****

**.מדריך אנגולר 14 :**מהו אנגולר?

בפיתוח תוכנה, AngularJS היא תשתית תוכנה (framework) בקוד פתוח ליישומי רשת, המתוחזקת על ידי גוגל ועל ידי קהילה רחבה של מפתחים. התשתית מיועדת לפתרון אתגרים בפיתוח יישומי דף-יחיד, ופישוט הפיתוח והבדיקות של יישומים אלו, באמצעות תשתית תוכנה לארכיטקטורות צד לקוח כמו MVC או MVVM, יחד עם רכיבים בהם משתמשים בדרך כלל ביישומי אינטרנט עשירים.

ספריית AngularJS פועלת על ידי קריאת דף ה-HTML, כאשר אל התגיות שבו נוספו תכונות נוספות.

Angular יפרש את התכונות הללו כהנחיות לקשר את אזורי הקלט או הפלט בדף למודל שמיוצג על ידי משתני JavaScript פשוטים. הערכים של משתני ה-JavaScript הללו ניתנים לשינוי על ידי קוד, וניתן לגשת אליהם בצורה סטאטית, או בצורה דינאמית בעזרת JSON.

בין יתרונתייה הבולטים:  
1.עבודה עם קופמנינטות

2. עבודת בשיטת MVC   
  
3. יכולת להציג אפליקציה בדף HTML בלי צורך לרענן את הדפדפן בין עמוד לעמוד.  
***ויקיפדיה*** 6-7-19

כיום, הגרסה האחרונה של אנגולר הינה 8 – 14 (בכל חצי שנה עד 8 חודשים בממוצע גוגל מוציאה גרסה חדשה, כאשר רוב ההבדלים בין הגרסאות הינם בביצועים ובמשקל האפליקציה הסופית).

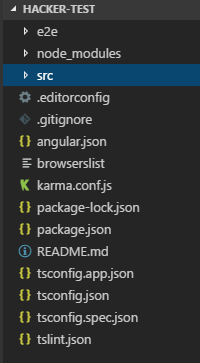
אינדקס :  
1. [**כיצד מתקינים את אנגולר?**](#bookmark=id.2p2csry)  
**2. מה היא קומפונינטה ?   
3. כיצד נראית קומפוננט   
4. כיצד מייצרים קומפנינטה בצורה נכונה ואוטומטית ב אנגולר  
5. מה הוא TYPE SCRIPT  
6. מה זה DATA BINDING (כולל two-way-binding, prop bind , event bind)  
7. DIRECTIVES   
8. מה זה ngIf**\* **9. מה זה ngStyle וngClass?  
10. שימוש ב NGFOR:  
11. שימוש ב NGSWITCH   
12. מה זה תוסף AUGURY וכיצד לבצע DEBUG  
13. BINDING CUSTOM PROP  
14. העברת CUSTOM EVENT בין קומפינינטות  
15. מה זה ng-content   
16. מבינים מה זה HOOKS LIFE CYCLE  
17. מה זה HOSTLISTENER DIRECTIVE (כולל HOSTBINDING ו PROP BINDING)  
18. SERVICES  
19. מה זהPIPES ולמה צריך אותם?  
20. כיצד לבצע פקודת SORT BY או FILTER באנגולר   
21. מה זה OBSERVABLE ולמה צריך אותם בשביל לבצע REQUEST לAPI  
22. עבודה מול REST API עם httpClient  
23. מה זה ROUTER ולמה צריך אותו?  
24. איך יוצרים לינקים באנגולר שישנו את ה ROUTER   
25. כיצד להעביר פרמטרים דרך ה ROUTER ?  
26. כיצד לייצר "עמוד לא נמצא 404"  
27. איך לבצע DEPLOY לאפליקציה שלנו  
28. ולידזאציה (בדיקת תקינות) לטפסים באנגולר (כולל form validation והצגת הודעות שגיאה)  
29. התממשקות ל FIREBASE !  
30. כיצד לקבל את המידע (רשימה) מהמסד נתונים שלנו בFIREBASE**

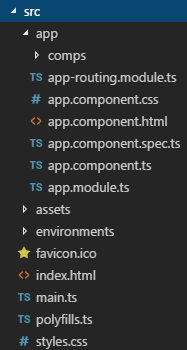
**31.**[**מה זה INTERFACE**](https://vegibit.com/how-to-use-an-interface-in-angular/)  
32. [איך לאסוף DATA אטרביוט](https://www.techiediaries.com/add-access-html-data-attribute-angular/)

**1. כיצד מתקינים את אנגולר?**  
ראשית, עלינו להתקין במחשבנו את ה CLI (COMMAND LINE) של ANGULAR כדי שנוכל לבצע זאת, עלינו קודם להתקין את הגרסה האחרונה של NODEJS (מדריך ניתן לראות בעונה 9)

לאחר מכן נתקין דרך NPM את האנגולר בשורה הבאה:  
npm install -g @angular/cli

ובתיקייה שבה נרצה לייצר את הפרויקט נכתוב את הקוד הבא:  
ng new <app-name>

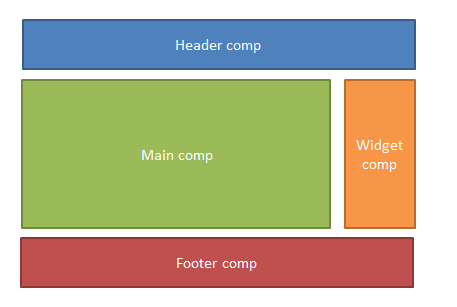
אחרי שנתקין פרויקט חדש, זו רשימת הקבצים והתיקיות שיותקנו לנו בפרויקט:  
(

ובתוך תיקיית SRC אלו הקבצים שיותקנו בה (תיקיית comps לא תותקן בהתקנה ההתחלתית):  


רוב כתיבת האפליקציה שלנו תכתב בתוך התיקייה SRC ובמיוחד בתיקיית APP שנמצאת בתיקיית SRC.

הקובץ הראשון באפליקציה כקומפנינטה שפועל ושולח פקודות לכולם הינו app.component.tsאשר מופעל על ידי main.ts .  
  
בנוסף, כל המודולים ("ספריות עם יכולות שונות") שנשתמש בהם נמצאים בקובץ APP.MODULE.TS אשר בשונה מריאקט שבה היינו צריכים לעשות IMPORT לכל מודול בכל קומפנינטה שרצינו להשתמש באותו מודול מחדש.

**2. מה היא קופנינטה ?**

באנגולר נוכל לבנות את האפליקציה /אתר שלנו (מעכשיו ייכתב אפליקציה בלבד אך הכוונה גם לאתר) ממרכיבים (קומפונינטות בשם המקצועי שלהם באנגולר/ריאקט) לדוגמא: אתר בסיסי שיש לו אזור HEADER,MAIN,WIDGETS SIDE,FOOTER נוכל לחלק אותו לקומפנינטות בצורה הבאה:  


כל קומפנינטה תיכתב עם מחלקה שלה ב JS ובדרך כלל כאשר נייצר אותה היא תכיל 3 קבצים (ישנו קובץ רביעי לטסטים שהוא לא הכרחי לפיתוח) :

1.הקוד מוח של הקומפנינטה (שיוצג כקובץ TS כ TYPE SCRIPT)

2. קוד השלד של ה VIEW של הקומפנינטה (יוצג כ HTML)

3. קוד העיצוב של הקופנינטה (יוצג כקובץ CSS)

**3. אז כיצד נראת קומפנינטה (בדוגמא קובץ app.component.ts)**

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'hacker-test';

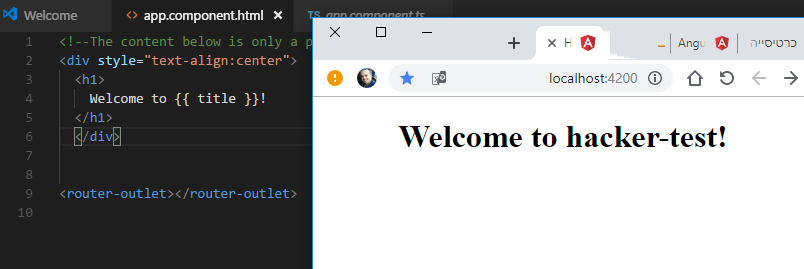
}

השורה הראשונה של ה IMPORT בעצם מחברת את המודול של אנגולר ומשתמשת ביכולות שלו להציג TYPESCPRIPT

החלק @COMPONENT הינו יכולת שה TYPESCRIPT ה "@" מייצג פקודת דיקרטור ובמקרה שלנו מכריז על המרכיבים של הקומפנינטה מ HTML , CSS וה ID של הקומפנינטה לשימוש במקרה של הקוד הנ"ל (app-root) . שימו לב! שחשוב לשמור את שם מיוחד ל ID (SELECTOR) ובדרך כלל מומלץ להוסיף את המילה app . בהמשך נראה שאנגולר ישתמש בסלקטור כתגית ממש כמו תגית HTML על מנת להציג ולהשתמש בקומפנינטות.

ובסוף אנו מבצעים EXPORT (פקודה שחייבת להתבצע על מנת שנוכל לבצע IMPORT) למחלקה של ECMA 6 ובנוסף בקוד הנ"ל נוסף משתנה שניתן "להזריק" לקוד ה HTML בקומפניניטה בשם TITLE

כדי להריץ את הקוד שלנו שיוצג בדפדפן יש ל רשום את הפקודה הבאה ב CMD בתקיית הפרוייקט:  
***ng serve***

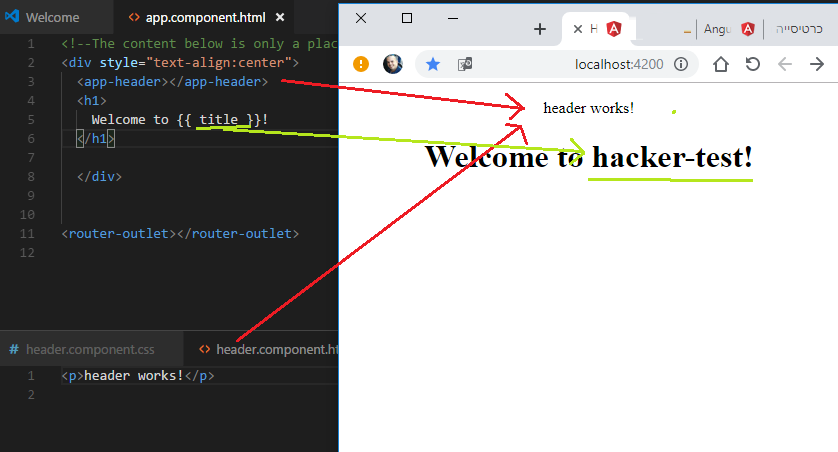
דוגמא לקוד ב HTML (חלק ה CSS נדבר עליו בהמשך המדריך) והתוצאה הסופית:  
  
\* נזכיר רק שבשורה 4 בתוך ה הסוגריים המסולסלים (מעתה יקרא מסולסלים) ישנו המשתנה TITLE שהגדרנו בקובץ   
app.component.ts שמציג את המחרוזת עם הערך " HACKER-TEST "

**4. איך מייצרים קומפנינטה בצורה נכונה ואוטומטית ב אנגולר:**  
בגדול ניתן לייצר ידנית את הקומפנינטה , אך מהר מאוד תבינו שהדבר דורש ייצור וכתיבה רבה של קוד (צריך לייצר לפחות 3 קבצים של HTML , TS,CSS) ובנוסף לעדכן בקובץ app.module.ts שהקומפנינטה קיימת.  
  
אז איך נעשה את זה חכם יותר?  
פשוט מאוד! בתיקייה של הפרויקט נכתוב את הפקודה הבאה:

ng generate component header   
או בקיצור

ng g c header>

בנוסף, ניתן גם לרשום את תיקיית היעד שבה נרצה שבתוך APP קבצי הקומפנינטה ייוצרו ומומלץ שמלבד ה APP כל הקומפינינטות יהיו בתוך תיקיית COMPS בתיקיית הAPP ולכן בדרך כלל נכתוב את הפקודה בצורה הבאה:  
ng g c comps/header

**כיצד משתמשים בקומפנינטה?**כל מה שעלינו לעשות על מנת לחבר קומפנינטה מקופמנינטת אב הוא פשוט לכתוב אותה כתגית לפי הסלקטור שלה,  
נניח שנרצה לייצר קומפנינטת HEADER בתוך קומפ' APP נשתמש בתגית לפי הסלקטור שלה ב TS (header-app)  
  
\*שימו לב: כאשר נייצר קומפנינטה דרך ה CLI , אנגולר תמיד תכתוב בתוך הHTML את שם הקומפנינטה ואת WORKS! בתגית P

אחרי שהבנו איך מייצרים קומפונינטה  **האתגר** הבא שלכם הוא לייצר 2 קומפנינטות שיוצגו ישר בהתחלה של SUCCESS ו WARNING ושיופיעו כשנפעיל את האפליקציה בצורה הבאה:  


ng g c comps/success  
ng g c comps/warning  
\* ניתן להשתמש למשימה בBOOTSTRAP

**5. מה הוא TYPE SCRIPT:**  
טייפ סקריפט הינו פורמט חדש לכתיבת JS בצורה מתוחכמת יותר הדומה לשפות מורכבות כגון C++ , JAVA ועוד...  
הפורמט מוסיף בעיקר את היכולות הבאות:  
א. אפשרות להגדיר טיפוסי (סוג) משתנים כגון NUMBER ',STRING וכו'...  
ב. יוצר INTERFACE במחלקה  
ג. קוד ברור יותר וחכם יותר  
ד. החיסרון היחידי הוא שכיום לא יכול "להתרנדר" לבד בדפדפן ולכן יש צורך בקומפייליר כגון ANGULAR CLI שיהפוך אותו ל JS ECMA5+ ככה שהדפדפנים יוכלו לקרוא את הקוד שלו.  
  
בסוף הפיתוח אנו נבצע DEPLOY/BUILD לקוד שלנו שיהפוך את הקוד לקבצי JS כך שנוכל לעלות אותו לכל שרת ואפילו ל הוסטינג של שרתי לינוקס (כגון GO DADDY, BLUE HOST וכו...)

להלן דוגמאות לשימוש ב TYPESCRIPT

// primitive

num:number = 5;

num2:number = 4.3;

num3:number = 0xff0000;

word:string = "hello";

flag:boolean = true;

// אפשר להכניס איזה מידע שאנחנו רוצים any

wordNum:any = "koko";

// מערכים

// מגדיר מערך שכל תא חייב להכיל סטרינג

words\_ar:string[] = ["koko","moshe","jack"];

// אופציה שניה להגדרת מערך של סטרינגים

words2\_ar:Array<string> = ["apple","mango","orange"]

// מערך שכל תא נקבע לו את סוג המידע שיקבל וגם את כמות התאים

hybridAr:[string,number] = ["may",5];

// ():number -> מחייב את הפונקציה להחזיר מספר

multi(\_x:number,\_y:number):number {

return \_x \* \_y ;

}

**6. מה זה DATA BINDING**

אחד מהיתרונות הגדולים של פריים וורק האנגולר הוא האפשרות להזריק משתנים לתוך ה VIEW של ה HTML בדומה לסעיף 3 עם משתנה ה TITLE   
ישנם מספר אפשרויות להזרקת מידע לתוך ה VIEW שלנו:  
1. STRING INTERPOLATION  
2. PROPERTY BINDING  
3.EVENT BINDING  
4. TWO WAY DATA-BINDING

נתחיל עם האופציה הבסיסית ביותר שנקראת **STRING INTERPOLATION**   
בקומפוננטה הראשית של בקובץ app.component.ts נייצר משתנה בשם myName וניתן לו את הערך "koko"

export class AppComponent {

title = 'hacker-test';

myName = "koko";

}

בתוך ה VIEW של הקומפנינטה בקובץ app.component.html נכתוב בתגית H2 את הקוד הבא:

<div style="text-align:center">

<app-header></app-header>

<h1>

Welcome to {{ title }}!

</h1>

<h2>Your name is: {{myName}}</h2>

</div>

כשנריץ את האפליקציה שלנו הקוד שנמצא בתוך הסוגריים המסולסלות יציג את ערך המשתנה בהם.  
במקרה שלנו KOKO  
דוגמא נוספת ל STRING INTERPOLATION יחד עם פונקציה  
  
בתוך קובץ הקומפנינטה TS של APP נייצר את הפונקציה הבאה כך שהקובץ כרגע יראה כך בתוך המחלקה:

export class AppComponent {

title = 'ang-hack';

myName = "koko";

myType = "password"

showMyNameAndType(){

return "my name is "+this.myName + " and my type is " + this.myType;

}

}

ובתוך ה HTML נוסיף את השורה הבאה שתשמש בפונקציה:

<p>Test: {{this.showMyNameAndType()}}</p>

נוכל לראות שניתן גם להוציא מידע מקריאה לפונקציה שמחזירה ערך   
  
**6.א מה זה PROPERTY BINDING?**  
נייצר משתנה בתוך ה APP קומפנינט בשם myType וניתן לו את הערך password (במקרה שלא עשיתם זאת עדיין בסעיף הקודם)

export class AppComponent {

title = 'ang-hack';

myName = "koko";

myType = "password"

}

ובקובץ HTML של הAPP נוסיף את שתי השורות הבאות:

<label>enter password:</label>

<input [type]="myType" name="pass" id="id\_pass">

שימו לב שהכנסנו את המשתנה MYTYPE לתוך האטרביות TYPE של ה INPUT בסוגריים מרובעות מה שמרמז לאנגולר בקימפול שמדובר ב PROPERTY BINDING

כאשר נריץ את האפליקציה ונכתוב בתוך ה INPUT נראה שאנו מקבלים נקודות שחורות והערך של ה MYTYPE נהפך ל PASSWORD  
  
*שימו לב לסגנון הכתיבה הנ"ל שישמש אותנו רבות בתוך הסוגריים המרובעים!*

**6.ב מה זה EVENT BINDING**  
בנוסף נוכל להעביר מידע בעת האזנה לאירוע , בדוגמא הבאה נאסוף מידע מ INPUT שכתבנו קודם של שדה הקלט PASSWORD  
ראשית נוסיף ל INPUT בתוך ה HTML של ה APP את המידע הבא:

<input [type]="myType" name="pass" id="id\_pass" (input)="onPassUpdate($event)">

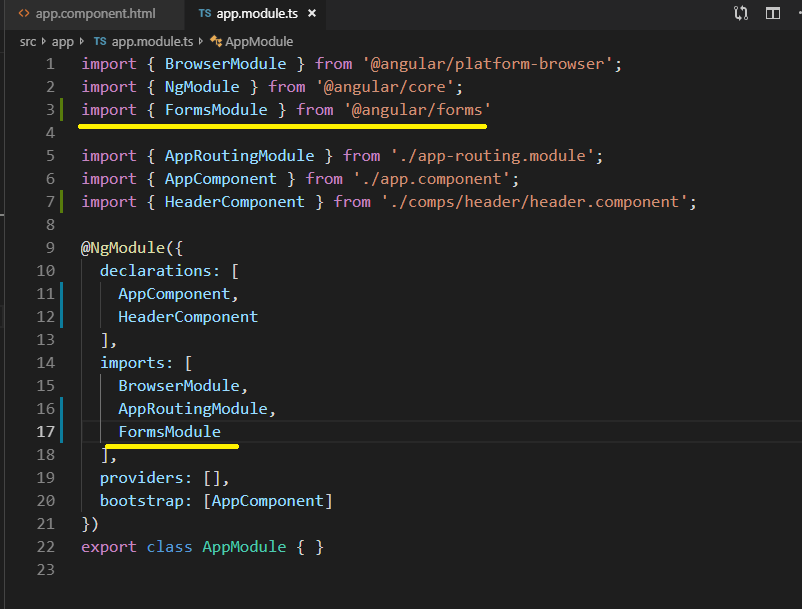
שימו לב שבמקום לרשום ONINPUT כמו בJS רגיל רשמנו את הINPUT בתוך סוגריים מעוגלים מה שמרמז לאנגולר שמדובר ב EVENTBINDING ואז רשמנו את שם הפונקציה (שנרשום בסעיף הבא ב APP ) בשם ONPASSUPDATE והעברנו לו את הארגיומנט EVENT כדי שישמש אותנו לאסוף את המידע בפונקציה מה INPUT .  
בתוך הקלאס של ה APP נכתוב את הפונקציה הבאה:

onPassUpdate(event){

this.myName = event.target.value;

}

שימו לב (נדבר על זה בהמשך בשיעור) שאם תעשו CONSOLE LOG ל פרמטר EVENT תראו שהמידע שנכתב בתוך ה האינפוט נמצא בתוך ה TARGET בפרופ VALUE וכשנפעיל את האפליקציה נראה שבאזור שהזרקנו ב HTML את   
המשתנה MYNAME המידע ישתנה אוטומטית בכל פעם שנכתוב משהו בINPUT

**6.ג. מה זה TWO WAY DATA BINDING?**  
בסעיף הקודם של BINDING EVENT ראינו שבשביל להעביר מידע מ INPUT או SELECT BOX עלינו לייצר פונקציה להעביר פרמטר ורק לאחר מכן בפונקציה לתת את הפקודה ורק אז להזריק את המידע שאנו רוצים, מה שיכול לסבך עניינים בשינויים קטנים שנרצה לעשות לפעמים, אז מה עושים ? אנגולר מגיעה "אוטומטית" עם המודול שנקרא NGMODEL שמאפשר די בקלות להאזין לאירוע לשנות את המשתנה בזיכרון במהירות ולהזריק אותו בחזרה למשתמש  
  
אך כדי שנוכל להשתמש ביכולת זו נהיה חייבים להוסיף בתוך קובץ app.module.ts את 2 השורות הבאות (מסומנות בקו צהוב):  
  
לאחר מכן כל מה שנצטרך כדי להשתמש ב TWO WAY DATA BINDING זה לייצר INPUT נוסף שנבקש מהמשתמש להכניס את גילו ושימו לב באיזו קלות (ללא צורך בכלל לגעת בקלאס עצמו) נציג לו את הבחירה שלו:

<label>Enter your age:</label>

<input type="number" name="age" id="id\_age" [(ngModel)] ="myAge" >

<h2>Your age is {{ myAge}}</h2>

שימו לב שה NGMODEL עטוף גם בסוגריים מרובעים וגם עגולים (מכיוון שהוא מייצג גם EVENT BINDING וגם   
PROPERTY BINDING יחד ! וזהו עכשיו מה שנרשום ב INPUT או (בSELECT BOX בבחירה) יעבור אוטומטית להזרקה של השם שנמצא בתוך ה NGMODEL

**אתגר** צרו SELECT BOX 5של צבעים שתבחרו וצרו תגית H2 שיש בתוכה מלל מסויים,   
א. בבחירה של צבע המילה בתגית H2 תשתנה למילה של הצבע.

ב. נסו שגם הצבע עצמו של הSTYLE ישתנה לצבע הנבחר

להלן הנושאים שנלמד במדריך זה לאחר שהבנו את הבסיס של אנגולר:  
  


**7. DIRECTIVES**

באנגולר ישנם מספר סוגים של דיירקטיבים

1. קומפ נינטה – כל הקומפינינטות שיצרנו עד היום נחשבים דיירקטיבים
2. STRUCTUAL DIRECTIVE – ששולטים בתצוגה של הDOM כגון IF ו FOR וכתובים תחילה עם " \* " (כוכבית)
3. ATTRIBUTE DIRECTIVES – משנים את התצוגה וההתנהגות של אלמנטים קומפנינטות או דיירקטיבים אחרים

נתחיל עם בניית דירקטיב משלנו ( (ATTRIBUTE DIRECTIVES (שהוא קצת מורכב יותר מבחינת הבנה) על מנת שיהיה לנו קל יותר להבין את הדרקטיבים האחרים   
  
תקציר : אני? נייצר דירקטיב אטריביוט פשוט שכאשר נוסיף אותו על תגית HTML מסויימת מסוג טקסט כגון כותרת או פסקה הרקע שלה ייהפך לירוק בהיר

על מנת לייצר קובץ דירקטיב משלנו (שדי דומה לקומפנינטה ) דרך הCLI נכתוב   
ng g d dir/green או ng generate directive dir/green  
  
לאחר יצירת הפקודה – תיווצר לנו תיקייה בשם "**DIR**" . שם יהיו כל הדירקטיבים שנייצר בעצמנו , ובתוכו נוצר בדומה לקומפנינטה קובץ דירקטיב בשם green.directive.ts , אנו נוסיף לו מספר שדרוגים על מנת שיעבוד חכם יותר (בהמשך הסבר מה כל דבר הוסיף)

import { Directive, OnInit , ElementRef } from '@angular/core';

@Directive({

selector: '[appGreen]'

})

// מייבא את היכולת של הוק לפעול במחלקה

export class GreenDirective implements OnInit {

// הפרייבט מאפשר להתשמש במשתנה שהעברנו לקונסטרקטור בכל מקום במחלקה גם במיטודות אחרות

constructor(private elementRef: ElementRef) { }

ngOnInit(){

// פקודה של דיבור עם האלמנט שעליו יהיה את הדרקטייב שיהפוך את הרקע שלו לירוק

this.elementRef.nativeElement.style.background = "yellowgreen"

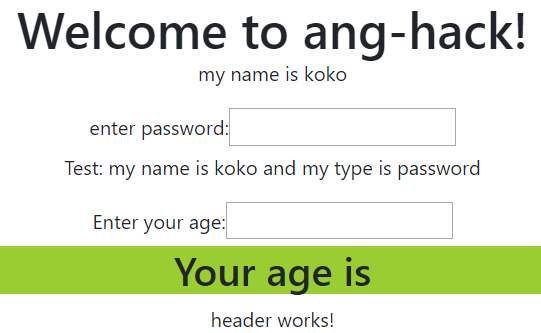
}

}

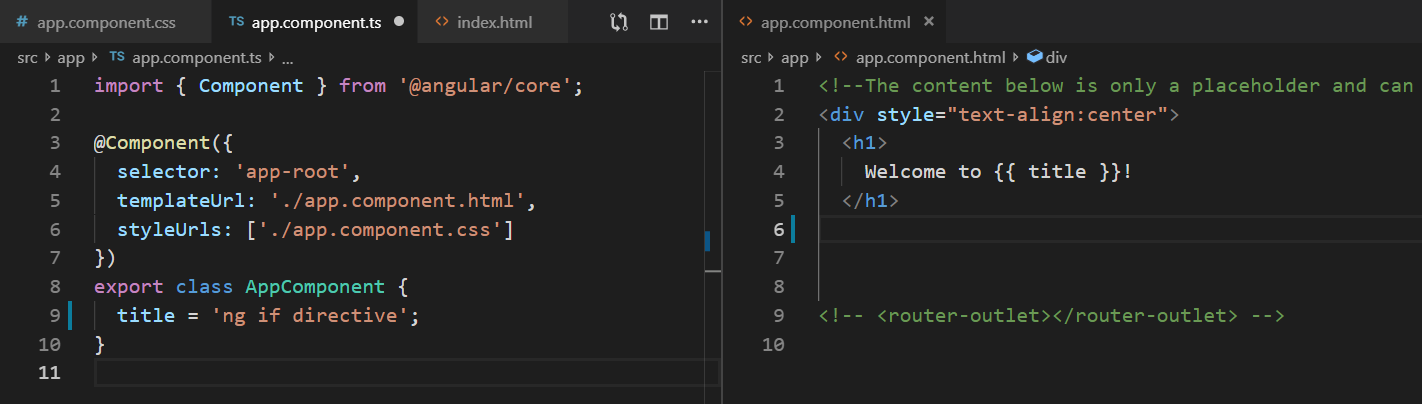
דבר ראשון ניתן לראות שבדומה לקומפנינטה גם כאן יש SELECTOR שהפעם ישמש אותנו בתור אטרביוט ולא בתור תגית.  
דבר שני ניתן לראות שב IMPORT ייבאנו 2 פקודות חדשות oninit (שהוא סוג של HOOK שפועל אחריי שהקומפנינטה נטענה במלואה שעליה נכניס את הDIRECTIVE ) וה ELEMENTREF שמאפשר לנו לדבר בקלות עם המאפיינים של התגית שעליה נשים את הדירקטיב ובמקרה שלנו נהפוך אותו לרקע ירקרק

לאחר מכן ב HTML של ה APP כל מה שנותר לנו לעשות על מנת שהדרקטיב יעבוד זה להוסיף אותו כאטריביוט לפי ה סלקטור שרשום לו בקובץ שלו (במקרה שלנו appGreen)

<h2 appGreen>Your age is {{ myAge}}</h2>

תוצאה (איפה שכתוב (YOUR AGE IS:  


לאחר שהבסיס של איך דירקטיבים בנויים הובן, עכשיו נתחיל להשתמש בדירקטיבים נפוצים שהם BUILD IN בתוך הפריימוורק של אנגולר (בדרך כלל יהיו מסוג STRUCTUAL ) :  
  
נתחיל עם :  
**8. מה זה ngIf**\*

בואו נחזור להתחלה נמחק את כל מה שעשינו ב HTMLוב TS של ה APP  
(אנו ממליצים לעשות גיבוי למה שעשיתם עד עכשיו בתיקייה אחרת במחשב שלא קשורה לפרוייקט)  
כך ששני הקבצים יראו פחות או יותר כך (מה שלא ירוק ב הערה):  


בתוך TS של ה APP נייצר משתנה בשם SHOWINFO שבתור התחלה יהיה שווה FALSE

export class AppComponent {

title = 'ng if directive';

showInfo = false;

}

ובתוך ה HTML נוסיף תגית H2 עם האטרביוט של \*NGIF שיראה כך:

<h2 \*ngIf="showInfo">show info is work</h2>

שימו לב שאם תנסו להריץ את האפליקציה כרגע ה תגית H2 לא תוצג כלל , אך אם תשנו את ערך המשתנה ל TRUE  
התגית אכן תופיע .  
אז כרגע למדנו איך אנחנו מייצרים את התנאי הראשון שלנו בתוך הVIEW של הקומפנינטה בקלות ... אך נשאלת השאלה איך אנחנו עושים ELSE?

בשביל זה נכיר תגית חדשה שאנגולר מאפשר לנו להשתמש בה שנקראת ng-template ובזכותה נוכל בקלות להגדיר תנאי ה ELSE שלנו (קודם ניתן לה תז עם # ולאחר מכן נוכל להשתמש בה ב פקודת ELSE) ונכתוב את הקוד כך:

<h2 \*ngIf="showInfo;else hideInfo">show info is work</h2>

<ng-template #hideInfo>not work</ng-template>

כדי שיהיה קל יותר לבדוק את הדוגמא ואת הדוגמאות הבאות של דירקטיבים נוספים ב HTML נייצר כפתור שישפיע על המשתנה showInfo ב TS.

<button class="btn btn-info" (click)="onBtnClick()">hide/show</button>

וב TS נגדיר את הפונקציה ONBTNCLICK:

onBtnClick(){

this.showInfo = !this.showInfo;

}

שימו לב שהשוונו את ה SHOWINFO לעצמו עם סימן קריאה מה שמייצר סוג של TOGGLE  
  
  
למי ששואל את עצמו איך הוא עושה תנאי SWITCH או ELSE IF נראה בהמשך איך נוכל להגיע למורכבות של תנאי כזה ב אנגולר דרך ה אך קודם בוא נכיר עוד מספר דירקטיבס חשובים בשם ngClass ו ngStyle

**9. א. מה זה ngStyle ומה היכולות המיוחדות שלו?**  
ב NGSTYLE נוכל להשתמש במספר אופנים  
באופן הרגיל נניח אם נרצה לשנות את הצבע של ה H1 נוכל לכתוב את הפקודה הבאה:

<h1 [ngStyle]="{color:'red'}"> Welcome to {{ title }}! </h1>

\*\* שימו לב שהפקודה NGSTYLE עצמה נמצאת בסוגריים מרובעים של PROPERTY BINDING אבל NG STYLE עצמו  
הינו דיירקטיב  
 בנוסף נוכל לייצר לעצמנו משתנה שנקרא txtColor ב TS ששווה ל BLUE ולהשתמש בו:

txtColor = "Blue";

וב VIEW בHTML על מנת להשתמש בו כל מה שנצטרך לעשות זה לכתוב אותו ללא גרשיים (בשונה מהדוגמא הקודמת)

<h1 [ngStyle]="{color: txtColor}"> Welcome to {{ title }}! </h1>

אפשרות נוספת שנוכל להשתמש בNGSTYLE היא להשתמש בפונקציה שמחזירה לנו ערך ולפי המצב של ה SHOWINFO הוא יציג לנו צבע אדום או כחול

getShowInfoColor(){

let rt = "";

(this.showInfo) ? rt = "blue" : rt = "red";

return rt;

}

ב HTML נשנה את הקוד ל:

<h1 [ngStyle]="{color: getShowInfoColor()}"> Welcome to {{ title }}! </h1>

לאחר שהבנו איך ה NGSTYLE עובד הגיע הזמן להבין כיצד ה NGCLASS עובד (הסינטקס די זהה רק שבמקום מאפיינים נשתמש בקלאסים של CSS)  
  
**9. ב. מה זה NGCLASS**  
על מנת שנוכל להשתמש בפקודה NGCLASS קודם כל עלינו להוסיף CLASS בקובץ CSS של הקומפנינטה של ה APP  
app.component.css שימו לב להשתמש במקף תחתון בלבד בשמות של הקלאסיים

.red\_box{

background-color: pink;

border:3px solid red;

padding:1em;

margin-bottom: 1em;

}

.green\_box{

background-color: greenyellow;

border:3px solid green;

padding:1em;

}

לאחר מכן נייצר תגית H3 ב HTML של ה APP שתראה כך ותקבל באופן הכי פשוט את הפקודה NGCLASS

<h3 [ngClass]="{red\_box:true}">ng class test</h3>

אך מכיוון שאנחנו בתוך BINDING נוכל גם להשתמש במשתנה בשביל להגדיר אם להציג את הקלאס או לא:

<h3 [ngClass]="{green\_box:showInfo}">ng class test</h3>

ואפילו נוכל לעשות סוג של TOGGLE בין קלאסיים בצורה הבאה ודי בקלות:

<h3 [ngClass]="{green\_box:showInfo,red\_box:!showInfo}">ng class test</h3>

\*\* בנוסף בדומה ל NGSTYLE נוכל לייצר פונקציה שתחזיר את הקלאס שנרצה (לא מוצג במדריך)

**אתגר:** צור 3 קלאסים ב CSS ובתוך ה TS צור מערך שיכיל בסטרינג את 3 המחלקות שיצרת.  
בHTML צור כפתור NEXT שבכל פעם שנלחץ עליו הוא יעבור למחלקה הבאה שתשפיע על H4 ותגרום לו כל פעם להיות מושפע מקלאס אחר לפי הסדר במערך , כאשר ניהיה בתא השני ונלחץ נקסט הוא יחזור לתא 0  
  
  
 **10. שימוש ב NGFOR:**

לרוב נשתמש בפקודה זו על מנת להציג מידע למשתמש שמגיע ממערך שיש לנו בTS (יכול להיות גם JSON שמגיע מREST API/AJAX) בדומה לפקודה **FOR OF** ו **MAP** של מערך

לכן בשביל הדוגמא בקובץ ה TS ניצור משתנה של מערך של פירות:

fruits\_ar = ["apple","banana","kiwi","orange"];

ובוא ננסה לייצר אותם כרשימה אז בתוך ה HTML שלנו.  
 נייצר רשימת UL ובתוכה פקודה מיוחדת בתוך תגית LI אחת:

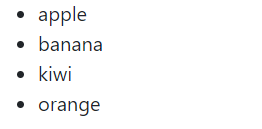
<ul>

<li \*ngFor="let item of fruits\_ar">

{{item}}

</li>

</ul>

והתוצאה:  


בהמשך המדריך נראה כיצד השימוש בפקודה NGFOR עוזר לנו לאסוף מידע מ JSON שקיבלנו ולייצר אותו די בקלות  
**אתגר:** צור מערך של תמונות שישבו בתוך תיקיית ASSETS באפליקציה ועל ידי פקודת NGFOR צור גלריית תמונות.

