WARMTEBOUW.

Angular Advanced - Smart components & View components

Peter Kassenaar
info@kassenaar.com

1. Content projection

2. Smart components / View Components

“In Angular, all components are equal. There is no sense of different ‘types’ of components”

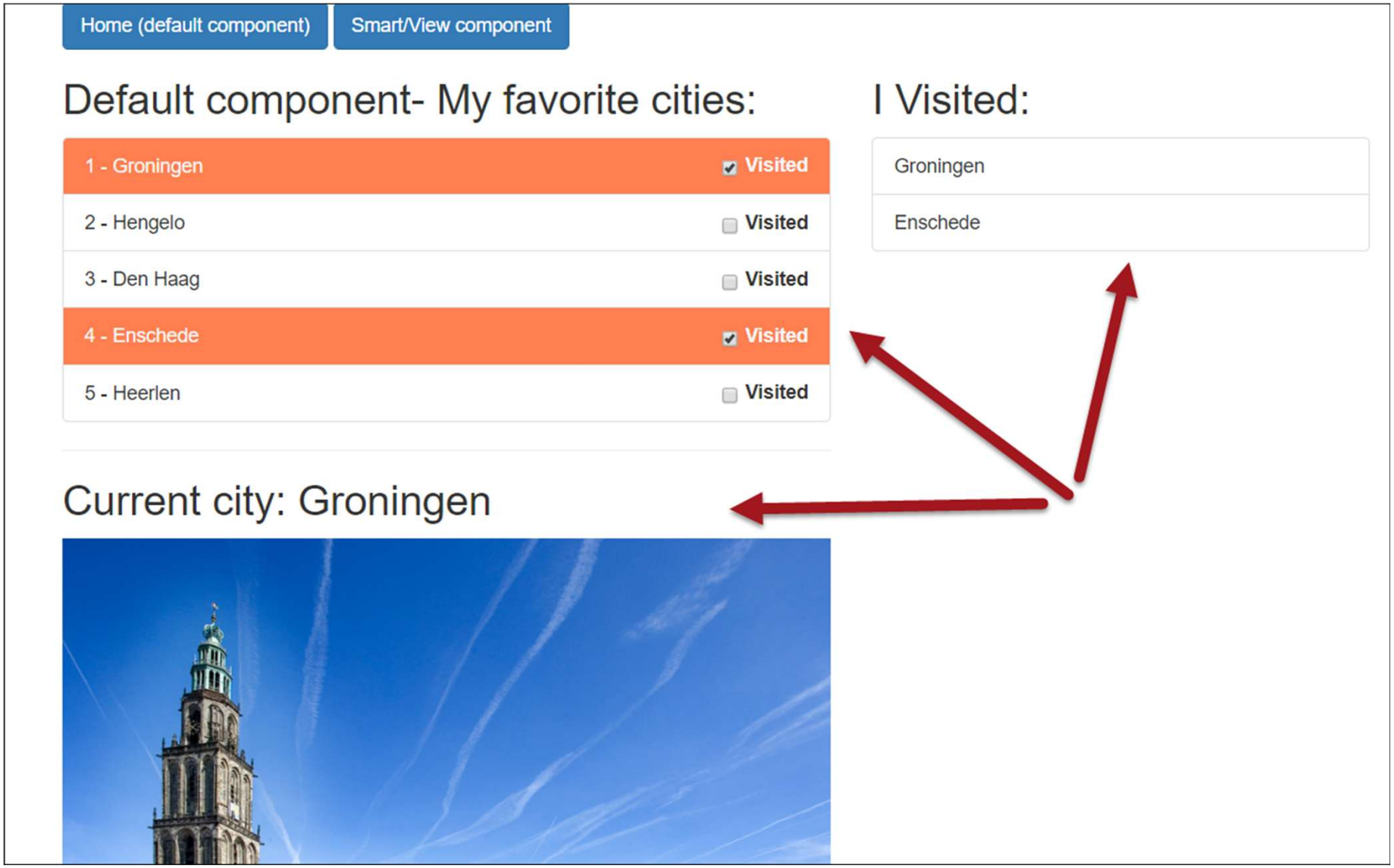
Smart components / View Components

- Design pattern
- Why ? Separation of concerns
  - View component is responsible for presentation (and *can* be used in a completely different environment with different component logic)
  - Smart component is responsible for logic → passing the [calculated] data to the view component.
- Smart components
  - AKA: *Container* components, *controller* components, *statefull* components
- View components
  - AKA: *pure* components, *dumb* components, stateless

