Angular Fundamentals

Component Trees

Peter Kassenaar –
info@kassenaar.com

Angular Fundamentals Module – Component Trees

Peter Kassenaar
info@kassenaar.com

Angular Fundamentals
Module 5 – Component Trees

Peter Kassenaar
info@kassenaar.com

Angular Fundamentals

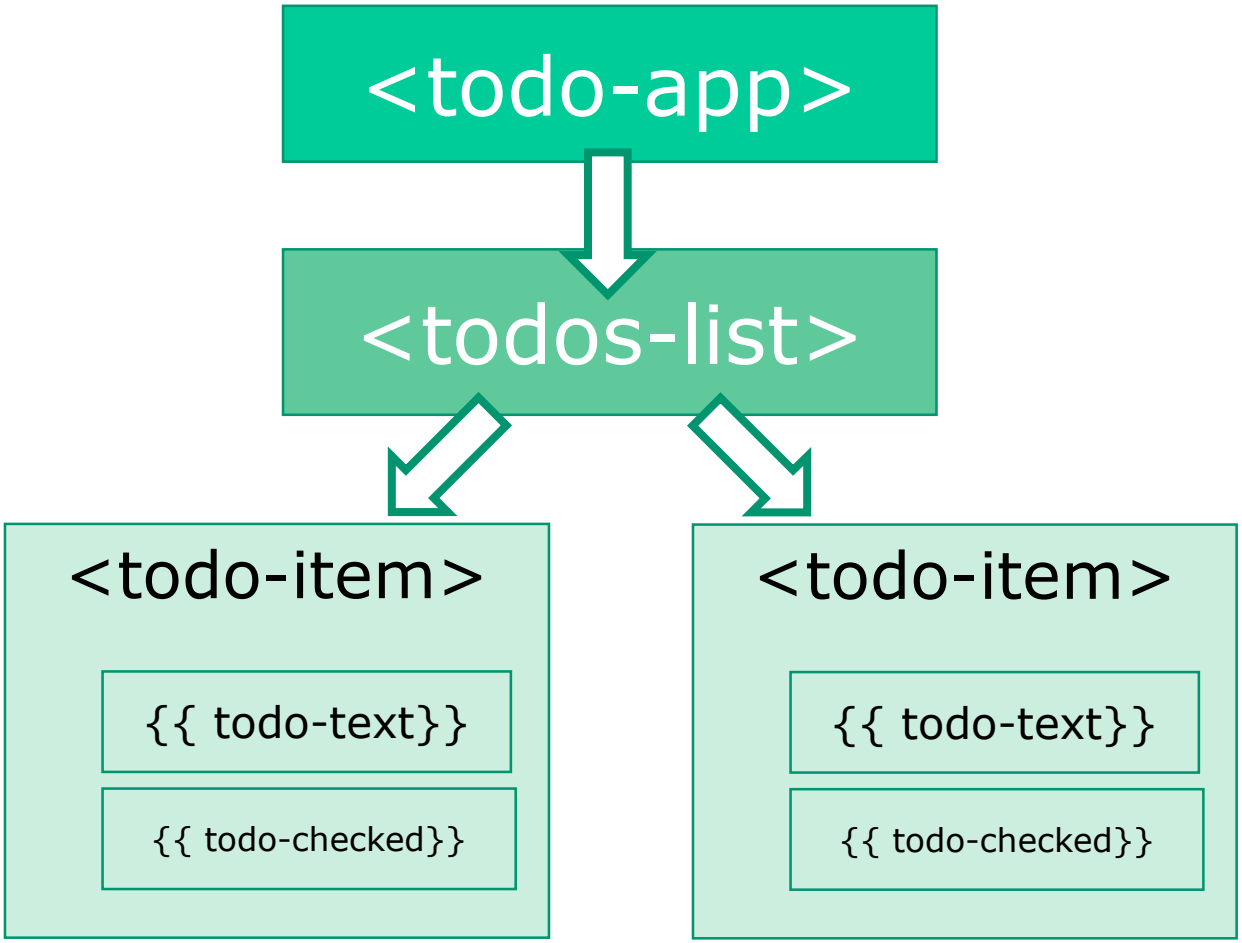
Module 5 – Component trees

Peter Kassenaar –
info@kassenaar.com

Hoofdstuk 7

p. 182 en verder

Angular-app: Tree of components



Application as a tree of components

- Recap - Multiple components?
  1. Create files manually – or let CLI handle this for you
     1. `ng generate component <component-name>`
     2. `ng g c <component-name>`
  2. Import in module – or (again) let CLI take care of this for you
  3. Add to `declarations : […]` section of `@ngModule`.
     1. IF you are working with ng modules, that is. Otherwise:

        import in component where you want to use it.
  4. Add via HTML to `parent`-component
- Repeat for every component

1. Add Detail component

```
// city.detail.ts
import { Component } from '@angular/core';

@Component({
   selector: 'city-detail',
   template: `
   <h2>City details</h2>
      <ul class="list-group">
         <li class="list-group-item">Naam: [naam van stad]</li>
         <li class="list-group-item">Provincie: [provincie]</li>
         <li class="list-group-item">Highlights: [highlights]</li>
      </ul>
   `
})

export class CityDetail{

}
```

