ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Typescript και Angular

Εισαγωγή στο Angular Framework

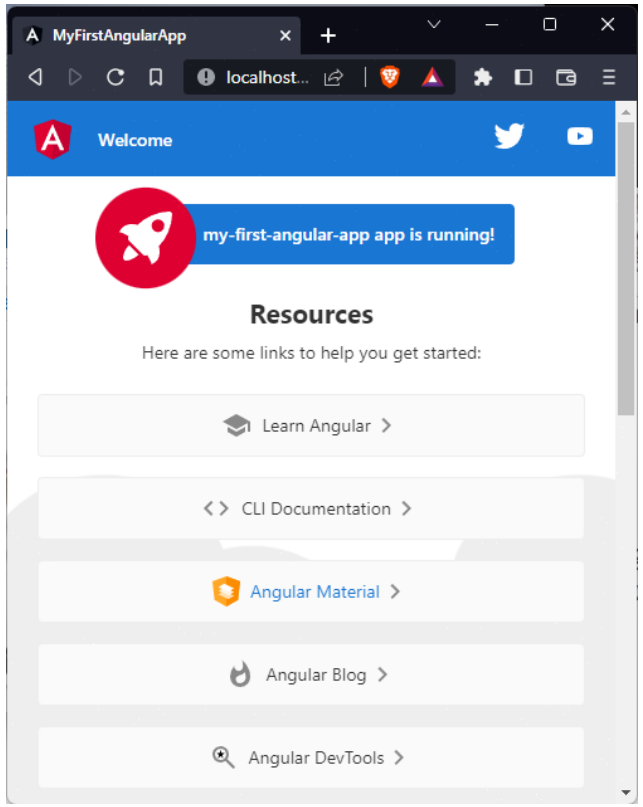
Χριστόδουλος Φραγκουδάκης

Το Angular Command Line

Το *Angular command line (Angular CLI)* είναι μια διεπαφή με το Angular Framework διαμέσου της γραμμής εντολών. Η εγκατάσταση και χρήση του είναι άμεση:

```
$ npm install -g @angular/cli
$ cd Workspace/
$ ng new my-first-angular-app
$ cd my-first-angular-app/
$ ng serve
```

- Δημιουργεί ένα σκελετό εφαρμογής που ήδη λειτουργεί και ακολουθεί όλες τις καλές προγραμματιστές πρακτικές
- Αυτοματοποιεί επαναμβαλόμενες ενέργειες γράφοντας αντί για εμάς τον κώδικα



Εγκατάσταση του Angular CLI

Εγκαθίσταται μέσω του `npm` με το διακόπτη `-g` και είναι διαθέσιμο μέσω της εντολής `ng`

```
$ npm install -g @angular/cli

$ ng version


     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 15.2.0
Node: 18.13.0
Package Manager: npm 8.19.3
OS: win32 x64

Angular: 15.2.0
... animations, cli, common, compiler, compiler-cli, core, forms
... platform-browser, platform-browser-dynamic, router

Package                         Version
---------------------------------------------------------
@angular-devkit/architect       0.1502.0
@angular-devkit/build-angular   15.2.0
@angular-devkit/core            15.2.0
@angular-devkit/schematics      15.2.0
@schematics/angular             15.2.0
rxjs                            7.8.0
typescript                      4.9.5
```

Βασικοί διακόπτες της εντολής ng

```
$ ng help
ng <command>

Commands:
  ng add <collection>            Adds support for an external library to your project.
  ng analytics                   Configures the gathering of Angular CLI usage metrics.
  ng build [project]             Compiles an Angular application or library into an output directory named dist/ at the given output path.     [aliases: b]
  ng cache                       Configure persistent disk cache and retrieve cache statistics.
  ng completion                  Set up Angular CLI autocompletion for your terminal.
  ng config [json-path] [value]  Retrieves or sets Angular configuration values in the angular.json file for the workspace.
  ng deploy [project]            Invokes the deploy builder for a specified project or for the default project in the workspace.
  ng doc <keyword>               Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword.         [aliases: d]
  ng e2e [project]               Builds and serves an Angular application, then runs end-to-end tests.                                         [aliases: e]
  ng extract-i18n [project]      Extracts i18n messages from source code.
  ng generate                    Generates and/or modifies files based on a schematic.                                                         [aliases: g]
  ng lint [project]              Runs linting tools on Angular application code in a given project folder.
  ng new [name]                  Creates a new Angular workspace.                                                                              [aliases: n]
  ng run <target>                Runs an Architect target with an optional custom builder configuration defined in your project.
  ng serve [project]             Builds and serves your application, rebuilding on file changes.                                               [aliases: s]
  ng test [project]              Runs unit tests in a project.                                                                                 [aliases: t]
  ng update [packages..]         Updates your workspace and its dependencies. See https://update.angular.io/.
  ng version                     Outputs Angular CLI version.                                                                                  [aliases: v]
Options:
  --help  Shows a help message for this command in the console.                                                                                   [boolean]
For more information, see https://angular.io/cli/.
```

Η πρώτη εφαρμογή

Η δημιουργία μιας εφαρμογής γίνεται με την εντολή `ng new <project-name>` και παράγεται στη γραμμή εντολών έξοδος όπως παρακάτω

```
$ ng new my-first-angular-app
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE my-first-angular-app/angular.json (2770 bytes)
CREATE my-first-angular-app/package.json (1051 bytes)
CREATE my-first-angular-app/README.md (1071 bytes)
CREATE my-first-angular-app/tsconfig.json (901 bytes)
CREATE my-first-angular-app/.editorconfig (274 bytes)
CREATE my-first-angular-app/.gitignore (548 bytes)
CREATE my-first-angular-app/tsconfig.app.json (263 bytes)
CREATE my-first-angular-app/tsconfig.spec.json (273 bytes)
CREATE my-first-angular-app/.vscode/extensions.json (130 bytes)
CREATE my-first-angular-app/.vscode/launch.json (474 bytes)
CREATE my-first-angular-app/.vscode/tasks.json (938 bytes)
CREATE my-first-angular-app/src/favicon.ico (948 bytes)
CREATE my-first-angular-app/src/index.html (303 bytes)
CREATE my-first-angular-app/src/main.ts (214 bytes)
CREATE my-first-angular-app/src/styles.css (80 bytes)
CREATE my-first-angular-app/src/assets/.gitkeep (0 bytes)
CREATE my-first-angular-app/src/app/app.module.ts (314 bytes)
CREATE my-first-angular-app/src/app/app.component.html (23083 bytes)
CREATE my-first-angular-app/src/app/app.component.spec.ts (998 bytes)
CREATE my-first-angular-app/src/app/app.component.ts (224 bytes)
CREATE my-first-angular-app/src/app/app.component.css (0 bytes)
✓ Packages installed successfully.
    Successfully initialized git.
```

