Global Knowledge®

Angular Fundamentals

Services

Peter Kassenaar –
info@kassenaar.com

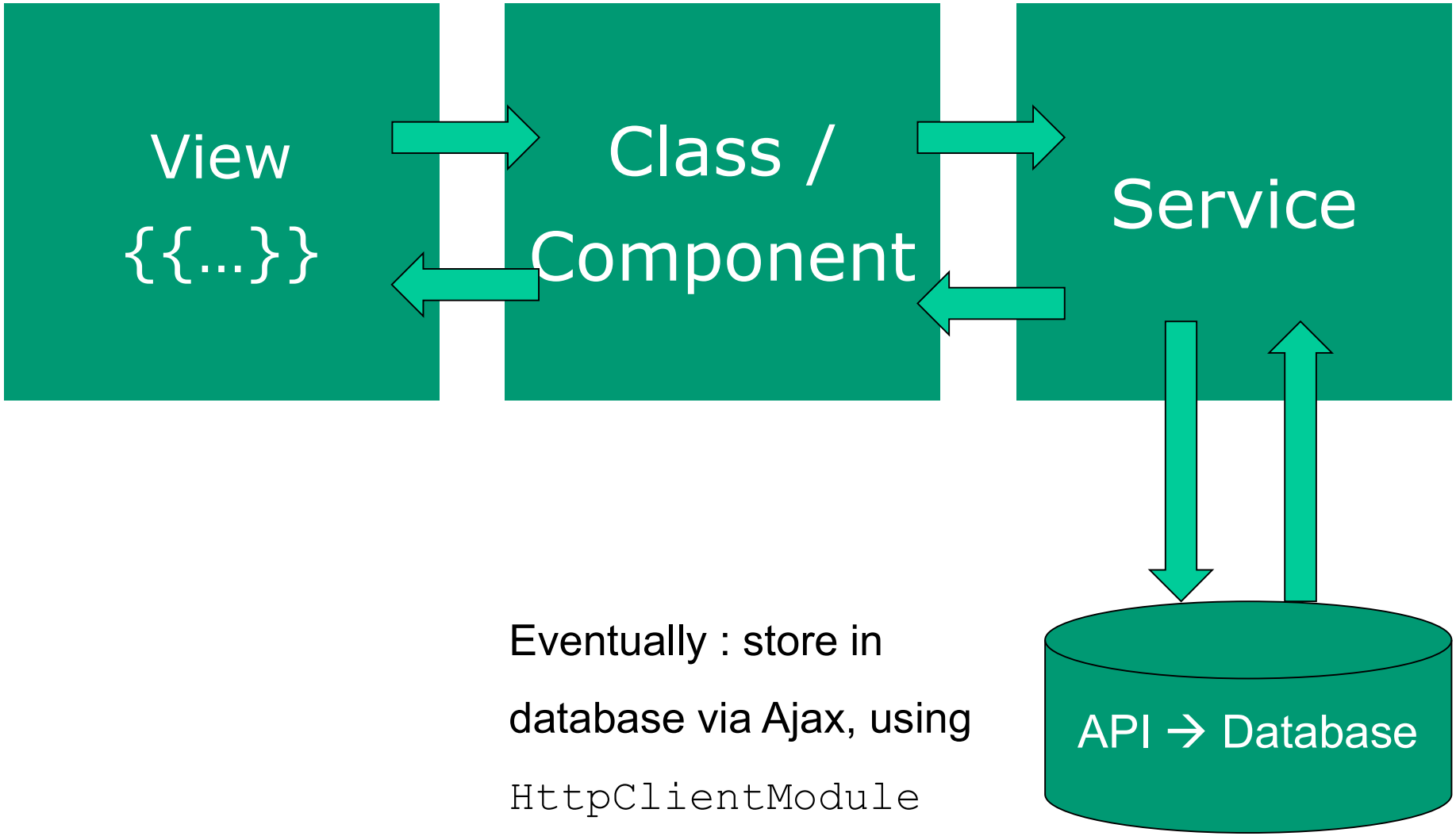
WORLDWIDE LOCATIONS

BELGIUM CANADA COLOMBIA DENMARK EGYPT FRANCE IRELAND JAPAN KOREA MALAYSIA MEXICO NETHERLANDS NORWAY QATAR
SAUDI ARABIA SINGAPORE SPAIN SWEDEN UNITED ARAB EMIRATES UNITED KINGDOM UNITED STATES OF AMERICA

