

# Displaying Data: Data Binding, Directives, and Pipes

---



**John Papa**

PRINCIPAL ARCHITECT

@john\_papa

[www.johnpapa.net](http://www.johnpapa.net)



# Overview



**Data Binding**

**Built-in Directives**

**Pipes**



# Data Binding

---



# Data Binding

We use data binding to help coordinate communication between a Component and its Template.



{{expression}}

← .....  
Interpolation

[property] = "expression"

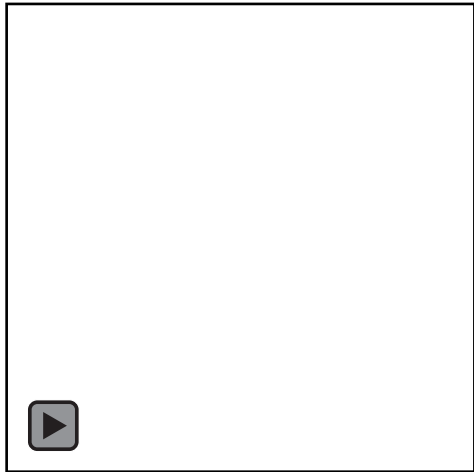
← .....  
One Way Binding

(event) = "statement"

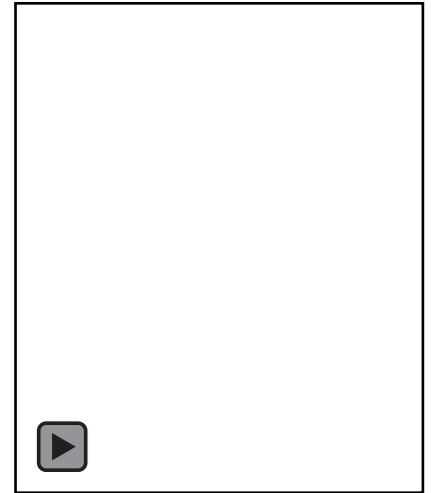
.....→  
Event Binding

[(ngModel)] = "property"

← .....→  
Two Way Binding



DOM



Component



Angular 2's change detection is based on unidirectional data flow



# Benefits of Angular 2's Unidirectional Data Flow

Easier widget  
integration

No more \$apply

No more repeated  
digest cycles

No more watchers

No more  
performance issues  
with digest cycle  
and watcher limits



# Interpolation

Using the `{{ }}` to render the bound value to the Component's Template





```
<h3>Vehicle: {{vehicle.name}}</h3>  
<div>  
    
  <a href="{{vehicle.wikiLink}}">Wiki</a>  
</div>
```

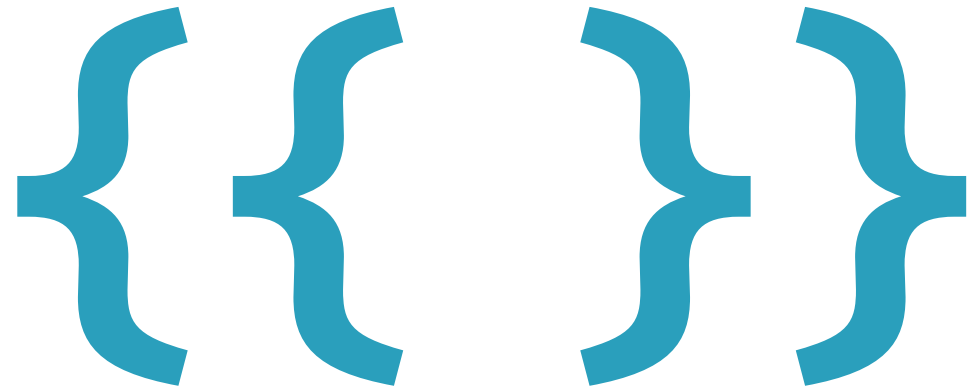
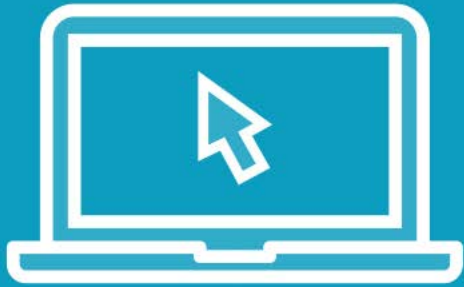
## Interpolation

