**inןLesson 57-62 – Angular**

#### Lesson 57 - Modules, Components, Layout

* יצאה ב 2010 ע"י google
* כמו react גם angular מבוססת על components
* ב angular אפשר להתשמש ב cli כדי לכתוב פקודות וליצור דברים בצורה יותר יעילה

פקודות בסיסיות

* ng new 'name-of-project' – פתיחת פרויקט חדש
* ng s –aot -o –פקודת הרצה של הפרויקט (בדומה ל npm start)

[ s (serve) – פקודת הרצה על שרת (דיפולט (4200

- aot (ahead of time)קומפיילר נוסף - בודק עד שגיאות שהיו יכולות לקרות בזמן הריצה

o – פתיחת דפדפן (ללא o תתבצע הרצה של התכונית ללא פתיחת דפדפן ]

* יצירת comp – ng g c components/"name-of comp" –skip-tests

באנגולר אנחנו מקבלים תיקיות comp מובנות

* כל תיקייה מכילה שלושה קבצים : html, css, ts
* הפקודה skipTests-- היא כדי שלא יוצר לנו קובץ נוסף (spect.ts)

# פקודות עם

* פקודות אלו מכונות annotation/decoration – אלו מילים שמורות שנכתבות מעל אלמנט תכנות כלשהו (לדוגמה אובייקט, משתנה, פונקציה)
* פקודה זו תתבצע פעם אחת במהלך ההרצה של האתר
* הפקודה מציינת ל framework של ng מה הייעוד של אותו אלמנט

לדוגמה

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.scss']

})

* ה annotation מסוג Component@ אומר ל ng שהאובייקט שבא אחרי זה יקבל הגדרות מסוימות
* במקרה הזה Component צריך לקבל שלושה דברים:

1. templateUrl – מיקום קובץ ה html
2. styleUrls – מיקום קובץ css
3. selector – תג שמייצג את ה component

# Modules

* class שמבחינה לוגית מכיל components (אזור נפרד במערכת שמכילה comp)
* כל component חייב להיות משויך ל module ויכול להיות משויך רק ל module אחד
* ישנם modulesמובנים של Angular (כמו FormsModules, RouteModule) בהם נשתמש באפליקציה
* כדי להכניס comp אל module צריך לכתוב ב cli את הפקודה -m
* באופן דיפולטיבי כל comp שניצור ייכנס תחת app.modules

# app.modules

* שימוש ב annotation נוסף – NgModule – מתאר שמחלקה היא מסוג module

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

NgModule מכיל כמה ארבעה חלקים:

1. declarations – מערך שמכיל את ה Comp שנמצאים תחת אותו module
2. imports – מערך שמכיל modules שאנחנו רוצם להתשמש באפליקציה שלנו

(ב module אותו אנחנו רוצים להחצין יהיה צריך להיות כתוב export על אותו comp)

1. providers – יכיל קבצי services (דפי ts שמכילים פונקציות ייעודיות)
2. bootstrap – איזה compעולה ראשון

# Setting new Project

1. בתוך src נפתח תיקיית components בה יהיו כל ה comp שנבנה
2. נפתח את הcli וניצור comp חדש – ng g c components/layout –skipTests
3. נמחק את קבצי ה html, css, ts ו test app
4. בתוך דף index.html הראשי נחליף את ה selector של app ב selector ה layout שייצרנו

<body>

  <app-layout></app-layout>

</body>

1. בתוף דף ה styles.css הראשי ניתן הגדרות בסיסיות לאתר

html, body {

  height: 100%;

  padding: 0;

  margin: 0;

  background-color: lightblue;

  font-size: x-large;

  text-align: center;

}

1. נכנס ל app.modules – נוכל לראות ש LayoutComponent יובא בצורה אוטומטית תחת ה declarations

@NgModule({

  declarations: [

    LayoutComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule

  ],

  providers: [],

  bootstrap: [LayoutComponent]

})

* בתוך bootstrap נכניס את LayoutComponent במקום AppComponent

#### Lesson 58 - Angular - Binding, Directives, Pipes, Routing

* נכיר קצת את מבנה ה comp שלנו

export class HomeComponent implements OnInit {

הגדרת משתנים

    ngOnInit(): void {

אתחול משתנים כאשר ה comp נבנה

            }

}

כתיבת פונקציות

* כל comp מקבלים כ class
* כל class מקבל אוטומטית Interface שנקרא OnIint – באזור הזה נגדיר את המשתנים שלנו
* הפונקציה ngOnInit דומה ל compDidMount ב react - באזור שלה נאתחל משתנים ונקרא לפונקציות שאנחנו רוצים שיבצעו כאשר ה comp נבנה (כמו קריאה לשרת)
* מתחתיה אנחנו כותבים פונקציות רגילות

לדוגמה

export class HomeComponent implements OnInit {

    public currentDiscount: number;

    ngOnInit(): void {

        this.currentDiscount = 10;

    }

}

* כיוון שאנחנו ב class כדי לקרוא לפרמטר או לפונקציה צריך להשתמש ב this

# Binding

1. **Interpolation** – שימוש בסוגריים מסולסלות {{}} כדי להזריק משתנים ל DOM

<h4>Only now - {{currentDiscount}}% discount on all products!</h4>

* ב angular אנחנו משתמשים בסוגריים כפולים

1. **Property Binding** – שינוי property של HTML Object (כמו color, width..)

