AngularCLI

Command line inferface for Angular - set of commands that will help us during development.

1. Setup

Command	Description
npm install -g @angular/cli	Install Angular CLI globally

2. New application

Command	Description
ng new best-practises --dry-run	just simulate ng new
ng new best-practises --skip-install	skip install means don't run npm install
ng new best-practises --prefix best	set prefix to best
ng new --help	check available command list

3. Lint - check and make sure that our code if free of code smells/ bad formatting

Command	Description
ng lint my-app --help	check available command list
ng lint my-app --format stylish	format code
ng lint my-app --fix	fix code smells
ng lint my-app	show warnings

4. Blueprints

Command	Description
ng g c my-component --flat true	don't create new folder for this component
--inline-template (-t)	will the template be in .ts file?
--inline-style (-s)	will the style be in .ts file?
--spec	generate spec?
--prefix	assign own prefix
ng g d directive-name	create directive
ng g s service-name	create service
ng g cl models/customer	create customer class in models folder
ng g i models/person	create create interface in models folder
ng g e models/gender	create create ENUM gender in models folder
ng g p init-caps	create create pipe

5. Building&Serving

Command	Description
ng build	build app to /dist folder
ng build --aot	build app without code that we don't need (optimatization)
ng build --prod	build for production
ng serve -o	serve with opening a browser
ng serve --live-reload	reload when changes occur
ng serve -ssl	serving using SSL

6. Add new capabilities

Command	Description
ng add @angular/material	add angular material to project
ng g @angular/material:material-nav --name nav	create material navigation component

Components and Templates

Components are the most basic UI building block of an Angular app. An Angular app contains a tree of Angular components.

Sample component ts file

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

@Component({
        // component attributes
        selector: 'app-root',
        templateUrl: './app.component.html',
        styleUrls: ['./app.component.less']
})

export class AppComponent {
        title = 'my-dogs-training';
}
```

Component attributes

Attribute	Description
changeDetection	The change-detection strategy to use for this component.
viewProviders	Defines the set of injectable objects that are visible to its view DOM children
moduleId	The module ID of the module that contains the component
encapsulation	An encapsulation policy for the template and CSS styles
interpolation	Overrides the default encapsulation start and end delimiters ({{ and }}
entryComponents	A set of components that should be compiled along with this component.
preserveWhitespaces	True to preserve or false to remove potentially superfluous whitespace characters from the compiled template.

Component life cycles

Life cycle	Description
ngOnInit	Called once, after the first ngOnChanges()

Life cycle	Description
ngOnChanges	Called before ngOnInit() and whenever one of input properties change.
ngOnDestroy	Called just before Angular destroys the directive/component
ngDoCheck	Called during every change detection run
ngAfterContentChecked	Called after the ngAfterContentInit() and every subsequent ngDoCheck()
ngAfterViewChecked	Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().
ngAfterContentInit	Called once after the first ngDoCheck().
ngAfterViewInit	Called once after the first ngAfterContentChecked().

Template syntax

Syntax	Description
{{user.name}}	Interpolation - just generate user name here
<img [src] = "user.imageUrl">	property binding - bind image url for user to src attribute
<button (click)="do()" ... />	Event - assign function to click event
<button *ngIf="user.showSth" ... />	Show button when user.showSth is true
*ngFor="let item of items"	Iterate through items list
<div [ngClass]="{green: isTrue(), bold: itTrue()}"/>	Angular ngClass attribute
<div [ngStyle]="{'color': isTrue() ? '#bbb' : '#ccc'}"/>	Angular ngStyle attribute

Input and Output

Input() To pass value into child component

Sample child component implementation

```
export class SampleComponent {
        @Input() value: any/string/object/…;
        ...
}
```

Sample parent component usage

```
<app-sample-component [value]="myValue"></app-sampe-component>
```

Output() Emiting event to parent component

Sample child component

```
@Output() myEvent: EventEmitter<MyModel> = new EventEmitter();
onRemoved(item: MyModel) {
        this.myEvent.emit(item);
}
```

Sample parent component

```
<app-my-component
(myEvent)="someFunction()"></app-my-component>
```

onRemoved in child component is calling someFunction in parent component

Content projection

Content projection is injection inner html into child component

Example:

Parent component template

```
<component>
        <div>
                (some html here)
        </div>
</component>
```

Child component template

```
<ng-content></ng-content>
```

(some html here) will be injection into

Two differents htmls

```
<component>
        <div well-title>
                (some html here)
        </div>
        <div well-body>
                (some html here)
        </div>
</component>
```