Evaluate an expression between double curly braces

**{{ expression }}**

# Interpolation

Demo

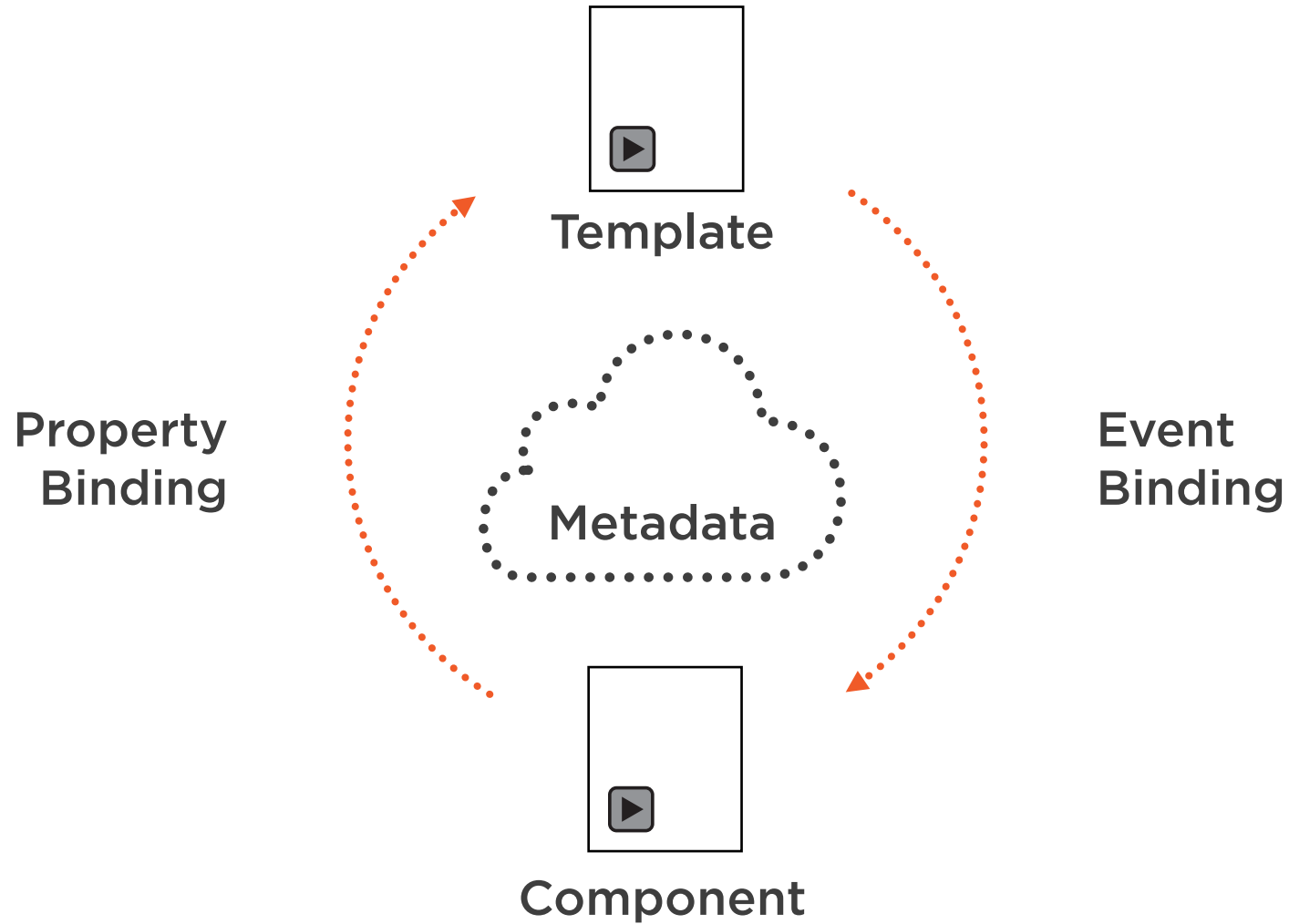


# Property Binding

Using the `[]` to send values from the Component to the Template



# Data Binding Communication



We set properties and events of  
DOM elements, not attributes



# One Way

Binding target property



```
graph TD; A[Binding target property] -- orange line --> B["{{expression}}"]; A -- orange line --> C["[target] = 'expression'"]; A -- orange line --> D["bind-target = 'expression'"];
```

`{{expression}}`

`[target] = "expression"`

`bind-target = "expression"`

Data source to view target

## One Way In

Element property

Component property

Directive property

```
<img [src]="vehicle.imageUrl">
```

```
<vehicle-detail [vehicle]="currentVehicle"></vehicle-detail>
```

```
<div [ngClass] = "{selected: isSelected}">X-Wing</div>
```

## Property Binding

**[property]="expression"**

Bind to element, Component or a directive property



## One Way In

```
<button [attr.aria-label]="ok">ok</button>
```

Attribute binding

```
<div [class.isStopped]="isStopped">Stopped</div>
```

Class property binding

```
<button [style.color]="isStopped ? 'red' : 'blue'">
```

Style property binding

## Property Binding

For attributes use **attr**

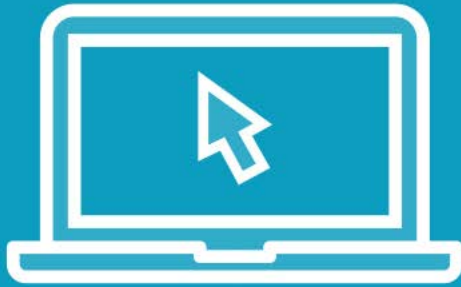
Use dots for nested properties





# Property Binding

Demo




# Event Binding

Using the ( ) to send events from the Template to the Component



# One Way

Binding target event



(target) = "statement"  
on-target = "statement"

View target to data source

## One Way to the Component

```
<button (click)="save()">Save</button>
```

Element event

```
<vehicle-detail (changed)="vehicleChanged()"></vehicle-detail>
```

Component event

## Event Binding


Execute an expression when an event occurs

**(event-target)="statement"**



## One Way to the Component

```
<input [value]="vehicle.name"  
      (input)="vehicle.name=$event.target.value">
```



The diagram consists of two orange rectangular boxes. The first box is positioned under the `(input)` part of the code and has a line extending downwards. The second box is positioned under the `$event` part of the code and has a line extending to the right. These two lines meet at a horizontal line that extends to the right, where it connects to the 'Input change event' label. A vertical line also extends from the bottom of the first box to this same horizontal line.

Event message

Input change event

`$event`

Contains a message about the event



```
@Input() vehicle: Vehicle;  
@Output() onChange = new EventEmitter<Vehicle>();  
changed() { this.onChange.emit(this.vehicle); }
```

Custom event

```
<vehicle-detail (onChange)="vehicleChanged($event)"  
[vehicle]="currentVehicle"> </vehicle-detail>
```

Output (event)

