# **Angular Definitions**

**Angular**

Framework מבית גוגל לבניית אתרי SPA.

התקנת אנגולר במחשב פעם אחת ברמה הגלובלית: npm i -g @angular/cli

התקנת פרויקט חדש בתיקייה ספציפית: ng new my-project-name

הרצת פרויקט אנגולר: ng serve --open

קיצור: ng s -o

אפשר לכתוב ב-package.json פקודה זו מאפיין start:

"start": "ng serve --open",

ואז מריצים ע"י הפקודה npm start

**Angular Directory Structure**

src

app

components

layout-area

header

header.component.css

header.component.html

header.component.ts

footer

menu

home-area

products-area

models

services

redux

assets

images

video

environments

environment.prod.ts

environment.ts

index.html

styles.css

הוספת Component ע"י הטרמינל (רכיב בשם Layout בתוך layout-area שנמצא בתיקיית Components):

ng g c components/layout-area/layout --skip-tests

**Annotation**

הוראה שאנו נותנים ל-Framework של אנגולר ע"י פקודה המתחילה ב-@ ונכתבת מעל למחלקה/פונקציה/משתנה וכו'...

ההוראה אומרת לאנגולר להתייחס למחלקה/פונקציה/משתנה בצורה כלשהי או לבצע דברים מסוימים איתם.

לדוגמה @Component נכתב מעל מחלקה ומתארת לאנגולר שהמחלקה הינה Component, כמו כן-מיהם קבצי ה-Component השונים ומהי תגית ה-HTML של ה-Component.

selector – הגדרת שם התגית של ה-Component

templateUrl – מיהו קובץ ה-HTML של ה-Component

styleUrls – מיהו קובץ ה-CSS של ה-Component

את התגית של ה-Component הראשון שעולה (לרוב ה-Layout) אנו שמים ב-index.html

**Module**

מעטפת של מספר רכיבים ואלמנטים נוספים המאוגדים יחד.

כברירת מחדל קיימים לנו שני מודולים:

app.module – המודול הראשי.

app-routing.module – מודול עבור מנגנון ה-Routing

ניתן לחלק את המערכת למודולים שונים.

כל רכיב חייב להיות שייך למודול אחד ויחיד.

ה-Component הראשון שעולה (לרוב ה-Layout) צריך להיות מוגדר בפקודה בשם bootstrap שב-app.module.

פקודות מרכזיות השייכות ל-Module:

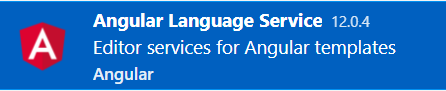
declarations – אלו רכיבים ואלמנטים נוספים משויכים למודול

imports – אלו מודולים אחרים המודול שלנו צריך להכיר

providers – אלו services שייכים למודול שלנו

bootstrap – מיהו הרכיב הראשי שעולה באתר – קיים רק ב-app.module

תוסף ל-VS CODE עבור מציאת פקודות אנגולר:



**Directive**

הוראה הניתנת לתגית.

היא נכתבת כתחביר של -HTML Attribute.

קיימים שני סוגי Directives:

1. Attribute Directive
2. Structural Directive

**Attribute Directive**

זוהי Directive המבצעת שינוי כלשהו בתגית שיכול להיות שינוי תצוגה, שינוי התנהגות, שינוי מראה (HTML) וכדומה.

התגית קיימת, ה-Attribute Directive משנה אותה באיזשהו אופן.

לדוגמה ngClass יכולה לתת לתגית CSS Class גם בצורה דינאמית.

**Structural Directive**

זוהי Directive שגורמת לתגית להיות ב-DOM בתנאי מסוים, או גורמת לתגית להיות ב-DOM מספר פעמים.

כל Structural Directive מתחיל ב-\*

לדוגמה \*ngIf מאפשרת להציג תגית כתלות בתנאי מסוים (כמו Conditional Rendering)

לדוגמה \*ngFor מאפשרת לשכפל תגית לפי מספר הפריטים שיש במערך ולהציג בתגיות את תוכן המערך.