```
<ng-content select="title"></ng-content>
<ng-content select="body"></ng-content>
```

ViewChild decorator

In order to have access to child component/directive/element

```
@ViewChild(NumberComponent)
private numberComponent: NumberComponent;
increase() {
this.numberComponent.increaseByOne(); //method from child component
}
decrease() {
this.numberComponent.decreaseByOne();  //method from child component
}
```

Sample for element: html:

```
<div #myElement></div>
```

component:

```
@ViewChild('myElement') myElement: ElementRef
```

Instead of ElementRef can be used specific element like FormControl for forms.

Reference to element in html:

```
<button (click)="doSth(myElement)"></button>
```

Routing

The Angular Router enables navigation from one view to the next as users perform application tasks.

Sample routing ts file

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

Then this should be added inside Angular.module imports

```
RouterModule.forRoot(appRoutes)
```

You can also turn on console tracking for your routing by adding enableTracing

```
imports: [
   RouterModule.forRoot(
     routes,
     {enableTracing: true}
    )
],
```

Usage

```
  <a routerLink="/crisis-center" routerLinkActive="active">Crisis Center</a>
```

routerLinkActive="active" will add active class to element when the link's route becomes active

```
//Navigate from code
this.router.navigate(['/heroes']);

// with parameters
this.router.navigate(['/heroes', { id: heroId, foo: 'foo' }]);

// Receive parameters without Observable
let id = this.route.snapshot.paramMap.get('id');
```

CanActivate and CanDeactivate

Interface that a class can implement to be a guard deciding if a route can be activated. If all guards return true, navigation will continue.

```
class AlwaysAuthGuard implements CanActivate {
        canActivate() {
                return true;
```

```
        }
}
```

and assing it in routing module:

```
 {
    path: 'artist/:artistId',
    component: ArtistComponent,
    canActivate: [AlwaysAuthGuard],
    children: [
      {path: '', redirectTo: 'tracks'},
      {path: 'tracks', component: ArtistTrackListComponent},
      {path: 'albums', component: ArtistAlbumListComponent},
    ]
  }
```

Modules

Angular apps are modular and Angular has its own modularity system called NgModules. NgModules are containers for a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities.

Sample module with comments

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
        declarations: [AppComponent], // components, pipes, directives
        imports: [BrowserModule, AppRoutingModule], // other modules
        providers: [], // services
        bootstrap: [AppComponent] // top component
})
export class AppModule { }
```

Services

Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on presenting data and delegate data access to a service.

Sample service with one function

```
@Injectable()
export class MyService {
        public items: Item[];
        constructor() { }

        getSth() {
                // some implementation
        }
}
```

Usage It should be injected before usage

```
constructor(private dogListService: MyService)
```

and add in module:

```
providers: [MyService]
```

HttpClient

To handle and consume http requests

1. Add import to module

```
import { HttpClientModule} from "@angular/common/http";
```

2. Usage

```
import {HttpClient} from '@angular/common/http';

...

// GET
public getData(): Observable<MyResponseModel> {
        return this.http.get<MyResponseModel>('api/users/2');
}

// POST
public send(val1: any, val2: any): Observable<any> {
        const object = new SendModel(val1, val2);
        const options = {headers: new HttpHeaders({'Content-type': 'application/json'})};
        return this.http.post<string>(environment.apiUrl + 'api/login', object, options);
}
```

Dependency Injection

Inject class into another class

```
@Injectable({
        providedIn: 'root',
})
export class SomeService {}
```

It accepts 'root' as a value or any module of your application

Declare global values

class:

```
import {InjectionToken} from '@angular/core';
export const CONTROLS_GLOBAL_CONFIG = new InjectionToken<ControlsConfig>('global-values');
export interface ControlsConfig {firstGlobalValue: string;}
```

module:

```
providers: [{provide: CONTROLS_GLOBAL_CONFIG, useValue: {firstGlobalValue : 'Some value' }},
```

usage (for example in component)

```
constructor(@Optional() @Inject(CONTROLS_GLOBAL_CONFIG) globalVlues: ControlsConfig) {
```

Pipes

Transform data/value to specific format, for example:

Show date in shortDate format:

```
{{model.birthsDay | date:'shortDate'}}
```

Pipe implementation

```
@Pipe({name: 'uselessPipe'})
export class uselessPipe implements PipeTransform {
  transform(value: string, before: string, after: string): string {
    let newStr = `${before} ${value} ${after}`;
    return newStr;
  }
}
```

usage

```
{{ user.name | uselessPipe:"Mr.":"the great" }}
```

Directives

An Attribute directive changes the appearance or behavior of a DOM element. For example [ngStyle] is a directive