על מנת להוציא את האינדקס בלולאה עלינו להוסיף פקודה בתוך הNGFOR

<li \*ngFor="let item of fruits\_ar; let i = index">

{{i}} - {{item}}

</li>

**אתגר** : צור רשימת פירות (8 פירות) והצג אותם , צבע הרקע של כל פרי שני יהיה בצבע צהוב שים לב שלא ניתן לשים 2 STRCTUAL DIRCTIVE על אותו אלמנט ולכן מומלץ להשתמש בSPAN פנימי לפקוד ה NGIF

**1****1. שימוש ב NGSWITCH**   
כפי שצויין קודם במידה ונרצה לעשות IF מורכב יותר אם סוג של ELSE IF נאלץ להשתמש בפקודת דיירקטיב בשם NGSWITCH   
והשימוש בה יהיה בצורה הבאה :

<ul>

<li \*ngFor="let item of fruits\_ar; let i = index" [ngSwitch]="item">

<span \*ngSwitchCase="'apple'" [ngStyle]="{color:'red'}">

{{i}} - {{item}}

</span>

<span \*ngSwitchCase="'kiwi'" [ngStyle]="{color:'blue'}">

{{i}} - {{item}}

</span>

<span \*ngSwitchDefault>

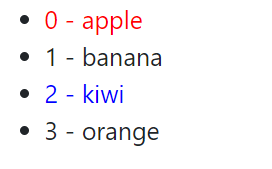
{{i}} - {{item}}

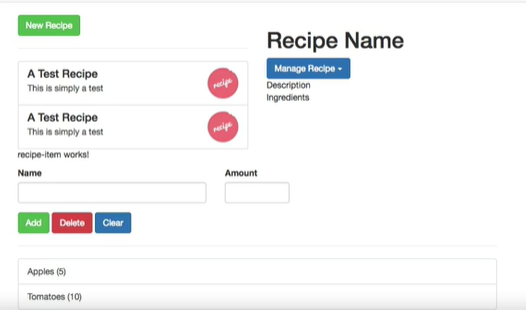
</span>

</li>

</ul>

שימו לב שב LI שמנו דירקטיב בתוך EVENT PROPERTY של NGSWITCH שבודק את הITEM (במקרה שלנו שם הפרי )

ובתוכו הכנסנו את האופציות בספאן , במידה והערך לא שווה ל KIWI או APPLE הברירת המחדל הינה NGSWITCHDEFAULT והתוצאה הסופית תיהיה:  


אתגר צור אפליקציה (תחשוב אילו קומפנינטות לייצר שבסופם יראו ככה אין צורך שהם יעבדו פונקציונלית):  


**12. מה זה AUGURY וכיצד לבצע DEBUG**על מנת שנוכל לבצע דיבאגינג ((DEBUG לקוד שלנו ולהבין טוב יותר מה קורה באפליקציה מומלץ שנוריד את התוסף לכרום בשם AUGURY שנלחץ F12 בכרום בנוסף ל console יתווסף טאב עם augury שייתן לנו מידע על האפליקציה מבחינת משתנים, פונקציות וSERVICES ב קומפנינטות ועוד

**13.א BINDING CUSTOM PROP**  
עד עכשיו יצרנו אפליקציות פשוטות שרובן בנויות על קומפנינטה אחת או לחלופין אין צורך לתקשורת בין הקומפונינטות  
  
בחלק הבא נבנה קודם אפליקציה פשוטה ביותר של רשימת קניות שהמשתמש יכול להוסיף לעצמו על ידי הכנסת מידע ל INPUT . בתחילת האפליקציה יהיה מערך שבתוכו יש כבר רשימה 2 מוצרים  
  
נתחיל לייצר אפליקציה מ0 (ממליצים לכם לשמור את תיקיית הSRC בתיקיית BACKUP ולהתחיל פרוייקט חדש)  
  
בתור התחלה בקומפיננטת ה app בקובץ ה html נכניס את המידע הבא:

<div class="container" style="text-align:center">

<h2>

Welcome to {{ title }}!

</h2>

<div class="user\_inputs">

<h3>Enter your product to the list:</h3>

<div class="input-group px-5">

<input type="text" class="form-control" [(ngModel)] = "fruit\_input" >

<button class="btn btn-info" (click)="addProduct()">add product</button>

</div>

</div>

<div class="list mt-5 p-5 text-left">

<h2>List of product you added</h2>

<div class="row">

<div class="col-7 border p-2" \*ngFor="let item of listFruits\_ar" >

{{item}}

</div>

</div>

</div>

</div>

\*\* חשוב לחזור לסעיף 6 ובקובץ app.module.ts לראות שהוספנו את המודול של formsModule על מנת שנוכל להשתמש בngModel בINPUT ולאסוף ממנו מידע (בדומה לfruit\_input.value ב JS רגיל)

ובתוך קובץ ה ts של ה app נכניס את הקוד :

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Alias events and props output and input';

listFruits\_ar = ["apple","mango"];

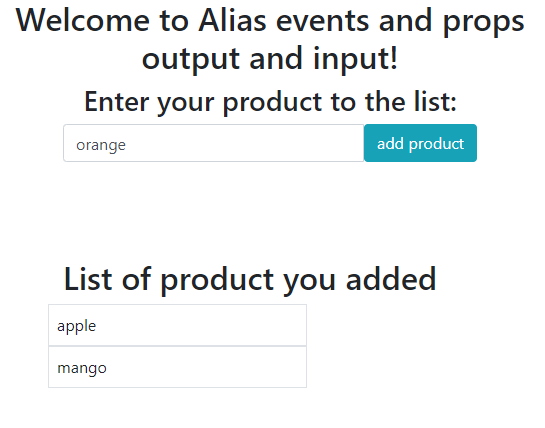
fruit\_input = "orange";

addProduct(){

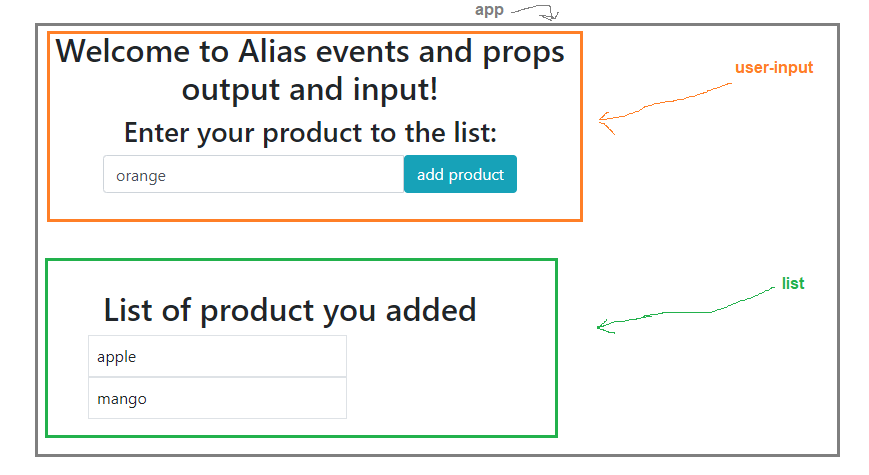
this.listFruits\_ar.push(this.fruit\_input)

}

}

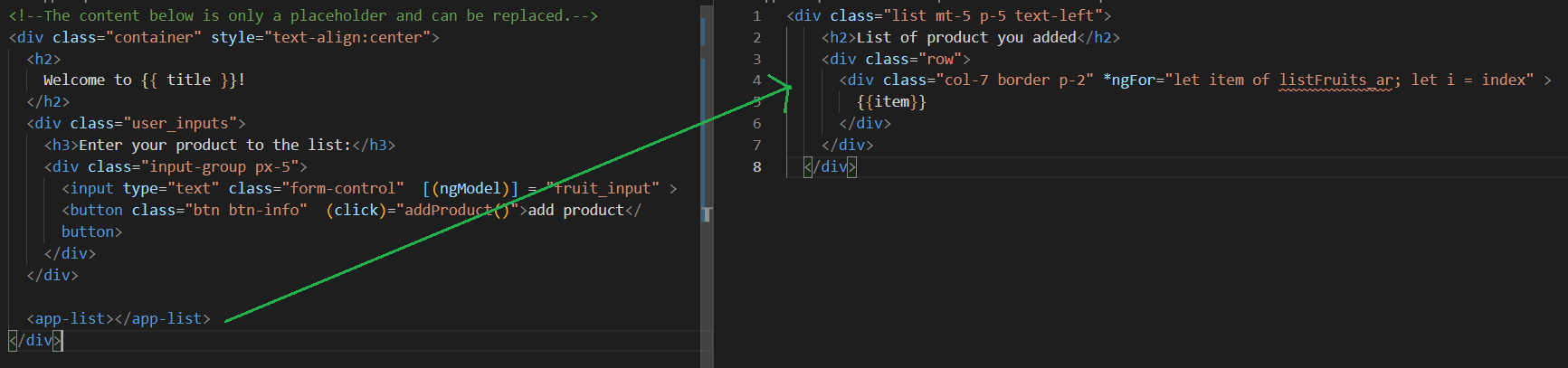
שימו לב שמכיוון שהכנסנו בהתחלה בתוך המחלקה למשתנה fruit\_input את הערך orange כאשר נריץ את האפליקציה הדבר הראשון שנראה בתוך הINPUT זה את המילה orange . והתוצאה תראה כך:  


שנלחץ על כפתור ה ADD PRODUCT שימו לב שנפעיל את הפונקציה בשם addProduct() שתוסיף למערך listFruits\_ar פריט נוסף ואוטמטית פקודת ה NGFOR תדע להציג את הפריט הנ"ל גם .

עכשיו המשימה שלנו מעט יותר מורכבת : כיוון שאנחנו רוצים לחלק את האפליקציה למספר קומפנינטות ואז ננסה להבין איך נוכל להעביר מידע מקומפונינטה בת לקומפונינטה הורה  
  
אז לפני שהאפלקציה שלנו תיהיה מורכבת מ3 קומפנינטות (APP, list,user-input) קודם נייצר את הקומפנינטה הירוקה (LIST) ו"הקומפונינטה הכתומה עדיין תישאר בתוך ה app .

נזכיר לכם ב CLI לכתוב את הפקודה הבאה:  
ng g c comps/list

ונעביר את כל ה DIV של list mt-5 לתוך הHTML של הקומפנינטה LIST

ובתוך ה HTML של APP נוסיף את התגית app-list (הסלקטור ברירת המחדל שה CLI מייצר לקומפ) ככה שהתוצאה של הקוד צריכה להראות כך:  


עכשיו כשנריץ את האפליקציה היא פשוט לא תעבוד מהסיבה הפשוטה שהקומפ LIST לא יודע מהו המערך:  
listFruits\_ar  
  
כדי לבצע את המהלך הבא אנחנו נעביר את המערך שלנו מ APP לקומפ list דרך הפקודה property binding וכל מה שנצטרך בשלב ראשון זה בhtml של הקופמ app להוסיף בשורה של התגית app-list

<app-list [listFruits\_ar]="listFruits\_ar"></app-list>

הבעיה שנקבל טעות פאטלית שאנגולר יזעק לנו שהוא לא מכיר דירקטיב או אליאס בשם listFruits\_ar בתוך ה TS של הקומפ list   
אז כדי שהקומפ list יכיר את המשתנה listFruits\_ar על פניו אנחנו צריכים להוסיף לתוכו את המשתנה , אך הבעיה שעדיין הקומפ list לא ידע שזה אמור להיות משתנה שמגיע כ PROP מה APP , לכן כדי לפתור את הבעיה נכתוב את הקוד הבא ב ts של הקומפ list ופשוט נוסיף הפקודה @INPUT לפני שם המשתנה

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

@Component({

selector: 'app-list',

templateUrl: './list.component.html',

styleUrls: ['./list.component.css']

})

export class ListComponent implements OnInit {

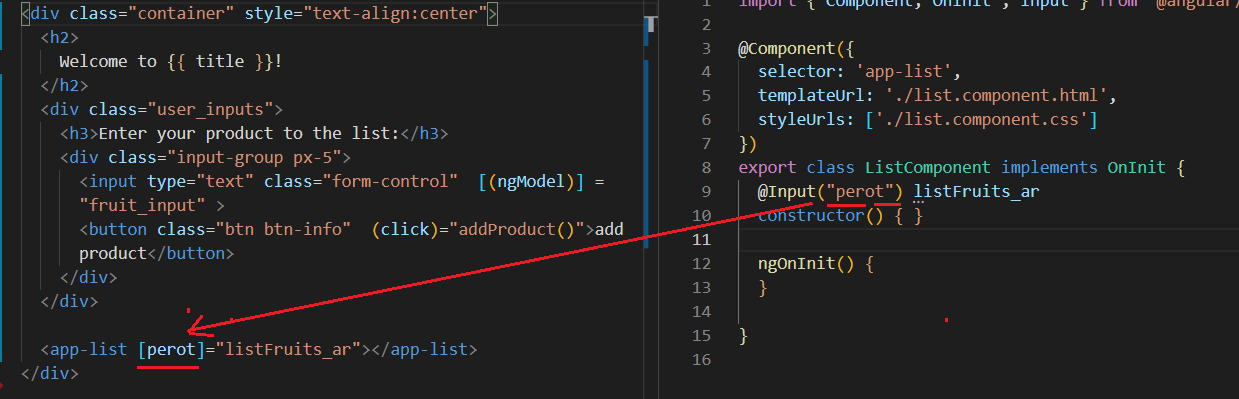
@Input() listFruits\_ar

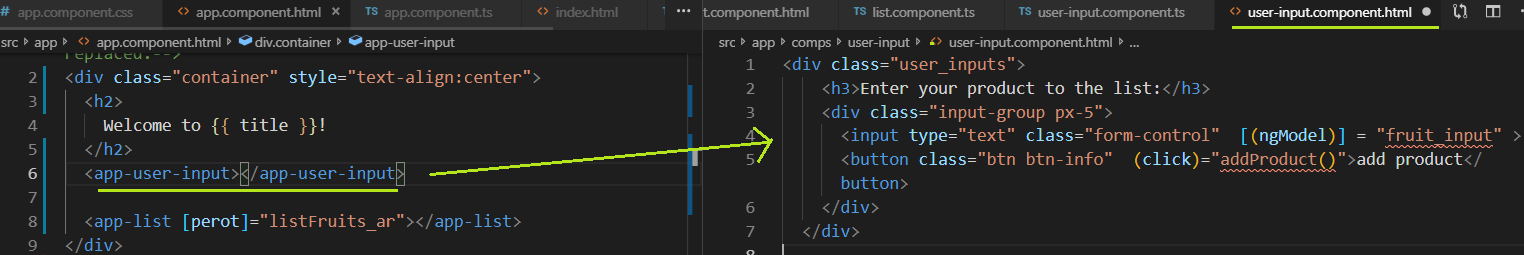
constructor() { }

ngOnInit() {

}

}

שימו לב שעשינו IMPORT למודול INPUT   
המודול הנ"ל הינו DECRATOR בדומה ל @COMPONENT אך הוא נותן לנו את היכולת לקשר בין PROP שהוספנו בתגית אב למשתנה בקומפ ילד , והכתיבה שלו די פשוטה כפי שאתם רואים.  
כרגע כאשר נפעיל את האפליקציה נוכל לראות שהיא עובדת באופן מושלם !  
  
  
**13.ב איך לשנות את כינוי (ALIAS) ה PROP** ב PROP BINDING שיצרנו  
בנוסף לפעמים נרצה אולי שהשם של הPROP יהיה שונה משם המשתנה שהוא מעביר לו מידע אז כל מה שנצטרך לעשות הוא לרשום את השם בסוגריים של ה INPUT ובPROP ב APP לתת את השם החדש לדוגמא:  
  
  
  
**14. העברת CUSTOM EVENT בין קןמפינינטות:**

עכשיו הגיע הזמן להעביר את חלק ה INPUT לקומפ משלו בשם user-input (הקומפנינטה הכתומה מהתרשים בסעיף 13) ושם נראה שניתקל בבעיה אחרת (העברת המידע מאינפוט של המשתמש בקומפ בת החדשה למערך הפירות שיהיה קיים בקומפ APP )  
אך קודם נייצר את הקומפ דרך הCLI כרגיל עם הקוד הבא:  
ng g c comps/user-input  
ונעביר לתוכו מhtml של הapp את כל המידע של תגית div class="user\_inputs" ובHTML של הAPP נכתוב את התגית של הקומפ החדש שלנו ככה שהקוד יראה כך:  


כאשר נריץ את האפליקציה שוב נוכל לראות שגם המילה orange לא רשומה כבר בקומפנינטה כי לקומפ user-input אין את המשתנה מוגדר ובנוסף לחיצה על הכפתור תזרוק לנו שגיאה פאטלית מכיוון שכרגע הקומפ לא מכיר כלל את הפונקציה addProduct שגם נוסיף ל TS שלו, לכן המחלקה בTS בשלב ראשון של user-input צריך להראות כך:

export class UserInputComponent implements OnInit {

fruit\_input = "melon"

constructor() { }

ngOnInit() {

}

addProduct(){

// send the input value to app comp

}

}

שימו לב שהוספנו את fruit\_input בשביל הדוגמא עם ערך חדש בשם מילון והוספנו מיטודה בשם addProduct

כרגע כשנריץ את האפליקציה לא נקבל ERROR פטאלי אך גם לא יעבור למערך שלנו הערך החדש בתוך האינפוט לאחר שנלחץ על הכפתור ADD PRODUCT  
  
דבר ראשון ב ts של ה קומפ APP נוסיף במחלקה את הפונקציה הבאה ואפשר למחוק את הפונקציה ADDPRODUCT ממנו ואת FRUITS\_INPUT מכיוון שכרגע הם מוכרזים בתוך הקומפ user-input   
ונייצר פונקציה שתקבל פרמטר בשם VAL שיתווסף למערך בשם addProductFromApp (כדי שיהיה קל בדוגמא לראות את ההבדל בשמות בין הפונקציה שיצרנו בקומפ user-input ובקומפ app )  
והתוצאה שהקומפ APP בקובץ TS תראה כך:

export class AppComponent {

title = 'Alias events and props output and input';

listFruits\_ar = ["apple","mango"];

//fruit\_input = "orange";

addProductFromApp(val){

this.listFruits\_ar.push(val)

}

}

ב HTML של ה APP נוסיף PROP BINDING על התגית של הקומפ user-input את הקוד הבא:

<app-user-input (addProdEvent)="addProductFromApp($event)"></app-user-input>

שימו לב שהעברנו ב EVENT BINDING את הפונקציה addProductFromApp באירוע שהמצאנו בשם addProdEvent (מטעמי לימוד שיהיה קל להבין שזה EVENT) ואם נשים לב העברנו לפונקציה ARGUMENT בשם $EVENT שמוטמע בילד אין בתוך שפת האנגולר על מנת שנוכל להעביר פרמטרים ומשתנים משיגור הEVENT לפונקציה וכך פרמטר VAL  
יוכל לקבל את המידע   
  
עכשיו על מנת שהקומפ user-input יכיר את ה EVENT נאלץ לכתוב את הקוד הבא ב TS שלו:

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

@Component({

selector: 'app-user-input',

templateUrl: './user-input.component.html',

styleUrls: ['./user-input.component.css']

})

export class UserInputComponent implements OnInit {

@Output() addProdEvent = new EventEmitter();

fruit\_input = "melon"

constructor() { }

… המשך קוצר מטעמי כתיבת החוברת

}

שימו לב שדבר ראשון עשינו יבוא לשתי מודלים חדשים OUTPUT שהוא בדיוק ההפך מה INPUT@ מהסעיף הקודם , כי במקרה הזה אנחנו נרצה לשגר החוצה לקומפ האב APP את התרחשות האירוע . והמודול השני בשם EVENTMITTER שנבין עוד מעט כיצד לשגר איתו משתנים לפונקציה של הקומפ אב APP .  
  