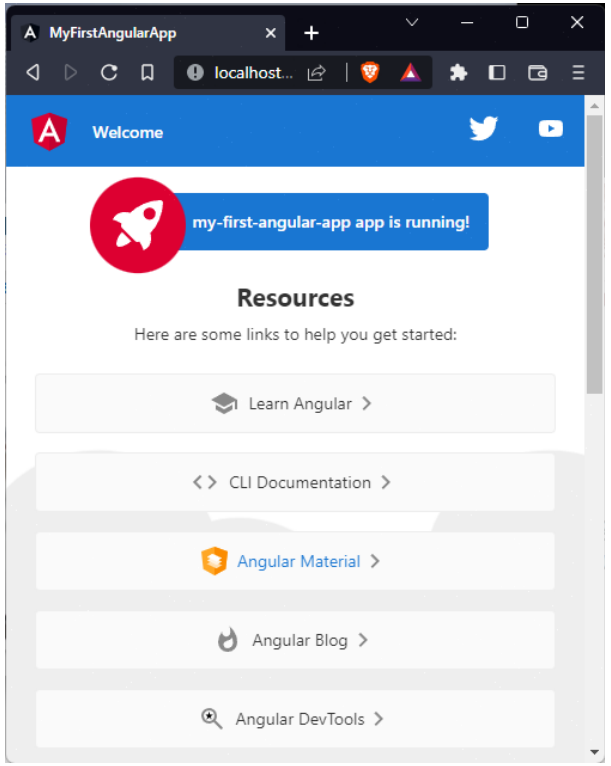
Στην ενότητα `scripts` του αρχείου `package.json` έχουν δημιουργηθεί `npm run` εντολές για την ανάπτυξη και τον έλεγχο της εφαρμογής μας

```
{
  "name": "my-first-angular-app",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
  ...
}
```

Εκτέλεση της εφαρμογής

Μετακινούμαστε στον κατάλογο `my-first-angular-app` και ξεκινάμε τον development server. Η εντολή `ng serve` *μετατρέπει την εφαρμογή σε Javascript* και ξεκινάει ένα web server που εξυπηρετεί την εφαρμογή μας στη διεύθυνση `http://localhost:4200`

```
$ ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   1.71 MB |
polyfills.js          | polyfills     | 314.29 kB |
styles.css, styles.js | styles        | 209.42 kB |
main.js               | main          |  46.02 kB |
runtime.js            | runtime       |   6.54 kB |

                      | Initial Total |   2.28 MB

Build at: 2023-02-28T15:34:29.086Z - Hash: b1217420ffa488bb - Time: 5071ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
```



Ο web server βρίσκεται σε συγχρονισμό με τον κώδικά μας (Typescript) και ακολουθεί τις αλλαγές όσο αυτός μετατρέπεται σε Javascript

Κατάλογος εφαρμογής και βασικό αρχείο

- Στον κατάλογο της εφαρμογής μας δημιουργήθηκαν πολλά αρχεία και φάκελλοι
- Τα αρχεία της εφαρμογής (Typescript) βρίσκονται στον κατάλογο `src/app`
- Το κεντρικό αρχείο είναι το `src/index.html`
- Το κεντρικό stylesheet είναι το `src/styles.css`
- Βασικός κατάλογος για στατικά αρχεία (εικόνες κτλ) είναι ο `src/assets`

```
.
├── angular.json
├── karma.conf.js
├── node_modules
│   ├── ...
│  ...
├── package.json
├── package-lock.json
├── README.md
├── src
│   ├── app
│   │   ├── ...
│   │  ...
│   ├── assets
│   ├── environments
│   ├── favicon.ico
│   ├── index.html      # Το βασικό αρχείο της εφαρμογής μας
│   ├── main.ts
│   ├── polyfills.ts
│   ├── styles.css
│   └── test.ts
├── tsconfig.app.json
├── tsconfig.json
└── tsconfig.spec.json
```

Βασικό αρχείο `index.html` vs `ng serve`

Η εντολή `ng serve` *μετατρέπει* (transpile) όλα τα αρχεία του καταλόγου `src/app` σε αρχεία Javascript που ο web development server τα κάνει διαθέσιμα στο βασικό αρχείο `index.html`

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>MyFirstAngularApp</title>
    <base href="/" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" type="image/x-icon" href="favicon.ico" />
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <app-root></app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```

```
$ ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   1.71 MB |
polyfills.js          | polyfills     | 314.29 kB |
styles.css, styles.js | styles        | 209.42 kB |
main.js               | main          |  46.02 kB |
runtime.js            | runtime       |   6.54 kB |

                      | Initial Total |   2.28 MB

Build at: 2023-02-28T15:34:29.086Z - Hash: b1217420ffa488bb - Time: 5071ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
```

Κάθε αλλαγή στον κατάλογο `src/app`, με ενεργό `ng serve`, προκαλεί αυτόματα την επαναδημιουργία των αρχείων Javascript και την ανανέωση της εφαρμογής στον browser

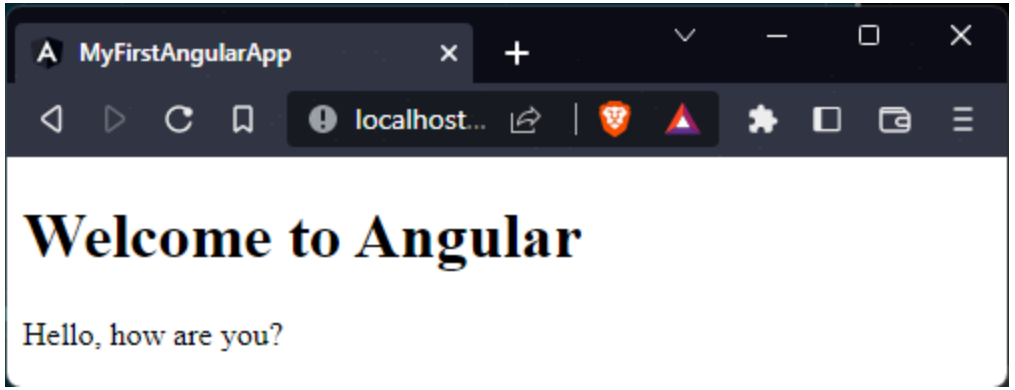
Ο κατάλογος `src/app`

Το Angular Framework ακολουθεί το μοντέλο MVC όπου το αρχείο `app.component.ts` περιέχει εκτός της λογικής του controller και τη λογική του view model. Το view model είναι αυτό που παρέχει δεδομένα στο view (`app.component.html`) και ανταποκρίνεται στην αλληλεπίδραση με το χρήστη

```
...
├── src
│   ├── app
│   │   ├── app.component.css
│   │   ├── app.component.html
│   │   ├── app.component.spec.ts
│   │   ├── app.component.ts
│   │   └── app.module.ts
...
```

Αλλάζουμε το αρχείο `src/app/app.component.html` σε

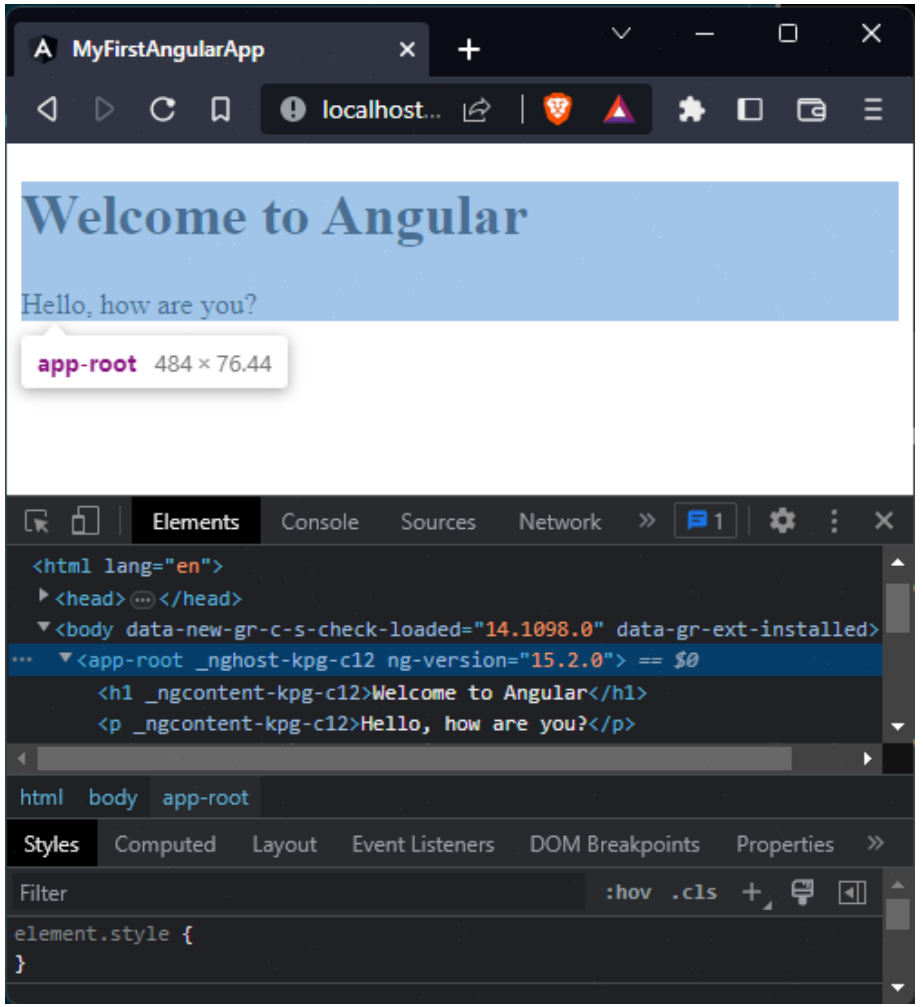
```
<h1>Welcome to Angular</h1>
<p>Hello, how are you?</p>
```



HTML tag `<app-root>` στο `index.html`

Ο controller της εφαρμογής είναι η Typescript κλάση `AppComponent` που βρίσκεται στο αρχείο `app.component.ts` όπου στις παραμέτρους του `@Component` decorator έχει ήδη οριστεί από το `ng new` η παράμετρος `selector` σαν `app-root`

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

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"],
})
export class AppComponent {
  title = "my-first-angular-app";
}
```



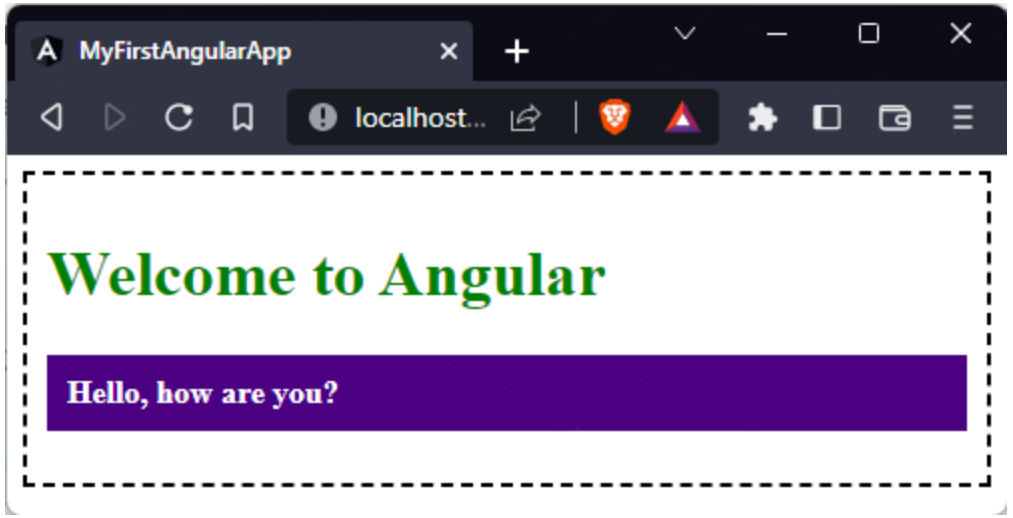
Τα αρχεία app.component.{html,css}

app.component.css

```css
.container {
  padding: 10px;
  border: dashed 2px black;
}
h1 {
  color: green;
}
p {
  background-color: indigo;
  font-weight: bold;
  color: white;
  padding: 10px;
}
```

app.component.html

```html
<div class="container">
  <h1>Welcome to Angular</h1>
  <p>Hello, how are you?</p>
</div>
```



Component-oriented web εφαρμογές

- Η ανάπτυξη εφαρμογών με το Angular Framework είναι *component-oriented* και επιτρέπει τη δημιουργία αυτόνομων συστατικών (components) που μπορούν να επαναχρησιμοποιηθούν και εύκολα να συνδυαστούν για να δημιουργήσουν πολύπλοκες διεπαφές χρήστη
- Βασικό πλεονέκτημα της προσέγγισης είναι πως επιτρέπει τη διάσπαση της εφαρμογής σε *μικρότερα πιο διαχειρίσιμα μέρη* και διευκολύνει τη συντήρηση και τις δοκιμές
- Τα component μπορούν να σχηματίζουν βιβλιοθήκες και να χρησιμοποιούνται σε διαφορετικά μέρη της εφαρμογής αλλά και σε διαφορετικές εφαρμογές
- Παρέχεται σαφής διαχωρισμός μεταξύ της λογικής και της παρουσίασης και διευκολύνεται η συντήρηση και η πραγματοποίηση αλλαγών χωρίς να επηρεάζεται συνολικά ο κώδικας

Ο διακοσμητής `@Component`

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

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"],
})
export class AppComponent {
  title = "my-first-angular-app";
}
```

- Είναι ένας διακοσμητής από τη βιλιοθήκη `@angular/core` που επιτρέπει τη δημιουργία ενός συστατικού της εφαρμογής μας όταν εφαρμοστεί σε μια κλάση της Typescript
- Αρχικά η εφαρμογή μας έχει ένα μόνο συστατικό, τον εαυτό της, την κλάση `AppComponent`
- Ο διακοσμητής προσθέτει στην κλάση μεταδεδομένα που χρησιμοποιεί το framework για να δημιουργήσει το component
- `selector` είναι το μοναδικό αναγνωριστικό του component που αυτόματα δημιουργεί νέο HTML tag
- `templateUrl` και `styleUrls` καθορίζουν την εμφάνιση του συστατικού

Νέο component `app-hello`

`app.component.ts`

```ts
import { Component } from "@angular/core";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"],
})
export class AppComponent {
  title = "my-first-angular-app";
}

@Component({
  selector: "app-hello",
  template: "<h1>Hello World!</h1>",
})
export class HelloComponent {}
```

`app.component.html`

```html
<div class="container">
  <h1>Welcome to Angular</h1>
  <p>Hello, how are you?</p>
  <hr />
  <app-hello></app-hello>
</div>
```

Λαμβάνουμε μήνυμα λάθους

```
Build at: 2023-03-01T07:24:49.595Z - Hash: 48007fdfe11414fb - Time: 395ms

Error: src/app/app.component.html:5:1 - error NG8001: 'app-hello' is not a known element:
1. If 'app-hello' is an Angular component, then verify that it is part of this module.
2. If 'app-hello' is a Web Component then add 'CUSTOM_ELEMENTS_SCHEMA' to the '@NgModule.schemas' of this component to suppress this message.

5 <app-hello></app-hello>
  ~~~~~~~~~~~

  src/app/app.component.ts:5:16
    5   templateUrl: './app.component.html',
                     ~~~~~~~~~~~~~~~~~~~~~~
    Error occurs in the template of component AppComponent.