Characteristics

- Smart components
  - Typically contain big(ger) chunks of logic
  - Typically have a small UI part (reference just the view component)
  - Pass the data to the view component
- View components
  - Typically contain no logic whatshowever
  - Get their data via `@Input()` decorators
  - Submit events via `@Output()` decorators
  - Have large chunks of UI

Example: ../300-smart-view-component



Home – Default component

- Contains all logic and UI, combined in one component.
  - `/home/home.component.html`
  - `/home/home.component.ts`
- This typically works well for smaller applications

```
<div class="row">
   <div class="col-md-6">
      <h2>Default component- My f
      <ul class="list-group">
         <li *ngFor="let city of
            class="list-group-ite
            [class.visited]="city
            (click)="getCity(city
            {{ city.id}} - {{ cit
            <span class="pull-rig
               <label>
      </ul>
..
</div>
```

```
@Component({
   selector    : 'app-home',
   templateUrl: './home.component.html'
})
export class HomeComponent implements OnInit {
   …
   ngOnInit() {
      this.cityService.getCities()
         .subscribe(cities => this.cities = cities);
   }

   getCity(city: City) {
      this.currentCity = city;
      this.cityPhoto   = `assets/img/${this.currentCity.name}.jpg`;
   }
   …
}
```

Splitting it up in view components

- `../smart-view/smart.component.html | .ts.`
- Passing `[cities]` in
- Getting `(events)` out.
- `[cities]` can also be an Observable

```
<div class="row">
  <div class="col-md-6">
    <h2>Smart/view component- My favorite cities:</h2>
    <!-- The <city-list>-component is now
         the view component for list of cities -->
    <city-list [cities]="cities"
               (selectCity)="getCity($event)"
               (toggleVisited)="toggleCity($event)">
    </city-list>
   …
  </div>
</div>
```

<city-list> View component

- Has no logic, just `@Input()`'s and `@Output()`'s

```
import {Component, EventEmitter, Input, Output} from '@angular/core';
import {City} from '../../shared/model/city.model';

@Component({
   selector   : 'city-list',
   templateUrl: './city-list.component.html'
})
export class CityListComponent {

   @Input() cities: City[];
   @Output() selectCity: EventEmitter<City>    = new EventEmitter<City>();
   @Output() toggleVisited: EventEmitter<City> = new EventEmitter<City>();

}
```

HTML of the view component

- …/`smart-view/city-list/city-list-component.html`
- Has a nested view component to toggle `visited` state
- Choice: events are directly emitted from the HTML
  - Can also be done via small functions

```
<ul class="list-group">
   <li *ngFor="let city of cities"
      class="list-group-item"
      [class.visited]="city.visited"
      (click)="selectCity.emit(city)">
      {{ city.id}} - {{ city.name }}
      <city-visited [visited]="city.visited"
                  (toggle)="city.visited = $event; toggleVisited.emit(city)">
      </city-visited>
   </li>
</ul>
```

`<city-visited>` View component

Again – just `@Input()`'s and `@Output()`'s.

