ANGULAR 2

One framework for web, mobile and desktop Apps

About Me

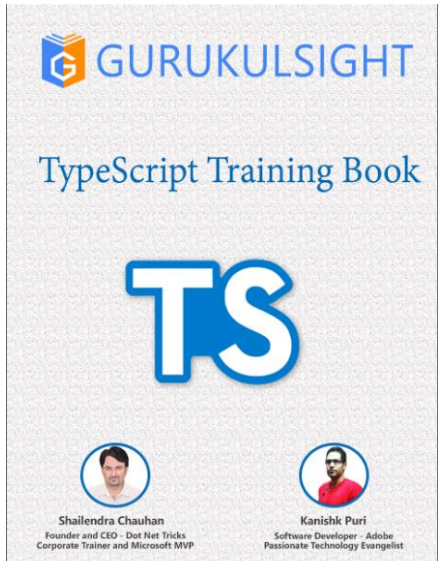
Hi, I'm Shailendra Chauhan

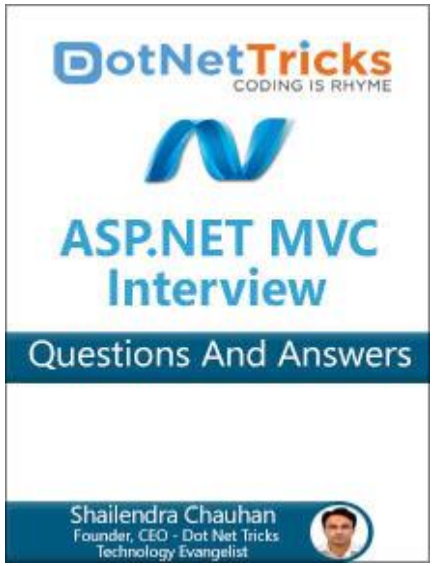
- Author
- Architect,
- Corporate Trainer
- Microsoft MVP
- Founder and CEO of Dot Net Tricks

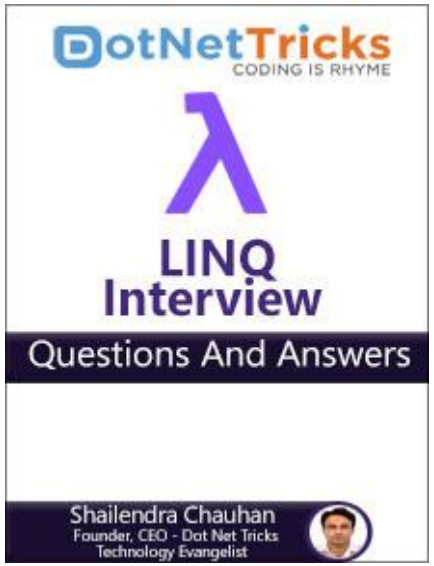
@proshailendra

@proshailendrachauhan

Author of Most Popular Free e-Books







Agenda

- Routing
- Directives
- Pipes
- Custom Directive
- Nested Component
- Components Inheritance
- Component Lifecycle Hooks

Routing

- Three main components are used to configure routing :
  - **Routes** describe the routes

```
const routes: Routes = [
    { path: '', component: DatabindingComponent },
    { path: 'about/:id', component: AboutComponent },
    { path: '**', redirectTo: 'notfound' },
    { path: 'notfound', component: NotfoundComponent }
];
```

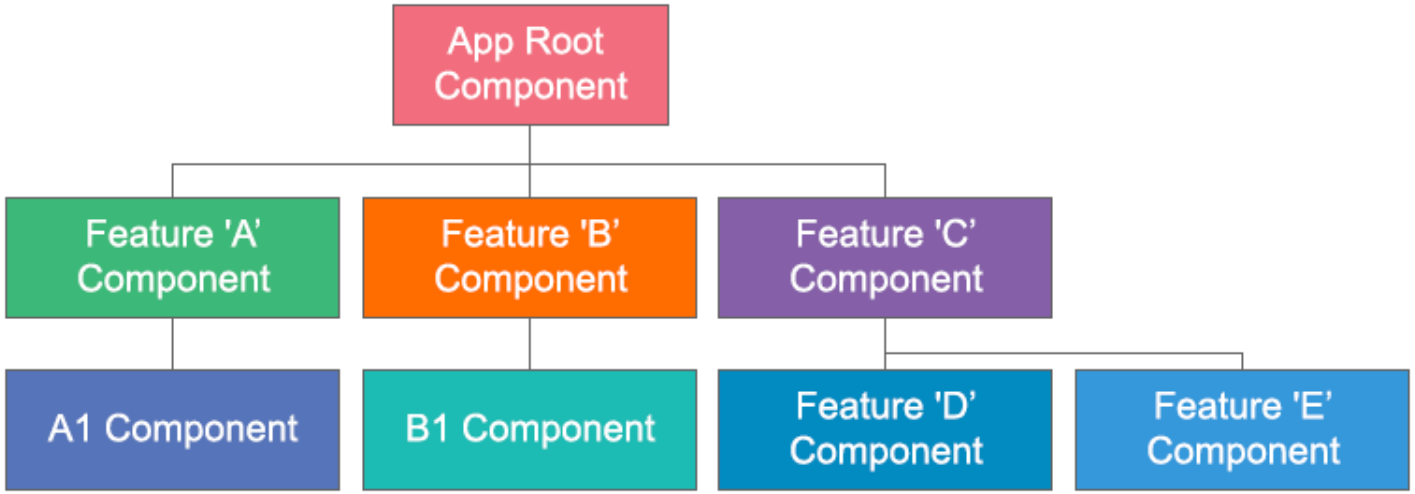
  - **RouterOutlet** is where the router render the component

```
<div class="container">
  <router-outlet></router-outlet>
</div>
```

