Demystifying *ngFor

The power of structural directives

rebuy

Oh no, audience participation

Hi, my name is Simon

🧑‍💻 Web Development, Self-Hosting, Infrastructure as Code

🎸 Making music, singing & playing drums, guitar and bass

🛠️ Repairing electronic devices, bikes, etc.

🌳 Hiking, biking, inline skating, camping

🐦 I don’t even have twitter
so follow me on GitHub (@similicious), I guess?

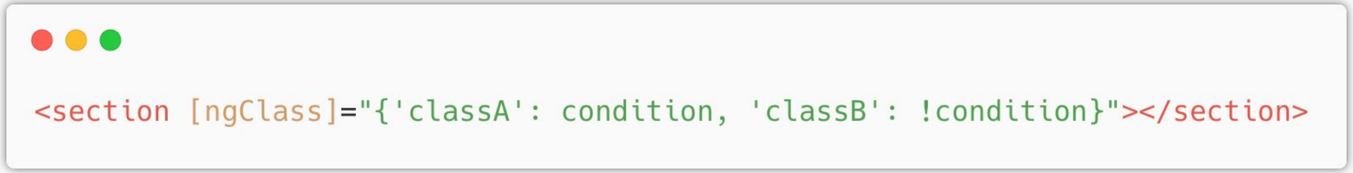
Simon Wienecke
Frontend Engineer at rebuy

Agenda

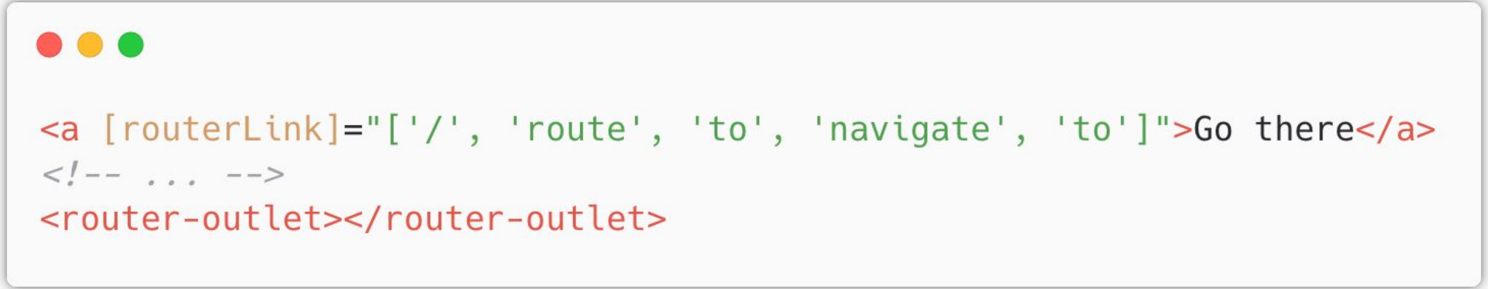
1. Directives
2. Structural directives
3. Implementing *ngFor ourselves
4. What else is possible?
5. (Strong typing)

Directives

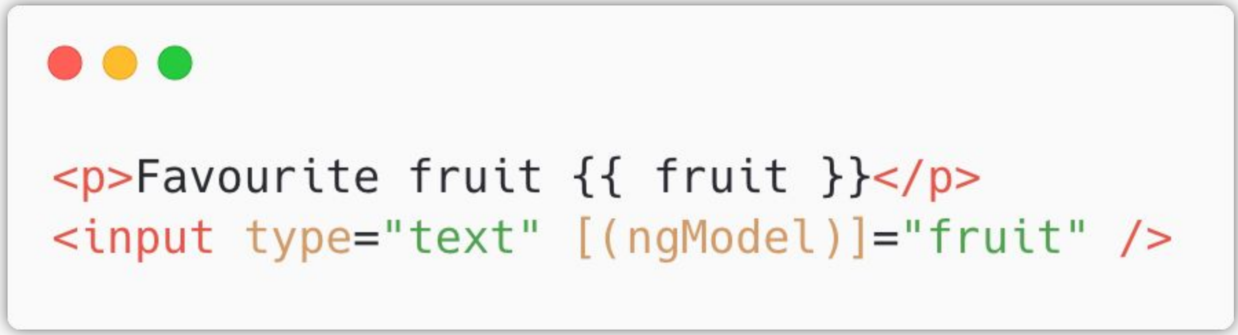
Built-in directives



```html
<section [ngClass]="{'classA': condition, 'classB': !condition}"></section>
```