Nieuwe selector

Nog in te vullen

2. Declaration in Module

```
// Angular Modules
…

// Custom Components
import {AppComponent} from './app.component';
import {CityDetail} from './city.detail';
import {CityService} from "./city.service";

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

Nieuwe component

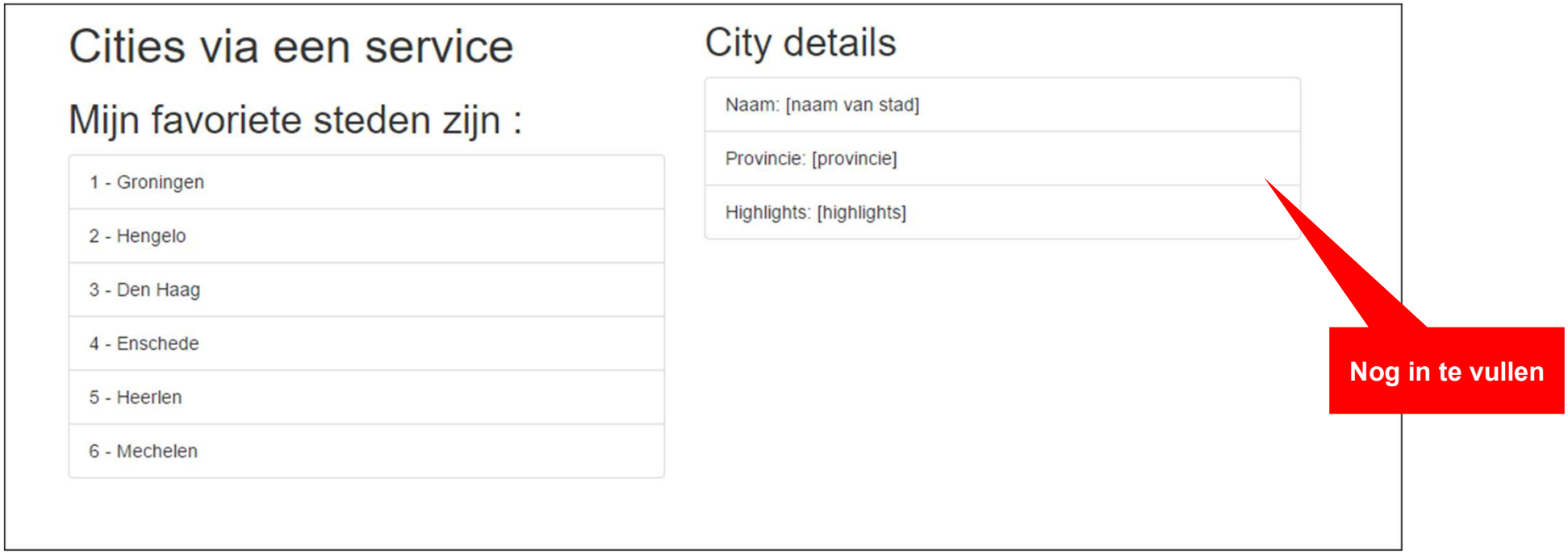
Toevoegen aan declarations: [ ]

3. Enclose in HTML

```
<!-- app.html -->
<div class="row">
    …
   <div class="col-md-6">
      …
      <city-detail></city-detail>
   </div>
</div>
```

Combineren met overige HTML

4. Result



Goal: show details of selected city in child-component

Data flow between components

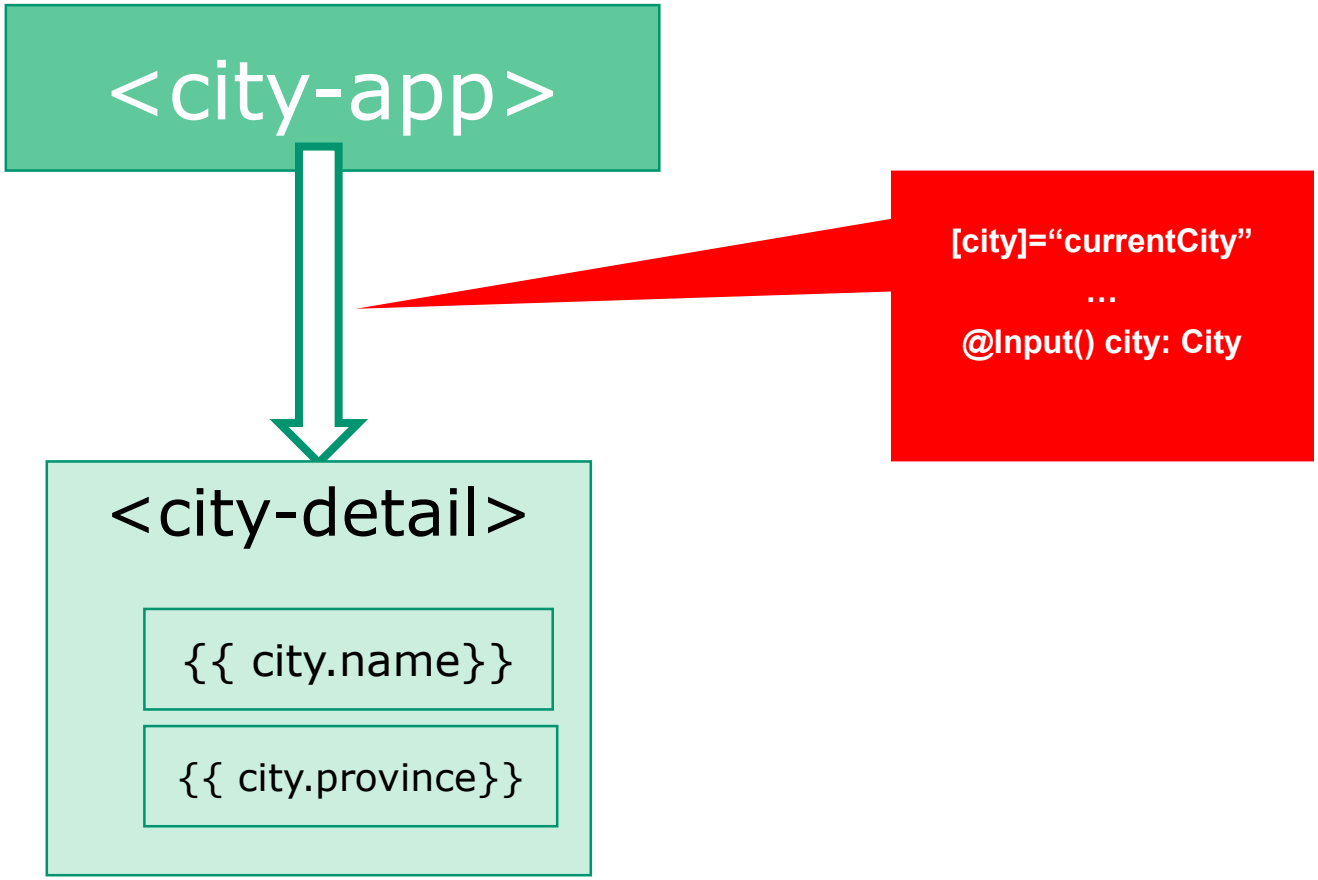
Using @Input()’s and @Output()’s

Data flow between components

"Data flows in to a component via `@Input()` *'s"*

Data flows out of a component via `@Output()`*'s"*

Parent-Child flow: decorator `@Input()`



Using @Input()

1. Import `Input` in component
2. Use decorator `@Input()` in class definition

```
// city.detail.ts
import { Component, Input } from '@angular/core';
import { City } from "./city.model";

@Component({
   …
})

export class CityDetail {
   @Input() city: City;
}
```

Input

Update Parent Component to send @Input