**@Input**

זהו Annotation שנכתב מעל משתנה מחלקת Component ומאפשר ל-Parent Component לשלוח אליו את המידע כ-attribute, בדומה ל-props של React.

**Property Binding**

הכנסת ערך הקיים במשתנה מחלקת Component ל-HTML Attribute.

עוטפים את ה-HTML Attribute בסוגריים מרובעים ואז ניתן להכניס לתוכם משתנה.

**Events**

מתבצעים ע"י עטיפת סוגריים עגולים מסביב לשם האירוע בתגית ה-HTML וקריאה לפונקציה.

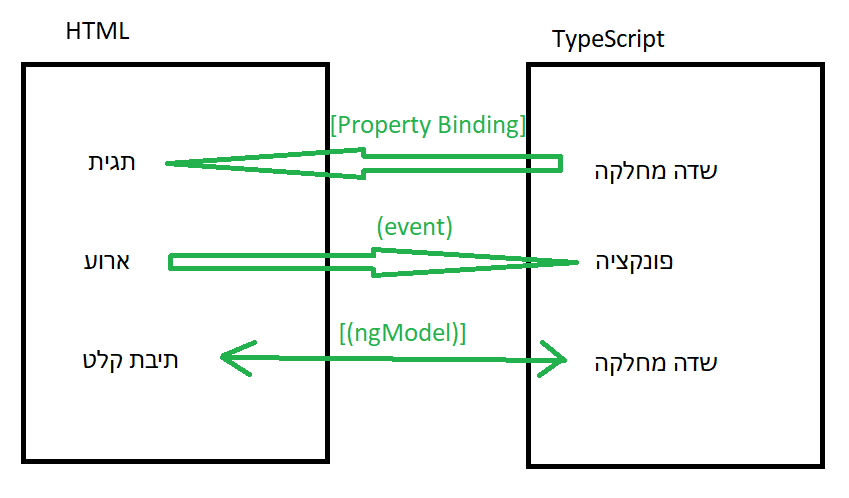
הפונקציה יכולה לקבל args מטיפוס ספציפי (יש לבדוק ב-console מה הטיפוס) אך במצב כזה יש לשלוח לאירוע את המשתנה $event.

**Two-Way Binding**

קישור תיבת input או select או textarea למשתנה מחלקה ב-Component.

כל שינוי של התיבה תגרום להכנסת הערך מהתיבה למשתנה.

כל שינוי של המשתנה – יוצג מיידית בתיבה.



[( )] נקרא Banana in a Box Syntax

לצורך ביצוע Two-Way Binding עלינו לייבא את FormsModule למודול שלנו (לדוגמה ל-AppModule). אז ניתן יהיה להשתמש בפקודת [(ngModel)] המשמשת ל-Two-Way Binding.

**State**

אין שום צורך בשום אובייקט State באנגולר, כי כל משתנה ברמת מחלקה מהווה State של ה-Component.

אנגולר אוטומטית תדע לרפרש את ה-UI אם כל משתנה יקבל ערך אחר.

**CSS Encapsulation**

ה-CSS Modules של React. אין שום צורך בזה באנגולר כי כל עיצוב של תגית ב-Component תקף אך ורק ל-Component שלו. אין כאן זליגת CSS. ה-CSS Modules של React מיושם כאן אוטומטית ברמת ה-Components.

**Pipe**

פונקציה מקבלת ערך של Interpolation, מבצעת עליו שינוי כלשהו, מחזירה ערך חדש שמוצג במקום של ה-Interpolation.

פונקציית Pipe יכולה לקבל ערכים לצורך הפעילות שלה. הערכים נשלחים אחרי סימן נקודותיים.

**Lifecycle Hooks**

פונקציות שמתבצעות בזמנים ספציפיים של ה-Component.

