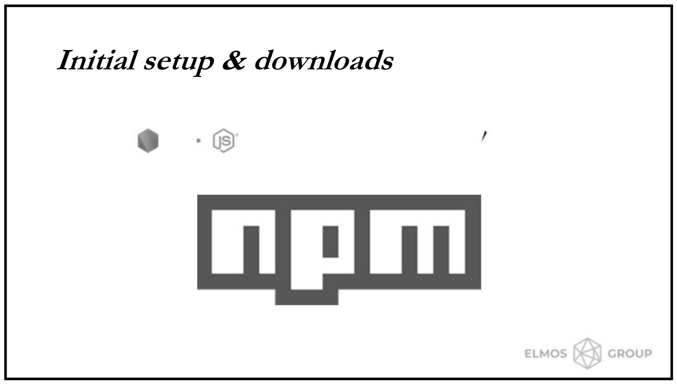
DELIVER WEB APPS
WITH CONFIDENCE

Masterclass September 2023

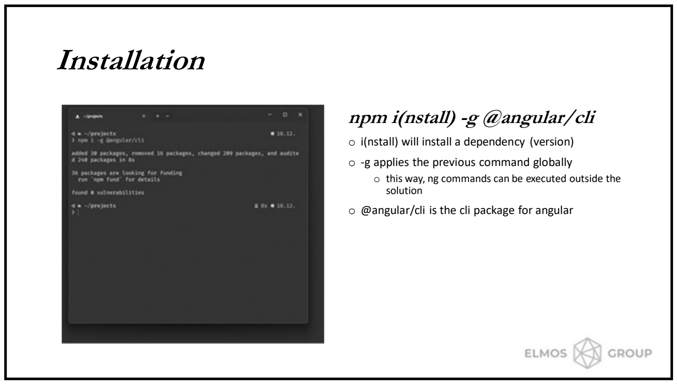
1



Initial setup & downloads

ELMOS GROUP

2



Installation

```
~/projects
~/projects                                                       18.12.
> npm i -g @angular/cli

added 30 packages, removed 16 packages, changed 209 packages, and audite
d 240 packages in 8s

36 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

~/projects                                                  8s   18.12.
>
```

npm i(nstall) -g @angular/cli

- i(nstall) will install a dependency (version)
- -g applies the previous command globally
  - this way, ng commands can be executed outside the solution
- @angular/cli is the cli package for angular

ELMOS GROUP

3

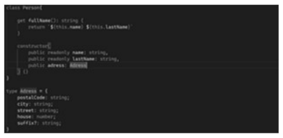
Typescript – Quick overview

```
[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]][([][(![]+[])[+[]]+(![]+[])[!
+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[!
+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+
!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![
]+[])[+!+[]]+(!![]+[])[+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])
[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]]((!![]+[])[+!+[]]+(!
![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+([][[]]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+!+[]]+(+[
![]]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+!+[]]]+(!![]
+[])[!+[]+!+[]+!+[]]+(+(!+[]+!+[]+!+[]+[+!+[]]))[(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+(![]+[]
)[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+([]+[])[([][(![]+[])[+[]]+(![]+[])[!+
[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[!+
[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!
+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]
+[])[+!+[]]+(!![]+[])[+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[
!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]][([][[]]+[])[+!+[]]+(
![]+[])[+!+[]]+((+[])[([][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]]+[])[
!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[+!
+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]
+[])[+[]]+([][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]]+[])[!+[]+!+[]+!
+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[
+!+[]+[+[]]]+(!![]+[])[+!+[]]]+[])[+!+[]+[+!+[]]]+(!![]+[])[!+[]+!+[]+!+[]]](!+[]+!+[]+!+[]
+[!+[]+!+[]])+(![]+[])[+!+[]]+(![]+[])[!+[]+!+[]])()((![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]
+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+([][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]
+[])[+!+[]]+(!![]+[])[+[]]]+[])[+!+[]+[!+[]+!+[]+!+[]]]+[+!+[]]+([+[]]+![]+[][(![]+[])[+[]]+(
![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]+(!![]+[])[+[]]])[!+[]+!+[]+[+[]]])
```

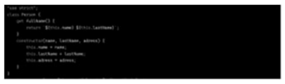
ELMOS GROUP

4

Typescript – Quick overview



```
class Person{

    get fullName(): string {
        return `${this.name} ${this.lastName}`
    }

    constructor(
        public readonly name: string,
        public readonly lastName: string,
        public adress: Adress
    ) {}
}

type Adress = {
    postalCode: string;
    city: string;
    street: string;
    house: number;
    suffix?: string;
}
```



```
"use strict";
class Person {
    get fullName() {
        return `${this.name} ${this.lastName}`;
    }
    constructor(name, lastName, adress) {
        this.name = name;
        this.lastName = lastName;
        this.adress = adress;
    }
}
```

Playground

- Compiles to Js
- Validates your code and shows potential errors
- Adds types
- And it allows your ide to give suggestions

[demo]

ELMOS GROUP

5

Typescript – Exercise

Easy:

Create a type for a authentication response.

It should have a value succeeded.

If that value is false, our type should have a array of messages. If the value is true, I want to read the token for the authenticated user

Challenging:

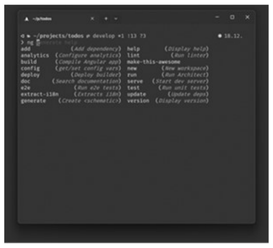
The type declared before can also handle messages not being strings. Make it generic.

Then I would like to create a function which reads the messages of this response. Provide a type which reads this.

ELMOS GROUP

6

Angular CLI



```
~/projects/todos ⎇ develop +1 !13 ?3                    ● 18.12.
> ng generate help
add            (Add dependency)  help          (Display help)
analytics (Configure analytics)  lint            (Run linter)
build     (Compile Angular app)  make-this-awesome
config    (get/set config vars)  new          (New workspace)
deploy          (Deploy builder)  run          (Run Architect)
doc     (Search documentation)  serve    (Start dev server)
e2e              (Run e2e tests)  test        (Run unit tests)
extract-i18n     (Extracts i18n)  update        (Update deps)
generate    (Create <schematic>)  version  (Display version)
```

[demo]

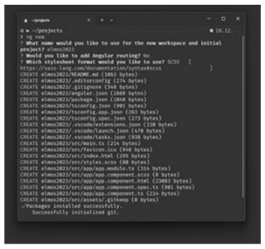
- ***ng new*** creates a new project
- ***ng generate*** can generate files
- ***ng serve*** will start your application

- Run ***ng completion*** for cli-autocomplete

ELMOS GROUP

7

Creating a new application



```
~/projects                                               ● 18.12.
> ng new
? What name would you like to use for the new workspace and initial
project? elmos2023
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? SCSS   [
https://sass-lang.com/documentation/syntax#scss                  ]
CREATE elmos2023/README.md (1063 bytes)
CREATE elmos2023/.editorconfig (274 bytes)
CREATE elmos2023/.gitignore (548 bytes)
CREATE elmos2023/angular.json (2889 bytes)
CREATE elmos2023/package.json (1040 bytes)
CREATE elmos2023/tsconfig.json (901 bytes)
CREATE elmos2023/tsconfig.app.json (263 bytes)
CREATE elmos2023/tsconfig.spec.json (273 bytes)
CREATE elmos2023/.vscode/extensions.json (130 bytes)
CREATE elmos2023/.vscode/launch.json (470 bytes)
CREATE elmos2023/.vscode/tasks.json (938 bytes)
CREATE elmos2023/src/main.ts (214 bytes)
CREATE elmos2023/src/favicon.ico (948 bytes)
CREATE elmos2023/src/index.html (295 bytes)
CREATE elmos2023/src/styles.scss (80 bytes)
CREATE elmos2023/src/app/app.module.ts (314 bytes)
CREATE elmos2023/src/app/app.component.scss (0 bytes)
CREATE elmos2023/src/app/app.component.html (23083 bytes)
CREATE elmos2023/src/app/app.component.spec.ts (901 bytes)
CREATE elmos2023/src/app/app.component.ts (214 bytes)
CREATE elmos2023/src/assets/.gitkeep (0 bytes)
√Packages installed successfully.
    Successfully initialized git.
```

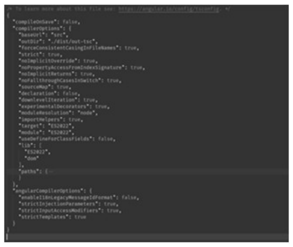
ng new [app-name]

--dry-run	do not save changes
--standalone	do not create modules
--routing	choice: routing enabled
--style	choice: styling language
--help	show help

ELMOS GROUP

8

Creating a new application – Setup (ts-config)



```
/* To learn more about this file see: https://angular.io/config/tsconfig. */
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "src",
    "outDir": "./dist/out-tsc",
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "noImplicitOverride": true,
    "noPropertyAccessFromIndexSignature": true,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": true,
    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "moduleResolution": "node",
    "importHelpers": true,
    "target": "ES2022",
    "module": "ES2022",
    "useDefineForClassFields": false,
    "lib": [
      "ES2022",
      "dom"
    ],
    "paths": {…
    }
  },
  "angularCompilerOptions": {
    "enableI18nLegacyMessageIdFormat": false,
    "strictInjectionParameters": true,
    "strictInputAccessModifiers": true,
    "strictTemplates": true
  }
}
```

Contains typescript configuration

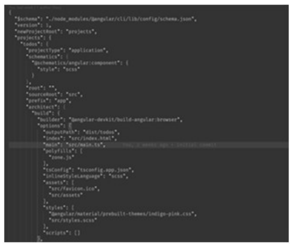
Important or usefull updates:

- ***'baseUrl': 'src'***
- ***'paths':***
  - ***'@[name]/*': '[path]/*'***

ELMOS GROUP

9

Creating a new application – Setup (angular)



```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "todos": {
      "projectType": "application",
      "schematics": {
        "@schematics/angular:component": {
          "style": "scss"
        }
      },
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/todos",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": [
              "zone.js"
            ],
            "tsConfig": "tsconfig.app.json",
            "inlineStyleLanguage": "scss",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "@angular/material/prebuilt-themes/indigo-pink.css",
              "src/styles.scss"
            ],
            "scripts": []
          },
```

Set up automatically by cli

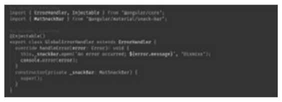
Keep out unless you know what you're doing

Will steer *ng generate*

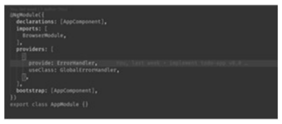
ELMOS GROUP

10

Creating a new application – Setup (errors)



```
import { ErrorHandler, Injectable } from '@angular/core';
import { MatSnackBar } from '@angular/material/snack-bar';

@Injectable()
export class GlobalErrorHandler extends ErrorHandler {
  override handleError(error: Error): void {
    this._snackBar.open(`An error occurred: ${error.message}`, 'Dismiss');
    console.error(error);
  }
  constructor(private _snackBar: MatSnackBar) {
    super();
  }
}
```



```
@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
  ],
  providers: [
    {
      provide: ErrorHandler,
      useClass: GlobalErrorHandler,
    },
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

- Global error handling can be provided with a class extending ErrorHandler.

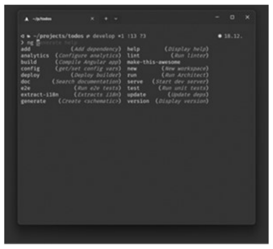
  [@angular/core]

- @Injectable allows it to be injected automatically
- Add to providers in appModule

ELMOS GROUP

11

Creating a new application – Dependencies



```
~/p/todos
~/projects/todos ⎇ develop +1 !13 ?3                              18.12.
> ng
add           (Add dependency)  help          (Display help)
analytics  (Configure analytics)  lint            (Run linter)
build      (Compile Angular app)  make-this-awesome
config      (get/set config vars)  new        (New workspace)
deploy           (Deploy builder)  run        (Run Architect)
doc       (Search documentation)  serve    (Start dev server)
e2e              (Run e2e tests)  test       (Run unit tests)
extract-i18n     (Extracts i18n)  update         (Update deps)
generate   (Create <schematic>)  version  (Display version)
```

Use ng add to add dependencies

Also works on packages not maintained by angular

Will update configuration when needed

ELMOS GROUP

12

Exercise: Create a new angular application

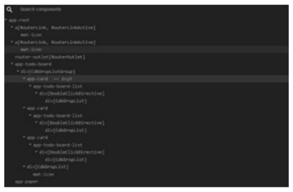
- Create a new angular application called todo.
  - Feel free to choose for a standalone application
  - Add error handling
  - Create a new component. Feel free to use tailwindcss

  - In case you're finished, set up your styling library.

ELMOS GROUP

13

Components & Modules



```
Search components
app-root
  a[RouterLink, RouterLinkActive]
    mat-icon
  a[RouterLink, RouterLinkActive]
    mat-icon
  router-outlet[RouterOutlet]
  app-todo-board
    div[CdkDropListGroup]
      app-card == $ng0
        app-todo-board-list
          div[DoubleClickDirective]
            div[CdkDropList]
      app-card
        app-todo-board-list
          div[DoubleClickDirective]
            div[CdkDropList]
      app-card
        app-todo-board-list
          div[DoubleClickDirective]
            div[CdkDropList]
      div[CdkDropList]
        mat-icon
  app-paper
```

Modules and components go hand in hand (this is changing with standalone components)

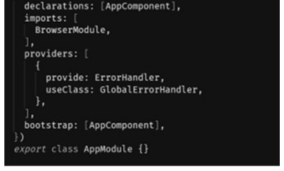
Components are snippets of code which can be rendered with their own html selector

Modules group components, add shared logic and can control accessability

ELMOS GROUP

14

Modules



```
You, 2 weeks ago | 1 author (You)
@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
  ],
  providers: [
    {
      provide: ErrorHandler,
      useClass: GlobalErrorHandler,
    },
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Groups components under declarations

Adds imports whichcan be used by all declared components

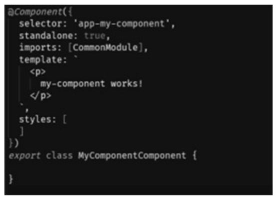
Enables dependency injection with *providers*

Can export components for use in other parts of the application

ELMOS GROUP

15

Components



```
@Component({
  selector: 'app-my-component',
  standalone: true,
  imports: [CommonModule],
  template: `
    <p>
      my-component works!
    </p>
  `,
  styles: [
  ]
})
export class MyComponentComponent {

}
```

