ANGULAR

Angular הוא framework (מסגרת-טכנולוגיה), ספריה של JavaScript

שמשמשת ליצירת אפליקציות מבוססות דף יחיד (Single Page Application) **SPA**

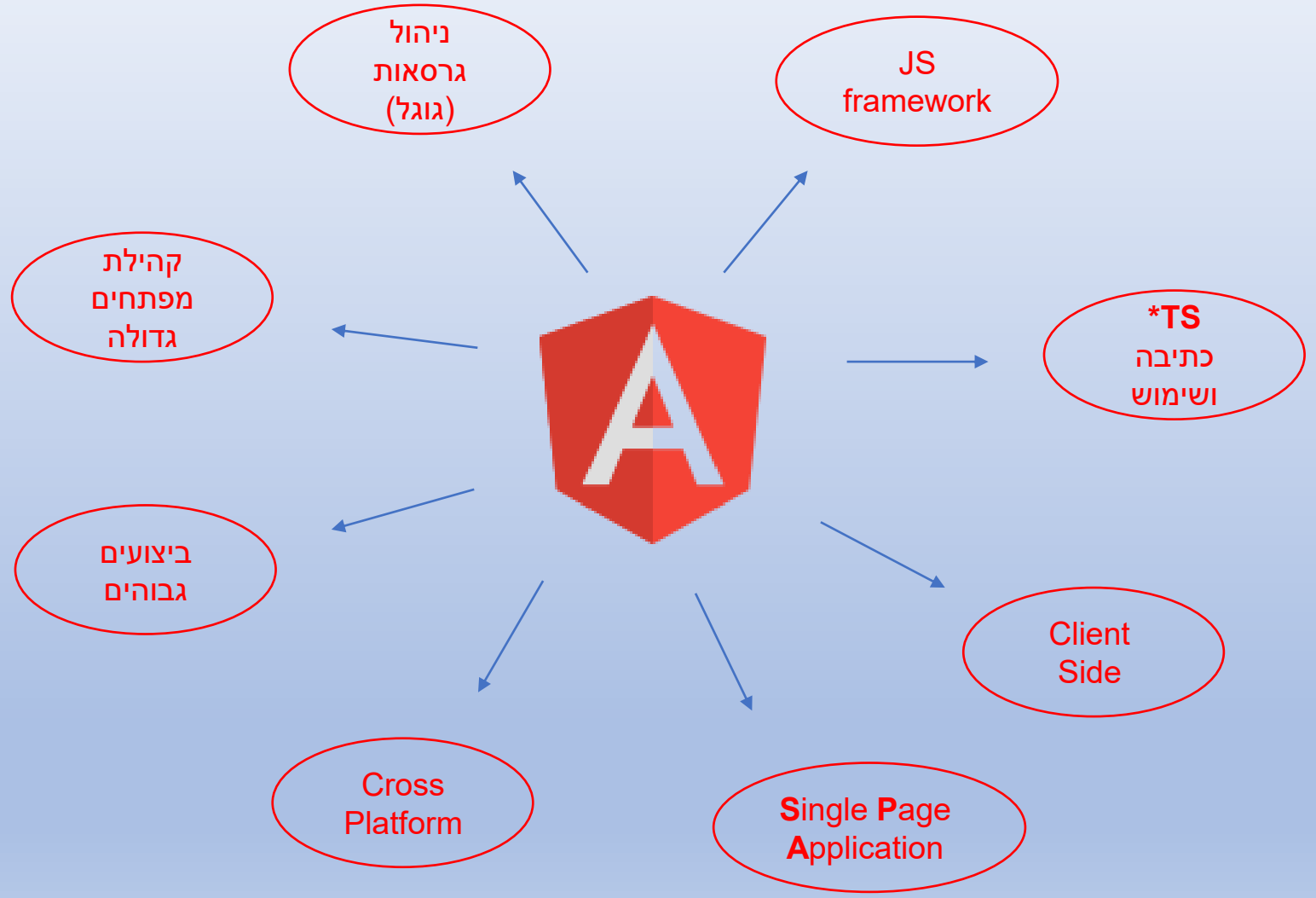
כתוצאה מכך, אתר האינטרנט מציג התנהגות חלקה וחוויית משתמש מעולה.