```
<!-- app.html -->
<div class="row">
   <div class="col-md-6">
      …
      <ul class="list-group">
         <li *ngFor="“let city of cities" class="list-group-item"
                (click)="getCity(city)">
            {{ city.id}} - {{ city.name }}
         </li>
      </ul>
      <button *ngIf="currentCity" class="btn btn-primary"
                (click)="clearCity()">Clear</button>
   </div>
   <div class="col-md-6">
   <div *ngIf="currentCity">
         <city-detail [city]="currentCity"></city-detail>
      </div>
   </div>
</div>
```

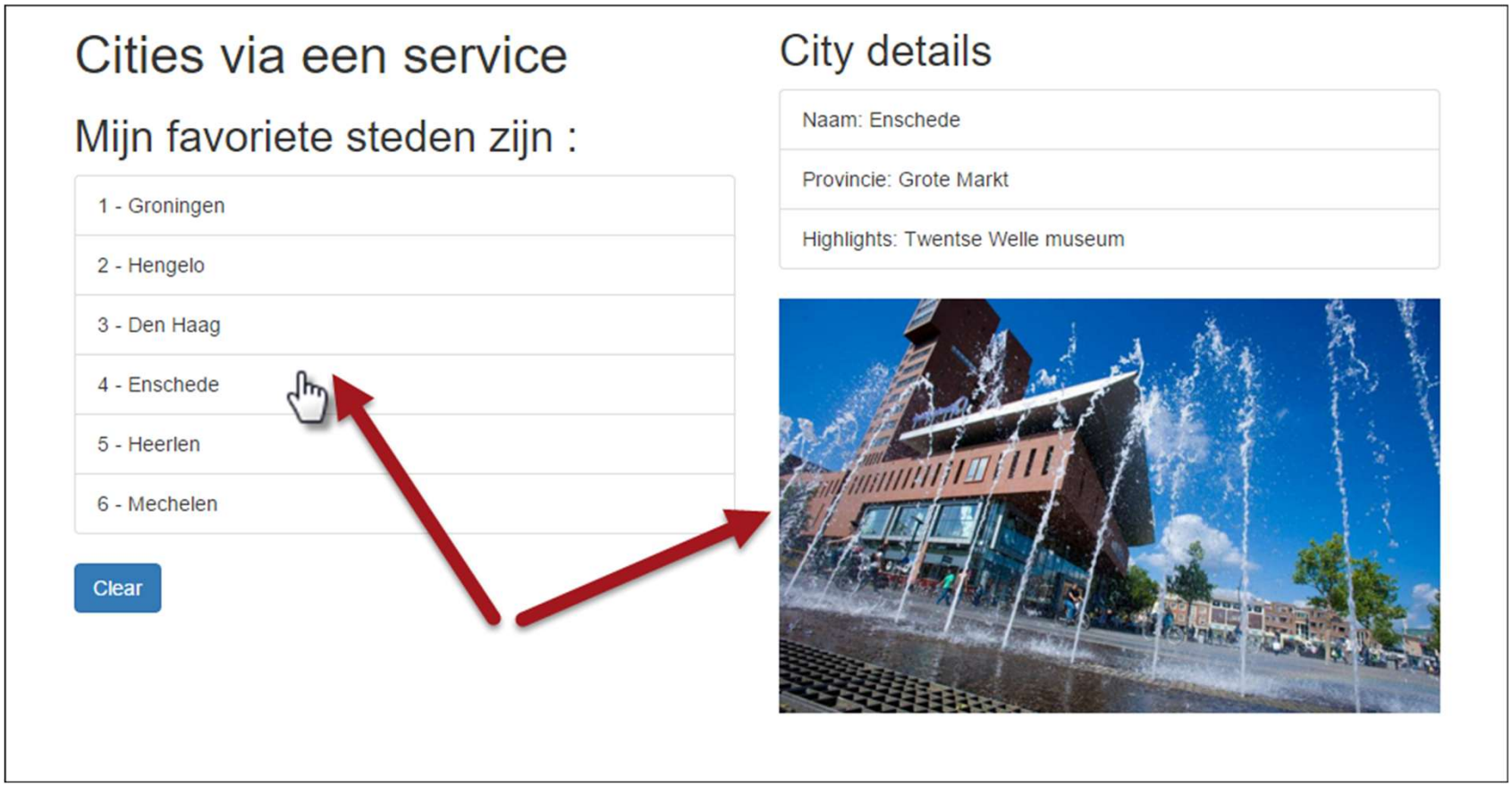
Aanpassing

Aanpassing!

Extend Parent Component Class

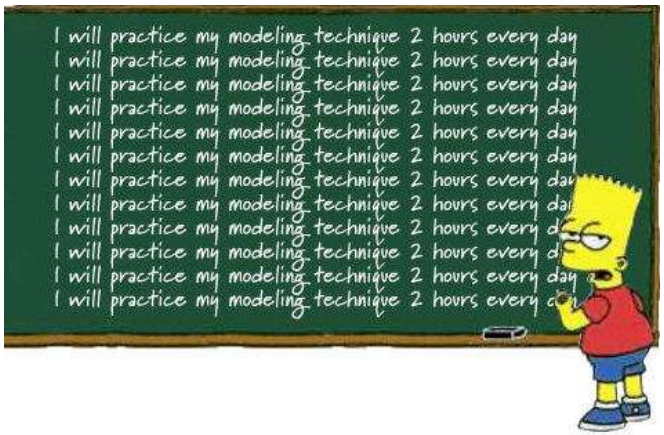
```
export class AppComponent {
   // Properties voor de component/class
   public cities:City[];
   public currentCity:City;