<img  src="/assets/images/home.png" [width]="imageWidth">

* את ה property שרוצים לשנות מכניסים בתוך סוגריים מרובעים
* את המשתנה שהגדרנו ב comp נכניס תחת "" כמו string (בשונה מ react ו html )

export class HomeComponent implements OnInit {

 public imageWidth: number;

1. **Event Binding** - שינוי property כתוצאה מהפעלת Event

    <button (click)="decreaseImage()">&darr;</button>

* אירוע נכתב ב lowercase ומוקף בסוגריים עגולות
* את הפונקציה נכניס תחת גרשיים כמו string

בדף ה ts נכתוב את ה method

 public decreaseImage(): void {

        this.imageWidth -= 10;

    }

* את ה method כותבים כ method של class (עם public/private)

* אפשר להגדיר מה ה method מחזירה
* פונקציות שנכתבות ב class נכתבות ללא arrow אלא רק עם סוגריים מסולסלים

1. **Two Ways Binding** –input שמתקבל מ dom משנה property שמשפיע בחזרה על ה dom

    <input type="range" min="100" max="500" [(ngModel)]="imageWidth">

* כדי שמשנה יהפוך ל two-ways צריך להתשמש ב directive מובנה של ng שנקרא ngModel

ל ngModel יש כמה שימושים:

1. שימוש ב two-ways-binding – אם נעטוף אותה בסוגריים עגולים (event-binding) וסוגריים מרובעים [property-binding], ngModel תיצור קשר בין המשתנה שנשים בתוכה לבין ה value שנגדיר לאותו משתנה בקובץ ts

* כלומר שבשונה מאיל שעבדנו עד עכשיו אין צורך להשתמש באירוע! שימוש ב two-binding ב ng יוצר חיבור בין ה את ה input שהמשתמש מכניס אל המשתנה שהגדרנו

נסביר שוב את העיקרון של Modules

* ngModel הוא כלי שנמצא ב module מובנה של angular שנקרא FormModule

FormModule

AppModule

NgModule

HomeComp

* כדי שנוכל להתשמש בו אנחנו חייבים קודם לייבא אותו לתוך ה app.module
* כדי לייבא את NgModule חייבים לייבא את ה module שמכיל אותו (FromModuel)
* ה Import הראשוני לא יעבוד בצורה אוטומטית, צריך לכתוב אותו בעצמנו

# Template Refence variable

* מזהה ייחודי לאלמנט HTML

<img #myImage src="/assets/images/home.png" [width]="imageWidth">

* נכתב עם סולמית
* המשתנה הזה הוא שווה ערך לפקודה getElementById
* דרכו אפשר לגשת ל properties של אלמנט ה HTML

# Directives

* הוראה שאנחנו נתונים לתגית HTML שמבצעת פעולה כלשהי על התגית

קיימים שני סוגים של directives

1. attr directives – מבצעת שינוי על תגית קיימת (צבע, מיקום התנהגות)

 <p ngClass="greenish">Apples</p>

* מתפקדת כמו class ?
* ה directive "greenish" מכיל css שמשפיע על תגית p

   <p  greenish >Apples</p>

* אפשר להכניס את ה directive כמו שהוא

1. structural directives – מכניסה את התג ל dom כתוצאה מתנאי

* לפני structural directive צריך לתת כוכבית \*
* כברירת מחדל התגית לא משויכת ל-DOM.

**קיימים שלושה כאלה:**

1. ngIf - דומה ל conditional rendering

    <p \*ngIf="isWinter">Oranges</p>

* יכניס את p רק אם התנאי שבפונקציה הוא true

יש אפשרות להשתמש ב else ולרנדר אלמנט ל dom כאשר התנאי הוא false

<div \*ngIf="condition; then thenBlock else elseBlock"></div>

<ng-template #thenBlock>Content to render when condition is true.</ng-template>

<ng-template #elseBlock>Content to render when condition is false.</ng-template>

* then – ירנדר ל dom כאשר true
* else – ירנדר ל dom כאשר false

הערה – לפי ng יש הבדל בין ngIf (directive's attribute name) ל NgIf (directive class )

1. ngFor – מתפקד כמו ל map ב react

    <h6 \*ngFor="let v of vegetables">{{v}}</h6>

* יוצר List מהמערך שנמצא ב vegetables

1. ngSwitch

<div [ngSwitch]="hero?.emotion">

<app-happy-hero \*ngSwitchCase="'happy'" [hero]="hero"></app-happy-hero>

<app-sad-hero \*ngSwitchCase="'sad'" [hero]="hero"></app-sad-hero>

<app-confused-hero \*ngSwitchCase="'confused'" [hero]="hero"></app-confused-hero>

<app-unknown-hero \*ngSwitchDefault [hero]="hero"></app-unknown-hero>

</div>

# Routing

1. ב app.module אנחנו רואים שיש לנו AppRoutingModule

import { AppRoutingModule } from './app-routing.module';

1. נלך אל ה layout ונוסיף את התגית router-outlet בתוך ה main

  <main>

        <router-outlet></router-outlet>

  </main>

* זה דומה ל switch שבנינו ב React, התוכן שבתוך ה main ישתנה בהתאם לנתיב

1. נלך את קובץ app-routing.module

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

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

const routes: Routes = [];

export class AppRoutingModule { }

* בתוך במשתנה routes יש מערך בו נכתוב את ה routesשלנו
* כל route נכתב כאובייקט - שמקבל path ו component
* ב ng ה path נכתבים ללא backslash ((/

const routes: Routes = [

  { path: "home", component: HomeComponent },

  { path: "", redirectTo: "/home", pathMatch: "full" }, // full = exact

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

];