לאחר מכן נוכל לראות שכתבנו את השורה הבאה:

@Output() addProdEvent = new EventEmitter();

שבעצם מגדירה לקומפ user-input כך שהוא יכיר את האירוע addProdEvent וגם אנו מגדירים אותו כ   
EventEmitter מה שיתן לו את ההבנה שהוא מסוג TYPE EVENT ובנוסף את היכולת לשגר פרמטרים (במקרה שלנו את השדה קלט שהמשתמש מילא בשם של פרי) לקומפ האב APP  
  
עכשיו כל מה שנשאר לנו כדי שהאפליקציה שלנו תעבוד באופן מושלם עם חיבור כל הקומפנינטות זה להוסיף לפנוקציה ADDPRODUCT בקומפ user-input את הפקודה הבאה:

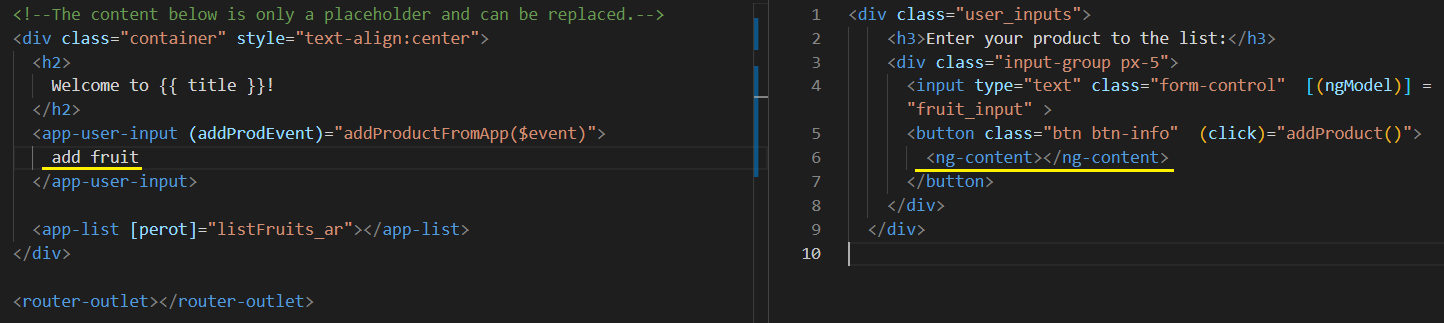
addProduct(){

this.addProdEvent.emit(this.fruit\_input);

}

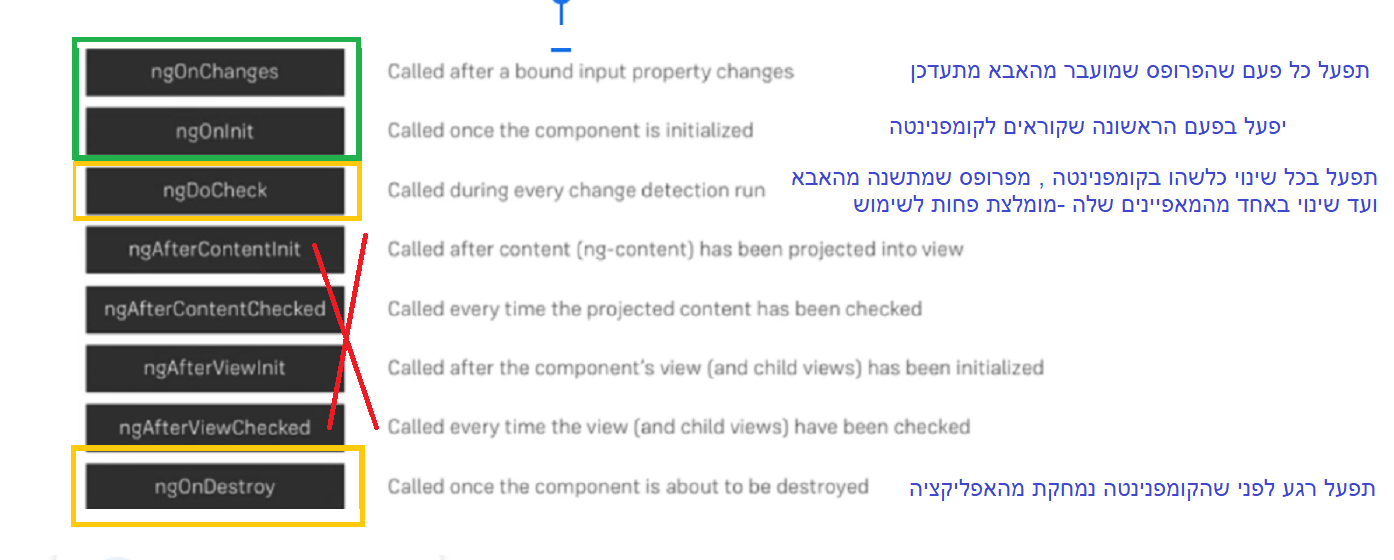
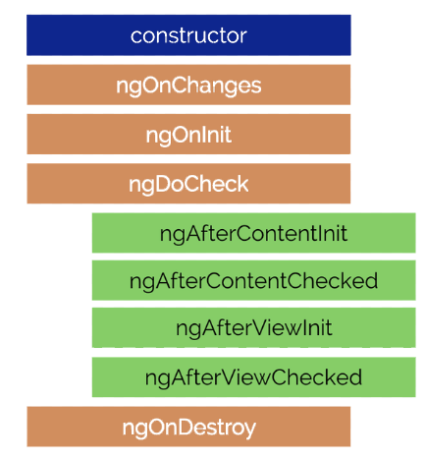
שימו לב שהמיטודה EMIT משגרת את האירוע לקומפ אב ובנוסף מעביר את הפרמטר שנמצא בסוגרים ,   
\*\* במידה ותרצו לשגר מעבר לפרמטר אחד , אנו ממליצים לעשות את זה עם משתנה מסוג מערך או אובייקט

וזהו, האפליקציה שלנו כרגע עובדת מצויין והגיע הזמן להבין עוד כמה דברים חשובים !  
בהמשך נבין שלעבוד בשיטה כזו בשביל להעביר מידע בין קומפנינטות היא פעולה מסובכת ברגע שמתחיל להיות קומפנינטות נכדים , נינים וכו'... ושם כבר נשתמש ביכולת SERVICE שבהמשך המדריך נדבר על יכולת זו.  
  
**15. מה זה ng-content**   
במקרים מסויימים נרצה כשנשתמש בתגית הקומפ להעביר מידע שירשם בה לא כ PROP BINDING אלא כ   
INNERHTML

על מנת שנוכל להעביר את המידע בצורה הזאת , נשתמש באפליקציה שעשינו ונייצר לוגיקה שה INNERHTML שנמצא בתוך הקומפ app-user-input ישמש כהטקסט שרשום על הכפתור , בתוך הHTML של הקומפ של app-user-input נכניס את התגית ng-content וכך אנחנו יכולים להזריק INNERHTML לקומפ שלנו  
  
והתוצאה כאשר נריץ את האפליקציה היא שהביטוי ""add fruit יופיע על הכפתור.

**16. מבינים מה זה HOOKS LIFE CYCLE**

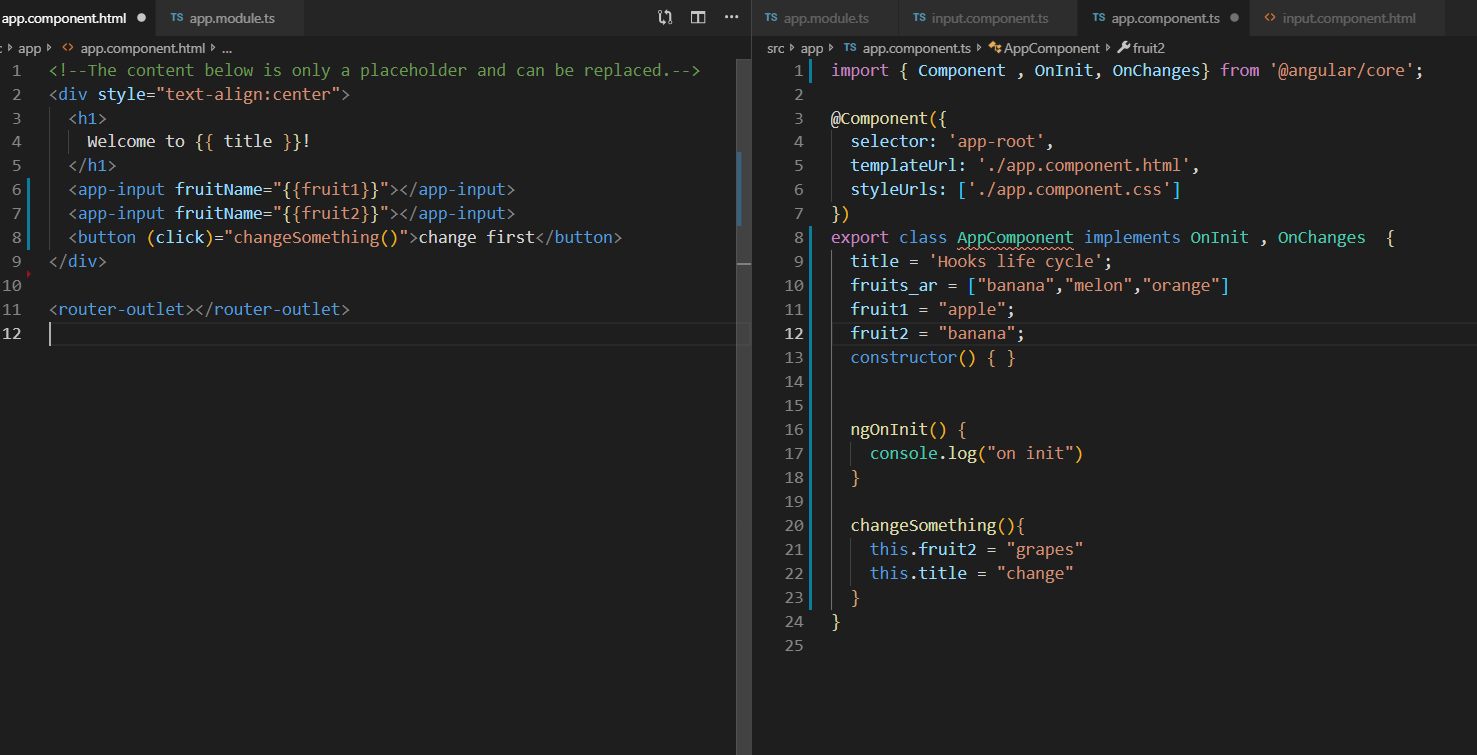
אחד מהדברים החשובים שחשוב לדעת בקומפנינטות באנגולר הינם HOOKS (שהם סוג של פונקציות) שפועלות במהלך חיי הקומפנינטה , להלן הסדר של ה פונקציות ה HOOKS

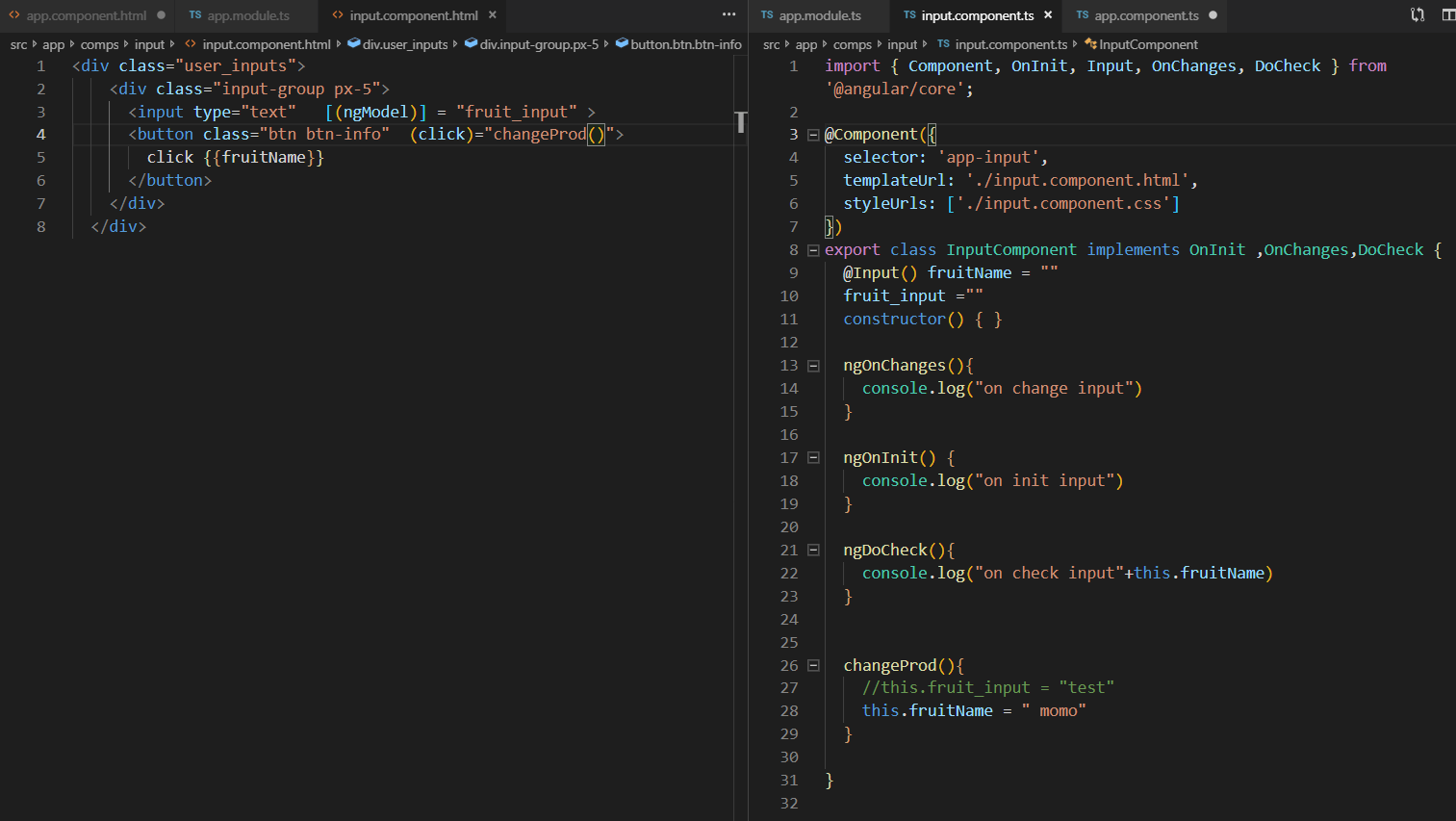


בחלק זה נתייחס בעיקר ל HOOKS בצבע חום /כתום מכיוון שנשתמש בעיקר בהם במהלך הפיתוח:  
  
**ngOnChanges-** הוק שנקרא כל פעם שישנו שינוי ממשי בתוך הקומפונינטה שאנחנו עובדים עמה מקומפנינטת האב, כגון PROP BIND שמשתנה (שימו לב שאם הקומפ נוצר על ידי לולאה כגון NGFOR הוא יפעיל גם את ההוק NGONINIT) הוק זה יפעל גם בפעם הראשונה שקומפ ילד פועל. מקביל ב REACT רגיל ל **COMPONENT DID UPDATE**

**ngOnInit** – יפעל בפעם הראשונה בתחילת הקומפנינטה עוד לפני שהיא הציגה מידע ובכל פעם שיהיה שינוי בקומפנינטה ב PROP BIND שלה מקומפנינטת אב - מקביל ב REACT רגיל ל **COMPONENT DID MOUNT**

**ngDoCheck** – כל שינוי יפעיל את הפונקציה הזאת גם בקומפנינטות אחיות שלה – לא מומלצת לשימוש כלל רק בקומפנינטות אב ראשיות

**ngOnDestroy** – יקרא פעם אחת לפני הקומפנינטה מושמדת/נמחקת (נניח ברשימת קניות המשתמש מחק את המוצר) יכול לשמש אותנו רגע לפני שהקומפנינטה נמחקת לשים אותה בהיסטוריה או לשלוח קוד אנאליטקס של סטטיסטיקה עם המחיקה ועוד....  
לבדיקה בואו נייצר ב APP את הקוד הבא ונייצר קומפ בשם input   


ה קומפ INPUT יראה כך:  
  
  
\*\* נוכל לראות שבשביל להשתמש בהוקים אנו ניהיה חייבים קודם ב TS לעשות להם IMPORT ולאחר מכן לציין אותם כ  
implements בתוך המחלקה.  
כשנריץ את האפליקציה נוכל לראות שההוק INIT של הקומפ app פועל לבדו רק כאשר נפעיל את האפליקציה ולעומת זאת בקומפ input של הבנים שלו גם ה הוק CHANGE וה INIT פועלים בהתחלה,  
אם נלחץ על הכפתור ב APP בשם CHANGE FIRST נראה שרק ההוק CHANGE פועל בקומפנינטה INPUT הראשונה שה PROPBIND בשם FRUITNAME שונה אצלה

\*\* בנוסף בתוך הקומפ Input אם נלחץ על כפתור השם שמשנה את המלל בתוכה נראה שהדבר יקרא להוק CHECK ,   
הבעיה היחידה שההוק הזה יקרא מספר פעמים כמספר האחים של הקומפ (במקרה הנל יש 2 קומפ אחים) מה שיכול באפליקציה מורכבת להאיט את האפליקציה ולכן חשוב להשתמש בהוק הנ"ל בחוכמה.