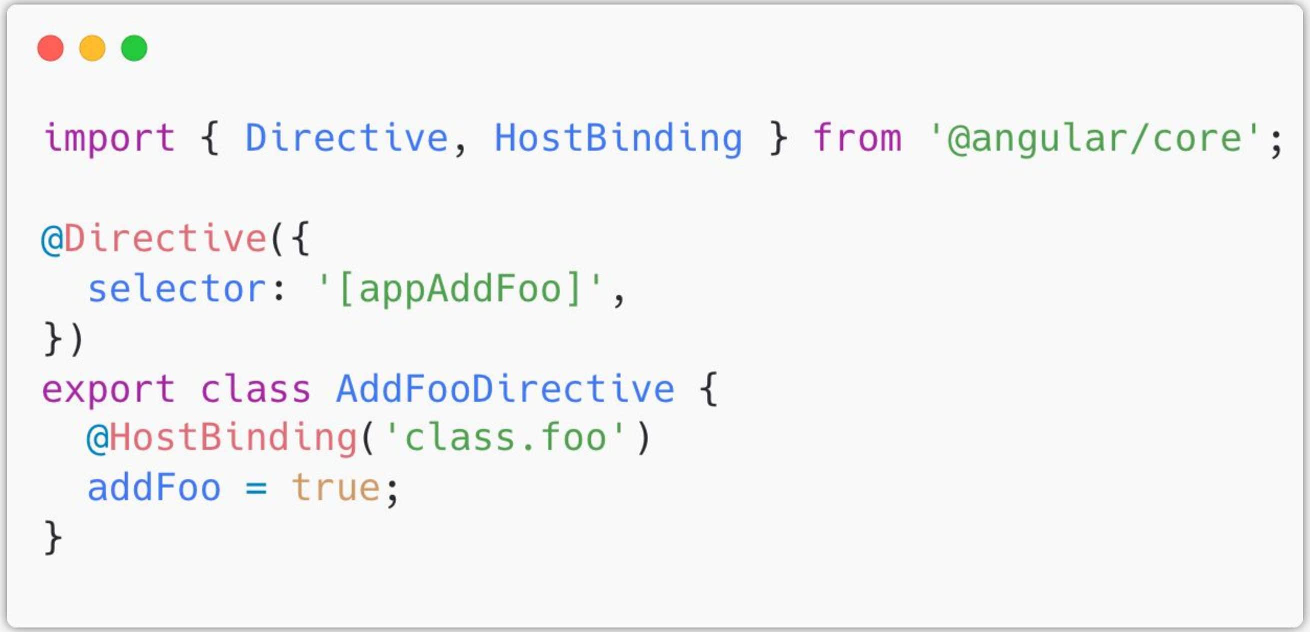


```html
<a [routerLink]="['/', 'route', 'to', 'navigate', 'to']">Go there</a>
<!-- ... -->
<router-outlet></router-outlet>
```