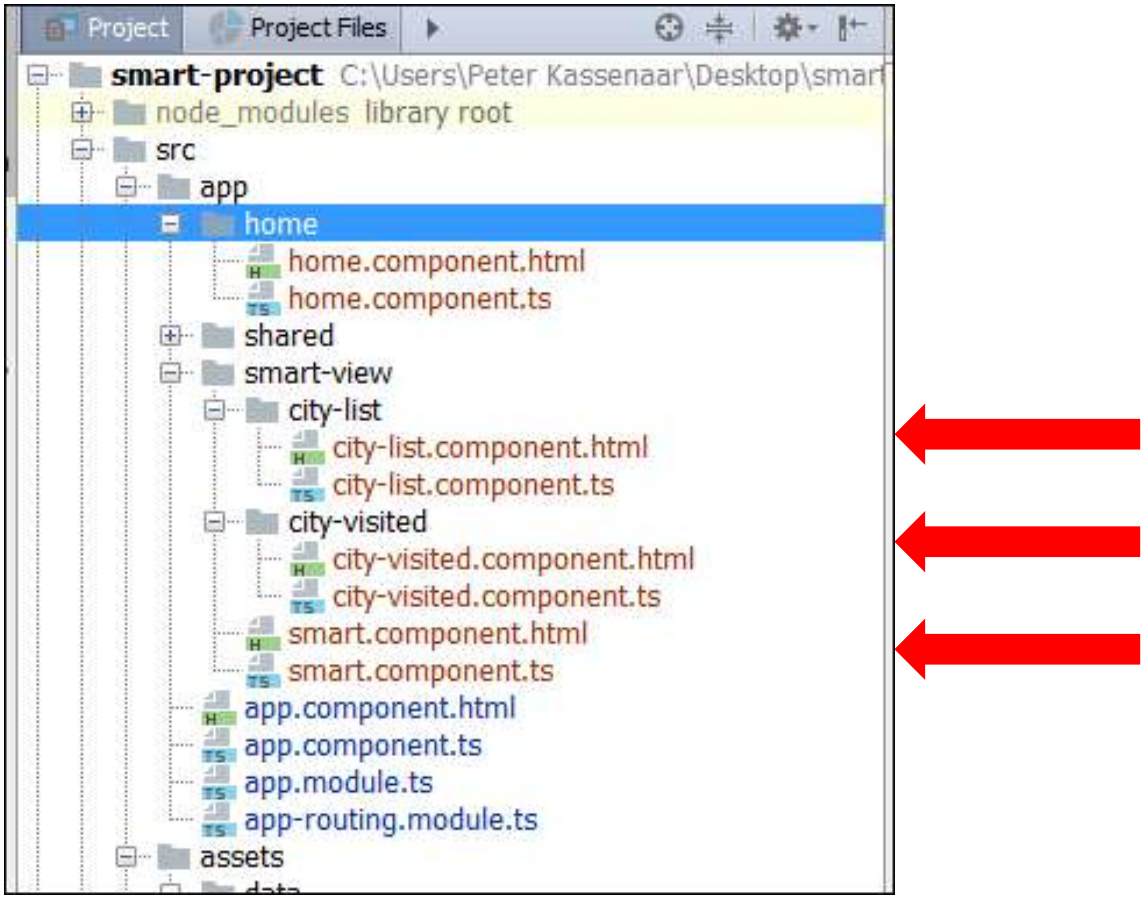
```
export class CityVisitedComponent {
  @Input() visited:boolean;
  @Output() toggle: EventEmitter<boolean>= new EventEmitter<boolean>();
}
```

HTML – again: attribute binding and event binding

```
<span class="pull-right">
   <label>
      <input type="checkbox"
            class="checkbox-inline"
            [checked]="visited"
            (change)="visited = !visited; toggle.emit(visited)"> Visited
   </label>
</span>
```

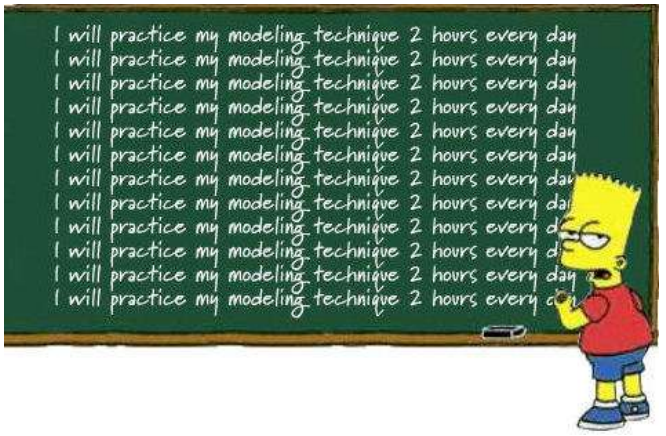
Application structure

- When following this pattern: typically more, smaller components
- The directory tree gets crowded!