כל פונקציה כזו (פרט ל-ctor) מגיעה מ-interface שעל ה-Component לממש (לא חובה אבל מקובל).

Lifecycles נפוצים:

1. constructor – האובייקט נוצר, נועד לאתחול מידע, נועד לביצוע DI (נלמד בהמשך).
2. ngOnInit – הרכיב מוכן לשימוש, הפונקציה מתבצעת פעם אחת, מיועדת לביצוע Side-Effects.
3. ngOnDestroy – הרכיב הולך להיהרס, אך עדיין לא נהרס. מיועד לביצוע פעולות סיום, פעולות סגירה וכו' ברמת הרכיב.

**Service**

זו מחלקת לוגיקה. מכילה פעילות לוגית ללא UI.

זה לא נכון שאנו המתכנתים נייצר אובייקט ממחלקת ה-Service.

לא מקובל לייצר אובייקט ממחלקת ה-Service בעצמנו.

קיים מנגנון המאפשר לייצר אוטומטית אובייקט ממחלקת ה-Service.

אובייקט זה יכול להיות נגיש במספר Scopes באפליקציה.

המנגנון נקרא: DI = Dependency Injection

**DI – Dependency Injection**

זהו מנגנון של Framework או סביבה כלשהי המאפשרת לייצר אובייקט ממחלקה ולהביא לנו את האובייקט מוכן לשימוש.

לצורך פעילות ה-Component אנו תלויים באובייקט הזה, ולכן מגיע השם Dependency.

המערכת יוצרת את האובייקט ו-"מזריקה" אותו ל-Component שלנו, ולכן מגיע השם Injection.

כל מה שצריך לבצע עבור DI זה:

1. לומר לאנגולר באיזה Scope של האפליקציה לייצר את האובייקט.

קיימים שלושה Scopes שבהם ניתן לייצר אובייקט עבור DI:

* Scope ראשון: ברמת ה-Component בלבד – האובייקט יהיה ניתן לשימוש רק ע"י ה-Component.
* Scope שני: ברמת Module – האובייקט יהיה ניתן לשימוש ע"י כל מי שמשויך למודול הזה.
* Scope שלישי: ברמת כל האפליקציה – האובייקט יהיה ניתן לשימוש בכל הרכיבים של כל האפליקציה.

יש שתי דרכים לבצע זאת:

דרך ראשונה: אם הגדרנו את ה-Service ברמת ה-AppModule.

דרך שנייה: הגדרת Annotation בשם @Injectable מעל למחלקת ה-Service המכילה את המאפיין providedIn: "root". זו השיטה המומלצת:

@Injectable({ providedIn: "root" })

1. יש לבקש את האובייקט ב-Constructor.

**מנגנון Routing**

מנגנון המאפשר להציג Component כתלות ב-Route.

הגדרת מנגנון ה-Routing מתבצעת בקובץ app-routing.module.ts.

כל Route מכיל את הנתיב עצמו ואיזה Component להציג.

ע"י התגית <router-outlet></router-outlet> אנו קובעים היכן רכיבי ה-Route יוצגו. במיקום התגית הזו – הם יוצגו על הדף.

בתפריט, במקום <a href="/some-route"> אנו משתמשים ב-<a routerLink="/some-route"> לצורך מניעת גלישה לשרת בהתאם ל-SPA.

**Web Access**

גלישה לשרת מרוחק והחזרת מידע.

ראשית, עלינו לבנות מחלקת Model מתאימה.

שנית, עלינו להביא את המידע. באנגולר קיים Service שמאפשר לגלוש לשרת מרוחק ולהביא מידע (בדומה ל-Axios). הוא נקרא HttpClient. הוא נמצא ב-Module בשם HttpClientModule שעלינו להביא ל-AppModule.

