Tooltip with Angular CDK

Nikita Poltoratsky Follow
Feb 13 · 5 min read

A short while ago I published an article about the integration of Angular CDK in Nebular—full-featured library for Angular applications we're developing at Akveo.

During the development, we've faced a bunch of interesting puzzles that Angular CDK has helped us to overcome. That's why I decided to start a series of articles on challenges Angular CDK may aid you with.

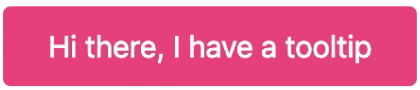
To start with, let's build a tooltip directive. It may sound like a simple component to build, but I believe it is a great showcase of a number of CDK features.

Table of contents

. . .

Introduction

For starters, let's take a step back and review what a tooltip component is. The main function of the tooltip is to show some text hint. Let's take a look at the following example:



And here is the usage example:

```
<span awesomeTooltip="Tooltip text">This is an
example</span>
```

. . .

Angular CDK Setup

Before we dive in, we need to set up the environment. Our tooltip is mainly built on top of Angular CDK features, so we need to install it first:

```
npm install @angular/cdk
```

To use `OverlayModule` we have to import `OverlayModule` in `AppModule`

```
import { OverlayModule } from '@angular/cdk/overlay';
@NgModule({
  imports: [ OverlayModule ],
})
```

And include overlay styles in the global stylesheet file:

```
@import '~@angular/cdk/overlay-prebuilt.css';
```

Now we have all the required functionality set up and ready to build the tooltip directive!

. . .

Building blocks

We're going to build tooltip as an Angular directive:

```
@Directive({ selector: '[awesomeTooltip]' })
export class AwesomeTooltipDirective {

  @Input('awesomeTooltip') text = '';

  @HostListener('mouseenter')
  show() { }

```

The directive manages the tooltip state. It listens to `mouseenter` and `mouseout` events and shows and hides tooltip as a reaction to the events.

The second thing we need to create is a component that will render the passed text:

```
@Component({
  selector: 'awesome-tooltip',
  template: `{{ text }}`,
})
export class AwesomeTooltipComponent {
  @Input() text = '';
```

Don't forget to add `AwesomeTooltipComponent` in `entryComponents`:

```
@NgModule({
  entryComponents: [AwesomeTooltipComponent],
})
export class AppModule {}
```

As we need to create `AwesomeTooltipComponent` dynamically in the runtime, we need to tell Angular compiler about that.

The next obvious step is to render `AwesomeTooltipComponent` with provided `text` input. Let's get to it.

. . .