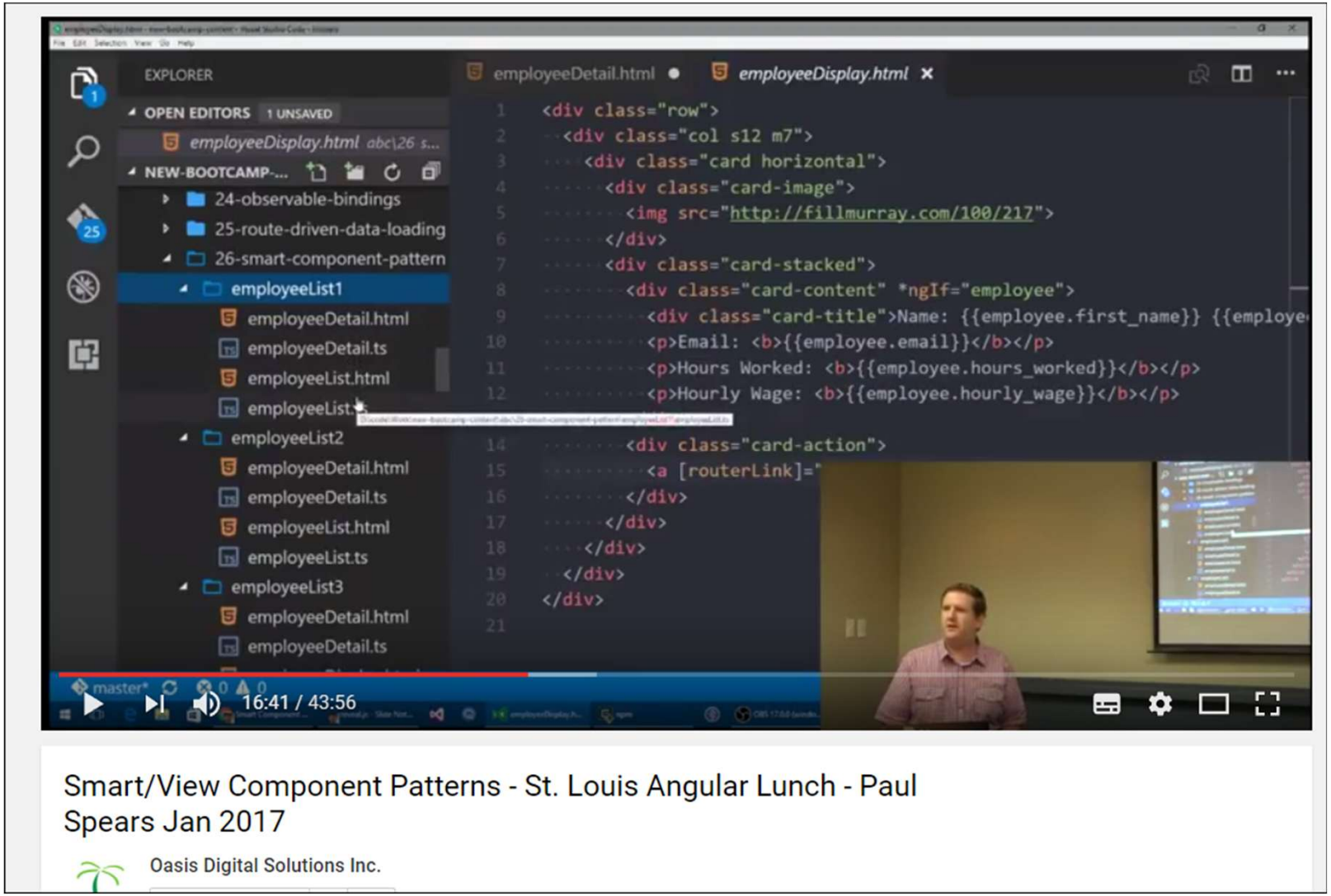


Workshop

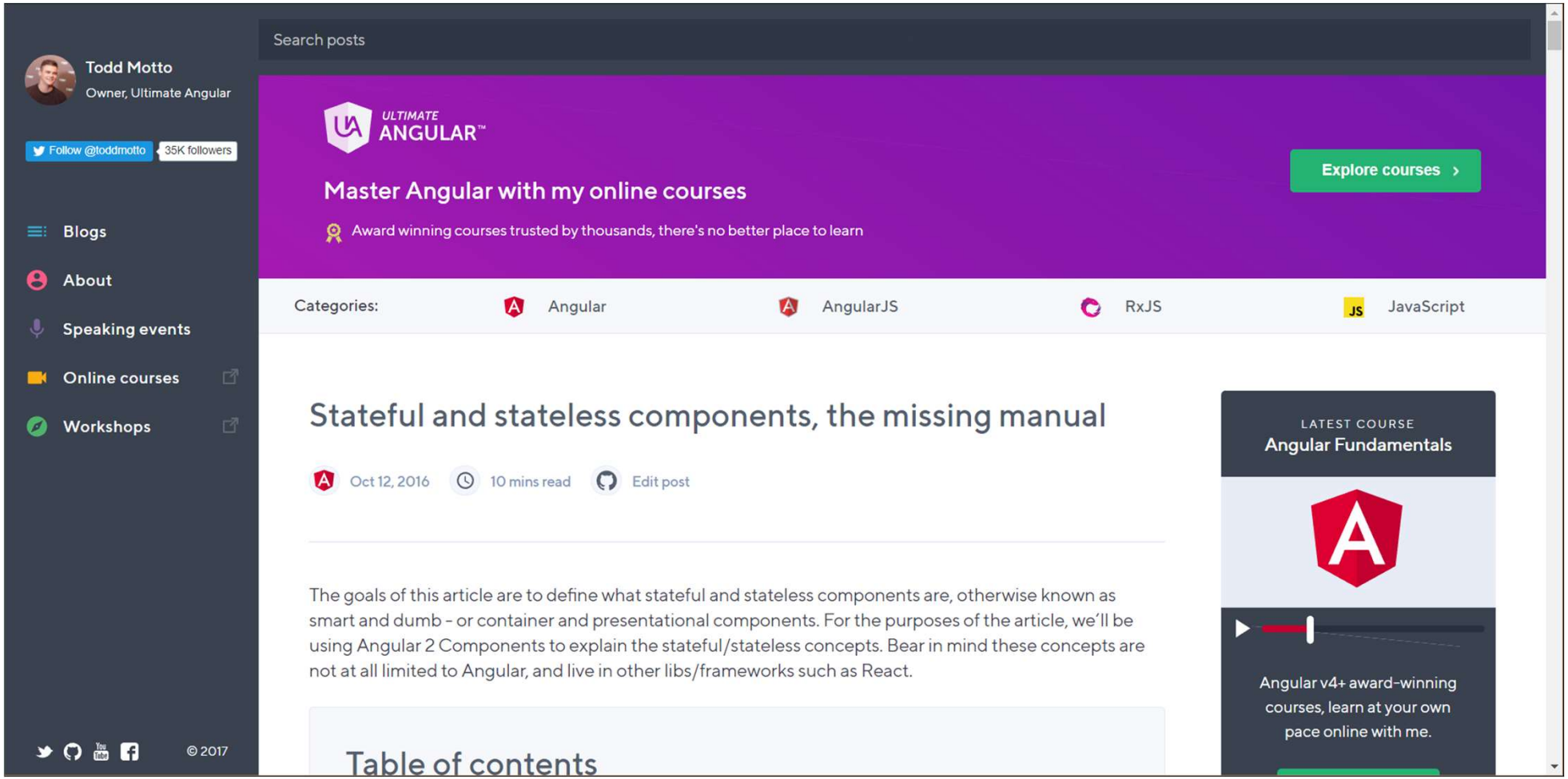
- Study the example `../300-smart-view-components`
- Create two additional view components:
  - One for displaying the current city
  - One for displaying the list of visited cities
- Optional: use Observables (or signals) instead of plain arrays.
  - `cities$ : Observable<City[]>`
  - `[cities]="cities$ | async"`