## Custom Events

**EventEmitter** defines a new event

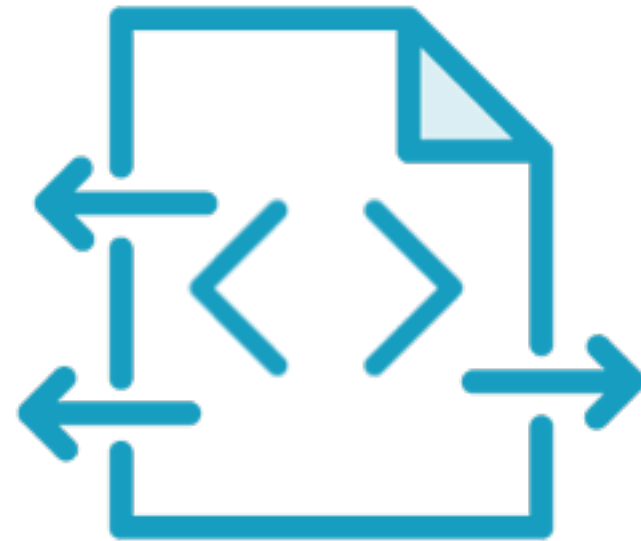
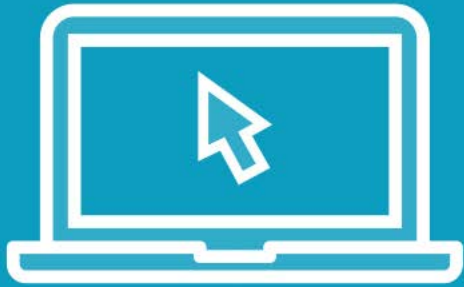
Fire its **emit** method to raise event with data

Bind to the event on the Component's Template



# Event Binding

Demo



# Two Way Binding

`[( )]` sends a value from Component to Template, and sends value changes in the Template to the Component





# Two Way

```
[(ngModel)] = "expression"  
bindon-ngModel= "expression"
```



## Value in, Value Out

```
<input [(ngModel)]="vehicle.name">
```

Built-in directive

## Two Way Binding

[( )] = Football in a box



## Importing the FormsModule

```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { FormsModule } from '@angular/forms';
```

```
@NgModule({  
  imports: [  
    BrowserModule,  
    FormsModule  
  ],  
  // ...  
})
```

**Import** FormsModule

**Import** into the Angular module

## Using `ngModel`

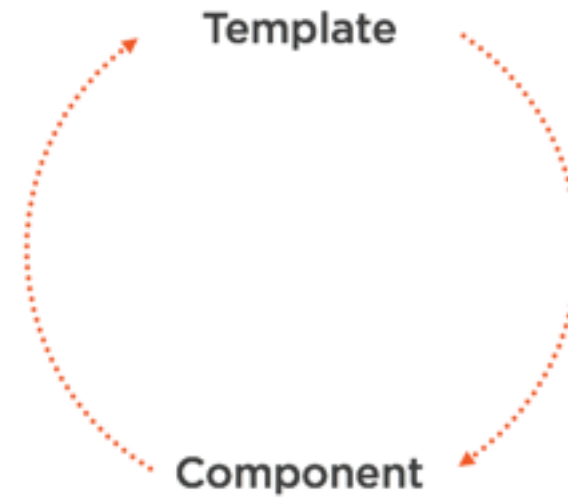
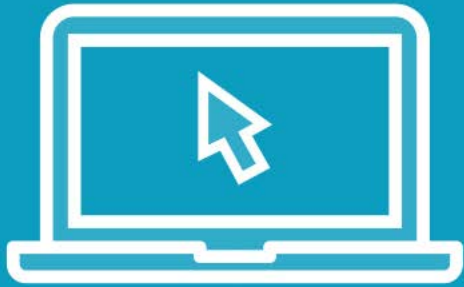
### Import `FormsModule`

Add it to the Angular module's import list



# Data Binding

Demo



# Built-in Directives

---



# Directives

When Angular renders templates, it transforms the DOM according to instructions from Directives