   …

   getCity(city) {
      this.currentCity = city;
   }

   clearCity() {
      this.currentCity = null;
   }

   …
}
```

Result

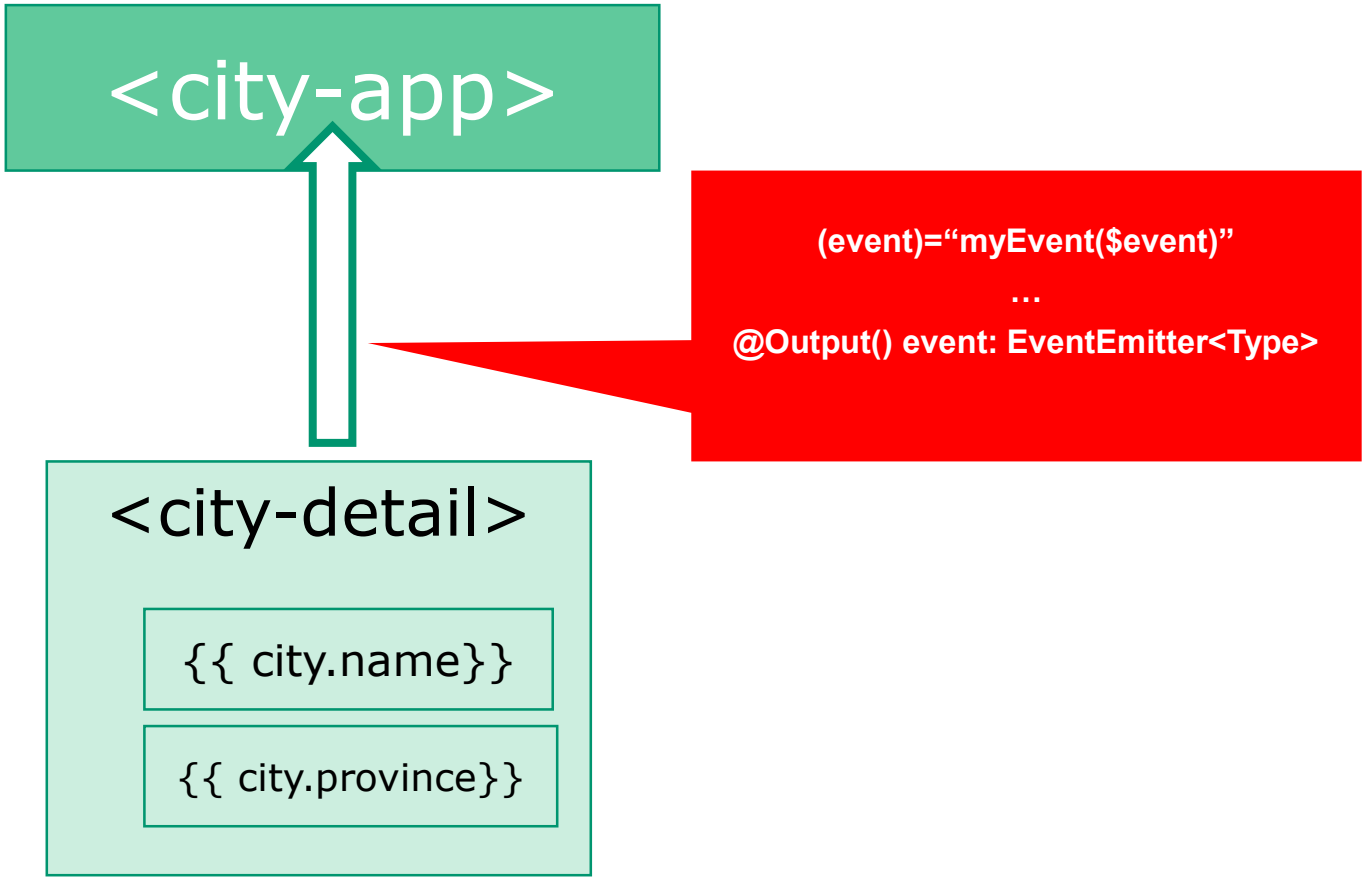

Cities via een service
Mijn favoriete steden zijn :
1 - Groningen
2 - Hengelo
3 - Den Haag
4 - Enschede
5 - Heerlen
6 - Mechelen
Clear
City details
Naam: Enschede
Provincie: Grote Markt
Highlights: Twentse Welle museum

Workshop

- Update your app so functionality is distributed over parent and child components
  - Rembember: components can be placed inside other components
  - Enhance the HTML of the Parent Component with selector of the Child Component
  - Remember to import the Child Component in `@ngModule`
- Data flow to Child Component : use `@Input()` and `[propName]="data"`
- Example: `/300-components`



Child-Parent flow: de annotatie `@Output()`



Method – equally, but the other way around

1. Import `Output` in component
2. Use decorator `@Output()` in class definition
3. New: define `EventEmitter` to emit events of certain type

"With @Output,

data flows up the Component Chain"

Rating our cities

```
// city.detail.ts
import { Component, Input, Output, EventEmitter} from ’@angular/core';

@Component({
   …
   template: `
   <h2>City details
      <button (click)="rate(1)">+1</button>
      <button (click)="rate(-1)">-1</button>
   </h2>
   …
   `
})

export class CityDetail {
   @Input() city:City;
   @Output() rating: EventEmitter<number> = new EventEmitter<number>();

   rate(num) {
      console.log('rating for ', this.city.name, ': ', num);
      this.rating.emit(num);
   }
}
```

Imports

Bind custom events to DOM

Define & handle custom @Output event

Prepare parent component for custom event

```
<!-- app.html -->
<div *ngIf="currentCity">
   <city-detail [city]="currentCity" (rating)="updateRating($event)">
   </city-detail>
</div>
```

Capture custom event

```
// app.component.ts
// increase or decrease rating on Event Emitted
updateRating(rating){
   this.currentCity.rating += rating;
}
```

Show rating in HTML