More info on Smart/View components

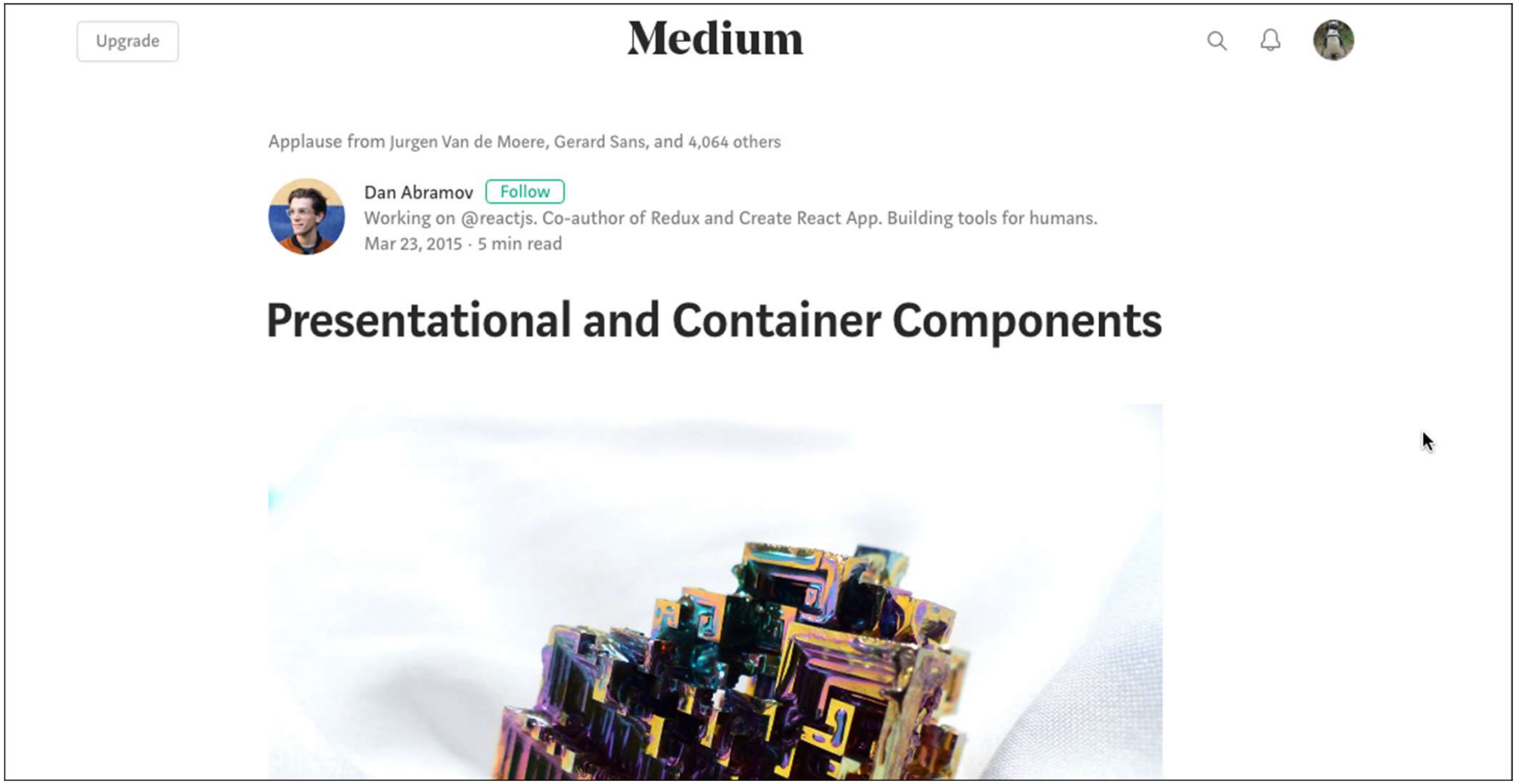


https://www.youtube.com/watch?v=ALm_JVdLT2E

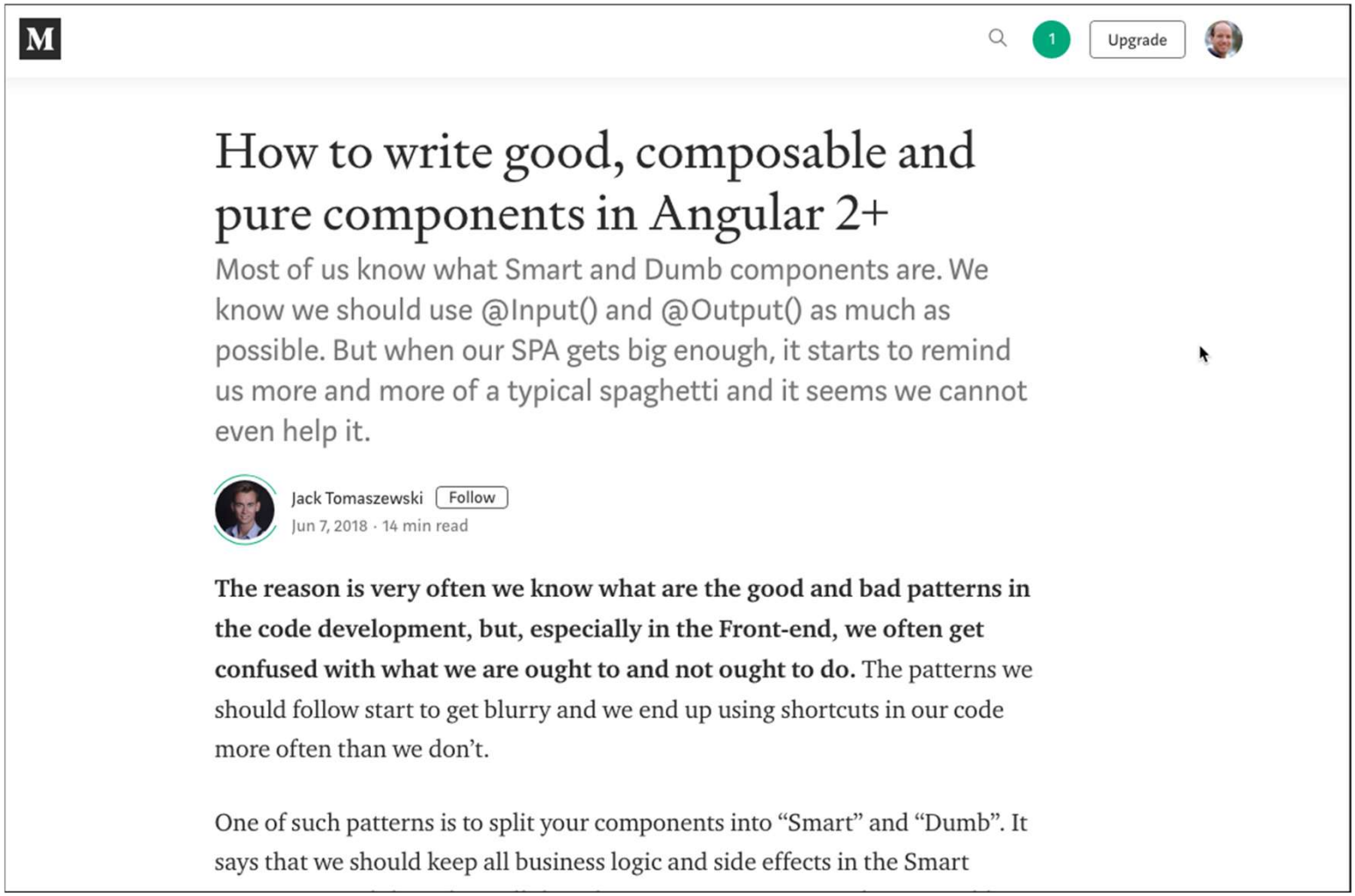


https://toddmotto.com/stateful-stateless-components

http://blog.angular-university.io/angular-2-smart-components-vs-presentation-components-whats-the-difference-when-to-use-each-and-why/



https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0



https://medium.com/@jtomaszewski/how-to-write-good-composable-and-pure-components-in-angular-2-1756945c0f5b