ANGULAR 8
בעשרה ימים

מדריך ידידותי וחוויתי לצלילה אל תוך האנגולר צעד אחר צעד

כולל דוגמאות, תרגילים ופתרונות

סימי לייכטר

עימוד ועריכה: חני פולק

simi0527@gmail.com

מבוא

ספר זה נכתב לאור ביקוש ללימוד אנגולר מקיף ומעמיק בשפה העברית.

הספר נכתב במקביל לקורסים פרונטאליים שמסרתי והותאם להבנת התלמידים.

בספרי הקפתי נושאים מרכזיים באנגולר, הגדרות ודוגמאות שימושיות על מנת להנגיש את החומר באופן ברור.

דגש רב הושם על הפשטת החומר ועל שפה ידידותית, קלה ונוחה להבנה.

ההגדרות שנכתבו נועדו לתת מענה להבנה טובה של המושגים לפי שימוש נפוץ ולא תמיד היקפתי את כל ההיבטים של ההגדרה. כתבתי כך על מנת שההגדרות והמושגים יהיו מובנים מאוד ללומד.

לימוד מהנה,

סימי לייכטר

תוכן העניינים

יתרונות האנגולר

מהו היתרון באנגולר?

אחד היתרונות המהותיים של אנגולר, היא האפשרות של Single Page Application.

אנגולר מאפשר ליצור דף html אחד בודד שעולה פעם אחת למסך של המשתמש בעליית המערכת, ומעתה ואילך קיימת רק תחלופה של קטעי html (ולא דפי html).

היתרון הוא שכאשר משנים ברמת ה-js את התצוגה בפועל של ה-html אל המשתמש זה נעשה באופן מאוד מהיר, בהבזק של זמן שהמשתמש אינו מרגיש כלל, בדומה להחלפת מסכים במובייל (פלאפון וכד').

כאשר מחליפים דפי html למשתמש, יש צורך לפנות לסרבר, לשלוף משם את הקוד למסך הרצוי, ולשלוח חזרה את הקוד אל המשתמש. פניות אלה גורמות לאיטיות כלשהי.

מהו ההבדל בין angular.js לבין angular 2/4+?

Angular.js זוהי גירסה שנקראת גם angular.js 1 ואילו גירסאות הבאות נקראות כבר angular 2/4+ ללא הסיומת js. בין גירסה 1 לבין 2 ישנו הבדל מהותי, הגירסה שוכתבה לגמרי מחדש.

גירסה 3 לא יצאה לאוויר, ואח"כ בין גירסה 2 ליתר הגירסאות אין הבדל מהותי מבחינת המתכנת. גירסה 8 היא הגירסה החדישה ביותר נכון ליום כתיבת מסמך זה, והיא תומכת עדיין תמיכה מלאה בגירסה 2, ולכן ניתן להשתדרג לגירסה הכי מתקדמת ולהנות מהאפשרויות המתקדמות.

הכוונה במושג "גירסה תומכת" היא שכאשר ישנה מערכת שכתובה בגירסה 2, ומשתדרגים לגירסה 8 המערכת תמשיך לעבוד ללא תקלות גם ללא כל שינוי מצד הקוד. עדיין יהיה אפשר להמשיך לפתח רק באנגולר 2 למרות שגירסה 8 מותקנת.

מומלץ לעבור לגירסאות מתקדמות וכן לבצע התאמה לגירסה החדישה ולשנות את הקוד לפי הגירסה החדישה, למרות שקיימת תמיכה, מכיון שלא מובטח שתמשיך להיות תמיכה מתמדת של גירסה חדישה בכל הגירסאות הישנות.

על typescript יורחב בפרק בנפרד.

לשם הבנה כללית ה-typescript הוא מעין javascript עם אפשרויות נוספות. כביכול javascript גירסה מתקדמת.

ישנה בעיה, שהדפדפן שאיתו מעלים את האתר עדיין אינו מזהה את ה-typescript ולכן יש צורך לקמפל את typescript ל-javascript.

חלק מהקמת סביבת האנגולר היא הגדרת הטרסטפיילר (דומה לקומפיילר) שיקמפל את הקוד של ts ל-js כך שהדפדפן יוכל לזהות את הקוד.