בשונה מ JavaScript שאנחנו מכירות שעל כל שינוי קטן במסך יטען דף חדש מה שגורם לאיטיות

האפליקציה ולחוסר יעילות!

ב- Angular - כל התוכן נטען על אותו דף - **index.html**

! Angular פותחה ע"י גוגל, ונכתבה בשפת TypeScript (של מייקרוסופט).



TypeScript

TypeScript פותחה ע"י מייקרוסופט

TypeScript היא ערכת העל של JavaScript, שפה המרחיבה את שפת JavaScript.

מכילה את כל הפונקציות, אובייקטים וכו' של בJS ומרחיבה את כל יכולות השפה באמצעות type – ים.

הגדרת משתנים, מחלקות, אובייקטים מסוג מסוים

עוזרת ומשפרת את נקיות הקוד, שגיאות זמן ריצה רבות ופתרון באגים וכו'.

TypeScript היא שפה לכל דבר, ניתן לכתוב OOP : מחלקות, ירושות, ממשקים ועוד

שפת TypeScript אינה נתמכת בכל הדפדפנים ולכן ישנו מנוע הדואג לקמפל את הקוד ולהמירו לשפת JavaScript.

קובץ TypeScript נשמר בקובץ בסיומת **.ts**

התקנה

-Node.js –לצורך העבודה עם אנגולר צריך להתקין על המחשב את node.js

node.js הוא מנוע שיודע להריץ JS על המחשב.

(בעבר קוד JS היה ניתן להרצה רק ע"י הדפדפן. כיום, בעזרת node.js ניתן להריץ קוד הכתוב בשפת JS.

- **N**ode **P**ackage **M**anager – NPM

כדי להתקין ספריות שונות המבוססות על המנוע, כמו אנגולר

ישנו כלי המנהל את כל הספריות האלו : npm מנהל חבילות הקוד של node.js.

* כלי זה כבר כלול בהתקנת node.js , אין צורך להתקין אותו במיוחד.

כל ספריה שנרצה להתקין – נתקין דרך הכלי הזה.

התקנה

אנגולר- נשתמש בסביבת הפיתוח
vs code – חינמית, קלה ונוחה.

פקודת התקנת האנגולר:

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

שימוש ב- npm להתקנת אנגולר.

g- ההתקנה תהיה גלובלית על המחשב (ולא בתיקייה אחת בודדת..)

@angular/cli – התקנת אנגולר בתוספת כלי cli – לצורך עבודה קלה ונוחה.

כלי זה מאפשר הרצת פקודות בcommand line וכך נוצרים קבצים/ קודים באופן אוטומטי וחוסך הרבה מאד זמן עבודה.

כלי זה משמש גם עבור test, build וכו'...

לאחר שהתקנו cli ניתן להשתמש בכל הפקודות שכלי זה יצר עבור אנגולר.

כל הפקודות תהיינה תחת שם ng.

יצירת פרויקט

פקודת יצירת פרויקט אנגולר חדש:

הפקודה נכתבת בטרמינל – פתיחה מקוצרת : ~ + ctrl

```
ng new hello-world
```

הרצת פקודה זו תעלה מספר שאלות לגבי הפרויקט.

- האם להתקין ניווט? כן – נשתמש בהמשך.
- באיזה פורמט stylesheet להשתמש? נשאר עם ברירת המחדל - CSS.

אחרי שה-cli יסיים את התקנת הפרויקט נוכל להיכנס אליו ולהריץ אותו.