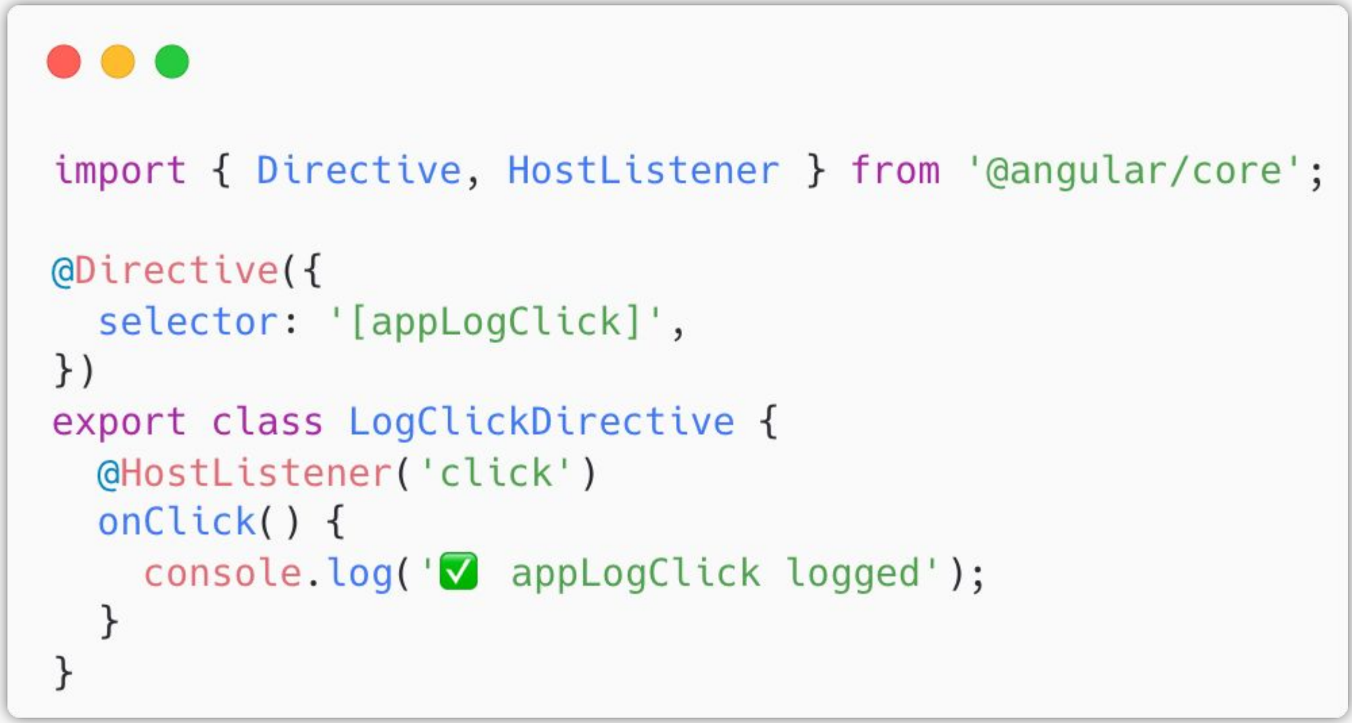
```html
<p>Favourite fruit {{ fruit }}</p>
<input type="text" [(ngModel)]="fruit" />
```

A basic directive



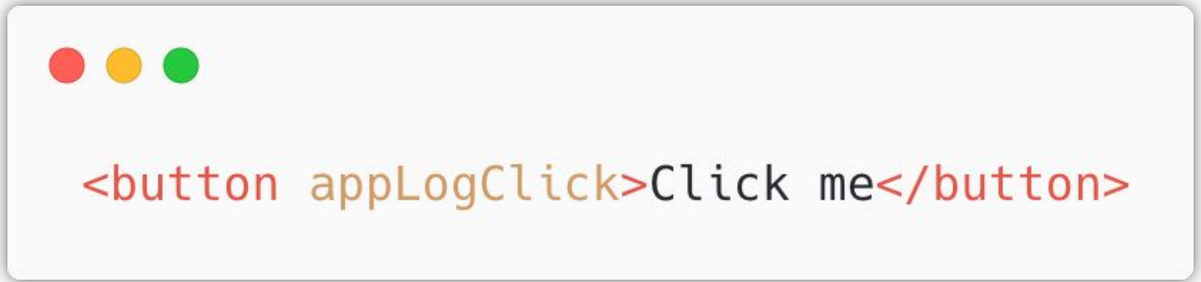
```
import { Directive, HostBinding } from '@angular/core';

@Directive({
  selector: '[appAddFoo]',
})
export class AddFooDirective {
  @HostBinding('class.foo')
  addFoo = true;
}
```