**17. מה זה HOSTLISTENER DIRECTIVE**

נייצר שוב אפליקציה APP חדשה והפעם נייצר לה דירקטיב משלנו שבנוסף לשינוי צבע של הרקע יבצע שגם שמעבר? עם העכבר ישנה נניח את הצבע הטקסט ובשביל זה נלמד משהו שנקרא HOSTLISTENER שמאפשר להאזין לאירועים באלמנט דרך דירקטיב שמוזרק לו   
  
א. נייצר דירקטיב בדומה לסעיף 7 הפעם אבל בשם better-green   
ng g d dir/better-green

בתוך הדירקטיב נכתוב את הקוד הבא:

import { Directive, OnInit , ElementRef, HostListener} from '@angular/core';

@Directive({

selector: '[appBetterGreen]'

})

export class BetterGreenDirective implements OnInit {

constructor(private elementRef: ElementRef) { }

ngOnInit(){

// פקודה של דיבור עם האלמנט שעליו יהיה את הדרקטייב שיהפוך את הרקע שלו לירוק

this.elementRef.nativeElement.style.background = "green"

this.elementRef.nativeElement.style.color = "white"

}

@HostListener('mouseenter') mouseover(){

this.elementRef.nativeElement.style.color = "red"

}

@HostListener('mouseleave') mouseleave(){

this.elementRef.nativeElement.style.color = "white"

}

}

\*\* בשונה מסעיף 6 הפעם השתמשנו בדיקורטור בשם hostlistener שבעצם מאפשר לנו להאזין לאירועים שונים במקרה של הקוד הנ"ל לכניסה של העכבר ויציאה של העכבר. בנוסף ניתן לראות שבהמשך לסעיף 6 אנו משנים את צבע את הגופן של האלמנט.  
רק נזכיר שב VIEW של הHTML פשוט נצטרך להוסיף האטרביוט לפי הסלקטור על האלמנט שנרצה לשנות לדוגמא בתגית H2 שנמציא:

<h2 appBetterGreen>Directives</h2>

**17. ב. שימוש ב HOSTBINDING**  
בנוסף ישנה דרך חכמה יותר לדבר עם מאפיין של אלמנט ששמנו עליו את במקום להשתמש ב ELEMENTREF עם כל המאפיינים שלו והדרך הזאת נקראת HOSTBINDING ושימו לב שבקוד הבא אנחנו גם חוסכים את NGONIT עם הגדרות התחלתיות וגם הכתיבה הרבה יותר נוחה:

import { Directive, OnInit , ElementRef, HostListener, HostBinding} from '@angular/core';

@Directive({

selector: '[appBetterGreen]'

})

export class BetterGreenDirective implements OnInit {

@HostBinding('style.color') clr = "white"

@HostBinding('style.background') bg = "greenyellow"

constructor(private elementRef: ElementRef) { }

ngOnInit(){ }

@HostListener('mouseenter') mouseover(){

this.clr = "yellow";

}

@HostListener('mouseleave') mouseleave(){

this.clr = "white";

}

}

**17. ג. שימוש ב PROP BINDING בדריקטיב**: לעיתים קרובות כאשר נייצר דירקטיב נרצה שתהיה לנו את האפשרות להעביר לו ערכים של מאפיינים (נניח את צבע הרקע במקרה שלנו ) שימו לב שזה יעבוד ממש בדומה בPROP BINDING שאנחנו מייצרים בקומפנינטה: בתוך הקלאס שלנו נגדיר את INPUT של BG שיגדיר את צבע הרקע.

export class BetterGreenDirective implements OnInit {

@Input() bgColor = "greenyellow"

@HostBinding('style.color') clr = "white"

@HostBinding('style.background') bg = "greenyellow"

constructor(private elementRef: ElementRef) { }

ngOnInit(){ this.bg = this.bgColor }

@HostListener('mouseenter') mouseover(){

this.clr = "yellow";

}

@HostListener('mouseleave') mouseleave(){

this.clr = "white";

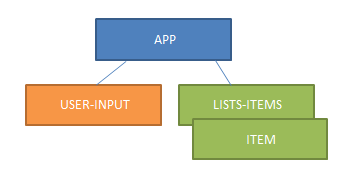
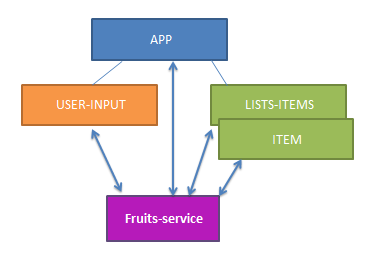
}

}

כרגע בקוד הנ"ל קבענו את הצבע ברירת מחדל ל greenyello ובהוק INIT השתמשנו בו , אך אם נרצה לקבוע לאלמנט שמשתנה בדירקטיב צבע אחר (נניח אפור) כל מה שנצטרך לעשות זה להוסיף בתוכו את הPROP BINDING כך:

<h2 appBetterGreen [bgColor]="'grey'">Directives</h2>

**18. SERVICES**  
הגיע הזמן לעבור לנושא אחד החשובים באנגולר והוא SERVICES שמאפשר לנו באפליקציות מורכבות להזרים משתנים ופונקציות בין קומפנינטות אב לקומפנינטות בנים ,נכדים ,נינים וכו'.... הלוך וחזור.  
  
תקציר יתרונות ה SERVICE :

א. קלאסיים שיכולים לשלוח פונקציליות ומשתנים למספר קומפנינטות שידעו לקבל ולהחזיר להם מידע  
ב. שומרת על קומפנינטות רזות וחוסכת מהם להכיל פונקציות גלובליות ומשתנים גלובים?  
ג. דואגת שלא היה DRY – אל . תחזור. על . עצמך  
ד. בדרך כלל יכיל את הפקודות AJAX/REST API לקריאה לשירות חיצוני  
  
וכדי שיהיה יותר ברור היתרון הגדול של הSERVICE  
נזכיר לעצמנו שבסעיף 13 שרצינו להעביר מידע מקומפ הורה לקומפ בן (מה APP לקומפ LIST) היינו צריכים להשתמש ב @INPUT ,  
וכשרצינו לעדכן משהו מקומפ בן לקומפ הורה (מ user-input לקומפ APP) כמו בסעיף 14 היינו צריכים להשתמש בOUTPUT כל עוד ההיררכיה היא אחת מאב לבן או בן לאב ניתן להשתמש ביכולות הללו .  
  
 אך מהר מאוד תגלו שבחלקים רבים באפליקציה שלכם תרצו להעביר נניח מקומפ APP מידע של מערך לתוך קומפ שנמצא בקומפ בן (מה שהופך אותו לנכד) ולפעמים תרצו גם שיהיה לכם אפשרות לעדכן את המידע בחזור משינוי שמתרחש בתוך הקומפ נכד לקומפ APP ושם SERVICE ממש יעזור לנו  
  
  
ובאפליקציה הבאה שאנחנו הולכים לייצר של רשימת מוצרים שתיהיה מורכבת מ4 קופנינטות בהיררכיה הבאה:  
  
\* די דומה לאפליקציה שעשינו ב חלק 13 רק שהפעם בתוך ה LIST-ITEMS כל מוצר יהיה מורכב עם "מוח" משלו בצורת קומפנינטה בשם ITEM שיכול גם למחוק את עצמו.  
וכאן בדיוק נכנס ה SERVICE כי תדמיינו שאם מה שאנחנו יודעים עד עכשיו ללא הSERVICE היינו צריכים לשגר את המערך שנמצא ל APP לנכד שלו בשם ITEM והנכד היה צריך בחזרה להחזיר לו מידע אם הוא נמחק ... היינו צריכים רק בשביל זה 40 – 50 שורות קוד (כי היינו צריכים לדבר דרך האב שידבר עם הקומפ סבא וכו'...), אך אל דאגה כאן נכנס ה SERVICE  
  
ול SERVICE שלנו נקרא בשם fruits מהסיבה הפשוטה שהוא הולך להכיל את המערך של הפירות והיתרון האדיר שלו הוא שאפשר כבר להבין מהתרשים שהוא יודע לשלוח ולקבל מידע מכל קומפנינטה!  
  
אז קודם כל בוא נייצר את האפליקציה שלנו מ0 ואז נבין איך לקשר בין כל הקומפנינטות עם ה SERVICE  
נייצר קודם את האפליקציה שלנו בדומה למה שעשינו בסעיף 13 (ונכון להסבר ההתחלתי נוותר על הקמופנינטה item)  
  
נכתוב את 2 הפקודות הבאות ב CLI  
ng g c comps/input-user  
ng g c comps/items-list  
  
בתור התחלה בקומפנינטה input-user נכנס רק את ה HTML עם טופס פשוט (בלי לוגיקה בקלאס ב TS) :

<div class="user\_inputs">

<div class="input-group px-5">

<input type="text" class="form-control" [(ngModel)] = "**fruit\_input**">

<button class="btn btn-info">add</button>

</div>

</div>

\*\* לא לשכוח לעשות את סעיף 6 עם ה IMPORT של הFORM על מנת להשתמש ב ngModel

ובקומפנינטה items-list בקלאס נוסיף את המערך הידוע שלנו בתור התחלה בשם fruits\_ar

listFruits\_ar = ["apple","kiwi"]

וב HTML ב view של הקומפ נכניס את הקוד הבא בדומה לסעיף 13:

<div class="list mt-5 p-5 text-left">

<h2>List of product you added</h2>

<div class="row">

<div class="col-7 border p-2" \*ngFor="let item of listFruits\_ar" >

{{item}}

</div>

</div>

</div>

וכמובן נכניס בקומפ APP הראשי את התגיות סלקטור של הקומפנינטות ב HTML:

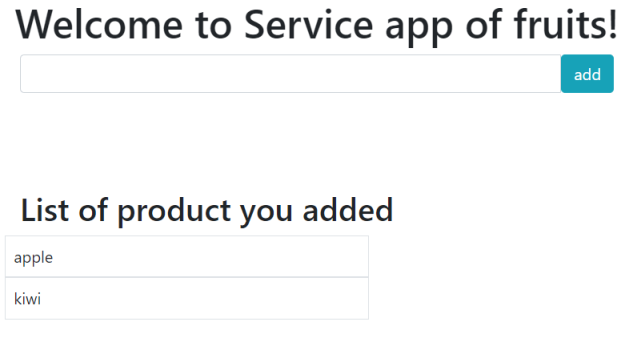
<div style="text-align:center" class="container">

<h1> Welcome to Service app of fruits! </h1>

<app-input-user></app-input-user>

<app-items-list></app-items-list>

</div>

כרגע כשנריץ את האפליקציה היא תראה כך  


הצעד הבא שנעשה זה לייצר את הSERVICE שלנו בשם fruits שכרגע יכיל בתוכו את המערך של הפירות שיקשר בין כל הקומפנינטות אך כרגע במיוחד עם הקומפנינטה של list-items

לכן ב CLI נפעיל את הפקודה הבאה:  
ng g s services/fruits  
  
וכמו ב DIRECTIVES תוכלו למחוק את הקובץ spec שנוצר בתוך התיקייה services ונכניס את המידע הבא ל SERVICE שיצרנו:

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

@Injectable({

providedIn: 'root'

})

export class FruitsService {

fruits\_ar = [];

constructor() {

this.fruits\_ar = ["melon","mango","orange"];

}

getFruits(){

return this.fruits\_ar;

}  
  
 addFruits(\_val){

this.fruits\_ar.push(\_val);

}

}

\*\* שימו לב שהסרביס משתמש במודול בשם INJECTABLE מה שמאפשר לנו לחבר אותו בקלות (לעומת גרסאות של אנגולר 5 ו6 שבהן היינו צריכים להכניס אותו לapp.module.ts ) לקומפנינטות שונות באפליקציה שיוכלו להשתמש בו  
ה PROVIDEDIN בעצם מייצג את שמהשורש של האפליקציה ב קומפנינטת APP נוכל להשתמש בו.  
  
שימו לב שהוספנו בתוך המחלקה שלו את מערך הפירות שלנו ובנוסף יצרנו מתודה שברגע שנקרא לה תחזיר את המערך הנ"ל.  
ובנוסף ישנה מיטודה ADDFRUIT שבה נשתמש בקומפ input-user כדי לדחוף למערך פרי חדש.  
  
כדי לחבר את ה SERVICE נכתוב במחלקה של הקומפ items-list את הקוד הבא:

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

import {FruitsService} from '../../services/fruits.service'

@Component({

selector: 'app-items-list',

templateUrl: './items-list.component.html',

styleUrls: ['./items-list.component.css']

})

export class ItemsListComponent implements OnInit {

listFruits\_ar = ["apple","kiwi"]

constructor(private \_fruitSer:FruitsService) {

this.listFruits\_ar = this.\_fruitSer.getFruits();

}

ngOnInit() { }

}

\*\* שימו לב שקודם כל בשביל להשתמש בSERVICE בתוך הקומפ עושים לו ייבוא לאחר מכן בקונסטרקטור אנו מגדרים אותו בתור פרמטר שקיים בתוך ה קומפ (ה PRIVATE מייצג שבתוך המחלקה אפשר לגשת למשתנה מכל מקום) ובתוך הבלוק של הקונסטרקטור אנחנו פשוט אוספים את המידע לתוך המשתנה listFruits\_ar .  
  
עכשיו הגיע הזמן שמהצד השני בקומפ של input-user נייבא את ה SERVICE ובלחיצת כפתור נזריק לתוך המערך שלו מידע ככה שה items-list יעודכן אוטומטית!  
ודי דומה לitems-list אנו נכתוב את המידע הבא בתוך הקומפ:

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

import { FruitsService } from '../../services/fruits.service';

@Component({

selector: 'app-input-user',

templateUrl: './input-user.component.html',

styleUrls: ['./input-user.component.css']

})

export class InputUserComponent implements OnInit {

**fruit\_input** = "melon"

constructor(private \_fruiteSer:FruitsService) { }

ngOnInit() { }

addFruit(){

this.\_fruiteSer.addFruit(this.**fruit\_input**);

//alert(this.fruit\_input)

}

}

נשים לב שבתוך הפונקציה (שנשים בכפתור בHTML) בשם addFruit אנו קוראים למיטודה addFruit שנמצאת בתוך הקלאס של ה service ושימו לב באיזה קלות חיברנו בין הקומפנינטות שלנו !  
  
משימה: נסו לייצר של פריט?? ברשימה הינו קומפנינט בשם item שיש בתוכו גם כפתור DELETE שלחיצה עליו תעיף אותו מהמערך של הפירות ב SERVICE ככה שהוא ימחק .  
  
  
**19. מה זהPIPES ולמה צריך אותם?**  
  
PIPE על שם הסימן " | " מאפשרים לנו לתת FORMAT למידע שאנו מציגים (כגון הצגה של המידע כ UPPERCASE הצגת סוג מטבע , אחוזים ועוד...) בנוסף בהמשך נבין ונלמד כיצד אנו יכולים לייצר PIPES עם פונקציות משלנו שיכולות לפלטר ולסדר מידע על פי הצורך שלנו, אבל נתחיל עם PIPES בסיסים שהם מובנים באנגולר.  
  
בקוד זה נמשיך מסעיף 18 (לפני המשימה) ונוסיף לעצמנו מערך של סוגי משקאות בתוך ה SERVICE של ה FRUITS   
(אנחנו יודעים שאין הגיון אבל זה כדי לחסוך עכשיו כתיבה של SERVICE חדש ☺ ) בפונקציית ה קונסטרקטור

this.drinks\_ar = [

{name:"beer", price: 3 , alcohol:0.05 ,expired:"2021-05-10"},

{name:"cola", price: 2.5 , alcohol:0 ,expired:"2020-05-10"}

]

בנוסף בקומפנינטה של ה items-list נחבר את רשימת המשקאות ל TS בצורה הבאה:

drinks\_ar = [];

constructor(private \_fruitSer:FruitsService) {

this.listFruits\_ar = this.\_fruitSer.getFruits();

this.drinks\_ar = this.\_fruitSer.drinks\_ar;

}

ובתוך הVIEW של ה items-list ב html נוסיף את השורות:

<div class="row">

<div class="col-6 border" \*ngFor="let item of drinks\_ar">

<h3>{{item.name}}</h3>

<div>Price: {{item.price}}</div>

<div>Alcohol: {{item.alcohol}}</div>

<div>Expired: {{item.expired}}</div>

</div>

</div>

עכשיו הגיע הזמן להסבר איך PIPE עובד נתחיל ונגדיר שהשמות של המוצרים יהיו ב UPPER CASE :

<h3>{{item.name | uppercase}}</h3>

שימו לב שכל מה שהיינו צריכים זה להוסיף את הסימן פייפ "|" ואחריו את המילה UPPERCASE

ועכשיו תכתבו את הקוד הבא ותוכלו להבין מה כל דבר עושה:

<div>Price: {{item.price | number: "1.2" | currency:"USD"}}</div>

<div>Alcohol: {{item.alcohol | percent}}</div>

<div>Expired: {{item.expired | date}}</div>

והתוצאה הסופית הינה:  
  
\*\* נסביר רק את הפילטר NUMBER שבעצם ה1 מייצג כמה ספרות מינימום יהיו בשלמים (אם היינו רושמים 2.2 היינו רואים בבירה למשל את המחיר 03.00 שבמצב שלנו פחות רלוונטי {רלווונטי אולי יותר במשתנים של זמנים})  
והנקודה 2 מייצגת את מספר הספרות לאחר הנקודה העשרונית.

**20. כיצד לבצע פקודת SORT BY או FILTER באנגולר**   
 על פי תקני האנגולר החדשים – מומלץ על פי מפתחי גוגל לבצע את הפעולות SORT ו FILTER בתוך הקלאס של הקומפנינטה בלבד, ולכן אם ידוע לכם שאתם צריכים לבצע סורטינג אנו ממליצים לכם להשתמש במודול LODASH שקודם תצטרכו לעשות התקנה:  
npm i lodash @types/lodash

ודוגמא לשימוש נניח ל SORT לפי מאפיין השם:

נניח ונרצה לגרום למערך המשקאות להיות מסודר לפי מחיר כל מה שנצטרך לעשות זה לכתוב ב קונסטרקטור את הפקודה הבאה:

this.sortBy(this.drinks\_ar,"price");

לגבי פילטור תוכלו לייצר את הפונקציה שאתם חפצים בה ופשוט להשתמש במיטודת FILTER שיש בתוך מערכם ב JS ניתן [לצפות כאן כיצד המיטודה](https://www.youtube.com/watch?v=nRKJ29tqoq0&list=PLP-UKVDk32Jh2QZmbqPb_r0uhdviMPvVC&index=6&t=100s) של הפילטור עובדת.

**21. מה זה OBSERVABLE ולמה צריך אותם בשביל לבצע REQUEST לAPI.**

עוד מעט נגיע לחלק המעניין של API REQUEST  
  
אך לפני זה חשוב להבין מה זה OBSERVABLE ולמה אנחנו צריכים אותם. בסעיף הבא נדבר על בקשות!  
  
נניח שנרצה שמערך המשקאות שלנו בקומפנינטה (כמו בסעיף הקודם) יקבל את המידע שלו רק כאשר ה SERVICE קיבל את המידע מ API מסויים אחרי קריאת REQUEST/AJAX כלשהו ... כרגע אנחנו קצת בבעיה קטנה מכיוון שאין לנו מושג איך להאזין לאירוע או לפעולה שהתבצעה מקומפנינטה/SERVICE א' לקומפנינטה ב' ובנוסף נניח שה SERVICE לא עושה בקשה אחת אלא מספר בקשות ואני רוצה רק לאחר שהוא הצליח לבצע את כל הבקשות רק אז שישלח לי את המידע המלא שהוא מכיל בתוכו  
  
בשביל להמחיש מקרה התחלתי פשוט של שימוש ל OBSERVABLE , אנו נייצר OBSERVABLE שכל שניה למשך 3 שניות אשלח 3 הודעות שונות.  
 בואו נייצר קודם SERVICE בשם crypto (שישמש אותנו בהמשך לפנייה ל API של שערי מטבעות קריפטו כגון ביטקויין אטיריום ועוד...) לכן קודם ב CLI נכתוב את הפקודה הבאה:  
ng g s services/crypto  
  
בקובץ הSERVICE שנוצר נוסיף את הפקודות הבאות:

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

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class CryptoService {

data = "";

constructor() { }

}

שימו לב שאנו נייבא את הפקודה observable מתוך המודול rxjs (נדבר עליו בהמשך)  
ובנוסף יצרנו משתנה בשם DATA שישמש אותנו כ OBSERVABLE למידע שבמקרה הבא נשלח כל שניה.  
  
לאחר מכן בואו נייצר את הפונקציה הבאה בתוך הקלאס של ה OBSERVABLE:

getTestData(){

this.data = new Observable(obs => ***{***

setTimeout(() => {

obs.next("bit 1000")

},1000)

setTimeout(() => {

obs.next("bit value 1002")

},2000)

setTimeout(() => {

obs.next("bit value 3000")

},3000)

***}***)

return this.data;

}

שימו לב שכאשר נשתמש ב OBSERVABLE נקרא לפוקנציה (שבמקרה הנ"ל העברנו לה פרמטר בשם obs)  
הפרמטר OBS מכיל בתוכו את האפשרות לבצע את פקודת ה NEXT שתשגר סוג של "אירוע" עם המידע שנרצה (נניח אחרי שניה שלחנו את המידע "bit 1000" ) ובסוף הפונקציה של getTestData אנו מחזירים את המשתנה DATA שמכיל בתוכו את האפשרות להאזנה כל פעם שמתבצע בתוך המאפניינים שלו את הפונקציה NEXT)  
  
כדי לראות איך הפקודה עובדת נייצר לעצמנו את הקומפנינטה crypto-list ונחבר אותו ב VIEW של ה APP  
ng g c comps/crypto-list  
  
ב VIEW של ה APP נוסיף את השורה הבאה:

<app-crypto-list></app-crypto-list>

וכדי להאזין לו בקלות נחבר את הSERIVCE ל קומפנינטה של הקריפטו בצורה הבאה:

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

import { CryptoService} from '../../services/crypto.service'

@Component({

selector: 'app-crypto-list',

templateUrl: './crypto-list.component.html',

styleUrls: ['./crypto-list.component.css']

})

export class CryptoListComponent implements OnInit {

constructor(private \_cryptoServ:CryptoService) {

}

ngOnInit() {

}

}

\*\* כפי שעשינו בסעיפים הקודמים ניתן לראות שייבאנו את ה SERVICE לתוך הקומפ ועכשיו נוכל לדבר איתה בכל המיטודות במחלקה עם המשתנה \_cryptoServ

עכשיו כדי לקבל מידע כל פעם שה OBSERVABLE בקומפ של קריפטו משגר מידע אנחנו נצטרך להשתמש בפקודה subscribe וכל פעם שיגיע מידע חדש מהפונקציה getTestData אנו נקבל תגובה בפונקציה עם הפרמטר שמכיל את המידע ששיגרנו (כגון bit value 1000 וכו'...) ובשביל זה בתוך ה NGONIT בקומפ TS נכתוב את הקוד הבא:

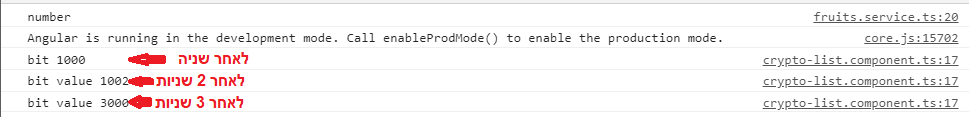
ngOnInit() {

this.\_cryptoServ.getTestData().subscribe( data => {

console.log(data);

})

}

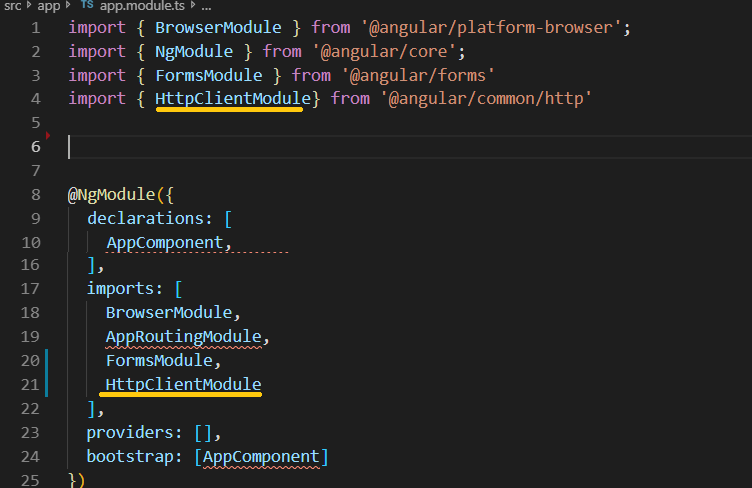
ההמחשה הכי טובה ל SUBSCRIBE תחשבו על ערוץ יוטיוב שאתם עושים לו SUBSCRIBE , הרי מה שמיוחד במה שעשיתם הוא שתקבלו כל פעם הודעה חדשה כאשר סרטון חדש עלה לערוץ.  
אז גם כאן ה SUBSCRIBE די דומה שאנחנו מקבלים מידע חדש כל פעם שהפונקציה שאנו פונים אליה מקבלת מידע חדש או/ו רוצה לעדכן אותנו במידע חדש)  
  
  
תפעיל את האפליקציה ואתם אמורים לראות ב CONSOLE במשך כל שניה שעוברת את ההודעות הבאות:  


עכשיו אחרי שהבנו איך OBSERVABLES עובדים הגיע הזמן לעבוד עם פקודה שמחזירה OBSERVABLE שכמעט תמיד תשתמשו בה בכל אפליקציה שתייצרו בשם HttpClient שמבצעת פעולות דומת ל AJAX ו REQUEST ל REST API

**22. עבודה מול REST API עם httpClient**

אז סוף סוף הגענו לחלק המעניין המאפשר לנו לתקשר עם שרתים באנגולר , בטח כפי שאתם מכירים ב JS רגיל ו JQUERY בדרך כלל באפליקציה נשלוף את המידע משרתים על ידי פקודות שונות של REST API כגון :  
AJAX, XMLREQUEST,FETCH,AXIOS ועוד...

שימו לב שהמודול שמתעסק בקריאות הללו הוא httpClient ולמזלנו בגרסאות האחרונות של אנגולר הוא בעצמו פקודה מסוג observable מה שחוסך לנו המון שורות קוד ויד בקלות נוכל לבצע קריאת REQUEST לשרת או קובץ מסוים

כדי שנוכל להשתמש ביכולות httpClient בדומה ל FORMMODULE שהגדרנו בסעיף 6 בשביל שנוכל להשתמש ביכול? ngmodel עלינו להכריז על היכולת הנ"ל בתוך קובץ app.module.ts 

הקוד של ה IMPORT ל HTTP מודול:

import {HttpClientModule} from '@angular/common/http'

דבר ראשון ביצענו import בשורה 4 ל httpClient והגדרנו אותו בתוך ה IMPORTS (שורה 17 והגדרנו אותו בשורה 21) כך שעכשיו יהיה לנו את האפשרות להגדיר את יכולות ה httpClient בכל SERVICE שנחפוץ להשתמש בו  
  
לכן הצעד הבא שנעשה הוא בSERVICE הקריפטו :

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

import {HttpClient ,HttpHeaders} from '@angular/common/http'

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class CryptoService {

bitUrl = "https://api.coinmarketcap.com/v1/ticker/?limit=10";

data = "";

constructor(private http:HttpClient) { }

}

א. זה לעשות IMPORT ל httpClient ו httpHeader (שבהמשך נבין מה המשמעות של הHEADER)  
ב. להגדיר את הURL של API שיחזיר לנו ערך מדוייק של עשרת מטבעות הקריפטו הנפוצים ביותר בעולם כמו ביטקויין ועוד... במשתנה BITURL  
ג. להגדיר משתנה PRIVATE שמשתמש ביכולות ה HTTPCLIENT על מנת שנוכל לבצע איתו קריאת REQUEST לכתובת ה API שלנו  
  
הצעד הבא שנעשה הוא להוסיף את הפונקציה הבאה למחלקה של ה SERVICE CRYPTO:

***makeApiRequest***(){

return this.http.get(this.bitUrl);

}

שימו לב שבסעיף הקודם למדנו על OBSERVABLE וזה בדיוק מה שמודול ה HTTP CLIENT מבצע גם , לאחר שהוא שולח במקרה שלנו בקשת GET הוא ישגר OBSERVABLE בחזרה מפנוקציית makeApiRequest שלאחר מכן יהיה ניתן לבצע לה subscribe מקומפנינטה שהסירבס מוטמע בו ,  
על מנת להאזין לשינוי או מידע שהתקבל במקרה שלנו מבקשת ה API  
ואת הSUBSCRIBE נבצע בתוך הקומפנינטה של הcrypto-list בפונקציית ngONINIT בצורה הבאה

ngOnInit() {

this.\_cryptoServ.***makeApiRequest()***.subscribe(data => {

console.log(data);

})

}

וכל מה שנשאר לנו על מנת להציג את המידע שקיבלנו מה API המסויים הנ"ל הוא:  
אחד להכריז ב קומפנינטה של ה crypto מערך:

coins\_ar = [];

וב פונקציה NGONINIT לכתוב את הקוד הבא:

ngOnInit() {

this.\_cryptoServ.makeApiRequest().subscribe(data => {

console.log(data);

this.coins\_ar = data;

})

}

וב VIEW של הקומפ CRYPTO לכתוב את הקוד הבא:

<div class="row text-left">

<div class="col-lg-6 border p-2" \*ngFor='let item of coins\_ar'>

<img

src="https://files.coinmarketcap.com/static/widget/coins\_legacy/64x64/{{item.id}}.png"

class="float-left mr-2"

>

<h2>{{item.name}}</h2>

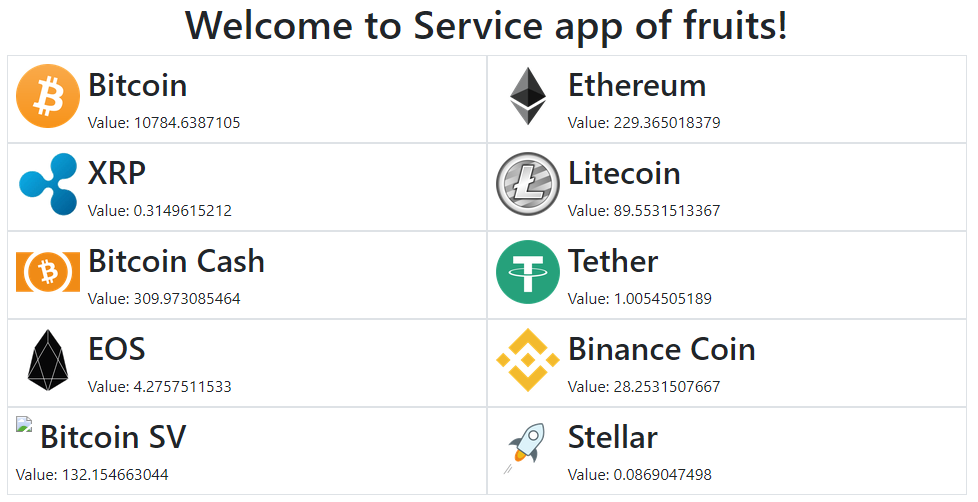
<div>Value: {{ item.price\_usd}}</div>

</div>

</div>

וזהו , כך אנו עושים בקלות בקשת API ומציגים את המידע.

להלן התוצאה:

  
אתגר התחברו ל API של מטבעות כסף אמיתיים ונסו להפוך את ערך מטבעות הקריפטו לערך של שקל לפי השער [האמיתי מה API (לחצו עם CTRL על הלינק הנוכחי):](http://apilayer.net/api/live?access_key=3c81786f9b3d2e267f40d08af97b97f2&currencies=usd%2Cils%2Ceur%2Cbtc%2Cthb)

**23. מה זה ROUTER ולמה צריך אותו?**

אחד מהיתרונות הגדולים של אנגולר הוא שהיא מתבססת על אפליקציית דף אחד , כלומר כאשר נעבור בלינקים לא יטען כל פעם כל הדף מחדש אלא רק החלק שנחפוץ בו.  
לכן על מנת שנוכל לעשות זאת עלינו להשתמש במודול ROUTER שמאפשר די בקלות לבצע את הפעולות הללו ,  
\*\* שימו לב שמגרסת אנגולר 7 + כאשר אנו מתקינים את האפליקציה שלנו נשאלת השאלה אם אנו נרצה להשתמש ב ROUTER ואם נכתוב שכן אוטומטית המודול יותקן בתוך האפליקציה.  
  