סביבת עבודה וקבצים בסיסיים במערכת

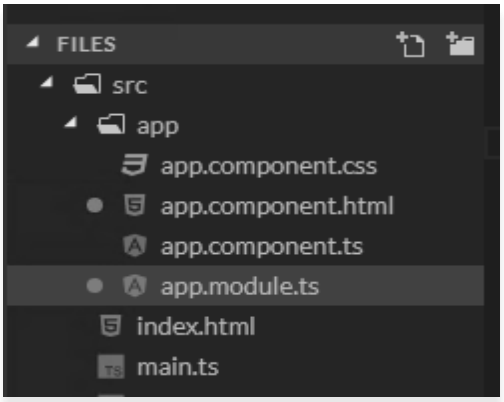
ישנן סביבות עבודה שונות לעבודה עם אנגולר, במסגרת מדריך זה, אנו נעסוק בסביבת העבודה של **stackblitz.com**. זהו אתר המאפשר פיתוח אנגולר אונליין, שמירת הפרוייקטים, שיתוף והורדה.

1. יש להיכנס לאתר https://stackblitz.com
2. לבחור באפשרות יצירת פרוייקט Angular. מבחירה זו יווצר פרוייקט חדש, מוכן לעבודה מיידית באנגולר ללא כל הורדה נוספת.
3. בראש העמוד מימין ישנה אפשרות כניסה לרשומים. מומלץ להיכנס עם שם משתמש, שם ישמרו כל הפרוייקטים שלך בעתיד.
4. בראש העמוד משמאל ישנה אפשרות של שמירה. מומלץ לקרוא שם משמעותי לפרוייקט.
5. יווצר שורת url בראש המסך, והוא הכתובת אליו ניגשים בפעם הבאה לפרוייקט. (למשתמשים רשומים, כל הלינקים שמורים ברשימה מסודרת).
6. Fork – יצירת העתק של הפרוייקט. ניתן לראות שלאחר לחיצה על כפתור זה ישתנה קישור הפרוייקט בשורת כתובת ה- url וההעתק מקבל כתובת url חדשה.
7. Share – ניתן לשתף את הפרוייקט המוגמר, ואף לאפשר את עריכת הקוד למשתמש אחר.
8. לשים לב! אם יוצאים מהמערכת ואח"כ נכנסים שוב, אם זה לא מאותו מחשב/משתמש (תלוי בהגדרות), הפרוייקט יהיה נעול לשינוי. ולכן האפשרות היא לבצע fork על הפרויקט ולעבוד על ההיעתק, או לחילופין לבצע share כבר בהתחלה, כך שבפעם הבאה שנכנסים הקוד לא יהיה נעול. האפשרות המומלצת ביותר היא לשמור תחת משתמש רשום.

סביבת עבודה זו ידידותית, מומלצת וחינמית.

מקובל לחלק את הפרויקט ולקרוא שמות לקבצים בצורה הבאה.

מומלץ לעקוב צמוד לפרויקט פתוח מאותחל.



קומפוננט

מהו קומפוננט?

קומפוננט – על פי רוב, משמש כתגית html שמקושרת לאוביקט נתונים. לכל קומפוננט יש את התבנית של ה-html בה יופיע כאשר יקראו לתגית זו, וכן יש את הקלאס הצמוד שלו. אין קומפוננט ללא קלאס! כל קומפוננט, מכיל אובייקט נתונים.

כיצד יוצרים קומפוננט?

להלן מספר שלבים למתכנת המתחיל.

ישנן סביבות עבודה המאפשרות קיצורי דרך. ב-stackblitz למשל, לחצן ימני על התיקייה הרצויה -> Generate Component פותח שדה טקסט יש להזין את שם הקומפוננט ואח"כ לחיצה על enter תיצור את הקומפוננט עם כל הרכיבים.

שלב ראשון

יש להוסיף תיקייה בשם הקומפוננט שתוגדר תחת תיקיית src.

יש להוסיף 3 קבצים מהסוגים הבאים: html, ts, css

לכל אחד מהם יש לתת את אותו השם, רק עם סיומת אחרת.

לדוגמא:

קומפוננט בשם server:

- שם התיקייה server
- שם קובץ html הוא server.component.html
- שם קובץ ts הוא server.component.ts
- שם קובץ css הוא server.component.css

שלב שני

יש למלא תוכן התחלתי בקובץ ts.