כניסה לפרויקט ע"י הפקודה :

```
cd hello-world
```

הרצת פרויקט

הרצת פרויקט אנגולר מתבצעת ע"י הפקודה:

```
ng serve
```

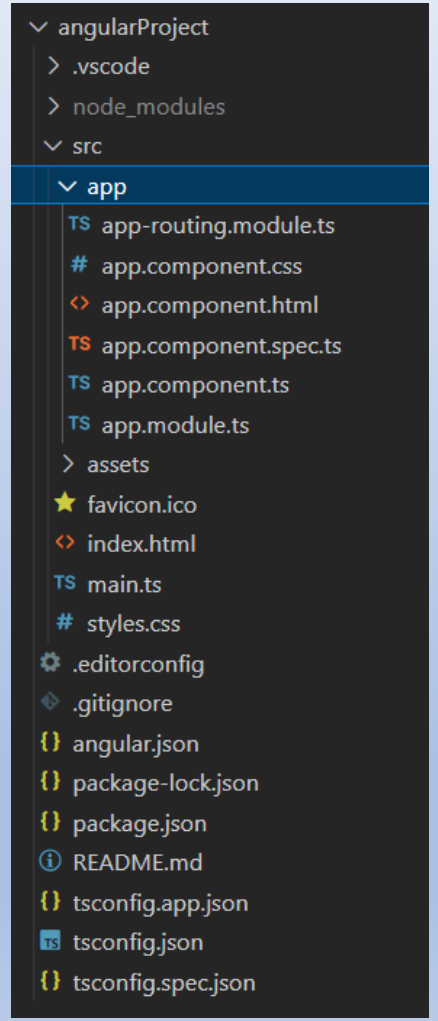
הפקודה serve מריצה את השרת וגם מקמפלת את קבצי ה- TypeScript ל- JavaScript.

דבר מאוד חשוב מפני שדפדפנים יודעים להריץ JavaScript בלבד.

אנגולר רצה על : **localhost:4200**

ניתן להשתמש בקיצור להרצת הפרויקט , ע"י הפקודה: ng s

מבנה הפרויקט



e2e - תיקייה זו מיועדת לבדיקות מקצה לקצה.

Node_modules – תיקייה זו מכילה את כל הספריות שנוריד מה-npm עבור הפרויקט הנוכחי.

src - תיקיית קוד האפליקציה – כל הקוד שנכתוב ונוסיף.

בתוכו קובץ **index.html** – קובץ הראשי, רץ ראשון .

styles.css – קובץ עיצוב המשותף לכלל הפרויקט – גלובלי.

editorconfig. – קובץ המיועד להגדות העבודה IDE , סביבת העבודה.

gitignore. – מיועד לעבודה מול גיט. מוגדר בו ממה להתעלם כאשר מעלים קוד לגיט.

angular.json – קובץ המנהל את כל האפליקציה דרך cli. מכיל בתוכו את הקובץ הראשי שמתניע את המערכת

מוגדר index – הדף שאותו צריך להציג בדפדפן (SP).

package.json – קובץ המנהל את שם הפרויקט וגירסתו, וניהול הספריות וגירסתן בהן הפרויקט תלוי.

מכיל בתוכו –script - כתיבת סקריפטים והרצתם.

- dependencies – ספריות שהשימוש בהן חובה בחיי הפרויקט.

- devdependencies – ספריות שהשימוש בהן בזמן הפיתוח.

package-lock.json – מתארת את הספריות המדויקות בהן הפרויקט עושה שימוש.

tsconfig.json – קובץ המשמש את הקומפיילר של typescript ומנחה לגבי ביצוע הקומפילציה ל –javascript .

מבנה ה-src

כתיבת קוד הפרויקט מתבצעת לתוך ה- src - > app. (קומפוננטות, מחלקות, סרביסים וכו'..)

מקובל – (למען הסדר הטוב) לפתוח תיקייה עבור כל אחד מהסוגים.

שם התיקייה – אותיות אנגלית קטנות.

components

Services

Classes

כאשר נרצה לפתוח קובץ – נפתח אותו בניתוב המדויק = תחת התיקייה המתאימה לו.

!! ניתן ללחוץ לחיצה ימנית על התיקייה הרצויה-> open in integrated terminal

קומפוננטת app – קומפוננטה ראשית
נוצרת בעת יצירת הפרויקט
עליה נניח קומפוננטות נוספות

יצירת קומפוננטה

יצירת קומפוננטה מתבצעת ע"י הפקודה :

Ng generate component componentName

או בקיצור : **ng g c componentName**

!! שם הקומפוננטה באותיות קטנות.

!! אנגולר מוסיף אוטומטית סיומת component – אין צורך להוסיף.

!! הוספת אות גדולה – תיצור מקף בשם הקומפוננטה.

יצירת קומפוננטה

קומפוננטה מכילה 4 קבצים:

componentName.css – קובץ המכיל את עיצוב הקומפוננטה.

componentName.html – קובץ המיכל את תצוגת הקומפוננטה.

componentName.spec.ts – קובץ המיועד לבדיקות (לא מתעסקים איתו- אפשר למחוק).

componentName.ts – קובץ המכיל את הגדרות, משתנים, פונקציות, לוגיקות וחישובים של הקופמוננטה.