# Angular Class and Style Directives

## Angular 1

**ng-class**

`ng-class="{active: isActive, color: myColor}"`

**ng-style**

`ng-style="{color: colorPreference}"`

## Angular 2

**ngClass**

`[ngClass]="{active: isActive, color: myColor}"`

**ngStyle**

`[ngStyle]="{color: colorPreference}"`

`[style.color]="colorPreference"`



## Style Binding

```
<div [ngStyle]="setStyles()">{{vehicle.name}}</div>
```

Style binding

# ngStyle

Alternative to `[style.style-name]`

Setting multiple styles





## Class Binding

```
<div [ngClass]="setClasses()">{{vehicle.name}}</div>
```

Class binding

## ngClass

Alternative to `[class.class-name]`

Setting multiple classes



# Angular Structural Directives

## Angular 1

## Angular 2

ng-repeat

\*ngFor

ng-if

\*ngIf

ng-switch

\*ngSwitch



Familiar concepts translate to Angular 2



## Conditional Template

Show template if truthy

```
<div *ngIf="currentVehicle">  
  You selected {{currentVehicle.name}}  
</div>
```

\*ngIf

Conditionally removes elements from the DOM

Structural directive

Use `[style.visibility]="isVisible()"` to hide



## Repeating a Template

```
<ul>
```

**Iterate** over the characters

```
<li *ngFor="let character of characters">
```

```
  {{ character.name }}
```

**Local variable**

```
</li>
```

```
</ul>
```

### \*ngFor

Structural directive


Show an element n number of times

**let** declares a local variable



## Creating an index

```
<ul>  
  <li *ngFor="let character of characters, let i = index">  
    {{i}}. {{ character.name }}  
  </li>  
</ul>
```



Local variable

## Local Variables

**let** declares a local variable



## Importing the CommonModule

```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { FormsModule } from '@angular/forms';
```

**Import** BrowserModule

```
@NgModule({  
  imports: [  
    BrowserModule,  
    FormsModule  
  ],  
  // ...
```

**Import** into the Angular module

## Using Built-In Directives

**BrowserModule** imports **CommonModule**

Import and add it to the Angular module's import list



Demo



Directives



# Pipes

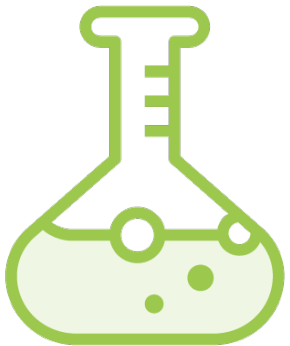
---





# Pipes

Pipes allow us to transform data for display in a Template.



# Angular Formatters

**Angular 1**

**Angular 2**

filters



pipes



## String Pipes

```
<p>{{character.name | uppercase}}</p>  
<p>{{character.name | lowercase}}</p>
```

Lowercase Pipe

## Built-in Pipes

Format a value in a Template



## Date formatting

```
<p>{{eventDate | date:'medium'}}</p>  
<p>{{eventDate | date:'yMMMd'}}</p>
```

Date Pipe

## Date Pipe

<https://angular.io/docs/ts/latest/api/>

**Date** accepts **format**

**expression | date[:format]**



## Numeric formatting

```
<p>{{price | currency}}</p>  
<p>{{value | percent:'1.1-1'}}</p>  
<p>{{value | number:'1.1-3'}}</p>
```

Number Pipe

## Numeric Pipes

**Number** and **Percent** accept **digitInfo**

**Expression | number[:digitInfo]**

**{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}**



# Async Pipe

Subscribes to a Promise or an Observable, returning the latest value emitted



```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'initCaps' })
export class InitCapsPipe implements PipeTransform {
  transform(value: string, args?: any[]) {
    return value.toLowerCase()
      .replace(/(?:^|\s)[a-z]/g, m => m.toUpperCase());
  }
}
```