A basic directive



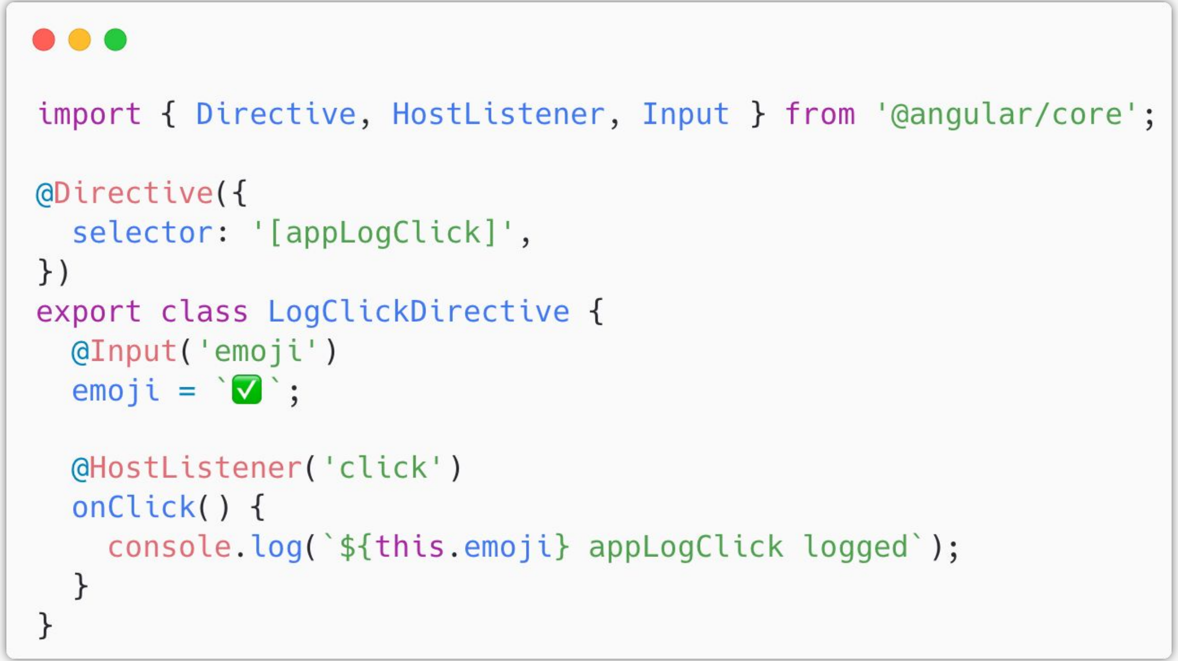
```
import { Directive, HostListener } from '@angular/core';

@Directive({
  selector: '[appLogClick]',
})
export class LogClickDirective {
  @HostListener('click')
  onClick() {
    console.log('✅ appLogClick logged');
  }
}
```

Attaching a directive to an element

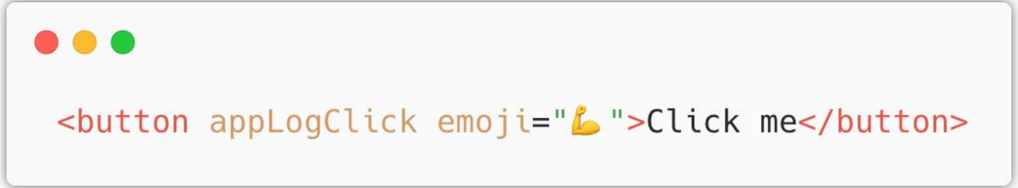


```
<button appLogClick>Click me</button>
```

Passing data to a directive

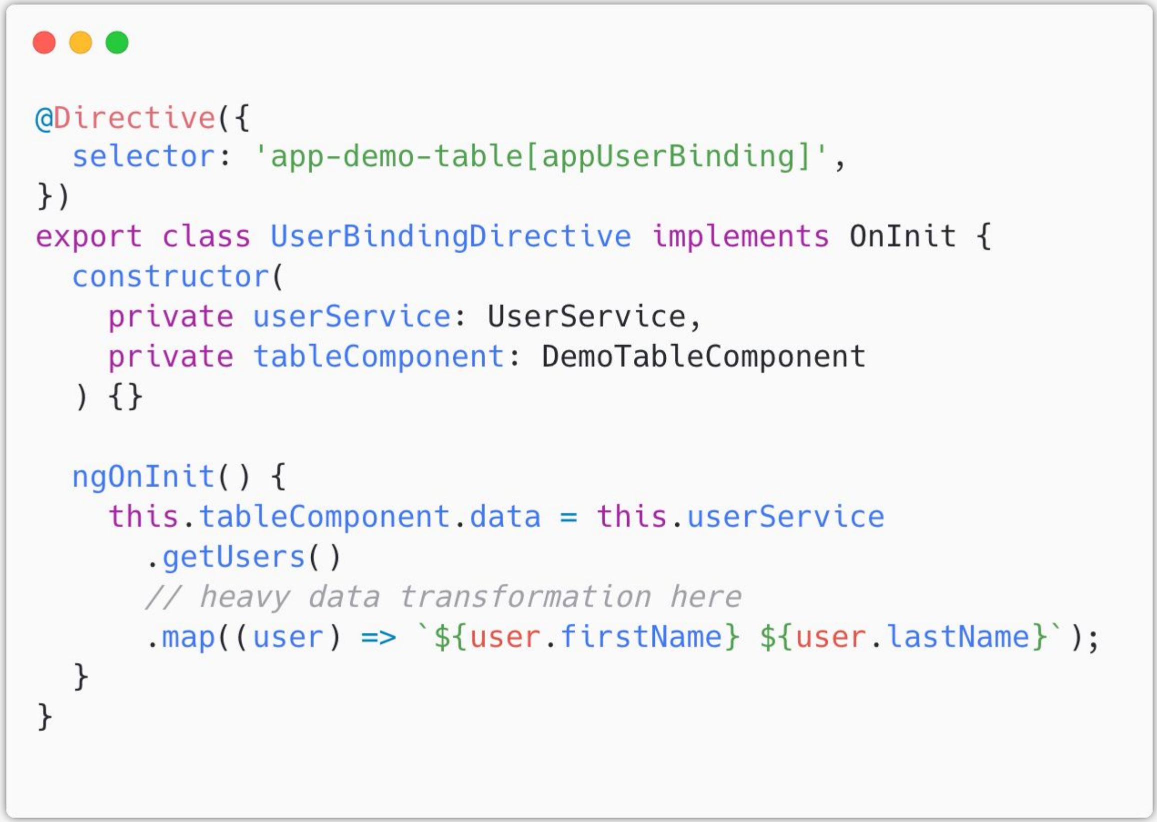


```
import { Directive, HostListener, Input } from '@angular/core';

@Directive({
  selector: '[appLogClick]',
})
export class LogClickDirective {
  @Input('emoji')
  emoji = `✅`;

  @HostListener('click')
  onClick() {
    console.log(`${this.emoji} appLogClick logged`);
  }
}
```



