

Angular 2 Essentials: Modules, Components, Templates, and Metadata



John Papa

PRINCIPAL ARCHITECT

@john_papa

www.johnpapa.net



Overview



Modules

Components

Templates

Metadata



Modules



Separate Code into Modules



Modules

We assemble our application from modules.

A module exports an asset such as a Service, Component, or a shared value



Angular 1 Modules

```
(function () {  
  angular  
    .module('app', [])  
    .controller('StoryController', StoryController);  
  
  function StoryController() {  
    var vm = this;  
    vm.story = { id: 100, name: 'The Force Awakens' };  
  }  
})();
```

Module definition

Angular 1 modules

Angular 1 had its own modules

Served as containers of controllers, services, and directives



We use ES6 style modules with
Angular 2



```
export interface Vehicle {  
  id: number;  
  name: string;  
}  
  
export class VehicleService {  
  //...  
}
```

Exporting modules

Assets can be exported using the **export** keyword


```
import { Component } from 'angular2/core';  
  
import { Vehicle, VehicleService } from '../vehicle.service';
```

Importing modules

Modules and their contents can be imported using the **import** keyword

We import the Vehicle and VehicleService using destructuring



Components



Angular 2 Components

A Component contains application logic that controls a region of the user interface that we call a view.



Anatomy of a Component

Imports (use other modules)

```
import { Component } from 'angular2/core';  
  
import { Vehicle } from './vehicle.service';
```

```
@Component({  
  selector: 'story-vehicles',  
  templateUrl: './vehicles.component.html'  
})
```

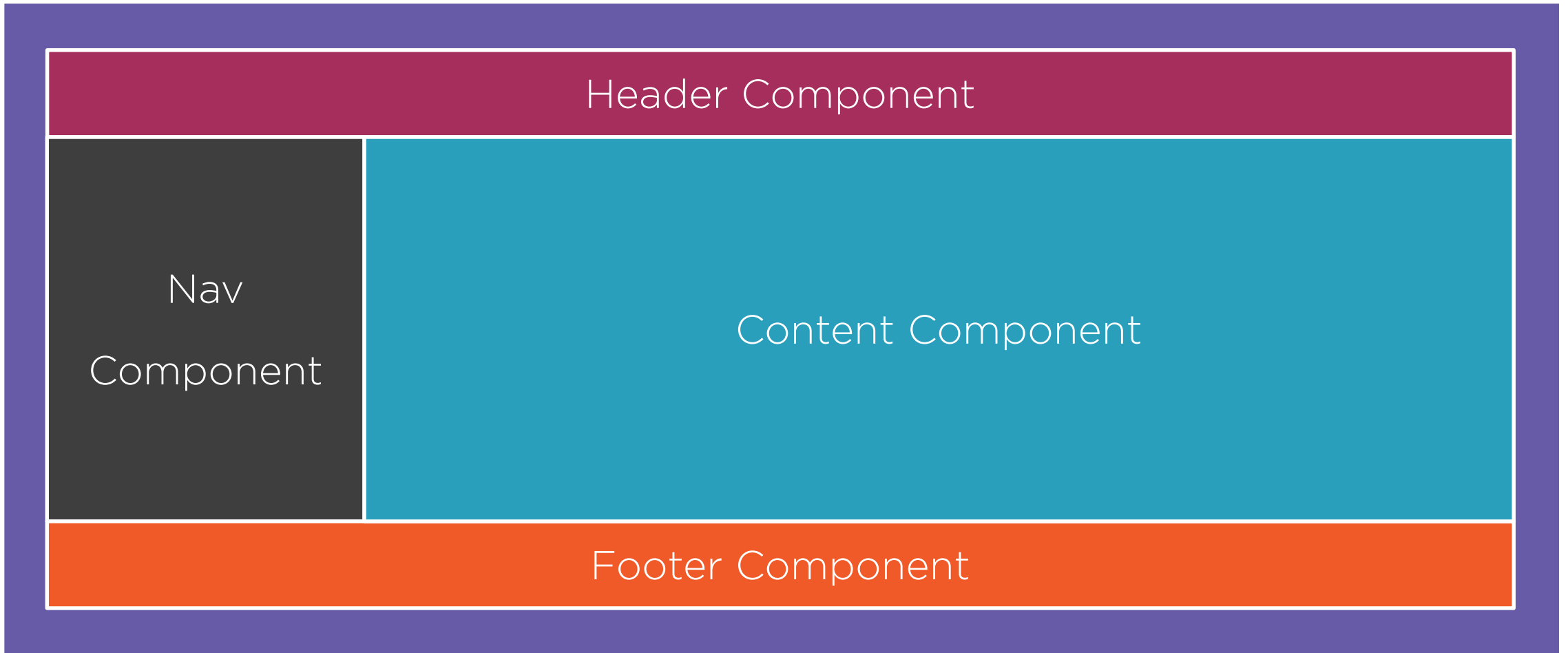
Metadata (describe the component)

Class (define the component)

```
export class VehicleListComponent {  
  vehicles: Vehicle[];  
}
```



Assembling Our App from Components



```
import { bootstrap } from 'angular2/platform/browser';  
import { StoryComponent } from './story.component';  
  
bootstrap(StoryComponent);
```

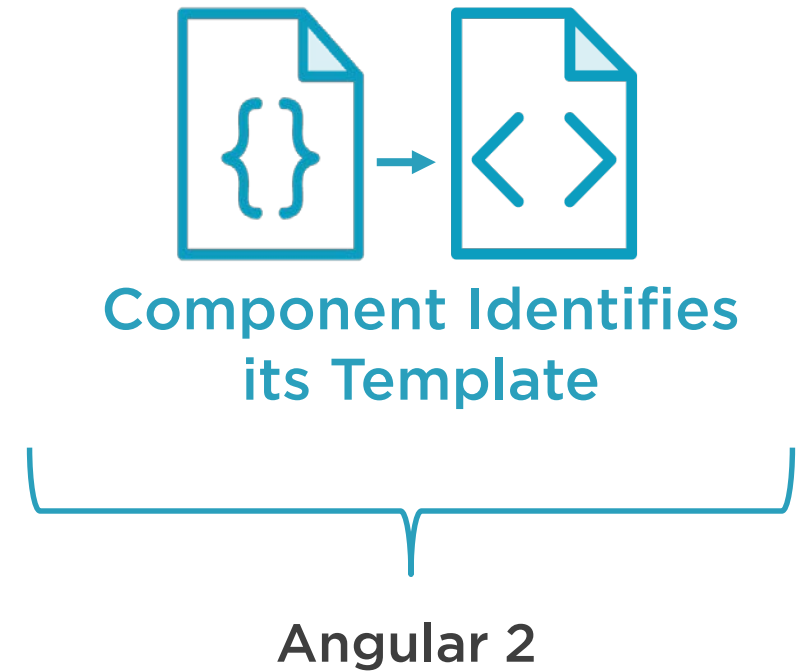
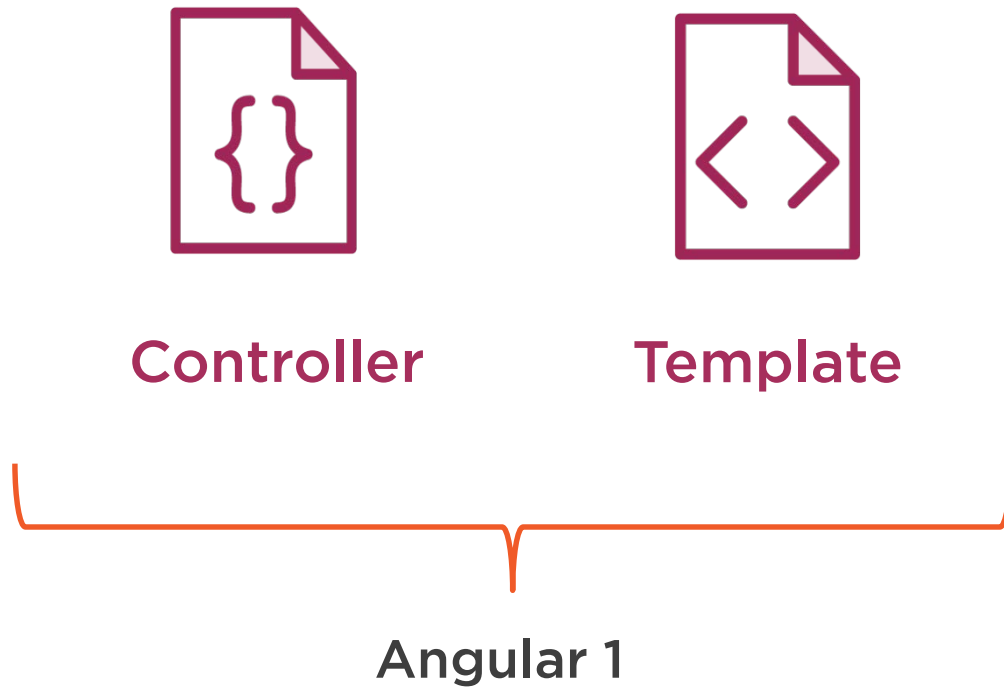
Bootstrapping a Component

Entry point for the app

This is where we start



Comparing Angular 1 to Angular 2



story.component.ts

Component has logic

```
@Component({
  selector: 'my-story',
  templateUrl: 'app/story.component.html'
})
export class StoryComponent {
  name = 'Han Solo';
}
```

story.component.html

What is rendered

```
<h3>My name is {{name}}</h3>
```

index.html

Where the component is placed

```
<my-story></my-story>
```



story.component.ts

Component has logic

```
@Component({  
  selector: 'my-story',  
  templateUrl: 'app/story.component.html'  
})  
export class StoryComponent {  
  name = 'Han Solo';  
}
```

story.component.html

What is rendered

```
<h3>My name is {{name}}</h3>
```

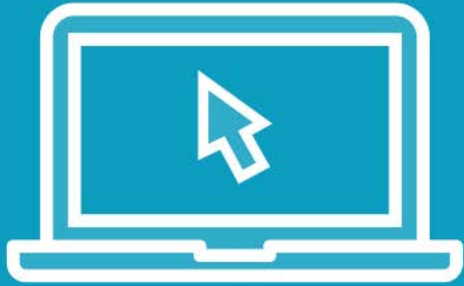
index.html

Where the component is placed

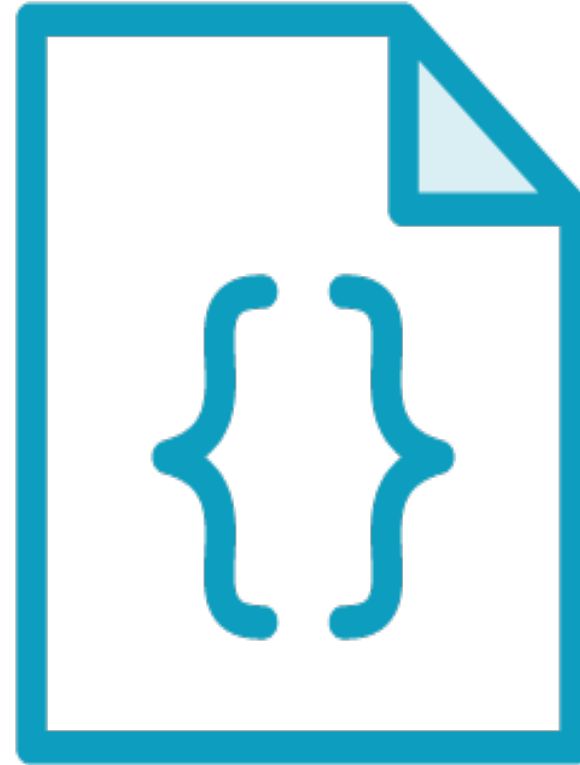
```
<my-story></my-story>
```



Components Demo



Components



Templates



Templates are the View

Templates are mostly HTML, with a little help from Angular. They tell Angular how to render the Component



```
<ul>  
  <li *ngFor="#vehicle of vehicles">  
    {{vehicle.id}}. {{vehicle.name}}  
  </li>  
</ul>  
  
<vehicle *ngIf="selectedVehicle"  
  [vehicle]="selectedVehicle"></vehicle>
```

Directives (e.g. *ngFor)

Interpolation

Nested Component

Templates

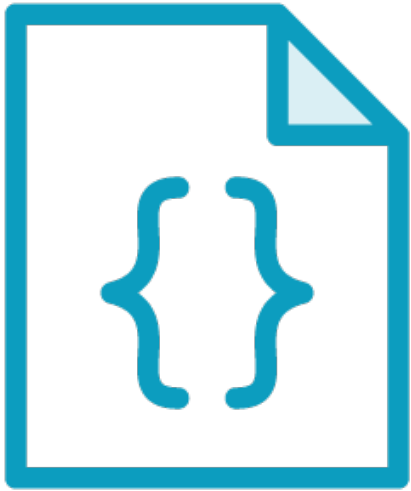
HTML

Directives, as needed

Template Binding Syntax



Connecting the Component to its Template



Component



Template

```
@Component({
  selector: 'story-vehicles',
  template: `
    <ul>
      <li *ngFor="#vehicle of vehicles">
        {{vehicle.name}}
      </li>
    </ul>
  `
})
export class VehicleListComponent { }
```

Template String

Inline Templates

template defines an embedded template string

Use back-ticks for multi-line strings



```
@Component({  
  selector: 'story-vehicles',  
  templateUrl: './vehicles.component.html'  
})  
export class VehicleListComponent { }
```



Template Url

Linked Templates

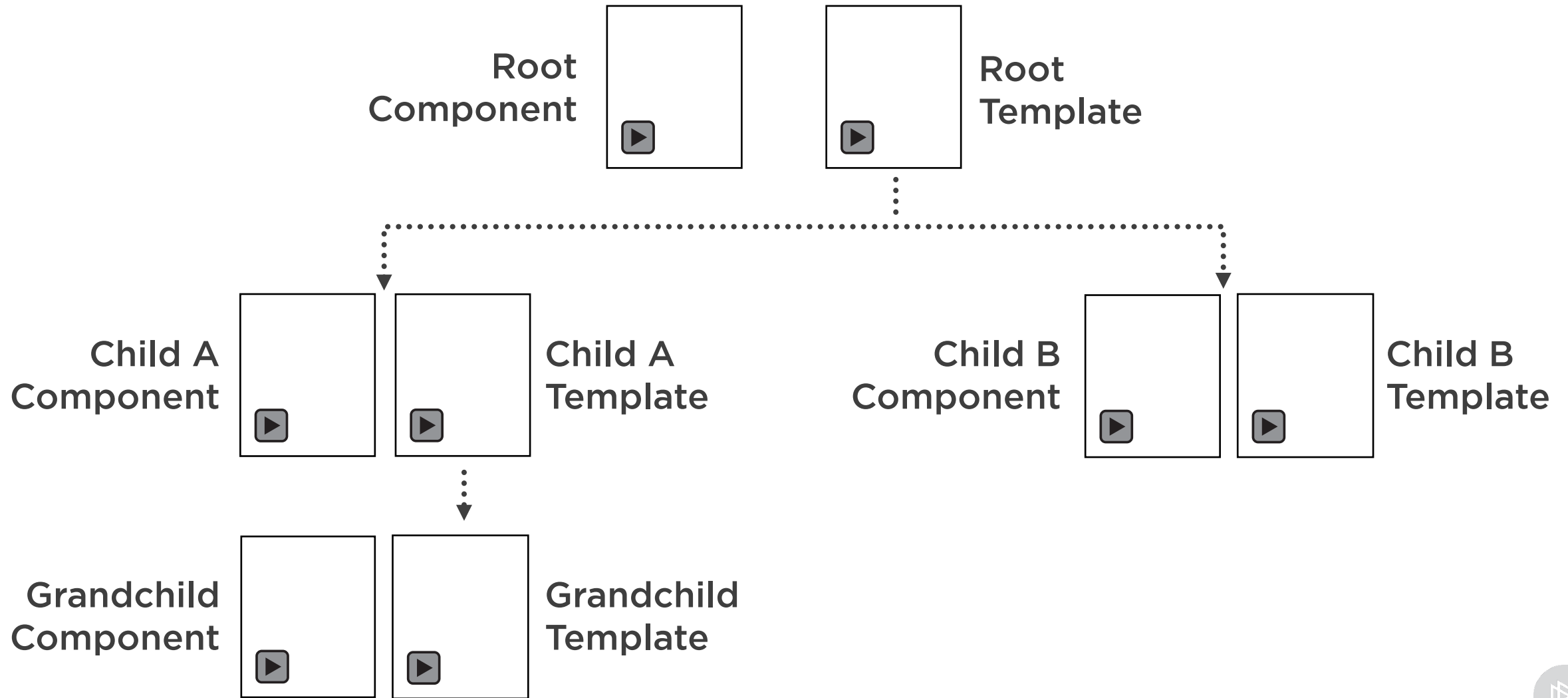
templateUrl links the Component to its Template



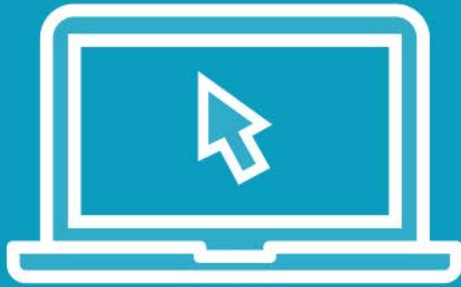
Components have templates,
which may contain other components



Templates Contain Other Components



Demo



Templates



Metadata

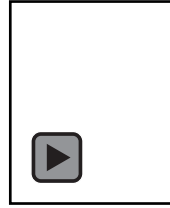


Metadata

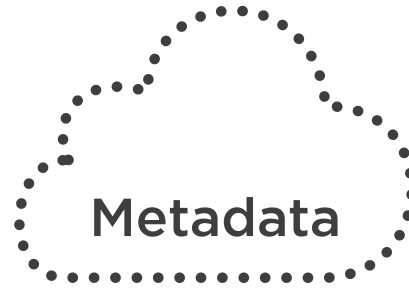
We use Metadata to tell Angular about the objects we build.



Metadata Links the Template to the Component



Template



Component



```
@Component({  
  selector: 'story-characters',  
  templateUrl: './app/characters.component.html',  
  styleUrls: ['./app/characters.component.css'],  
  directives: [CharacterDetailComponent],  
  providers: [HTTP_PROVIDERS, CharacterService]  
})
```

Declare all directives

Nesting Components

Built-in directives like ***ngFor** and **ngClass** are already available

Angular needs to know what **<character-detail>** is in the Template



```
@Component({  
  selector: 'story-characters',  
  templateUrl: './app/characters.component.html',  
  styleUrls: ['./app/characters.component.css'],  
  directives: [CharacterDetailComponent],  
  providers: [HTTP_PROVIDERS, CharacterService]  
})
```

Decorators

metadata for the Component

@Component Decorator

Selector is the name of the element in the Template

templateUrl and **styleUrls** point to the template and styles

directives declare custom directives it uses

providers declare services the Component needs



Examining a Component and its Metadata



Decorators

The @ is a decorator that provides metadata describing the Component

Component

Component definition class. Controls a patch of screen real estate that we call a View

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterComponent],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this._characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```



Template and Styles

Tells the Component where to find them.

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterComponent],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this._characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```



Directives

Tell the Component which directives we will be using in the Template

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterComponent],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this._characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```



Providers

These services will be registered with Angular's injector. Only do this once.

```
@Component({  
  selector: 'story-characters',  
  templateUrl: './app/characters.component.html',  
  styleUrls: ['./app/characters.component.css'],  
  directives: [CharacterComponent],  
  providers: [CharacterService]  
})
```

```
export class CharactersComponent implements OnInit {  
  @Output() changed = new EventEmitter<Character>();  
  @Input() storyId: number;  
  characters: Character[];  
  selectedCharacter: Character;
```

Injectors

Inject a Service into another object.

```
constructor(private _characterService: CharacterService) { }
```

```
ngOnInit() {  
  this._characterService.getCharacters(this.storyId)  
    .subscribe(characters => this.characters = characters);  
}
```

```
select(selectedCharacter: Character) {  
  this.selectedCharacter = selectedCharacter;  
  this.changed.emit(selectedCharacter);  
}  
}
```



Output

Component can communicate to anyone hosting it

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterComponent],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this._characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Emit Events

Component emits events via output



Input

Pass values into
the Component

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterComponent],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this._characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```



Properties

Component exposes properties that can be bound to its Template

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterComponent],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this._characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```



Actions

Functions can be exposed, bound and called by the Template to handle events

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterComponent],
  providers: [CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Character[];
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this._characterService.getCharacters(this.storyId)
      .subscribe(characters => this.characters = characters);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```



Input and Output

Components allow input properties to flow in, while output events allow a child Component to communicate with a parent Component.



Output

```
export class CharactersComponent implements OnInit {  
  @Output() changed = new EventEmitter<Character>();  
  @Input() storyId: number;  
  characters: Character[];  
  selectedCharacter: Character;  
  
  select(selectedCharacter: Character) {  
    this.selectedCharacter = selectedCharacter;  
    this.changed.emit(selectedCharacter);  
  }  
}
```

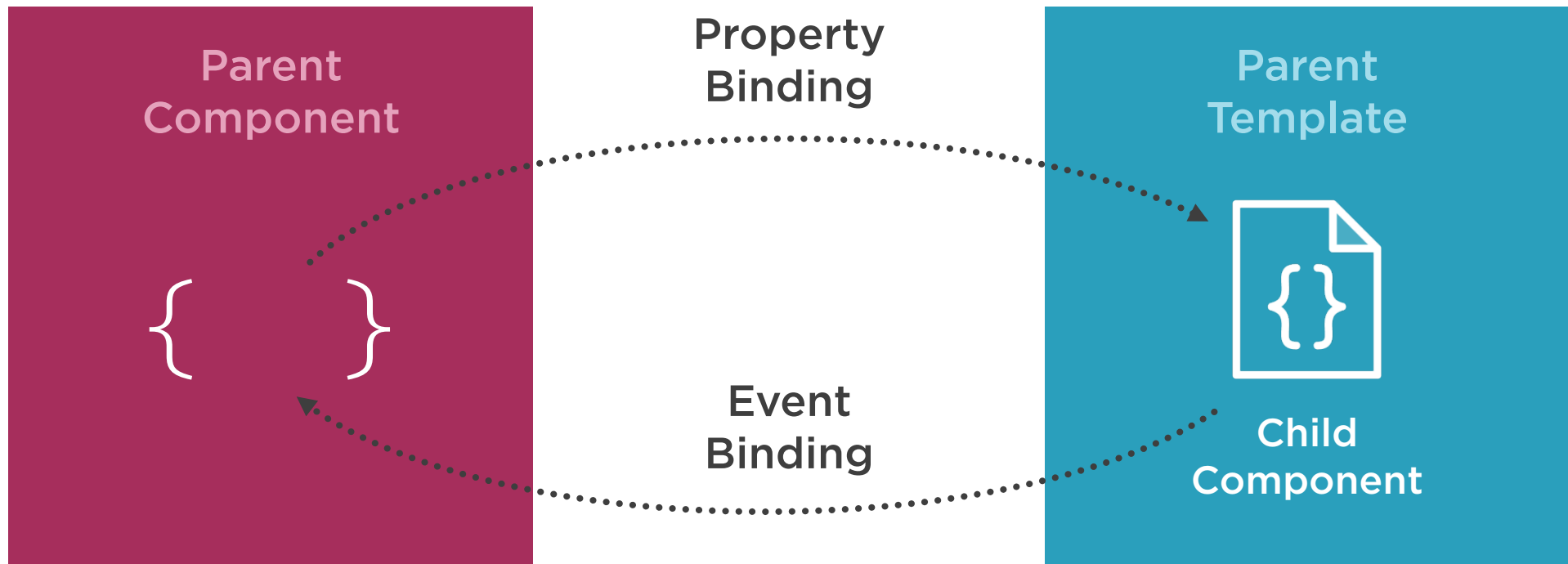
Output property

Emit the output

```
<div>  
  <h1>Storyline Tracker</h1>  
  <h3>Component Demo</h3>  
  <story-characters [storyId]="7"  
    (changed)=changed($event)>  
  </story-characters>  
</div>
```

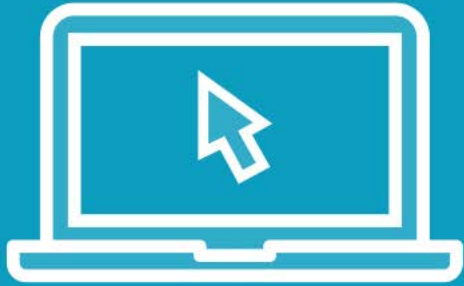
Bind to the event in
the Parent
Component





Component Input and Output

Demo



ViewChild

Use ViewChild when a parent Component needs to access a member of its child Component



ViewChild



Components

Child

```
export class FilterComponent {  
  @Output() changed: EventEmitter<string>;  
  filter: string;  
  
  clear() {  
    this.filter = '';  
  }  
  // ...  
}
```

Child
Component's
function

Parent

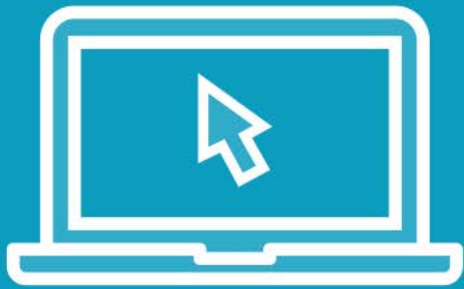
```
export class CharacterListComponent {  
  characters: Character[];  
  @ViewChild(FilterComponent) filter: FilterComponent;  
  
  filtered = this.characters;  
  
  getCharacters() {  
    this._characterService.getCharacters()  
      .subscribe(characters => {  
        this.characters = this.filtered = characters;  
        this.filter.clear();  
      });  
  }  
  
  // ...  
}
```

Grab the child

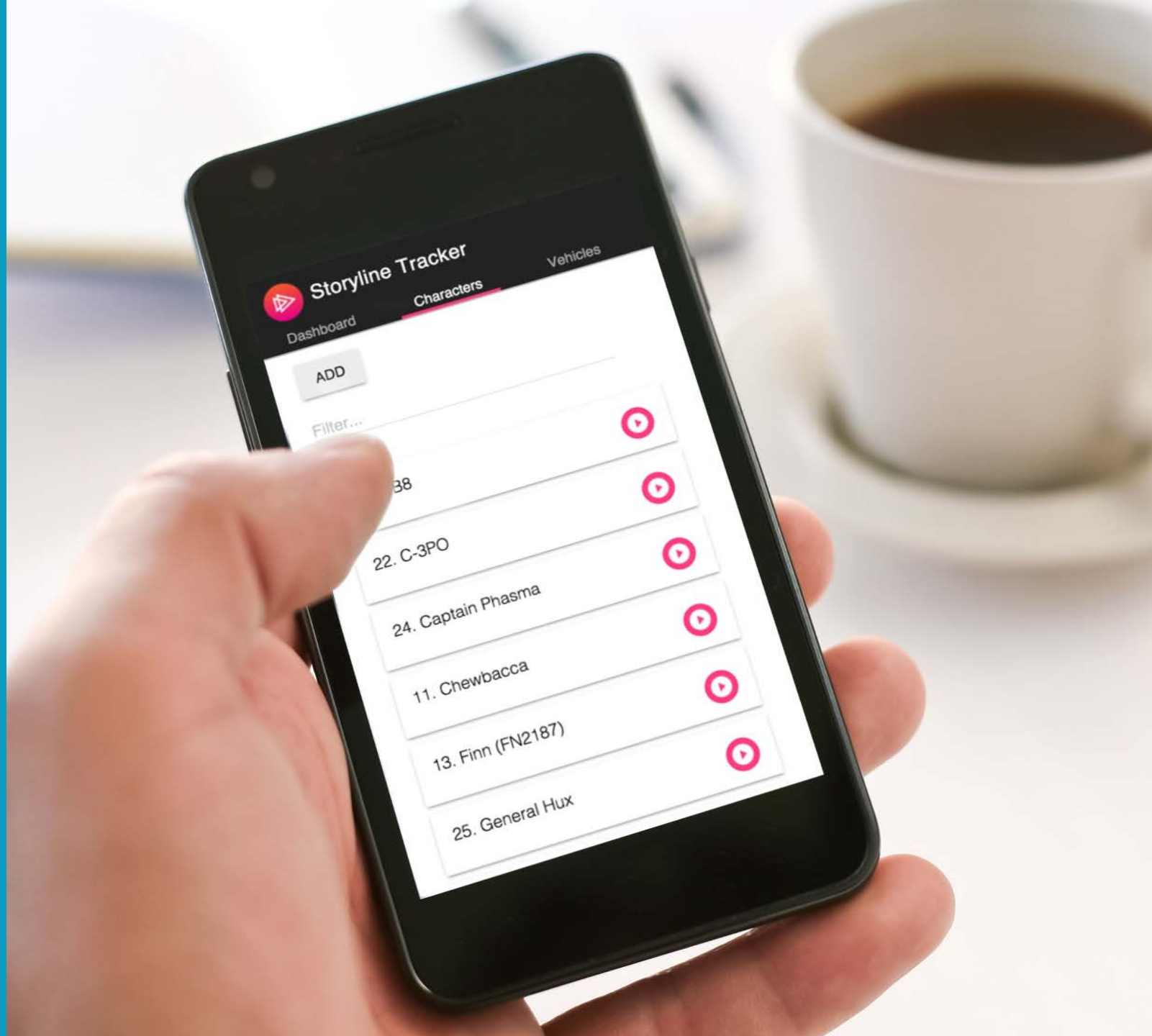
Call its member



Demo



Putting It All Together



Summary



Modules encapsulate like objects

Components control a region of the page

Templates tell Angular how to render

Metadata describes objects