Implement the interface

## Custom Pipes

**@Pipe** decorator

**value** to transform

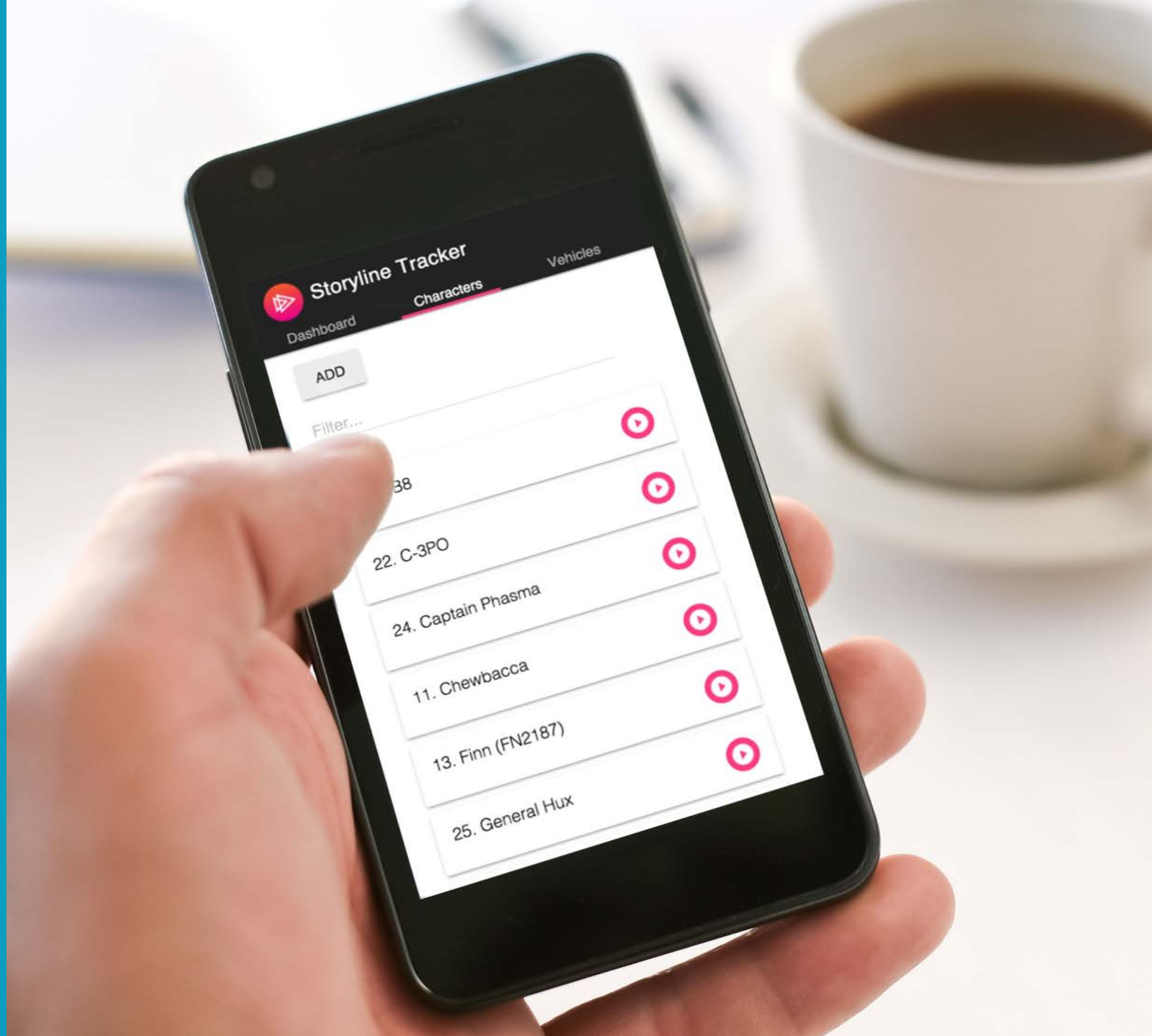
Optional **args**



# Demo



## Putting It All Together





# Template Syntax



**Data Binding**

**Unidirectional Data Flow**

**Attribute Directives**

**Structural Directives**

**Pipes**