* כדי ליצור redirect – path מקבל string ריק (הניתוב הראשי) , redirectTo מקבל את ה path (במקרה הזה home), ו pathMatch מקבל full
* כדי לטפל בשגיאות של route ניצור route שלתוך ה path מקבל (\*\*) – במקרה ומשתמש ינסה לגלוש ל route שלא קיים, ה route הזה יופעל (לכן הוא מכונה גם wildcard)

1. לבסוף ניצור comp תפריט (menu) בו נכתוב את הלינקים

* ng g c components/menu --skipTests

<a routerLink="/home" routerLinkActive="active"

[routerLinkActiveOptions]="{exact: true}">Home</a>

* routerLink – directive של ng שמשמנה את ההתנהגות של תג a – בתוכו נכתוב את ה path הפעם עם backslash (לא משתמשים ב href כדי לא לרענן את הדף)
* routerLinkActice – class דרכו נוכל לבצע css כאשר הלינק פעיל
* [routerLinkActiveOptions]="{exact: true}" – מדמה exact שהיה לנו ב react

[הערה – משהו שראיתי בסרטון - הגנה על route בשימוש ב guard

<https://www.youtube.com/watch?v=Np3ULAMqwNo> (דקה 8)

* הפקודה ng g guard "name-of-comp" יוצרת comp מסוג guard
* Guard הוא שירות של ng שמאפשר לתת תנאים ל router
* בתוך אובייקט ה route עליו רוצים להגן (לדוגמה rout של admin) נכניס field שנקרא canActivate : וניתן לו מערך שמכיל את AdmibGuard
* canActivate יאפשר להיכנס ל route רק אם AdmibGuard יחזיר לו true ]

# Programming Routing

* ניווט מתוך דף ts

1. ב constructor עושים ID ל Module של ng שנקרא Router

constructor (private myRouter: Router) { }

1. מתוך האובייקט ש ng יוצרת קוראים לפונקציה navigateByUrl()

this.myRouter.navigateByUrl("/products");

* לתוך הפונקציה מכניסים את ה route אליו אנחנו רוצים לנווט

## Guards

* Service שעוזר להגן על routes
* יוצרים בעזרת ה cli – ng g g

למדתי על שני method:

1. canActivate – מאפשרת לחסום כניסה ל route (לדוגמה במצב של login או admin)

* canActivteChild – מאפשרת לחסום רק childRoutes

1. canDeActivted – מאפשרת לחסום יציאה מ rouse

לדוגמה – guard מסוג canAcrive

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

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from '@angular/router';

import { Observable } from 'rxjs';

@Injectable({

  providedIn: 'root'

})

export class EditGuard implements CanActivate {

  canActivate(

    next: ActivatedRouteSnapshot,

    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

    return true;

  }

}

* ה class יחזיר true או false ובהתאם לכך תותר הכניסה ל route
* צריך להכניס את ה guard אל ה route

{

    path: "recipes", component: RecipeComponent,

    canActivate : [LoginGuard],

},

## Resolver

* method של ng שמאפשרת להעביר data דרך ה route
* על ידי resolver אפשר להעביר data ל comp לפני שהוא נטען

לדוגמה – resolver שמבצע קריאה לשרת להביא data

@Injectable({

  providedIn: 'root'

})

export class RecipeResolver implements Resolve<RecipeModel> {

  constructor(private recipeService: RecipeService) { }

  resolve(

    route: ActivatedRouteSnapshot,

    state: RouterStateSnapshot

  ): Observable<RecipeModel> | Promise<RecipeModel> | RecipeModel {

    return this.recipeService.getRecipeAsync(route.params.id)

  }

}

* צריך להטמיע את ה resolver ב route

{ path: ":id", component: RecipeDetailsComponent,

resolve: { recipe: RecipeResolver } },

# Pipes

* שינוי המתבצע על ערך ב-Interpolation (ועוד).

פקודה שהינה בסופו של דבר פונקציה :

* הערך שאנו רוצים להציג ב-Interpolation נשלח לפונקציה.
* הפונקציה מחזירה ערך חדש והערך החדש מוצג במקום הישן.

לדוגמה – הצגנו date בפורמט מסויים

1. בקובץ ts יוצרים משתנה date ואתחנלנו אותו

export class HomeComponent implements OnInit {

    public currentDate: Date;

    ngOnInit(): void {

        this.currentDate = new Date();

    }

1. על ידי {{}} נזריק את המשתנה ל dom

Only now, in {{currentDate }}

* אם נשאיר את משתנה כמו שהוא נקבל

Mon Mar 30 2020 14:15:23 GMT+0300 (שעון ישראל (קיץ))

* על ידי שימוש ב Pipe נוכל לעצב את התכון של המשתנה

1. נשתמש ב pipe של ng שנקרא date

Only now, in {{currentDate | date:"MMMM yyyy"}}

* pipe מסומן כקו יחיד |
* משמאל לקו נכניס את המשתנה שלנו
* מימין נכתוב את שם ה pipe (date)
* במקרה הזה בחרנו להציג בפורמט של חודש ושנה (MMM yyyy)