! בעת יצירת הקומפוננטה קובץ ה-app,module מתעדכן אוטומטית.

קובץ app.module

קובץ ה-app.module הוא כלי המאגד בתוכו אוסף של התנהגויות ויכולות הקשורות לנושא אחד.

מודול נכתב בשפת ts ומוצהר ע"י @NgModule – הצהרה המסמנת לאנגולר את המחלקה כמנהלת את האוסף.

רכיבי המודול

Import – יבוא קבצים חיצוניים

@NgModule הצהרת המחלקה כמודול

Declerations – הצהרות על שימוש במחלקות התצוגה(כל קומפוננטה שנוסיף – נוודא שמופיעה בdeclaration.

Imports– ייבוא מחלקות מודולים חיצוניים.

Providers – הגדרת ספקי השירותים(סרביסים)

Bootstrap- מופיע רק במודול הראשי – מתניעה את פעילות האפליקציה.

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomepageComponent } from './components/homepage/homepage.component';

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

decorator =@

יצירת קומפוננטה

בעת יצירת הקומפוננטה – נוצר קובץ

componentName.spec.ts – קובץ זה מיותר – כי מיועד לבדיקות.

ניתן לשנות בהגדרות יצירת קומפוננטה בקובץ angular.json

תחת schematics :

את skipTests : true

ובנוסף ניתן להתערב ב style וב-profix .

!

או בפקודת יצירת הקומפוננטה להוסיף : ng g c componentName **–skipTest true**

מבנה קומפוננטה

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

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent {

}
```

@component = Decorator מצהיר על מחלקה זו = קומפוננטה.

Metadata – הנתונים בתוך }(

Selector – שם הקומפוננטה (בעת שימוש בקומפוננטה – נקרא בשם זה)

templateURL – הנתיב בו נשמר קובץ ה-html של הקומפוננטה.

styleURLs – הנתיב בו נשמר קובץ ה-css של הקומפוננטה.

})

!! ניתן להוסיף קוד במקום הניתוב לקבצים.

!! ישנם עוד הגדרות שנלמד בהמשך כמו : providers , encapsulation.

חיבור בין קבצי הקומפוננטה נעשה באמצעות **databinding** – מנגנון אנגולרי מיוחד המקשר בין החלק הלוגי לחלק הויזואלי.

תרגיל

פתחי קומפוננטה firstName

הציגי בתוכה תגית ושדה קלט.

פתחי קומפוננטה lastName

הציגי בתוכה תגית ושדה קלט.

הציגי את 2 הקומפוננטות על הקומפוננטה הראשית.

הוסיפי כפתור שליחה.

עצבי...

binding

Data binding

מנגנון אנגולרי מיוחד המקשר בין החלק הלוגי לחלק הויזואלי.

String binding

הצגת משתנים ע"י {{ name }} -

צומדיים מסולסלות מכילות string בלבד , ערך בוליאני מבצע המרה, יכול להכיל משתנים ופונקציות.

Property binding

שינוי ערכי ה-property בהתאם לנתונים הנמצאים במחלקה.

```
<button [disabled]="isTrue">button</button>
```

צומדיים מרובעות

Event binding (ארועים)

```
<button (click)="function()">button</button>
```

סוגריים עגולים

Two way binding

[(ngModel)] – נלמד בהמשך...

שימוש בקובץ ts.

כאשר נגדיר משתנים בקובץ ts

מצטרך להכריז את טיפוס המשתנה – את ה-type שלו.

propertyName : propertyType = value

סוגי המשתנים הם:

String –מחרוזות.

Number –מספרים.

Array –מערכים.

Boolean –ערך בוליאני true/false .

Date –תאריך.

Any – כאשר לא ידוע סוג המשתנה.

נגדיר את המשתנים בתוך המחלקה של הקומפוננטה בה נרצה להשתמש בהם.

גישה למשתנים בתוך קומפוננטה בעזרת המילה this.

הגדרת משתנה בתוך פונקציה -שימוש בlet/var .

ביטול חיוב אתחול משתנים בקובץ tsconfig.json תחת angularCompilerOptions

```
"strictPropertyInitialization": false
```

מחלקה - class

ניצור מחלקות עבור מאפיינים מורכבים, ישויות וכדו'..

בקומפוננטה נגדיר מאפיין מסוג המחלקה.

את המחלקה ניצור בקובץ ts .

נגדיר אותה כ export כדי שנכיר אותה בקבצים חיצוניים.

נכתוב את המאפיינים בתוך סוגריים של בנאי –

צורת כתיבה זו מצהירה על הבנאי שמקבל את הפרמטרים מסוגם ומאתחל אותם ביצירת מופע ממחלקה זו.