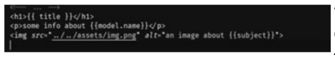
Components are nuggets of code with their own styling, templating and logic

- Can be *standalone*: Not part of a module
- *Template*: html code declaring templating
- *Styles*: Css code governing styles
- *Templateurl*: path to the template
- *StyleUrls*: paths to the css

ELMOS GROUP

16

Components – String interpolation



```
<h1>{{ title }}</h1>
<p>some info about {{model.name}}</p>
<img src="../../assets/img.png" alt="an image about {{subject}}">
```

It is possible to use variables in your component class in your template

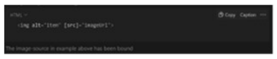
Wrap these variables in double curly braces

This can be done in both tekst, but also in template statements

ELMOS GROUP

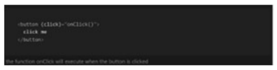
17

Components – Binding (1/4)



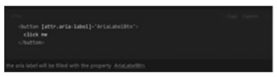
```
<img alt="item" [src]="imageUrl">
```

The image-source in example above has been bound



```
<button (click)="onClick()">
  click me
</button>
```

the function onClick will execute when the button is clicked



```
<button [attr.aria-label]="AriaLabelBtn">
  click me
</button>
```

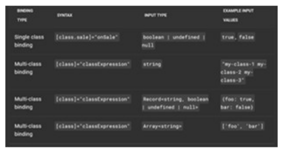
the aria label will be filled with the property AriaLabelBtn

- **Property-binding**: a typescript object is bound to the template
- **Event-binding**: a response to an event is bound to the template
- **Attribute-binding**: some template properties will not be suitable to data-binding, attribute binding can be used as a fallback

ELMOS GROUP

18

Components – Binding (2/4)



BINDING TYPE	SYNTAX	INPUT TYPE	EXAMPLE INPUT VALUES
Single class binding	`[class.sale]="onSale"`	`boolean | undefined | null`	`true`, `false`
Multi-class binding	`[class]="classExpression"`	`string`	`"my-class-1 my-class-2 my-class-3"`
Multi-class binding	`[class]="classExpression"`	`Record<string, boolean | undefined | null>`	`{foo: true, bar: false}`
Multi-class binding	`[class]="classExpression"`	`Array<string>`	`['foo', 'bar']`

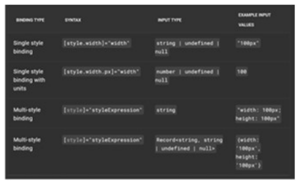
Class-binding: adding classses dynamically can be done as well:

- Single classes can be bound with ***[class.specificClass]="expression"*** with expression a boolean or undefined
- Multiple classes can be bound with ***[class]="expression"*** with expression described left

ELMOS GROUP

19

Components – Binding (3/4)



BINDING TYPE	SYNTAX	INPUT TYPE	EXAMPLE INPUT VALUES
Single style binding	`[style.width]="width"`	`string | undefined | null`	`"100px"`
Single style binding with units	`[style.width.px]="width"`	`number | undefined | null`	`100`
Multi-style binding	`[style]="styleExpression"`	`string`	`"width: 100px; height: 100px"`
Multi-style binding	`[style]="styleExpression"`	`Record<string, string | undefined | null>`	`{width: '100px', height: '100px'}`

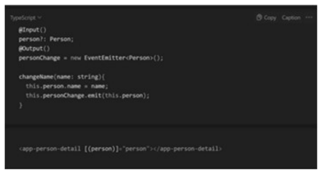
Style-binding:

- Single styles can be bound with ***[style.specificStyle]="expression"*** with expression a boolean or undefined or ***[style.specificStyle.unit]="expression"*** to provide a unit
- Multiple classes can be bound with ***[class]="expression"*** with expression described left

ELMOS GROUP

20

Components – Binding (4/4)



```
@Input()
person?: Person;
@Output()
personChange = new EventEmitter<Person>();

changeName(name: string){
  this.person.name = name;
  this.personChange.emit(this.person);
}
```

```
<app-person-detail [(person)]="person"></app-person-detail>
```

2-way binding:

- It is possible to pass data along components using property and event-binding, as well as combining these
- Banana-box
(not box in a banana)
- Equivalent to
<smt
[prop]="prop"
(propChange)="prop = $event"
/>

ELMOS GROUP

21

Exercise: Create a new todo component

- Create a new todo component. It should show a todo.
- provide some nice styling
- Pass the todo using component binding

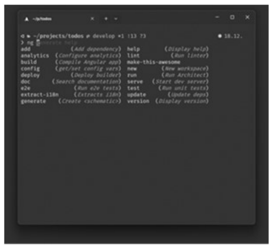
Advanced:

- Create a button to create/update the todo at runtime.

ELMOS GROUP

22

Components – Host ?



```
~/p/todos
⊲ ~/projects/todos ⌥ develop +1 !13 ?3                         ● 10.12.
❯ ng generate help
add              (Add dependency)  help          (Display help)
analytics   (Configure analytics)  lint             (Run linter)
build       (Compile Angular app)  make-this-awesome
config      (get/set config vars)  new        (New workspace)
deploy           (Deploy builder)  run        (Run Architect)
doc      (Search documentation)    serve   (Start dev server)
e2e              (Run e2e tests)   test      (Run unit tests)
extract-i18n     (Extracts i18n)   update       (Update deps)
generate    (Create <schematic>)   version  (Display version)
```

hostBinding

You can bind styles and classes to the component using HostBinding

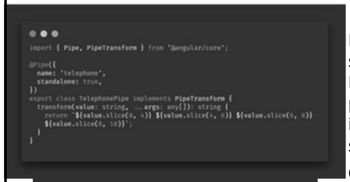
hostListener

Listen to events on the host by using hostListener

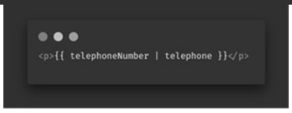
ELMOS GROUP

23

Pipes



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

@Pipe({
  name: 'telephone',
  standalone: true,
})
export class TelephonePipe implements PipeTransform {
  transform(value: string, ...args: any[]): string {
    return `${value.slice(0, 4)} ${value.slice(4, 6)} ${value.slice(6, 8)}
    ${value.slice(8, 10)}`;
  }
}
```



```
<p>{{ telephoneNumber | telephone }}</p>
```

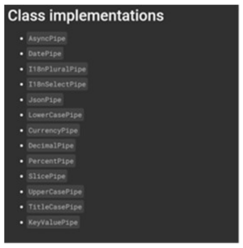
Pipes can be applied to data in string interpolations to format it. It also allows modifications, but prefers these to be pure: the same input should always result in the same output,

Common use-cases are decimal rounding, currencies or translations…

ELMOS GROUP

24

Pipes – Existing pipes



Class implementations

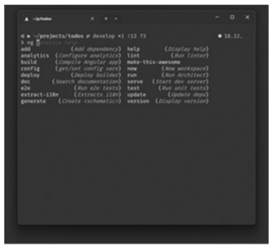
- `AsyncPipe`
- `DatePipe`
- `I18nPluralPipe`
- `I18nSelectPipe`
- `JsonPipe`
- `LowerCasePipe`
- `CurrencyPipe`
- `DecimalPipe`
- `PercentPipe`
- `SlicePipe`
- `UpperCasePipe`
- `TitleCasePipe`
- `KeyValuePipe`

Existing pipes allow you to fetch async observable values, parse values such as dates and much more

ELMOS GROUP

25

Components – Templates



```
◁ ➤ ~/projects/todos ⎇ develop +1 !13 ?3                    ● 10.12.
> ng generate help
add            (Add dependency)  help        (Display help)
analytics  (Configure analytics)  lint          (Run linter)
build      (Compile Angular app)  make-this-awesome
config     (get/set config vars)  new       (New workspace)
deploy          (Deploy builder)  run       (Run Architect)
doc       (Search documentation)  serve  (Start dev server)
e2e              (Run e2e tests)  test     (Run unit tests)
extract-i18n     (Extracts i18n)  update      (Update deps)
generate    (Create <schematic>)  version (Display version)
```

Angular commonModule allows us to manipulate the dom using syntax in the html template:

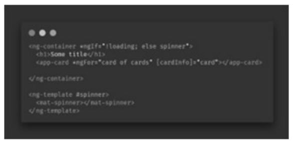
Loop over a collection with ***ngFor***

Conditionally show content with ***ngIf***

ELMOS GROUP

26

Components – Advanced templates with ngIf and NgFor



```
<ng-container *ngIf="!loading; else spinner">
  <h1>Some title</h1>
  <app-card *ngFor="card of cards" [cardInfo]="card"></app-card>

</ng-container>

<ng-template #spinner>
  <mat-spinner></mat-spinner>
</ng-template>
```

ngIf: conditionally show this element if the statement is truthy

NgFor: Loop this template for each element in this collection

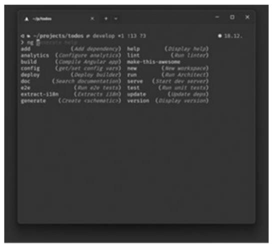
Ng-container: use this element to add extra directives, does not create dom-elements

Ng-template: only render this element when needed

ELMOS GROUP

27

Components – Templates & directives



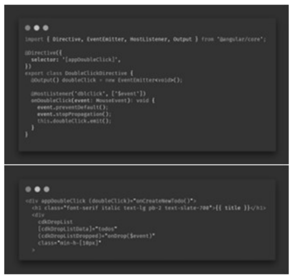
```
~/p/todos
~/projects/todos ⎇ develop +1 !13 ?3                         18.12.
❯ ng generate help
add            (Add dependency)  help         (Display help)
analytics  (Configure analytics)  lint           (Run linter)
build      (Compile Angular app)  make-this-awesome
config     (get/set config vars)  new        (New workspace)
deploy           (Deploy builder)  run        (Run Architect)
doc      (Search documentation)  serve   (Start dev server)
e2e               (Run e2e tests)  test      (Run unit tests)
extract-i18n      (Extracts i18n)  update        (Update deps)
generate   (Create <schematic>)  version  (Display version)
```

- The things we just described are called directives.
- We can also create our own directives
- There's 2 types of directives:
  - Attribute directives
  - Structural directives

ELMOS GROUP

28

Directives – attribute directives