#### Lesson 59 - Angular - ng-content, Service, DI

# ng-content

* תגית של ng שמאפשרת ליצור תוכן מודולרי
* במקום ng-contentאשר להכניס כ מה שרוצים – text, html elements
* השתמשנו בה כדי ליצור כותרת בעלת אותו css עם תוכן שונה

1. יצרנוheadilng comp
2. בחלק ה html יצרנו כותרת שבתוכו כתבנו את התגית ng-content

<h2>

    <ng-content></ng-content>

</h2>

1. נתנו css
2. ב comp אחר אנחנו קוראים ל comp שיצרנו

<app-heading>Welcome to Northwind Website!</app-heading>

1. נכניס טקסט ונראה שהוא מופיע עם ה css שנתנו

# Angular Services

* מחלקות המבצעות פעולות לוגיות בלבד!
* כדי ליצור service נשתמש בפקודה bg g s services/""name-of-file" –skipTests

דוגמה בכיתה - יצרנו CalcService – class בה יהיו פונקציות שקשורות לחישוב

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

@Injectable({

    providedIn: 'root'

})

export class CalcService {

}

* בתוך ה class נכתוב את הפונקציות

export class CalcService {

    public add(a: number, b: number): number {

        return a + b;

    }

    public sub(a: number, b: number): number {

        return a - b;

    }

    public mul(a: number, b: number): number {

        return a \* b;

    }

    public div(a: number, b: number): number {

        return a / b;

    }

}

* כדי שנוכל להתשמש בפונקציות אנחנו צריכים ליצור אובייקט חדש מה class
* ב ng אנחנו לא יוצרים את האובייקט בעצמו! - אנחנו צריכים להשתמש במנגנון שנקרא Dependency Injection

# Dependency Injection (DI)

* בגדול DI הוא צורת כתיבה שבה class מקבל אובייקטים ממקור חיצוני ולא יוצר אותם בעצמו

משמעות השם:

* Dependency - כי ה-Component שלנו תלוי באובייקט שאנחנו נותנים.
* Injection - כי זה נחשב שהמערכת יוצרת את האובייקט ו"מזריקה" אותו אלינו ל-Component.

יתרונות:

* שימוש חוזר באותו האובייקט
* מניעת הצורך ליצור אותו בעצמנו
* בדיקות - המערכת יכולה ליצור עבורנו אובייקטים כאלו

דוגמה בכיתה – השתמשנו ב CalcService ב about comp

1. ניגשים ל constructor של ה comp בו רוצים להשתמש ב service ומגדירים פרמטר שמקבל את ה service כ type

constructor(private myCalcService: CalcService) { }

* כדי לחסוך בקוד נשתמש ב typescript magic – אם נכתוב ב constructor לפני הפרמטר את השיוך שלו (public/private) לא נצטרך לאתחל אותו
* עכשיו כאשר נקרא לפרמטר ng יוצרת בשבילנו אובייקט ואנחנו יכולים לקרוא לפונקציות שב service.

public testCalcService(): void {

        const addResult = this.myCalcService.add(10, 20);

        const subResult = this.myCalcService.sub(10, 20);

        const mulResult = this.myCalcService.mul(10, 20);

        const divResult = this.myCalcService.div(10, 20);

    }

1. כדי ש ng תזהה את ה service צריך לומר לה היכן למצוא אותו

הservice יכול להיות חשוף ב 3 רמות:

1. רק לאותו comp – ב annotation של ה Comp צריך להוסיף providers אליו נכניס את ה service

@Component({

    selector: 'app-about',

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

    styleUrls: ['./about.component.css'],

    providers: [MathService] // צור אובייקט מהמחלקת השירות הזו אך ורק עבור הקומפוננט הזה

})

1. ברמת ה module – גם כן בחלק של ה providers צריך לקרוא ל service

providers: [MathService], // צור אובייקט ממחלקת השירות הזו עבור כל הרכיבים ששייכים למודול

* במקרה שמשייכים ל app.module זה משייך לכל ה comp שבאפליקציה

1. ברמת כל ה app – זו הגדרת ה default שיש ל service כל עד משאירים את ה annotation מסוג Injectable

@Injectable({

    providedIn: 'root'

})

#### Lesson 60 - Observable, @Input, @Output

# Observable

* ל ng יש module שמאפשר לנו לבצע קריאה ל data משרת - HttpClientModule
* בשונה מ fetch או ajaxשהיו מחזירות לנו את ה data כאובייקט Promise, הפונקציות ב HttpClientModule יחזירו לנו את ה data כאובייקט מסוג Observable

הבדלים בין Promise ל-Observable

* יצירת אובייקט Promise מיד מתחילה אותו.
* יצירת אובייקט Observable לא מתחילה אותו, אלא רק קריאה לפונקציה שנקראת subscribe מאתחלת אותו
* על ידי subscribe אפשר להאזין לשינויים באובייקט (לדוגמה אם משתמשים ב observable כדי להאזין לשינויים ב route)
* Promise ברגע שמתחיל - לא ניתן לביטול. או שהוא יצליח או שיכשל.
* Observable ניתן לבטל לאחר שהתחיל גם אם הוא לא סיים.
* Promise יכול לדווח הצלחה אחת או כשלון אחד.
* Observable יכול לדווח אינסוף הצלחות או כשלון אחד.
* ברגע ש Observable מסתיים בהצלחה - לאחר דיווח ההצלחה האחרונה -
* הוא מפעיל עוד פונקציה אחת עבור Complete. זו הפונקציה השלישית ב-subscribe.
* עם Promise ניתן לעבוד ע"י async-await.
* עם Observable לא ניתן.

דוגמה בכיתה – הצגת מוצרים משרת

1. בניית תשתית

* ייבאנו קובץ products.json והרמנו אותו כשרת ע"י server-json products.json
* ייצרנו תיקיית models עם ProductModel

1. נלך ל app.module ונייבא את ה HttpClientModule

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

* כיוון שזו הפעם הראשונה שאנחנו מייבאים צריך לכתוב את ה module בעצמנו
* לא לשכוח להוסיף לחלק import של ה module

1. נפתח service חדש בו נכתוב את הפונקציות להבאת data – אנחנו הולכים להביא Products אז נקרא לו ProducsService
2. בתוך ה constructor נכתוב משתנה http שיקבל את HttpClient כ type

    constructor(private http: HttpClient) { }

* דרכו נוכל לקרוא לפונקציות שב module

1. נכתוב את הפונקציה: (לשים לב שלא כותבים עם arrow)

public getAllProductsAsync2(): Observable<ProductModel[]> {

    }

* אחרי שם הפונקציה - פותחים נקודתיים ומצהירים על סוג האובייקט שהפונקציה מחזירה (במקרה הזה observable)
* לאחר מכאן - פותחים סוגריים משולשים ומצהירים ועל התוכן שנקבל (מערך של ProductModel)

1. הפקודה לקריאה לשרת מורכבת משורה בודדת ולכן אפשר להחזיר אותה ב return

return this.http.get<ProductModel[]>("http://localhost:3000/products");

}

* נקרא לפרמטר http ו לפונקציה get
* צריך שוב להצהיר על מה שאנחנו מתכוונים לקבל – מערך של מוצרים
* נפתח סוגריים ונכניס את הurl

1. ניצור comp מסוג products – ng g c components/products –skipTests
2. נגדיר משתנה products אליו נכניס את ה data

export class ProductsComponent implements OnInit {

    public products: ProductModel[];

1. בתוך ה constructor נייבא את ה service

constructor(private myProductsService: ProductsService) { }

1. בתוך ngOnInit (החלק שמאתחל את המשתנים כאשר ה comp עולה) נקרא לפונקציה

this.myProductsService.getAllProductsAsync2()

            .subscribe(

                products => this.products = products,

                err => alert("Error: " + err.message),

                () => console.log("Done")

            );

* כאמור כדי לבצע גלישה אנחנו צריכים לקרוא לפונקציה subscribe

הפונקציה מקבלת אליה כפרמטרים שלושה פונקציות :

1. תתפקד כמו then של fetch – תנהל את ה data במקרה של הצלחה - במקרה שלנו הכנסה של ה data שמתקבל לתוך משתנה products של ה class
2. תתפקד כמו catch של fetch – תתרחש במצב של טעות
3. פונקציה שתמיד תתבצע

**דרך נוספת - המרת ה observable ל promise**

* במידה ואנחנו רוצים להשתמש במנגנון async-await נוכל להמיר את observable ל promise

1. כתבנו פונקציה נוספת

public getAllProductsAsync3(): Promise<ProductModel[]> {

        return this.http.get<ProductModel[]>("http://localhost:3000/products").toPromise();

    }

* נצהיר שסוג האובייקט שנקבל הוא promise
* נשרשר פקודה שנקראת toPromise()

1. בחזרה ל products – עכשיו נוכל להתשמש במנגנון async-await

 async ngOnInit() {

  try {

this.products = await this.myProductsService.getAllProductsAsync3();

        }

  catch (err) {

  alert("Error: " + err.message);

        }

}

* לפני ngOnInit נכתוב async
* נפתח try-catch כדי לתפוס טעות

# Parent-Child Communication - @input

* אנחנו רוצים לבנות comp שיקבל בגדרות מבחוץ (בדומה ל propsב React)

1. נפתח comp חדש – ng g c components/thumbnail –skipTests
2. בחלק ה html ניצור תג Img

<div > img> </div>

1. אנחנו רוצים שהתג יקבל property מבחוץ ולכן נשתמש ב property-Biding

* בדף ts - ניצור משתנים ל src, width ו height

    public imageSource: string;

    public imageWidth: number;

    public imageHeight: number;

* בדף html - נסמן את ה attr ב Img בתוך סוגרים מרובעים

<img [src]="imageSource" [width]="imageWidth" [height]="imageHeight">

1. אבל כמו שאמרנו אנחנו מעוניינים ש thumbnail יקבל את הערכים ל property האלה מ comp אחר ! (parent)

* בשביל זה אנחנו משתמשים ב annotation שנקרא @input – זה מדווח ל ng שהמשתנה הזה חשוף ל comp אחרים

    @Input() // יוכל לשלוח אליו את המידע Parent-כלומר ה Parent-חשיפת המשתנה הזה עבור ה

    public imageSource: string;

    @Input("w")

    public imageWidth: number;

    @Input("h")

    public imageHeight: number;

* אפשר בתוך הסוגריים להכניס allies שייצג את המשתנה ב parent

1. נקרא ל thumbnail ב products וניתן לו את הערכים שאנחנו רוצים

<app-thumbnail

imageSource="/assets/images/products/{{p.id}}.jpg"

w="50"

h="50"

app-thumbnail>

# Child-Parent Communication - @Output

* עכשיו אנחנו רוצים לבצע את התקשורת בכיוון ההפוך – בעקבות אירוע שמתרחש ב child יהיו שינוי ב parent.

דוגמה בכיתה – אנחנו רוצים שבעקבות אירוע התמונה שב thumbnail תוצג בצד

* אנחנו יכולים לעשות זאת על ידי העברה של ImageSource מthumbnail (ה (child ל products (ה parent) כאשר אירוע כלשהו יתבצע ב child (במקרה שלנו האירוע יהיה כניסת עכבר)

כדי לעשות זאת אנחנו צריכים להשתמש בשני אלמנטים של ng:

1. @Output - annotation שמדווח לng שהפונקציה מחצינה data החוצה
2. אובייקט EventEmitter – דרכו ניצור פונקציה שתחצין את ה data

חלק א – הקוד ב child

1. ב html הוספנו אירוע mouseenter ל div עם הפונקציה imageEnter()

<div (mouseenter)="imageEnter()">

  <img [src]="imageSource" [width]="imageWidth" [height]="imageHeight">

</div>

1. בדף ה ts יצרנו אובייקט EventEmitter חדש וקראנו לו enter

   @Output()

    public enter = new EventEmitter<string>();

* לשים לב, יש כמה אובייקטים מסוג emit צריך לבחור את זה של @angular/core
* צריך להצהיר ל enter איזה datatype הוא הולך לקבל (במקרה הזה string)
* בנוסף הודענו ל ng על ידי @Output ש enter שולחת מידע אל parent

1. נכתוב את הפונקציה ImageEnter

   public imageEnter(): void {

        this.enter.emit(this.imageSource);

    }

* הפונקציה imageEnter של thumbnail קוראת לאובייקט enter
* האובייקט קורא לפונקציה emit שמקבלת אליה את imageSource – הפעולה הזו נקראת "העלאת אירוע"

חלק ב – הקוד ב parent

1. בתוך התגית של app-thumbnail נקרא ל enter ונכניס אליו פונקציה displayPreview()

<app-thumbnail

    imageSource="/assets/images/products/{{p.id}}.jpg"

    w="50"

    h="50"

   (enter)="displayPreview($event)"

app-thumbnail>

* enter מציינת אירוע ולכן אנחנו מקיפים אותה בסוגריים עגולים
* הפונקציה displayPreview מקבלת אליה משתנה $event – המשתנה מייצג אובייקט שנוצר בעקבות event דרכו אנחנו יכולים לגשת ל data על התגית
* במקרה הזה ב data שנקבל יהיה ה ImageSource!

1. בדף ts

* נאתחל משתנה hoverdImage – המשתנה הזה יקבל אליו את ה data שמגיע מה child

  public hoveredImage: string;

1. נכתוב את הפונקציה displayPreview()

    public displayPreview(imageSource: string): void {

        this.hoveredImage = imageSource;

    }

* הפונקציה כאמור מקבלת אליה את ImageSource ולכן אנחנו יכולים להכניס אותה לתוך משתנה המקומי (hoverdImage)

1. לבסוף - נחזור ל products.html וניצור תג img

<img [src]="hoveredImage">

* נשתמש ב property binding על src ונכניס לתוכו את hoverdImage

**אותו דבר בדרך נוספת –**

* אנחנו יכולים לחסוך את השימוש בפונקציות כדי להעביר את המידע על ידי שימוש ב Template Refence Variable (trv)

1. ניתן trv לתג ה img החדש שיצרנו

<img #myHovered class="hovered">

1. על ידי #myHovered  אנחנו יכולים לגשת ל properties של img בכל מקום בדף
2. נחזור ל enter ובמקום פונקציה ניתן לו את #myHovered

 <app-thumbnail

   imageSource="/assets/images/products/{{p.id}}.jpg"

   w="50"

h="50"

(enter)="myHovered.src=$event"

   ></app-thumbnail>

#### Lesson 61 - Angular – axios, Route Parameter, Form

# Axios

1. דרך נוספת להביא data משרת
2. צריך להתקין npm i axios
3. גם axios מחזירה Promise ולכן נוכל להתשמש בה ב async-await

  public getAllProductsAsync4(): Promise<ProductModel[]> {

        return new Promise<ProductModel[]>((resolve, reject) => {

            axios.get<ProductModel[]>("http://localhost:3001/products")

                .then(response => resolve(response.data))

                .catch(err => reject(err));

        });

    }

1. צריך לבנות את הפונקציה עם resolve, reject
2. ה response שמתקבל מכיל דברים נוספים חוץ מה data שאנחנו רוצים
3. לכן אנחנו צריכים לחלץ ממנו את ה data שלנו שנמצא תחת אובייקט data

.then(response => resolve(response.data))

# Route Parameters

* הוצאת pramater שנמצא בתוך ה route
* לרוב נשתמש כדי לחלץ משם id (במקרה שלהו id של מוצר) שאיתו נרצה לבצע דברים (במקרה שלהו הבאת פרטים של מוצר ספציפי)

דוגמה בכיתה – נבנהcomp שיציג לנו פרטים של מוצר ספציפי

1. ב ng cli ניצור comp חדש – ng g c components/details –skipTests
2. נלך ל app.router ונבנה route ל comp החדש

    { path: "products/:prodID", component: DetailsComponent },

* לשים לב - ה route מתחיל ללא /
* נלך לproducts comp ונהפוך את p.name לקישור

<a routerLink="/products/{{p.id}}">{{p.name}}</a>

* נעטוף את p.name בתג a – לשים לב - ה route בתג a מתחיל עם /
* נכניס לקישור את product.id על ידי interpolation
* לחיצה על הקישור צריכה להביא אותנו ל details comp
* עכשיו אנחנו רוצם לחלץ את ה id שנמצא ב route