```
export class Car {
    constructor(public categoryCode:number,
                public companyName: string,
                public isNew : boolean,
                public testing?:Array<Date>  )
                {   }
}
```

בעת כתיבת מאפיין נוסיף לו הרשאת public/ private.

ניתן להוסיף למאפיין האם הנתון הוא חובה/לא :

ע"י הוספת ? למשתנה – הופך אותו ללא חובה.

משתנים שהם לא חובה- יופיעו בסוף.

! ניתן לממש ירושות.

תרגיל

מחלקת אדם שתכיל :

שם פרטי, שם משפחה, גבר-ערך בוליאני, כתובת, פלאפון.

הגדירי אדם בקומפוננטה הראשית.

הציגי את פרטיו.

הוסיפי כפתור שיהיה ניתן ללחוץ עליו כאשר האדם = אישה,

בלחיצה כתבי "שעשני כרצונו"

תנאי *nglf

*nglf הוא directive של אנגולר (נרחיב בהמשך) המאפשר לבצע הוספה והסרה של אלמנט ע"פ תנאי.

במקרה שנרצה להוסיף תנאי לתגית – האם תוצג או לא.

נוסיף לתגית את התנאי *nglf="“

התנאי ייכתב בתוך " "

התנאי יכול להיות ביטוי כלשהו...

! אנגולר לא יסתיר / יציג את האלמנט אלא יוסיף / יסיר מה- Dom.

לולאת *ngFor

*ngFor הוא directive של אנגולר (נרחיב בהמשך) המאפשר לבצע לולאה- שתעבור על מספר איברים,

ובכל איטרציה תציג את האיבר הנוכחי.

```
<div *ngFor="let c of Cars">
    {{ c.categoryCode}}
</div>
```

c – משתנה שנוצר בזמן ריצה ולכן ניגש אליו כך: `{{c.categoryCode}}`

צורת המעבר היא "`let c of Cars`" כאשר c הוא מילה נבחרת.

ngStyle

ngStyle הוא directive של אנגולר (נרחיב בהמשך) המאפשר לבצע עיצוב מותנה על אלמנט,

האלמנט עליו הוא יוגדר יקבל הגדרות style לפי הביטויים שנרשום

```
<p [ngStyle]="{'background': car.categoryCode == 3 ? 'green' : 'red'}">Hello World</p>
```

ngSwitch – ngSwitchCase - ngSwitchDefault

שלושת מאפיינים אלה משמשים יחד כדי ליצור תנאי switch על הצגת אלמנטים לפי ביטוי שנכתוב.

ngSwitch – מגדיר את המשתנה עליו יכול החישוב.

ngSwitchCase – מציג אפשרויות.

ngSwitchDefault – ערך ברירת מחדל ,

אם אין שום ngSwitchCase – מקרה, ייכנס להציג את התוכן ב ngSwitchDefault