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

// …

imports: [

// …

HttpClientModule

],

**בניית טופס**

קיימות שתי שיטות:

1. Template Form: אנו יוצרים אובייקט ריק במחלקת ב-Component ומבצעים Two-Way Binding אליו ע"י תיבות הטקסט של הטופס. בנוסף הוולידציה מתבצעת ע"י Attributes ב-HTML.
2. Reactive Form: לא משתמשים ב-Two-Way Binding אלא מקשרים בין תיבות הקלט לאובייקטים מיוחדים שנמצאים ב-TypeScript. אובייקטים אלו מכילים בתוכם גם הגדרות ולידציה.

**ולידציה**

אימות קלט המשתמש בצד ה-Frontend.

לכל תיבת קלט יש מאפיינים הקשורים לפעילות שלה. אלו שלושה זוגות של מאפיינים:

זוג ראשון:

pristine – ערך התיבה עדיין לא שונה.

dirty – ערך התיבה שונה

זוג שני:

untouched – הפוקוס עדיין לא הגיע או לא עזב את התיבה

touched – הפוקוס כבר עזב את התיבה (ביקרו בה)

זוג שלישי:

invalid – ערך התיבה לא חוקי בהתאם לוולידציה שלה.

valid – ערך התיבה חוקי בהתאם לוולידציה שלה.

גם ל-form יש את ששת המאפיינים הנ"ל שהינם מעיין סיכום של כל תיבות הקלט.

שלבי ולידציה:

1. הגדרת הוולידציה ע"י Attributes בתיבות.

לדוגמה, required עבור שדה חובה, email עבור שדה אימייל, minlength עבור אורך מינימלי וכו'.

1. הצגת הודעת שגיאה המתאימה לבעיה שתוצג רק אם מצב התיבה הוא invalid
2. ביטול אפשרות לחיצה על הלחצן אם הוולידציה של הטופס נכשלה

**Regular Expression (RegEx)**

תחביר לביצוע פעולות על מחרוזות כמו בדיקות, חיפוש, החלפה ועוד.

קיימת ולידציה בשם pattern שמקבלת תחביר RegEx.

**Observable**

אובייקט המאפשר להריץ קוד אסינכרוני ולדווח הצלחה או כשלון כמו ש-Promise יודע.

Observable מכיל יותר יכולות מ-Promise:

השוואה:

* Promise מתחיל לעבוד אוטומטית כאשר יצרנו אותו ע"י new.

Observable לא מתחיל לעבוד אוטומטית ברגע שיצרנו אותו, אלא רק כאשר נרשמנו לפונקציית דיווח ההצלחה שלו (ביצוע subscribe)

* Promise ברגע שהתחיל – לא ניתן לעצירה או לביטול.

Observable מכיל פונקציית cancel לצורך ביטול או עצירת הפעילות שלו, אפילו שהוא עדיין לא סיים.

* Promise יכול לדווח הצלחה אחת (resolve) או כשלון אחד (reject). כלומר Promise מיועד לביצוע פעולה אסינכרונית בעלת תוצאה אחת ויחידה.

Observable יכול לדווח המון הצלחות (next) – אחת אחרי השנייה או כשלון אחד (error). כלומר Observable מיועד לביצוע דיווח בלתי פוסק של מידע. דומה ל-stream.

* ברגע ש-Promise דיווח את ההצלחה שלו (resolve) הוא מסתיים אוטומטית ולא ניתן יותר לשימוש.

Observable מסוגל לדווח הצלחות שוב ושוב (next) מבלי שהוא מסתיים או נהרס.

* ל-Observable יש פונקציה נוספת שהוא מדווח בשם complete המיועדת לציין שכל רצף הדיווחים הסתיים בהצלחה. לאחר מכן ה-Observable נחשב כהסתיים לגמרי ולא ניתן לשימוש יותר. אם דווח כשלון (error) הוא גם מסתיים בדומה ל-Promise.
* ב-Promise ניתן להשתמש ב-async-await.

ב-Observable לא ניתן להשתמש ב-async-await.