1. ב constructor אנחנו צריכים להתשמש ב service של ng שנקרא Activated Route

constructor(private myActivatedRoute: ActivatedRoute) { }

* לא לשכוח לאתחל את המשתנה על ידי Public/private
* נלך לפונקציה ngOnInit (שמופעלת כאשר האובייקט נבנה) ונשתמש ב myActivatedRoute כדי לחלץ את ה id

const id = +this.myActivatedRoute.snapshot.params.prodID;

snapshot – מבצע "צילום" של ה route (בדומה ל match מ react)

params – תן לי את כל הפרמטרים של ה route (כל משתנה שנמצא מאחורי נקודתיים :(

ProdID – הפרמטר הספציפי שאנחנו רוצים

1. נלך אל ProductServices ונכתוב פונקציה שתביא לנו פריט יחיד

    public getOneProductAsync(id: number): Promise<ProductModel> {

        return this.http.get<ProductModel>("http://localhost:3001/products/" + id).toPromise();

    }

* הפונקציה תקבל id – מזהה ייחודי לפריט שאנחנו רוצים
* הפעם אנחהו רוצים שהפונקציה תחזיר לנו Promise ולכן נשרשר לה את הפקודה toPromise()

1. נחזור ל details comp

* נבצע DI ל ProductServices ב constructor

constructor(private myActivatedRoute: ActivatedRoute, private myProductsService: ProductsService) { }

1. נגדיר משתנה שיקבל אליו את המוצר

    public product: ProductModel;

1. נקרא לפונקציה בחלק של ngOnInit

    async ngOnInit() {

        try {

    const id = +this.myActivatedRoute.snapshot.params.prodID;

    this.product = await this.myProductsService.getOneProductAsync(id);

        }

        catch (err) {

            alert(err.message);

        }

    }

* כיוון שאנחנו מחזירים Promise אפשר להשתמש במנגנון async-await

# Forms

* יצירת form להוספת מוצר

1. יצירת insert comp – ng g c component/insert –skipTests
2. נגדיר משתנה product חדש

public product = new ProductModel();

1. בחלק ה html נבנה form עם 3 תיבות Input ו button

<form>

<input type="text" name="name" [(ngModel)]="product.name">

    <input type="number" name="price" [(ngModel)]="product.price">

    <input type="number" name="stock" [(ngModel)]="product.stock">

    <button (click)="addProduct()">Add</button>

</form>

* המערכת של ng מחייבת אותנו לתת name (מזהה ייחודי ב form) לכל אחת מתיבות ב Input
* נבצע two-way-binding עם product – זה קושר את attr מסוג value של תיבת input למשתנה שייצרנו (וזה חוסך לנו שימוש בפונקציות change)

1. נלך אל ProductServices ונכתוב פונקציה להוספת מוצר

 public addProductAsync(product: ProductModel): Promise<ProductModel> {

  return this.http.post<ProductModel>("http://localhost:3001/products", product).toPromise();

    }

* ה request היא מסוג post כיוון שאנחנו שולחים data ב body (במקרה הזה המוצר החדש)

1. נחזור ל details comp

* נבצע DI ל ProductServices ב constructor

constructor(private myProductsService: ProductsService) { }

1. בחלק ה html נשים אירוע click על ה button

    <button (click)="addProduct()">Add</button>

1. האירוע קורא לפונקציה addProduct שתפעיל את הפונקציה שייצרנו להוספת מוצר ל db

 public async addProduct() {

    try {

    const addedProduct = await this.myProductsService.addProductAsync

(this.product);

            alert("Product has been added. ID: " + addedProduct.id);

            this.myRouter.navigateByUrl("/products");

        }

        catch(err) {

            alert(err.message);

        }

    }

* לבסוף אנחנו רוצים אחרי הוספת המוצר לנווט בחזרה לדף המוצרים

1. נבצע DI ל service מובנה של ng שנקרא Router

constructor(private myRouter: Router) { }

* אחרי ה alert נוסיף פקודת ניווט

 this.myRouter.navigateByUrl("/products");

#### Lesson 62 - Validation, Lazy Loading, Child Routes, Custom Pipe, Custom Directive

# Forms Validation

* עושים תמיד validation ל data מתיבות Input גם ב client וגם ב serve
* רק כאשר צריך לוודא מול db אי אפשר לעשות ולדייציה ל data ב client

ולדייציה ב client

* לתת מענה מהיר ל client
* לא להעמיס request על ה server

ולדייציה ב server

* לחסום התקפות של האקרים

דוגמה בכיתה – ולדייציה לטופס insert

1. נלך ל Input ונכתב הודעת שגיאה לכל אחד מהתיבות

<input type="text"  name="name" [(ngModel)]="product.name" >

    <span>Missing name</span>

* כרגע ההודעות מופיעות כל הזמן ואנחנו רוצים שהן יופיעו רק כאשר יש טעות
* נכניס attr של validation שאנחנו מקבלים מתנה מ ng

input type="text name="name" [(ngModel)]="product.name"

        required minlength="2" #nameInfo="ngModel" autofocus>

* במקרה שלנו בחרנו את required (שדה חובה) ו minlength (מינימום תווים)
* ניתן ל Input מזהה ייחודי (#nameInfo) ונכניס לתוכה את NgModel
* זה שימוש נוסף שאנחנו יכולים לעשות עם NgModel – בצורה הזאת ngModel נותן לנו מידע שלא קיים בתיבת input – במקרה שלנו אובייקט errors (אובייקט שמכיל שגיאות ולדייציה) ו "מצבים" של תיבת ה Input ושל טופס (form)