  - **RouterLink** a link to a route name

```
<a [routerLink]="['/about','4']">About</a>
```

Child Routing




```
const routes: Routes = [
    { path: '', component: DatabindingComponent },
    {
        path: 'dir', component: BuitindirComponent,
        children: [
            { path: 'details', component: DetailsComponent },
            { path: 'about/:id', component: AboutComponent }
            ]
    }
];
```

Lazy Loading

- Angular2 Supports lazy loading for module
- Lazy loading is easy to use and easy to configure
- For supporting lazy loading change component to loadChildren

```
const routes: Routes = [
    { path: 'lazy', loadChildren: 'app/lazy/lazy.module#LazyModule' }
];
```

- Lazy module loading is useful for developing more than one area(modules) in an application

Router Guard

- Router Guard are used to protect routes from unauthorize access
- Guards control how the user navigates in Angular App
- Guards are functions which are called when router tries to activate or deactivate certain routes
- Angular supports following guards:
  - CanActivate - check route access
  - CanActivateChild - check child route access
  - CanDeactivate - ask permission to discard unsaved changes
  - CanLoad - check before loading feature module assets

Directives

- Html elements or attributes used to extend the power of Html
- A directive is a class with directive metadata
- There are four kinds of directives in Angular2
  - Components – A directive with a template. A @Component decorator is actually a @Directive decorator extended with template-oriented features.
  - Structural directives – Alter layout by adding, removing, and replacing elements in DOM. Eg. *ngIf, *ngFor, *ngSwitch
  - Attribute directives – Alter the appearance or behavior of an existing element. Eg. ngModel, ngStyle, ngClass
  - Custom directives – Your own directive

Custom Directive

- *Directive* decorator is used to create a custom directive
- Custom directive can be used as an attribute and class

```
@Directive({
  selector: '[appCustomDirective]'
})
export class CustomDirective {
  constructor(private elementRef: ElementRef, private renderer: Renderer) {
    this.renderer.setElementStyle(this.elementRef.nativeElement,
      'background-color', 'green');
  }
}
```

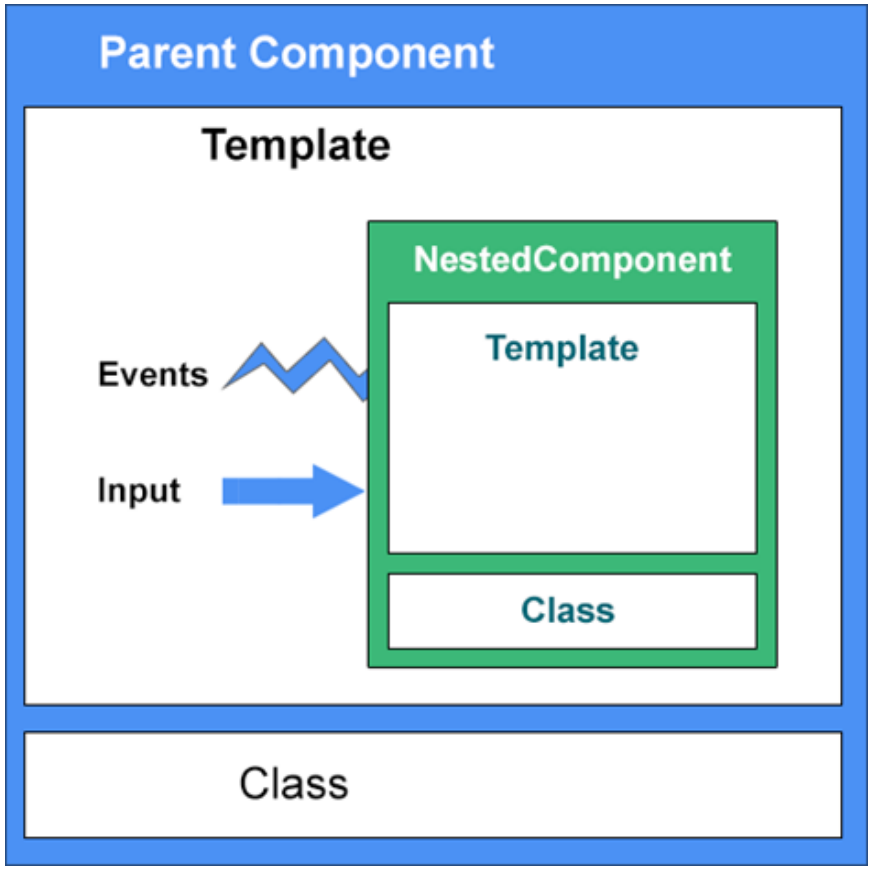
Pipes

- Used to transform bound properties before displaying
- Built-In Pipes are – lowercase, uppercase, date, currency, json, decimal, async, slice etc.
- Custom Pipes

```
@Pipe({
  name: 'reverse'
})
export class ReversePipe implements PipeTransform {
  transform(value: any, args?: any): any {
    let result = '';
    for (let i = 0; i < value.length; i++) {
      result = value[i] + result;
    }
    return result;
  }
}
```

Nested Components

- Nested components can be created to create master/detail view
- @Input decorator is used to pass data from parent to child component
- @Output decorator is used to pass data from child to parent with the help of events

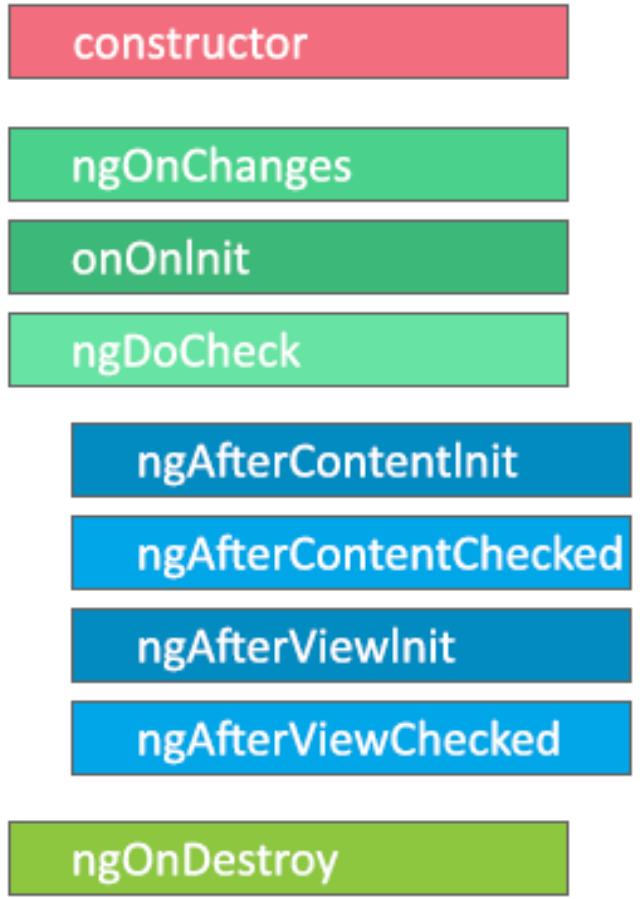


Components Inheritance

- Introduced with Angular 2.3
- Improve code reusability and speed up development
- Component Inheritance covers all of the following :
  - **Metadata (decorators)**: metadata defined in a derived class will override any previous metadata in the inheritance chain otherwise the base class metadata will be used.
  - **Constructor**: the base class constructor will be used if the derived class doesn't have one.
  - **Lifecycle hooks**: parent lifecycle hooks will be called even when are not defined in the derived class

Component Lifecycle Hooks

- Component lifecycle is managed by Angular itself
- Angular manages creation, rendering, data-bound properties etc.
- Angular offers following lifecycle hooks that allow to execute custom logic
- The hooks are executed in the order as shown



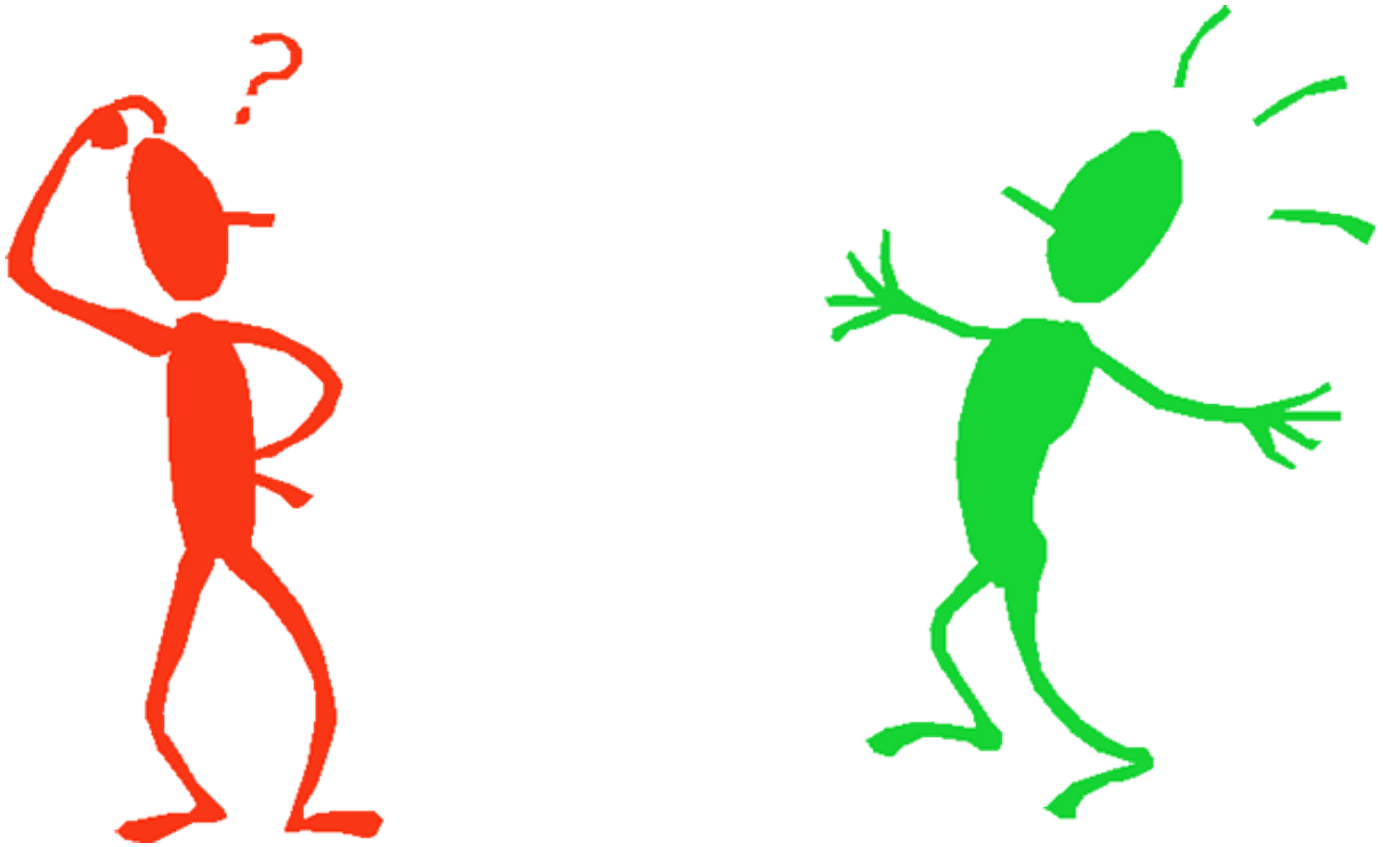
Hooks for the Component

- **constructor** - Invoked when Angular creates a component or directive by calling *new* on the class.
- **ngOnChanges** - Invoked every time there is a change in one of the input properties of the component.
- **ngOnInit** - Invoked when given component is initialized. This hook is only called once after the first ngOnChanges.
- **ngDoCheck** - Invoked when the change detector of the given component is invoked. It allows us to implement our own change detection algorithm for the given component.
- **ngOnDestroy** - Invoked just before destroying the component. Useful to unsubscribe observables and detach event handlers to avoid memory leaks.

Hooks for the Component's Child

- **ngAfterContentInit** - Invoked after the content of the given component is initialized.
- **ngAfterContentChecked** - Invoked each time the content of the given component is checked by the Angular change detection mechanism.
- **ngAfterViewInit** - Invoked after the component's view(s) is initialized.
- **ngAfterViewChecked** - Invoked each time the view of the given component is checked by the Angular change detection mechanism.
- **Note -** These hooks are only called for components and not for directives.

Q&A




THANK
YOU !