```
<div [ngSwitch]="country">
    <p *ngSwitchCase="'israel'">Made in israel</p>
    <p *ngSwitchCase="'china'">Made in china</p>
    <p *ngSwitchDefault>Made in belgium</p>
</div>
```

Life cycle hooks

מחזור חיי הקומפוננטה מורכב מכמה שלבים:

הקומפוננטה נוצרת –ngOnInit

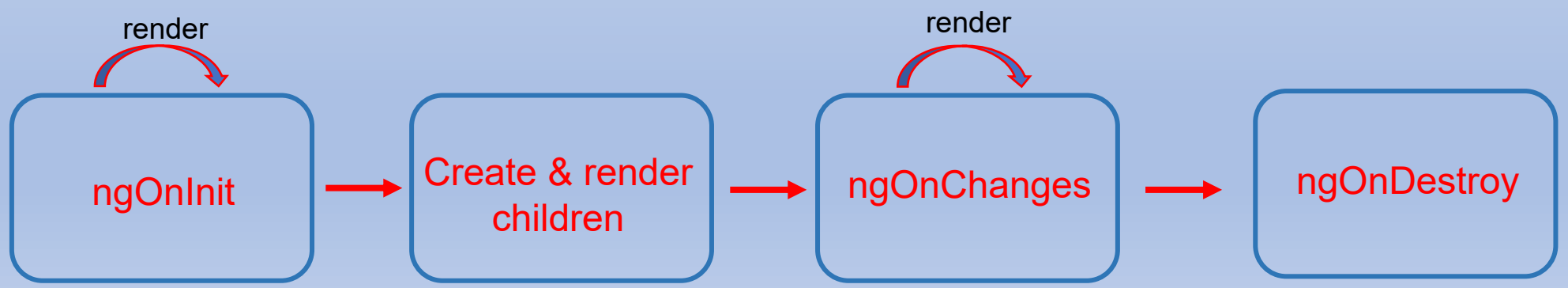
מתרנדרת ngOnChanges

יוצרת את הילדים תחתיה ומרנדרת אותם.

מתרנדרת בכל שינוי.

הריסת הקומפוננטה והסרתה מה DOM - ngOnDestroy

!! ישנם עוד hooks ביניהם – אפשר לקרוא עליהם...



ngOnInit

פונקציית ngOnInit היא hook – אחת ממחזור החיים של הקומפוננטה (נרחיב בהמשך)

פונקציית ngOnInit מתרחשת בעת טעינת הקומפוננטה.

נשתמש בה לאתחול נתונים, קריאות שרת מיד בהתחלה, או כל פונקציה שנרצה שתקרה מיד בטעינה.

כדי להשתמש ב ngOnInit יש לממש את הממשק OnInit ע"י הוספת : implements OnInit

והוספה ל- import .

הממשק חייב לממש את הפונקציה ngOnInit()

```
import { Component, OnInit } from '@angular/core'
export class HomepageComponent implements OnInit {
  ngOnInit(): void { }
```

תרגיל

הגדירי מחלקת פלאפונים - לכל פלאפון יש לשמור את התכונות הבאות:

קוד-חובה, שם- חובה, בלוטוס-חובה, כשר-חובה, תמונה לא חובה.

צרי קומפוננטת פלאפונים

הגדירי בקומפוננטה רשימה מסוג פלאפון.

מלאי את הרשימה בעת טעינת הקומפוננטה.

הציגי בקומפוננטה את כל הפלאפונים בטבלה.

אם הפלאפון כשר – הציגי צבע רקע ירוק, אם לא – הציגי צבע רקע אדום.

אם יש תמונה הציגי תמונה אם אין תמונה כתבי אין כרגע תמונה.

אפשרי מחיקה של פלאפון מהטבלה - מהרשימה.

הוסיפי כפתור "ראיתי" שיהיה מאופשר רק במקרה שיש יותר מ-5 פלאפונים ברשימה,

בלחיצה על הכפתור יוצג alert כלשהו...

הוסיפי תכונת ארץ יצור במחלקה. יופיע למטה טקסט עבור הפלאפון האחרון ברשימה "Made in _" (שימוש ב ngSwitch)

בהצלחה!!

* * * * *

Template reference variable

כאשר נרצה לבצע גישה לאלמנט, ולהעביר נתונים לקומפוננטה –

נוכל להגדיר משתנה בתוך ה-template ולהעביר אותו לקומפוננטה -Template reference variable

לדוגמא:

כאשר נרצה למלא איזשהו שדה ולשלוח את הערך לקומפוננטה נוכל לבצע:

נוסיף משתנה לתגית ע"י #variable ונשלח אותו לקומפוננטה.

```
<input type="text" #box>
<button (click)="Changes(box.value)">OK</button>
```

```
<input type="text" #box (keyup)="Changes(box.value)">
<p>{{values}}</p>
```

[(ngModel)]

html ⇔ ts

כאשר יש צורך לקשר בין שדה קלט למשתנה ובין המשתנה לשדה הקלט. Two way binding

לדוגמא,

נרצה להכניס קלט לתיבת קלט ובעת ההקלדה יופיע שינוי הקלט בטקסט,

כלומר הערך שנכניס בכל הקלדה ישמר במשתנה וכל שינוי של המשתנה יופיע בטקסט.

קישור דו צדדי מתבצע ע"י ngModel

- ייבוא FormsModule למערך ה- imports בקובץ app.module - מאפשר זרימה דו-כיוונית
- הגדרת משתנה בקומפוננטה בקובץ ה-ts.
- הוספת מאפיין [(ngModel)] לתגית הקלט.

מאפיין זה מופיע ושימושי בטפסים.

העברת נתונים @Input

כל קומפוננטה חיה בפני עצמה ואין לה קשר לקומפוננטות אחרות.

איך נוכל להעביר נתונים ביניהם? (אחד הדרכים)

כאשר יש צורך להעביר נתונים בין קומפוננטות / כאשר יש צורך לקבל ערך- משתנה- פרמטר מקומפוננטה חיצונית

נוכל להגדירה כ-input.

בקומפוננטה המקבלת נגדיר משתנה מסוג ()input@ (להוסיפו ב import)

```
@Input() newCar: Car
```

כאשר נשתמש בקומפוננטה זו נשלח אליה כפרמטר את הערך הרצוי.

```
<app-child [newCar]="car"></app-child>
```

```
<p>{{ newCar.categoryCode }}</p>
<p>{{ newCar.companyName }}</p>
<p>{{ newCar.isNew }}</p>
<p>{{ newCar.testing }}</p>
```

תרגיל

- צרי מחלקת אדם : קוד, שם משפחה, שם פרטי, גיל, פלאפון, רשימת שמות ילדים.
- צרי קומפוננטת **עובדים**.
- צרי רשימת עובדים והציגי אותה (בטבלה/ דרך רצויה) הציגי בטבלה את שדות : קוד, שם משפחה וגיל.
- צרי כפתור – בלחיצה עליו יפתח טופס הוספת עובד. (שדות קלט להכנסת פרטים-השתמשי ב ngModel וב- reference template).
- בלחיצה על שמור יתוסף העובד לרשימה ויוצג מייד בטבלת העובדים.
- צרי קומפוננטת **עובד.**
- בלחיצה על עובד מסוים בטבלה (בקומפוננטת עובדים) תפתח קומפוננטת עובד עם פרטיו המלאים: קוד, שם משפחה, שם פרטי, גיל, פלאפון, רשימת שמות ילדים.
- לכל עובד – אם יש ילדים הציגי את שמותיהם ברשימה נגללת/ ברשימה, אחרת תוצג ההודעה "אין ילדים☹"

בהצלחה!!

העברת נתונים @output

ארועים:

אחד הדברים המיוחדים באנגולר הוא יצירת ארועים בעצמינו (בדומה לאירועים מובנים כמו לחיצה וכו')

אירועים – פלט כלשהו מהקומפוננטה לחוץ, מי שנרשם לאירוע מקבל אליו נתונים הקשורים לאירוע והוא כבר ישתמש בהם.

הקומפוננטה פולטת החוצה נתונים ע"י הרשמה – מי שנרשם לאירוע יכול לקבל את הנתונים כ-event .

app-child

```
counter:number = 0;
@Output() onOut:EventEmitter<number> = new EventEmitter<number>();

add(){
  this.counter++;
  this.onOut.emit(this.counter);
}
```

```
<button (click)="add()">add</button>
```

app-parent

```
output(e:any){
  this.number= e;
}
```

```
<app-child (onOut) = "output($event)"></app-child>
 <p>{{ number }}</p>
```

3 שלבים ליצירה ושימוש ב output:

1. יצירת משתנה מסוג eventEmitter – מפעיל ארועים, והגדרות ב- output().
2. הפעלת האירוע ע"י פקודת emit()
3. רישום לאירוע ע"י הקומפוננטה המארחת.

תרגיל

שדרגי את התרגיל הקודם:

- הוצאת טופס הוספת עובד מקומפוננטת עובדים לקומפוננטת הוספת עובד.
- בלחיצה על כפתור הוספת עובד – תפתח קומפוננטת הוספת עובד
- הטופס יחיל שדות קלט להכנסת פרטי עובד חדש.
- בלחיצה על שמירה – העובד החדש יוצג בקומפונטטת עובדים.

! השתמשי ב @output

! שימי לב לאפס את תיבות הקלט אחרי ההוספה

בהצלחה!!

* * * * *