× Failed to compile.
```

Λάβαμε το μήνυμα λάθους γιατί κάθε νέο συστατικό (component) πρέπει να δηλώνεται στην κεντρική ενότητα (module) της εφαρμογής μας που είναι το αρχείο `app.module.ts`

Η κεντρική ενότητα `app.module.ts`

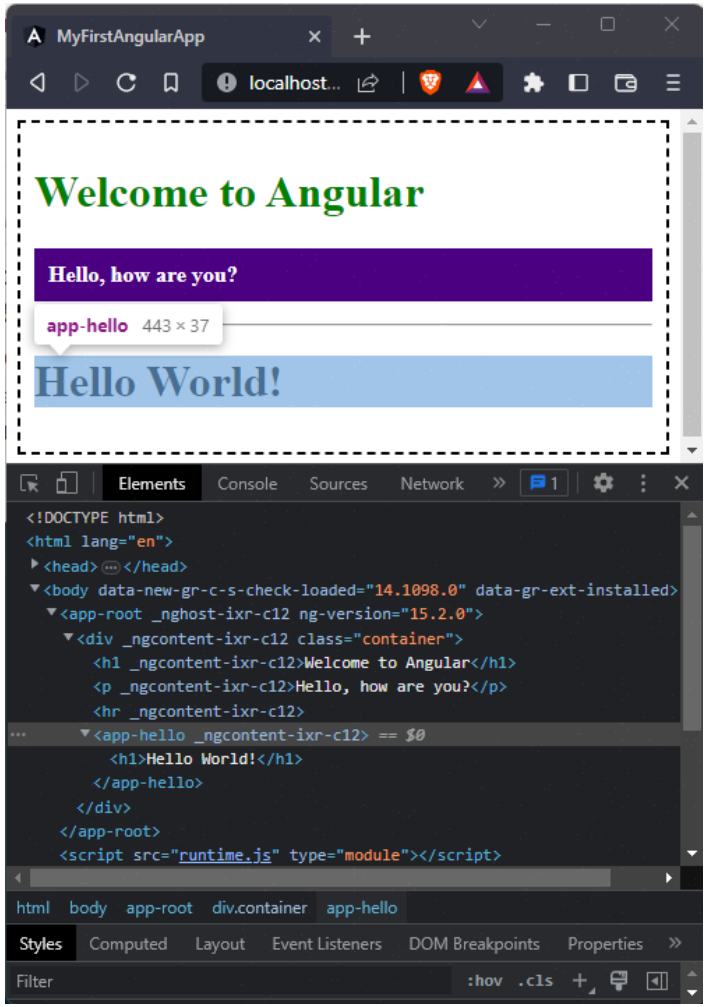
```
import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";

import { AppComponent, HelloComponent } from "./app.component";

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

- Χρησιμοποιεί το διακοσμητή `NgModule` από τη βιβιοθήκη `@angular/core`
- Περνά μεταδεδομένα στην κλάση `AppModule` που αναπαριστά σαν σύνολο την εφαρμογή που αναπτύσσουμε

- `declarations` είναι ένας πίνακας κλάσεων που αντιστοιχούν στα συστατικά (component) της εφαρμογής
- `imports` είναι ένας πίνακας απαραίτητων κλάσεων από το Angular Framework ή τρίτες πηγές για τη λειτουργία της εφαρμογής
- `providers` είναι πίνακας κλάσεων που παρέχουν λειτουργικότητα στην ενότητα
- `bootstrap` είναι το συστατικό βάσης (ρίζα όλων των άλλων) για την ενότητα

Το νέο component σε χρήση

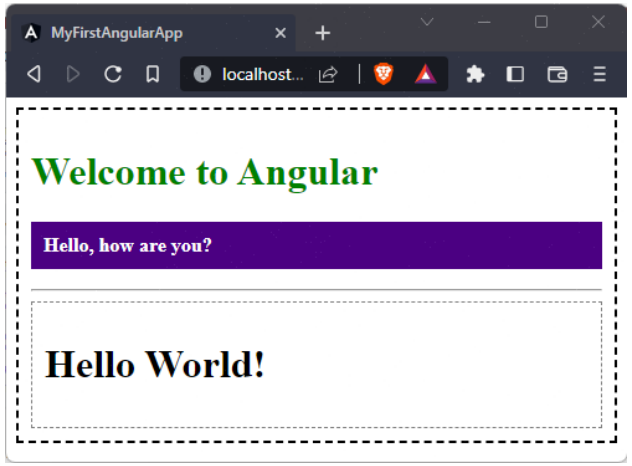
- Το νέο component με selector `app-hello` εμφανίζεται άμεσα εντός του κεντρικού component με selector `app-root`
- Έχουν δημιουργηθεί δύο νέα HTML tag, `<app-hello>` και `<app-root>` αντίτοιχα
- Το `<app-root>` περιέχει το `<app-hello>` με την ίδια λογική που ένα `<div>` tag μπορεί να περιέχει άλλο `<div>` tag
- Μπορούμε να δημιουργήσουμε νέο αρχείο Typescript `hello.component.ts` για το νέο component `app-hello`



Νέο αρχείο `hello.component.ts`

```ts
// hello.component.ts

import { Component } from "@angular/core";

@Component({
  selector: "app-hello",
  template: '<div class="container"><h1>Hello World!</h1></div>',
  styles: [".container { border: dashed 1px grey; padding: 10px}"],
})
export class HelloComponent {}
```

```ts
// app.module.ts

import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";

import { AppComponent } from "./app.component";
import { HelloComponent } from "./hello.component";

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



Υπάρχει δυνατότητα για *inline* template και style αν στο διακοσμητή `@Component` ορίσουμε τα metadata `template` και `style`

Η δυνατότητα είναι χρήσιμη αν αυτά είναι *λίγες γραμμές*, αλλιώς χρησιμοποιούμε τα `templateUrl` και `styleUrls`

Αυτόματη δημιουργία νέου component

Χρησιμοποιούμε το *Angular command line*

```
$ ng generate component new-hello
CREATE src/app/new-hello/new-hello.component.html (24 bytes)
CREATE src/app/new-hello/new-hello.component.spec.ts (614 bytes)
CREATE src/app/new-hello/new-hello.component.ts (213 bytes)
CREATE src/app/new-hello/new-hello.component.css (0 bytes)
UPDATE src/app/app.module.ts (760 bytes)
```

Δημιουργείται αυτόματα νέος κατάλογος και αρχεία σκελετοί και ενημερώνεται κατάλληλα το αρχείο `app.module.ts`

```
<!-- src/app/new-hello/new-hello.component.html -->

<p>new-hello works!</p>
```

```
// app.module.ts

...
import { NewHelloComponent } from './new-hello/new-hello.component';
...
declarations: [AppComponent, HelloComponent, NewHelloComponent]
...
```

```
// src/app/new-hello/new-hello.component.ts

import { Component } from "@angular/core";

