'https://gorest.co.in/public-api/todos';
  const Options = {
    params: {
      paramName: 'paramVal',
      second: 1
    }
  };
  return this._http.get<ToDoResponse>(URL, Options);
}
```

```
https://gorest.co.in/public-api/todos?paramName=paramVal&second=1
```

Adding http headers

in a sec...

patch()

```
1 return this._http.patch(this._endpoint + '/' + id);
```

patch()

```
return this._http.patch(this._endpoint + '/' + id,
  body
);
```

patch()

```
const body = { title: 'some text' };

return this._http.patch(this._endpoint + '/' + id,
  body
);
```

patch()

```
const body = { title: 'some text' };

return this._http.patch(this._endpoint + '/' + id,
  body,
  options
);
```

patch()

```
1 const body = { title: 'some text' };
2 const options = { headers };
3
4 return this._http.patch(this._endpoint + '/' + id,
5   body,
6   options
7 );
```

patch()

```
const headers = new HttpHeaders({
    Authorization: 'Bearer ' + TOKEN,
    'Content-Type': 'application/json',
});

const body = { title: 'some text' };
const options = { headers };

return this._http.patch(this._endpoint + '/' + id,
  body,
  options
);
```