יתרון חשוב נוסף למה להשתמש בROUTER ולא סתם להחליף מידע במידע דרך INNERHTML הינו:  
1. סדר בקוד – כל דף נמצא בקומפנינטה שלו  
2. גוגל יאהב שלכל דף יש את הכתובת שלו  
3. תצוגת מידע מהירה יותר

אז קודם כל שימו לב שהותקן לכם הקובץ app-routing.module.ts אם הוא כבר קיים זה אומר שיש לכם את היכולת לעשות ROUTING וגם בקובץ ה app.module.ts אמור להיות לכם תזכור לזה עם IMPORT של appRoutingModule

עכשיו נצא מנקודת הנחה שיש לכם 2 קומפנינטות לפחות אחת מהן היא הקירפטו שיצרנו קודם והשני היא רשימת הקניות עם items-list שעכשיו נשתמש רק בשביל התצוגה שלהם  
  
אז קודם כל בחלק של ה app.component נכניס את התגית של המודול של ה app-routing עם התגית  
 <router-outlet> שמוגדר בתוך המודול עצמו של ה ROUTER כאשר כבר התקנו אותו (למי ששואל) ונמחק את כל הקומפנינטות הקודמות

<div style="text-align:center" class="container">

<h1> Welcome to Service app of fruits! </h1>

<router-outlet></router-outlet>

</div>

דבר שני שנעשה הוא להגדיר את הנתיבים שיש לנו באפליקציה שלנו , נניח שנרצה שבעמוד הבית תופיע רשימת מטבעות הקריפטו ובכתובת localhost:4200/list יופיע עמוד items-list עם כל מה שכתבנו בו:

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

import { Routes, RouterModule } from '@angular/router';

import { CryptoListComponent } from './comps/crypto-list/crypto-list.component';

import { *ItemsListComponent* } from './comps/items-list/items-list.component';

const **routes**: Routes = [

{path:"" , component: CryptoListComponent },

{path:"list" , component: *ItemsListComponent* }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

\*\* דבר ראשון נעשה IMPORT לכל לכל הקומפנינטות שנרצה להשתמש בהן ב ROUTE , והדבר הבא בתוך המשתנה rotues נגדיר כאובייקט את הנתיב של קומפנינטה, ואנו יכולים לראות שה PATH:"" מייצג את עמוד הבית נשתמש ברשימה הקירפטו crypto-list . וב path:"/list" אנו נציג את הקומפנינטה של ה items-list  
  
נסו להיכנס עכשיו לכתובת localhost:4200 ותראו את הקומפנינטה של רשימת הקריפטו מופיעה במיקום ששמנו ב app את התגית router-outler . ואם נכנס ל localhost:4200/list נקבל במקום את רשימת המוצרים , משקאות ועוד... שעשינו בדוגמאות הקודמות

**24. איך עושים לינקים באנגולר שישנו את ה ROUTER אך לא יבצעו refresh לדפדפן !**

שימו לב שהמטרה הבאה שלנו היא לבצע לינק מיוחד, שלא ישלח אותנו לכתובת מה שיגרום לדפדפן לרפרש את עצמו ויאיט לנו את הניווט , אלא מאחוריי הקלעים הוא יחליף אזור מסויים בדף (במקרה שלנו את התגית router-outlet ) ובנוסף יעדכן את כתובת URL של האתר ללא REFRESH ככה שגם מבחינת SEO (קידום אתרים) האפליקציה/אתר שלנו יהיו לפי התקן.  
אז קודם כל בקומפנינטת APP בVIEW נכניס את הקוד הבא:

<div style="text-align:center" class="container">

<a href="#">Crypto</a> |

<a href="#">Items</a>

<h1> Welcome to Service app of fruits! </h1>

<router-outlet></router-outlet>

</div>

על פניו נשמע שאם נרצה לייצר לינק ל/list נכתוב בתגית A

<a href="/list">Items</a>

\*\* אך שימו לב שפעולה זו תעשה בדיוק את מה שלא רצינו שהיא תעשה, היא פשוט תשגר אותנו לדף של LIST . הוא אומנם יוצג, אך כל פעם שנלחץ כל הדף יתרפרש (יטען מחדש)  
 אז כדי לפתור את הבעיה אנגולר נותנת לנו להשתמש באטרביוט שנקרא routerLink במקום ה HREF ויתרון נוסף שניתן לכתוב אותו בנוסף ישירות בתוך תגיות של תמונות וכפתורים שיצרנו בלי לעטוף אותם בתגית A  
והקוד של 2 הלינקים שלנו בעצם יצטרך להראות כך בקומפ APP :

<a routerLink="/">Crypto</a> |

<a routerLink="/list">Items</a>

וזהו בגדול. תוכלו עכשיו להכניס את הrouterLink בכל מקום מכיוון שהוא מחובר דרך ה App.module.ts

במידה ונרצה להוסיף ACTIVE LINK כלומר שאם אני נמצא בעמוד של הלינק כבר שהוא יציג אותו בהבלטה כל מה שנצטרך לעשות זה להוסיף את האטרביוט עם PROP BINDING בשם routerLinkActive

<a routerLink="/" [routerLinkActive]="['active']">Crypto</a> |

<a routerLink="/list" [routerLinkActive]="['active']">Items</a>

\*\* שימו לב שהACTIVE בתוך הסוגריים המרובעים מייצג קלאס שנצטרך לייצר בCSS  
  
עכשיו כשתפעילו תיתקלו בבעיה מוזרה, בעמוד הבית רק ה לינק של הcrypto אכן עובד אך כשניכנס ל /list נראה פתאום ששני הלינקים עובדים, זה קורה כי ברירת המחדל של אנגולר ה ROUTER מחפש צירוף ומכיוון שעמוד הבית עומד בסטרינג של הצירוף ב URL מבחינתו הוא צריך להיות מוצג, כדי למנוע את זה, מה שצריך להוסיף זה את האטרביוט הבא שאומר לACTIVELINK לעבוד רק כאשר הצירוף הוא מדוייק ב URL:ה

<a routerLink="/" [routerLinkActive]="['active']" [routerLinkActiveOptions]="{exact:true}"> Crypto </a>

**25. כיצד להעביר פרמטרים דרך ה ROUTER ?**

אז לאחר שהבנו איך הראוטר עובד , הגיע הזמן להבין איך אפשר לאסוף ממנו מידע, נניח נייצר קומפנינטה שיקראו לה FOODs שבתוכה יהיה מערך עם 4 מאכלים וה URL שיפעיל אותה בברירת מחדל יהיה   
localhost:4200/foods  
ואנו רוצים שאם נניח המשתמש יגיע לעמוד localhost:4200/foods/1 יוצג לו מהמערך המאכל השני (נזכיר שמערך מתחיל מ0) ואם במקרה הוא יגיע לעמוד localhost:4200/foods/2 יוצג לו הפרי השלישי ברשימה וכך הלאה,  
לכן קודם כל בואו נייצר לעצנו קומפנינטה בשם FOOD (מזמן לא עשינו את זה ☺ )  
ng g c comps/foods  
הצעד הבא שנבצע זה להוסיף לקומפנינטה מערך של מאכלים במחלקה ומשתנה של האינדקס של האוכל הנבחר:

export class FoodsComponent implements OnInit {

foods\_ar = ["cake","pizza","burger","salad","chocolate"];

choosenFood = 0; ...

ב APP-routing.module.ts נוסיף את השורות הבאות:

import { FoodsComponent } from './comps/foods/foods.component';

const routes: Routes = [

{path:"" , component: CryptoListComponent },

{path:"list" , component: ItemsListComponent },

{path:"foods" , component: FoodsComponent },

{path:"foods/**:id**" , component: FoodsComponent }

];

\*\* שימו לב לאובייקט האחרון שבו כתבנו במאפיין path:"foods/:id" ה ID במקרה שלנו מייצג את האינדקס של הפרי שנרצה להציג ב HTML כאשר קומפנינטה ה FOOD עובדת.  
  
בתוך הקומפ של ה FOODS נייבא את המודול הבא:

import {ActivatedRoute} from "@angular/router";

האובייקט ACTIVATEROUTE יאפשר לנו לקבל את פרמטר ה ID , אך לפני נצטרך להכריז על משתנה מהסוג שלו ב קונסטרקטור ככה שנוכל להשתמש בו באפליקציה שלנו (בדומה לSERVICE שהיינו מחברים) והקלאס של ה FOODS יראה כך:

export class FoodsComponent implements OnInit {

foods\_ar = ["cake","pizza","burger","salad","chocolate"];

choosenFood:any = 0;

constructor( private route:ActivatedRoute ) { }

ngOnInit() {

this.choosenFood = this.route.snapshot.paramMap.get(**'id'**);

}

}

ושימו לב שהגדרנו שכל פעם שהקומפנינטה מופיעה ב ngonInIt המשתנה choosenFood יאסוף את המידע מה ROUTER וכרגע ב HTML של ה FOODS נכתוב את הקוד הבא:

<h2> Your choosen food is: {{choosenFood}}</h2>

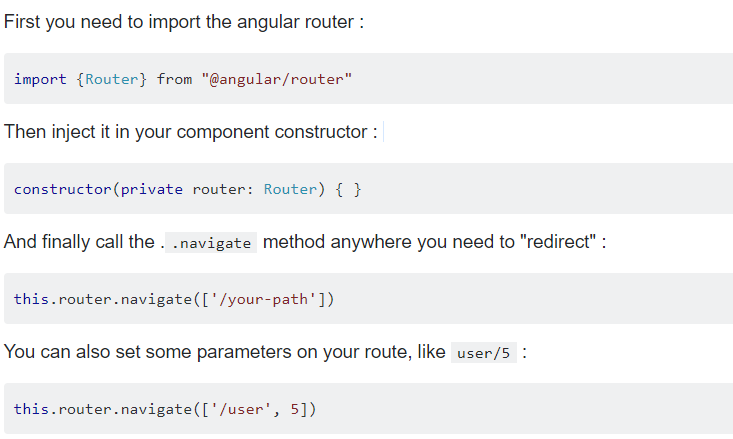
כרגע אם ניכנס נניח ל כתובת <http://localhost:4200/foods/2>  
יוצג לנו במשתנה הספרה 2 , אם נרצה שיוצג הפרי מהמערך כל מה שנצטרך לשנות בקומפ זה ב NGONINIT

ngOnInit() {

let getId = this.route.snapshot.paramMap.get('id');

this.choosenFood = this.foods\_ar[getId];

}

מודול נוסף שניתן להשתמש בו הינו LOCATION שמאפשר להשיג את הכתובת המדויקת שבה כרגע אנו נמצאים באפליקציה שבהמשך נבין את הצורך שבו.  


import {Location} from "@angular/common";

**26. כיצד לייצר "עמוד לא נמצא 404"**  
  
אז בטח כמו שהנכם כבר מכירים , כאשר המשתמש יגיע לכתובת שהוא לא אמור להגיע אליה או שכבר אינה קיימת (לא מומלץ למחוק ככה כתובת מבחינת SEO ) נרצה שהמשתמש יגיע לדף 404 כלשהו שאנחנו נעצב   
אז בואו נייצר קומפנינטה חדשה בשם page-404

Ng g c comps/page-error404

כמובן תוכלו לעצב את ה HTML וה CSS שלו כפי שתרצו , אך הדבר הנוסף שבסך הכל נצטרך להגדיר הוא שאם המשתמש נכנס לעמוד שלא קיים הוא בקובץ המודול app-routing.module.ts במערך ה ROUTESעם הערך "\*\*":

const routes: Routes = [

{path:"" , component: CryptoListComponent },

{path:"list" , component: ItemsListComponent },

{path:"foods" , component: FoodsComponent },

{path:"foods/:id" , component: FoodsComponent },

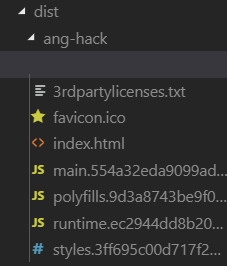
{path:"\*\*" , component: PageError404Component }

];

וזהו יש לנו עמוד 404 !

**27. איך לבצע DEPLOY לאפליקציה שלנו**

אז חכו אומנם יש עוד כמה נושאים בסיסיים שחשוב שנדע , אבל הגיע הזמן לעשות DEPLOY לאפליקציה שלנו ולעלות אותה לשרת !  
  
אז כל מה שאנחנו צריכים לכתוב בתיקיית הפרוייקט שלנו הוא :  
**ng build --prod**

ונוצרת לכם בתוך הפרוייקט תיקיית DIST עם תיקייה בשם הפרוייקט שתכנסו אליה (במקרה שלנו ang-hack) תראו את הקבצים הבאים:  
  
הקובץ מסוג JS שמתחיל ב MAIN מכיל בתוכו את כל הקוד של הג'אווה סקריפט שלנו וכך גם ה STYLE לגבי הCSS מה שהופך את טעינת האתר להרבה יותר מהיר!  
בנוסף על מנת שנוכל לעלות את הפרויקט נהיה חייבים לייצר בתיקייה שבDIST איפה שנמצאים כל הקבצים את הקובץ   
.htaccess  
מכיוון שאם משתמש יגיע לתיקייה מסוימת בפרויקט שלנו השרת ידע להציג לו אותה ולא יחפש את הקובץ או התיקייה,  
  
 לדוגמא אם המשתמש יכנס ל  
my-domain.co.il/list  
בברירת מחדל של כל השרתים ובמיוחד לינוקס השרת יחפש תיקייה בשם list שלא קיימת, לכן בתוך קובץ ה .htaccess נכנס את הקוד הבא:

RewriteEngine on

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule . /index.html [L]

וזהו עכשיו נעלה את הקבצים AS IS לשרת וזה פשוט יעבוד לנו !  
ברכות, הפרוייקט שלכם באוויר!

**28. ולידזאציה (בדיקת תקינות) לטפסים באנגולר**  
אנגולר מגיע עם יכולת חכמה לעשות ולדיזציה בצד קליינט לטפסים (שימו לב שעדיין תהיו חייבים לעשות ולדיזציה גם בצד שרת כדי לאבטח את עצמכם) ככה שהעניין יחסוך חלק מהבקשות של צד השרת שלכם (במקרה של טעות בקלט הלקוח כגון מספר טלפון שגוי , אימייל שגוי או כל טעות אחרת)

אז בואו נתחיל שוב מהתחלה את האפליקציה שלנו ונתקין בה ישירות קומפנינטה חדשה בשם FORM:

Ng g c comps/form  
שיראה כך:

<div class="row">

<div class="col-3"></div>

<form class="col-6 text-left">

<div class="form-group">

<label for="exampleInputEmail1">Full name:</label>

<input type="text" class="form-control" id="full\_name" name="full\_name" >

</div>

<div class="form-group">

<label for="exampleInputEmail1">Email address</label>

<input type="email" class="form-control" id="exampleInputEmail1" name="email" >

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

</div>

**28.א כיצד לדבר עם אלמנטים ולהוציא את הערכים שלהם בקלות עם @CONTENT CHILD**

אז לפני שנתחיל לשלוף מידע מטופס ולבדוק ולדיזציה עליו, יהיה חכם קודם כל להבין כיצד בצורה הבסיסית אנגולר מאפשר לדבר עם אלמנטים שונים ב DOM שבקומפנינטה שלי עם הדיקורטור שנקרא @viewChild  
אז לדוגמא הבאה נוסיף בתוך ה VIEW של קומפ ה FORM את התגית הבאה ועליה נוסיף אטרביוט מיוחד שמתחיל ב סולמית (ונקרא בשם המקצועי שלו template reference variable ):

<p #myPara>para test</p>

בנוסף בואו נאזין לשיגור הטופס על יד המשתמש עם EVENT חדש בשם ngSubmit ועל התגית של הFORM נכתוב את הקוד הבא:

<form class="col-6 text-left" (ngSubmit)="onSub()">

עכשיו כדי שנוכל לאסוף את המידע מה PARAGRAPH כל מה שאנחנו צריכים לעשות ב TS של הקומפ הוא:

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

@Component({

selector: 'app-form',

templateUrl: './form.component.html',

styleUrls: ['./form.component.css']

})

export class FormComponent implements OnInit {

@ViewChild('myPara') para;

ngOnInit() {

}

onSub(){

console.log(this.para)

}

}

ושימו לב שכל מה שאנחנו צריכים על מנת לאסוף את המידע הוא להכריז על משתנה עם הפקודה VIEWCHILD עם ה# שרשומה ב VIEW

כאשר נלחץ על הכפתור SUBMIT נראה שנקבל אובייקט בקונסול שמכיל המון מידע (לאלמנטים שנדבר עליהם בהמשך כגון תגית FORM תהיה משמעות ממש חשובה) שבמקרה שלנו בשביל לאסוף נניח את ה INNERHTML שלו נצטרך לרשום :

console.log(this.para.nativeElement.innerHTML)

עכשיו לאחר שהבנו איך אנו יכולים לאסוף מידע מתגיות ילד של הקומפנינטה נוכל להסיק שניתן להוסיף לכל תגיות האינפוט שלנו את # (VAR (TEMPLATE REFERNCE ומשם לאסוף את ה VALUE, אך ישנה שיטה טובה יותר באנגולר !

**28 ב. אנגולר form validation**

אנגולר מגיע עם דירקטיבים ועוד פקודות נחמדות שעוזרות לנו להציג שגיאות ולבדוק את שדות הקלט שהמשתמש הכניס, קודם כל כדי שנוכל להשתמש ביכולות הללו נוסיף על תגית ה FORM שלנו template referenc עם תוספת מיוחדת שמשווה אותו אותו לסוג ngForm כך שהקונסול ייתן לנו להיכנס אל כל המידע שיש בטופס, לכן בVIEW ב קומפ FORM נוסף את האטרביוט הבא:

<form class="col-6 text-left" (ngSubmit)="onSub()" #f="ngForm">

בנוסף נצטרך להוסיף את האטרביוט ngModel על כל האינטפוטים שנרצה לבדוק על מנת שאנגולר יבין שיש קשר אליהם לטופס (ככה TYPE מסוג ngForm עובד , שימו לב שאצלכם בקוד 2 האינפוטים ממוקמים רחוק אחד מהשני)

<input type="text" class="form-control"

id="full\_name" ngModel name="full\_name" > ...

<input type="email" class="form-control"

id="exampleInputEmail1" ngModel name="email" >

ובTS של הקומפ נכריז על המשתנה כ VIEWCHILD ונכניס אותו למשתנה בשם myForm :

@ViewChild('f') myForm;

ובפונקציה של השיגור נעשה console למשתנה שלנו:

onSub(){

console.log(this.myForm);

}

מלאו את שדות הקלט כאשר תפתחו הפעם את ה CONSOLE תראו רשימה הכוללת (אם תחקרו לעומק) את כל ***הVALUE*** של שדות הקלט בטופס שלנו (שנמצאים במאפיין ***VALUE***) ואפילו תראו מאפיין מעניין ביותר בשם ***VALID*** שכבר בודק אם הטופס שלנו ***VALID*** (ובמקרה שלנו הוא יהיה כרגע תמיד TRUE מכיוון שלא הכנסנו דירקטיבים שאנגולר מאפשר לנו להשתמש בהם ב VIEW)  
  
 אז בואו נתחיל עם משהו בסיסי ונוסיף אטרביוט שאנגולר הפכה אותו ליכולת מיוחדת בשם required בשני האינפוטים שלנו (שבVIEW שלכם נמצאים בשורות רחוקות ממה שמוצג בדוגמא מטה נוסיף את הREQUIRED)

<input type="text" class="form-control"

id="full\_name" ngModel name="full\_name" required > ...