@Component({
  selector: "app-new-hello",
  templateUrl: "./new-hello.component.html",
  styleUrls: ["./new-hello.component.css"],
})
export class NewHelloComponent {}
```

Το component είναι διαθέσιμο για άμεση χρήση σε template με χρήση του επιλογέα `<app-new-hello>`

Component και template

- Template ονομάζεται το αρχείο της HTML που *συνοδεύει* το component. *Το αρχείο της HTML συνδέεται άμεσα με το αρχείο της Typescript που περιέχει την κλάση του component.* Η σύνδεση των αρχείων γίνεται μέσω των *παραμέτρων* του διακοσμητή `@Component` που εφαρμόζεται στην κλάση του component.
- Το Angular Framework δίνει τη δυνατότητα να *χειριζόμαστε τα στοιχεία της HTML σαν αντικείμενα με τύπους* και να μπορούμε να ορίσουμε *δυναμικά* όλα τα στοιχεία του DOM.
- *Σχεδόν όλη η HTML σύνταξη είναι ορθή σύνταξη και για τα Angular templates*, όμως αφού το κάθε template είναι υποσύνολο του κεντρικού αρχείου `index.html`, δεν μπορούν να περιέχουν τα HTML tag `<html>`, `<base>` και `<body>`.
- Για λόγους ασφάλειας, όλα τα tag `<script>` και τα περιεχόμενά τους αγνοούνται
- *Υπάρχει άμεση δυνατότητα για παρεμβολή τιμών από την κλάση στο template με περίκλειση της ιδιότητας της κλάσης σε* `{{` *και* `}}`.

Παρεμβολή (interpolation) τιμών

`app.component.html`

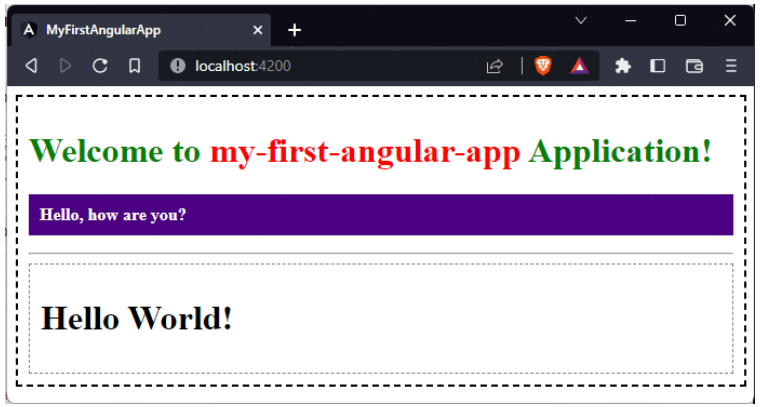
```
<div class="container">
  <h1>Welcome to <span>{{ title }}</span> Application!</h1>
  <p>Hello, how are you?</p>
  <hr />
  <app-hello></app-hello>
</div>
```

`app.component.css`

```
... span {
  color: red;
}
```

`app.component.ts`

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

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"],
})
export class AppComponent {
  title = "my-first-angular-app";
}
```



Μπορούμε να παρεμβάλουμε τιμές οπουδήποτε εντός της εμβέλειας των HTML tag ή στην ανάθεση τιμών στα χαρακτηριστικά τους, π.χ. `<p> {{ content }} </p>` ***ή*** `<img src="{{ imageURL }}">`

Παραδοσιακή παρεμβολή τιμών

- Με απλή Javascript, για να πετύχουμε παρεμβολή τιμών, ονοματίζουμε ένα HTML στοιχείο με ένα μοναδικό όνομα
- Στη συνέχεια περνάμε το όνομα σαν παράμετρο στη μέθοδο `document.getElementById` για να εντοπίσουμε προγραμματιστικά το στοιχείο
- Τέλος αναθέτουμε στην ιδιότητα `innerHTML` του στοιχείου την τιμή που θέλουμε να παρεμβάλουμε

```
<!DOCTYPE html>
<html>
  <body>
    <h1 id="greeting"></h1>

    <script>
      const name = "John";
      const greetingElement = document.getElementById("greeting");
      greetingElement.innerHTML = `Hello, ${name}!`;
    </script>
  </body>
</html>
```

Στο Angular Framework η παρεμβολή τιμών από τις ιδιότητες της κλάσης του controller στο template είναι άμεση και απαιτεί μόνο την χρήση των οριοθετητών `{{` και `}}`

Δέσμευση ιδιοτήτων (property binding) μεταξύ του component και του template

- Με τον όρο δέσμευση ιδιοτήτων εννοούμε τη δυνατότητα να δεσμευτεί (bind) μια ιδιότητα της κλάσης του component με την ιδιότητα της κλάσης κάποιου tag της HTML στο template.
- Μια ιδιότητα της κλάσης του component μπορεί να δεσμεύεται πολλές φορές στο template.
- *Οι αλλαγές στις τιμές των δεσμευμένων ιδιοτήτων αντικατοπτρίζονται άμεσα στο template.*

- Η ιδιότητα `value` του HTML `input` στο template δεσμεύει την ιδιότητα `username` της κλάσης του component
- Το γνώρισμα `src` του HTML `img` δεσμεύεται με την ιδιότητα `imageURL` της κλάσης του component

```
<input [value]="username" />
<img [src]="imageUrl" alt="Product Image" />
```

Χρησιμοποιούμε τα `[` και `]` για να περικλείσουμε την ιδιότητα του tag στο template οπότε *αυτό που ακολουθεί είναι όνομα ιδιότητας της κλάσης και όχι τιμή string*

Παραδείγματα παρεμβολής τιμών και δέσμευσης ιδιοτήτων της κλάσης στο template

Στο περιεχόμενο του `<p>` και στο `value` του `<input>`

```
<p>{{ message }}</p>
<input type="text" [value]="username" />
```

```
export class MyComponent {
  message = "Enter your name:";
  username = "";
}
```

Στο `src` του `<img>` και στο `disabled` του `<button>`

```
<img [src]="imageUrl" /> <button [disabled]="isDisabled">Click me</button>
```

```
export class MyComponent {
  imageUrl = "https://example.com/image.jpg";
  isDisabled = true;
}
```

Στο περιεχόμενο του `<a>` και στο `title` του `<a>`

```
<a [title]="tooltip">{{ linkText }}</a>
```

```
export class MyComponent {
  item = "Item 1";
  tooltip = "Click me for more info";
  linkText = "Learn more";
}
```

Στα `src` και `alt` του `<img>`

```
<img [src]="imageUrl" [alt]="imageAlt" />
```

```
export class MyComponent {
  imageUrl = "https://example.com/image.jpg";
  imageAlt = "Example image";
}
```

Δέσμευση συμβάντων στο template

- Με τον όρο *δέσμευση συμβάντων (event binding)* εννοούμε τη δυνατότητα να δεσμευτούν τα συμβάντα του template (mouse clicks, πληκτρολόγηση σε HTML input, κτλ) με μεθόδους της κλάσης του controller, ώστε να μεταφέρονται εκεί δεδομένα (κλίκ, χαρακτήρες, κτλ)
- Για να δεσμεύσουμε ένα συμβάν στο template με μια μέθοδο στον controller, αρχικά εσωκλείουμε το όνομα του συμβάντος σε `(` και `)` και ακολουθεί ανάθεση στο όνομα της μεθόδου που δεσμεύουμε που ενδεχομένως δέχεται και δεδομένα με το `$event`

`some.component.ts`

```
...
export SomeComponentClass {
    onButtonClick() {
        console.log('Button Clicked!');
    }
    onInputChange(data:string) {
        console.log(data);
    }
}
```

`some.component.html`

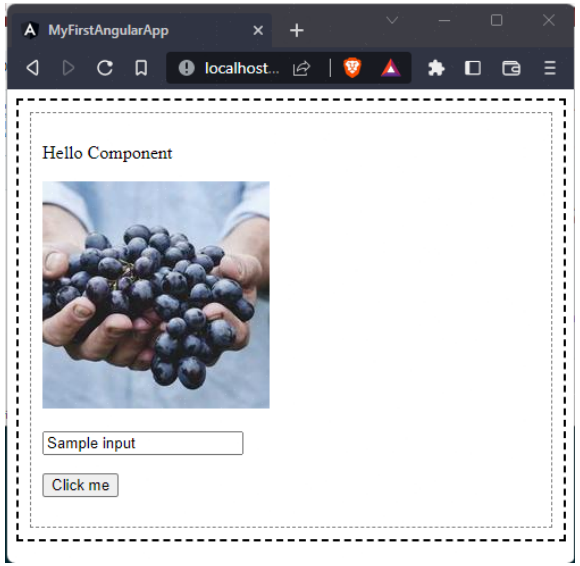
```
<button (click)="onButtonClick()">Click me</button>
<input (input)="onInputChange($event.target.value)" />
```