```
<li *ngFor="let city of cities"
    class="list-group-item" (click)="getCity(city)">
   {{ city.id}} - {{ city.name }} ({{i}})
   <span class="badge">{{city.rating}}</span>
</li>
```

Rating

Result

Cities via een service

Mijn favoriete steden zijn :

1 - Groningen	0
2 - Hengelo	0
3 - Den Haag	-3
4 - Enschede	0
5 - Heerlen	2
6 - Mechelen	5

Clear

City details +1 -1

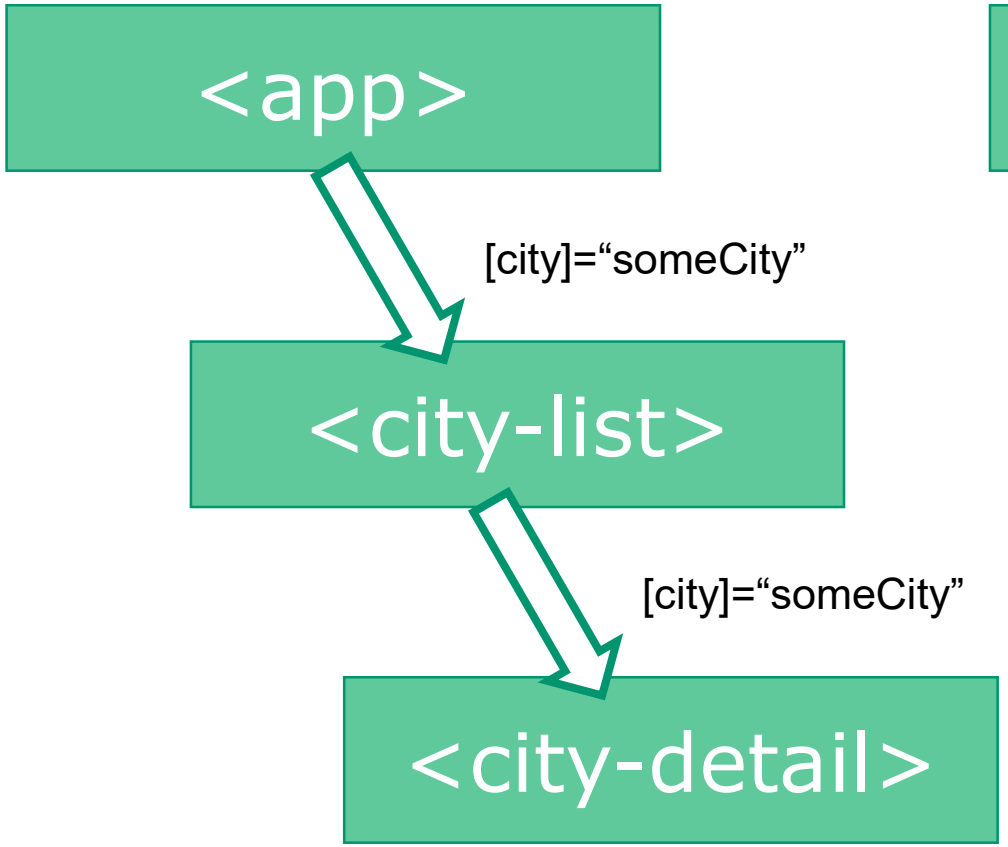
Naam: Den Haag

Provincie: Zuid-Holland

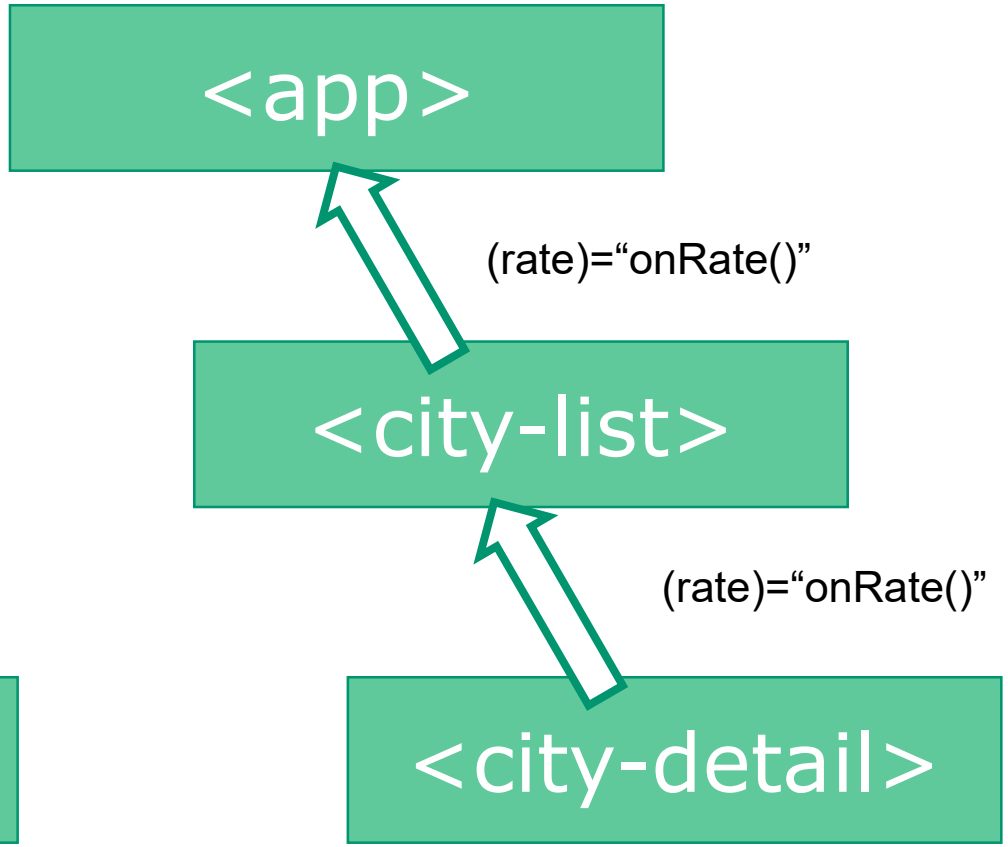
Highlights: Binnenhof

Summary

Parent → Child

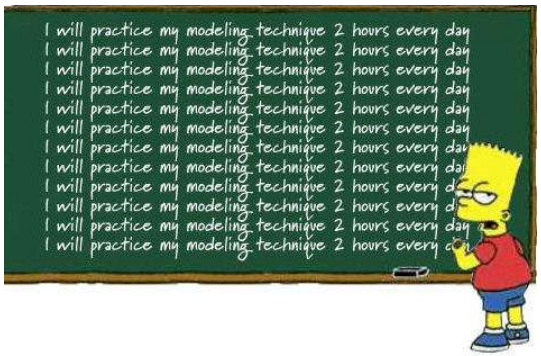


Child → Parent



Workshop

- Create a ‘favorite’ item in your child component
  - Parent component must react if an element is favorited.
- Remember: data flow to Parent Component : using `@Output()` and `(eventName)="eventHandler($event)"`
- You can throw *any type* of data with the `EventEmitter`.
- Example: `/302-components-output`
- More info: https://vsavkin.com/the-core-concepts-of-angular-2-c3d6cbe04d04

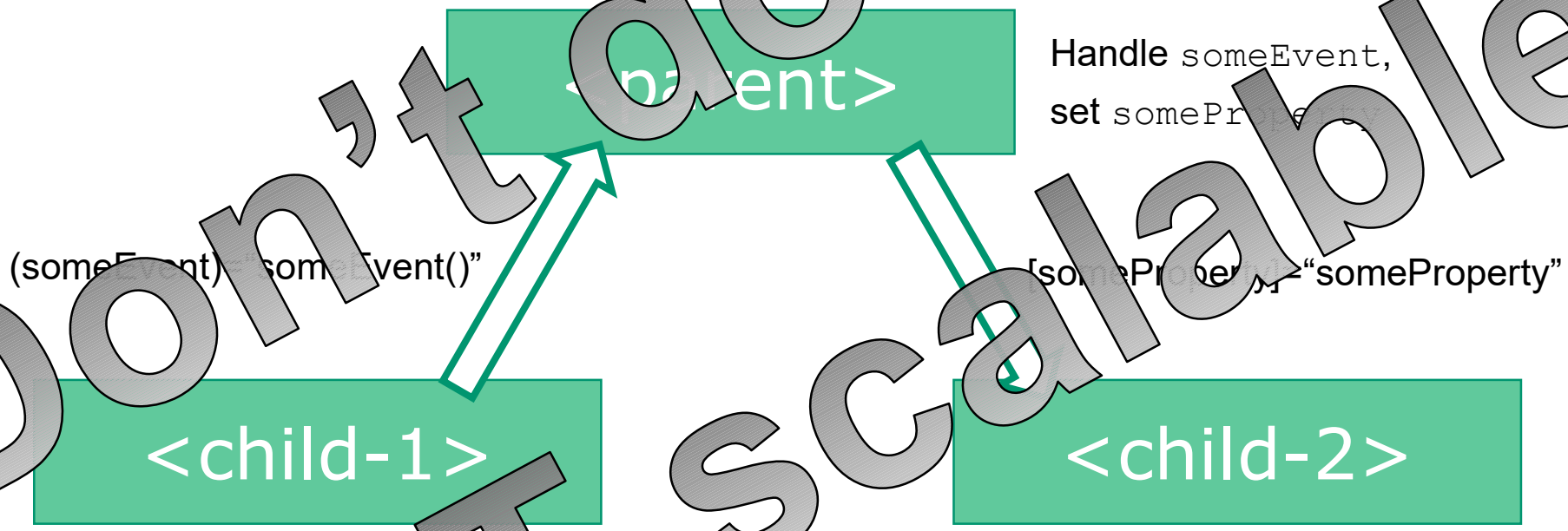


Sibling communication