Custom directive

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

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {

  constructor(private el: ElementRef) { }

  @Input('appHighlight') highlightColor: string;
  @Input('otherPar') otherPar: any;     //it will be taken from other attribute named [otherPar]

  @HostListener('mouseenter') onMouseEnter() {
    this.highlight(this.highlightColor || 'red');
  }

  private highlight(color: string) {
    this.el.nativeElement.style.backgroundColor = color;
  }
}
```

Usage

```
<p [appHighlight]="color" [otherPar]="someValue">Highlight me!</p>
```

Animations

Animations - moving from style state to another style state. Before add BrowserModule and BrowserAnimationsModule to module

Implementation:

```
  animations: [
      trigger('openClose', [
        state('open', style({
```

```
          height: '400px',
          opacity: 1.5,
        })),
        state('closed', style({
          height: '100px',
          opacity: 0.5,
        })),
        transition('open => closed', [
          animate('1s')
        ]),
        transition('closed => open', [
          animate('1s')
        ])
      ])
    ]
```

usage

```
<div [@openClose]="isShowed ? 'open' : 'closed'">
```

Angular Forms

Template driven forms

Form logic (validation, properties) are kept in template

sample html

```
<form name="form" (ngSubmit)="f.form.valid && onSubmit()" #f="ngForm" novalidate>
                    <div class="form-group">
                        <label for="firstName">First Name</label>
                        <input type="text" class="form-control" name="firstName" [(ngModel)]="model.firstName" #firstName="ngMo
                        <div *ngIf="f.submitted && firstName.invalid" class="invalid-feedback">
                            <div *ngIf="firstName.errors.required">First Name is required</div>
                        </div>
                    </div>
                    <div class="form-group">
                        <button class="btn btn-primary">Register</button>
                    </div>
                </form>
```

sample component

```
@ViewChild("f") form: any;
firstName: string = "";
langs: string[] = ["English", "French", "German"];

onSubmit() {
        if (this.form.valid) {
                console.log("Form Submitted!");
                this.form.reset();
        }
}
```

Reactive forms

Form logic (validation, properties) are kept in component

sample html

```
<form [formGroup]="registerForm" (ngSubmit)="onSubmit()">
        <div class="form-group">
                <label>Email</label>
                <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-invalid': submitted && f.email
                <div *ngIf="submitted && f.email.errors" class="invalid-feedback">
                        <div *ngIf="f.email.errors.required">Email is required</div>
                        <div *ngIf="f.email.errors.email">Email must be a valid email address</div>
                </div>
        </div>

        <div class="form-group">
                <button [disabled]="loading" class="btn btn-primary">Register</button>
        </div>
 </form>
```

sample component

```
registerForm: FormGroup;
submitted = false;

constructor(private formBuilder: FormBuilder) { }

ngOnInit() {
        this.registerForm = this.formBuilder.group({
                firstName: [{{here default value}}, Validators.required],
                lastName: ['', Validators.required],
                email: ['', [Validators.required, Validators.email]],
                password: ['', [Validators.required, Validators.minLength(6)]]
        });
}

// convenience getter for easy access to form fields
get f() { return this.registerForm.controls; }

 onSubmit() {
         this.submitted = true;

        // stop here if form is invalid
        if (this.registerForm.invalid) {
                return;
        }

        alert('SUCCESS!! :-)')
}
```

Custom Validator for Reactive forms

Function

```
validateUrl(control: AbstractControl) {
    if (!control.value || control.value.includes('.png') || control.value.includes('.jpg')) {
      return null;
    }
    return { validUrl: true };
  }
```

Usage

```
this.secondFormGroup = this._formBuilder.group({
      imageCtrl: ['', [Validators.required, this.validateUrl]]
});
```

Multi-field validation

```
validateNameShire(group: FormGroup) {
    if (group) {
      if (group.get('isShireCtrl').value && !group.get('nameCtrl').value.toString().toLowerCase().includes('shire')) {
        return { nameShire : true };
      }
    }
    return null;
 }
```

Multi-field validation usage*

```
this.firstFormGroup.setValidators(this.validateNameShire);
```

Error handling

```
<div *ngIf="firstFormGroup.controls.nameCtrl.errors.maxlength">Name is too long</div>
<div *ngIf="firstFormGroup.errors.nameShire">Shire dogs should have "shire" in name</div>
```

Custom Validator Directive for Template driven forms

To register our custom validation directive to NG_VALIDATORS service we have to do it like this: (thanks to multi parameter we won't override NG_VALIDATORS but just add CustomValidator to NG_VALIDATORS)