`$event` είναι ένα ειδικό αντικείμενο που "γεννιέται" όταν προκύπτει ένα συμβάν

Παράδειγμα δέσμευσης ιδιοτήτων και συμβάντων

hello.component.html

```
<div class="container">
  <p [textContent]="title"></p>
  <p><img [src]="imageUrl" [alt]="imageAlt" /></p>
  <p><input [value]="inputValue" (input)="onInputChange($event)" /></p>
  <p>
    <button [disabled]="isDisabled" (click)="onButtonClick()">Click me</button>
  </p>
</div>
```



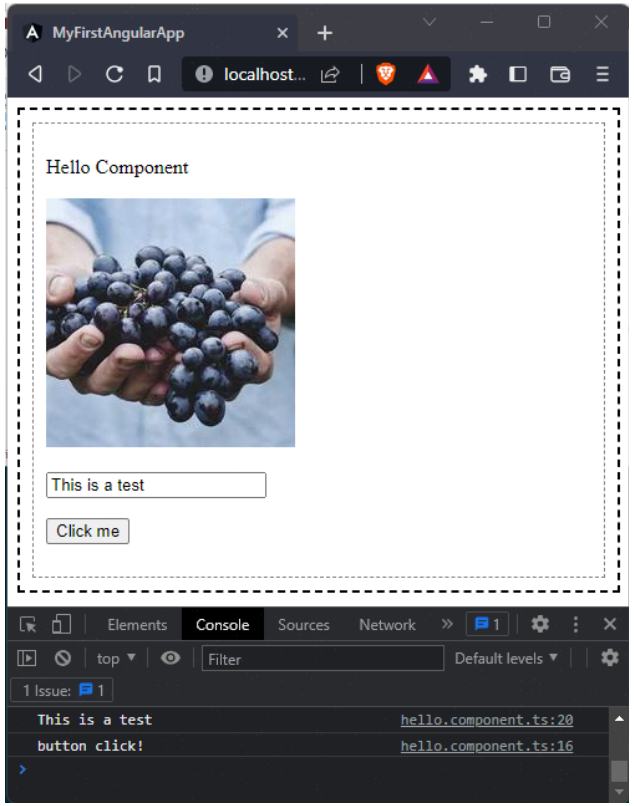
hello.component.ts

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

@Component({
  selector: "app-hello",
  templateUrl: "hello.component.html",
  styleUrls: ["hello.component.css"],
})
export class HelloComponent {
  title = "Hello Component";
  imageUrl = "https://picsum.photos/200";
  imageAlt = "A picture from lorem picsum";
  inputValue = "Sample input";
  isDisabled = false;

  onButtonClick() {
    console.log("button click!");
  }

  onInputChange(event: Event) {
    console.log((event.target as HTMLInputElement).value);
  }
}
```

Παράδειγμα δέσμευσης ιδιοτήτων και συμβάντων

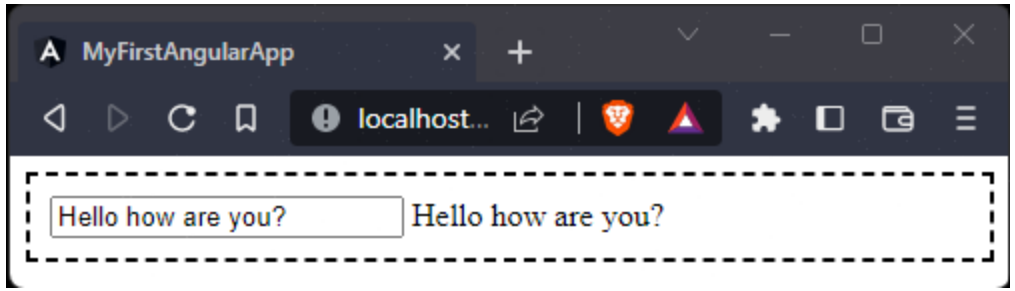
Δέσμευση ιδιοτήτων σε δράση

Δέσμευση συμβάντων σε δράση



Διπλή δέσμευση με το `ngModel`

- Η διπλή δέσμευση είναι δέσμευση ιδιότητας και συμβάντος την ίδια στιγμή. Μια ιδιότητα της κλάσης δεσμεύεται με το input HTML στοιχείο στο template και οι αλλαγές στο input στο template ρέουν πίσω στην κλάση σαν αυτόματα συμβάντα
- Η διπλή δέσμευση γίνεται με το `[(ngModel)]="property"` στο template, όπου `property` είναι η ιδιότητα της κλάσης που δεσμεύεται διπλά
- Για να ενεργοποιηθεί η δυνατότητα χρήσης του `ngModel` στο template είναι απαραίτητο να εισάγουμε το `FormsModule` από την ενότητα `@angular/forms`



Μπορούμε πολύ εύκολα να δημιουργήσουμε ένα HTML input που οτιδήποτε πληκτρολογούμε μέσα του αντικατοπτίζεται άμεσα και σε άλλο μέρος της σελίδας

Διπλή δεύσμευση με το ngModel

app.module.ts

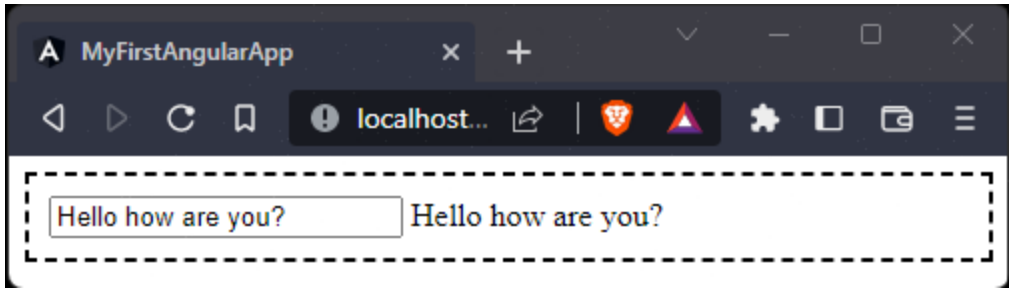
```
import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";
import { FormsModule } from "@angular/forms";

import { AppComponent } from "./app.component";
import { HelloComponent } from "./hello.component";
import { DDComponent } from "./dd.component";

@NgModule({
  declarations: [AppComponent, HelloComponent, DDComponent],
  imports: [BrowserModule, FormsModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