לדוגמא:

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-server',
  templateUrl: './server.component.html',
  styleUrls: [ './server.component.css' ]
})
export class ServerComponent  {
}
```

הסבר:

@Component – הגדרה שמדובר ביצירת קומפוננט.

selector – מייצג הקומפוננט, במידה וזו תגית, זה יהיה שם התגית.

templateUrl – הנתיב בו נשמר ה-html, התבנית של הקומפוננט.

styleUrls – הנתיבים בהם נשמרים קבצי ה-css המתאימים לקומפוננט זה. מערך נתיבים לקבצי css.

server.component.html

לדוגמא:

```
<div class="title">This is the Server Component</div>
```

server.component.css

לדוגמא:

```
.title{
  Color: yellow;
  font-size: 30px;
 }
```

שלב שלישי

יש להוסיף בקובץ moulde.ts שנמצא תחת תיקיית app במערך של declarations את שם הקלאס של הקומפוננט החדש שנוצר שיודגש כאן בשביל הדוגמא:

```
@NgModule({
  imports:      [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, ServerComponent],
  bootstrap:    [ AppComponent ]
})
```

יש להוסיף בראש הקובץ module.ts נתיב של מיקום הקובץ. יש לציין, שבהוספת נתיב לקובץ ts בקובץ זה, אין צורך לרשום שהקובץ הוא סיומת של ts, משמיטים את הסיומת.

לדוגמא:

```
import { ServerComponent } from './server/server.component';
```

שלב רביעי

כעת ניתן להוסיף שימוש בפועל של הקומפוננט.

לדוגמא:

בקובץ app.component.html יהיה הקוד הבא:

```
<p>
This is the app
</p>
<my-server></my-server>
```

כעת ניתן לראות פלט של הקומפוננט החדש שנוצר!

חשוב!

- ההגדרה export לקלאס נועדה לתת "הרשאה" לקבצים אחרים. (לא יורחב כעת.)
- ברגע שיש קריאה לתגית זו, נוצר אובייקט מסוג הקלאס של הקומפוננט.
- היות שיש כאן שימוש ב @Component שזה פיצ'ר של האנגולר, יש להוסיף בראש הקובץ ts את שורת הפקודה הבאה:

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

הרחבה

שימוש בפיצ'ר המתחיל עם @ של האנגולר, עלול לדרוש import מתאים.

כאשר תיכתב התגית <my-app></my-app> יעלה התוכן של הקובץ app.component.html.

בשלב זה, לא ניתן להכניס עוד נתונים בין תגית הפתיחה לסגירה. במידה ואכן יש צורך בכך, תורחב אפשרות זו בהמשך החומר.

לפי מה נקבעת שם התגית? לפי ה-selector.

selector - הערך השימושי ביותר הוא שם התגית. היות שב html אין הבדל בין אותיות גדולות לקטנות, לכן לא תהיה משמעות לשם "myApp" בצורה זו, זה ייקרא בדיוק כמו "myapp" ולכן על מנת להפריד בין מילים יש להשתמש עם קו מפריד "my-app".

ישנו ענין קצת עמוק יותר ב-selector, לאחר תירגול ועבודה קל יותר להבין. ובקצרה, ה-selector יכול להיות גם שם של קלאס ויראה כך 'selector:'.my-app עם נקודת קידומת כמו selector ב css עבור קלאס של css.

על מנת להשתמש בתגית זו, אפשר להשתמש כך:

```
<div class="my-app"></div>
```

ניתן להגדיר סלקטורים שונים כמו ב css.

לסיכום:

קומפוננט מורכב משלושה חלקים:

1. קובץ html שזו התבנית של הקומפוננט.
2. קובץ ts המכיל קלאס עם שדות נתונים, שם התגית, והגדרת הקשר ל-html ולקבצי ה-css.
3. קובץ css מותאם לקומפוננט זה.

styles

ניתן להגדיר לקומפוננט במקום styleUrls הגדרה של styles ולפרט בו במקום קלאסים או הגדרות שונות של css. לא ניתן להגדיר עבור אותו קומפוננט גם styleUrls וגם styles.

template

ניתן לתת את ה-template (מבנה ה-html) בקובץ עצמו, ולא נתיב לקובץ התבנית. במקרה זה לא ניתן להגדיר עבור אותו קומפוננט גם template וגם templateUrl.

לדוגמא:

```
@Component({
  selector: 'my-app',
  template: `<h1>Hello!</h1>`,
  styles: [`h1 { font-family: Lato; }`]
})
```

הערה חשובה:

ישנו סוג של גרש שקוראים לו "גרש הפוך". הוא לא הגרש הרגיל. ניתן לראות שהגרש בהגדרת ה-selector הוא גרש רגיל, ואילו הגרש שנמצא בהגדרת ה-template הוא גרש מוטה יותר. גרש אחר.

כאשר כותבים קוד בקובץ js לא ניתן לעבור שורה ללא סיום של פקודה או התחלה של סוגריים וכד'. לא ניתן לשבור string באמצע ללא תו מקשר. במקרה זה, יש צורך לשבור את ה-string לשורות נפרדות לפי התגיות (בשביל הנוחות). כל תגית תתחיל בשורה חדשה. לשם כך, ניתן להשתמש בגרש ההפוך ולשבור שורות בצורה חופשית בדיוק כמו בקובץ html ללא צורך בתו מקשר.

בקובץ מסוג html, אין בעיה שכל תגית תהיה בשורה נפרדת, כי כן ניתן לכתוב כל תגית בשורה נפרדת או אפילו באמצע תגית "לשבור" שורה. אך כאן מדובר בקובץ ts!

כנ"ל בהגדרת קלאסים של css, בהגדרה של styles. יש צורך להפריד כל הגדרת style ולהתחיל כל הגדרה בשורה נפרדת (לנוחיות). ולכן, על מנת לאפשר "שבירת שורות" ללא סימון נוסף מקשר בין שורה לשורה, יש לעטוף את כל ה-string בגרש הפוך.

Data-binding

מהי המשמעות של מושג זה?

הקשר בין ה-html לבין הנתונים והלוגיקה.

קישור נתונים מתוך קבצי ts לתוך ה-html.

לכל קטע של html ישנו קלאס בקובץ ts המתאים לו. כל קטע html יכול להשתמש בשדות שמוגדרות בקלאס שלו, בקומפוננט שלו, ורק בהם. בהמשך, יפורטו עוד אפשרויות.

ישנם 4 סוגים עיקריים של קשר בין הלוגיקה/המידע לבין ה-html:

1. String binding
2. Property binding
3. Event binding
4. 2-ways-binding

1. String binding

קישור נתונים של string . הקישור יתבצע ע"י שימוש ב {{ }}.

לדוגמא:

```
export class AppComponent  {
  name = 'Angular 8';
}
```

app.component.html

```
<div>{{name}}</div>
```

התוכן המשתנה name יופיע בתוך ה div.

חשוב לציין,

לא רק ערכי string ממש יכולים להופיע בתוך הסוגריים הכפולים, אלא כל ערך שיכול ע"י המרה לקבל ערך של string. ניתן להכניס ערך מספרי, ואפילו פונקציה שמחזירה ערך, כיון שהערך המוחזר מקבל המרה ל-string. זה גם יכול להיות ערך בוליאני, וההמרה האוטומטית תהיה "true/false" לפי הערך.

לדוגמא:

```
export class AppComponent  {
  isRight:boolean=true;
}
```

התוכן של {{ isRight }} יהיה תקין והפלט יהיה true.

מה לא יהיה תקין?

לא ניתן לרשום מספר פקודות, או התניה וכדומה.

חשוב!

כאשר קוראים לקומפוננט, מיד נוצר מופע של הקלאס המתאים ואפשר כבר להשתמש עם הערכים של השדות. הקלאס המתאים הוא הקלאס של הקומפוננט (מפורט בשיעור 1).

<u>שימוש בקומפוננט מספר פעמים:</u>

לכל קומפוננט קיים מופע משל עצמו, ולכן אם משתמשים בקומפוננט מספר פעמים, ייווצרו מספר מופעים של אותו הקלאס. שינוי ערך של שדה בקומפוננט אחד אינו משפיע על שדה בקומפוננט אחר למרות שנראה שאם מדובר באותו קומפוננט כל השדות מקושרים. אין ביניהם כל קשר, שהרי מדובר במופעים שונים לחלוטין.

לדוגמא:

להלן קריאה לקומפוננט פעמיים:

```
<first-name>{{name}}</first-name>
<first-name>{{name}}</first-name>
```

יש כאן קריאה פעמיים לאותו קומפוננט, ולכן ייווצרו 2 מופעים מסוג FirstNameComponent.

אם ישתנה השדה name של אחד מהם, הערך name של הקומפוננט השני אינו משתנה בהתאם, היות ומדובר במופעים שונים.

2. Property binding

שינוי ערכי ה-properties של תגיות html יהיה ע"י סוגריים מרובעות.

לדוגמא, כפתור יהיה disabled בהתאם לערך המשתנה המקושר:

```
<button [disabled]="isRight">press</button>
```

מה ההבדל בין **Property** לבין **attribute**?

ה-dom הוא התוצאה של ה-html למשתמש. ישנו קטע קוד html שבו מוגדרות התגיות השונות. לזה קוראים html. לאחר "קימפול", הפעלה של קוד Js ועוד, הקובץ מתורגם ל-dom וזה מה שלמעשה מוצג למשתמש. לדוגמא אם יש מאפיין של display:none ה-dom יתרגם את המסך בצורה כזאת שהתגית הצריכה להיות מוסתרת אכן תהיה מוסתרת.

Attribute הוא מאפיין השייך ל-html. אם המסך אינו מתרנדר, גם אם משנים את ה-attribute של ה-html, המשתמש למעשה נשאר עם ה-dom הקודם ואינו רואה את השינוי. באנגולר זה יכול לקרות, מכיון שניתן לשנות ישירות ערכי attr מבלי לרנדר את הדף.

Property הוא מאפיין השייך ל-dom. אם משנים prop ועדיין המסך לא התרנדר, השינוי יקבל תוקף באופן מיידי, היות והמשתמש תמיד רואה את תוצאת ה-dom.

ולכן באנגולר, יש לפנות תמיד ל-prop על מנת לקבל שינוי מיידי. השמות של ה-attr ושל ה-prop ברובם זהים. הבדל עיקרי באנגולר הוא, שכאשר יש צורך לשנות ערך של prop יש לפנות ע"י סוגריים מרובעות. ואילו ל-attr לא פונים עם סוגריים מרובעות.

פנייה ל attr אינה מומלצת במצבים דינמיים.

3. Event binding

הפעלת קוד ע"י אירוע. הקוד יכול גם להיות קריאה לפונקציה. יש לעטוף את האירוע בסוגריים **עגולים**.

רוב האירועים המוכרים מ-javascript ייכתבו אותו הדבר באנגולר, מלבד הקידומת "on".

אירועים של: onmousehover, onclick ייכתבו באנגולר (mousehover), (click) וכן הלאה.

בקובץ ה-ts הגדרת הפונקציה כאשר היא בתוך הקלאס נכתבת ללא המילה "function".
כמו כן, פניה למשתנים בתוך פונקציה תהיה ע"י קידומת "this".

לדוגמא:

```
export class AppComponent  {
  isRight:boolean=false;

  enableBtn(){
  this.isRight=true;
}}
```

app.component.html

```
<button [disabled]="isRight" (click)="enableBtn();">press</button>
```

בלחיצה על הכפתור, הפונקציה המקושרת לאירוע מתבצעת ותפקידה לשנות את הערך של המשתנה isRight. המשתנה הזה מקושר ל-property של disabled ולכן ברגע שלוחצים על הכפתור, הכפתור מקבל 'disabled='true שזה בעצם נעילת הכפתור...

אגב, אירוע (click) אינו מוכרח להיות מוגדר דווקא בתגית button אלא יכול להיות מוגדר גם על תגית של div ועוד תגיות רבות מקבלות אירוע זה.

4. 2-ways-binding

כיצד משנים ערך של שדה דרך input?

לכאורה היה צורך בשני שלבים:

שלב ראשון, לקשר את ה-input לאירוע של keyup וכל הוספת תו, האירוע יקרא לפונקציה שתעדכן את הערך של השדה לערך החדש שנוצר לאחר הוספת תו.

שלב שני, לקשר את ה-value של ה- input לשדה.

לדוגמא:

```
  <input [value]="myName" (keyup)="myFunc()"/>
```

במקום זה, ניתן להגדיר את הקישור בפקודה אחת.

הקישור יוגדר כך:

```
<input  [(ngModel)]="myName" />
```

שימו לב! **ngModel** אות גדולה של M, וזה לא **ngMoudle**...

לדוגמא:

```
export class AppComponent  {
  isRight:boolean=false;
  myName=" ";
}
```

app.component.html

```
<input type="text" [(ngModel)]="myName"/>
```

יש להוסיף הצהרה של **FormsModule** בקובץ app.module.ts שנמצא תחת תיקיית app.

```
import { FormsModule } from '@angular/forms';
```

וכן, יש להוסיף למערך imports את FormsModule:

```
@NgModule({
  imports:      [ BrowserModule, FormsModule ],
…
})
```

ניתן להוסיף ל-html תגית שבו ניתן יהיה לצפות מיד בשינוי ה-Input את השינוי.

לדוגמא:

```
<div>{{myName}}</div>
```