```
@Directive({
        selector: '[CustomValidator]',
        providers: [{provide: NG_VALIDATORS, useExisting: CustomValidator, multi:true}]
})
```

Example:

```
@Directive({
        selector: '[customValidation]',
        providers: [{provide: NG_VALIDATORS, useExisting:       EmailValidationDirective, multi: true}]
})
export class CustomValidation implements Validator {
        constructor() { }
        validate(control: AbstractControl): ValidationErrors {
        return (control.value && control.value.length <= 300) ?
                {myValue : true } : null;
        }
}
```

For multiple fields:

```
validate(formGroup: FormGroup): ValidationErrors {
        const passwordControl = formGroup.controls["password"];
        const emailControl = formGroup.controls["login"];
        if (!passwordControl || !emailControl || !passwordControl.value || !emailControl.value) {
                return null;
        }

        if (passwordControl.value.length > emailControl.value.length) {
                passwordControl.setErrors({ tooLong: true });
        } else {
                passwordControl.setErrors(null);
        }
        return formGroup;
}
```

ngModel in custom component

1. Add to module:

```
providers: [
        {
                provide: NG_VALUE_ACCESSOR,
                useExisting: forwardRef(() => TextAreaComponent),
                multi: true
        }
]
```

2. Implement ControlValueAccessor interface

```
interface ControlValueAccessor {
        writeValue(obj: any): void
        registerOnChange(fn: any): void
        registerOnTouched(fn: any): void
        setDisabledState(isDisabled: boolean)?: void
}
```

Function	Description
registerOnChange	Register a function to tell Angular when the value of the input changes
registerOnTouched	Register a function to tell Angular when the value was touched
writeValue	tell Angular how to Write a value to the input

Sample implementation:

```
@Component({
  selector: 'app-text-area',
  templateUrl: './text-area.component.html',
  styleUrls: ['./text-area.component.less'],
  providers: [
    {
      provide: NG_VALUE_ACCESSOR,
      useExisting: forwardRef(() => TextAreaComponent),
      multi: true
    }
  ]
})
export class TextAreaComponent implements ControlValueAccessor, OnInit {
  @Input() value: string;

  private _onChange = (data: any) => { console.log('changed: ' + data); };
  private _onTouched = (data?: any) => {console.log('touched: ' + data); };

  ngOnInit(): void {
    const self = this;
  }

  constructor() {}

  writeValue(obj: any): void {
    this.value = obj;
  }

  registerOnChange(fn) {
    this._onChange = fn;
  }

  registerOnTouched(fn: any): void {
    this._onTouched = fn;
```

```
  }
}
```

Tests

Unit tests

Service

```
describe('MyService', () => {
        let service: MyService;
        beforeEach(() => service = new MyService();
        it('#fetch should update data', () => {
                service.fetchData();
                expect(service.data.length).toBe(4);
                expect(service.data[0].id).toBe(1);
        });
});
```

For async functions

```
it('#fetch should update data', (done: DoneFn) => {
        // some code
        done();  // we need 'done' to avoid test finishing before date was received
        // some code
});
```

example async test:

```
it('http client works', (done: DoneFn) => {
    service.getUser().subscribe((data) => {
      expect(data).toBe('test');
      done();
    });
 });
```

Spy and stub

Spy:

```
// create spied object by copy getDataAsync from HttpService
const valueServiceSpy =
jasmine.createSpyObj('HttpService', ['getDataAsync']);
```

Stub:

```
const stubValue = of('StubValue');
valueServiceSpy.getDataAsync.and.returnValue(stubValue);
```

TestBed Mock whole module/environment for unit tests

```
beforeEach(() => {
        let httpClientMock = TestBed.configureTestingModule({ providers: [{ provide: MyService, useValue: new MyService(httpCli
});
```

Then use tested object (for example service) like this:

```
service = TestBed.get(MyService);
```

we can add schemas: [NO_ERRORS_SCHEMA]. This means that we don't have to mock children component dependencies of this component as Angular won't yell at us anymore for our lack of doing so.

Others

Http interceptor

Intercepts and handles an HttpRequest or HttpResponse.

Class:

```
@Injectable()
export class MyInterceptor implements HttpInterceptor {

        constructor() { }

        intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
                // do sth (like check and throw error)
                return next.handle(request); //if want continue
        }
}
```

Module:

```
{ provide: HTTP_INTERCEPTORS, useClass: TokenInterceptor, multi: true },
```

host

Refer to host element/component

Value	Description
:host(selector) { ... }	to match attributes, classes on the host element and add styling to it
:host-context(selector) { ... }	to match elements, classes on parent components and add styling to it
:host ::ng-deep	styling will be applied also to all child components

Interview questions

When would you use the useFactory provider method?

With useFactory we can use a factory at runtime to decide which kind of service we want to return if it got requested by any other class in our application or you need to parameterize the construction of a service

```
export function sampleFactory() {
  return new Service1();
}

export class Service1 {}

@Injectable({
  providedIn: 'root',
  useFactory: xyzFactory
})
export class Service2 {}
```

Service1 will be injected into another class

What is router-outlet

Acts as a placeholder that Angular dynamically fills based on the current router state. Generally in place of your main app component will be generated

How to declare global value?

Use InjectionToken

Which decorator lets you inject a service registered with an Injection Token?

@Inject

for example

```
  @Inject(CONTROLS_GLOBAL_CONFIG) globalVlues: ControlsConfig
```

How to mimick environment for components/services in tests?

Use TestBed. See Unit tests

What is Resolve interface?

Interface that classes can implement to be a data provider for component while routing.

example:

```
@Injectable()
class UserResolver implements Resolve<User> {
  constructor(private service: MySampleService) {}

  resolve(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<any> {
    return this.service.fetchData(route.params.id);
  }
}

@NgModule({
  imports: [
    RouterModule.forRoot([
      {
        path: 'user/:id',
        component: UserComponent,
        resolve: {
          userData: UserResolver
        }
      }
    ])
  ],
  providers: [UserResolver]
})
```

We can use it to pre-load data for a component before the component is displayed

How to begin validation after the user will enter a value and pause?

Use debounceTime, for example

```
this.formCtrlSub = this.firstNameControl.valueChanges
      .debounceTime(1000)
      .subscribe(newValue => this.firstName = newValue);
```

What is valueChanges in form control?

To catch value changes and implement some logic in observable result. See example above

How to execute canActivate if any of child routes will change?

Use canActivateChild

What are compilation types in Angular?

Type	Description
AoT	Ahead of time - compile full application / module when application was opened. Used mainly on production
JiT	Just in time - compile specific element when it has been opened. Used mainly while programming
Ivy	Since Angular 8 - engine based on concept Incremental DOM

What is ng-content in Angular?

See ng-content

How to create application with cutom prefix?

```
ng new app-name --prefix my-cutom-prefix
```

What is module lazy loading?

Instead of loading all modules while app starts we can load particular module when needed.

```
const routes: Routes = [
  {
    path: 'users',
    loadChildren: () => import('./users/users.module').then(m => m.UserModule)
  }
```

This is usually used for big apps in order to improve performance

Why should we consider using ngIf instead of ngClass/ngStyle for hiding element?

ngIf won't generate element when condition result is fale, so html will be lighter. ngClass/ngStyle will just hide element but it will be still existing in DOM

What is Done() function in tests?

We need 'done' to avoid test finishing before date was received See done

What "import", "providers" and "declarations" stand for in NgModule?

```
declarations: [AppComponent], // components, pipes, directives
imports: [BrowserModule, AppRoutingModule], // other modules
providers: [], // services
```

See Sample module

Explain the difference between Constructor and ngOnInit

Constructor is a method assigned to a class, so it is called when class object was initialized. ngOnInit is part of Component life cycle and it is dependent on the current state of view initialization. Constructor is called before ngOnInit

What is a difference between ElementRef and TemplateRef?

ElementRef is reference to particular element while TemplateRef can refer to whole ng-template

```
<ng-template #test>
   Test
</ng-template>
```

```
export class SthComponent  {
        @ViewChild('msg')
        private testTempRef : TemplateRef<any>
}
```

Point all data biding ways for element

Bindint type	Example
Property binding	[src]="this.src"
Event binding	(click)="this.doSth()"
Two way data bidning	[(ngModel)]="this.form.userName"

How to handle ngModel property in custom component?

Implement ControlValueAccessor interface. See ngModel

What is differenct between default and onPush change detector?

default change detector will check bidnings in whole application component tree after event OnPush change detector informs Angular that our component biding is depend on input parameters. Moreover it won't check bindings in whole application but only in subtree where our component belongs.

Why is it better to use pure pipes instead of some functions in template view?

If we want to calculate for example user age and show it on template then we can do it with function:

```
{{this.getUserAge(user.birthDate)}}
```

But it will be calculated every time when change detector is run, so it can affect on performance. Instead we can use pipe for that:

```
{{user.birthDate | calculateAge}}
```

Now age calculation won't be performed so many times.

How to detect change for any @Input property?