Geen directe parent-child relatie tussen componenten

Sibling communication

- Pass `Output()` of childcomponent, to `@Input()` of other childcomponent



Don't do this!

NOT scalable

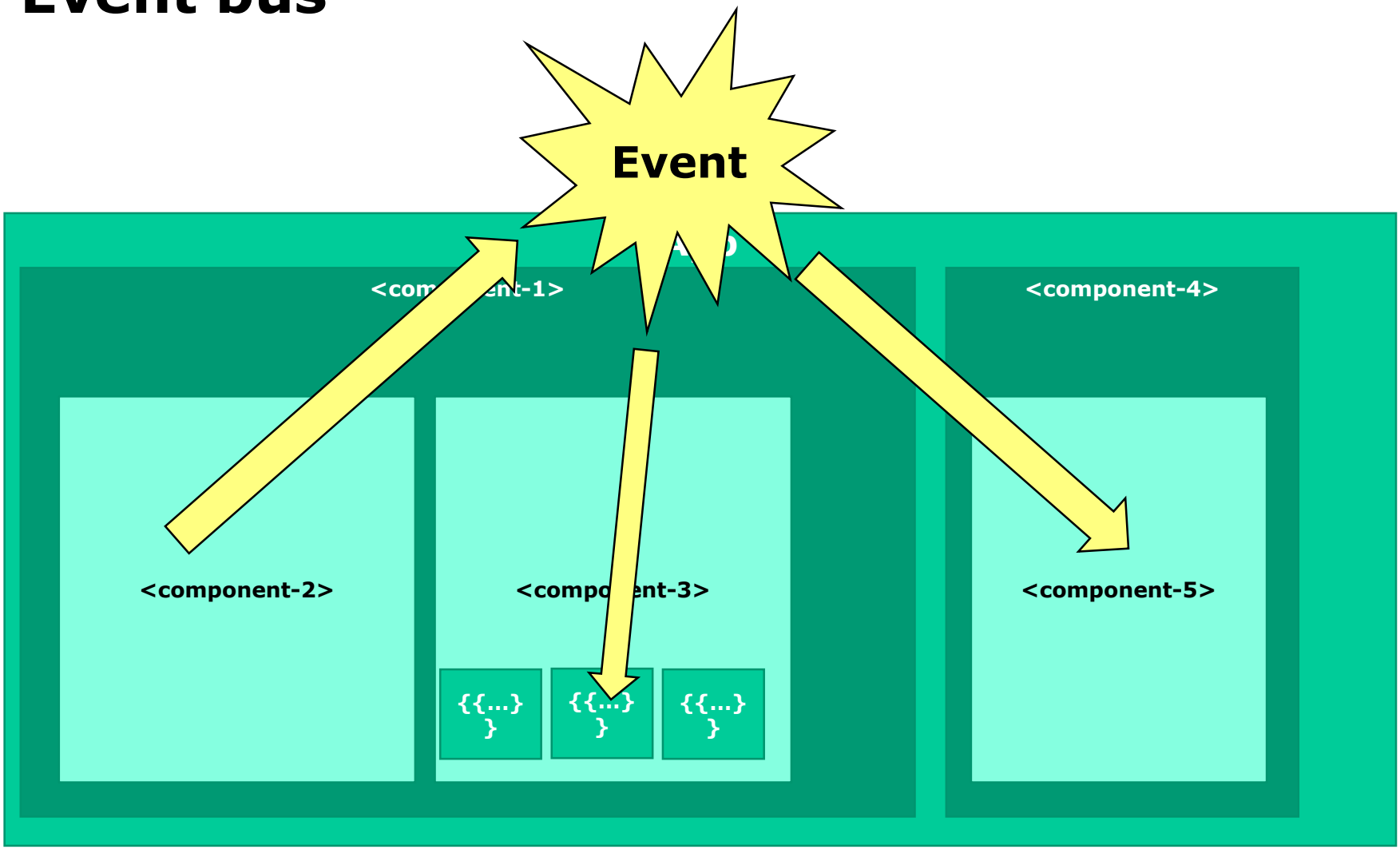
Better solution – using a Pub/Sub-system with Observables

- http://www.syntaxsuccess.com/viewarticle/pub-sub-in-angular-2.0

"Custom events,

write an event bus"

Event bus


Event
<component-1>
<component-4>
<component-2>
<component-3>
<component-5>
{{...}
}
{{...}
}
{{...}
}

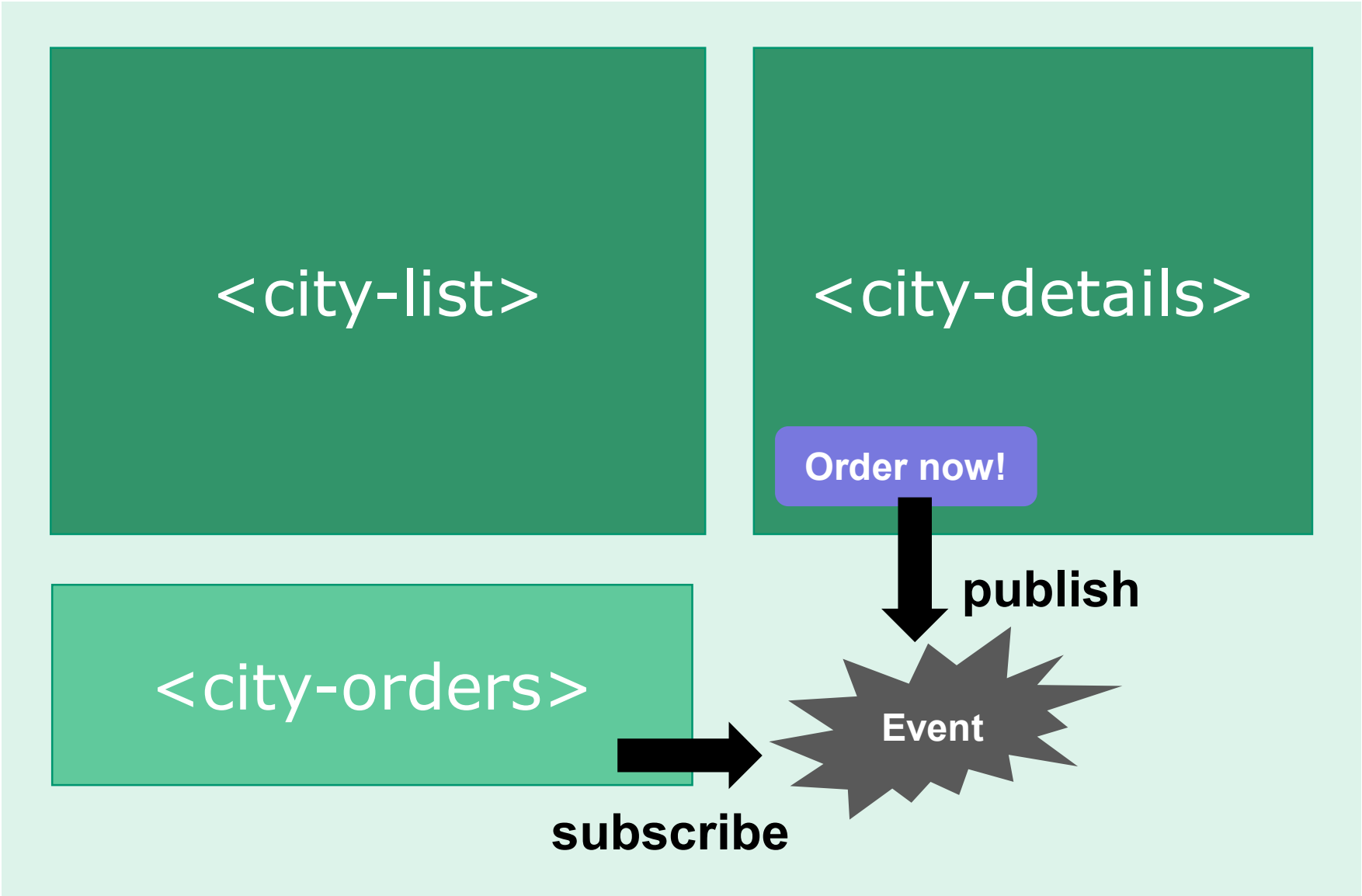
Options

From RxJS library, work with

- `EventEmitter()`
- `Observable()`
- `Observer()`
- `Subject<T>()` - implements `Observable` and `Observer`

"Publish and Subscribe"

PubSub design pattern


<city-list>
<city-details>
Order now!
publish
<city-orders>
Event
subscribe

Creating a PubSub-service

- Step 1 – create Publication Service
- Step 2 – Create ‘Producer’, or ‘Publish’ – component
- Step 3 – Create Subscriber-component

You can also see a PubSub-service with `Subject<T>()`

as a *“poor man’s state management solution”*

1. OrderService