Services

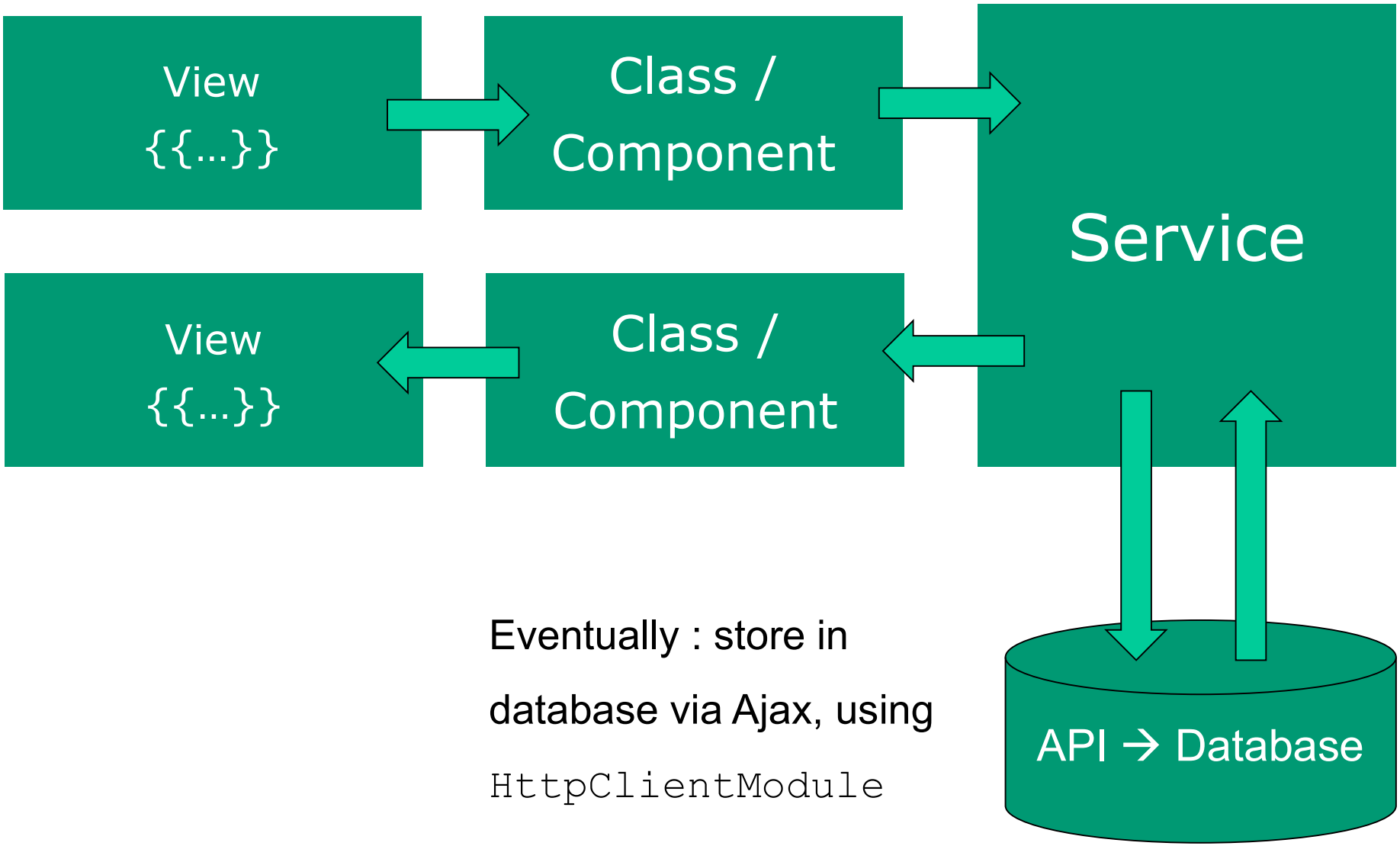
- Goal – *reuse* data functionality over different components
  - Data retrieval
  - Data caching
  - Data Storage,
  - …
- Angular: create a class without a UI (no template)
  - `export class myDataService { … }`

Data flow



Eventually : store in database via Ajax, using `HttpClientModule`

…and with multiple components



Services in Angular

Data services in Angular 1:

```
angular.module('myApp')
  .service(…)
  .factory(…)
  .provider(…)
```

Data services in Angular 2+:

```
import {Injectable} from '@angular/core';

@Injectable()
export class CityService{
   //....
}
```

Make sure to use @Injectable

Why? – Dependency Injection (DI) en metadata!

"TypeScript sees the @Injectable() decorator and emits metadata about our service, metadata that Angular may need to inject other dependencies into this service."

https://angular.io/docs/ts/latest/tutorial/toh-pt4.html

But...

"Our service doesn't have any dependencies at the moment. Add the decorator anyway.

It is a best practice to apply the `@Injectable()` *decorator from the start both for consistency and for future-proofing"*

Creating a service

Creating a service in 3 steps

Creating a service – 3 steps

1. Create/generate your service
2. Consume/inject service into component
3. Make service available in the module
    - NOT necessary when using modern notation (see further)

Creating a service

```
ng generate service [name]
```

Step 1 – create service (static data)

```
import { Injectable } from '@angular/core';
import { City } from './city.model'

@Injectable()
export class CityService {
   private cities:City[] = [
      new City(1, 'Groningen', 'Groningen'),
      …
   ];

   // return all cities
   getCities() {
      return this.cities
   }

   // return city based on id
   getCity(id:number) {
       return this.cities.find(c => c.id === id);
   }
}
```

Step 2 – Inject/consume service – using constructor injection

```
…
import {CityService} from "./city.service";

@Component({
   selector    : 'hello-world',
   templateUrl: 'app/app.component.html',
})

export class AppComponent implements OnInit {
   // Properties for component/class
   currentCity: City;
   cities: City[];
   cityPhoto: string;

   constructor(private cityService: CityService) {
   }

   ngOnInit() {
      this.cities = this.cityService.getCities();
   }

   getCity(city: City) {
      this.currentCity = this.cityService.getCity(city.id);
     ……
 }
}
```

local variables

Dependency Injection

Constructor: shorthand to instantiate private variable

Call service method in ngOnInit()

Instantiation?

- Let op: geen `new()` instantie van de Service!
  - Services zijn Singletons
  - Worden opgehaald uit de Module en/*of* geïnstantieerd in een `constructor()`

```
constructor(private cityService:CityService) { … }
```

"The constructor itself does nothing.

The parameter simultaneously defines a private cityService property and identifies it as a CityService injection service."

Instantiation?

- Pay attention: no manual `new()` instance of Service!
  - Services are -often- Singletons
  - Are fetched from the Module and/or instantiated in

    `constructor()`

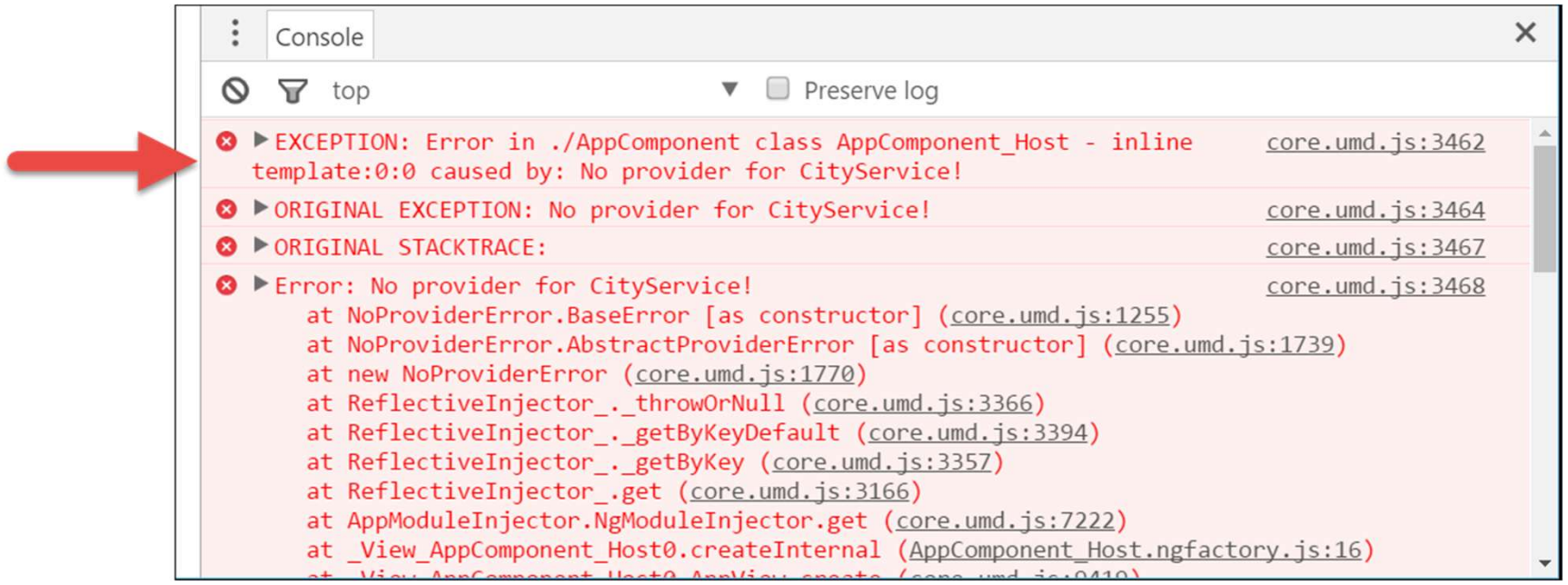
```
constructor(private cityService:CityService) { … }
```

"The constructor itself does nothing.