Make tooltip floating

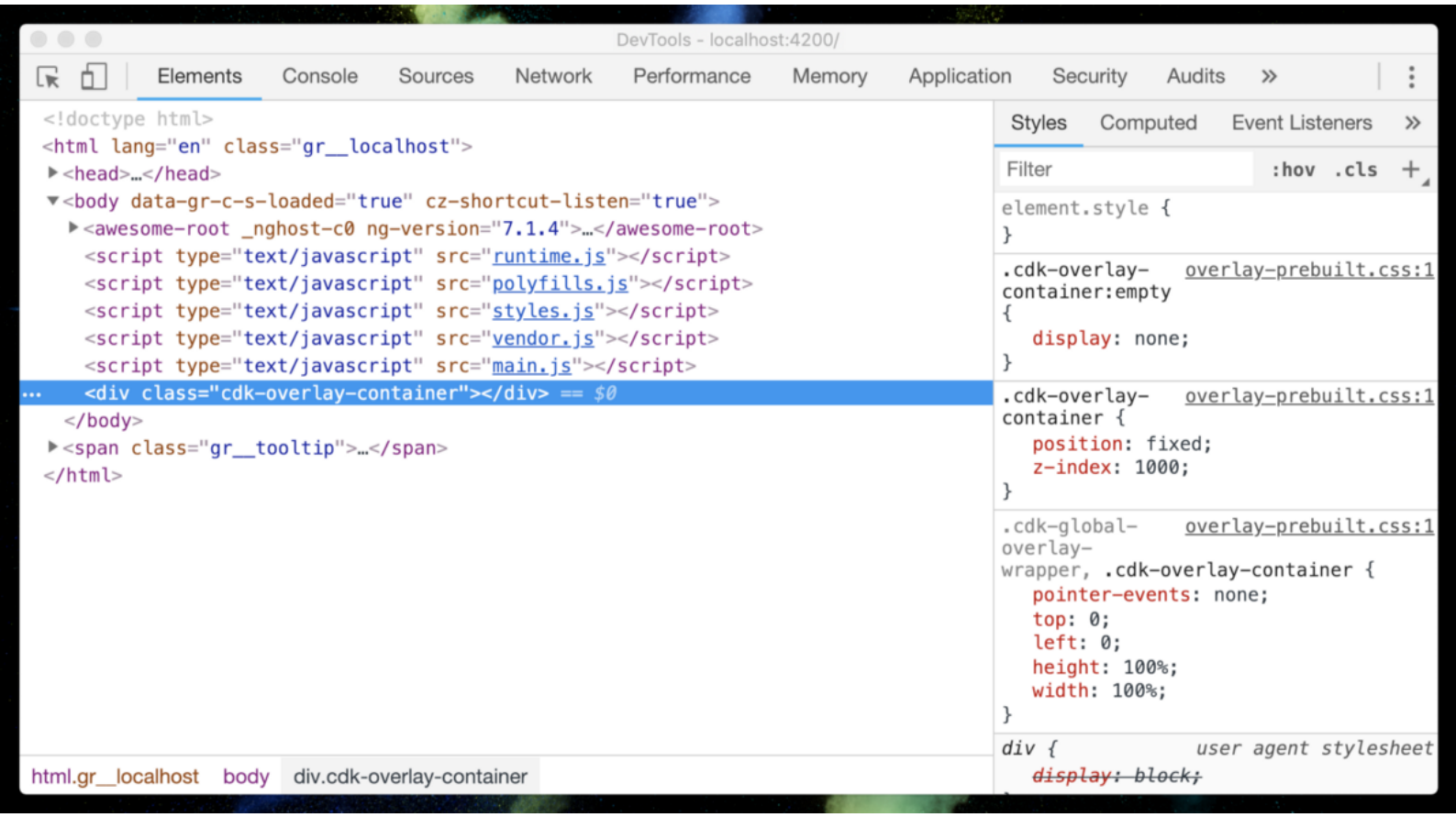
As we already know, we need to render the tooltip above some other component. Moreover, we have to make sure the tooltip is not cut off by the container component styles (`overlow: hidden` on the container could be a real pain here). So our best bet is to render the tooltip somewhere on top of the document tree and then position it properly. And CDK `Overlay` module would help us with that.

Firstly, we're injecting overlay service in tooltip directive and creating new overlay:

```
@Directive({ selector: '[awesomeTooltip]' })
export class AwesomeTooltipDirective implements OnInit

  private overlayRef: OverlayRef;

  constructor(private overlay: Overlay) {}

  ngOnInit() {
```

By calling `this.overlay.create()` we're creating somewhat called `OverlayRef` . You may think of it as of a remote controller, which allows us to insert some dynamically created component somewhere on top of the document tree.

During the instantiation, it creates the `div.cdk-overlay-container` element that serves as a container root for all components inserted into it.



Now let's complete `show` and `hide` methods with some CDK functions to make the magic work:

```
@Directive({ selector: '[awesomeTooltip]' })
export class AwesomeTooltipDirective implements OnInit

  private overlayRef: OverlayRef;

  constructor(private overlay: Overlay) {}

  ngOnInit(): void {
    this.overlayRef = this.overlay.create({});
  }

  @HostListener('mouseenter')
  show() {
    // Create tooltip portal
    const tooltipPortal = new ComponentPortal(AwesomeT

    // Attach tooltip portal to overlay
    const tooltipRef: ComponentRef<AwesomeTooltipCompo

```

Basically, we are instantiating the `AwesomeTooltipComponent` component and inserting it into the `overlayRef` we created in the previous step.

Last thing here, we are passing the text into the newly instantiated tooltip component reference.

But what is going on in the `show` method? Firstly, we're creating the tooltip portal as `new ComponentPortal()` . Then, we're attaching it to the `overlayRef` and assigning text to the created tooltip component instance. 🤯

A bit of magic, huh? Let me explain 😅

. . .

Overlays Explained

Let's start with `OverlayRef` . It is an instance of `PortalOutlet` . You may think of it as of a placeholder in your application that may be dynamically replaced with the different content.

`PortalOutlet` in its turn doesn't accept any content, but only instances of `Portal` . `Portal` , per se, is a thin wrapper that let you

operate with `PortalOutlet` .

It's exactly what we did. We created the tooltip portal:

```
const tooltipPortal = new
ComponentPortal(AwesomeTooltipComponent);
```

And attached it to the `PortalOutlet` instantiated previously:

```
const tooltipRef: ComponentRef<AwesomeTooltipComponent>
      = this.overlayRef.attach(tooltipPortal);
```

After attaching tooltip portal to the `OverlayRef` we're getting `ComponentRef` , which refers to the created `AwesomeTooltipComponent` . We may access `AwesomeTooltipComponent` instance ehrough that reference and pass the tooltip text into.

```
tooltipRef.instance.text = this.text;
```

The hardest part is behind now 🥳 .