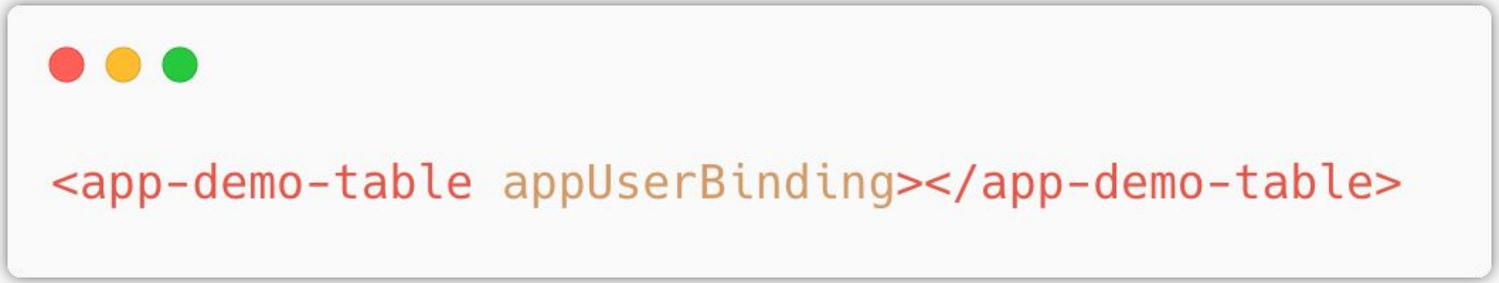
```
<button appLogClick emoji="💪">Click me</button>
```

You can also leverage DI



```
@Directive({
  selector: 'app-demo-table[appUserBinding]',
})
export class UserBindingDirective implements OnInit {
  constructor(
    private userService: UserService,
    private tableComponent: DemoTableComponent
  ) {}

  ngOnInit() {
    this.tableComponent.data = this.userService
      .getUsers()
      // heavy data transformation here
      .map((user) => `${user.firstName} ${user.lastName}`);
  }
}
```

.. to encapsulate data binding logic



```
<app-demo-table appUserBinding></app-demo-table>
```

Summary

- Directives
    - are classes decorated with @Directive
    - are applied to an element via their selector
    - modify the behaviour of elements
- Use
    - @HostBinding to get/set attributes
    - @HostListener to listen to events
    - Dependency injection to get a reference to Services, Components, Directives …
- Allows
    - to extract code from Components into reusable Directives

Quiz Round 1

Which decorator allows you to set a property on the element a directive has been applied to?

A: @Input()

B: @HostListener()

C: @HostProperty()

D: @HostBinding()

Quiz Round 1

To which element does a directive with this selector apply:
‘img[ngSrc]’

A: <img alt=”Some image” />

B: <img src=”picture.jpg” />

C: <img ngSrc=”picture.jpg” />

D: <figure><img src=”...”></figure>

Quiz Round 1

Which of these directives does not come with Angular?

A: NgSingular

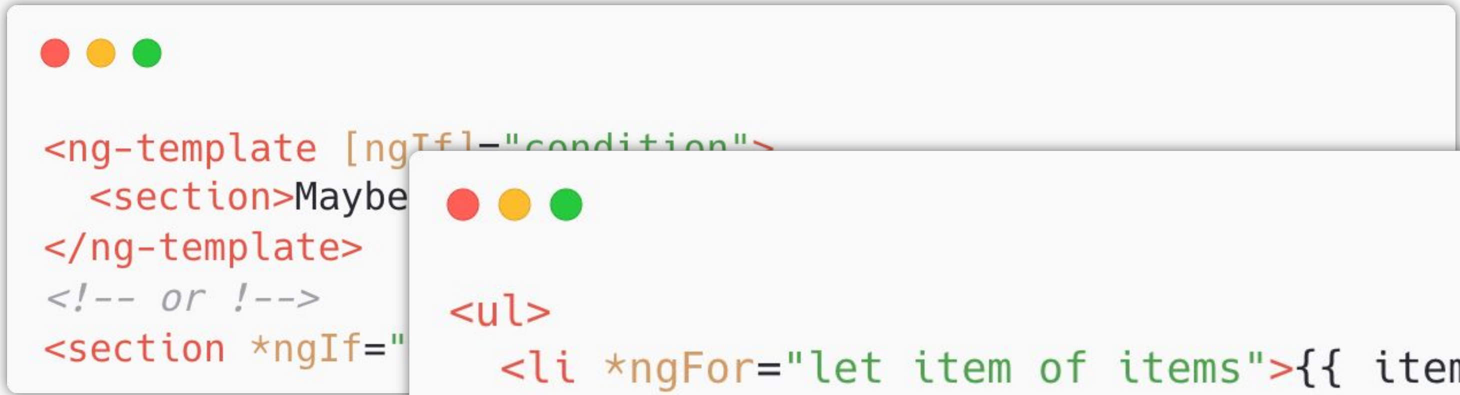
B: NgComponentOutlet

C: RouterLinkActive

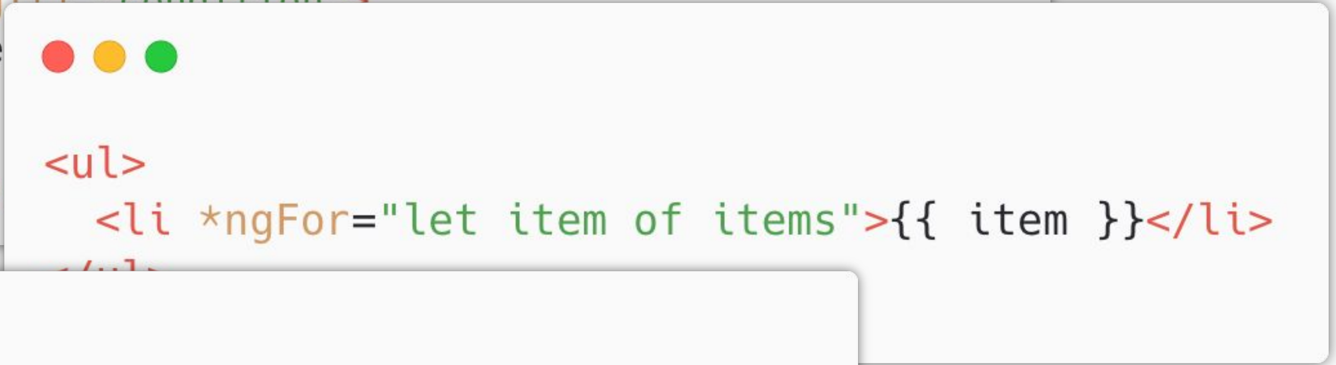
D: NgPlural

Structural Directives

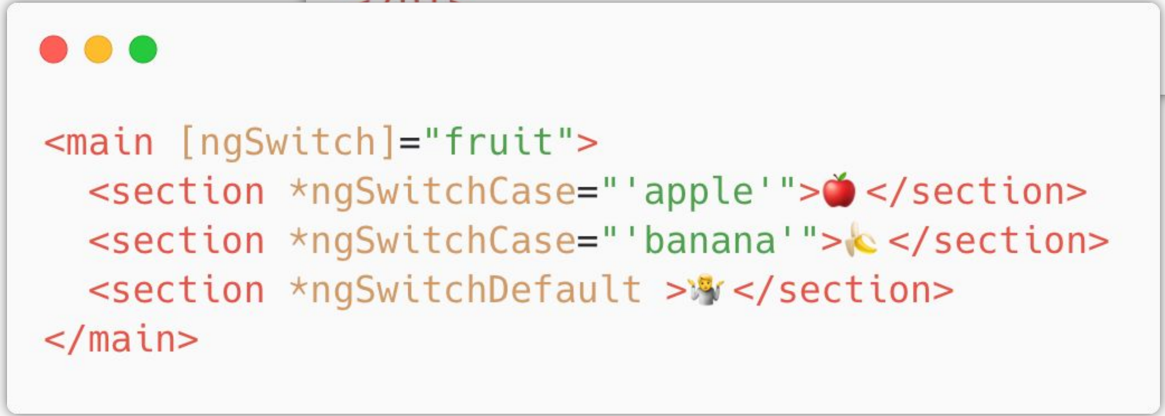
Built-in structural directives



```
<ng-template [ngIf]="condition">
  <section>Maybe
</ng-template>
<!-- or !-->
<section *ngIf="
```

