Applause from you and 56 others

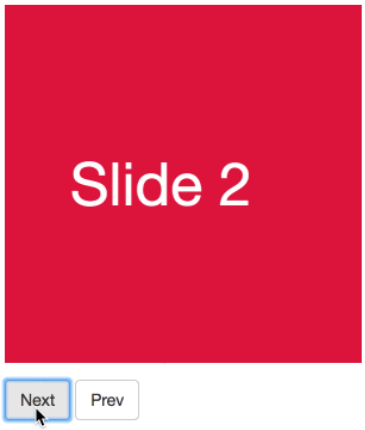
Netanel Basal
Sep 12 · 3 min read

Building a Simple Carousel Component with Angular

In this article, we will create a simple carousel component with Angular that includes animation with the help of the Animation Builder service. We will also discuss several approaches to querying the DOM.

The following will be our final result:

```
<carousel>
  <ng-container *ngFor="let item of items;">
    <ng-container *carouselItem>
      <div class="item">{{item.title}}</div>
    </ng-container>
  </ng-container>
```



Let's start by creating the `carousel` component.

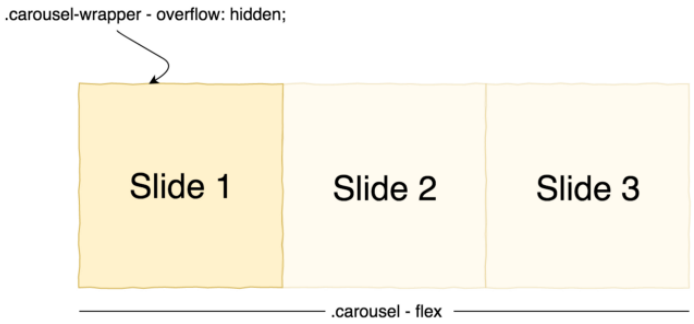
The Component CSS

```
@Component({
  selector: 'carousel',
  template: `
    <section class="carousel-wrapper">
      <ul class="carousel" #carousel>
         <!--  We need to loop over the items -->
         <li></li>
      </ul>
    </section>
  `,
  styles: [`

    .carousel-wrapper {
      overflow: hidden;
    }

    .carousel {
      list-style: none;
      margin: 0;
```

Here, we'll use a well-known CSS technique. We need a wrapper element with `overflow: hidden` and a fixed width we will define later as equal to the width of a carousel item. We must also set the carousel element to `display: flex` so the items appear in the same row. Later, we will use the Animation Builder service to animate the transform property of the carousel element.



Let's continue to see how we can pull the items out of the template.

Building the Carousel Item Directive

```
@Directive({
  selector: '[carouselItem]'
})
export class CarouselItemDirective {

  constructor( public tpl : TemplateRef<any> ) {
  }
```

The Carousel Item directive is a structural directive. We can leverage Dependency Injection to get a reference to the template.

Rendering The Items

Now, let's see how we can use this data in the parent component.

```
@Component({
  selector: 'carousel',
  template: `
    <section class="carousel-wrapper">
      <ul class="carousel-inner" #carousel>
        <li *ngFor="let item of items;" class="carouse
          <ng-container [ngTemplateOutlet]="item.tpl">
        </li>
      </ul>
    </section>
  `,
```

We can query for the `CarouselItemDirective` instances in our template with the `ContentChildren` decorator. Then we loop through them, passing the `CarouselItemDirective.tpl` property that holds a reference to a template to the `ngTemplateOutlet` directive. (which we can get from the `CarouselItemDirective` constructor via DI).

Setting The Wrapper Width

As I mentioned before, we need to set the `wrapper` element width to the same as the first item width by querying for the `.carousel-item` element. For this task, I will use a directive with selector `.carousel-item` .

```
1  @Directive({
2    selector: '.carousel-item'
3  })
4  export class CarouselItemElement {
```

Now we can obtain a reference to every element in the template matching the directive selector.

```
1  @ViewChildren(CarouselItemElement, { read: ElementRef }
```

Pay attention to two things here: we used `ViewChildren` and not `ContentChildren` , and we asked for the native DOM element by setting the read property to `ElementRef` .

Now we can get the width of the first element and set the wrapper element to the same value.

```
@Directive({
  selector: '.carousel-item'
})
export class CarouselItemElement {
}

@Component({
  selector: 'carousel',
  template: `
    <section class="carousel-wrapper" [ngStyle]="carou
      <ul class="carousel-inner" #carousel>
        <li *ngFor="let item of items;" class="carouse
          <ng-container [ngTemplateOutlet]="item.tpl">
        </li>
      </ul>
    </section>
  `,
  styles: [`...`]
})
export class CarouselComponent implements AfterViewIni
  @ContentChildren(CarouselItemDirective) items : Quer
  @ViewChildren(CarouselItemElement, { read: ElementRe
```

Note: You can also use `ViewChild` to get the first element, but I wanted to illustrate this method in case you want to build a carousel with dynamic width.