At this stage, we have an overlay component created dynamically when we're moving a mouse on text and destroyed when we're moving the mouse out.

But the tooltip is rendered somewhere misplaced. Let's position it exactly above the host element where we need to show the tooltip.

. . .

Tooltip Positioning

Since the tooltip is rendered outside of the document elements flow with `position: absolute` we need to provide it with exact coordinates of the host element. Luckily, Angular CDK has some built-in functionality for this as well.

The positioning of the overlay components could be implemented using `OverlayPositionBuilder` abstraction:

```
@Directive({ selector: '[awesomeTooltip]' })
export class AwesomeTooltipDirective implements OnInit

  constructor(private overlayPositionBuilder: OverlayP
              private elementRef: ElementRef,
              private overlay: Overlay) {}

  ngOnInit() {
    const positionStrategy = this.overlayPositionBuild
      // Create position attached to the elementRef
      .flexibleConnectedTo(this.elementRef)
      // Describe how to connect overlay to the elemen
      // Means, attach overlay's center bottom point t
      // top center point of the elementRef.
      .withPositions([{
```

`OverlayPositionBuilder` is an entity that knows how to position your overlay element relatively to the host element.

We're creating a new position strategy connected to the `elementRef`, which means the position of the created overlays will be connected to the `elementRef` the following way: center bottom point of the `overlay` will be connected to the center top point of the `elementRef`.

Literally, it means that the component will appear above the `elementRef`.

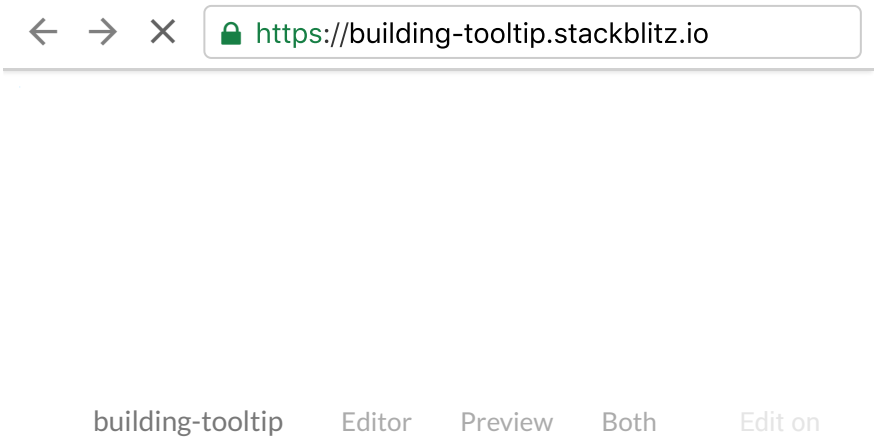
The last step we need to do is to connect a positioning strategy to the already created overlay:

```
@Directive({ selector: '[awesomeTooltip]' })
export class AwesomeTooltipDirective implements OnInit

  constructor(private overlayPositionBuilder: OverlayP
              private elementRef: ElementRef,
              private overlay: Overlay) {}

  ngOnInit() {
    const positionStrategy = this.overlayPositionBuild
      .flexibleConnectedTo(this.elementRef)
      .withPositions([{
        originX: 'center',
        originY: 'top',
        overlayX: 'center',
```

At this stage, we have a fully functional tooltip and it is positioned properly above the host component.

Results

So, we did it! 😚 Here you may find the live example of tooltip directive:



Also, I've published sources of the example tooltip at my GitHub, check it if you just need a working example.

Recap

We have built an Angular Tooltip Directive using Angular CDK `OverlayModule`. Now we know how to deal with dynamic components rendering using Angular CDK `OverlayModule`. And we're ready for new challenges!

As you may notice we could add more to the article: animations, styling, etc. But these topics are out of Angular CDK scope, and I decided to focus on Angular CDK features only.

However, there is one more important thing I haven't mentioned—repositioning of the tooltip during the page scrolling. I hope we can get back to this part in the upcoming articles.

Stay tuned and let me know if you have any particular CDK topics you would like to hear about!

Resources

Learn more about Angular CDK portals and overlays from the official documentation:

- Portals—https://material.angular.io/cdk/portal/overview
- Overlays—https://material.angular.io/cdk/overlay/overview