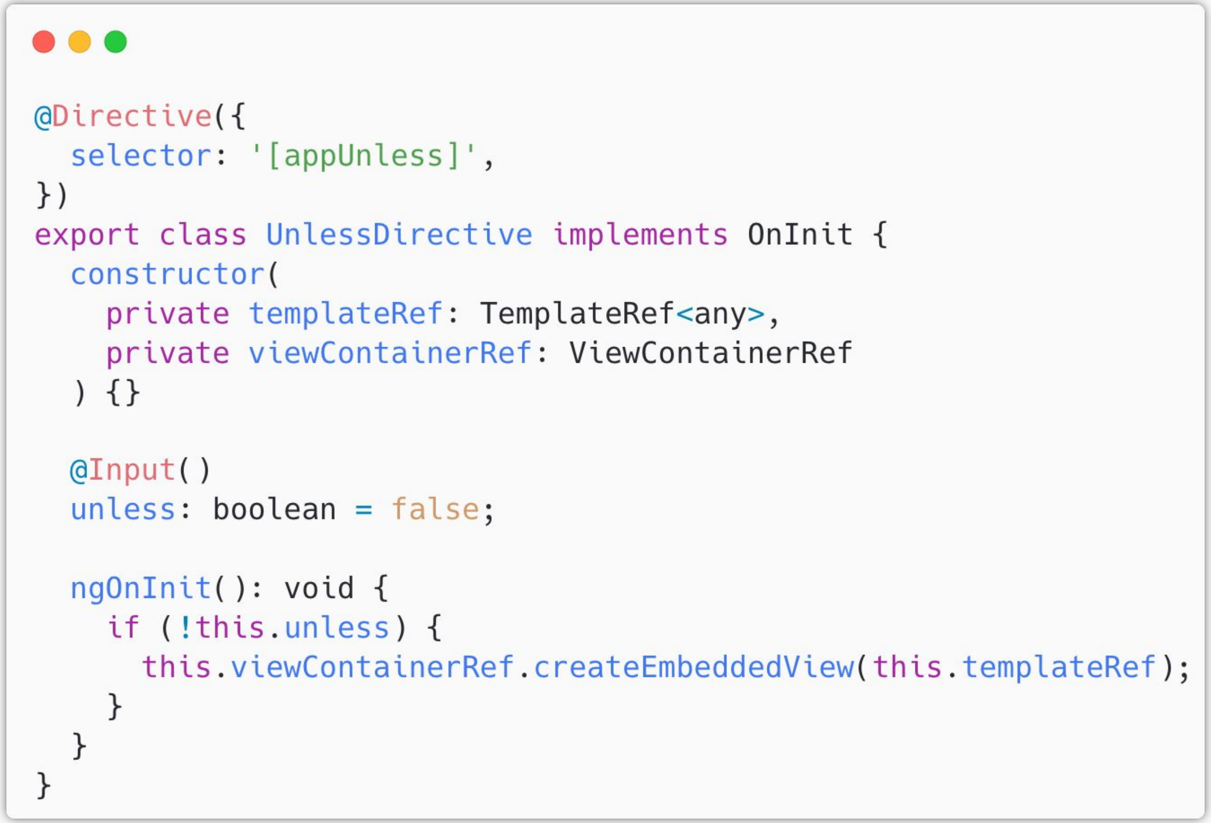


```
<ul>
  <li *ngFor="let item of items">{{ item }}</li>
</ul>
```