app.component.html

```
<div class="container">
  <app-dd></app-dd>
</div>
```

dd.component.ts

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

@Component({
  selector: "app-dd",
  template: '<input [(ngModel)]="property"> {{ property }}',
})
export class DDComponent {
  property = "Initial";
}
```

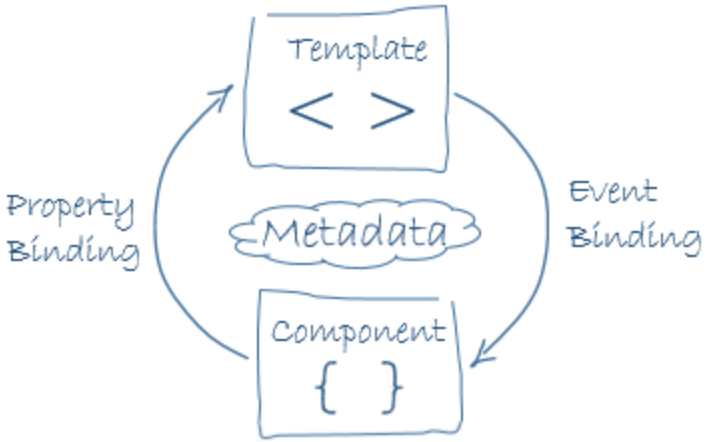


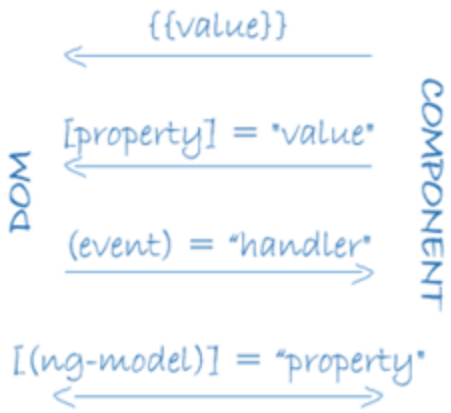
Διπλή δεύσμευση με απλή Javascript

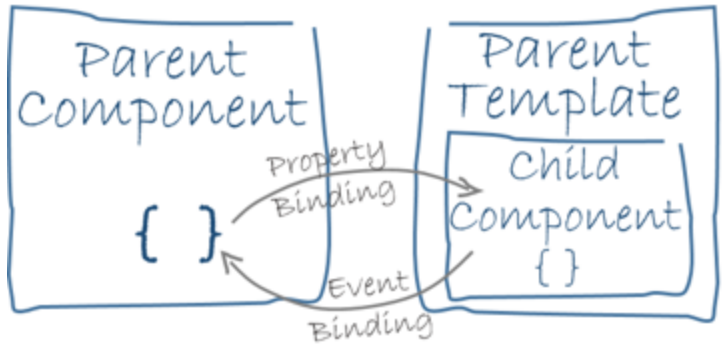
```
<!DOCTYPE html>
<html>
  <body>
    <input id="input-field" type="text" />
    <p id="output-field"></p>

    <script>
      const inputField = document.getElementById("input-field");
      const outputField = document.getElementById("output-field");

      inputField.addEventListener("input", function () {
        outputField.textContent = inputField.value;
      });
    </script>
  </body>
</html>
```

Όλοι οι μορφές της δέσμευσης

- Το Angular Framework επεξεργάζεται όλες τις μορφές της δέσμευσης ανά κάθε κύκλο των συμβάτων της Javascript και από το κεντρικό component της εφαρμογής προς όλα τα περιεχόμενα component
- Η δέσμευση έχει κεντρική θέση στην επικοινωνία δεδομένων μεταξύ του template και του component αλλά και μεταξύ των διάφορων component







Επικοινωνία από τον γονέα προς το παιδί

Ενα component template (γονέας) μπορεί να περιέχει επιλογέα άλλου component (παιδί)

```
// parent component

import { Component } from "@angular/core";

@Component({
  selector: "app-parent",
  template: `
    <h1>Parent Component</h1>
    <app-child [message]="parentMessage"></app-child>
  `,
})
export class ParentComponent {
  parentMessage = "Hello from the parent component";
}
```

```
// child component

import { Component, Input } from "@angular/core";

@Component({
  selector: "app-child",
  template: `
    <h2>Child Component</h2>
    <p>{{ message }}</p>
  `,
})
export class ChildComponent {
  @Input() message: string;
}
```

Η επικοινωνία του γονέα προς το παιδί γίνεται με την εφαρμογή του διακοσμητή `@Input()` σε μια ιδιότητα της κλάσης του παιδιού. Η εφαρμογή του διακοσμητή κάνει την ιδιότητα του παιδιού διαθέσιμη για δέσμευση από κάποιο γονέα. Η δέσμευση γίνεται στο template του γονέα στον επιλογέα του παιδιού όπου περικλείεται η ιδιότητα του παιδιού σε `[` και `]` και ανατίθεται η ιδιότητα του γονέα που δεσμεύεται *(είναι το όνομα της ιδιότητας, όχι string)*

Επικοινωνία από το παιδί προς τον γονέα

Το παδί επικοινωνεί με κάποιο γονέα με δέσμευση των συμβάντων του παιδιού στον γονέα

```
// parent component

import { Component } from "@angular/core";

@Component({
  selector: "app-parent",
  template: `
    <h1>Parent Component</h1>
    <app-child (messageSent)="onMessageReceived($event)"></app-child>
    <p>Message received from child: {{ receivedMessage }}</p>
  `,
})
export class ParentComponent {
  receivedMessage: string;

  onMessageReceived(message: string) {
    this.receivedMessage = message;
  }
}
```

```
// clild component

import { Component, EventEmitter, Output } from "@angular/core";

@Component({
  selector: "app-child",
  template: `
    <h2>Child Component</h2>
    <button (click)="sendMessage()">Send Message to Parent</button>
  `,
})
export class ChildComponent {
  @Output() messageSent = new EventEmitter<string>();

  sendMessage() {
    const message = "Hello from the child component";
    this.messageSent.emit(message);
  }
}
```

Ο διακοσμητής `@Output()` εφοδιάζει το χαρακτηριστικό του παιδιού με την ικανότητα να δεσμευτεί σαν συμβάν από κάποιο γονέα. Εφαρμόζεται σε στιγιότυπο της κλάσης `EventEmitter` που επιτρέπει τη μετάδοση δεδομένων (με τύπο) προς το γονέα

Οδηγίες (directives)

- Οι *οδηγίες (directives)* στο Angular Framework μας επιτρέπουν να επεκτείνουμε τη συμπεριφορά των HTML tags και να δημιουργούμε επαναχρησιμοποιήσιμη λειτουργικότητα.
- Οι *οδηγίες γνωρισμάτων (attribute directives)* τροποποιούν τη συμπεριφορά ή την εμφάνιση ενός HTML tag. Χρησιμοποιούμε τον επιλογέα (selector) της οδηγίας σαν επιπλέον γνώρισμα του HTML tag και έτσι προσθέτουμε ή αφαιρούμε γνωρίσματα, αλλάζουμε το στύλ ή προσθέτουμε λειτουργικότητα, π.χ. `ngStyle`, `ngClass`
- Οι *δομικές οδηγίες (structural directives)* τροποποιούν τη δομή του DOM με την προσθήκη, διαγραφή ή χειρισμό των HTML tags. Με τη χρήση τους μπρούμε να εμφανίζουμε HTML tags υπό όρους ή να διατρέχουμε πίνακες δεδομένων που "γεννούν" αντίστοιχα HTML tags, τελικά δηλαδή να *διαμορφώνουμε δυναμικά templates*. Χαρακτηριστική είναι η χρήση του `*` πριν το όνομα του επιλογέα, π.χ. `*ngIf`, `*ngFor`
- Μια Typescript κλάση γίνεται οδηγία με τη χρήση του διακοσμητή `@Directive`

Attribute directives

Η οδηγία `ngClass` επιτρέπει την υπο όρους εφαρμογή μιας CSS κλάσης στο HTML tag, π.χ. αν το `isActive` είναι `true` εφαρμόζεται η κλάση `highlight`

```
<div [ngClass]="{'highlight': isActive}">This element is highlighted</div>
```

Η οδηγία `ngStyle` επιτρέπει την υπο όρους εφαρμογή ενός CSS στύλ στο HTML tag

```
<div [ngStyle]="{'font-size': isLarge ? '24px' : '16px', 'color': isDark ? 'black' : 'white'}" > Hello world </div>
```

Η οδηγία `disabled` επιτρέπει την υπο όρους απενεργοποίηση ενός HTML tag

```
<button [disabled]="isDisabled">Click me</button>
```

Η οδηγία `hidden` επιτρέπει την υπό όρους απόκρυψη ενός HTML tag

```
<div [hidden]="isError">No errors to display</div>
```

Structural directives

Η οδηγία `*ngIf` επιτρέπει την υπο όρους *προσθήκη* (αλλιώς *διαγραφή*) στο DOM