The parameter simultaneously defines a private `cityService` property and identifies it as a `CityService` injection service."

Error might occur: "No provider for CityService"

- Solution: inject in `app.module.ts`



Step 3, option 1 – Inject service in Module

Only an import/reference to `CityService` is not sufficient.

Angular has to *inject* the service in the module

Use the annotation `providers: [ … ]`

```
// Module declaration
@NgModule({
   imports      : [BrowserModule],
   declarations: [AppComponent],
   bootstrap    : [AppComponent],
   providers    : [CityService] // DI for service
})
export class AppModule {
}
```

Array with Service-dependencies

Step 3, option 2, modern: use `providedIn`

- "Tree shakeable providers"
- Don't tell the Module which services to use, the other way around:
- tell the service in which module it is used

```
@Injectable({
   providedIn: 'root'
})
export class CityService {
      …
}
```

```
@NgModule({
   imports      : [BrowserModule],
   declarations: [AppComponent],
   bootstrap    : [AppComponent],
   // providers    : [CityService]
})
```

https://blog.angular.io/version-6-of-angular-now-available-cc56b0efa7a4

Traditional – constructor based DI

- Not Wrong! This is still valid in Angular 17+

```
export class InjectComponent {

  // using classic constructor based Dependency Injection:
  constructor(public userService: UserService) {
  }

  //…
}
```

Modern Angular (17+) the `inject()` function

- Technically available since Angular 14
  - but now a lot more useful!
  - Ditching *Constructor based Dependency Injection*
- Technical benefits
  - Readability benefits – a lot
  - It's a lot more clearer what dependencies a component needs – even if you don't fully understand DI.
  - Inheritance becomes much simpler. No need to use `super(service1, service2)` … on `extends BaseComponent`

```
export class InjectComponent {
  // using inject() based Dependency Injection
  userService = inject(UserService);
}
```

Warning! `Inject` only during constructor-time

- Note: the `inject()` function only works when constructing the component
- So you CAN NOT use `inject()` in other methods

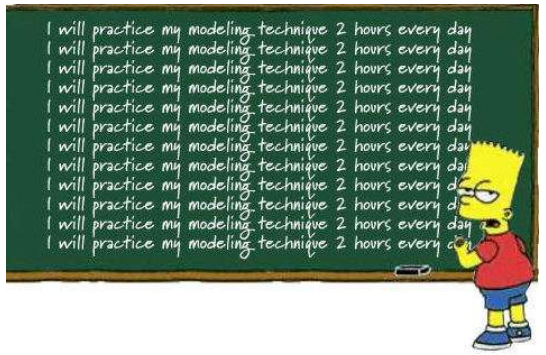
```
export class InjectComponent {

  // works - executed in construction phase of component
  userService = inject(UserService);

  ngOnInit(){
    // won't work, this function is executed later
    inject(SomeService).someValue
  }
  someMethod(){
    // won't work, this function is executed later
    inject(SomeService).someValue
  }
}
```

Singleton?

- Services are (usually) singletons
  - But: it depends where the service is provided/instantiated!
  - Services are singleton for Component/Module and all child components.
  - Using Module/Site-wide? (recommended)
    - Instantiate service in `app.module.ts`
    - `Or use { providedIn: 'root'}`

Workshop

- While working, you have developed some components, displaying data. Move this data from the component class to a service.
- Create a new file (for example `city.service.ts`) and import the correct annotations. Use `@Injectable()` to decorate the service.
  - 3 options: use `providers:[]`, constructor DI, or `inject()`.
- Write a `getCities()` method and methods to add or delete
  - or the data you are working with, of course.
- Example: `\200-services-static`



Checkpoint

- Every service in Angular is a `class`
- Use the `@Injectable()` decorator on service classes
- Dependency Inject in `constructor()` of the component that consumes/needs access to the service methods
- Add service to `providers: []` or use `providedIn`
- Modern Angular: use the `inject()` function in the component