```
// order.service.ts
import {Subject} from "rxjs/Subject";
import {Injectable} from "@angular/core";
import {City} from "../model/city.model";

@Injectable()
export class OrderService {
   Stream:Subject<City>;

   constructor() {
      this.Stream = new Subject<City>();
   }
}
```

2. **Producer component** (‘Order now’-button)

In de HTML:

```
<h2>Prijs voor een weekendje weg:
{{ city.price | currency:'EUR':true:'1.2' }}
<button class="btn btn-lg btn-info"
   (click)="order(city)">Boek nu!</button>
</h2>
```

In de class:

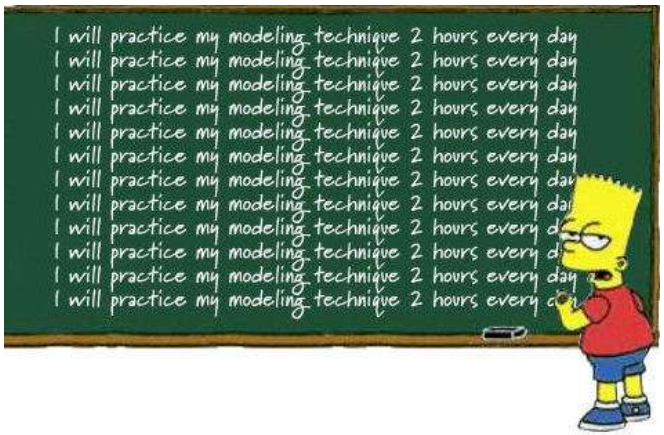
```
// Order plaatsen. Event emitten voor deze stad.
// Dit gaan opvangen in city.orders.ts
order(city) {
   console.log(`Stedentripje geboekt voor: ${this.city.name});
   this.orderService.Stream.next(city);
}
```

3. Subscriber component

```
// city.orders.ts – a kind of simple shopping cart,
// register which city trips are booked.
import …