* Observable יכול להמיר את עצמו ל-Promise אבל זה יהיה הגיוני אך ורק אם ה-Observable מדווח הצלחה אחת ויחידה.

Promise לא מכיל אפשרות להמרה ל-Observable.

* Observable מכיל API שלם לביצוע פעולות או חישובים על המידע שהוא מדווח, כאשר המידע המדווח ע"י ה-Observable הינו מידע לאורך זמן. פעילות זו נקראת Reactive Programming.

**Angular Material**

זוהי הספרייה של MUI (Material UI) עבור אנגולר.

העיצובים מאוד דומים כי זה אותו הקונספט.

השימוש מתאים לאנגולר.

(גוגל הם אלו שכתבו את MUI, את Angular Material)

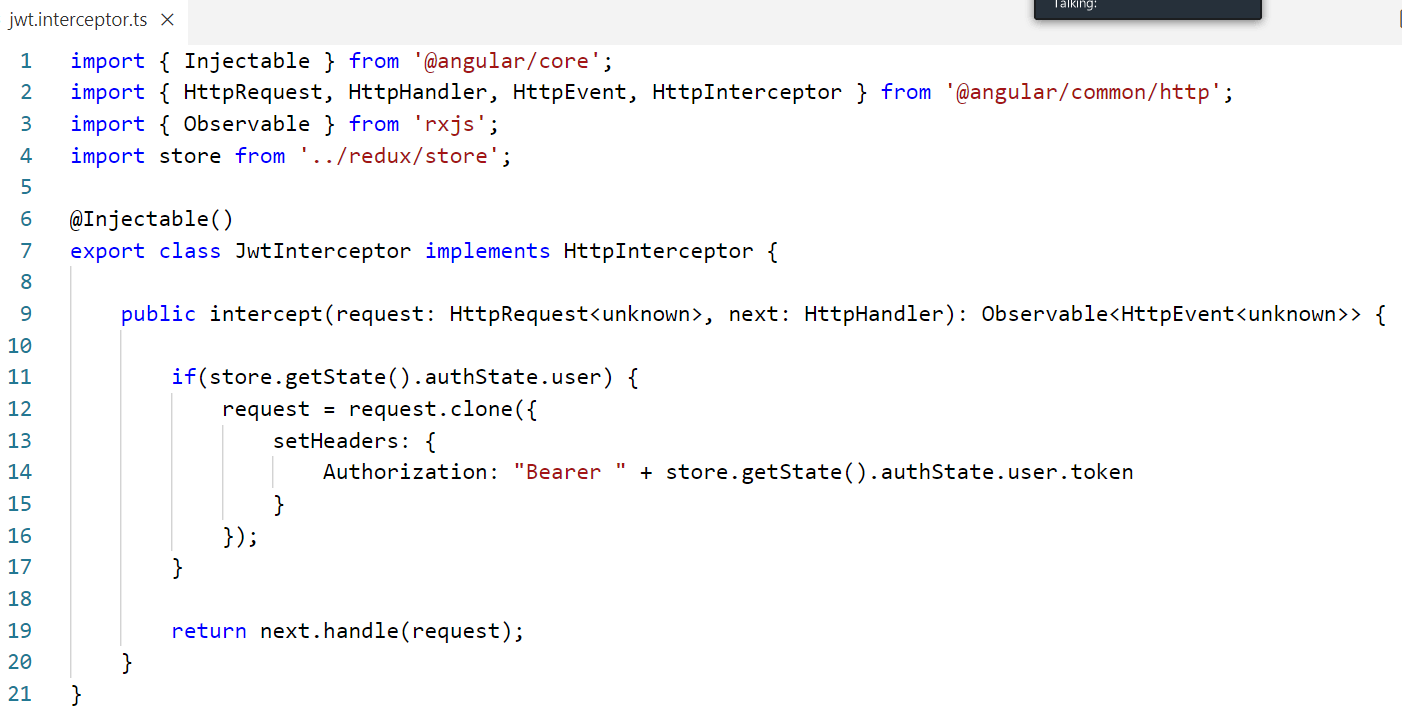
התקנה:

npm i @angular/material @angular/cdk

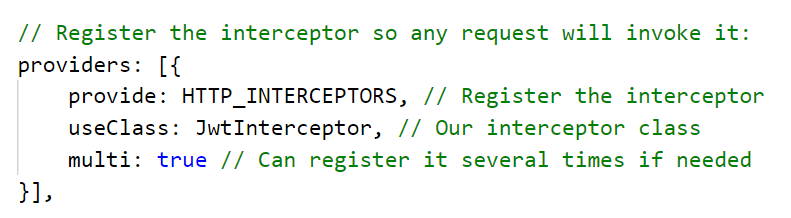
(CDK = Component Development Kit)

**Interceptor**

מתבצע ע"י Service:

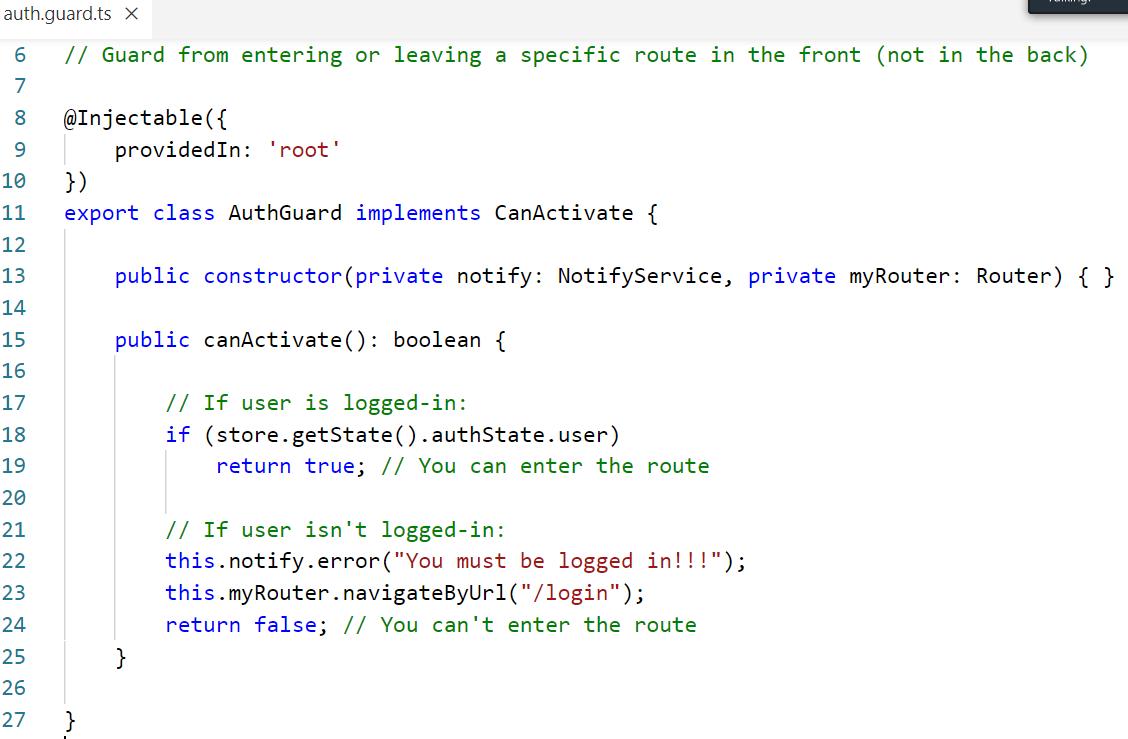


יש לרשום אותו ב-AppModule:



**Route Guard**

זהו Service המאפשר למנוע גישה ל-Route (ב-Front) או למנוע יציאה מ-Route.



יש להגדיר אותו ב-Route המתאים:

