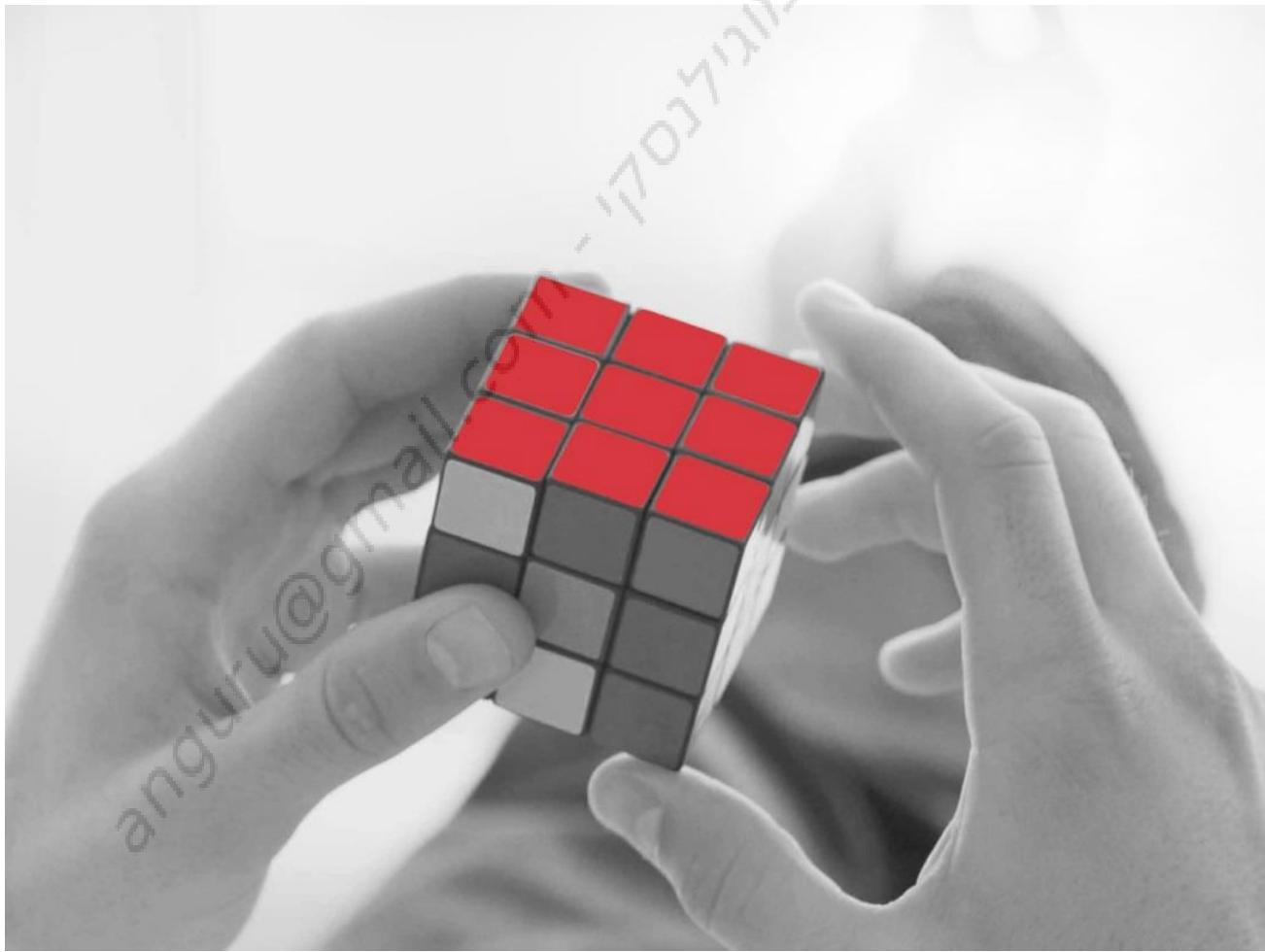


Angular 8

Angular



תוכן עניינים

5.....	1. הקדמה.....
5.....	1.1. הצורך בתשתיות SPA.....
5.....	1.2. יתרונות SPA.....
5.....	1.3. Angular.....
6.....	1.4. גרסאות Angular.....
7.....	1.5. יתרונות Angular.....
7.....	1.6. Component – Based User Interfaces
8.....	2. יצרת פרויקט בסיסי ב-Angular.....
8.....	2.1. יצרת פרויקט – Angular – שימוש ב- CLI
17.....	2.2. השימושים ב-CLI.....Angular-CLI.....
18.....	2.3. ארכיטקטורת Angular.....Angular
18.....	2.3.1. NgModule
18.....	2.3.2. Root Module
19.....	2.3.3. Components
20.....	2.3.4. Directives
20.....	2.3.5. Services
20.....	2.3.6. Dependency injection
22.....	3. Components.....Components
22.....	3.1. מבט עמוק – Components.....Components
22.....	3.1.1. Architecting with Components.....Components
24.....	3.1.2. קובץ app.component.ts
24.....	3.1.3. Interpolation
25.....	3.1.4. הגדרת תבנית html ועיצוב css לקומפוננטה
26.....	3.1.5. הגדרת קישור חיצוני ל-html ו�-css
27.....	3.1.6. הגדרת תוכן ישיר ל-html ו�-css
28.....	3.2. יצרת Component חדש
37.....	3.3. Debugging with Augury

40	View Encapsulation .3.4
45	Angular Bindings .4
45	Template Syntax .4.1
46	One-way Binding – Interpolation .4.1.1
47	Binding One-way: from View Target to Data Source .4.1.2
48	Two-Way Binding .4.1.3
49	Template Reference Variables .4.2
52	Angular Event Cycle Hooks .5
52	Directives and Components LifeCycle hooks .5.1
53	Angular Directives .6
53	Built in Attribute directives .6.1
53	Built in Structural Directives .6.2
54	Nglf .6.2.1
58	NgSwitch .6.2.2
59	NgFor .6.2.3
61	NgForTrackBy .6.2.4
63	תרגילים .6.3
63	תרגיל 1
63	תרגיל 2
65	תרגיל 3
68	Angular Pipes .7
68	The Pipe Operator () .7.1
70	Custom Pipe .7.2
73	Components Communication .8
73	@Input and @Output .8.1
75	תרגיל
76	Angular Services .9
76	Component Constructor .9.1
76	Dependency Injection .9.1.1

77	The Injector Tree .9.1.2
78	Services .9.2
84	Promises .9.3
87	Async – Observables .9.4
89	Angular Routing .10
89	Routing .10.1
91	Router-Outlet .10.2
93	Nested Routing .10.3
96	Router Lifecycle Hooks .10.4
97	Angular Forms .11
97	Angular Forms .11.1
97	Angular template driven Forms .11.2
101	Angular model driven Forms .11.3
102	FormControl-1 FormGroup .11.3.1
103	Validations .11.3.2
104	Form valid/dirty .11.3.3
106	Angular async Forms .11.4
108	תרגיל
110	Angular onPush .12
110	Change Detection in Angular .12.1
110	Change Detectors .12.2

1. הקדמה

1.1. הצורך בתשתיות SPA

.Single-Page Application – SPA

ישומי דף ייחיד הם יישומי רשת, שтратרתם לחת חווית משתמש מהירה וזרמת יותר, הדומה לתוכנת מחשב רגילה (שaina יישום רשת). בישומים אלו, כל הקוד הדרוש (HTML, CSS, JS) מגע לדפדף בטיענית דף אחד, ומשארים נוספים נוטעים באופן דינמי, בדרך כלל, כתגובה לפעולות המשתמש. דף האינטרנט לא מבצע טענה מחדש מחדש, אולם כתובת האינטרנט עשויה להשתנות מעט, על מנת לתת למשתמש הבנה יותר טובה של הניוט בדף.

SPA הוא ההתקפות הטכנולוגיות של multi-page application ושימוש ב-AJAX.

1.2. יתרונות SPA

- זמן טעינה מהיר יותר של דף.
- טעינה ברקע לצד השרת, מגדילה את חווית המשתמש.
- אין צורך לרענן דפים בשרת בכל פעולה משתמש.
- אין תלות חזקה בין הצד השרת לצד הלוקה.
- הפיתוח תואם לעולם אפליקציות המובייל, משומן שכןון להשתמש באותו שרת לאפליקציית מובייל שונה (מסך קטן יותר, וכו').

Angular .1.3

בפיתוח תוכנה Angular היא תשתיית תוכנה (framework) ליישומי רשת. התשתיית זו הינה קוד פתוח המבוסס על TypeScript, והיא מתוחזקת על ידי Google ועל ידי קהילה רחבה של מפתחים.

התשתיית, מיועדת לפתרון אתגרים בפיתוח יישומי דף-יחיד, ופשטת הפיתוח והבדיקות של יישומים אלו, באמצעות תשתיית תוכנה לארכיטקטורות צד לקוח כמו MVC או MVVM יחד עם RECEIPTS, בהם משתמשים, בדרך כלל, בישומי אינטרנט עשירים.

Angular היא למעשה הדור הבא של JS angular. ולפי צוות הפיתוח של angular, הונגה שימוש במונח AngularJS לגרסהות X וAngular ללא "JS" לגרסהות 2 ומעלה.

1.4. גרסאות Angular

2.0.0

הגרסה הסופית של 2 Angular פורסמה ב- 14 בספטמבר 2016.

3.0.0

הגרסה זו לא הופצה על מנת למנוע הבלבול עקב חוסר ההיערכות של `version` package's router.

4.0.0

ב- 13 בדצמבר, 2016 הוצאה לראשונה לאור גרסת 4 Angular, והגרסה הסופית פורסמה ב- 23 במרץ 2017.

Angular 4 תומך לאחרור ב-2 Angular 4.

גרסה 4.3 היא גרסה משנה, והשיפורים העיקריים שלה הם:

- HttpClient - ספירה בעל נפח קטן וקלת לשימוש עבור ייצור בקשות HTTP.
- ארבעה אירועים חדשים בדיספנסר生活 cycle events GuardsCheckStart, :router life cycle events GuardsCheckEnd, ResolveStart, ResolveEnd
- הוסף event GuardsCheckEnd, ResolveStart, ResolveEnd NavigationStart. הקיימים כגן Conditionally disable animations

5.0.0

Angular 5 פורסם ב- 1 בנובמבר 2017. והשיפורים העיקריים שלה הם:

- Support for progressive Web apps
- Build optimizer
- Improvements related to Material Design

6.0.0

Angular 6 פורסם באפריל 2018. בגרסה זו הוסיףו ושינו מספר נושאים:

- update - יכולת לבצע שידרוג גירסאות אנגולר בצורה קלה.
- RxJS V6 - עדכון את הספריות החזקות והשימושיות בעולם האנגולרי.
- Angular-cli - שדרוג היכולת של פקודות cli.

7.0.0

גרסה 7 של אנגולר יצאה ב-20 לאוקטובר. רוב החידושים הם "מאחורי הקלעים".

- TypeScript 3.1 support
- Angular elements
- Slots with Angular elements

גרסת 8.0.0

גרסת 8 של אングולר יצאła ב- 28 במאי 19. גם כאן רוב החידושים הם "מאחוריו הקלעים".

- Dynamic imports for lazy routes
- CLI workflow improvements
- TypeScript 3.4 support
- New features for ngUpgrade

1.5. יתרונות Angular

- שפת Angular - מודרנית ובעל יכולות פיתוח גבוהות יותר, בשילוב מובנה עם type script. זו שפה מודולרית וקלת ללמידה (ב相较ה ל-x 1.0). Angular.
- ארכיטקטורת Angular - הינה מודולרית עם תלות פחותה בין האובייקטים (ב相较ה ל-x 1.0). יכולה זו מובילה לפיתוח נכון ארכיטקטוני, שחוסך תקלות.
- ביצועים ב- Angular - טובים יותר. היא הופכת תבניות לקוד. תבניות אלו עוברות אופטימיזציה, המשפרת (מבחינת ביצועים) שימושית את קוד Java script המתקבל.

Component – Based User Interfaces .1.6

בפרק זה נסקור כיצד ממשקי משתמש components עוזרים לנו לבנות בקלות יישומים גדולים יותר. ומדובר השימוש ב components נפוץ כל כך עם web frameworks שונים בכלל, ו- Angular.

כימ – ממשקי המשתמש בדף אין מרכיבים רק html elements בסיסיים, אלא ממשקים מודרניים משלבים עיצוב חזותי חדשני של תוכן אינטראקטיבי יותר מתמיד. אולם, אנו עדין נוטים לחשב על הארכיטקטורה של יישומי האינטרנט שלנו כאוסף של דפים, למרות המעבר לשימוש בapps – המרכיב מדף אחד בלבד.

"We're not designing pages, we're designing systems of components."

- Stephen Hay -

הציטוט הבא מביא אותנו לנקודת שינוי מעיצוב הממשק ע"י חלוקה לפי דפים. ולהבנה שכדי ליצור ממשק משתמש יעיל גם למשתמשים, וגם למתקנים האחראים לתחזוקת הקוד, יש לתכנן מערכת של רכיבים.

ואכן השימוש של Angular ב-component-style directives משנה את כל ה-workflow בתכנון ועיצוב אתרי אינטרנט. הוא מאפשר להגדיל את השימוש החזר בקטעי קוד או בלוקים של html שנכתבו על ידי צוות הפיתוח, רכיבים משותפים (components) במערכת, כאשר כל רכיב הוא עצמאי, אבל יכול לתקשר עם רכיבים אחרים וליצור רכיב גדול יותר על ידי הכליה פנימית של רכיבים אחרים.

המרכיבים הבסיסיים של components:

1. **Encapsulation – הכלוסה** – Encapsulation פירושו שילוב לוגיקה ו-data יחד בתחום container בלבד. כאשר encapsulation נכוון מסוים לנו בארגון הקוד שלנו. ללא צורך לזכור ולהבין לעומק את כל ה-logic internal של המכלול הסגורה ואת לוגיקת האלידציות המורכבת שיצרנו עבור הנתונים. ובכך לעבד ברמה גבוהה יותר הפешטה (higher-abstraction level).

ב-components השימוש ב-Encapsulation מכוון ליצירת רכיבים קטנים ותמציתיים, שבסופו של דבר ישולבו למערכת של רכיבים. ובכך מותאפשרת במהלך הפיתוח, היכולת להתמקד בתוכן ובלוגיקה הפנימית של רכיב אחד בלבד, ואספקת גישה לקוד ספציפי בקלות, תוך כדי יצירת האפשרות להתמקד בשכבה אחת של הקוד, ולבטוח ב-underlying implementations implementations המומסים בתוך הרכיבים.

2. **Composition - הכללה** – כל component הוא רכיב שהוא עצמאי, אבל יכול לתקשר עם רכיבים אחרים וליצור רכיב גדול יותר על ידי הכללה הפנימית של רכיבים אחרים.

2. ייצרת פרויקט בסיסי ב-Angular

2.1. ייצרת פרויקט Angular – שימוש ב-CLI

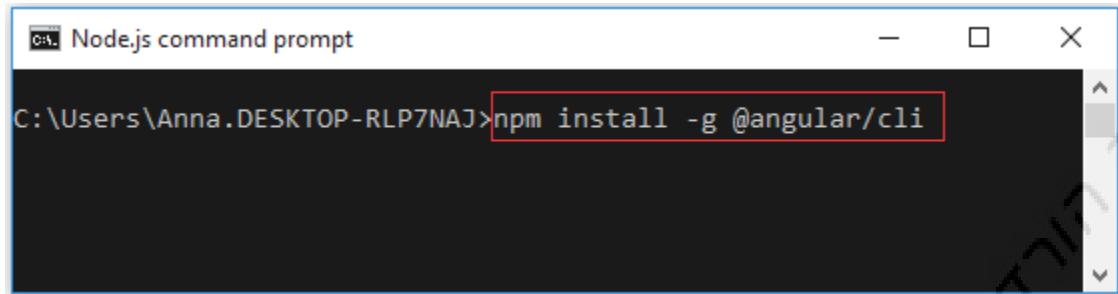
יש מספר דרכים לייצר פרויקט בעולם האנגלי. הדרך המומלצת שתיחסור לנו לא מעט זמן היא שימוש בCLI (Command Line Interface), שבעזרתו גם ניציר את הפרויקט עצמו וגם נרחיב אותו בעtid. על מנת להתקין את הכללי יש להתקין קודם node ו-npm.

Node ניתן להתקין מהלינק הבא: <https://nodejs.org/en/download>

לאחר ההתקנה ניתן להשתמש ב-node על מנת להתקין את רכיב ה-CLI הייעוד.

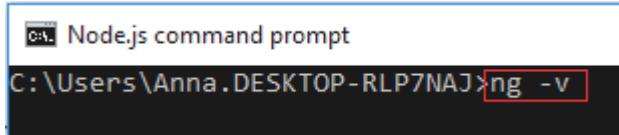
node הוא כלי להתקנת Packages מאוד נוח לשימוש. נתקין בעזרתו בעתיד גם ספריות עזר לפרויקט.

כעת נתקין את כל ה CLI בرمאה כללית על תחנה עלייה אנחנו עבדים (שימוש בפורמט g – כוונתו global)



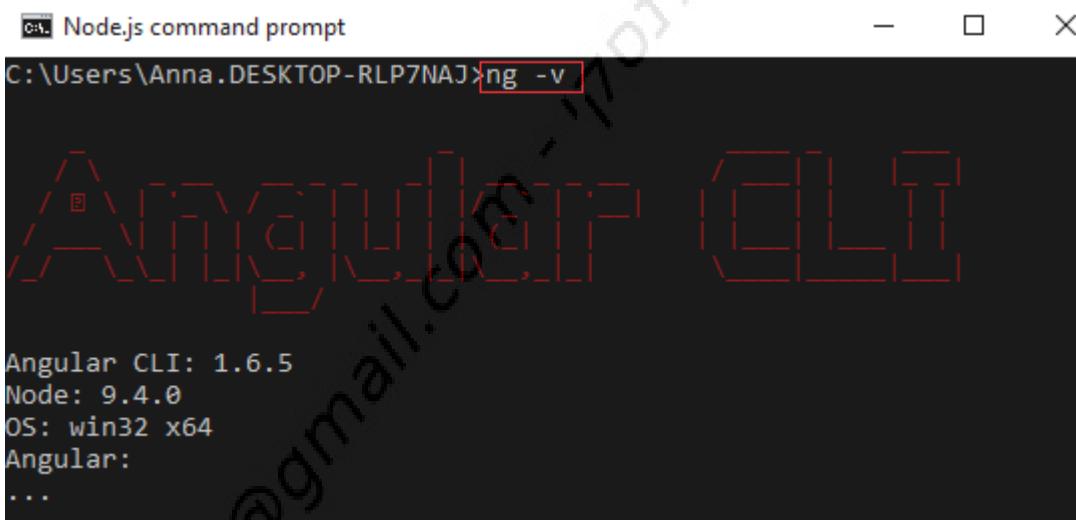
```
Node.js command prompt
C:\Users\Anna\Desktop-RLP7NAJ>npm install -g @angular/cli
```

אם הנ"ל רץ בהצלחה נוכל מעכשו להשתמש בפקודות עם הקידומת `ng` , ופקודות אלו יבוצעו ע"י `Angular CLI`.
בכדי לבדוק אם התקינה בוצעה כראוי נריץ את הפקודה:



```
Node.js command prompt
C:\Users\Anna\Desktop-RLP7NAJ>ng -v
```

ונקבל את הפלט הבא, המאשר שהתקנת `Angular CLI` בוצעה בהצלחה:



```
Node.js command prompt
C:\Users\Anna\Desktop-RLP7NAJ>ng -v
Angular CLI: 1.6.5
Node: 9.4.0
OS: win32 x64
Angular:
```

כעת, לאחר שוויידammo ש-CLI, מותקן בצוורה תקינה ברמה גלובלית, נוכל ליצור פרויקט אングולר חדש באמצעות הפקודה `ng new` או היוצר פרויקט אングולר בסיסי בתיקיה הנוכחית בה הוריצה הפקודה `ng`-CLI.
ולכן – לפני שניצור תיקייה פרויקט חדשה, נעבור לנطיב הרצוי ליצור פרויקט, ע"י הפקודה `cd` (שפירושה:

```
C:\Users\Anna\Desktop-RLP7NAJ> נתיב לפני שינוי  
C:\Users\Anna\Desktop-RLP7NAJ> cd desktop  
נתיב לאחר שינוי -> C:\Users\Anna\Desktop-RLP7NAJ\Desktop
```

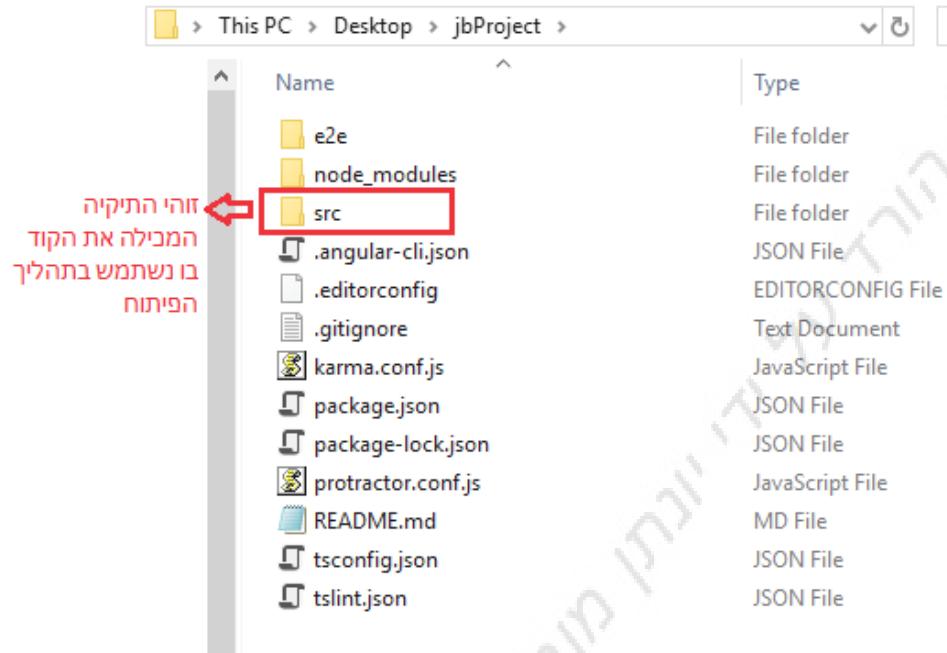
בשלב זה, אנו נמצאים בנתיב הרצוי, ונוכל לאותחל בו פרויקט אングולר חדש באמצעות הפקודה הבאה:

```
C:\Users\Anna\Desktop-RLP7NAJ\Desktop> ng new jbProject  
פוקודה מובנית  
בנוסף לשם הפרויקט (לפי בחירת המפתח)  
המשמשת  
ליצירת פרויקט חדש
```

פקודה זו עשויה להימשך זמן מה, ותסתיים רק כאשר נראה שהיא-CLI סימן את האיתחול ופנו להרצת פקודות אחרות:

```
Node.js command prompt  
added 1457 packages in 415.401s  
Installed packages for tooling via npm.  
Successfully initialized git.  
Project 'jbProject' successfully created.  
C:\Users\Anna\Desktop-RLP7NAJ\Desktop>
```

הפקודה יוצרת תיקיה ראשית בשם `jb`, ובתוכו אותה תיקיה נוכל למצוא קובץ בשם `package.json`. בתוכו ישנה רשימה של כל dependencies required npm והגרסאות של ה-packages. כמו כן נראה את כל הקבצים והתיקיות החדשות שנוצרו בצורה אוטומטית:



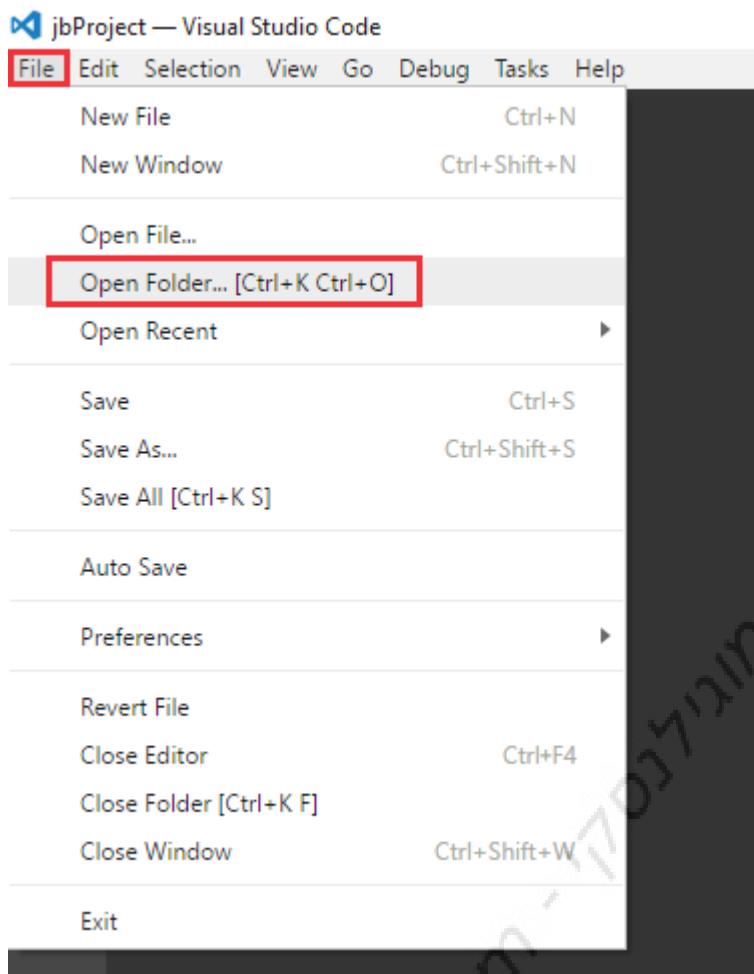
על מנת לעבוד על קוד הפרויקט יש לפתח IDE מותאים. ה-IDE שנעבדו אותו בחוברת זו יהיה Visual studio code

הוא IDE חינמי. ניתן להוריד ולהתקין אותו מהלינק הבא:

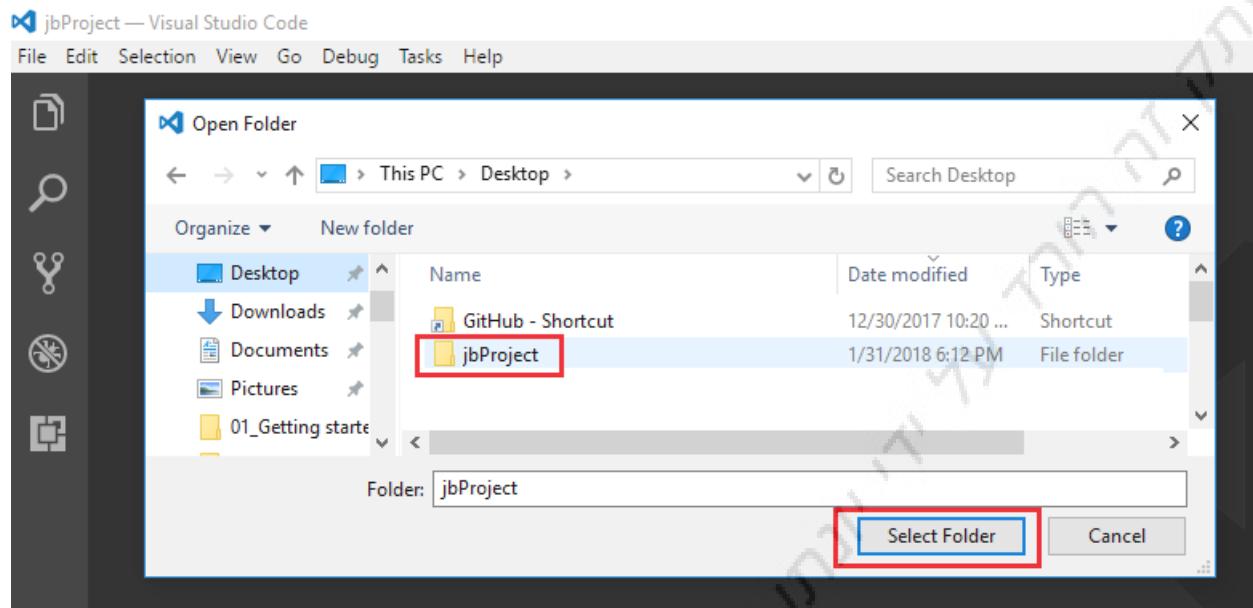
<https://code.visualstudio.com/download>

לאחר הורדת והתקנת סביבת הפיתוח, נפתח בעזרתה את תיקיית הפרויקט על ידי הפעולות הבאות:

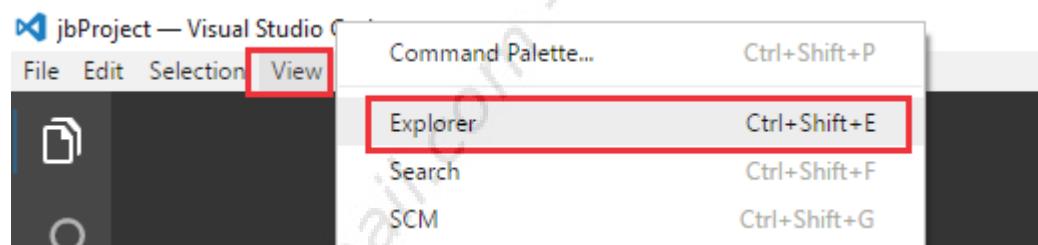
שלב א':



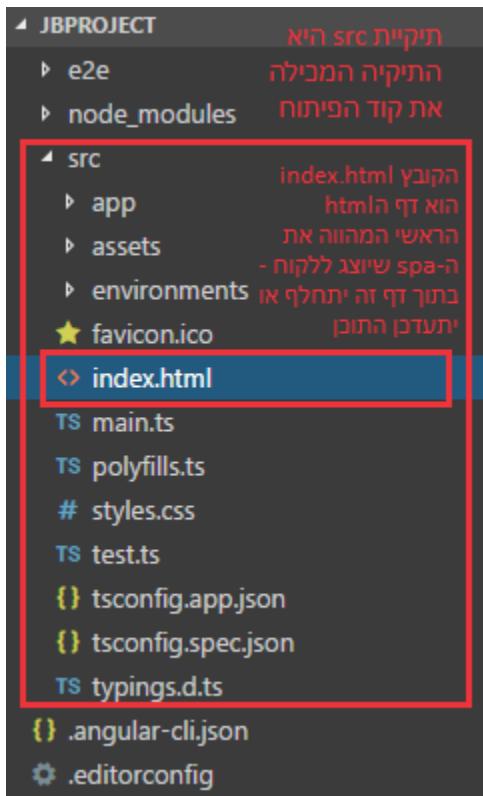
שלב ב:



שלב ג':



בחלון שנפתח נוכל לראות בטאב הצדדי את תוכנת התקינה:



וכאשר ניכנו לקובץ index.html נראה את התוכן הבא:

```

<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>JbProject</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
    <app-root></app-root>
</body>
</html>

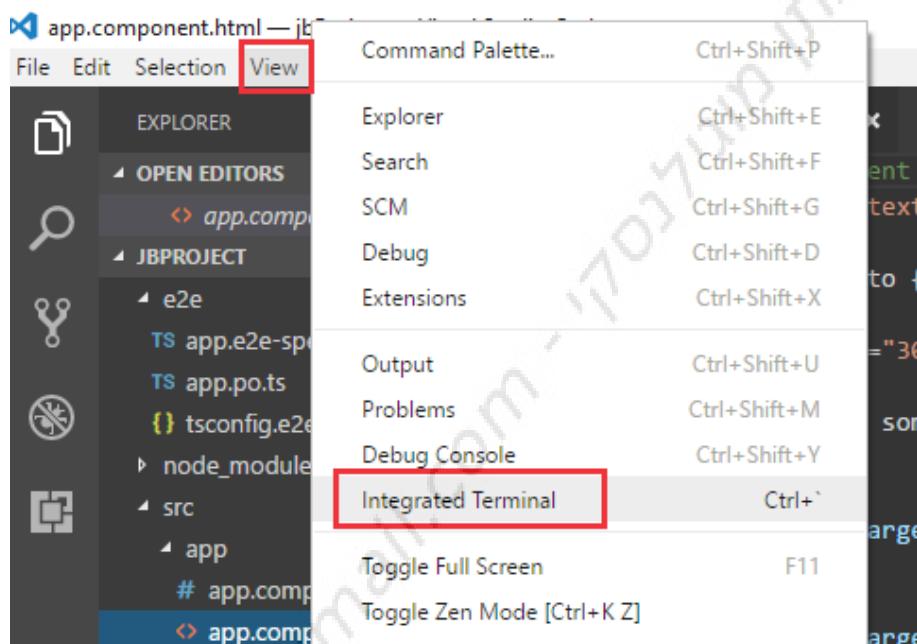
```

כפי שננו רואים, דף html.index.html הינו דף רגיל, אולם בחלק של ה-body מופיעת תגית שאינה tag html, אלא תגית של קומפוננטת אנגלור (נרחיב בפרקם הבאים על משמעותה הקומפוננטית ואופן השימוש בה).

כעת, בתור שלב אחרון לפרק זה, נבצע הרצה של הפרויקט באמצעות הפקודה serve השויכת ל-CLI. הפקודה serve מ Ritchה את הפרויקט בצורה דיפולטית על שרת מקומי ומאפשרת לגשת אליו מהדף על ידי גישה לכתובת <http://localhost:4200>

הפעם נריץ את הפקודה ישירות מחלון ה-CLI המובנה בתוך visual studio code, בעזרת הפעולות הבאות:

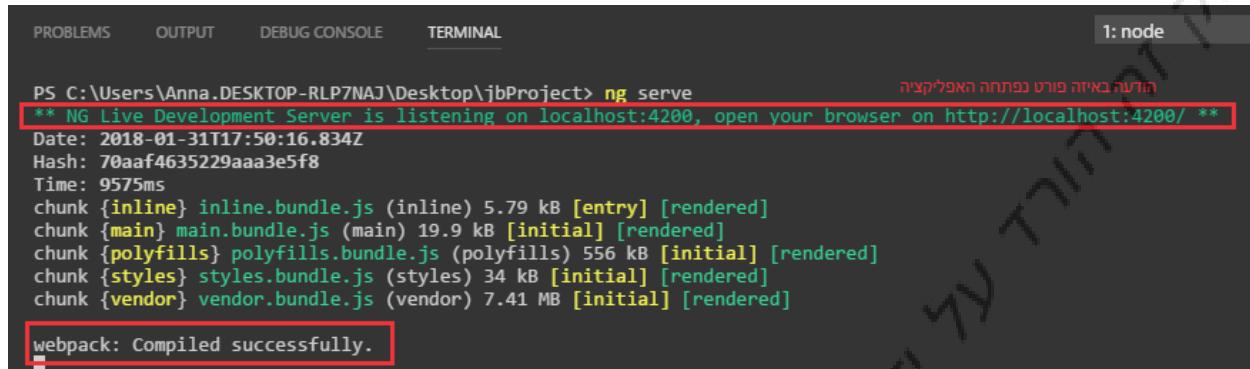
1. נפתח את החלון של ה-CLI



2. נריץ את הפקודה ng serve

The screenshot shows the VS Code terminal window. The tab bar above it has 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL' tabs, with 'TERMINAL' currently selected. The terminal shows a Windows PowerShell prompt: 'PS C:\Users\Anna\Desktop\RLP7NAJ\Desktop\jbProject>'. A red box highlights the command 'ng serve' as it is being typed into the terminal.

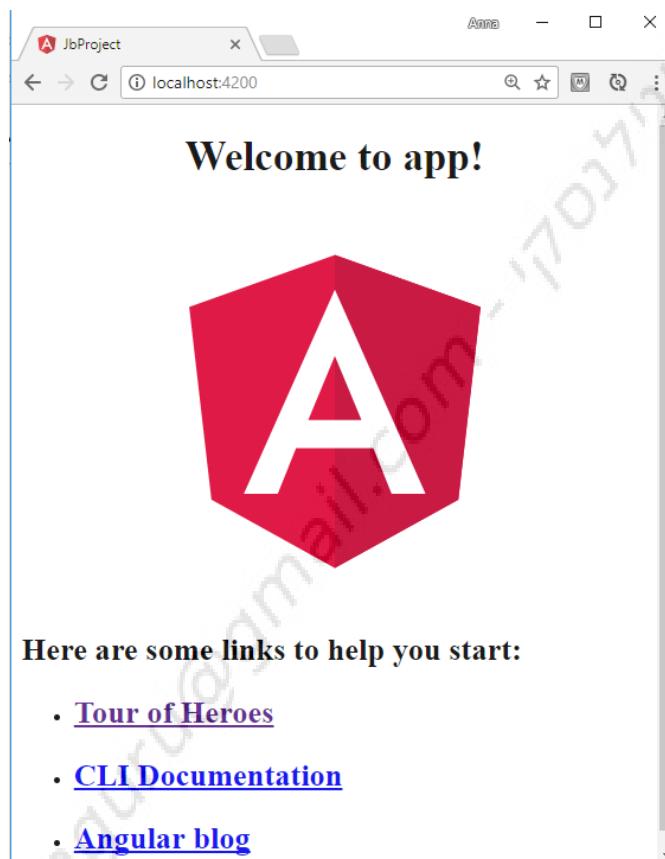
3. נמתין עד שההרצה תסתיים בהצלחה.



```
PS C:\Users\Anna\Desktop-RLP7NAJ\Desktop\jbProject> ng serve
** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
Date: 2018-01-31T17:50:16.834Z
Hash: 70aaaf4635229aaa3e5f8
Time: 9575ms
chunk {inline} inline.bundle.js (inline) 5.79 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 19.9 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 556 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 34 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 7.41 MB [initial] [rendered]

webpack: Compiled successfully.
```

4. נפתח בדפדפן דף עם הכתובת: localhost :4200 , ונקבל את התוצאה הבאה:



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

2.2. השימוש ב-CLI Angular

שימוש מסיע בפעולות הבאות:

1. **Bootstrapping לפרויקט**

Angular CLI יוצר את מבנה הפרויקט הראשוני עם NgModule root component ומרכיב .platformBootstrapDynamic.

הפרויקט שנוצר על ידי ה-CLI מוגדר גם להשתמש בwebpack loader אשר אחראי ל: module loading, bundling and minification

2. **ביצוע Serving או Reloading לפרויקט**

ה-CLI מפעיל web-server מקומי כדי שנוכל להציג את הפרויקט בדף. בזרה דיפולטיבית בהרצה ע"י Angular CLI ופקודת serve, הפרויקט יפתח בדף בכתובת localhost: 4200

בנוסף, ה-CLI גם צופה בכל שינוי שמבצע בקבצים כאשר האתר במצב ריצה, וב敖פן אוטומטי מבצע refresh ומעדכן את הדף בדף.

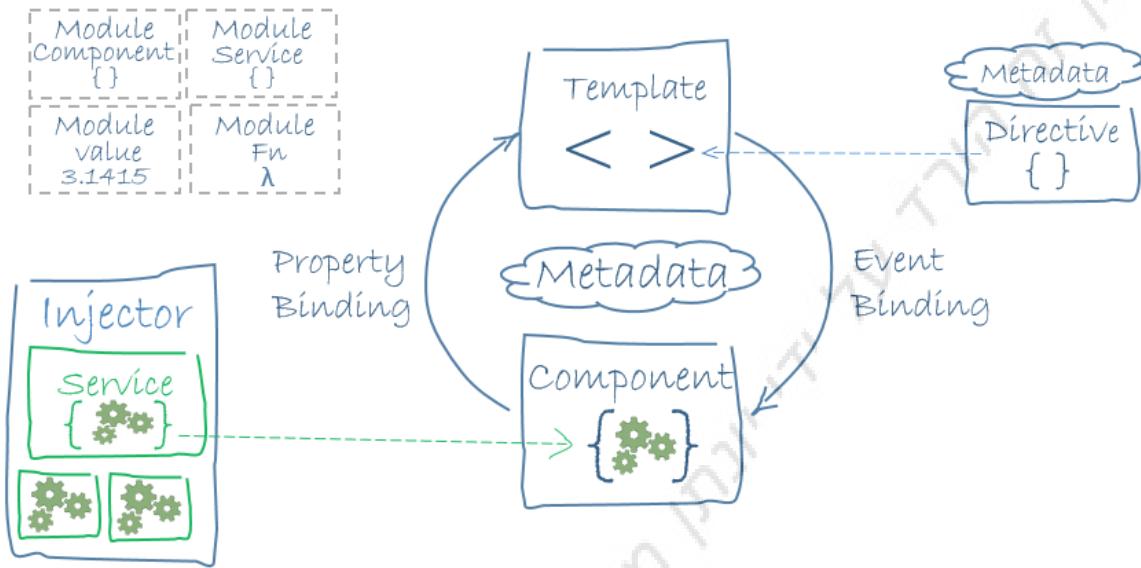
3. **הידור קוד - Code generation**

באמצעות CLI אנו יכולים ליצור components directives, services, pipes, ועוד, זאת באמצעות הרצת הפקודה המתאימה ב-CLI Angular.

4. **Packaging and Releasing**

CLI משמש ככלי עזר לא רק עבור הפיתוח, אלא באמצעותו אנו יכולים לבצע package release לאפליקציה, ולהcinן אותו לשלב.

Angular 2.3. ארכיטקטורת



נתבונן על מספר מרכיבים עיקריים:

- `NgModule`
- `Components`
- `Directives`
- `Services`
- `dependency injection`

NgModule .2.3.1

זה רכיב המסייע לסדר ולארגן את האפליקציה. התשתיות מחליטה כיצד להדר (compile) ולהריץ את הקוד, על סמך ה-`metadata` שניתן לאובייקט.

Root Module .2.3.2

בכל אפליקציה, מודול מרכזי אחד מהוות את הבסיס לכל האפליקציה. מודול זה נקרא `Root Module`.
המודול מגדר במה האפליקציה משתמש, ואת אופן ההרצה:

```

1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { AppComponent } from './app.component';
4
5 @NgModule({
6   declarations: [AppComponent, MyComboboxComponent,
7                 CollapsibleDirective, CustomCurrencyPipe],
8   imports: [BrowserModule],
9   providers: [UserService, LessonsService],
10  bootstrap: [AppComponent]
11})
12
13 export class AppModule {
14}
15

```

לדוגמא:
אפליקציה זו משתמשת
במספר רכיבים.
AppComponent
היא מייבאת מודול בשם
Browser ומשתמשת בשני
שירותים: User ו-Lessons.

הרכיב הראשון שירוץ הינו
ה-AppComponent
כך-קובל.

בהמשך נתמקד בתפקיד
של כל חלק וחלק בהגדרות
אלו. כרגע, נבין, חלק זה,
הוא איגוד רכיבים, בהם
משתמשת האפליקציה.

Components .2.3.3

ה-component הינו אבן הבניה העיקרי של Angular. הוא מכיל את הלוגיקה, תבנית העיצוב (HTML)
ועיצוב (CSS).

```

+/** Created by shai vashdi on 27/08/2016. ...*/
import {Component} from '@angular/core';

@Component({
  selector: 'headerControl',
  templateUrl: './header.control.template.html',
  styleUrls: ['./header.control.style.css'],
})

export class HeaderControlComponent {
}

```

Directives. 2.3.4

כל ייצוג של Angular בתוך תגית html כלשהי, נקרא Directive. תפקיד ה-Directive הוא לייצר קשר בין אלמנט ב-DOM לבין האפליקציה.

ישנם שלושה סוגים של Directive:

▪ Component – ראו לעיל.

▪ Structural directive – משנה את ה-DOM על ידי הוספה והסרה של אלמנטים, למשל:
.ngFor ו-.ngIf

▪ Attribute directive – משנה את המראה או התנהלות של אלמנט, למשל: `NgStyle`.

למעשה, Component הוא ה-Directive הנפוץ ביותר מפני שהוא משלב HTML template.

Services. 2.3.5

אפליקציות מורכבות ממספר תות מערכות, למשל: logging, data access, caching. המושג Service ב-Angular, הינו כימוי (encapsulation) של פונקציונליות כלשהי, כדי שתוכל לספק את הפונקציונליות זו, באופן בלתי תלוי, לשאר חלקי האפליקציה.

ב-Angular, ה-services מועברים דרך constructor באמצעות dependency injection.

Dependency injection. 2.3.6

זרקת תלויות (באנגלית: dependency injection), הינה תבנית עיצוב, המאפשרת בחירה שלרכיבי תוכנה, בזמן ריצה (ולא בזמן הרידור). התבנית זו יכולה, לדוגמה, לשמש כדרך פשוטה לטעינה דינמית של `sin-inject` או בחירה באובייקטי דמה (mock objects) בסביבות בדיקה, במקומם להשתמש באובייקטים אמיתיים של סביבת הייצור. תבנית עיצוב זו, מזריקה את האלמנט שתלוים בו (אובייקט או עריך) וכדומה) אל היעד שלו, בהתבסס על ידיעה של צרכי היעד.

מנגנון ה-dependency injection ב-Angular, מורכב משלושה חלקים עיקריים:

- Injector – הרכיב שמייצר את ה-instance של האובייקט המזרק.
- Provider – הרכיב שמאגדיר כיצד לייצר את האובייקט. למעשה, ממפה את הדרישה לייצור האובייקט אל פונקציית factory, המייצרת אותו בפועל.
- Dependency – סוג האובייקט אותו מבקשים לייצר בדרך זו.

שלבי הביצוע:

```

import ...

@Injectable()
export class ActionsService {

    @Component({
        selector: 'home',
        styleUrls: ['./home.style.css'],
        templateUrl: './home.template.html',
        directives: [ActionBarControlComponent],
        providers: [ActionsService, ProductsService],
    })
    export class Home {
    }
}

export class ProductEditModuleComponent implements OnInit, OnDestroy{

    public selectedProductId: number = NaN;
    public product: IProduct = null;
    paramsSub: Subscription;
    actionsSub: Subscription;

    constructor(private router: Router, private activatedRoute: ActivatedRoute,
               private productsService: ProductsService,
               private actionsService: ActionsService)
    {
    }
}

```

- ראשית יש לסמן את ה- Type של האובייקט המיועד כ- Injectable

- נגידר כיצד ליצור את האובייקט (במקרה זה ללא פרמטרים) באמצעות ה- provider

- נגידר היכן משתמשים באובייקט באמצעות ה- constructor

במהלך, נבין מדוע האובייקט המוזרק הוא singleton בתוך scope ההגדירה וכייד יוכל ליצור יותר מופעים שלו תחת scope מסוים.



Components .3

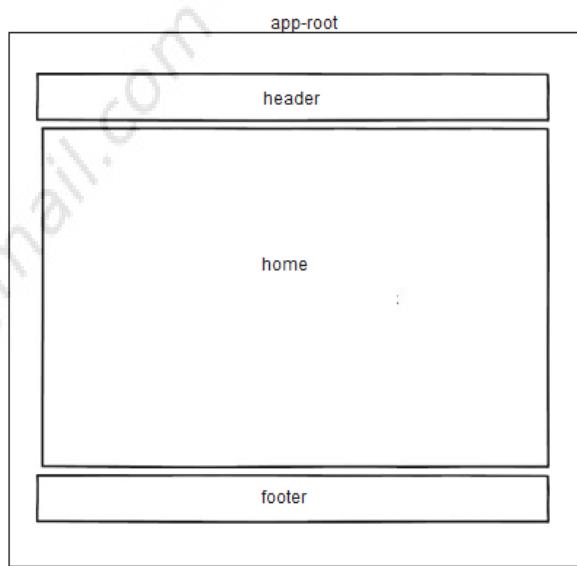
– מבט מעמיק Components .3.1

Angular applications הם אבן היסוד של Components. Composable Components הם יכולם לבנות רכיבים גדולים על ידי הכלת רכיבים קטנים. Angular application הוא רק עץ של Components, כאשר כל רכיב שעובר תהליך render ומוכנס לדף html, הוא מכניס בצורה רקורסיבית את הchildren Components. root Component הוא הרכיב הראשון, רכיב ה-.html. השורש של העץ הוא הרכיב ברמה העליונה, רכיב ה-root Component.

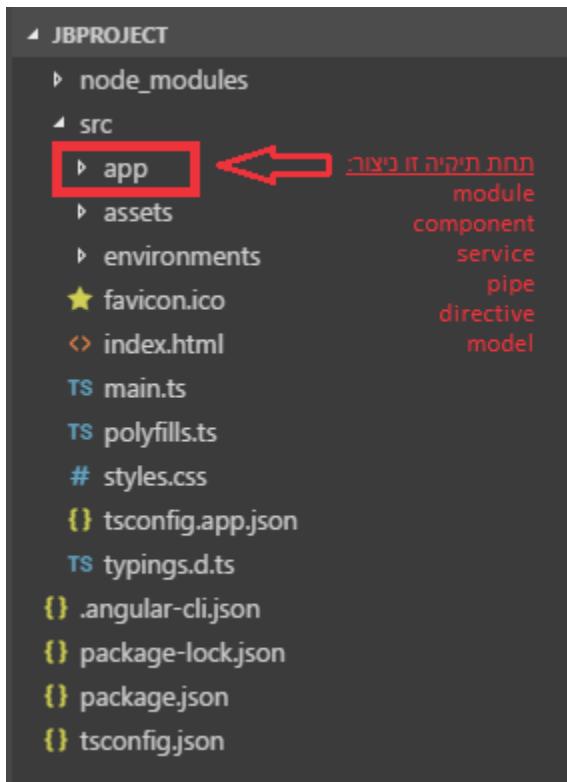
כשהם מבצעים תהליכי bootstrapangular application אנחנו אומרים לדפדן להתחיל לבצע ל-root Component ובקernel מבצעים פעולה render לכל child Component שיבצע render לכל ה-child Components.

Architecting with Components .3.1.1

כאשר אנו מתחילה בבניית Angular application חדש, אנו מתחילה על ידי הפרדת ה-components לתוכם נפרדים. בפרק זה לדוגמה נבנה דף המורכב מהמבנה הבא:

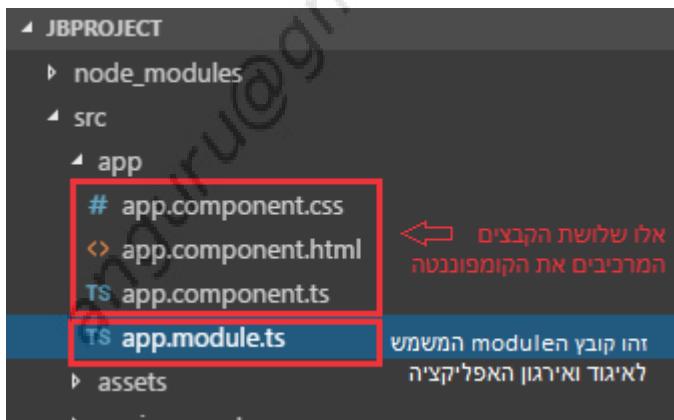


את התבנית הנ"ל ניתן ליצור בפרויקט שיצרנו בפרקים הקודמים על ידי הפקודה `ng new app`, ונתמקד בתוכן תיקית app:



בצורה דיפולטית, כאשר אנו יוצרים פרויקט אングולר פשוט, נוצרת לנו בתוך תיקיית app קומפוננטה בשם AppComponent המורכבת משלושה קבצים:

1. TypeScript - קוד של מחלקה המייצג את ה-component, נכתב ב-.component.ts.
2. HTML - תבנית הרכיב, שנכתבה ב-.html, ויכולת תוצאות שיוצגו בדף.
3. CSS - סגנון CSS הפרטיאם של ה-component, כתוב ב-.component.css.



3.1.2. קובץ app.component.ts

בקובץ app.component.ts נראה את התוכן:

```
import { Component } from '@angular/core';  
//יבוא שנבעצע מספרית אנגליר, כדי שטובל להשתמש בcomponent  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
)  
export class AppComponent {  
  //שם המחלקה תמיד יתחיל באות גזלה, ויסתיים עם הסימות עליון הקומפוננטה זו  
  title="hello angular";  
}
```

נשים לב, לשתי תוספות חשובות, שיש להקפיד להוסיף ביצירת ה-class עברו ה-component:

- **@Component** •
פקודה מעלה Class שמנגידירה את ה-class Component.
- **export** •
פקודה הנכתבת לפני class ומאפשרת להשתמש ב-class בקבצי TypeScript אחרים.

בעוד שהקוד ב-class component מכיל את לוגיקת הרכיב, תבנית העיצוב מכילה את הייצוג היזואלי של הרכיב.

לכן, על מנת להבין טוב יותר את מושג ה-component, נתבונן על הקשר שבין ה-component לתבנית העיצוב שלו (component template).

3.1.3. Interpolation

ב- Angular החיבור הבסיסי בין תצוגה ללוגיקה מתבצע באמצעות סוגרים מסולסלים כפויים: {}{ }.

להלן מספר דוגמאות:

```
<h3>  
  {{title}}  
    
</h3>
```

חיבור למשנה title
בכותרת h3.
 לחברו למשנה heroImageUrl
על מנת לקבל את כתובות התמונה.

```
<!-- "The sum of 1 + 1 is 2" -->
<p>The sum of 1 + 1 is {{1 + 1}}</p>
```

כאן משתמשים
במספרים קבועים ולא
במשתנים על מנת לבצע
חישוב בתוך ה-
template בלבד.

```
<!-- "The sum of 1 + 1 is not 4" -->
<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}</p>
```

חיבור לפונקציה
מסויימת הנמצאת
ברכיב. הפונקציה
getVal מחזיר ערך
כלשהו אשר יטרף
לערך $2(1+1)$.

3.1.4. הגדרת תבנית html ועיצוב css להומפוננטה

ישן שתי דרכים בהן ניתן להגדיר את תבנית HTML של הרכיב:

- על ידי קישור לדף HTML נפרד
- כמחרוזת המיצגת דף HTML בהגדרת component

וכן, ישן שתי דרכים עיקריות להגדיר עיצוב לרכיב:

- styleUrls array – הגדרת מערך של מצלבים css לקבץ המכילים אלמנטים עיצוביים
- Styles array – הגדרת מערך של אלמנטים עיצוביים

3.1.5. הגדרת קישור חיצוני ל-`html` ול-`css`

נפתח את שלושת הקבצים, ונשנה את תוכנם לפי הקוד הבא:

```

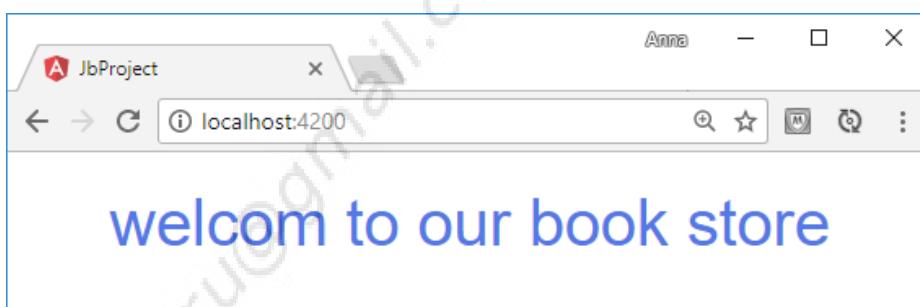
app.component.html
1 <h1>welcom to our {{title}}</h1>
2

# app.component.css
1 h1{
2   text-align: center;
3   margin: 20px;
4   color: royalblue;
5 }

TS app.component.ts
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title="book store";
10 }
11

```

כעת, נרץ את האפליקציה על ידי הפקודה `ng serve` ונקבל את התוצאה הבאה:

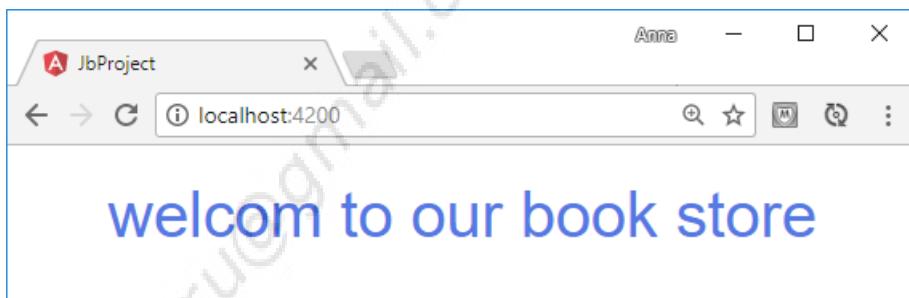


3.1.6. הגדרת תוכן ישיר ל-.html ול-.css

נפתח את קובץ app.component.ts בלבד, ונסנה את תוכנו לפי הקוד הבא:

```
TS app.component.ts x
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   template: '<h1>welcom to our {{title}}</h1>',
6   styles: [
7     h1{
8       text-align: center;
9       margin: 20px;
10      color:royalblue;
11    }
12  ]
13})
14 export class AppComponent {
15   | title="book store";
16 }
```

כעת, נריץ את האפליקציה על ידי הפקודה `ng serve` ונקבל את התוצאה הבאה:



3.2. ייצור Component חדש

הווסף קומפוננטה חדשה לפרויקט, יכולה להתבצע בקלה על ידי שימוש ב-CLI עם הפקודה הבאה:

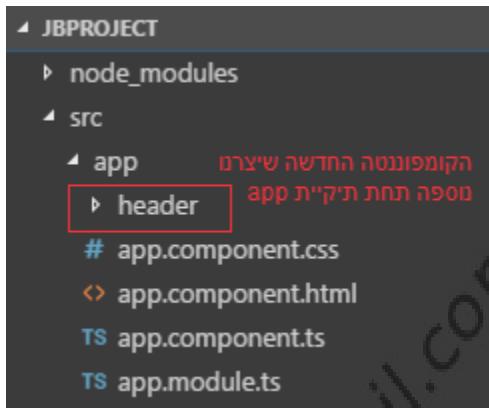
C:\Users\Anna.DESKTOP-RLP7NAJ\Desktop\jbProject> **ng g c header**

generate ←
component ↓
שם הקומפוננטה
(לפי בחירת המתכנן)

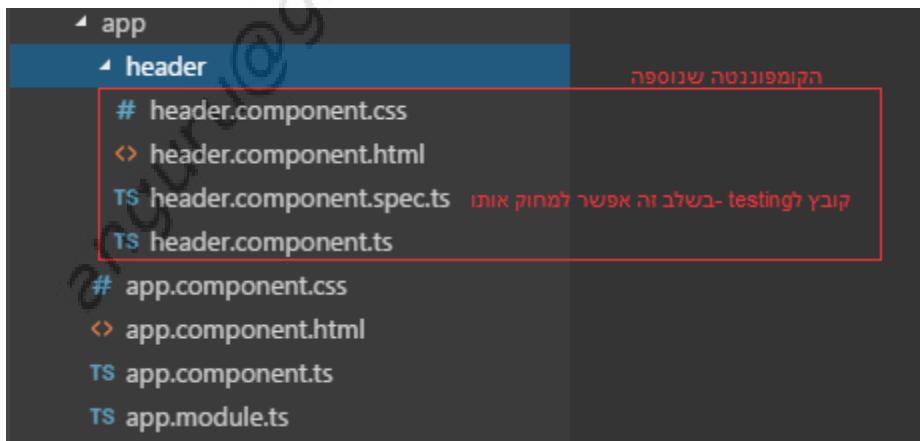
לאחר סיום הרצת בפקודה בהצלחה, נקבל את הפלט הבא:

```
header
create src/app/header/header.component.html (25 bytes)
create src/app/header/header.component.spec.ts (628 bytes)
create src/app/header/header.component.ts (269 bytes)
create src/app/header/header.component.css (0 bytes)
update src/app/app.module.ts (462 bytes)
```

וכאשר נבדוק את תקינות app נראה שהתווסף לה תת-תיקיה אשר שמה הוא שם הקומפוננטה שיצרנו:



בתוך התיקייה הנ"ל, נראה שהתווסף לנו כל הקבצים המרכיבים קומפוננטה (כלומר קובץ ts, קובץ html, קובץ css):



ニיכנו לתוכן ונראה שההויסף למערך declarations שם ה-class של הקומפוננטה שיצרנו, וכן import של הקומפוננטה (על מנת שנוכל להויסף אותה למערך declarations):

```
TS header.component.ts ✘
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4      selector: 'app-header',
5      templateUrl: './header.component.html',
6      styleUrls: ['./header.component.css']
7  })
8  export class HeaderComponent implements OnInit {
9
10     constructor() { }
11
12     ngOnInit() {
13     }
14
15 }
16
```



```
TS app.module.ts ✘
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { FormsModule } from '@angular/forms';
4
5  import { AppComponent } from './app.component';
6  import { HeaderComponent } from './header/header.component';

7
8
9  @NgModule({
10     declarations: [
11         AppComponent,
12         HeaderComponent
13     ],
14     imports: [
15         BrowserModule,
16         FormsModule
17     ],
18     providers: [],
19     bootstrap: [AppComponent]
20 })
21 export class AppModule { }
```

כעת, ניכנס לתוך הקוד של הקבצים שהתווספו, ונסנה אותו לתוך הבא:

AppComponent

The screenshot shows three files in the Angular code editor:

- app.component.html**: Contains the template code: `

welcom to our {{title}}</h1>` and an `app-header` component.
- app.component.css**: Contains the CSS rule: `h1{ text-align: center; margin: 20px; color: royalblue; }`.
- app.component.ts**: Contains the TypeScript code defining the AppComponent class with a title property set to "book store".

Annotations in Hebrew explain the code structure:

- "הכללה של מופע קו-פונקציה" points to the `app-header` component in the HTML template.
- "על ידי שירות וניתן עם selector שמשה HeaderComponent" points to the `HeaderComponent` selector in the HTML template.
- "שם ה-HeaderComponent שעובדך ל'" points to the `HeaderComponent` selector in the CSS file.

HeaderComponent

The screenshot shows three files in the Angular code editor:

- header.component.html**: Contains the template code: `

{{title}}</h1>`.
- header.component.css**: Contains the CSS rule: `h1{ text-align: center; margin: 20px; color: red; }`.
- header.component.ts**: Contains the TypeScript code defining the HeaderComponent class as implementing OnInit, with a title property set to "header section".

Annotations in Hebrew explain the code structure:

- "על ידי הפיקוד על ידיה הפיקודה נקבל את התוצאה הבאה בדף:" points to the `{{title}}` placeholder in the HTML template.
- "ה-HeaderComponent שעובדך ל'" points to the `HeaderComponent` selector in the CSS file.

כאשר נריץ את הפרויקט על ידי הפיקודה `ng serve` נקבל את התוצאה הבאה בדף:



welcom to our book store

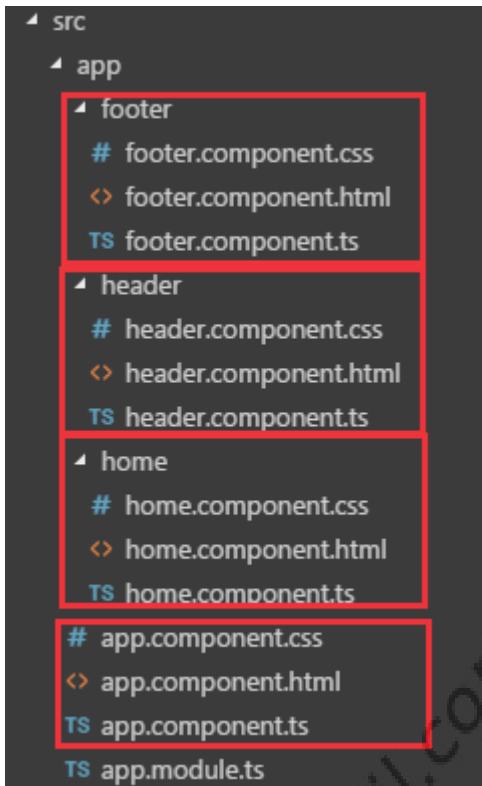
header section

בשלב זה, נוסיף עוד שני קומפוננטות:

C:\Users\Anna\Desktop-RLP7NAJ\Desktop> **ng g c footer**

C:\Users\Anna\Desktop-RLP7NAJ\Desktop> **ng g c home**

ותחת תקית דם נקבל את התוצאה הבאה:



להלן הקוד של הפרוייקט עבור השלב הנוכחי:

app-component

```

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

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

app.component.ts

```
<div id="page">
  <header>
    <app-header></app-header>
  </header>
  <main>
    <app-home></app-home>
  </main>
  <footer>
    <app-footer></app-footer>
  </footer>
</div>
```

app.component.html

```
#page {
  padding: 0 0 50px 0;
  min-height: 100%;
  box-sizing: border-box;
  position: relative;
  text-align: center;
  border: 2px solid black;
  background-image: linear-gradient(white, gray);
}

header {
  height: 100px;
  line-height: 100px;
  border: 0 solid black;
  border-bottom-width: 2px;
}

footer {
  height: 50px;
  line-height: 50px;
  width: 100%;
  border: 0 solid black;
  border-top-width: 2px;
  position: fixed;
  bottom: 0px;
}
```

app.component.css

header-component

<pre>import { Component, OnInit } from '@angular/core'; @Component({ selector: 'app-header', templateUrl: './header.component.html', styleUrls: ['./header.component.css'] }) export class HeaderComponent implements OnInit { title = "header section"; constructor() { } ngOnInit() { } }</pre>	header/header.component.ts
<pre><h1>{{title}}</h1></pre>	header/header.component.html
<pre>h1{ text-align: center; margin: 20px; color:red; }</pre>	header/header.component.css

home-component

<pre>import { Component, OnInit } from '@angular/core'; @Component({ selector: 'app-home', templateUrl: './home.component.html', styleUrls: ['./home.component.css'] }) export class HomeComponent implements OnInit { title="book store"; constructor() { } ngOnInit() { } }</pre>	home/home.component.ts
<pre><h1>welcom to our {{title}}</h1> <p>home - section</p></pre>	home/home.component.html
<pre>h1{ text-align: center; margin: 20px; color:royalblue; }</pre>	home/home.component.css

footer-component

<pre>import { Component, OnInit } from '@angular/core'; @Component({ selector: 'app-footer', templateUrl: './footer.component.html', styleUrls: ['./footer.component.css'] }) export class FooterComponent implements OnInit { constructor() { } ngOnInit() { } }</pre>	footer/footer.component.ts
<pre><p> all rights reserved </p></pre>	footer/footer.component.html
<pre>p{ text-align: center; }</pre>	footer/footer.component.css

app.module

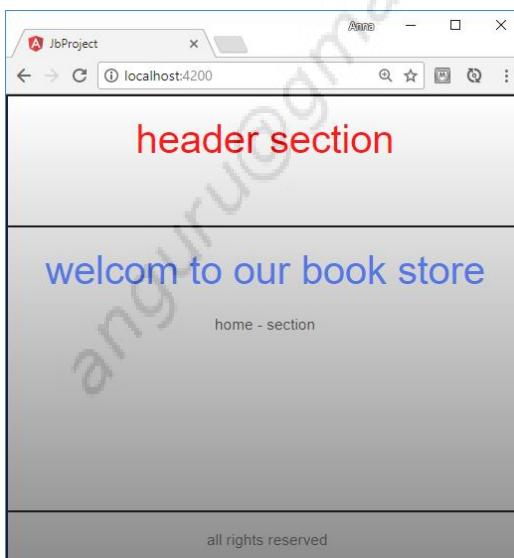
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { FooterComponent } from './footer/footer.component';
import { HomeComponent } from './home/home.component';

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

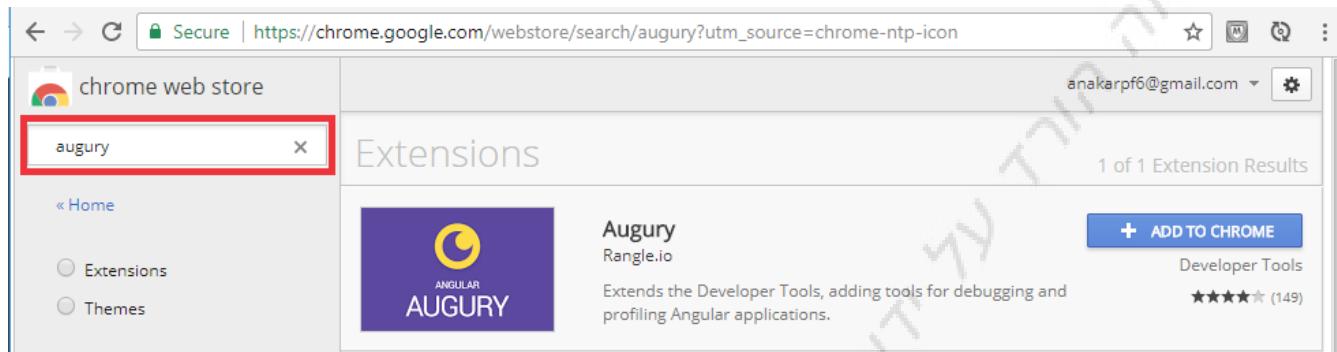
app.module.ts

כasher נרץ בשלב זה את הפרויקט על ידי הפקודה serve או ונפתח את הדף נראה את התוצאה
הבהא:

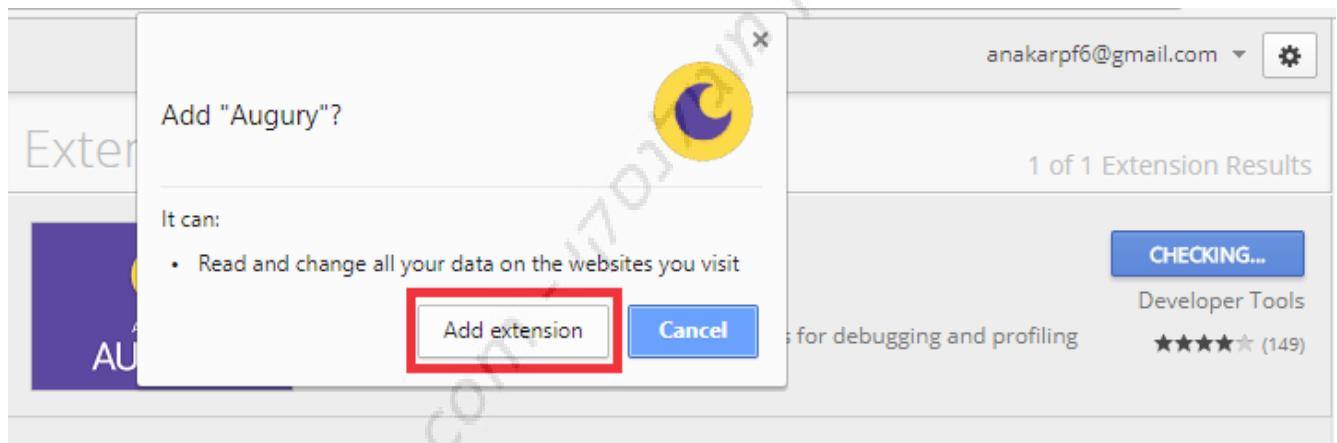


Debugging with Augury .3.3

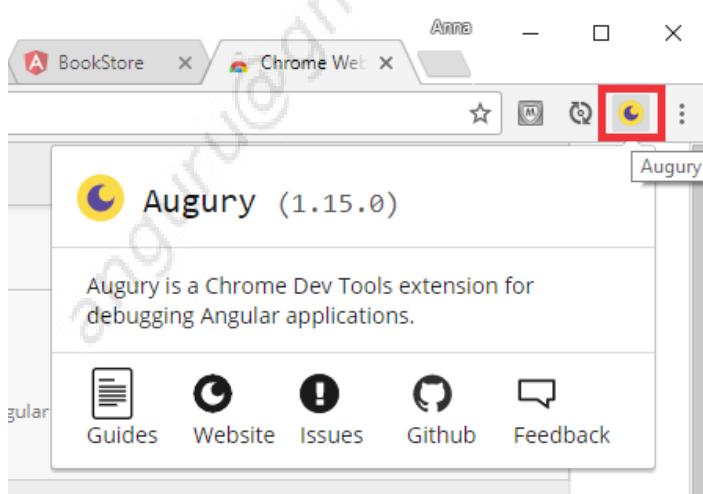
מומלץ להשתמש בתוסף כרום הבא:



בוצע את ההתקנה:



ונראה שההווסף הותוסף בהצלחה:



כעת, נחזור לדף האנגולר שהרכינו בדף, ובחילונית של dev tools נשימוש בתוסוף:

The screenshot shows the Chrome DevTools interface with the Network tab selected. In the bottom right corner of the toolbar, there is a red box around the 'Augury' button. To its left, under the 'Audits' section, is another red box around the 'Augury' button. This indicates that the Augury extension is active and integrated into the DevTools.

בצורה זו נוכל לראות את חלוקת הקומפוננטות בדף:

The screenshot shows a browser window displaying an Angular application at `localhost:4200`. The page has a header section with the text "header section", a main content area with "welcom to our book store" and "home - section", and a footer section with "all rights reserved". Above the browser window, the Augury extension's interface is visible in the DevTools. It includes a Component Tree tab, a Router Tree tab, and an NgModules tab. The Component Tree tab is selected and shows the component hierarchy: AppComponent > HeaderComponent > HomeComponent > FooterComponent. The HeaderComponent node is highlighted with a red box. The Injector Graph tab is also highlighted with a red box and shows a dependency tree starting from the root component, with AppComponent and HeaderComponent as children.

וכן נוכל לראות את המשתנים השונים של הקומפוננטה, או לגשת ישירות לקוד הדס של הקומפוננטה:

The screenshot shows a browser window displaying a web application. The header contains the text "header section". The main content area says "welcom to our book store" and has a sub-header "home - section". The footer area says "all rights reserved". Below the browser is the Augury extension interface. The "Component Tree" tab is selected, showing the component hierarchy: AppComponent > div > header > HeaderComponent. The "Properties" tab is selected in the Augury interface. It shows the component "HeaderComponent" with a "View Source" button highlighted by a red box. The "title" property is also highlighted by a red box and set to "header section". Other properties shown include "Change Detection: Default" and "State". The "Injector Graph" tab is also visible.

כל זה הינו שימושי מאד, ועזר פעם רבות לזכור את תהליכי ההתחמאות באפליקציות מסועפות ומסובכות.

להלן-link לדוקומנטציה המלאה עבור שימוש בתוסף זה:

<https://augury.angular.io/pages/guides/>



View Encapsulation .3.4

לסיום פרק זה, נתבונן בחלק css ששייכנו ל-`html templates` ע"י הקומפוננטות. את הגדרות ה-`css` רשםנו بصورة הבאה:

```
# home.component.css
1 h1{
2     text-align: center;
3     margin: 20px;
4     color: royalblue;
5 }
6

# header.component.css
1 h1{
2     text-align: center;
3     margin: 20px;
4     color: red;
5 }
6
```

ובדף, קיבלנו את התוצאה הבאה:

The screenshot shows the browser output and the Angular DevTools component tree. The browser displays a header with 'header section' (red) and a main content area with 'welcom to our book store' (blue). The DevTools tree shows the component structure:

- AppComponent
 - div
 - header
 - HeaderComponent
 - main
 - HomeComponent
 - footer
 - FooterComponent

בדרכו הכלל, אם אנו מגדירים מאפייני עיצוב עבור תגית `h1` על ידי CSS האפקט נראה לאורך כל תגיוט ה-`h1` של הדף הנוכחי בדף, אולם באפליקציה שלנו ה-`style` CSS שכתבנו לקומפוננטה מסוימת משפיע רק על תגיוט ה-`h1` של אותה קומפוננטה.

הסיבה לתופעה זו היא **View Encapsulation** הממש באפליקציה אנגלר את עקרון ה-`shadow DOM` שמאפשר לנו לכלול סגנוןות עיצוב לתוך Components מביי לשיפור מחוץ להיקף של ה-Component.

אנגולר מספקת, על ידי ה-`enum` הבא, שלושה ערכים אפשריים עבור ה-`encapsulation` של הגדרות העיצוב:

```
export declare enum ViewEncapsulation {
  Emulated = 0,
  Native = 1,
  None = 2,
}
```

[ViewEncapsulation.Emulated](#)

זהו הערך הדיפולטיבי המוגדר על ידי אנגולר עבור הגדרות CSS שנוספו על ידי קומפוננטות. במצב `ViewEncapsulation.Emulated` ל-`selector` CSS נקבע מינה את ה-`selector` המקורי יותר הבוחר רק את התגיוט שנוצרו ברכיב.

להלן מימוש הגדרת ה-`ViewEncapsulation.Emulated` בצורה מפורשת בקוד:

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'],
  encapsulation: ViewEncapsulation.Emulated
})
```

```
@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css'],
  encapsulation: ViewEncapsulation.Emulated
})
```

כאשר נרים כעת את הדף בדף נקבל את התוצאה הבאה:

The screenshot shows the browser's developer tools with the 'inspect' tool applied to the main header element. The DOM tree on the left shows the following structure:

```
<main _ngcontent-c0>
  <app-home _ngcontent-c0 _nghost-c2>
    <h1 _ngcontent-c2>welcom to our book store</h1>
    <p _ngcontent-c2>home - section</p>
  </app-home>
</main>
```

The CSS styles panel on the right shows the styles applied to the `h1` element:

```
h1[_ngcontent-c2] {
  text-align: center;
  margin: ▶ 20px;
  color: ■ royalblue;
}
```

Red arrows point from the Hebrew text "הא צ שמתקובל על inspect די" to both the DOM node and the CSS rule.

The screenshot shows the browser's developer tools with the 'inspect' tool applied to the header component. The DOM tree on the left shows the following structure:

```
<header _ngcontent-c0>
  <app-header _ngcontent-c0 _nghost-c1>
    <h1 _ngcontent-c1>header section</h1> == $0
  </app-header>
</header>
```

The CSS styles panel on the right shows the styles applied to the `h1` element:

```
h1[_ngcontent-c1] {
  text-align: center;
  margin: ▶ 20px;
  color: ■ red;
}
```

Red arrows point from the Hebrew text "הא צ שמתקובל על inspect די" to both the DOM node and the CSS rule.

ViewEncapsulation.Native

אם אנחנו רוצים להשתמש בDOM shadow אנו יכולים להגדיר את הערך של `ViewEncapsulation` ל-`.Native`.

הערך `ViewEncapsulation.Native` קובע שהגדירות css על יחולו מחוץ ל-`component`.

עם זאת,

- האדרת ViewEncapsulation.Native שלנו תגרום גם לבודד אותו מן סגנונות העיצוב הגלובליים שהגדכנו עבור היישום שלנו.
- דריש תכונה בשם DOM shadow אשר אינה נתמכת על ידי כל הדפסנים.

להלן מימוש האדרת ViewEncapsulation.Native בקוד:

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'],
  encapsulation: ViewEncapsulation.Native
})
```

```
@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css'],
  encapsulation: ViewEncapsulation.Native
})
```

כאשר נריץ כעת את הדף בדפסן נקבל את התוצאה הבאה:

The screenshot displays a web application interface and its underlying DOM structure. On the left, the application shows a header section with the text "header section" in red and a main section with the text "welcom to our book store" in blue. Below the main section, there is a footer with the text "home - section". On the right, the DOM tree is shown with various elements and their styles. Two specific sections of the DOM tree are highlighted with red boxes, indicating shadow roots. The first highlighted section is under the header element, showing a style block for an h1 element with text-align: center; margin: 20px; color: red;. The second highlighted section is under the app-home element, showing a style block for an h1 element with text-align: center; margin: 20px; color: royalblue;.

```

<header _ngcontent-c0>
  <app-header _ngcontent-c0>
    <#shadow-root (open)>
      <style>...</style>
      <style>h1{
        text-align: center;
        margin: 20px;
        color:red;
      }</style>
      <style>p[_ngcontent-c3]<
        text-align: center;
      ></style>
    <h1>header section</h1> == $0
  </app-header>
</header>
<main _ngcontent-c0>
  <app-home _ngcontent-c0>
    <#shadow-root (open)>
      <style>...</style>
      <style>h1{
        text-align: center;
        margin: 20px;
        color:royalblue;
      }</style>
      <style>...</style>
      <style>p[_ngcontent-c3]<
        text-align: center;
      ></style>
    <h1>welcom to our book store</h1>
    <p>home - section</p>
  </app-home>
</main>
<footer _ngcontent-c0>...</footer>

```

ViewEncapsulation.None

אם אנו לא רוצים להגדיר בכלל encapsulation להגדרות ה-css שהוספנו על ידי הקומפוננטה, אנחנו יכולים להשתמש ViewEncapsulation.None, וAz Angular תוסיף את ההגדרות בחלק ה-head של הדף, סוגנון גלובלי עבור כל תגית מתאימה.

להלן מימוש הגדרת ה-native ViewEncapsulation.Native בקוד:

```
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'],
  encapsulation: ViewEncapsulation.None
})
```

```
@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css'],
  encapsulation: ViewEncapsulation.None
})
```

כasher נרץ כעת את הדף בדף נקבל את התוצאה הבאה:

The screenshot shows a browser window at localhost:4200. The page content includes a header section with the text "header section" and a main section with the text "welcom to our book store". The browser's developer tools are open, specifically the Elements tab of the DevTools. The DOM tree is visible, showing the structure of the components and their respective ngcontent-c0 slots. A tooltip from the DevTools highlights the "header" component's template, which contains an h1 element with the text "header section". Another tooltip highlights the "app-home" component's template, which contains an h1 element with the text "welcom to our book store" and a p element with the text "home - section". To the right of the DOM tree, the Styles panel displays the CSS rules applied to the elements. It shows three h1 selectors: one for the header component (text-align: center, margin: 20px, color: royalblue), one for the app-home component (text-align: center, margin: 20px, color: red), and a general rule for h1 elements (font-size: 36px).

Angular Bindings .4

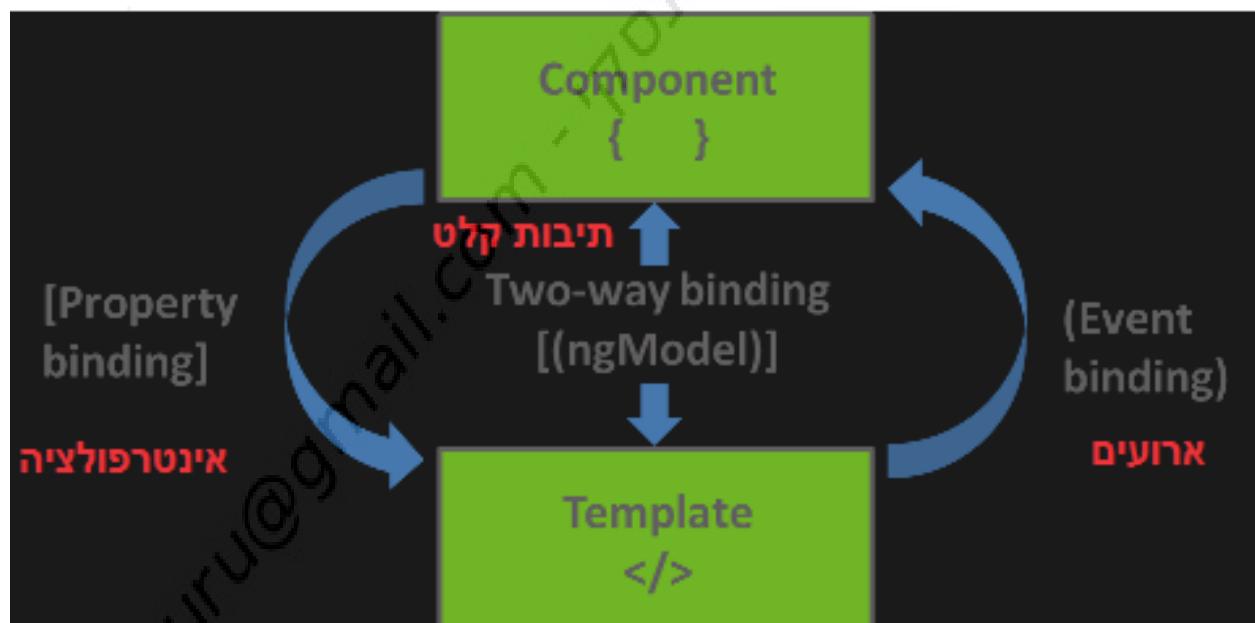
Template Syntax .4.1

המושג template מזכיר את הנושא של הפרדת קוד, המוכר ממודלים רבים כגון: MVVM, MVC.
מטרת מודלים אלו היא, לחבר בצורה בלתי תלולה בין הלוגיקה לבין התצוגה.

ב-Angular שפת ה-template הינה HTML, והקשר של html לאזור הלוגי של הקוד יתבטא במספר נקודות מפגש בין הרכיב למבנה העיצוב שלו, המתבטאים בשלושה סוגי Binding:

- One-way: from data source to view target
- One-way: from view target to data source
- Two-Way binding

שלושת האופציות הנ"ל מתבטאות כך:



ומוממשות על ידי הקוד بصورة הבאה:

Data direction	Syntax	Binding type
One-way from data source to view target	<pre> {{expression}} [target] = "expression" bind-target = "expression"</pre>	Interpolation Property Attribute Class Style
One-way from view target to data source	<pre>(target) = "statement" on-target = "statement"</pre>	Event
Two-way	<pre>[(target)] = "expression" bindon-target = "expression"</pre>	Two-way

One-way Binding – Interpolation .4.1.1

ניתן לחבר בין תצוגה ללוגיקה באופן בסיסי באמצעות סוגרים מסולסלים כפולים: {{}}. way: from data source to view target

החיבור בכיוון אחד מהלוגיקה אל התצוגה הימ וריאציות שונות של החיבור אשר ראיינו באמצעות סוגרים מסולסלים כפולים: {{}}.

<pre><h3>Home Component</h3> <p>Welcome to Angular Seed my name is {{name}}</p></pre>	<u>Interpolation</u>
<pre> <custom-control [data]="someData"></custom-control> <div [ngClass] = "{selected: isSelected}"></div></pre>	<u>Property</u> ניתן להתבונן בחיבור בסוגרים מרובעים של ה- <code>src</code> של אובייקט התמונה (חיבור <code>property</code>) וכן ל- <code>data</code> של האובייקט <code>control</code> <code>div</code> <code>ngClass</code> . שימושו של ה- <code>ngClass</code> במאזענות.

```
<!-- expression calculates colspan=2 -->
<tr><td [attr.colspan]="1 + 1">One-Two</td></tr>
<!-- ERROR: There is no `colspan` property to set!
<tr><td colspan="{{1 + 1}}>Three-Four</td></tr>
-->
```

Attribute

חיבור ל-`attribute` של ה-`tag`
כasher ain lo property מותאים. ولكن משתמש ב-`attr.`

```
    .special
CSS:  {
        background: red;
    }

<div [class.special]="isSpecial">Special</div>
```

Class

בדוגמא זו, מתחברים ל-`div`, class property ומשנים את העיצוב שלו על פ' ה-`.css`.

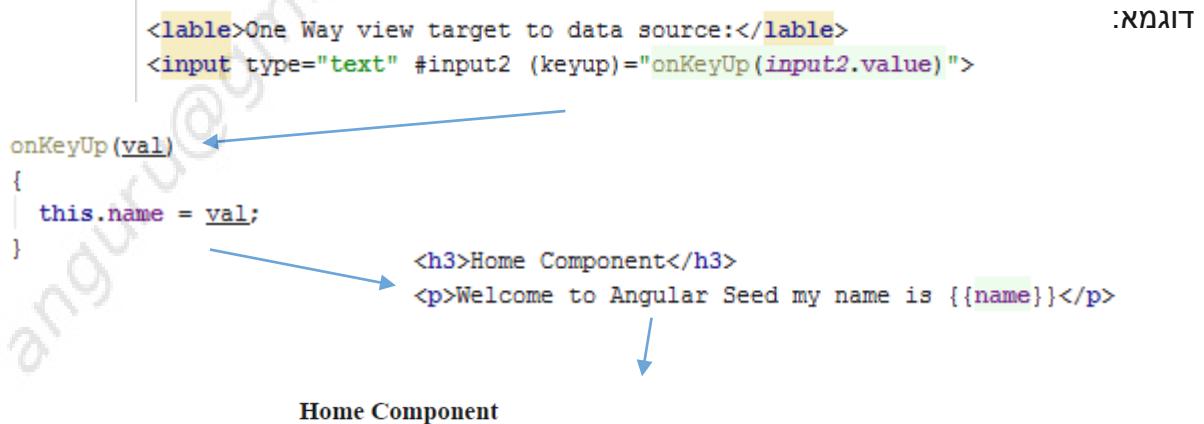
```
<button [style.color] = "isSpecial ? 'red' : 'green'"> My Button </button>
```

Style

דרך נוספת לשנות את העיצוב היא על ידי התחברות ל-`style` property. בשונה מהדוגמא הקודמת. כאן אין חיבור להגדרות בקובץ `.css`.

Binding One-way: from View Target to Data Source .4.1.2

על מנת לחבר בין התצוגה לוגיקה בכיוון אחד: תצוגה ← לוגיקה, יש להשתמש ב-`events`. זאת כיון שהדריך לדעת האם המשתמש (באמצעות התצוגה) ביצע פעולה כלשהי היא להאזין לפעולות אלה. האזנה לפעולות המשתמש היא אירוע (event).



התחברנו לאירוע "key up" של `text input`. הפונקציה המטפלת, מוגדרת בקוד הלוגיקה הנמצא ברכיב. בקוד זה נשנה את המשתנה `name`, והדבר יتبטא בתצוגה.

Two-Way Binding .4.1.3

על מנת לחבר באופן דו-כיווני בין התצוגה לבין הלוגיקה משתמש ב-`ngModel` עם סוגרים עגולים ולאחר מכן מרובעים:

```
<label>Two Way:</label>
<input type="text" [(ngModel)]="name">
```

נתבון על תהליך הבא לדוגמה:

```
<h3>Home Component</h3>
<div>
  <p>Welcome to Angular Seed my name is {{name}}</p>
</div>

<label>One Way view target to data source:</label>
<input type="text" #input2 (keyup)="onKeyUp(input2.value)">

<label>Two Way:</label>
<input type="text" [(ngModel)]="name">
```

Home Component

Welcome to Angular Seed my name is Shai Vashdi2

One Way view target to data source: Two Way:

הקלדת השם, שינתה את המשתנה `name` (כיוון תצוגה ללוגיקה). הדבר השתקף בתצוגה במקום אחר.

Home Component

Welcome to Angular Seed my name is Shimon Dahan

One Way view target to data source: Two Way:



שינוי השם בתיבת `one way view target to data source`, שינתה את המשתנה `name` (כיוון תצוגה ללוגיקה). בעוד, המשתנה `name` משנה את התצוגה (השדה של - Two Way Binding) בכך ש- לוגיקה בתצוגה.

Template Reference Variables .4.2

בתוך html template ניתן להגדיר מלבניים לאלמנטים ב-html על מנת שנוכל להשתמש במלבני
אליהם באלמנטים אחרים. לדוגמה:

```
<!-- phone refers to the input element; pass its `value` to an event handler -->
<input #phone placeholder="phone number">
<button (click)="callPhone(phone.value)">call</button>

<!-- fax refers to the input element; pass its `value` to an event handler -->
<input ref-fax placeholder="fax number">
<button (click)="callFax(fax.value)">Fax</button>
```

הגדרנו בתיבת ה-input מלבן בשם phone. ניתן להשתמש בערך ה-input בו, על מנת לשלוח בו click event של האלמנט button.
ניתן להגדיר זאת על ידי שימוש ב-# או בתחילת-ref.

לדוגמה, ניצור תיבת קלט עם מלבן, וניגש לתוכן תיבת הקלט על ידי האירוע click לשם הוספת התוכן
למערכת:

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

@Component({
  selector: 'app-root',
  template: `
    <input #newName>

    <button (click)="addName(newName.value); newName.value=''">
      Add
    </button>

    <ul><li *ngFor="let x of nameList">{{x}}</li></ul>
  `)
export class AppComponent {
  nameList = ['Dan', 'Gil'];
  addName(val: string) {
    if(val){
      this.nameList.push(val);
    }
  }
}
```

כאשר נרים את האפליקציה נקבל:

- Dan
- Gil

לאחר הקלדת השם:

- Dan
- Gil

לאחר לחיצה על כפתור:

- Dan
- Gil
- Li

דוגמה נוספת, הפעם נשלח את הערך לקומפוננטה ע"י האירוע `blur` (כאשר תיבת הקלט TAB פוקוס):

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

@Component({
  selector: 'app-root',
  template: `
    <input #newName
      (blur)="addName(newName.value); newName.value=' ' ">
    <ul><li *ngFor="let x of nameList">{{x}}</li></ul>
  `})
export class AppComponent {
  nameList = ['Dan', 'Gil'];
  addName(val: string) {
    if(val){
      this.nameList.push(val);
    }
  }
}
```

כasher נרץ את הדוגמה נקבל:

- Dan
- Gil

ולאחר הקלדת שם ויציאה מאחור תיבת הקלט, השם החדש התווסף למערך, ללא צורך בלחיצה על כפתור:

- Dan
- Gil
- Li

Angular Event Cycle Hooks .5.

Directives and Components LifeCycle hooks .5.1

cutת נתיחה לתהליכי השינויים בחיו של הרכיב, ולאחרן בו נוכל לכתוב לוגיקה נוספת, בכל שלב בחיו של הרכיב. זהו שלב ה-LifeCycle hooks.

נתבונן בטבלה:

מטרה	שלב
זהו שלב האיתחול של הרכיב או ה- <u>directive</u> , directive, לאחר שהוכנסו ה- <u>տוקון</u> -ים שהוגדרו לו.	ngOnInit
זהו שלב שבו הרכיב מקבל את ה- <u>տוקון</u> -ים שהוגדרו לו או כאשר הם משתנים.	ngOnChanges
זהו שלב בו ניתן לבדוק שינויים שהתחתרית לא הבינה בהם בעצמה. הוא נקרא בכל פעם שהתחתרית מחפש שינויים ברכיב ומשיע לה להרחיב את הבדיקה.	ngDoCheck
זהו שלב הרישת אובייקט הרכיב או ה- <u>directive</u> . בשלב זה ננקה רישום ל- <u>event</u> -ים או רישומים ל- <u>observables</u> , על מנת למנוע דיליפת זיכרון.	ngOnDestroy

בשונה מ-component, ל-directive שלבים נוספים:

מטרה	שלב
זהו שלב שבו התשתית מכניסה את רכבי הבנים של הרכיב לתצוגה שלו.	ngAfterContentInit
זהו שלב שבו התשתית בודקת את ה- <u>bindings</u> של רכבי הבנים של הרכיב.	ngAfterContentChecked
זהו שלב שבו התשתית יוצרת את התצוגה של הרכיב.	ngAfterViewInit
זהו שלב שבו התשתית בודקת את ה- <u>bindings</u> של התצוגה של הרכיב.	ngAfterViewChecked

נתבונן על תיזמון האירועים בחיו של רכיב או directive:

מתי?	שלב
לפני OnInit ובכל פעם שמתבצע שינוי בקלטים של הרכיב.	ngOnChanges
לאחר הפעם הראשונה ש- <u>OnChanges</u> נזקרא.	ngOnInit
בכל פעם שהתחתרית מחפשת שינויים ברכיב.	ngDoCheck
לאחר שהתחתרית מכניסה את רכבי הבנים של הרכיב לתצוגה שלו.	ngAfterContentInit
לאחר שהתחתרית בודקת את ה- <u>bindings</u> של רכבי הבנים של הרכיב.	ngAfterContentChecked
לאחר שהתחתרית יוצרת את התצוגה של הרכיב.	ngAfterViewInit
לאחר שהתחתרית בודקת את ה- <u>bindings</u> של התצוגה של הרכיב.	ngAfterViewChecked
בדיוק לפני שאנגולר הורסת את הרכיב או ה- <u>directive</u> .	ngOnDestroy

Angular Directives .6

קיימים 2 סוגי Directives

1. מושנים את המראה וההתקנות של אלמנטים בדף - **Attribute Directives**
2. יוצרים או מסירים אלמנטים מה HTML - **Structural Directives**

Built in Attribute directives .6.1

משנים את המראה וההתקנות של האלמנטים בהם הם נכתבו.
לדוגמה, נסנה את העיצוב של הווין בעזרת directive של ngClass:

```
<div [ngClass]="{'a': x>3 , 'b': x<1 }"></div>
```

- כשה משתנה x קטן מ-2 אז הווין מקבל את b class
- כשה משתנה x גדול מ-3 אז הווין מקבל את a class

Built in Structural Directives .6.2

משנים את מבנה DOM על ידי הוספה או הסרה של אלמנטים.

נתבונן על שלושה structural Built-in directives עיקריים:

- NgIf
- NgSwitch
- NgFor

ביטויים אלו מסייעים לנו לעשות מניפולציה חכמה בתוך דף ה-html תוך שימוש בלוגיקה שנכתוב ב- class של הרכיב.

הנחיות אלו פועלות על ידי שימוש בתג <template> - זהו תג חדש ב- HTML5 אשר מיועד להחזק קוד תבניות.

אולם, מקום להשתמש בכל פעם ב- template נוכל להשתמש ב- * Syntax sugar *
כאשר אנו מוסיפים ל- directive קידומת של * אנו מסמנים שהאלמנט הנוכחי יחשב כ- template.

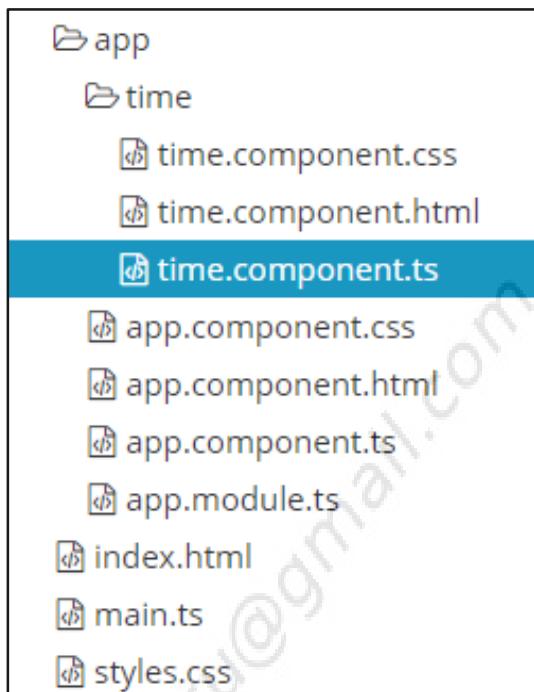
Nglf .6.2.1

תיאור: ifog בודק את הערך הבוליאני של מה שנמצא בתוך המרכאות שלו, ובהתאם מחליט האם להציג את האלמנט בו הוא נמצא (עם הערך יהיה שווה true), או לא להציג אותו (אם הערך יהיה שווה false) ifog יכול לקבל בתוך המרכאות שלו:

- משתנה מהקומפוננטה
- ערך מפורש (לדוגמא: *ngIf="false")
- ביטוי המחזיר ערך בוליאני

השימוש ב-ifog שונה מטכנית show/hide של אלמנט. במצב זה, כל תת-העץ אינו מופיע ב-HTML אם התנאי אינו מתקיים ולא רק מוסתר. בדף HTML גדולים יש לכך שימוש רב.

להלן דוגמה מלאה להמחשת השימוש ב-ifog* המכילה בתוך תיקית src את הקבצים הבאים:



שלב 1- בקובץ `app.component.ts` יוגדר סלקטור של `app-root` וכן בתוך המחלקה קיימים משתנים פונקציה:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  var1:string;
  var2:string="Shahar";
  var3:number=0;
  var4:number=100;
  var5:string=null;
  var6:boolean=true;
  var7:boolean=false;
  var8:number=NaN;

  constructor(){
    this.func();
  }

  func():void{
    setInterval(()=>{
      this.var3+=1;
    },1000)
  }
}
```

משתנים בرمת המחלקה.
המשתנים מכילים ערכים שוכרים חלקם מהערכים משמשות חלקם מהערכים משמשות `false` `true` בהמשך במשתנים האלו כדי נשתמש במשתנים אלו כדי לקובע אם אלמנט יוצג או לא יוצג

הבנייה של המחלקה מבצע קרייה לפונקציה ששívית למחלקה `func()`.

שימוש `setInterval` זה פונקציה שרצה שבו ושוב בפסק זמן שנקבעו לה- כאן קבוע שהזמן יהיה כל 1000 מילישניות. בכל פעם ש`setInterval` מתחבשת- היא מקדמת את תוכן המשתנה `var3` ב-1.

שימוש ל-`func`: קראם לפונקציה זו מהבנייה. אך מרגע יצירת הקומפוננטה הפונקציה הוא ריצה ללא הפסקה כל 1000 מילישניות

שלב 2- נוסיף בקובץ html שימוש ב`ng*` וכן אינטראקטיביות:

```
<div>
  <p *ngIf="">p1</p>

  <p *ngIf="0">p2</p>
  <p *ngIf="1">p3</p>
  <p *ngIf="-1">p4</p>

  <p *ngIf="false">p5</p>
  <p *ngIf="true">p6</p>

  <p *ngIf="'''">p7</p>
  <p *ngIf="''shahar'">p8</p>

  <p *ngIf="null">p9</p>
  <p *ngIf="undefined">p10</p>
  <p *ngIf="NaN">p11</p>

  <p *ngIf="var1">{{var1}}</p>
  <p *ngIf="var2">{{var2}}</p>
  <p *ngIf="var3">{{var3}}</p>
  <p *ngIf="var4">{{var4}}</p>
  <p *ngIf="var5">{{var5}}</p>
  <p *ngIf="var6">{{var6}}</p>
  <p *ngIf="var7">{{var7}}</p>
  <p *ngIf="var8">{{var8}}</p>

</div>
<app-time></app-time>
```

כל מה שרשום בין {} הוא שימוש
באינטרקטיביות של אングולר, והמשמשות
של הדבר היא שבתוך ה{} אוטו-
רשותים של של משתנה שנמצא
במחלקה AppComponent.
נשען ריצה בפועל יוצג בדף ה-
הערך העדכני שנמצא בתוך המשתנה

כאן יצרנו מופע של הקומפוננטה
TimeComponent שנמצאת תחת
התיקיה time שתחת התיקיה app

שלב 3- בקובץ `time.component.ts` ניצור את התוכן הבא:

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

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

  dateField: Date;
  secondField: number;

  constructor() {
    this.func();
  }

  func(): void {
    setInterval(() => {
      this.dateField = new Date();
      this.secondField = new Date().getSeconds();
    }, 1000);
  }
}
```

בתוך הבניין ביצענו קריאה לפונקציה `func` ששicityת למחלקה
`func()`:
`void`
`setInterval((()=>{`
`this.dateField=new Date();`
`this.secondField=new Date().getSeconds();`
`},1000)`
`)` בתרן () הוספנו `setInterval` שרע ללא הפסקה
 כל 1000 מילישניות, וمعدכן את המשתנים של
 המחלקה לתאריך העכשווי (`dateField` מכיל את
 התאריך הנוכחי, `secondField` מכיל את הערך
 של השניות של הזמן הנוכחי)

שלב 4- נבחן את `time.component.html` ונראה שיש שם שימוש ב`ifng*` וכן אינטראפולדציה:

```
<p *ngIf="secondField%10!=0">
  the time is:{{dateField}}
</p>
```

באן בtemplate ביצענו אפקט זהו שرك כאשר
 מספר השניות לא מתחלק בעשר, התאריך יוצג
 (באמצעות האינטראפולדציה `{{dateField}}` שתציג
 את תוכן המשתנה `dateField` שנמצא בתוך
 המחלקה `TimeComponent` (TimeComponent)

באמצעות דוגמא זו ראיינו שב`ifng*` אפשר לחת גם
 ביטוי שמחזיר ערך בוליאני, ולא רק שם משתנה או
 ערך בוליאני פשוט.

שלב 5- להלן התוצאה המופיעה בדף:

```
p3
p4
p6
p8
Shahar
2
100
true
the time is:Sat Dec 02 2017 22:16:53 GMT+0200 (Eastern Europe
Standard Time)
```

NgSwitch . 6.2.2

נשתמש ב-NgSwitch כאשר נרצה אלמנט אחד (או תת-עץ) מתוך קבוצת אלמנטים בתוך ה-DOM. בדומה ל-`NgIf`, רק האלמנט (או תת-עץ) המקיים את התנאי יתווסף לעץ.

```
<span [ngSwitch]="toChoice">
  <span *ngSwitchCase="'Eenie'">Eenie</span>
  <span *ngSwitchCase="'Meanie'">Meanie</span>
  <span *ngSwitchCase="'Miney'">Miney</span>
  <span *ngSwitchCase="'Moe'">Moe</span>
  <span *ngSwitchDefault>other</span>
</span>
```

בדוגמא זו, על פי המשתנה `toChoice`, יוחלט איזה חלק יוצג. במידה שמשתנה זה שווה למחוזת 'Eenie', רק ה-`span` המכיל את `ngSwitchCase` שווה ל-'Eenie' יהיה חלק מה-DOM.

נתבונן ב-3 חלקים של מבנה זה:

- `ngSwitch` – לחבר את המבנה למשתנה שעל פי יוחלט איזה אלמנט (או תת-עץ) יוצג.
- `ngSwitchCase` – לחבר כל אלמנט (או תת-עץ) לתנאי שלו.
- `ngSwitchDefault` – לחבר לאלמנט (או תת-עץ) שיוצג במידה שאף תנאי אינו מתקין.

NgFor . 6.2.3

תיאור: ngFor מקבל מערך, וע"י איטרטור עובר על כל תא המערך ומציג כל אחד מהם.

ngFor מבצע את השינויים המתאימים ל- DOM:

- כאשר פריט נוסף לערך, מופע חדש של התבנית מתווסף ל- DOM.
- כאשר פריט מוסר מהערך, מופע התבנית שלו מוסר מ- DOM.

לדוגמה, הקוד הבא בtemplate :

```
<h1>  
color list:  
</h1>  
<p *ngFor="let z of ['red','green','blue','yellow','magenta','black']">  
<span>{{z}}</span>  
</p>
```

ציג את התוצאה הבאה:

color list:

red
green
blue
yellow
magenta
black

בנוסף, נוכל להשתמש ב-`ngFor*` גם במערכות דו- מימדיים.

לדוגמה - נוסיף(component) את `property` הבא:

```
mat:Array<Array<string>>=[  
  ["Sefy", "Shahar", "Avi", "Ben"],  
  ["A", "B", "C", "D"]  
];
```

כעת, נבצע את האיתרציה על המטריצה באמצעות `ngFor*` בצורה הבאה:

```
<h1>
matrix:
</h1>
<table border="1">
  <tr *ngFor="let a of mat" >
    <td *ngFor="let b of a">{{b}}</td>
  </tr>
</table>

<ul>
  <li *ngFor="let c of mat">{{c}}</li>
</ul>
```

וכתוצאה נקבל בדף:

Sefy	Shahar	Avi	Ben
A	B	C	D

- Sefy,Shahar,Avi,Ben
- A,B,C,D

אפשרה נוספת המתאפשרת על ידי `ngFor*` היא לקבל בכל איטרציה את ה-`.index`.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  animalList=[
    {
      name:"cow",
      numOfLegs:4,
      color:"brown",
      isFlying:false,
      print:():string=>{return `hello caow`}
    },
    {
      name:"bird",
      numOfLegs:2,
      color:"black",
      isFlying:true,
      print:():string=>{return `hello bird`}
    }
  ];
}
```

ניצור את התוכן הבא בcontroller:

כעת, נבצע איטרציה על תאי המערך בתוך html template באמצעות `ngFor`, ונוסיף משתנה בשם `i` שיכיל בתוכו את האינדקס של התא הנוכחי בכל איטרציה:

```
<ul>
  <li *ngFor="let animal of animalList; let i=index">
    <h3>animal {{i}}</h3>
    <p>name: {{animal.name}}</p>
    <p>color: {{animal.color}}</p>
    <p>{{animal.print()}}</p>
  </li>
</ul>
```

כאן, נוכל להציג את ה-`Index` של החיה ברשימה:

- **animal 0**
 - name: cow
 - color: brown
 - hello caow

- **animal 1**
 - name: bird
 - color: black
 - hello bird

ה-`index` מתחילה מ-0 כמו בכל מערך



NgForTrackBy .6.2.4

אפשרות נוספת ב-`NgFor` הינה שימוש ב-`trackBy`. שימוש זה נועד על מנת לשפר את הביצועים של `For`, במיוחד כאשר מדובר במערכות גדולים.

הfonקציית `trackByName` הינה פונקציית מעקב. כך, שאם המערך השתנה, היא תעדכן רק את האלמנטים שהשתנו ולא את כל האלמנטים במערך.

```

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  animalList=[
    {
      name:"cow",
      numOfLegs:4,
      color:"brown",
      isFlying:false,
      print():string=>{return `hello caow`}
    },
    {
      name:"bird",
      numOfLegs:2,
      color:"black",
      isFlying:true,
      print():string=>{return `hello bird`}
    }
  ];

  trackByName(index:number, animal){
    return animal.name;
  }
}

```

כפי שניתן לראות בדוגמה לעיל, ביצענו מעקב על ידי ה-name הייחודי.

```

<ul>
  <li *ngFor="let animal of animalList; let i=index; trackBy:trackByName">
    <h3>animal {{i}}</h3>
    <p>name: {{animal.name}}</p>
    <p>color: {{animal.color}}</p>
    <p>{{animal.print()}}</p>
  </li>
</ul>

```

3. תרגילים

תרגיל 1



צרו את לוח הכלול מוצג בטבלה בעזרת מערך דו מימדי ו-`ngFor`

תרגיל 2

מצורף מערך אובייקטי `SONs`, עליום לבצע את הפעולות הבאות:

- צרו קומפוננטה בשם `pepoleInfo` בשם `pepoleInfo`
- צרו בתוך הקומפוננטה משתנה בשם `arr` ותחסנו בתוכו את מערך אובייקטי `SONs`
- צרו את חלק ה-HTML של הקומפוננטה כך שיציג רשימה מסווג `>ao>` ובה כל `<ao>` יציג מידע של אובייקט אחד ממערך `arr`

התוצאה הנדרשת היא:

- Id info:** 0
Name info: Carrillo Stewart
Gender info: male
Favorite Fruit info: melon
- Id info:** 1
Name info: Ramos Wall
Gender info: male
Favorite Fruit info: strawberry
- Id info:** 2
Name info: Figueroa Sampson
Gender info: male
Favorite Fruit info: mango
- Id info:** 3
Name info: Marsh Stanley
Gender info: male
Favorite Fruit info: melon
- Id info:** 4
Name info: Rush McMahon
Gender info: male
Favorite Fruit info: apple
- Id info:** 5
Name info: Downs Meyer
Gender info: male
Favorite Fruit info: strawberry
- Id info:** 6
Name info: Yesenia McGee
Gender info: female
Favorite Fruit info: mango
- Id info:** 7
Name info: Charlene Jacobs
Gender info: female
Favorite Fruit info: melon

להלן מערך NOSQL:

```
[  

  {  

    "id": 0,  

    "name": "Carrillo Stewart",  

    "gender": "male",  

    "favoriteFruit": "melon"  

  },  

  {  

    "id": 1,  

    "name": "Ramos Wall",  

    "gender": "male",  

    "favoriteFruit": "strawberry"  

  },  

  {  

    "id": 2,  

    "name": "Figueroa Sampson",  

    "gender": "male",  

    "favoriteFruit": "mango"  

  },  

  {  

    "id": 3,  

    "name": "Marsh Stanley",  

    "gender": "male",  

    "favoriteFruit": "melon"  

  },  

  {  

    "id": 4,  

    "name": "Rush Mcmahon",  

    "gender": "male",  

    "favoriteFruit": "apple"  

  },  

  {  

    "id": 5,  

    "name": "Downs Meyer",  

    "gender": "male",  

    "favoriteFruit": "strawberry"  

  },  

  {  

    "id": 6,  

    "name": "Yesenia McGee",  

    "gender": "female",  

    "favoriteFruit": "mango"  

  },  

  {  

    "id": 7,  

    "name": "Charlene Jacobs",  

    "gender": "female",  

    "favoriteFruit": "melon"  

  }  

]
```

תרגיל 3

להלן המערך הבא:

```
students= [
    {
        "id": 0,
        "name": "Gamble Foster",
        "age": 27,
        "email": "gamblefoster@moreganic.com",
        "phone": "+1 (804) 469-2701",
        "address": "662 Gerry Street, Torboy, West Virginia, 6142",
        "avg_grade": 74
    },
    {
        "id": 1,
        "name": "Dona Todd",
        "age": 38,
        "email": "donatodd@moreganic.com",
        "phone": "+1 (878) 412-2366",
        "address": "339 Montrose Avenue, Wyano, Pennsylvania, 6854",
        "avg_grade": 90
    },
    {
        "id": 2,
        "name": "Welch Cooke",
        "age": 32,
        "email": "welchcooke@moreganic.com",
        "phone": "+1 (812) 574-2702",
        "address": "568 Evergreen Avenue, Yardville, Minnesota, 2420",
        "avg_grade": 96
    },
    {
        "id": 3,
        "name": "Hickman Petersen",
        "age": 33,
        "email": "hickmanpetersen@moreganic.com",
        "phone": "+1 (933) 422-3102",
        "address": "854 Preston Court, Lutsen, Washington, 5807",
        "avg_grade": 91
    },
    {
        "id": 4,
        "name": "Lilly Ochoa",
        "age": 22,
        "email": "lillyochoa@moreganic.com",
        "phone": "+1 (861) 479-2248",
        "address": "443 Greene Avenue, Idamay, Virgin Islands, 6364",
        "avg_grade": 93
    },
    {
        "id": 5,
        "name": "Campbell Calhoun",
        "age": 26,
        "email": "campbellcalhoun@moreganic.com",
        "phone": "+1 (875) 419-3783",
        "address": "578 Aitken Place, Valle, Oregon, 8222",
        "avg_grade": 89
    },
    {
        "id": 6,
        "name": "Ethel Patrick",
        "age": 26,
        "email": "ethelpatrick@moreganic.com",
        "phone": "+1 (924) 466-3077",
    }
]
```

```
"address": "794 Bunker Street, Brandermill, Texas, 9029",
"avg_grade": 100
},
{
"id": 7,
"name": "Benjamin House",
"age": 23,
"email": "benjaminhouse@moreganic.com",
"phone": "+1 (837) 584-2164",
"address": "372 Heath Place, Bladensburg, Colorado, 5430",
"avg_grade": 71
},
{
"id": 8,
"name": "Acosta Bridges",
"age": 36,
"email": "acostabridges@moreganic.com",
"phone": "+1 (891) 462-3863",
"address": "357 Calder Place, Lacombe, Arkansas, 3942",
"avg_grade": 90
},
{
"id": 9,
"name": "French Cote",
"age": 33,
"email": "frenchcote@moreganic.com",
"phone": "+1 (954) 551-2675",
"address": "906 Reed Street, Ernsville, Connecticut, 3935",
"avg_grade": 74
},
{
"id": 10,
"name": "Cruz May",
"age": 20,
"email": "cruzmay@moreganic.com",
"phone": "+1 (971) 493-3391",
"address": "191 Court Street, Onton, Alaska, 1689",
"avg_grade": 97
},
{
"id": 11,
"name": "Barton Bruce",
"age": 25,
"email": "bartonbruce@moreganic.com",
"phone": "+1 (947) 452-2091",
"address": "575 Dictum Court, Cresaptown, District Of Columbia, 3362",
"avg_grade": 85
},
{
"id": 12,
"name": "Bernice Brady",
"age": 33,
"email": "bernicebrady@moreganic.com",
"phone": "+1 (919) 513-2260",
"address": "417 Chauncey Street, Babb, New Hampshire, 7991",
"avg_grade": 77
},
{
"id": 13,
"name": "Hartman Pollard",
"age": 26,
"email": "hartmanpollard@moreganic.com",
"phone": "+1 (851) 462-2401",
"address": "126 Hill Street, Cetronia, Hawaii, 261",
"avg_grade": 82
},
```

```

    "id": 14,
    "name": "Queen Watkins",
    "age": 23,
    "email": "queenwatkins@moreganic.com",
    "phone": "+1 (937) 548-3674",
    "address": "344 Vermont Court, Garberville, Delaware, 516",
    "avg_grade": 87
}
]
  
```

- צרו בעוזרת אנגולר קומפוננטות מתאימות, כך שהມידע שבמערך הנ"ל, יוצג בתורה טבלאית:

filter options:
all students ▾

STUDENTS INFORMATION

id	name	age	email	phone	address	avg_grade
0	Gamble Foster	27	gamblefoster@moreganic.com	+1 (804) 469-2701	662 Gerry Street, Torboy, West Virginia, 6142	74
1	Dona Todd	38	donatodd@moreganic.com	+1 (878) 412-2366	339 Montrose Avenue, Wyano, Pennsylvania, 6854	90
2	Welch	66	lucywelch@moreganic.com	+1 (812)	568 Evergreen Avenue,	66

- בראש העמוד יש ליצור דרפ-דאון המכיל את אפשרויות הפילטור הבאות:
(בכל בחירה של הלקוח באופציה מסוימת מהדרפ-דאון, יש להציג בטבלה רק את הסטודנטים שעומדים בדרישת הפילטור)

filter options:
all students ▾

id between 1-6
age is bigger than 30
avg grade over 90
phone number ends with 4
name starts with b

STUDENTS INFORMATION

		age	email
0	Gamble Foster	27	gamblefoster@moreganic.com

The Pipe Operator (|) .7.1

האופרטור (|), הנקרא גם Pipe, הינו אופרטור שימושי מאוד ב-Angular. האופרטור זה, הינו לשם טרנספורמציה לערך או קבוצת ערכים. למשל, אם נרצה להציג מספר מסוים כתיקסט או לכפות שהtekst יהיה upper case או להציג רק אלמנטים המקיימים תנאי מסוים בתוך מערך.

ישנם מספר pipes מוכנים מראש ב-Angular כגון:

- DatePipe – מאפשר להציג תאריך בפורמט הרצוי.
- CurrencyPipe – מאפשר להציג מטבע בפורמט הרצוי.
- LowerCasePipe – מאפשר להציג מחזורת C-case.
- UpperCasePipe – מאפשר להציג מחזורת C-case.

לדוגמא:

```
<div>Name through uppercase pipe: {{name | uppercase}}</div>
          ↑
          → Name through uppercase pipe: SHAI VASHDI
```

דוגמאות נוספות:

```

<div class="pipe">
  <h2>Date Pipe</h2>
  <div>
    {{date | date:'yMMMMEEEEd'}}</div>
  <h2>Number Pipe</h2>
  <div>
    {{4.566 | number:'1.2-2'}}</div>
  <h2>Currency Pipe</h2>
  <div>
    {{15.99 | currency:'EUR':true:'1.0-0'}}</div>
  <h2>Stateful Pipe</h2>
  <div>
    {{randomData | async}}</div>
  </div>
</div>

<div>Name through uppercase pipe: {{name | uppercase}}</div>

```

דוגמא:

והתוצאה:

Date Pipe

Tuesday, August 30, 2016

Number Pipe

4.57

Currency Pipe

€16

Stateful Pipe

Random data!

דוגמא זו מכילה: Date pipe שמאפשר קביעת תאריך על פי פורמט, Number pipe שמציג 2 ספרות לאחר הנקודה, Currency pipe – שמציג מחיר ב יורו, ו- Stateful pipe שמאפשר הצגת נתונים במצבה א-סינכורנית, ולמעשה מחייב לערוך המגע מהפונקציה:

```

randomData = new Promise((resolve, reject) => {
  setTimeout(() => resolve('Random data!'), 1000);
});
```

Custom Pipe .7.2

ניתן לבנות Pipe כך' ש諾ול להגדיר בעצממו את מהות השינוי שיתבצע.

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

@Pipe({name: 'exponential'})

export class ExponentialPipe implements PipeTransform {
  transform(value: number, exponent: string): number {
    let exp = parseFloat(exponent);
    return Math.pow(value, isNaN(exp) ? 1 : exp);
  }
}
```

לדוגמה נכתב pipe ExponentialPipe בשם שמחש את התוצאה של חזקה y של איבר x כלשהו:

```
import {ExponentialPipe} from './customPipe';

@Component({
  selector: 'home',
  styleUrls: ['./home.css'],
  templateUrl: './home.html',
  pipes: [ExponentialPipe],
})

export class Home {
```

ראשית, נכתב את class ה-pipe, שמממן את ה-PipeTransform:interface. ה-interface מחייב למש את הפונקציה transform, אשר מגדרה את השינוי. במקרה שינוי הערך הוא חישוב התוצאה של החזקה exponent של ה-value. נודא שהגדירנו את שם ה-pipe ב-@Pipe:decorator. בהתאם@gurushopping הגדרת הרכיב נצהיר על השימוש בpipe זה:

```
<p>exponentialt: {{2 | exponential: 10}}</p>
```

לאחר מכן יוכל להשתמש בו ב- html template:

exponentialt: 1024

והתוצאה:

תרגיל

נתון מערך `movies` המציג חמישה סרטים כולל המחיר שלהם בדולר, ותאריך בפורמט `yyyy\mm\dd`:



```
[{
    "id": 1,
    "movie_name": "Coraline",
    "price": "7.13",
    "movie_date": "7/9/2017"
}, {
    "id": 2,
    "movie_name": "Finding Dory",
    "price": "10.09",
    "movie_date": "12/12/2016"
}, {
    "id": 3,
    "movie_name": "The Wailing",
    "price": "10.69",
    "movie_date": "3/9/2017"
}, {
    "id": 4,
    "movie_name": "The Void",
    "price": "8.85",
    "movie_date": "10/30/2017"
}, {
    "id": 5,
    "movie_name": "Tower",
    "price": "8.47",
    "movie_date": "4/6/2017"
}]
```

להלן הנחות עבודה:

- מחיר בשקלים מחושב לפי דולר ביחס של פי 4
- תאריך ישראלי מוצג בפורמט `yyyy\mm\dd`
- תאריך אמריקאי מוצג בפורמט `mm\dd\yyyy`

צרו בעזרה אנגלור קומפוננטה המציגת למשתמש רשימה של שמות הסרטים בתצוגה הבאה:

movie list

1	Coraline
2	Finding Dory
3	The Wailing
4	The Void
5	Tower

בכל לחיצה על סרט, פרטיו יופיעו למטה, כפי שהם מופיעים בתמונה המסך שלහלן: (יש להשתמש באירוע אנגולר)

movie list

1 Coraline

2 Finding Dory

3 The Wailing

4 The Void

5 Tower

selected movie: Coraline

Country:	price:	Date:
USA	\$7.13	9/7/2017
IL	₪28.52	7/9/2017

Components Communication .8

@Input and @Output .8.1

נניח שברצוננו ליצור את האפליקציה הבאה:

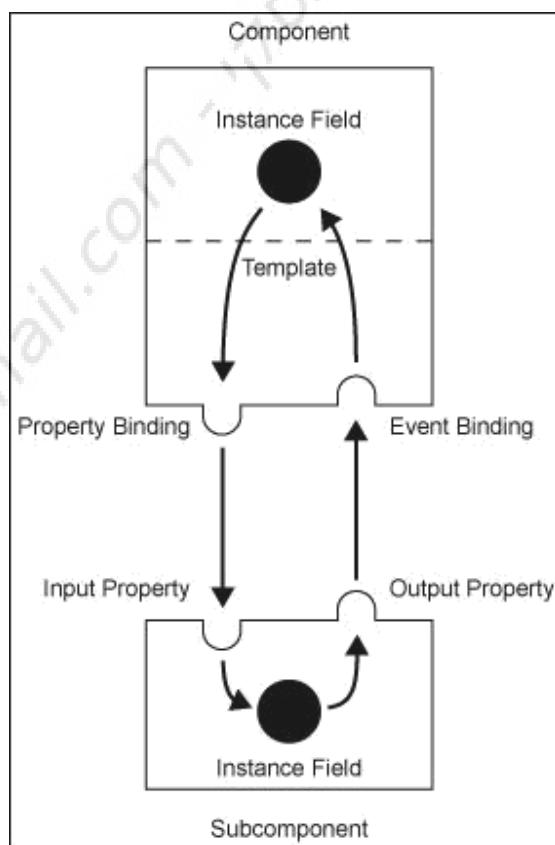
- קומפוננטה שבה הלקוח יכול לבחור את הצבע המועדף עליו מרשימה צבעים קיימת
- קומפוננטה שבה מוצג ללקוח הצבע המועדף שבחר על ידי קומפוננטה מס' 1

בכדי להעביר את המידע שהוזן בקומפוננטה 1 אל קומפוננטה 2, علينا ליצור "ערוץ תקשורת" בין הקומפוננטות.

אפשרית אחת להשגת המטרה היא ליצור קומפוננטת "הוראה" משותפת לקומפוננטות האלו, ודרך ההורה המשותף הן יכולים להעביר ביניהם את המידע בצורה הבאה:

קומפוננטה 1 תעליה אירע שיתפס בקומפוננטת האב - בכל פעם שהלקוח בוחר צבע.
קומפוננטה 2 תקבל את הצבע שהלקוח בחר מקומפוננטת האב.

קומפוננטת ההוראה כוללת ב-template שלה מופיע של קומפוננטה 1 ומופיע של קומפוננטה 2.



שלב 1 – ייצרת קומפוננטה בן ראשונה

עת נראה כיצד ליצור event מקומפוננטת הילד, ואיך מתבצע תהליך הרישום לארוע מקומפוננטת האב. לצורך כך נבין מהו Event Emmiter :Event Emmiter custom event .custom event

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

@Component({
  selector: 'app-child1',
  template: `
    <div *ngFor="let x of ['red','green','blue','yellow']">
      <button (click)="onColorClick(x)">{{x}}</button>
    </div>
  `)
export class Child1Component {
  @Output() changeColor:EventEmitter<string>=new EventEmitter<string>();

  onColorClick(color:string):void{
    this.changeColor.emit(color);
  }
}
```

רכיב שרצואה לשילוח
לקומפוננטת event
האב, מגדר Event
Emmiter וחוושף
אתו כ-
@output

- בתגובה להקלקה על הceptor, מופעל האירוע שמבצע את האירוע emit באמצעות .changeColor
- לשם כך אנו משתמשים ב-EventEmitter שמאתחל את האירוע addCar וublisher לו את הסוג שהוא צריך לקבל (הצורה הכללית של האובייקט) - למשל, הגדרנו Event Emmiter ששולח event עם ערך מחרוזתי (string) והוא נקרא changeColor (string).
- ה-decorator @Output () משמש כדי לאפשר לקומפוננטה של ההורה להאזין להפעלת האירוע.
- את הפעמטר אותו אנו מעבירים ב-emit נוכל לקבל בקומפוננטת ההורה בזמן הרישום לארוע \$event באמצעות event

שלב 2 – ייצרת קומפוננטה בן שנייה

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

@Component({
  selector: 'app-child2',
  template: `
    <p>Selected color:<span>{{selectedColor}}</span></p>
  `)
export class Child2Component {
  @Input() selectedColor:string;
}
```

צריך להזכיר שהגדרת property את
ה@Input() decorator כדי לאפשר את
יבוא המידע מההורה יליד.
את Input מייבאים מangular/core,
צורת שליחת המידע מkomponentot ההורה,
לקומפוננטה היזן, הוא ע"י שליחת directive
עם שם זהה לשם input שהגדרנו כאן.

שלב 3 – יצירת קומponentה הוראה

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

@Component({
  selector: 'app-root',
  template: `
    <app-child1 (changeColor)="onColorUpdate($event)"></app-child1>
    <app-child2 [selectedColor]="currentColor"></app-child2>
  `,
})
export class AppComponent {
  currentColor:string;

  onColorUpdate(color:string):void{
    this.currentColor=color;
  }
}
```

בהגדרת המופיע של קומponentה 1 בתוך קומponentה האב, נרשם לארוע שישלח event emitter מקומponentה הבן. כאשר מטפלים ב- event , מקבלים את event העורר שהינו הצבע שנבחר, נאחסן אותו בתוך משתנה currentColor . שהוא נשלח בתוך @input :לקומponentה 2:

תרגיל



- צרו את האתר הבא בעזרת אングולר. הוראות:
- קומponentה אחת תקלוט מהמשתמש את שמו – ותשמר את השם כאשר ילחץ על הכפתור
 - קומponentה שנייה תציג למשתמש את השם שהזין

The screenshot shows a two-component Angular application. The top component is a form with a title "Change your name:" and a text input field containing "Anna". Below the input is a green "submit" button. The bottom component is a display box showing the text "Your name is: Anna". To the right of the display box, there are two red labels: "קומponentה ראשונה" above the form and "קומponentה שנייה" above the display.

Angular Services .9

Component Constructor .9.1

בדומה לשפטות אחרות, ה-constructor הינו פונקציה המתחילה אובייקט, ונקראת באופן אוטומטי מיד לאחר שהאובייקט נוצר.

בדרכן כלל, מגדירים סוגים שונים של constructors כאשר חלקם מקבלים פרמטרים הרלוונטייםiae לאיתחול האובייקט. כגון: שם האדם לאובייקט מסווג person.

ב-Angular לא מעבירים פרמטרים לרכיבים בדרך המקובלת, אלא באמצעות מנגןון dependency injection.

```
constructor(private router: Router, private activatedRoute: ActivatedRoute,
           private productsService: ProductsService,
           private actionsService: ActionsService)
```

ניתן לראות כי רכיב זה צריך שירותים שונים, כגון: router , products service, action service,ect. באמצעות dependency injection.

דברנו עד כה בכלויות על מנגןון dependency injection. במאור של חוברת זו,icut נדבר על כך בהרחבה.

Dependency Injection .9.1.1

הזרקת תלויות (באנגלית: dependency injection), הינה תבנית עיצוב, המאפשרת בחירה של רכיבי תוכנה, בזמן ריצה (ולא בזמן ההידור). תבנית זו יכולה, לדוגמה, לשמש כדרך פשוטה לטעינה דינמית של `sinusoid` או בחירה באובייקטי דמה (mock objects) בסביבות בדיקה, במקומם להשתמש באובייקטים אמיתיים של סביבת הייצור. תבנית עיצוב זו, מזריקה את האלמנט שתלויים בו (אובייקט או ערוץ, וכדומה) אל היעד שלו, בהתאם על ידיעה של צרכי היעד.

מנגןון dependency injection ב-Angular, מרכיב משלושה של שלושה חלקים עיקריים:

- Injector – הרכיב שמייצר את ה-instance של האובייקט המזמין.
- Provider – הרכיב שMagevir כיצד לייצר את האובייקט. למעשה, ממפה את הדרישה לייצור האובייקט אל פונקציית factory, המיצרת אותו בפועל.
- Dependency – סוג האובייקט אותו מבקשים לייצר בדרך זו.

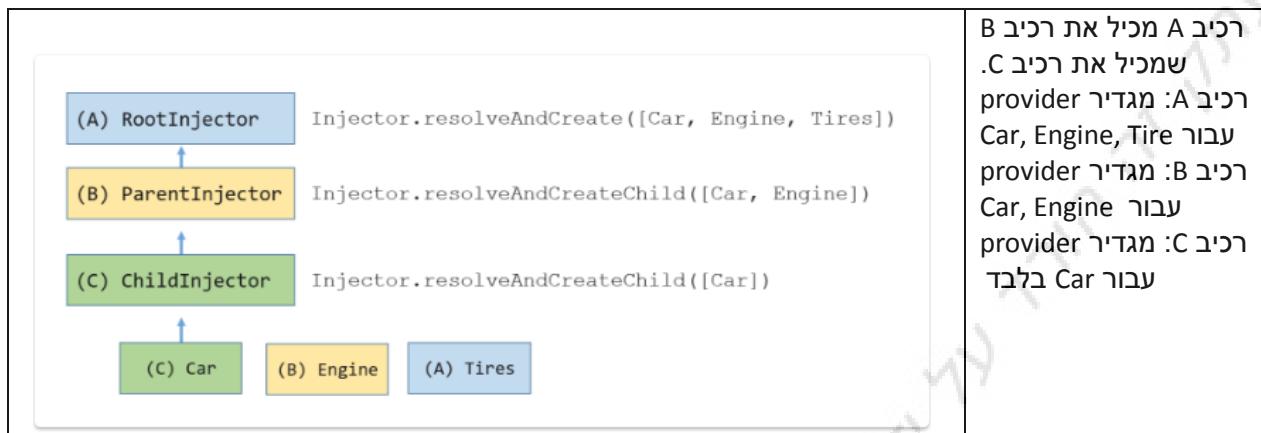
שלבי הביצוע:

<code>import ...</code>	1. ראשית יש לסמן את ה- Type של האובייקט המיועד כ- Injectable
<code>@Component({ selector: 'home', styleUrls: ['./home.style.css'], templateUrl: './home.template.html', directives: [ActionBarControlComponent], providers: [ActionsService, ProductsService], }) export class Home { }</code>	2. נגדיר כיצד ליצר את האובייקט (במקרה זה ללא פרמטרים) באמצעות ה- provider
<code>export class ProductEditModuleComponent implements OnInit, OnDestroy{ public selectedProductId: number = NaN; public product: IProduct = null; paramsSub: Subscription; actionsSub: Subscription; constructor(private router: Router, private activatedRoute: ActivatedRoute, private productsService: ProductsService, private actionsService: ActionsService) { } }</code>	3. נגדיר היכן משתמשים באובייקט באמצעות ה- constructor

האובייקט המזורק הוא singleton scope בתוך ההגדרה שלו, מודיע? כדי לענות על כך נצטרך להבין כיצד מנגנון ה- injection עובד.

The Injector Tree .9.1.2

אפליקציית Angular הינה, למעשה, עצם של רכיבים. לכל רכיב בעץ, קיים Injector (וירטואלי) משלה. בדרך זו, עצם הרכיבים מקביל לעץ ה-Injector -ים. בכל שלב, כאשר רכיב כלשהו צריך Dependency מסוים, המנגנון מטפס במעלה העץ ומחפש את ה-provider, המaddir רכיב זה. במקרה זה, הוא יפעיל את ה-Injector של אותו רכיב ויקבל את ה- instance עבור האובייקט המבוקש. נתבונן בדוגמה הבאה:



כאשר רכיב A יקבל את ה- Dependency של Tire, המנגנון יעלה במעלה העץ ויחפש provider עבור Tire. הוא ימצא אותו ברכיב A ולכן יספק לו את אותו instance של רכיב A.

כאשר רכיב A יקבל את ה- Dependency של Engine, המנגנון יעלה במעלה העץ ויחפש provider עבור Engine. הוא ימצא אותו ברכיב B ולכן יספק לו את אותו instance של רכיב B.

כאשר רכיב A יקבל את ה- Dependency של Car הוא יקבל instance ייחודי משלו, השונה מה- instance של הרכיבים B ו-C.

Services .9.2

אפליקציות מורכבות ממספר תת מערכות שונות למשל: logging, data access, caching וכו'.

המושג של Service ב-Angular הינו כיווס (encapsulation) של פונקציונליות כלשהי על מנת שתוכל לספק את הפונקציונליות זו בצורה בלתי תלולה לשאר חלק האפליקציה.

לכן כאשר מספר רכיבים משתמשים בפונקציונליות מסוימת זהה במקום להעתיק את הפונקציונליות זו מספק פעמים נכתוב אותו פעם אחת בתור service.

ונכל בדרך זו להימנע משיכפול קוד ואך משיכפול ה-instance על ידי הזרקת ה-service לרכיב.

הפרדה זו של לוגיקה מהרכיב, מאפשרת להתרכר ב כתיבת הלוגיקה הרלוונטיות לתצוגתו של הרכיב בלבד, ללא שיקול לוגיקה נוספים.

בנוסף, קל לבדוק את הרכיב, כאשר הוא מוגדר בפני עצמו, על ידי כתיבת mock service המדממת את ה-.service

התבנית הבסיסית ביותר להוספת service לפרויקט אングולר, היוצר מופע אחד של ה-service עבור עליון קומponentות, היא התבנית הבא:

שלב 1:

```
import { Injectable } from '@angular/core';
@Injectable()
export class XXService {
}
```

את השם
נחלף בכל פעם לשם
הרצוי, אך תמיד נוסיף
סימנת Service

יש לשים לב שעבור ייצרת service נבצע import של מנת שnochol להגדר מעל ה-class service עבורי את הדקורייטיב של @injectable, ובכך לאפשר שימוש במנגנון הווים בתוך ה-.service

שלב 2:

הווספה ה-service ל-app.module.ts מערך providers של ה-class service

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

import { AppComponent } from './app.component';
import { XXService } from './app.xx.service';

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

שלב 3:

השימוש ב-service בתוך הקומפוננטה יעשה ע"י ביצוע import של ה-service, ויצירת משתנה מטיפוס ה-service בסוגרים של ה-constructor.

שימוש לב שלא מתבצע שום אתחול ע"י new ליצירת מופע של ה-service, מכיוון שמדובר על שימוש באובייקט ה-singleton שנוצר עבור כל האפליקציה, ולא נוצר אובייקט חדש של ה-service זהה במיוחד עבור הקומפוננטה הנוכחית.

```

import { Component, OnInit } from '@angular/core';
import { XXService } from './app.xx.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{

  constructor(private myX :XXService){}

  ngOnInit(){
  }
}

```

service אשר מקום ההצהרה עליו הוא במרק h-providers שבמודול, נוצר פעמי אחת בלבד עבור כל הקומפוננטות, וכך כל שינוי שיבוצע ב-service על ידי קומפוננטה מסוימת, ישתקף גם בקומפוננטות האחרות.



נראה כעת דוגמא פשוטה בה הוספנו שני משתנים ל-service והצגנו אותם ב-Component :

שלב 1:

צרנו את קובץ service :

```

import { Injectable } from '@angular/core';

@Injectable()
export class XXService {

  personName:any="Anna Karp";
  person:string={name:"Li Dor"};
}

```

שלב 2:

הוספה ה-service ל-`app.module.ts` import "xxService", והוספה ה-class של ה-service ל-`providers`

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

import { AppComponent } from './app.component';
import { XXService } from './app.xx.service';

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

שלב 3:

השימוש ב-service בתוכן הקומפוננטה ע"י ביצוע import של ה-service, ויצירת משתנה מטיפוס ה-service בסוגרים של ה-service.`.constructor`

וכן בתוכן הפונקציה `ngOnInit`, הוספנו אתחול ערכים של משתנים מקומיים השייכים לקומפוננטה, ע"י גישה לקריאת הערכים של משתני ה-service והעתיקתם לתוכן המשתני הקומפוננטה.

```
import { Component, OnInit } from '@angular/core';
import { XXService } from './app.xx.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{
  constructor(private myX :XXService){}
  personName:any;
  person:string;
  ngOnInit(){
    this.personName=this.myX.personName;
    this.person=this.myX.person;
  }
}
```

שלב 4:

ניצור את ה-`html template`, ובו עזרת אינטראפלציה נציג את שני המשתנים שהגדכנו בקומפוננטה:

```
<p>{{personName}}</p>
<p>{{person.name}}</p>
```

שלב 5:

התוצאה שנקלבל בדף היא:

Anna Karp

Li Dor

בפונקציה `itOnInit` בצענו העתקה של הערכים המקוריים בסרוויס `service` לתוכן משתנה מקומי של הקומפוננטה, ואת המשתנה המקומי של הקומפוננטה הצגנו בתוך `template.html`.
 עלינו לזכור המשתנה פרימיטיבי (`Boolean / number / string`) מועתקים למשתנה אחר ע"י העתקה `by value`, ולכן מרגע שביצענו בתוך `itOnInit` את העתקת הערך של משתנה ה `string` בהשיג לאל משתנה מקומי של הקומפוננטה, כל שינוי שנבצע על המשתנה מסוג `string` בתוך `service` לא ישתקף לנו בקומפוננטה.
 אבל – משתנה מתייחס `byRef` (כגון: אובייקטים של `class`, או אובייקטי `host` וכו'), מועתקים רק ע"י העתקת המצביע לאוטו אובייקט, ולכן כל שינוי על האובייקט השיג לאל `service` ישתקף גם באובייקט של הקומפוננטה, מכיוון ששניהם מצביעים לאוטו אובייקט.



נפשת את האמור לעיל, על ידי שימוש בדוגמה הקודמת.
 נסיף ל-`service` פונקציה שתשנה את תוכן המשתנים של ה-`service`.

שלב 1:

יצרנו את קובץ ה-`service`:

```
import { Injectable } from '@angular/core';
@Injectable()
export class XXService {

  personName:any="Anna Karp";
  person:string={name:"Li Dor"};

  changeName(){
    this.personName="Roy";
    this.person.name="Loren";
  }
}
```

שלב 2:

הוספה ה-service ל-service-class של ה-app.module import "app.module" service

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

import { AppComponent } from './app.component';
import { XXService } from './app.xx.service';

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

שלב 3:

השימוש ב-service בתוך הקומפוננטה ע"י ביצוע import של ה-service, ויצירת משתנה מטיפוס ה-service בסוגרים של ה-constructor.

וכן בתוך הפונקציה ngOnInit, הוספנו אתחול ערכים של משתנים מקומיים השיכים לקומפוננטה, ע"י גישה לקריאת הערכים של משתני ה-service והעתיקתם לתוך משתני הקומפוננטה.

בנוסף, הוספנו לקומפוננטה פונקציה בשם setName שtaboezע ע"י האירוע click שנוסיף לכפתור שניצוץ ב-`html template`, הפונקציה setName פונה ל-service和服务ה בו את הפונקציה changeName שמשנה ב-service את ערכי המשתנים של ה-service:

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

import { XXService } from './app.xx.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{
  constructor(private myX :XXService){}

  personName:any;
  person:string;

  ngOnInit(){
    this.personName=this.myX.personName;
    this.person=this.myX.person;
  }

  setName(){
    this.myX.changeName();
  }
}
```

שלב 4:

ניצור את ה-template.html, ובו באמצעות אינטראפולדיצה נציג את שני המשתנים שהגדכנו בקומפוננטה, וכך נסיף כפטור ונדיר בו באירוע (click) קרייה לפונקציה setName שנותצת בתוך הקומפוננטה:

```
<p>{{personName}}</p>
<p>{{person.name}}</p>

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

שלב 5:

התוצאה שנקלע בדף היא:

Anna Karp

Li Dor

click me

וכאשר נלחץ על הכפתור נראה שנקלע את התוצאה הבאה:

Anna Karp

Loren

click me

כלומר – כפי שהסביר לעיל, משתנה פרימיטיבי מטיפוס string לא התעדכן, ואילו משתנה by ref מטיפוס אובייקט, ידע להציג את השינוי שהתבצע באובייקט ה-service מכיוון שהוא מצביע בדיק לאותו אובייקט עליו משתנה ה-service מצביע.

Promises .9.3

בדרך כלל, המ א-סינכרוניים. لكن, ע"מ להבין טוב יותר את הנושא נדרש להבין מספר מושגים א-סינכרוניים בתשתיית כגון: Promises.

השימוש ב-promises מקל לכתוב קוד א-סינכרוני בהשווואה ל-callbacks.
Promises – הוא הבטחה לקבל משהו, במועד מסוים.

עובדים בצורה הבאה:

- נרשמים להבטחה כלשהי.

- עושים עבודה מסוימת באופן א-סינכרוני.

- ולבסוף התוצאה מגיעה לפונקציה המטפלת עם תשובה: הצלחה או כישלון.



נגיד הבטחה מסוימת:

```

private peopleData = PEOPLE;

getPeople(): Promise<any[]> {
    return new Promise<any[]>((resolve, reject) => this.getPeopleResolver(resolve, reject));
}

private getPeopleResolver(resolve, reject) {
    setTimeout(() => {
        if (this.peopleData == null) {
            reject('unable to get the data');
            return;
        }
        resolve([...this.peopleData]);
    }, 1000);
}
  
```

הגדרנו הבטחה שתתקיים לאחר 1000 מיל-שניות להחזיר מערך של `people`.
במידה שהמערך שווה ל-`null` נחזיר דחיה (משמע שגיאה).
הכתיבה מתבצעת תוך הגדרת הערך המוחזר, במקרה זה `[any[]]` מערך שלא אלמנטים מסווג לא ידוע.
לאחר מכן נכתבת הפונקציה אשר תבצע את הבדיקה.

הטיפול ב-`Promise` הוא די פשוט:

```

        this._service.getPeople().then(
            /* success function */
            (peopleList) => {
                this.people = peopleList;
            }).catch(
            /* error function */
            (reason) => {
                console.log("PeopleService-getPeople Error:" + reason);
            });
    
```

פונים לפונקציה שמחזירה את ההבטחה וכאשר היא מתקיימת (then) מגדירים מה לעשות במקרה של הצלחה (עם הערך) או כישלון (עם הסיבה).

עת, נוכל לכתוב service לדוגמא:

```

/** Created by Shai Vashdi on 30/08/2016. ...*/

import {Injectable} from '@angular/core';

const PEOPLE =
[
    [
        {name: 'Shai' , phone: '050-1111111'},
        {name: 'Dani' , phone: '050-1111112'},
        {name: 'Shimon' , phone: '050-1111113'},
        {name: 'Ania' , phone: '050-1111114'},
        {name: 'Erick' , phone: '050-1111115'}
    ];
}

@Injectable()
export class PeopleService {

    private peopleData = PEOPLE;

    getPeople(): Promise<any[]> {
        return new Promise<any[]>((resolve, reject) => this.getPeopleResolver(resolve, reject));
    }

    private getPeopleResolver(resolve, reject) {
        setTimeout(() => {
            if (this.peopleData == null) {
                reject('unable to get the data');
                return;
            }
            resolve([...this.peopleData]);
        }, 1000);
    }
}

```

זהו service, אשר מחזיר לאחר 1000 ملي-שניות, מערך קבוע של people. ה-service מוגדר כ-.dependency injection, ולכן מקיים את תנאי ה-.dependency injection
כיצד נשתמש בשירות?

```
@Component({
  selector: 'about',
  styleUrls: ['./about.css'],
  templateUrl: './about.html',
  providers: [PeopleService],
  directives: [PeopleListControlComponent],
})
})
```

ראשית נגיד' provider
על מנת שנוכל
להזrik אותו
באמצעות
dependency
.injection

```
export class About {

  constructor(private _service: PeopleService)
  {
    this._service.getPeople().then(
      /* success function */
      (peopleList) => {
        this.people = peopleList;
      }).catch(
      /* error function */
      (reason) => {
        console.log("PeopleService-getPeople Error:" + reason);
      });
  }
}
```

לאחר מכן נשימוש
בשירות בדרך זהה
לשימוש ב-
:promise

Async – Observables .9.4

בדומה ל-promise, ישנה ספרייהצד שלישית של RxJS ("Reactive Extensions"), שאומצת על ידי Angular, המספקת פתרונות א-סינכרוניים.

ברוב המדריכים והדוגמאות של Angular, מותקן השימוש ב-RxJS על ידי npm, ו-Angular תומכת בכך, באמצעות system.js.

זאת, משום שה-client Http הנפוץ, עובד עם RxJS observables ללקחה את השימוש צעד אחד קדימה, על מנת להקל את השימוש ב-observables.

ההבדלים בין promises ל-observables

- promise מחזיר callback אחד ויחיד עם תוצאה של הצלחה או כישלון.
- observable מחזיר זרם של מידע ממשען מספר לא קבוע של callbacks.
- observable ניתן לביטול ולא רק כישלון.

הערה: לרוב נועד observable מכיוון שהם מספקים לנו יותר אפשרויות של החזרת מידע.

לדוגמה:

```
public People : Observable<any> = new Observable(observer => {
    setTimeout(() => {
        if(this.peopleData == null)
            observer.error('unable to get the data');
        else
            observer.next([...this.peopleData]);
    }, 1000);

    setTimeout(() => {
        if(this.extraObj == null)
            observer.error('unable to get the data');
        else
            observer.next([this.extraObj]);
    }, 2000);

    setTimeout(() => {
        observer.complete();
    }, 3000);
});
```

ולאחר 1000 מיל-שניות הוא מוחזיר רישימה נסotta. לבסוף, הוא מודיע על סיום.

טיפול ב-observable הוא שונה. לדוגמה:

```
this._service.People.subscribe(  
    values => this.people = this.people.concat(values),  
    error => console.log("PeopleService-getPeople Error:" + error),  
    () => console.log("PeopleService-People request Complete.")  
);
```

הסוגרים () בסוף הטיפול מצינו מה לעשות ברגע שהקريا הושלמה. כללי נפסיק רישום זה רק ב-`ngOnDestroy`.

Angular Routing .10

Routing .10.1

התשתיות של Angular מספקת דרך לנוסח בין רכיבים, כתוצאה מפעולות המשתמש. כידוע, אפליקציות רשת מוצגות באמצעות הדף, אשר מונע על ידי כתובות url. לכן, לוגיקה זו מככיבה את האופן שבו נבצע ניווט בין רכיבים.

ראשית, יש להגדיר מהו ה-`url` הבסיסי של האפליקציה. נבצע זאת באמצעות הגדרת `<base href=...>` בראוב המקרים, באפליקציות Angular, נבחר להשאיר אותו ריק. כך, לאחר מכן כתובות האתר נוכלים להוסיף את שמות הרכיבים אליהם הגענו:

<http://localhost:4200/home>

פירוש ה`url` הנ"ל הוא כתובות האתר ולאחר מכן אנחנו נמצאים ברכיב `home`.

כדי לבצע זאת, נגדיר ב-`index.html` את כתובות הבסיס `-"/"` משמע ריקה:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=UTF-8>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Angular2: Building modular applications by Shai Vashdi</title>
    <base href="/">
  </head>
  <body>

    <app>
      Loading...
    </app>

    <script src="/polyfills.bundle.js"></script>
    <script src="/vendor.bundle.js"></script>
    <script src="/main.bundle.js"></script>

  </body>
</html>
```

כעת אנחנו בתוך האפליקציה ונרצה להגדיר לה את השימוש בשירות ה-`routing`.

לכן ב-Root module ניבא את השירות ונזהיר על השימוש בו:

```
import {RouterModule} from "@angular/router";

@NgModule({
  declarations: [AppComponent, About, RepoBrowser, RepoList, RepoDetail, Home],
  imports: [BrowserModule, FormsModule, HttpClientModule, RouterModule.forRoot(rootRouterConfig)],
  providers: [Github, {provide: LocationStrategy, useClass: HashLocationStrategy}],
  bootstrap: [AppComponent]
})
```

הזהרה על השימוש, מעבירה לשירות את הקונפיגורציה של הנויוט באפליקציה.
הkonfiguracija של הנויוט באפליקציה הינה קובץ הגדרות בשם app.routes.ts:

```
const routes: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'home', component: HomeComponent},
  {path: 'about', component: AboutComponent},
  {path: 'products', component: ProductsComponent},
  {path: 'addProducts', component: AddProductsComponent},
  {path: 'editProducts', component: EditProductsComponent},
  {path: '**', component: PageNotFoundComponent}
];
```

ניתן לראות כי נתיב ברירת המחדל הוא כפי שרצינו:

<http://localhost:4200/home>

נתיב about יוצג באופן הבא:

<http://localhost:4200/about>

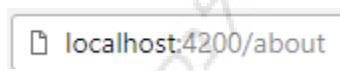
עד כה, הגדרנו נויוט לרכיבים מסוימים והנויוט משתקף בכתובת ה-[url](#).
אולם היכן מוצגים הרכיבים אליהם אנו מנוטים? בתחום איזור הנקרא: Router-Outlet.

Router-Outlet .10.2

בתוך הרכיב הראשי של האפליקציה, ניצר איזור אליו ינותו הרכיבים, באמצעות ה-router.
לכן נגדיר בתצוגה של ה-router-outlet בקובץ AppComponent.html ניצור ניוטו:

```
<div class="mainContainer">
  <div>
    <headerControl></headerControl>
  </div>
  <main>
    <router-outlet></router-outlet>
  </main>
  <footer>
    <p>Shai Vashdi © 2016</p>
  </footer>
</div>
```

איזור ניוטו זה המוגדר באמצעות התגית router-outlet הוא האיזור בו יוצג הרכיב הנבחר.
למשל:



יגרום לכך שבתוך התגית router-outlet תוצג הקומponentה AboutComponent.

ניתן לנoot באופן סטטי באמצעות ה-RouterLink:

אלו למעשה תגיות שניתן לשימ בគורתת הדף או בכל מקום אחר בדף הראשי:

```
<nav>
  <a [routerLink]="'['/']'">
    Home
  </a>
  |
  <a [routerLink]="'['/about']'">
    About
  </a>
</nav>
```

localhost:4200/home

localhost:4200/about

כאשר לחיצה על קישורים אלו מוביל לנתיבים הסטטיים האפשריים.

ניווט דינامي

ניתן לבצע ניווט דינאמי מתוך הקוד, על סמך לוגיקה, במיוחד לנtíבim שאינן דרך אחרת לנוט אליהם, כגון הרכיב `productEdit` אשר הניווט אליו כולל את ה-`id` של ה-`product` שנבחר. נוכל לעשות זאת ע"י השימוש בשירות הנויוט בקוד:

```
this.router.navigate(['products']);
```

העברה פרמטרים

נוכל להעביר פרמטרים בסאן לדוגמה:

ProductDetailsComponent
path: /products/:id handler: ProductDetailsComponent

כל חלק בסאן שיש לו את הקידומת של נקודותים, הוא פרמטר דינמי, וכן ניתן ליצור את ההגדרת ניתוב הבאה:

```
const routes: Routes = [
  {path: 'products/:id', component: ProductDetailsComponent},
];
```

ולגש ת אליה על ידי העברת פרמטר דינמי, לדוגמה:

<http://localhost:4200/products/1>

<http://localhost:4200/products/100>

<http://localhost:4200/products/abc>

וכן הלאה...

את הפורמטר נוכל לקרוא בתוך ה-ProductDetailsComponent בצורה הבאה:

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-product-details',
  templateUrl: './product-details.component.html',
  styleUrls: ['./product-details.component.css']
})
export class ProductDetailsComponent implements OnInit {
  currentId: any;

  constructor(private router: Router,
    private route: ActivatedRoute) { }

  ngOnInit() {

    this.route.params.subscribe(params => {
      this.currentId = params.id;
    });

  }
}
```

Nested Routing .10.3

בנוסף להגדרת איזור הנקרא Router-Outlet בעמוד הראשי, בו תוצג הקומפוננטה המתאימה ל-url הנוכחי לפי הגדרות routing, ניתן גם ליצור קינון משני של ניתוב, על ידי הוספת תגית:

```
<router-outlet></router-outlet>
```

אזרז ניוט זה המוגדר באמצעות התגית router-outlet הוא האזור בו יוצג הרכיב המקורי הנבחר, וכן בתוך html.products.conmponent.html. יש אזרז ניוט נוסף בשימוש נוסף בתגית router-outlet

```
<div class="mainContainer">
  <div class="horizontalContainer">
    <actionsBarControl></actionsBarControl>
  </div>
  <div class="horizontalContainer">
    <router-outlet></router-outlet>
  </div>
</div>
```

את הקונפיגורציות נגדיר באמצעות הוספת האופציה children הcolaلت את הגדרות הניתוב המקבנות:

```
const routes: Routes = [
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: 'home', component: HomeComponent},
  {
    path: 'products',
    component: ProductsComponent,
    children: [
      { path: 'add', component: AddProductsComponent},
      { path: 'edit', component: EditProductsComponent },
      { path: '', redirectTo: 'add' },
    ]
  },
  {path: '**', component: PageNotFoundComponent}
];
```

כאשר גם לנטייב המיקון יש ברירת מחדל, והוא add. אך, ברירת המחדל של הניתוב המיקון השלם יהיה:

<http://localhost:4200/products/add>

כלומר, בתחילת האפליקציה תלך לרכיב home:

<http://localhost:4200/home>

וכאשר על ידי ניוטו נסנה את ה- url ל:

<http://localhost:4200/products>

הוא יעבור בצורה דיפולטיבית להיוות:

<http://localhost:4200/products/add>

ובתווך ה- Router-Outlet שבקומponent ProductsComponent יוצג הבן שלו, שהינו רכיב בשם .AddProductsComponent

ניתן לנוט באופן סטטי באמצעות ה-RouterLink:

אלו למעשה תגיות שנitin לשים בכותרת הדף או בכל מקום אחר בדף הראשי:

```
<a routerLink="/products/edit">
  add product
<a>
```

```
<a routerLink="/products/add" >
  add product
<a >
```

כאשר לחיצה על קישורים אלו מוביל לנתיבים הסטטיים האפשריים.

ניווט דינامي

ניתן לבצע ניווט דינאמי מתוך הקוד, על סמך לוגיקה, במיוחד לנתיבים שאין דרך אחרת לנוט אליהם, כגון רכיבים אשר הניותו אליו כולל את פרמטרים דינמיים המבוססים על פעולות הלוקה.

אפשר לעשות זאת ע"י השימוש בשירות הניות בקוד:

```
class HomeComponent {
  constructor(private router: Router) {}

  goHome() {
    this.router.navigate(['products/edit']);
  }

  goSearch() {
    this.router.navigate(['products ']);
  }
}
```

Router Lifecycle Hooks .10.4

כמו components אשר עברים סדרה של שלבים שונים במהלך חייהם, גם ה- routing operation עובר שלבים שונים במהלך החיים. כל אחד מהם נגיש מ lifecycle hook שונה, אשר בדיקן כמו רכיבים, ניתן לטפל על ידי שימוש interface ספציפי לנושא של פועלות הניתוב:

1. CanActivate
2. CanActivateChild
3. CanDeactivate
4. Resolve
5. CanLoad

Angular Forms .11

Angular Forms .11.1

טפסים הם איגוד של הזנת מידע על ידי המשתמש תחת קורת גג-אחת, כאשר למידע יש משמעות לוגית משותפת.

לדוגמא: טופס `on-log` להתחברות למערכת, טופס מילוי פרטי קשר להזנה באתר וכו'. Angular Forms מספקת דרך נוחה וקלה לעשות זאת באמצעות התשתיות מאפשרת לנו לכתוב טפסים מורכבים, תוך חיסכון בכתיבה קוד. בנוסף, יכולת זו מאפשרת לנו את הטפסים בצורה קלה יותר, מבלי לערב רכיבים אחרים למערכת.

חשוב להבין: תשתיות זו אינה מחוללת טפסים באמצעות אשף (wizard) כפי שראינו בעבר בערכות אחרות. אלא תשתיות המסייעת לנו לצורך בניית טפסים בדיקות תקינות ושליחה.



באמצעות אנגולר ניתן לבנות טפסים בכמה דרכים, נטרץ בפרק זה בשתי האפשרויות הבאות:

- **על ידי כתיבת** form-specific directives **עם** (model-driven)
- **על ידי שימוש ב-** reactive (או)

Angular template driven Forms .11.2

בכדי שנוכל להשתמש ב[`(ngModel)`] חובה לייבא את `FormsModule` במודול:

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

import { AppComponent } from './app.component';

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

ה-directive של NgModel לא רק עוקב אחרי ה-state, אלא גם מאפשר את השליטה עם מחלקות של אנגולר ב-CSS.

להלן רשימה של שמות המחלקות הנינטנות לשימוש:

<u>State</u>	<u>Class if true</u>	<u>Class if false</u>
The control has been visited	ng-touched	ng-untouched
The control's value has changed	ng-pristine	ng-dirty
The control's value is valid	ng-valid	ng-invalid

לדוגמה, ניצור את הקומפוננטה הבאה:

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

@Component({
  selector: 'app-root',
  template: `

<div [hidden]="submitted">
  <h1>Login Form</h1>
  <form (ngSubmit)="onSubmit()" #loginForm="ngForm">

    <label for="name">Name</label>
    <input required
      [(ngModel)]="model.name" name="name"
      #name="ngModel">
    <div [hidden]="name.valid || name.pristine" >
      Name is required
    </div>

    <br/>
    <label for="platform">platform:</label>
    <select required
      [(ngModel)]="model.platform" name="platform"
      #platform="ngModel">
      <option *ngFor="let x of ['facebook', 'gmail', 'github']" [value]="x">
        {{x}}
      </option>
    </select>
    <div [hidden]="platform.valid || platform.pristine">
      Platform is required
    </div>
  <br/><br/>
  <button type="submit" [disabled]="!loginForm.form.valid">Submit</button>
</div>
```

```

        <button type="button" (click)="loginForm.reset()">Reset</button>
      </form>
    </div>

    <div [hidden]="!submitted">
      <h2>You Logged as:</h2>
      {{ model.name }}
    </div>
    `,
    styles: [
      .ng-valid {
        color: blue;
      }

      .ng-invalid {
        color: red;
      }

      .ng-valid[required], .ng-valid.required {
        border-bottom: 5px solid green;
      }

      .ng-invalid:not(form) {
        border-bottom: 5px solid red;
      }
    ]
  })
export class AppComponent {

  model = {};
  submitted = false;

  onSubmit() { this.submitted = true; }

}
    
```

כasher נרץ את האפליקציה, נקבל את התוצאה הבאה (מכיוון שיש שדות שם חובה, והם ריקים, הטופס לא תקין וכפטור ה-submit הוא במצב disable, disable, וכן הטופס עם כתוב אדום וכן תחתון אדום לכל תיבת):

Login Form

Name

platform: ▾

כאשר נמלא את תיבת הקלט הראשונה, נראה שהיא כעת עומדת בדרישה של הוואלידציה, ולכן הקו התיכון שלה הוא בצבע ירוק, ותוכנה בצבע כחול, אולם, הטופס עדין לא במצב תקין מכיוון שתיבת הקלט השנייה שגם חייבת להכיל קלט, עדין ריקה, ולכן הקו התיכון שלה אדום, וכפתור ה submit הוא במצב disable:

Login Form

Name

platform:

כאשר נמלא את תיבת הקלט השנייה, נראה שהיא כעת עומדת בדרישה של הוואלידציה, ולכן הקו התיכון שלה הוא בצבע ירוק, ותוכנה בצבע כחול, וכן הטופס כולו תקין וכפתור ה submit כבר לא במצב disable:

Login Form

Name

platform:

לאחר שנלחץ על ה submit קיבל את התוצאה הבאה:

You Logged as:

test

Angular model driven Forms .11.3

נתבונן בנוסאים הבאים:



נראה כיצד ליבא את השימוש ב-module הטפסים:

```

import {FormsModule} from '@angular/forms';
@NgModule({
  declarations: [AppComponent, About, RepoBrowser, RepoList, RepoDetail, Home],
  imports:      [BrowserModule, FormsModule, HttpClientModule, RouterModule.forRoot(rootRouterConfig)],
  providers:   [Github, {provide: LocationStrategy, useClass: HashLocationStrategy}],
  bootstrap:  [AppComponent]
})
export class AppModule {
}
  
```

ב-Root module של האפליקציה ניבא את הטפסים מ-'@angular/forms' ונרשם את המודול הנ"ל ב-.imports

כasher רכיב מסוים ירצה להשתמש ב-FormsModule הוא יציר על השימוש ברכיבי ה-form.

```

import {REACTIVE_FORM_DIRECTIVES, FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'about',
  styleUrls: ['./about.css'],
  templateUrl: './about.html',
  providers: [PeopleService],
  directives: [PeopleListComponent, REACTIVE_FORM_DIRECTIVES],
})
  
```

לדוגמא, ייבנו את ה- `FormGroup` וה- `FormControl`, על מנת להגדיר את ה- `form` בלוגיקה ואת ה- `REACTIVE_FORM_DIRECTIVES`.

FormControl ו FormGroup .11.3.1

באמצעות רכיבים אלו נגדיר את ה- `form` בלוגיקה:

```
editForm = new FormGroup({
  name: new FormControl("Product 1"),
  description: new FormControl("description 1"),
  price: new FormControl(551),
});
```

עבור כל שדה בטופס נגידר `FormControl`. `FormGroup` הוא אוסף של `FormControl`-ים. לדוגמא: נגידר שלושה שדות בטופס: שם, תיאור ומחר. לכל שדה נגידר ערך תחומי (למשל, לשדה מחיר (551).

```
<form [formGroup]="editForm" (ngSubmit)="onSubmit()">
  <div>
    <label for="name">Name</label>
    <input id="name" type="text"
      [formControl]="editForm.controls['name']" />
  </div>
  <div>
    <label for="description">Description</label>
    <textarea cols="1" rows="3" id="description"
      [formControl]="editForm.controls['description']">
    </textarea>
  </div>
  <div>
    <label for="price">Price</label>
    <input class="shortInput" id="price" type="text"
      [formControl]="editForm.controls['price']"/><label>$</label>
  </div>
  <div>
    <button type="submit" [disabled]="!isSubmitEnabled()>Submit</button>
  </div>
</form>
```

השתמשנו ב-`tag` בשם "form" והגדכנו את שם `FormGroup` שנמצא בלוגיקה. במקרה זה, `editForm` לאחר מכן, הגדרנו את פונקציית ה- `submit` ע"י התחברות ל- `ngSubmit` event. לכל רכיב הגדרנו את `FormControl` שהוא מייצג (למשל, ה- `input` השלישי מייצג את רכיב המחיר).

Name

Description

Price

 \$

עד כה נראה שלא חסכנו דבר, כתבנו תצוגה והתחברנו לכפתור submit.

אולם מה לגבי בדיקות תקינות של הטופס?

כאן נכנסו כלិ מואוד עיל בשם validations.

Validations. 11.3.2

ניתן להוסיף בדיקות לשדות בטופס כאשר חלון כבר מובנות ומוכנות לשימוש כגון:

- Required - מצין כי השדה הוא שדה חובה שחייב למלא ואני יכול להיות ריק.
 - minlength - מצין כי הנתונים בשדה צריכים להיות לפחות באורך מסוים.
 - maxlength - מצין כי הנתונים בשדה לא יכולים לעלות על אורך מסוים.
 - pattern - מצין כי הנתונים בשדה צריכים לעמוד חarakטים regex pattern מסוים (נראה דוגמא בהמשך).
- בנוסף, נראה כי ניתן גם להגדיר validation custom על פי כל לוגיקה שנבחר.

נראה דוגמאות לשימוש בבדיקות מובנות:

```
editForm = new FormGroup({
  name: new FormControl("Product 1",
    Validators.compose([Validators.required, Validators.maxLength(30)])),
  description: new FormControl("description 1",),
  price: new FormControl(551,
    Validators.compose([Validators.required, Validators.pattern(PRICE_REGEX_PATTERN)]))
});
```

השدة שם הוא שדה חובה (לא יכול להיות מוצר ללא שם) ועל השם להיות באורך של 30 תווים לכל היוטר.

השدة תיאור הוא שדה אופציונלי, אבל אורק התווים בו לא יכול לעלות על 200 תווים.
השدة מחיר הוא שדה חובה (אין מוצר ללא מחיר) והוא חייב לעמוד בפורמט כתיבה של מחירים, כגון:
מספר בלבד אך עם אפשרות לנוקודה עשרונית:

```
const PRICE_REGEX_PATTERN: string = "^\d{1,3}.\d{1,2}";
```

כרגע יש לשודות בדיקות תקינות אך כיצד יוכל להשתמש בכך?

Form valid/dirty .11.3.3

על סמך בדיקות אלו ניתן לדעת האם הטופס תקין או לא ובהתאם לכך ניתן לעשות פעולות שונות.
למשל לא אפשר לשלוח את נתוני הטופס אם הוא אינו תקין.

בנוסף, ניתן לבדוק אם הטופס השתנה, רק במידה שהוא השתנה, לאפשר לשלוח את העדכון:

```
isSubmitEnabled(): boolean {
  //enable if has changes (for edit existing) and valid.
  return this.editForm.valid && this.editForm.dirty;
}
```

נחבר זאת לתצוגה, כך שהכפטור לא יהיה זמין במידה שהטופס לא השתנה או אינו תקין:

```
<div>
  <button type="submit" [disabled]="!isSubmitEnabled()">Submit</button>
</div>
```

הדבר משפייע על זמינות כפטור submit, אולם מה לגבי הודעה למשתמש?

```

<form [formGroup]="editForm" (ngSubmit)="onSubmit()">
  <div>
    <label for="name">Name</label>
    <span class="notValid">
      *ngIf="!editForm.controls['name'].valid">required, up to 30 characters</span>
  </div>
  <div>
    <input id="name" type="text" [formControl]="editForm.controls['name']" />
  </div>
  <div>
    <label for="description">Description</label>
    <span class="notValid">
      *ngIf="!editForm.controls['description'].valid">up to 200 characters</span>
  </div>
  <div>
    <textarea cols="1" rows="3" id="description" [formControl]="editForm.controls['description']">
    </textarea>
  </div>
  <div>
    <label for="price">Price</label>
    <span class="notValid">
      *ngIf="!editForm.controls['price'].valid">required, larger than zero</span>
  </div>
  <div>
    <input class="shortInput" id="price" type="text"
           [formControl]="editForm.controls['price']"/><label>$</label>
  </div>
  <br />
  <div>
    <button type="submit" [disabled]="!isSubmitEnabled()>Submit</button>
  </div>
</form>

```

אפשר לכתוב הודעות למשתמש על בסיס התקינות של השדות.

ניתן לראות שיש שדות שיוצגו רק במידה שהטופו איננו תקין. ייצג כך:

Name **required, up to 30 characters**

Description **up to 200 characters**

```

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa

```

Price **required, larger than zero**

 \$

Submit

כעת, נטפל בלוגיקה של הגשת הטופס, משמע submit. ראיינו שהתחברנו לאיורע זה באמצעות הפונקציה submit הנמצאת בלוגיקה. הנה נראה כיצד לכתוב אותה:

```
onSubmit() {
  var name = this.editForm.controls['name'].value;
  var description = this.editForm.controls['description'].value;
  var price = this.editForm.controls['price'].value;
}
```

זהו פונקציה פשוטה, שבעת הגשת הטופס שולפת את הערכים מה-FormControl וכתעת יכולה לשלחן אותם להלאה, למשל באמצעות service.

Angular async Forms .11.4

בכדי שנוכל להשתמש ב-ReactiveFormsModule חובה ליבא את ReactiveFormsModule במודול:

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

import { AppComponent } from './app.component';

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

כעת, ניצור קומפוננטה בסיסית, המօסיפה בצורה אסינכרונית (בעזרת ארוע מובנה בשם valueChanges) למערך את הערך הנוכחי של תיבת הקלט, בכל פעם שהALKOHOT שינה את תוכן התיבה:

```

import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';
import 'rxjs/Rx';

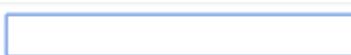
@Component({
  selector: 'app-root',
  template: `<input type="search"
    class="form-control"
    [FormControl]="searchField">
<ul>
  <li *ngFor="let s of searches">{{ s }}</li>
</ul>
` 
})
export class AppComponent implements OnInit {

  searchField: FormControl;
  searches: string[] = [];

  ngOnInit() {
    this.searchField = new FormControl();
    this.searchField.valueChanges
      .debounceTime(400)
      .distinctUntilChanged()
      .subscribe(x => {
        this.searches.push(x);
      });
  }
}

```

כאשר נריץ את האפליקציה נקבל בדף את התוצאה הבאה:



ולאחר שנקליד פעם את המילה `test` בתוך תיבת הקלט, נקבל את התוכן הבא:



- t
- te
- tes
- test

תרגיל



- צרו מחלוקת בשם person המכילה ת.ז. ושם
- צרו קומפוננטה המכילה מערך ריק של person
- צרו בדף ה HTML תיבת קלט לקבלת ת.ז. ושם, וכפתור הוסף
- יש להציג בטבלה את כל האנשים שהתווסףו

דוגמים

- יש ליצור בדיקת ואלידציה עבור תעודה זהה, לפי האלגוריתם המצורף:
A. רושמים את מספר תעודה זהה ב 9 ספרות, כאשר אם מספר תעודה זהה הינו פחות מ 9 ספרות, משלימים אפסים מובילים בצד שמאל של המספר

1	2	3	4	5	6	7	8	2
---	---	---	---	---	---	---	---	---

- ב. מתוך כל סירה של מספר תעודה זהה, רושמים החל מימין של המספר את הספרות 1, אחר נס, אחר נס שוב 1, אחר נס שוב 2 וחזר חיללה עד לסיום. מספרים אלו נקראים משקלים.

1	2	3	4	5	6	7	8	2
1	2	1	2	1	2	1	2	1

- ג. מכפילים כל סירה במספר תעודה זהה עם הסירה מתחתיה (המשקל). את התוצאה, גם אם היא בשני ספרות, רושמים מתחת בשורה שלישי.

ולכן מכפיל כל סירה של תעודה זהה בספרה שרשemo תחתיתה, קלומר: 1 כפול 1, 2 כפול 2, 3 כפול 1 וכו'... .

1	2	3	4	5	6	7	8	2
1	2	1	2	1	2	1	2	1
1	4	3	8	5	12	7	16	2

- ד. את התוצאות הופכים לתוכאות של סירה אחת. למשל, אם תוכאה כלשהי היא מספר דו סיפרתי, מחבירים את כל ספרות המספר לסירה אחת. למשל, את התוצאה 14 מחבירים כ 4 ועוד 1, ומתקבלים 5. את התוצאה 10 מחבירים כ 1 ועוד 0 ומתקבלים 1. את כל התוצאות רושמים בשורה רביעית.

1	2	3	4	5	6	7	8	2
1	2	1	2	1	2	1	2	1
1	4	3	8	5	12	7	16	2
1	4	3	8	5	3	7	7	2

ג. כעת, מחברים את כל הספרות בחיבור חשבוני פשוט עד לקבלת תוצאה. התוצאה חייבת להיות מספר המתחלק ב 10 ללא שארית, כלומר שסכום האחדות שלו יהיה 0.

$$1+4+3+8+5+3+7+7+2 = 40$$

ה. תוצאה שאינה מתחלקת ב 10 ללא שארית, מורה על מספר כלשהו שגוי בתוך מספר תעודת זהה ('הקלדת לא נכונה למשל'). תוצאה שמתחלקת ב 10 ללא שארית מורה על מספר תעודת זהות תקין.

Angular onPush .12

Change Detection in Angular .12.1

כאשר אנו מושנים ערך באחד מהמודלים שלנו בפרויקט Angular, אז'i מזהה את השינויים מיד מעדק את ה-view. זהו change detection. מטרת מנגנון זה היא לוודא שהתוכן המוצג בדף תמיד מסונכרן עם המודלים המתאים שלו. תכונה זו היא הליבה של Angular והוא זו שעשוה את ה framework הזה לאפקטיבי עבור פיתוח ישומי אינטרנט מודרני.

מודל ב-Angular יכול להשתנות כתוצאה מהתרחישים הבאים:

- DOM events (click, hover over, etc.)
- AJAX requests
- Timers (setTimer(), setInterval())

Change Detectors .12.2

כל ישומי Angular מרכיבים מעץ היררכי של רכיבים. בזמן הריצה Angular, יוצרת מחלוקת נפרדת של change detector עבור כל component בעץ, אשר בסופו של דבר יוצר היררכיה של change detectors הדומים לעץ היררכיה של רכיבים.

בכל פעם ש-change detection מופעל Angular, יורד למטה בעץ זה של change detectors כדי לבדוק אם אחד מה-detectors דיווח על שינוי.

מחזור מבחן change detection מבוצע תמיד פעם אחת עבור כל detected change ומתחילה מ-root change detector ומשם ממשיך למטרה בצורה עקבית. בחירה זו של סקירה מהஸורש לכיוון מטה, מעדכנת את המודל בצורה צפואה, שכן אנו יודעים כי נתונים רכיב יכולים להגיעה רק מההורה.

ה-change detectors מספקים דרך לעקוב אחר המצבים הקודמים של components וה-states הנוכחי שלהם, כמו גם המבנה שלו על מנת לדוח על שינויים בזוויתית.

אם Angular מקבל דיווח מה-change detector, הוא מכoon את הרכיב המתאים כדי לבצע לו re-render ולעדכן את DOM בהתאם.

כדי להבין מהי אסטרטגיית change detection ומדוע היא פועלת, עלינו להבין תחילתה את ההבדל בין JavaScript reference types ל-value types.

Value Types

- Boolean
- Null
- Undefined
- Number
- String

Reference Types

- Arrays
- Objects
- Functions

הבחנה החשובה בין **value types** ל-**reference types** היא, ש כדי לקרוא את הערך של סוג הערך, אנחנו פשוט צריכים לשולף את הערך מ-**stack memory**, אבל כדי לקרוא את הערך של **reference types**, אנחנו צריכים קודם לפנות אל ה-**stack memory** כדי לקבל את ההפניה ולאחר מכן להשתמש בפניה זו לאיתור הערך בתוך ה-**heap memory**.

כפי שציינו קודם קודם Angular, עוקבת אחר שינויים ב-**model** כדי לוודא שהוא מעדכן את כל השינויים. זה בדוק כל הבדלים בין ה-**state** הקודם ל-**state** הנוכחי של **application model**.

אוף הבדיקה של Angular - היא האמ' ערך כלשהו במודל השתנה. אבל עבור **reference type**, הבדיקה האם כל הערכים שוים היא מורכבת יותר ודורשת יותר זמן ומשאבים.

זהו המקום שבו **OnPush** נכנס. הרעיון המרכזי מאחורי אסטרטגיית **OnPush** מתבטא בהבנה שאם נתייחס ל-**reference types** כ-**immutable**, נוכל לזהות אם ערך השתנה מהר יותר. כאשר(reference type) הוא **immutable**, פירוש הדבר בכל פעם שהוא מटעדכן, ה-**reference** לשיכון ה-**heap** נמצא ב-**stack memory** שנמצאת ב-**stack memory** תיה חייבות לשנתנות. עכשו אנחנו יכולים פשטוט לבדוק: האם הכתובת ב-**stack** לאובייקט הנמצא ב-**heap** השתנה, ורק אם כן נבדוק את כל הערכים (של האובייקט בתוך הקnop).

האסטרטגיה **OnPush** מפעילה את הבדיקה לשתי שאלות במקום אחת:

- האם השנתנה הכתובת ב-**stack** של האובייקט מסווג **reference type** או
- אם כן, אז תבצע בדיקה של הערכים השווים לאובייקט וממוקמים ב-**heap**

לדוגמא, נניח שיש לנו immutable array עם 30 אלמנטים וANO רוצים לדעת אם יש שינויים. ANO יודעים שכדי שייהו עדכנים ל-reference, immutable array ב-stack יהיה חייב לשתנות. ולכן ANO יכולים לבדוק תחילה האם reference ב-stack השתנה, ואם הוא לא השתנה – ANO יודעים בוודאות שהמערך לא עבר שינוי, וכך חסכנו 30 בדיקות נוספת כדי לקבוע איזה אלמנט שונה.

Treating objects as mutable	Treating objects as immutable
<pre>static mutable() { var before = { foo: "bar" }; var current = before; current.foo = "hello"; console.log(before === current); // => true }</pre>	<pre>static mutable() { var before = { foo: "bar" }; var current = before; current = { foo: "hello" }; console.log(before === current); // => false }</pre>

בדוגמאות לעיל, אנחנו "מתיחסים" ל-immutable reference types כב-objects, ונוכל לישם את אסטרטגיית @Component changeDetectionparameter ב-Component-over OnPush annotation

לדוגמא, הקוד הבסיסי הבא:

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

@Component({
  ...
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class OnPushComponent {
  ...
}
```