Angular 2 Components – Displaying Data, Handling User Input, Forms and Validation. Services and DI

Trayan Iliev

IPT – Intellectual Products & Technologies
e-mail: tiliev@iproduct.org
web: http://www.iproduct.org

Agenda I

1. Displaying data using template interpolation and directives: *NgModel, NgFor, NgIf, NgSwitch, NgStyle, NgClass*.
2. Handling user input – binding event handlers, getting data from the *$event* object, using local template variables, event filtering, handling multiple events
3. Building forms – two-way data binding: [(ngModel)]
4. Change tracking, validation, and error handling: ngControl, providing custom styles for different control states, resetting and submitting forms, edit/display forms.
5. Template-driven forms – NgForm, NgForm, NgModelGroup

Agenda II

6. Reactive forms – ReactiveFormsModule: FormControlName, FormGroupName, FormArrayName, FormControlDirective, FormGroupDirective, FormBuilder, etc.
7. Using form validators – RequiredValidator, PatternValidator MaxLengthValidator, MinLengthValidator,
8. Directives and Validators class.
9. Building custom validator directives by implementing Validator interface.
10. Building asynchronous form validators.
11. Angular 2 services - building custom services

Agenda III

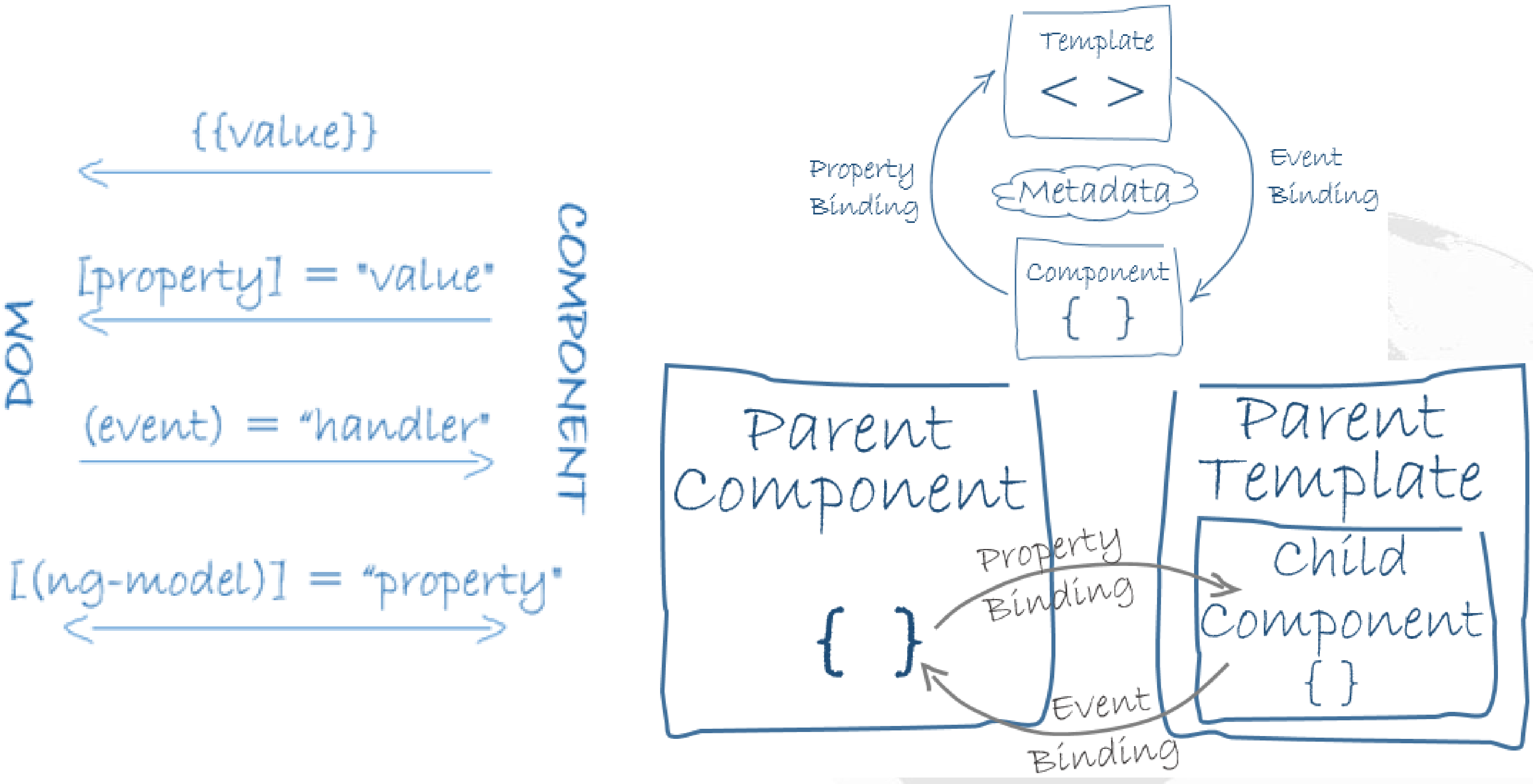
12. Understanding Angular 2 Dependency Injection (DI)
13. Configuring the injector
14. Registering providers in modules and components
15. Declarative and programmatic dependency injection
16. Injecting service dependencies using @Injectable() decorator
17. Registering custom providers using provide function, dependency injection tokens
18. Using hierarchical injectors

Where is The Code?

Angular 2 and TypeScript Web App Development code is available @GitHub:

https://github.com/iproduct/course-angular2

Angular 2 Data and Event Bindings



Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html
License: CC BY 4.0.

Displaying Data: ViewModel → View

- Using **template interpolation**:

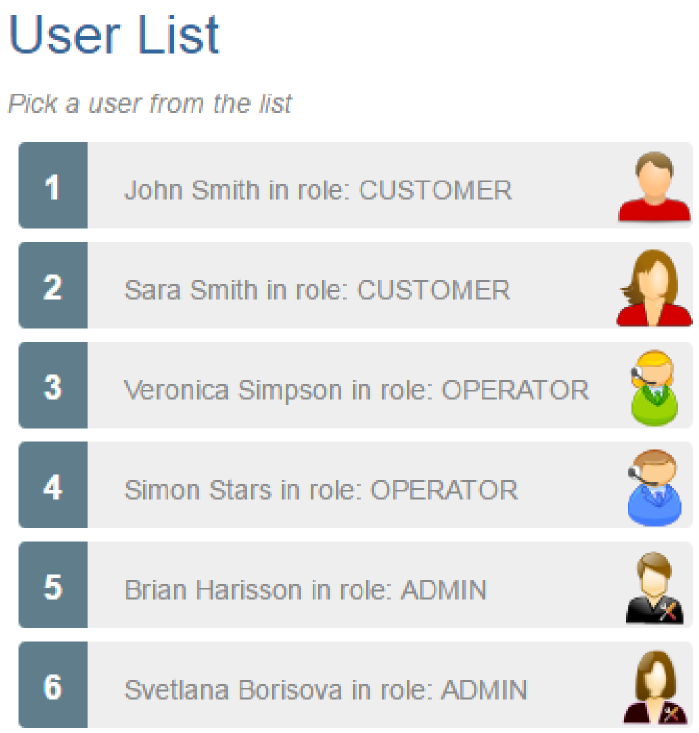
```
<span class="badge">{{product.id}}</span> {{product.name}}
- {{product.price | currency:'USD':true:'1.2-2'}}
```

- Using built-in **directives**:
  - **NgModel**: `<input [(ngModel)]="product.name">`
  - **NgFor**: `<li *ngFor="let product of products"></li>`
  - **NgIf**: `<prod-detail *ngIf="selectedProd"></prod-detail>`
  - **NgSwitch + *ngSwitchCase + *ngSwitchDefault**
  - **Style binding** or **NgStyle**
  - **Class binding** or **NgClass**

NgSwitch + *ngSwitchCase + *ngSwitchDefault

```
<span class="user-icon" [ngSwitch]="user.role + user.gender">
  <span *ngSwitchCase="CUSTOMER_MALE">
        <img [src]="customerMaleImage"></span>
  <span *ngSwitchCase="CUSTOMER_FEMALE">
        <img [src]="customerFemaleImage"></span>
  <span *ngSwitchCase="OPERATOR_MALE">
        <img [src]="operatorMaleImage"></span>
  <span *ngSwitchCase="OPERATOR_FEMALE">
        <img [src]="operatorFemaleImage"></span>
  <span *ngSwitchCase="ADMIN_MALE">
        <img [src]="adminMaleImage"></span>
  <span *ngSwitchCase="ADMIN_FEMALE">
        <img [src]="adminFemaleImage"></span>
  <span *ngSwitchDefault>Other</span>
</span>
```

NgSwitch + *ngSwitchCase + *ngSwitchDefault



User List

Pick a user from the list

1 John Smith in role: CUSTOMER

2 Sara Smith in role: CUSTOMER

3 Veronica Simpson in role: OPERATOR

4 Simon Stars in role: OPERATOR

5 Brian Harisson in role: ADMIN

6 Svetlana Borisova in role: ADMIN

Style Binding or NgStyle

- Style property binding:

```
<div [style.display]="isVisible ? 'block' : 'none'">
 Show me with style  </div>
```

- Using NgStyle directive:

```
<input type="text" name="name" [ngStyle]="getStyles()"
[(ngModel)]="product.name" />
```

```
getStyles() {
    return {
        'color': this.isValid ? 'black' : 'red',
        'font-weight': this.isChanged ? 'bold' : 'normal'
    };
}
```

Class Binding or NgClass

- Class property binding:

```
<div [class.custom]="isCustom">Show me with class</div>
```

- Using NgClass directive:

```
<input type="text" name="name" [ngClass]="getClasses()"
 [(ngModel)]="product.name" />
```

```
getClasses() {
  return {
    modifiable: this.canModify,
    changed: this.isChanged,
    invalid: !this.isValid
  };
}
```

Handling User Input

- Binding event handlers:

```
<button (click)=calcTax(amount.value)>CalculateTax</button>
```

- Getting data from the **$event** object:

```
<input (keyup)="onKey($event)">
onKey(event:any) { this.value += event.target.value; }
```

- Using **template reference variables**:

```
<input #amount (change)="0">
<div>{{amount.value}}</div>
```

- Event filtering:

```
<input #amount (keyup.enter)="calcTax(amount.value)">
```

- Handling multiple events: `<input #amount (keyup.enter)= "calcTax(amount.value)" (blur)="calcTax(amount.value)">`

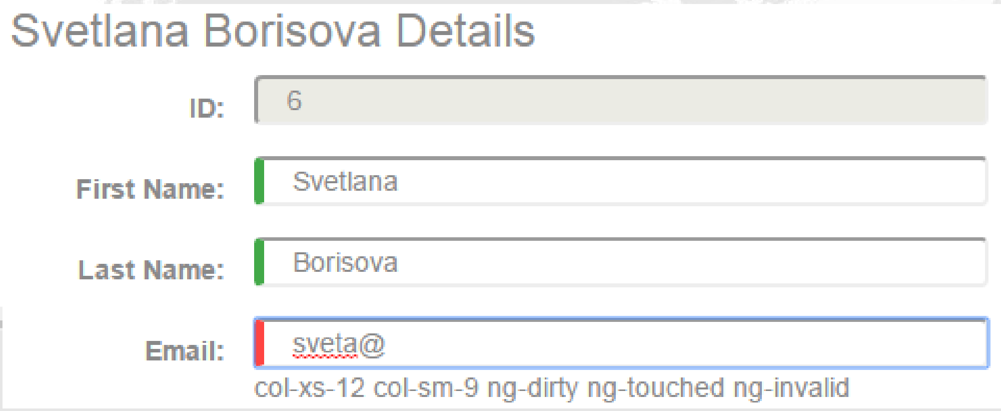
Building Template-Driven Forms using **ngModel**

- Two-way data binding with **ngModel**:

```
<input [(ngModel)]="product.name" id="name" name="name">
Diagnostic data: {{product.name}}
<input type="text" id="name" name="name" required
  [ngModel]="product.name"
  (ngModelChange)="product.name = $event">
```

- Change tracking, validation, and error handling with **ngModel**:

```
<input [(ngModel)]="user.email" id="email" name="email"
required pattern="^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-
Za-z]{2,}$" #email>
<p>{{email.className}}</p>
```


Svetlana Borisova Details

ID:	6
First Name:	Svetlana
Last Name:	Borisova
Email:	sveta@

col-xs-12 col-sm-9 ng-dirty ng-touched ng-invalid

Angular Forms

- Providing custom styles for different control states

Control/Form State	*Class if **True***	*Class if **False***
Control has been visited	**ng-touched**	**ng-untouched**
Control's value changed	**ng-dirty**	**ng-pristine**
Control's value valid	**ng-valid**	**ng-invalid**

- Using **NgForm** / **NgModelGroup** directives
- Resetting and submitting forms
- Add/ Edit/ Display forms
- Reactive forms using **ReactiveFormsModule** directives: FormArrayName, FormControlDirective, FormGroupDirective, FormBuilder, RequiredValidator, PatternValidator, etc.

Building Forms – Example:

```
template: `
    <form #f="ngForm" (ngSubmit)="onSubmit(f)">
      <p *ngIf="nameCtrl.invalid">Name is invalid.</p>
      <div ngModelGroup="name" #nameCtrl="ngModelGroup">
        <input name="first" [ngModel]="name.first" minlength="2">
        <input name="last" [ngModel]="name.last" required>
      </div>
      <input name="email" ngModel> <button>Submit</button>
    </form> <button (click)="setValue()">Set value</button>
  `

export class SimpleFormComponent {
  public name = {first: 'John', last: 'Smith'};
  public onSubmit(f:NgForm){console.log(f.value); console.log(f.valid);}
  public setValue() { this.name = {first: 'Brian', last: 'Adams'}; }
}
```

Angular Standard Validators

- To validate user input, add HTML validation attributes to the elements → Angular adds validator functions to control model:

Attribute	*Input types supported*	*Constraint description*
required	text, search, url, email, tel, date, time, password, checkbox, number, radio, file, <select>, <textarea>	There must be a value
minlength	text, search, url, password, tel, email, <textarea>	The number of characters must not be less than the value
maxlength	text, search, url, password, tel, email, <textarea>	The number of characters must not exceed the value
email	email	Value must be a valid email
pattern	text, search, url, tel, email, password	The value must match the pattern

Custom Validators

[https://angular.io/docs/ts/latest/cookbook/form-validation.html#custom-validation]

```
import { Control, ValidatorFn, AsyncValidatorFn } from
"@angular/forms";
interface ValidationErrors{
   [key: string]: any;
}
export function forbiddenNameValidator(nameRe: RegExp):
ValidatorFn {
  return (control: AbstractControl): ValidationErrors => {
    const name = control.value;
    const no = nameRe.test(name);
    return no ? {forbiddenName: {name}} : null;
  };
}
```

Custom Validators Async (Promise)

[https://github.com/daviddt/angular2-form-validation-example/]

```
export function usernameTakenValidator(): AsyncValidatorFn {
  return (control: AbstractControl):Promise<ValidationErrors> =>
  {
    return new Promise((resolve, reject) => {
         setTimeout(() => {
             if (control.value === "John") {
                 resolve({ "usernameTaken": true })
             } else {
                 resolve(null);
             };
         }, 1000);
      });
  };
}
```

Custom Validation Directive - I

[https://angular.io/docs/ts/latest/cookbook/form-validation.html#custom-validation]

```
@Directive({
  selector: '[forbiddenName]',
  providers: [{provide: NG_VALIDATORS,
         useExisting: ForbiddenValidatorDirective, multi: true}]
})
export class ForbiddenValidatorDirective implements Validator,
OnChanges {
  @Input() private forbiddenName: string;
  private valFn = Validators.nullValidator;

  public validate(control: AbstractControl): ValidationErrors {
    return this.valFn(control);
  }
  ...
```

Custom Validation Directive - II

[https://angular.io/docs/ts/latest/cookbook/form-validation.html#custom-validation]

```
  ...
  public ngOnChanges(changes: SimpleChanges): void {
    const change = changes['forbiddenName'];
    if (change) {
      const val: string | RegExp = change.currentValue;
      const re = val instanceof RegExp ? val
                                       : new RegExp(val, 'i');
      this.valFn = forbiddenNameValidator(re);
    } else {
      this.valFn = Validators.nullValidator;
    }
  }
}
```

Validation Directive (Template Driven Form)

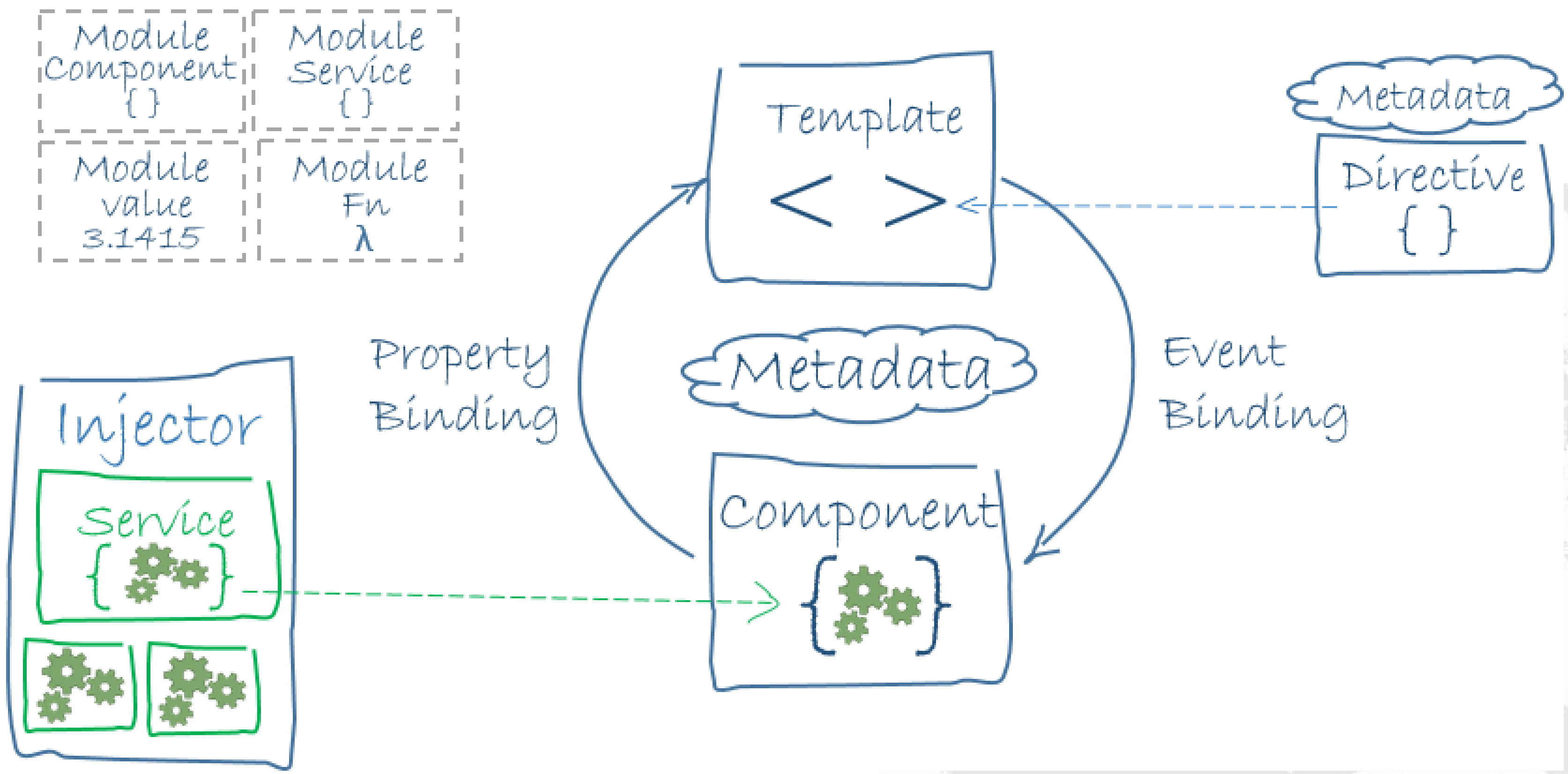
[https://angular.io/docs/ts/latest/cookbook/form-validation.html#custom-validation]

```
<form #heroForm="ngForm"  (ngSubmit)="onSubmit()">
  <div class="form-group">
    <label for="name">Name</label>

    <input type="text" id="name" class="form-control"
       required minlength="4" maxlength="24" forbiddenName="sam"
       name="name" [(ngModel)]="hero.name" >

    <div *ngIf="formErrors.name" class="alert alert-danger">
       {{ formErrors.name }}
    </div>
  </div>
  ...
</form>
```

Using Custom Validators (Reactive Form)

```
constructor(private builder: FormBuilder) {
    this.username = new FormControl(
      "",
      [ Validators.required,   //Sync validators
        Validators.minLength(3),
        Validators.maxLength(25),
        forbiddenNameValidator(/sam/i)],
      usernameTakenValidator() //Async validator
    );

    this.form = builder.group({
        username: this.username
    });
}
```

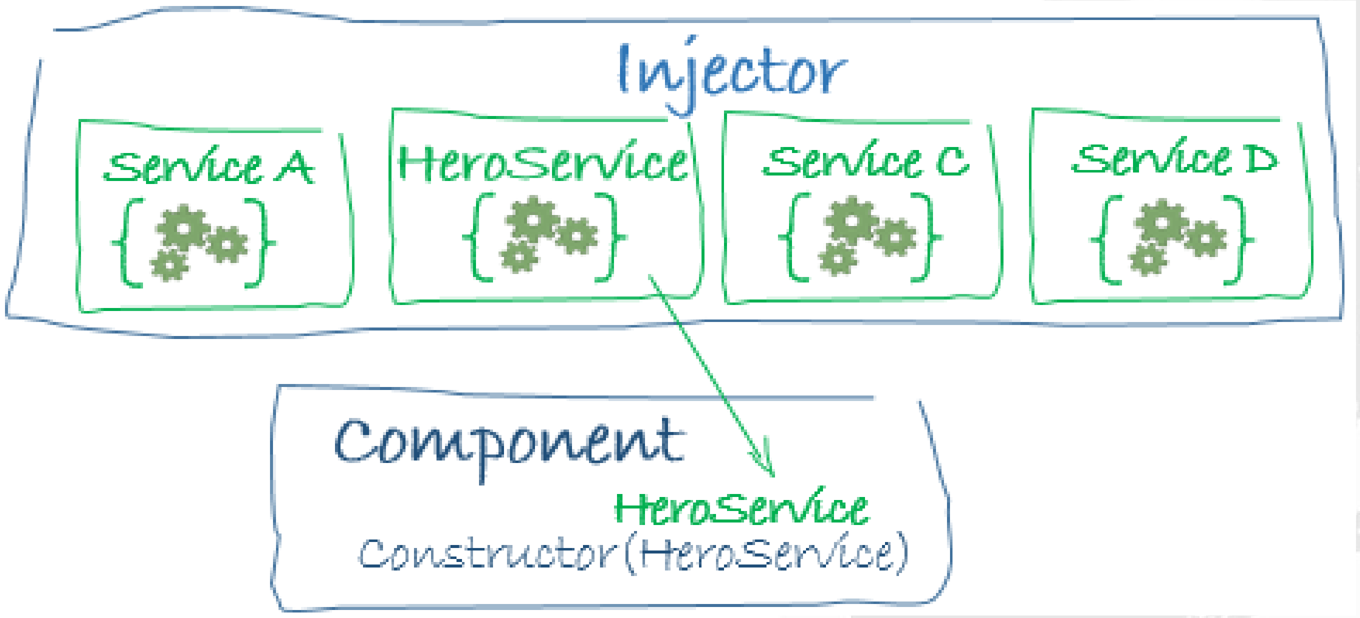
Angular 2 Services



Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html
License: CC BY 4.0.

Angular 2 Dependency Injection (DI)

- Dependency Injection (DI) using constructors
- Hierarchical dependency injectors – module or component provided services



Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html
License: CC BY 4.0.

Simple Service Example – logger.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class Logger {
  public log(msg: any)   { console.log(msg); }
  public error(msg: any)
{ console.error(msg); }
  public warn(msg: any)  { console.warn(msg); }
}
```

Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html

Backend Service Example – backend.service.ts

```
import { Injectable, Type } from '@angular/core';
import { Logger } from './logger.service';
import { Product } from './product.model';
const PRODUCTS = [new Product('Logitech Mouse', 12.99, 'Super mouse'),
        new Product('Whiteboard Marker', 0.32, 'Drawing is fun!')];
@Injectable()
export class BackendService {
  constructor(private logger: Logger){}← Dependency Injection (DI)
  public getAll(type: Type<any>): Promise<any> {
    if (type === Product) {
      return Promise.resolve(PRODUCTS);  // TODO get from the database
    }
    let err = new Error(`Cannot get object of this type : ${type}`);
    return Promise.reject(err);
  }
}
```

Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html, License: CC BY 4.0.

Product Service Example – product.service.ts

```
import { Injectable } from '@angular/core';
import { Product } from './product.model';
import { BackendService } from './backend.service';
import { Logger } from './logger.service';
@Injectable()
export class ProductService {
  private products: Product[] = [];
  constructor(private backend: BackendService, private logger: Logger){}
  public getProducts() {
    this.backend.getAll(Product).then( (products: Product[]) => {
      this.logger.log(`Fetched ${products.length} products.`);
      this.products.push(...products); // fill cache
    }).catch(
      err => this.logger.error(`ProductService Error: ` + err)
    );
    return this.products;
  }
}
```

Dependency Injection (DI)

Injecting Service into Component

```
import { Component, OnInit } from '@angular/core';
import { Product } from './product.model';
import { ProductService } from './product.service';
@Component({
  selector: 'product-list',
  templateUrl: './product-list.component.html',
  providers: [ProductService]    ⟵ Registering DI Providers
})
export class ProductListComponent implements OnInit {
  public products: Product[];
  public selectedProd: Product;
  constructor(private service: ProductService) { }
  public ngOnInit() {
    this.products = this.service.getProducts();
  }
  public selectProduct(product: Product){this.selectedProd = product;}
}
```

Dependency Injection (DI)

Components Testability and DI

```
let expectedProducts = [new Product('Test Product', 10.50, 'Test
product description')]

let mockService = <ProductService> {
    getProducts: () => expectedProducts
}

it('should have products when ProductListComponent created', () => {
  let plc = new ProductListComponent(mockService);
  expect(plc.products.length).toEqual(expectedProducts.length);
});
```

DI Providers

- Provider class and **provide** object literal:
  - `providers: [Logger]`
  - `[{ provide: Logger, useClass: Logger }]`
  - `[{ provide: Logger, useClass: SuperiorLogger }]`

  -
    ```
    [ SuperiorLogger,
     { provide: Logger, useExisting: SuperiorLogger}]
    ```

  -
    ```
    [{ provide: Logger, useValue: { log: (message) => {
        console.log(`Custom: ${message}`); } }]
    ```

  -
    ```
    [{ provide: ProductService, useFactory:
    productServiceFactory, deps: [Logger, UserService]}]
    ```

Source: Angular 2 Developer Guide: Architecture overview
https://angular.io/docs/ts/latest/guide/architecture.html

Declarative and Programatic DI

- Declarative dependency injection:

```
constructor(productService: ProductService) { … }
```

- Programmatic dependency injection:

```
productService: productService = this.injector.get(ProductService);
```

- Programmatic DI example:

```
const injector: Injector =    ReflectiveInjector.resolveAndCreate(
      [{provide: 'token', useValue: 'Value'}]);
expect(injector.get('token')).toEqual('Value');
expect(() => injector.get('noSuchToken')).toThrowError();
expect(injector.get('noSuchToken','Not Found')).toEqual('Not Found');
```

DI Tokens

- Dependency injection tokens – *can not inject by interface types*:

```
export interface ProductServiceConfig {
  productServiceUrl: string;
}
export const PRODUCT_DI_CONFIG: ProductServiceConfig = {
  productServiceUrl: 'http://localhost:8080/api'  };
import { OpaqueToken } from '@angular/core';
export let PROD_CONFIG = new OpaqueToken('product.config');
providers: [{ provide: PROD_CONFIG, useValue: PRODUCT_DI_CONFIG }]
constructor(@Inject(PROD_CONFIG) config: ProductServiceConfig) {
  this.productServiceUrl = config.productServiceUrl;
}
```

DI Tokens

- Dependency injection tokens – *can not inject by interface types*:

```
export interface ProductServiceConfig {
  productServiceUrl: string;
}
export const PRODUCT_DI_CONFIG: ProductServiceConfig = {
  productServiceUrl: 'http://localhost:8080/api'  };
import { InjectionToken } from '@angular/core';
export let PROD_CONFIG =
       new InjectionToken<ProductServiceConfig>('product.config');
providers: [{ provide: PROD_CONFIG, useValue: PRODUCT_DI_CONFIG }]
constructor(@Inject(PROD_CONFIG) config: ProductServiceConfig) {
   this.productServiceUrl = config.productServiceUrl;
}
```

New in Angular 4.x (typesafe DI)

Thanks for Your Attention!

Questions?