```
import { Directive, EventEmitter, HostListener, Output } from '@angular/core';

@Directive({
  selector: '[appDoubleClick]',
})
export class DoubleClickDirective {
  @Output() doubleClick = new EventEmitter<void>();

  @HostListener('dblclick', ['$event'])
  onDoubleClick(event: MouseEvent): void {
    event.preventDefault();
    event.stopPropagation();
    this.doubleClick.emit();
  }
}
```

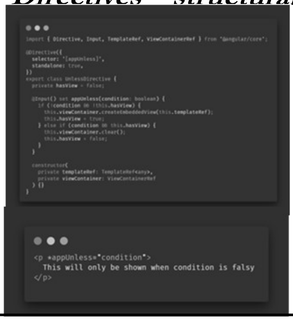
```
<div appDoubleClick (doubleClick)="onCreateNewTodo()">
  <h1 class="font-serif italic text-lg pb-2 text-slate-700">{{ title }}</h1>
  <div
    cdkDropList
    [cdkDropListData]="todos"
    (cdkDropListDropped)="onDrop($event)"
    class="min-h-[10px]"
  >
```

Attribute directives allow you to add behaviour to components without having to create components for these.

ELMOS GROUP

29

Directives – structural directives



```
import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';

@Directive({
  selector: '[appUnless]',
  standalone: true,
})
export class UnlessDirective {
  private hasView = false;

  @Input() set appUnless(condition: boolean) {
    if (!condition && !this.hasView) {
      this.viewContainer.createEmbeddedView(this.templateRef);
      this.hasView = true;
    } else if (condition && this.hasView) {
      this.viewContainer.clear();
      this.hasView = false;
    }
  }

  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainer: ViewContainerRef
  ) {}
}
```

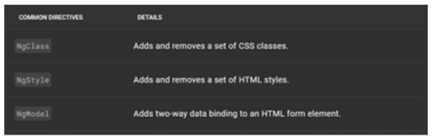
```
<p *appUnless="condition">
  This will only be shown when condition is falsy
</p>
```

Structural directives allow you to manipulate views and templates.

ELMOS GROUP

30

Directives – Existing directives - Atttribute

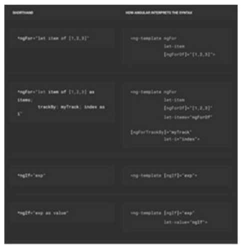


COMMON DIRECTIVES	DETAILS
NgClass	Adds and removes a set of CSS classes.
NgStyle	Adds and removes a set of HTML styles.
NgModel	Adds two-way data binding to an HTML form element.

ELMOS GROUP

31

Directives – Existing directives - Structural

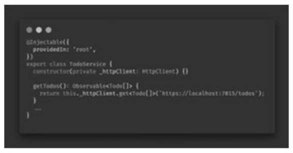


SHORTHAND	HOW ANGULAR INTERPRETS THE SYNTAX
`*ngFor="let item of [1,2,3]"`	`<ng-template ngFor let-item [ngForOf]="[1,2,3]">`
`*ngFor="let item of [1,2,3] as items; trackBy: myTrack; index as i"`	`<ng-template ngFor let-item [ngForOf]="[1,2,3]" let-items="ngForOf" [ngForTrackBy]="myTrack" let-i="index">`
`*ngIf="exp"`	`<ng-template [ngIf]="exp">`
`*ngIf="exp as value"`	`<ng-template [ngIf]="exp" let-value="ngIf">`

ELMOS GROUP

32

Services



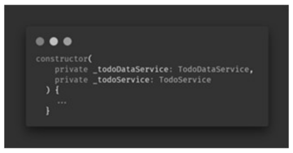
```
@Injectable({
  providedIn: 'root',
})
export class TodoService {
  constructor(private _httpClient: HttpClient) {}

  getTodos(): Observable<Todo[]> {
    return this._httpClient.get<Todo[]>('https://localhost:7015/todos');
  }
  ...
}
```

Service is a broad category encompassing any value, function, or feature that an application needs. A service is typically a class with a narrow, well-defined purpose.

ELMOS GROUP

33

Dependency Injection