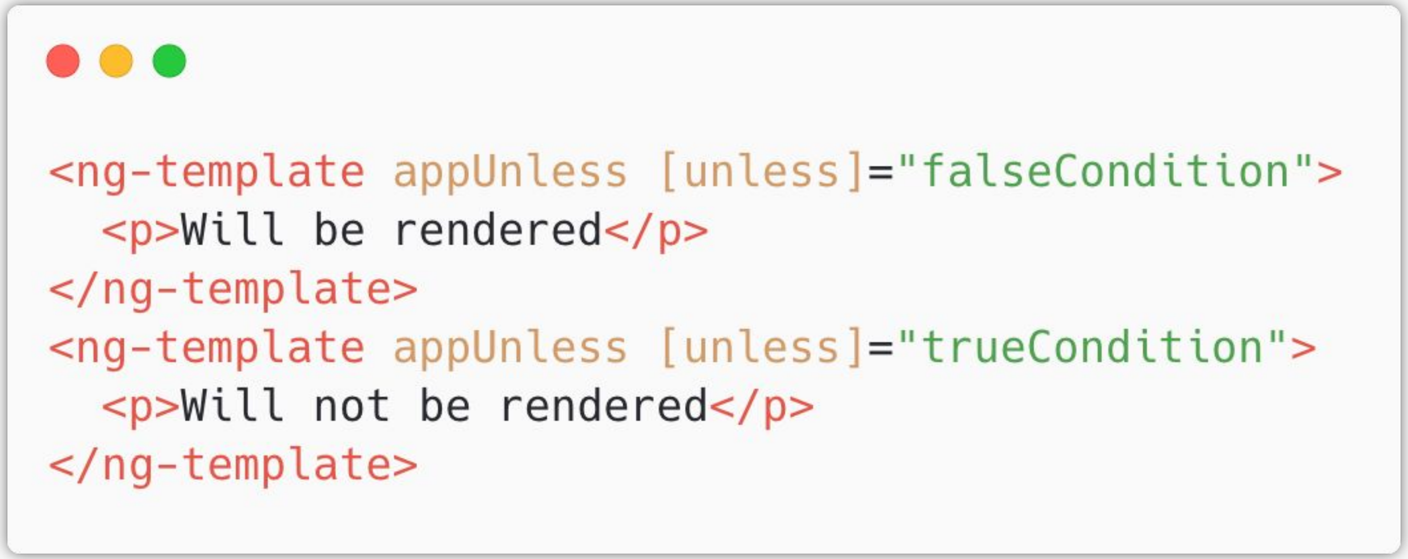
```
<main [ngSwitch]="fruit">
  <section *ngSwitchCase="'apple'">🍎</section>
  <section *ngSwitchCase="'banana'">🍌</section>
  <section *ngSwitchDefault >🤷</section>
</main>
```

A simple structural directive



```
@Directive({
  selector: '[appUnless]',
})
export class UnlessDirective implements OnInit {
  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainerRef: ViewContainerRef
  ) {}

  @Input()
  unless: boolean = false;

  ngOnInit(): void {
    if (!this.unless) {
      this.viewContainerRef.createEmbeddedView(this.templateRef);
    }
  }
}
```

appUnless in action



```
<ng-template appUnless [unless]="falseCondition">
  <p>Will be rendered</p>
</ng-template>
<ng-template appUnless [unless]="trueCondition">
  <p>Will not be rendered</p>
</ng-template>
```

That's all you need to know. Let's live code.

Star syntax explained

```
*:prefix="

<ng-template                                 <li
    ngFor                                        *
    [ngForOf]=”users”
    let-user
    let-index=”index”
    let-isFirstItem=”first”
    let-isLastItem=”last”                    >...</li>
>...</ng-template>
```

Summary

- Structural directives
  - are normal directives
  - that are placed on a template
  - can use star-syntax to simplify the code
  - The star syntax will be converted to ng-template syntax
- They work by
  - injecting the template
  - rendering it via ViewContainerRef
  - passing a context object to the consumer

Quiz Round 2

Which statement is not true?

A: The star syntax is compiled into <ng-template …>

B: Structural directives allow decoupling logic from the template.

C: ViewContainerRef is used to render a template.

D: Multiple structural directives can be used on one element.

Quiz Round 2

Which star syntax is invalid?

A: *ngFor="of users let user = $implicit;"

B: *ngFor="let user; let index = index; of: users"

C: *ngIf="{ a: 'foo' }; let bar = ngIf"

D: *ngFor="let index = index; trackBy: trackByFn; of: users"

What else is possible?

Why structural directives are awesome

User supplies the template

you bring the behaviour.

Star syntax is expressive

One can almost form english sentences.

Total control over DOM

Slides & Code

https://t.ly/jvqNK

Any questions?