Adding The Control Buttons

```
@Component({
  selector: 'carousel',
  template: `
    <section class="carousel-wrapper" [ngStyle]="carou
      ...
    </section>

    <div *ngIf="showControls">
      <button (click)="next()">Next</button>
      <button (click)="prev()">Previous</button>
    </div>
```

Before we see the `next()` and `prev()` methods, let's create a couple properties that will help us to build the carousel.

```
@Component({
  selector: 'carousel',
  template: `
    <section class="carousel-wrapper" [ngStyle]="carou
      <ul class="carousel-inner" #carousel>
        <li *ngFor="let item of items;" class="carouse
          <ng-container [ngTemplateOutlet]="item.tpl">
        </li>
      </ul>
      <div *ngIf="showControls">
        <button (click)="next()">Next</button>
        <button (click)="prev()">Previous</button>
      </div>
    </section>
  `,
  styles: [`...`]
})
export class CarouselComponent implements AfterViewIni
```

The `carousel` property is a reference to the native DOM element of the carousel. Note that in this case I'm using a local template variable to query the element from the template.

We also have two `Inputs`, one for the animation timing and one that relays whether we need to show the controls. (we will see why this is useful later)

The remaining properties simply keep track of the carousel position.

Implementing The next() Method

```
export class CarouselComponent implements AfterViewIni
  ...
  @ViewChild('carousel') private carousel : ElementRef

  constructor( private builder : AnimationBuilder ) {
  }

  next() {
    if( this.currentSlide + 1 === this.items.length )

    this.currentSlide = (this.currentSlide + 1) % this

    const offset = this.currentSlide * this.itemWidth;

    const myAnimation : AnimationFactory = this.builde
```

We need to do some math to calculate the carousel position. Then, we can use the Animation Builder service to create and play the animation based on the timing and offset variables.

The purpose of Animation Builder service is to produce an animation sequence programmatically within an Angular component or directive.

Programmatic animations are first built and then a player is created when the build animation is attached to an element.

When an animation is built an instance of `AnimationFactory` will be returned. Using that an `AnimationPlayer` can be created which can then be used to start the animation.

Implementing The prev() Method

```
export class CarouselComponent implements AfterViewIni
  ...
  @ViewChild('carousel') private carousel : ElementRef

  constructor( private builder : AnimationBuilder ) {
  }

  prev() {
   if( this.currentSlide === 0 ) return;

    this.currentSlide = ((this.currentSlide - 1) + thi
    const offset = this.currentSlide * this.itemWidth;

    const myAnimation : AnimationFactory = this.builde
      animate(this.timing, style({ transform: `transla
```

Here, we can use the same process as the `next()` method: calculate the carousel position and building and starting the animation.

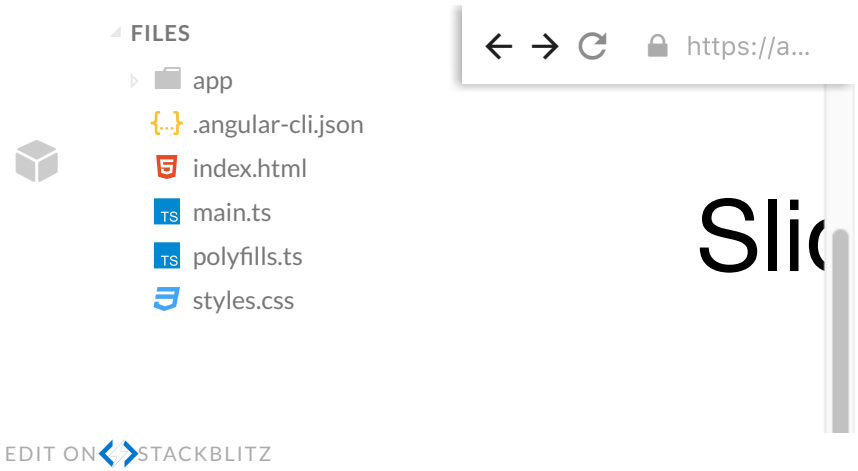
Using Custom Controls

We can export our component to the consumer template with the `exportAs` property on the component metadata.

```
@Component({
  selector: 'carousel',
  exportAs: 'carousel'
  ....
})

<carousel #carousel="carousel" [showControls]="false">
  <ng-container *ngFor="let item of items;">
    <ng-container *carouselItem>
      <div class="item">{{item.title}}</div>
    </ng-container>
  </ng-container>
```

That's all.



Follow me on Medium or Twitter to read more about Angular, Vue and JS!