```
constructor(
    private _todoDataService: TodoDataService,
    private _todoService: TodoService
  ) {
    ...
  }
```

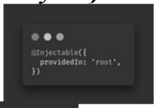
To inject a service as a dependency, you can use component's constructor

Angular recognizes dependencies, if they are annotated with the @Injectable decorator.

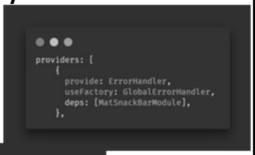
ELMOS GROUP

34

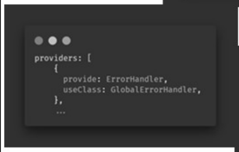
Dependency Injection – Deep dive



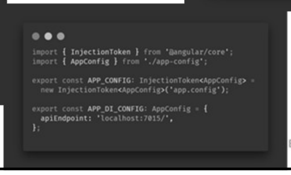
```
@Injectable({
  providedIn: 'root',
})
```



```
providers: [
    {
      provide: ErrorHandler,
      useFactory: GlobalErrorHandler,
      deps: [MatSnackBarModule],
    },
```



```
providers: [
    {
      provide: ErrorHandler,
      useClass: GlobalErrorHandler,
    },
    ...
```



```
import { InjectionToken } from '@angular/core';
import { AppConfig } from './app-config';

export const APP_CONFIG: InjectionToken<AppConfig> =
  new InjectionToken<AppConfig>('app.config');

export const APP_DI_CONFIG: AppConfig = {
  apiEndpoint: 'localhost:7015/',
};
```

ELMOS GROUP

35

Reactive programming

"Reactive programming is an asynchronous programming paradigm concerned with data streams and the propagation of change"

-wikipedia

ELMOS GROUP

36

Exercise: Reactive programming

Easy

Use rxjs `of` to create a stream of 5 consecutive numbers (0-5).

For every number, log it to console.

For every even number, merge the subscription with a new stream of 2 characters (a, b).

Log the final result to the console.

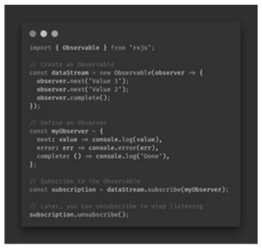
Challenging

Create a timer with a button pauze/resume, and a button restart.

ELMOS GROUP

37

Reactive programming – Subscriptions



```
import { Observable } from 'rxjs';

// Create an Observable
const dataStream = new Observable(observer => {
  observer.next('Value 1');
  observer.next('Value 2');
  observer.complete();
});

// Define an Observer
const myObserver = {
  next: value => console.log(value),
  error: err => console.error(err),
  complete: () => console.log('Done'),
};

// Subscribe to the Observable
const subscription = dataStream.subscribe(myObserver);

// Later, you can unsubscribe to stop listening
subscription.unsubscribe();
```

- **Observable**: Represents asynchronous data or event streams.
- **Observer**: Defines how to react to emitted values (next), errors (error), and completion (complete).
- **Subscription**: Connects Observables and Observers.

ELMOS GROUP

38

Reactive programming – Subjects & streams

- **Subject**: Special type of Observable.
- Acts as both producer and consumer.
- Facilitates multicasting and state sharing.

Special types of subjects:

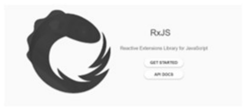
- **BehaviorSubject**: Holds and shares the most recent value.
- **ReplaySubject**: Records and replays a specific number of values.
- **AsyncSubject**: Emits only the last value upon completion.

ELMOS GROUP

39

Reactive programming – Pipe & operators

1.Transform Data: You can use operators like map, pluck, filter, and scan to transform the data emitted by an observable. For example, you can map each item to a different value, filter out unwanted data, or accumulate values over time.
2.Combine Observables: Operators like merge, concat, combineLatest, and zip enable you to combine multiple observables into a single observable. This is useful for scenarios where you need to work with data from multiple sources.
3.Throttle and Debounce: Use operators like throttleTime and debounceTime to control the rate at which data is emitted by the observable. This can help in scenarios like implementing search suggestions or preventing excessive API requests.
4.Error Handling: Operators like catchError, retry, and throwIfEmpty allow you to handle errors gracefully within the observable stream.
5.Conditional Operations: Operators like take, takeUntil, skip, and takeWhile help you control how many values are emitted or when the observable should complete based on certain conditions.
6.Delay and Timer Operations: You can introduce delays between emissions using delay or create observables that emit values at specific intervals using interval, timer, or delayWhen.
7.Buffering: Operators like buffer, bufferTime, and window allow you to group emitted values into buffers or windows, which can be useful for batching data.
8.Custom Operators: You can create your own custom operators using the pipe method, which makes it easy to reuse and share complex data transformation logic.
9.Combining Streams: With operators like switchMap, mergeMap, and concatMap, you can switch or merge observables based on the values emitted by the source observable. This is often used for handling asynchronous requests and responses.
10.Reducing and Aggregating Data: Operators like reduce, scan, and toArray help you reduce a stream of values into a single value or an array of values.
11.Conditional Logic: You can introduce conditional logic within your observable pipeline using operators like switchMap, filter,

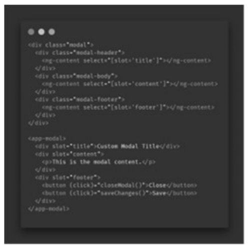


Learn RxJS

ELMOS GROUP

40

Templates and content-projection