```
<div *ngIf="isShown">This element is shown</div>
```

Η οδηγία `*ngFor` επιτρέπει τη διάσχιση ενός πίνακα και τη *δημιουργία* αντίστοιχων στοιχείων στο DOM

```
<ul>
  <li *ngFor="let item of items">{{ item }}</li>
</ul>
```

Η οδηγία `ngSwitch` επιτρέπει την υπο όρους *προσθήκη* στο DOM με βάση σύνολο τιμών

```
<div [ngSwitch]="myValue">
  <div *ngSwitchCase="'value1'">This is value 1</div>
  <div *ngSwitchCase="'value2'">This is value 2</div>
  <div *ngSwitchDefault>This is the default value</div>
</div>
```

Η οδηγία `ngTemplateOutlet` δέχεται αναφορά σε template και το εισάγει δυναμικά στο DOM

```
<ng-container *ngTemplateOutlet="myTemplate"></ng-container>

<ng-template #myTemplate>
  <div>This is my template</div>
</ng-template>
```

Ομαδοποίηση με το `ng-container`

Η χρήση του `ng-container` επιτρέπει την ομαδοποίηση στοιχείων του DOM χωρίς τη χρήση επιπλέον στοιχείου στο DOM (π.χ. `<div>`).

Χρήση σαν "περιτύλιγμα" δομικών οδηγιών για την προσθήκη υπό όρους στο DOM

```
<ng-container *ngIf="isShown">
  <div>This content is shown when isShown is true</div>
</ng-container>
```

Χρήση σαν placeholder στοιχείων που θα προστεθούν στο DOM αργότερα

```
<ng-container *ngTemplateOutlet="myTemplate"></ng-container>

<ng-template #myTemplate>
  <div>This is my template</div>
</ng-template>
```

Χρήση σαν ιδεατός χώρος ομαδοποίησης χωρίς τη χρήση επιπλέον στοιχείου στο DOM

```
<div *ngFor="let item of items">
  <ng-container *ngIf="item.isActive">
    <div>{{ item.name }}</div>
    <div>{{ item.description }}</div>
  </ng-container>
</div>
```

Ο διακοσμητής `@Directive`

- Δημιουργούμε προσαρμοσμένες (custom) οδηγίες με την εφαρμογή του διακοσμητή `@Directive` σε μια κλάση Typescript
- Προσθέτουμε την κλάση στον πίνακα `Declarations` στο `app.module.ts`
- Το όνομα της νέας οδηγίας περνά σαν το μεταδεδομένο `selector` στο διακοσμητή
- Στο παράδειγμα χρησιμοποιούμε το όνομα `[appHighlight]`
- Μέσω του `ElementRef` περνά η αναφορά στο HTML tag που εφαρμόζεται η οδηγία

- Η οδηγία του παραδείγματος αλλάζει στο DOM το χρώμα του φόντου στο HTML που εφαρμόζεται

```
import { Directive, ElementRef } from "@angular/core";

@Directive({
  selector: "[appHighlight]",
})
export class HighlightDirective {
  constructor(private el: ElementRef) {
    this.el.nativeElement.style.backgroundColor = "yellow";
  }
}
```

- Χρήση της οδηγίας

```
<p appHighlight>Highlight me!</p>
```

Αυτόματη δημιουργία οδηγίας

Χρησιμοποιούμε το Angular command line

```
$ ng generate directive example
CREATE src/app/example.directive.spec.ts (228 bytes)
CREATE src/app/example.directive.ts (143 bytes)
UPDATE src/app/app.module.ts (760 bytes)
```

Δημιουργούνται αυτόματα νέοι σκελετοί αρχείων και ενημερώνεται κατάλληλα το αρχείο `app.module.ts`

```
// src/app/app.module.ts

...
import { ExampleDirective } from './example.directive';
...
declarations:[...,ExampleDirective,...]
...
```

Σκελετός για τη νέα οδηγία

```
// src/app/example.directive.ts

import { Directive } from "@angular/core";

@Directive({
  selector: "[appExample]",
})
export class ExampleDirective {
  constructor() {}
}
```

Single Page Applications (SPA)

- Οι εφαρμογές που αναπτύσσονται με το Angular Framework λειτουργούν στα πλαίσια *μιας και μοναδικής σελίδας*, αυτής που παράγεται από το αρχείο `src/index.html`.
- Το περιεχόμενο της σελίδας ενημερώνεται δυναμικά με την αλληλεπίδραση του χρήστη.
- Καθώς ο χρήστης πλοηγείται στην εφαρμογή *δεν φορτώνονται νέες σελίδες* από τον server.
- Όλα συμβαίνουν αυτόματα στην ίδια σελίδα δίνοντας μια ομαλή εμπειρία που είναι παρόμοια με μια εφαρμογή που εκτελείται στο λειτουργικό του υπολογιστή.
- Βασικά εργαλεία του Angular Framework για την ανάπτυξη SPA:
  - *Αρχιτεκτονική βασισμένη στα component:* Κάθε component είναι μια αυτόνομη μονάδα διεπαφής χρήστη και λειτουργικότητας που συνδυάζεται και ενσωματώνεται άμεσα.
  - *Angular Router:* Αντιστοιχίζει URL στα component και επιτρέπει την πλοήγηση μεταξύ τους, ενώ παράλληλα διατηρεί και το ιστορικό του προγράμματος περιήγησης.

Χρήση του `RouterModule`

- Το `RouterModule` εφοδιάζει την εφαρμογή μας με τη δυνατότητα πλοήγησης μεταξύ των components.
- `Routes` είναι ένα *interface* για τα αντικείμενα της πλοήγησης, *απαραίτητα* με
  - `path` : το URL που επιθυμούμε
  - `component` : το αντίστοιχο component
- Κενό `path` έχει η "αρχική σελίδα" της εφαρμογής, ενώ το `'**'` είναι για τις περιπτώσεις που δεν υπάρχει αντίστοιχο component για το URL.

```
// app.module.ts
...
import { RouterModule, Routes } from "@angular/router";
...
const routes: Routes = [
  { path: 'path_to_A', component: AComponent },
  { path: 'path_to_B', component: BComponent },
  { path: '', component: WelcomeComponent },
  { path: '**', component: PageNotFoundComponent },
];
...
@NgModule({
...
imports: [..., RouterModule.forRooot(routes), ...],
...
})
```

- Στο import χρησιμοποιούμε το `forRoot()` για το βασικό module της εφραρμογής και το `forChild()` για τα submodule (θα αναφερθούμε αργότερα).

`<router_outlet>` στο βασικό template και `routerLink` για την πλοήγηση

- Το `RouterModule` εφοδιάζει τα template με το tag `<router_outlet>` που λειτουργεί σαν το *πλαίσιο της εμφάνισης του component* όταν ο χρήστης πλοηγηθεί στο *αντίστοιχο path*.
- Για την πλοήγηση δημιουργούμε "μενού πλοήγησης" με την HTML, χρησιμοποιούμε όμως το directive `routerLink` αντί του `href`.
- Το "μενού πλοήγησης" μπορεί να είναι ένα διαφορετικό component.

```
<!-- app.component.html -->

<div class="menu">
  <a routerLink="path_to_A"> Menu Item for A </a>
  <a routerLink="path_to_B"> Menu Item for B </a>
</div>

<router_outlet></router_outlet>
```

- Κατά την πλοήγηση από το ένα component στο άλλο, στο πλαίσιο του `<router_outlet>`, τα component *καταστρέφονται και επαναδημιουργούνται* ανάλογα με το ενεργό `path` (διαπιστώστε το με χρήση των Developer Tools).