<input type="email" class="form-control"

id="exampleInputEmail1" ngModel name="email" required >

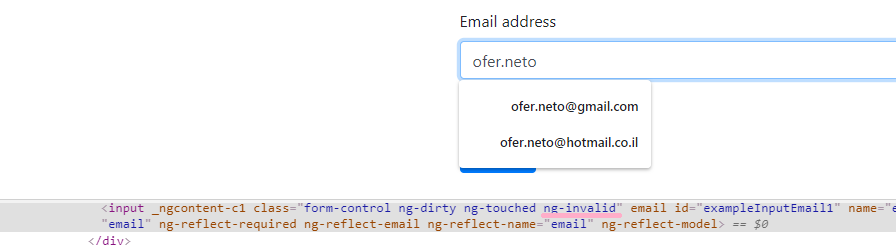
אם תנסו לשגר הפעם את הטופס בלי שמילאתם את הטופס תשימו לב שהפעם המשתנה VALID מציג את עצמו כ FALSE אך אם תנסו למלא אפילו ג'יבריש בשדה הקלט של האימייל ללא שטרודל הVALIDATOR לא יודע לזהות כרגע שמדובר בשדה קלט שצריך לקבל EMAIL . כדי שהוא ידע נצטרך להוסיף לשדה הקלט את האטרביוט הבא:

<input type="email" class="form-control"

id="exampleInputEmail1" ngModel name="email" required email>

עכשיו נוכל כבר להשתמש ביכולות המיוחדות הללו של אנגולר על ידי הוספת PROP BIND בשם DISABLED על כפתור ה SUBMIT שמשווה אותו לf.valild עם ! כדי לבדוק אם הוא TRUE או FALE ולתת את התגובה ההפוכה:

<button [disabled]="**!f.valid**" #btn2 class="btn btn-primary">Submit</button>

לפני שנמשיך נרצה להראות לכם עוד משהו מעניין אם תלחצו בדפדפן כרום עם כפתור ימני על האלמנט INPUT ותבחרו בתפריט ב INSPECT ELEMENT שתתחילו למלא את שדה הקלט תראו משהו מעניין ביותר:  
  
שימו לב שנוצר על הINPUT שלנו קלאסיים ולמשל באימייל נוסף קלאס ng-invalid כל עוד השדה קלט לא מכיל EMAIL תקני, וכאשר הכל כן תקני תראו שהקלאס מתחלף לקלאס: ng-validוכל זאת תודות לדירקטיב ngModel

בנוסף ניתן לראות גם את הקלאס ng-touched שמרמז שהמשתמש כבר רשם בתוך האינפוט ובהמשך יעזור לנו לבצע פקודות נוספות כגון לסמן באדום אינפוטים עם שגיאה וכדי לעשות זאת ב CSS של הקומפ FORM נוסיף את הסלקטור:

input.ng-invalid.ng-touched{

border:1px solid red;

}

וזהו הסלקטורים שלנו יהפכו להיות עם BORDER אדום כאשר המשתמש יעשה טעות,

**28 ג. כיצד להוסיף הודעה לשגיאה של המשתמש?**

בנוסף חוץ מסימון יהיה נחמד אם המשתמשים שלנו יקבלו איזושהי הודעה מה בדיוק השגיאה שהם ביצעו  
ובשביל זה נוסיף בVIEW את הקוד הבא:

<input type="email" ngModel class="form-control"

id="exampleInputEmail1" name="email" #emailInp="ngModel" required email >

<span class="help-block" \*ngIf="!emailInp.valid && emailInp.touched">\*Please enter valid email</span>

\*\* שימו לב ב INPUT של האימייל הוספנו TEMPLATE REF (#) בשם emailInp ונתנו לו את ה TYPE בשם ngModel   
ובהודעה שמופיעה ב SPAN בדקנו אם המאפיין שלו VALID ובנוסף אם המשתמש כבר שינה אותו (touched)

28.ד. איך לגרום ב SELECT BOX שיהיה אופציה כברירת מחדל  
בואו נוסיף לטופס שלנו את הקוד הבא בזריזות של בחירת עוגה:

<div class="form-group">

<label for="exampleInputEmail1">Choose your cake</label>

<select id="select" class="form-control" name="select\_cake" ngModel>

<option value="orange">Orange cake</option>

<option value="Vanilla">vanilla cake</option>

</select>

</div>

כל מה שעשלינו לעשות הוא להוסיף לתגית SELECT את NGMODEL בתור PROP BIND

<select id="select" class="form-control" name="select\_cake" [ngModel]="cake">

ובתוך הקלאס של ה TS של הקומפ של FORM נכריז על המשתנה:

cake = "Vanilla"

ושימו לב שה OPTION של ה VANILLA יהיה זה שיופיע ראשון !  
  
אתגר נסו לבקש מהמשתמש להכניס PASSWORD ולהכניס PASSWORD שוב והציגו הודעת שגיאה במידה ו2 הסיסמאות לא זהות . במידה והסיסמא זהה לאחר שהמשתמש ילחץ LOGIN (נסו לבדוק כיצד ידנית דרך ה JS תוכלו לשגר אותו באנגולר לכתובת אחרת : חפשו בגוגל **router.navigate**)

28. ד כיצד לייצר PATTERN עם REG EXP משלנו (למשל שניתן להכניס רק אותיות באנגלית)

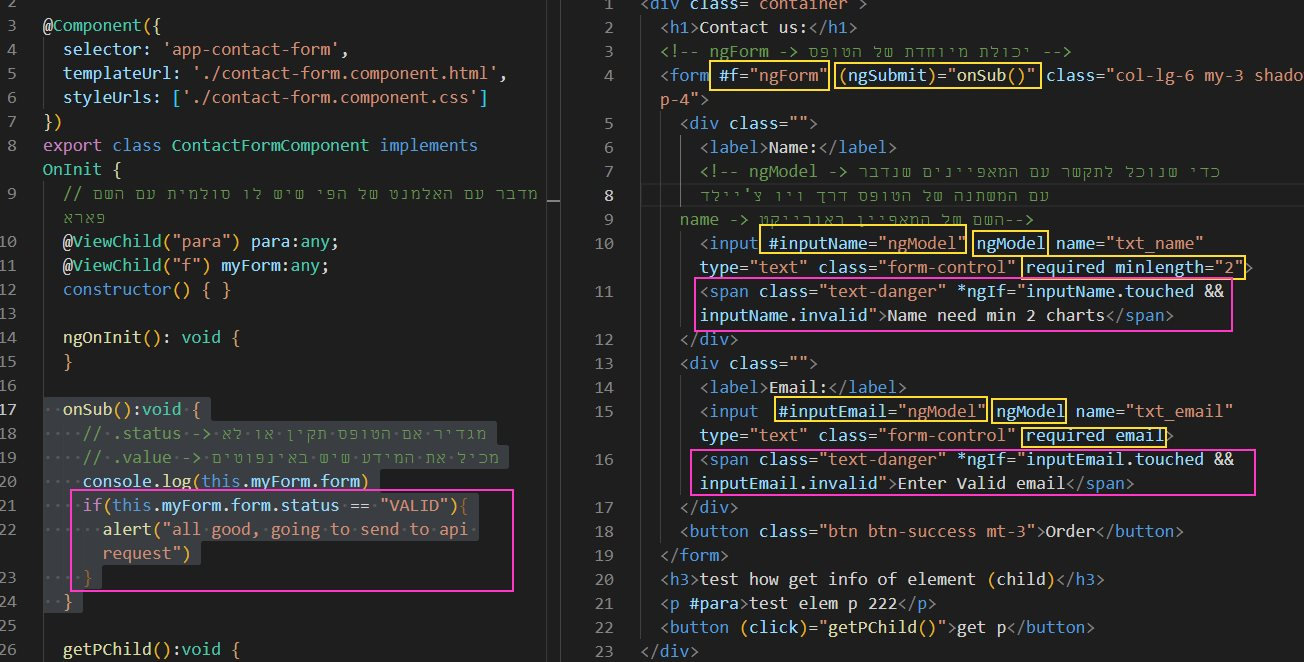
כל מה שנצטרך לעשות הוא בקבוץ ה TS להוסיף את המשתנה עם ה REGEXP למשל (נותן להכניס רק אותיות באנגלית):

***nameOnly***:any = /^[a-zA-Z!@#%&\*]{3,9999}$/

ובטופס עצמו ב INPUT שנרצה שה PATTERN יחול עליו (ואם לא הוא יציג שגיאה) נשתמש ב PROPERTY BINDING בשם PATTERN עם שם המשתנה:

<input [**pattern**]="***nameOnly***" ngModel type="text" #userInp="ngModel" required class="form-control" id="full\_name" name="full\_name">

(צילום סיכום לכל הפקודות שיש להוסיף ב HTML ו ב TS לשימוש בטופס:)



**29.** **התממשקות ל FIREBASE !**נתחיל שוב פרוייקט חדש ונתקין קומפנינטה שנקראת:  
ng g c comps/fireb

ובנוסף בחלק זה אנו יוצאים מנקודת הנחה שפתחת כבר DB בפיירבייס (עם טכנולוגיית FIRESTORE) ובתוכו ישנו קולקשיין שהדוקומנטים שלו נראים בצורה הבאה ואנו ממליצים לקרוא ל COLLECTION בשם "USERS" על מנת להבין את הפרק הבא:

1. 0:
   1. balance: 100
   2. email: "mindalizeev@gmail.com"
   3. name: "zeev"
   4. phone: "99-9999-99"
2. 1:
   1. balance: "53"
   2. email: "momo@out.com"
   3. name: "momo"
   4. phone: "222-21121-1"

עכשיו לפני שנתחיל נצטרך להתקין 2 מודולים שאנגולר דורש בשביל להתחבר ל FIRE BASE ולכן נתקין:

npm i firebase @angular/fire

לאחר ההתקנה נחבר אותם קודם כל ל app.module.ts :

import {environment} from '../environments/environment';

import {AngularFireModule} from '@angular/fire';

import {AngularFirestoreModule} from '@angular/fire/firestore';

וב IMPORTS :

imports: [

BrowserModule,

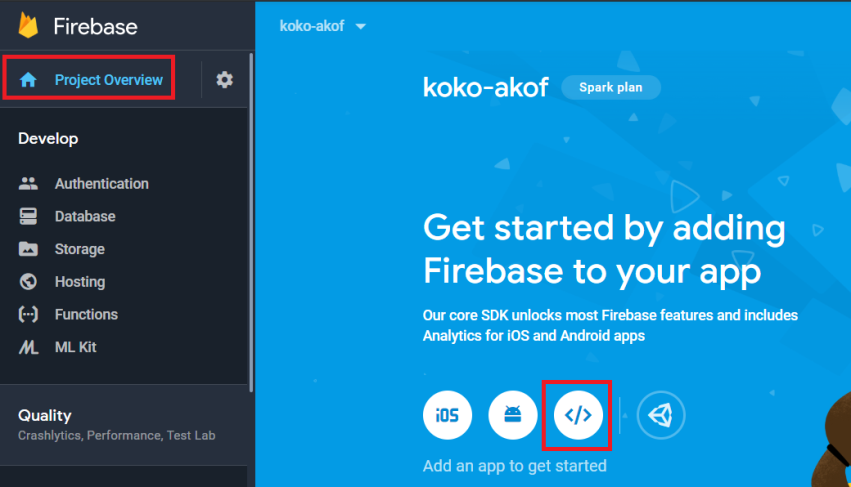
AppRoutingModule,

AngularFireModule.initializeApp(environment.firebase,"clientpanel"),

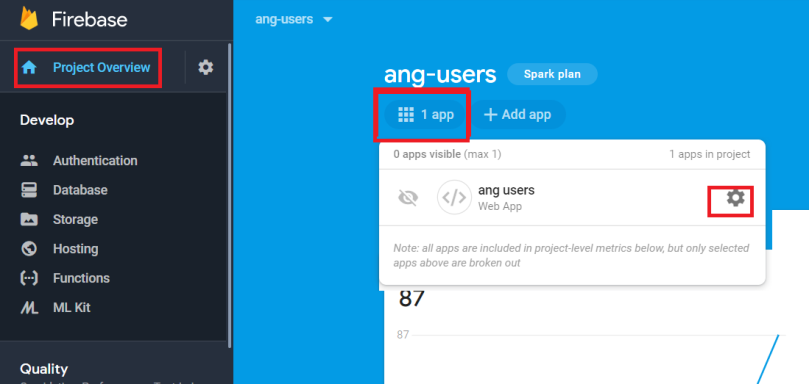
AngularFirestoreModule,

FormsModule

]

אך יש צעד נוסף שנצטרך לעשות והוא להגדיר את בקובץ במיקום environment/environment.ts את ההתחברות ל DB (כפי שרואים בקוד למעלה אנו משתמשים בקובץ זה והוא יכיל את מאפייני ההתחברות שלנו .  
את הפרטים של ה DB תוכלו לקחת מהפאנל של FIRE BASE עצמו : לחצו משמאל לימין על הריבועים האדומים  


ולאחר שתתנו שם לאפליקציה שלכם ב ADD FIREBASE SDK יופיע לכם הקוד הבא:  


במידה וכבר יצרתם אפליקציה יש ללחוץ על PROJECT OVERVIEW   
  
  
ולאחר שתלחצו על גלגל השיניים גללו למטה ותראו את הקוד. צריך להדביק את המאפיינים של הקוד הנ"ל בצורה הבאה בקובץ ה environment/environment.ts( וגם בקובץ environment.prod.ts שיהיה זהה רק שהמאפיין PRODUCTION אצלו יהיה ב TRUE ) שהוזכר קודם שיראה כך:

export const environment = {

production: false,

firebase: {

apiKey: "AIzaSyBXXXw9atDLt6rF1Z7aeR0h3Mv-XXXX",

authDomain: "XXX.firebaseapp.com",

databaseURL: "https://XXX.firebaseio.com",

projectId: "XXX",

storageBucket: "XXX.appspot.com",

messagingSenderId: "52XXX126",

appId: "1:52924XXXX26:web:22b99966XXXc8ac"

}

};

\*\*\* כמובן ערכי המאפיינים כגון APIKEY צריכים להיות לפי הקוד שקיבלתם מהפאנל של ה FIREBASE

הצעד הבא שנבצע הוא לעשות IMPORT בקומפ ה FIREB של מודול ה angularFirestore

import { AngularFireDatabase } from '@angular/fire/database';

כמו כן בקונסטרקטור של המחלקה של הקומפ נזריק כמשתנה PRIVATE משתנה בשם afs (angular fire store )שישתמש ביכולות המודול של הFIREBASE וגם נכריז על משתנה במחלקה מסוג מערך ((clients\_ar שיכיל את המידע שנקבל מה DB כ JSON במערך:

clients\_ar = [];

constructor(private afs:AngularFireDatabase){}

**30. כיצד לבדוק שאכן המשתמש מחובר למערכת ה FIRESTORE**   
צרו SERVICE חדש בשם   
ng g s services/auth  
  
ושימו בו את הקוד הבא:

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

import \* as firebase from 'firebase/app';

import { AngularFireAuth} from '@angular/fire/auth'

// https://www.techiediaries.com/angular-firebase/angular-9-firebase-authentication-email-google-and-password/

@Injectable({

providedIn: 'root'

})

export class AuthService {

user:any = "";

constructor(private af:AngularFireAuth) {

this.checkIfUserLogIng();

}

async trySignUp(value:any){

console.log(value);

let result = await this.af.createUserWithEmailAndPassword(value.email,value.password)

return result;

}

async tryLogin(value:any){

// this.af.auth.signOut();

console.log(value)

let result = await this.af.signInWithEmailAndPassword(value.email,value.password)

return result

}

async logOut(){

await this.af.signOut();

}

checkIfUserLogIng(){

this.af.authState.subscribe(user => {

if (user){

this.user = user;

console.log(user)

localStorage.setItem('user', JSON.stringify(this.user));

} else {

localStorage.removeItem('user')

}

})

}

}

בנוסף שימו לב שיש לייבא ב APP.module את השורות הבאות:

import { AngularFireAuthModule } from '@angular/fire/auth';

וכמובן יש להוסיף אותו למערך ה IMPORTS בקובץ APP.module  
  
כל מה שנשאר לנו הוא בדף ה APP לשים את הקוד הבא בשביל להתחיל לעבוד עמו:  
המשתנה showLogin בדוגמא מטה מייצג האם להציג את עמוד ה LOGIN/SIGNUP

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

import {AuthService} from "./services/auth.service"

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

  title = 'ang-hack-crm';

  showLogIn = false;

  constructor(private as:AuthService){

  }

  ngOnInit(){

    this.as.checkIfUserLogIng().subscribe(res => {

      if(res != null){

        console.log(res.email)

        if(res.uid){

          this.showLogIn = false;

        }

      }

      else{

        this.showLogIn = true;

      }

      console.log("aaaa",res)

    })

  }

}

**31. איך לאסוף מידע מ FIREBASE REALTIME**

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

import { AngularFireDatabase } from '@angular/fire/database';

@Component({

selector: 'app-fire-test',

templateUrl: './fire-test.component.html',

styleUrls: ['./fire-test.component.css']

})

export class FireTestComponent implements OnInit {

clients\_ar:any = [];

constructor(private afs:AngularFireDatabase) { }

ngOnInit(): void {

this.getRealFoods()

}

getObserfoods():any {

return this.afs.list("firstdb/food/foods1").snapshotChanges();

}

getRealfoods = () => {

this.getObserfoods().subscribe((res:any) => {

console.log(res);

this.foods\_ar =[];

res.map((item:any) => {

let newItem = item.payload.val()

// we add the key for able to del and edit the item after

newItem.**key** = item.payload.key

this.foods\_ar.push(newItem)

// console.log(item.payload)

})

console.log(this.foods\_ar);

})

}

**32.א איך להוסיף מידע מ FIREBASE REALTIME**

דוגמא לפונקציה שמוסיפה מוצר סטטי של בורגר לקולקשן של FOODS:

addFood = () => {

let newFood = {

name:"burger",

img:"bur.jpg",

cal:800,

price:50

}

this.afs.list("firstdb/food/foods1").push(newFood);

}

**32.ב איך למחוק מידע מ FIREBASE REALTIME**

כדי למחוק נשתמש בפונקציה הפשוטה הבאה, שימו לב ש keyItem שהוא הקיי של ה ITEM שאספנו כשיצרנו את הרשימה של אותה רשומה שנרצה למחוק

delFood = () => {

this.afs.list("firstdb/food/foods1/**keyItem**").remove();

}

**32.ג איך לערוך רשומה ב FIREBASE REALTIME**

שימו לב שהפרמטר KEY הוא ה KEY של ה ITEM שנרצה לערוך , בדוגמא בשונה מהמחיקה כבר הכנסתי פרמטר כדי שיהיה חכם כמובן ניתן לבצע את אותה סגנון פונקציה ל DEL

editFood = (**\_key**:any) => {

this.afs.object("firstdb/food/foods1/"+**\_key**).update({

name:"banana",

img:"bur.jpg",

cal:100,

price:11

})

}