```
<div class="modal">
  <div class="modal-header">
    <ng-content select="[slot='title']"></ng-content>
  </div>
  <div class="modal-body">
    <ng-content select="[slot='content']"></ng-content>
  </div>
  <div class="modal-footer">
    <ng-content select="[slot='footer']"></ng-content>
  </div>
</div>

<app-modal>
  <div slot="title">Custom Modal Title</div>
  <div slot="content">
    <p>This is the modal content.</p>
  </div>
  <div slot="footer">
    <button (click)="closeModal()">Close</button>
    <button (click)="saveChanges()">Save</button>
  </div>
</app-modal>
```

ng-content: Enabling Content Projection

- Use *<ng-content></ng-content>* within the template of the receiving component.
- It acts as a placeholder for content from the parent component.

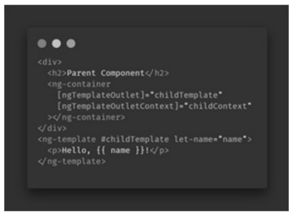
Named Slots: Customizing Content Projection

- Multiple *<ng-content>* elements in the component's template.
- Each with a different name or selector.
- Parent component specifies content placement using select attributes.

ELMOS GROUP

41

Templates



```
<div>
  <h2>Parent Component</h2>
  <ng-container
    [ngTemplateOutlet]="childTemplate"
    [ngTemplateOutletContext]="childContext"
  ></ng-container>
</div>
<ng-template #childTemplate let-name="name">
  <p>Hello, {{ name }}!</p>
</ng-template>
```

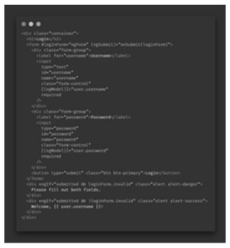
Usage:

- Display using *ngTemplateOutlet.
- Conditionally render with structural directives.
- Pass as context to components
- We can provide scoped context

ELMOS GROUP

42

Angular in depth: Template-Driven Forms



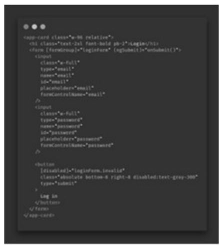
```
<div class="container">
  <h2>Login</h2>
  <form #loginForm="ngForm" (ngSubmit)="onSubmit(loginForm)">
    <div class="form-group">
      <label for="username">Username</label>
      <input
        type="text"
        id="username"
        name="username"
        class="form-control"
        [(ngModel)]="user.username"
        required
      />
    </div>
    <div class="form-group">
      <label for="password">Password</label>
      <input
        type="password"
        id="password"
        name="password"
        class="form-control"
        [(ngModel)]="user.password"
        required
      />
    </div>
    <button type="submit" class="btn btn-primary">Login</button>
  </form>
  <div *ngIf="submitted && loginForm.invalid" class="alert alert-danger">
    Please fill out both fields.
  </div>
  <div *ngIf="submitted && !loginForm.invalid" class="alert alert-success">
    Welcome, {{ user.username }}!
  </div>
</div>
```

- HTML-Centric: In template-driven forms, the form structure is primarily defined in the HTML template.
- Two-Way Data Binding: Template-driven forms often use two-way data binding with *[(ngModel)]*
- Validation: Angular's built-in directives
- Async Validation: both synchronous and asynchronous form validation.
- Simple Use Cases: simple forms with basic validation requirements

ELMOS GROUP

43

Angular in depth: Reactive forms



```
<app-card class="w-96 relative">
  <h1 class="text-2xl font-bold pb-2">Login</h1>
  <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
    <input
      class="w-full"
      type="email"
      name="email"
      id="email"
      placeholder="email"
      formControlName="email"
    />
    <input
      class="w-full"
      type="password"
      name="password"
      id="password"
      placeholder="password"
      formControlName="password"
    />

    <button
      [disabled]="loginForm.invalid"
      class="absolute bottom-8 right-8 disabled:text-gray-300"
      type="submit"
    >
      Log in
    </button>
  </form>
</app-card>
```

- **Reactive**: Code-centric approach.
- **Programmatic**: Form controls defined in the component.
- **Immutable Data Model**: FormGroup, FormControl, FormArray.
- **Advanced Control**: Precise validation, dynamic forms.
- **Observable-Powered**: Real-time updates with RxJS.
- **Complex Use Cases**: Ideal for complex forms.
- **Fine-Grained Control**: Dynamic behavior and validation.
- **Robust**: Suitable for large-scale applications.

ELMOS GROUP

44