@Component({
   selector: 'city-orders',
   template: `
   <div *ngIf="currentOrders.length > 0">
      …
})

export class CityOrders {
   …
   ngOnInit() {
      this.orderService.Stream
         .subscribe(
            (city:City) => this.processOrder(city),
            (err)=>console.log('Error handling City-order'),
            ()=>console.log('Complete...')
         )
   }
`    …
}
```

Workshop

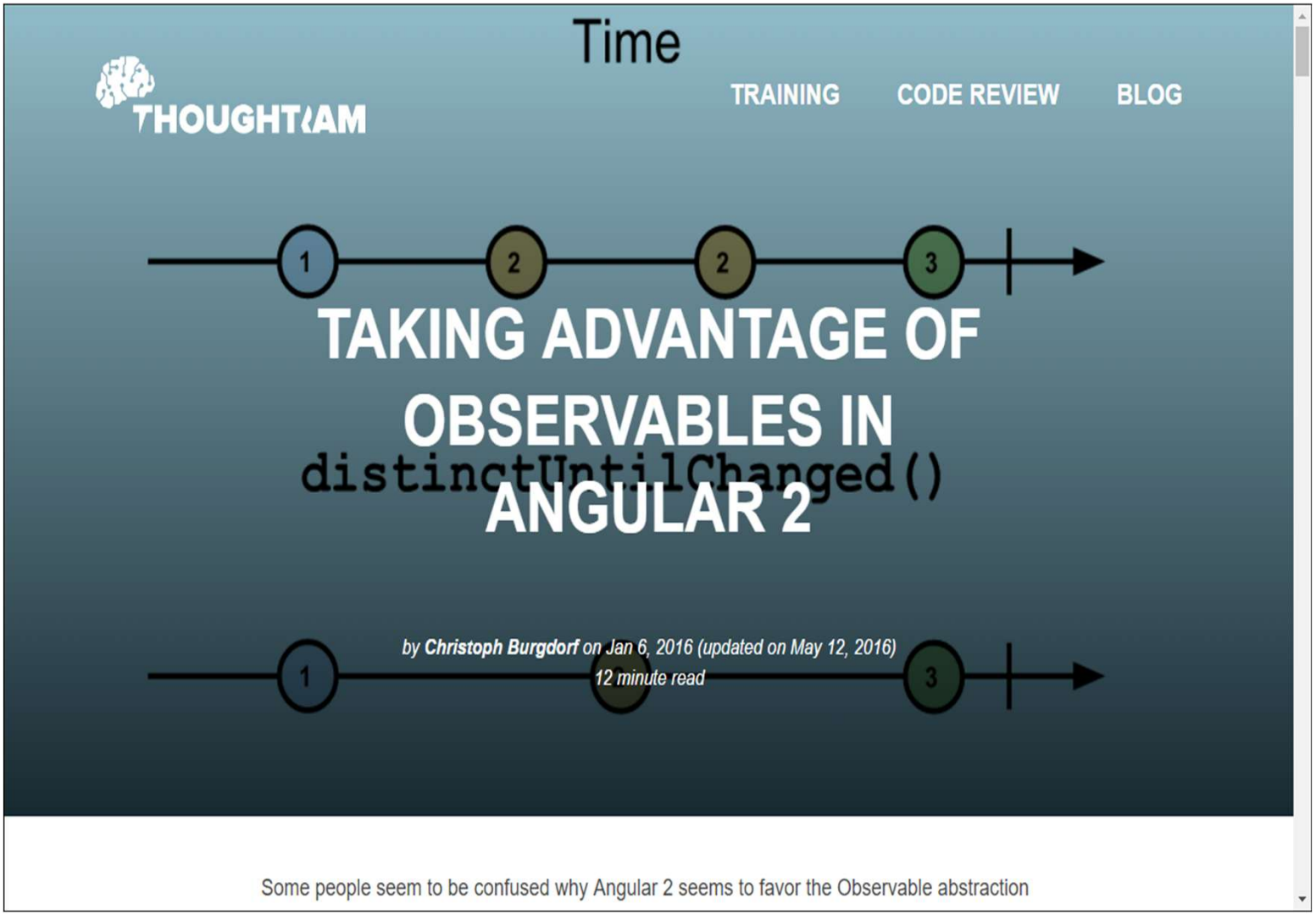
- Event Bus : work with 'invisible' Streams and `Subject`
- There are options on working with Observable Streams.
- Task: Create a simple e-commerce shopping website. It has:
  - A Store – 4 or 5 products (as static data)
  - Detail component – clicking on a product shows details
  - Shopping cart – user can place an item in their 'cart'
  - Order button – show total + receipt/price
- Example: `/303-pubsub-ordercomponent`
- (Optional/advanced: use a @ngrx/store state management solution)



More info

Elsewhere on the internet… more information on Observables and Stores.

Meer over Observables



http://blog.thoughtram.io/angular/2016/01/06/taking-advantage-of-observables-in-angular2.html

CORY RYLAN WORK CONTACT

My name is Cory Rylan, Senior Front End Engineer at Vintage Software and Angular Boot Camp instructor. I specialize in creating fast, scalable, and responsive web applications.

Follow @SplinterCode

Angular 2 Observable Data Services

Nov 17, 2015
Updated May 6, 2016 - 8 min read

Angular 2 brings many new concepts that can can improve our JavaScript applications. The first new concept to Angular is the use of Observables. Observables are a proposed feature for ES2016 (ES7). I wont go in depth into Observables but will just cover some of the high level concepts. If you want a introduction to Observables check out my screen cast.

INTRO TO RXJS OBSERVABLES AND ANGULAR 2

The rest of this post will cover more data and application state management in a Angular 2 application. At the time of this writing Angular is on version Beta 1. This post has been updated as of Beta 15. The syntax of how Observables and their

https://coryrylan.com/blog/angular-2-observable-data-services

ChatGPT - https://chatgpt.com/share/6826ea7d-ffcc-8006-937a-f6162e0a820f + more of course

ChatGPT 4o

Delen

🔄 Observable Basics

An **Observable** is a stream of data that you can subscribe to. It emits values over time—this can be anything: a HTTP response, a button click, user input, etc.

🧩 Types of Observables (based on use-case, not actual class types)

1. **Cold Observables**
   - Created fresh for each subscriber.
   - HTTP requests, timers, `fromEvent()`, etc.
   - Every subscription triggers the logic again.
2. **Hot Observables**
   - Shared source; subscribers tap into a live stream.
   - Subjects (like `BehaviorSubject`, `ReplaySubject`, etc).
   - Useful for shared state, live data.
3. **Unicast vs Multicast**
   - Unicast: One observer = one execution (cold).
   - Multicast: One producer shared by many (ho