בכל רגע נתון יש 6 מצבים שמתקיימים כאשר 3 מהם true ו 3 מהם :false

1. pristine vs dirty – האם הערך של השדה שונה - כל עד לא הוכנס ערך על ידי משתמש pristine יהיה **true** ו dirty יהיה **false**
2. touched vs untouched – האם השדה איבד focus - כל עד לא השדה לא איבד focus הערך touched יהיה **true** ו untouched יהיה **false**
3. valid vs invalid – יקבלו true ו false בהתאם לחוקיות של ה validation שהגדרנו על Input
4. בחזרה לתרגיל – נשים את המזהה הייחודי שמכיל את ngModel על תג span

<span \*ngIf="nameInfo.invalid>Missing name</span>

* כיון שהכנסנו אותו תחת ngIf ה span לא יוצג כאשר התנאי הוא false (ואז כאשר נכניס ערך לתיבה הוא ייעלם)
* המצב עדיין לא אופטימאלי כיוון שאנחנו מקבלים את הודעת השגיאה על התחלה

1. בשביל לפתור את העניין אפשר להתשמש בשאר המצבים שהראינו מקודם

<span \*ngIf="nameInfo.invalid && nameInfo.touched">Missing name</span>

* בתנאי שהגדרנו התג יוצג כאשר הערך יהיה לא חוקי וגם אחרי שיצאנו מפוקוס

1. נוסיף עד תנאי ל name – מינימום אורך

<input type="text name="name" [(ngModel)]="product.name"

        required minlength="2" #nameInfo="ngModel" autofocus>

1. נוסיף עד span עם הודעת שגיאה

<span \*ngIf="nameInfo.invalid && nameInfo.touched">Name too short</span>

* עכשיו אנחנו מקבלים בעיה אחרת – שתי הודעות מוצגות ביחד וזאת כיוון ששניהם עונים על התנאי של untouched ו invalid

1. בשביל לפתור את זה אנחנו צריכים שיהיה הבדל בין שתי ההודעות

 <span \*ngIf="nameInfo.errors?.required && nameInfo.touched"> </span>

<span \*ngIf="nameInfo.errors?.minlength && nameInfo.touched"> </span>

* במקום להשתמש ב Invalid נבדוק אם הוולדיציה שאנחנו רוצים קיימת באובייקט errors (שאנחנו מקבלים במתנה מ ng)
* עכשיו ההודעה תוצג רק אם היא קיימת באובייקט,
* הדבר האחרון שאנחנו צריכים לבצע זה להשתמש ב elvis operator - ?. – זה בודק לננו אם אובייקט errors קיים ולא יקריס לנו את הקוד אם האובייקט ריק

1. אנחנו רוצים לעשות validation ברמת ה form

* נלך לתג ה form ונשתמש ב directive של ng שנקרא ngFom

<form #formInfo="ngForm">

* עכשיו אנחנו יכולים לקבל data על כל הטופס כמו שקיבלנו על ה input

1. נוכל להשתמש בזה כדי לבטל לחיצה על button - מספיק שתיבת Input אחת תהיה Invalid בשביל שכל הטופס יהיה Invalid

<button [disabled]="formInfo.form.invalid" (click)="addProduct()">Add</button>

# Lazy Loading

* מגנון שעוזרת מבחינת performance
* באופן רגיל כל ה components שמופיעים ב app-routing.module נטענים כאשר האתר עולה
* Comp שנמצאים תחת מנגנון lazy loading ייטענו רק כאשר נגלוש לנתיב שלהם (כלומר קובץ ה js שלהם ייטעו מהרשת רק כאשר נגלוש לנתיב שלהם
* מנגנון lazy loading חייב להיות ברמת ה module – אנחנו צריכים לעשות הפרדה של ה comp ל module נפרד ואז הם ייטענו רק כאשר נגלוש לאותו module

דוגמה בכיתה – lazy loading על admin module

1. נפתח module חדש – ng g m admin
2. ניצור admin comp– ng g c admin/admin

* ng אוטומטית משייכת אותו ל admin.module כיוון שהוא הכי קרוב אליו בהיררכיה של התיקיות
* עיקרון חשוב ב Lazy-loading – ה comp שמשויכים ל module עליו אנחנו רוצים לבצע lazy-loading חייבים להיות רק באותו module, אם נקרא לאחד מהם ב app-routing.module נאבד את האפקט של lazy loading
* מכאן יוצא שאנחנו לא יכולים לכתוב route כמו שכתבנו עד עכשיו כיוון שאנחנו לא יכולים לייבא את ה comp לתוך app-routing.module

1. אנחנו יוצרים מערכת Routes ב admin.module

const adminRoutes: Routes = [

    { path: "", component: AdminComponent},

];

* לתוך המערכת הזו אין לנו בעיה להכניס את AdminComponent כיוון שהוא בתוך admin.module
* ה path הראשי יהיה empty string כיוון ש path : admin יהיה כתוב ב app-routing.module

1. נשלב את מנגנון ה route שיצרנו במערכת ה route הכללית

imports: [CommonModule, RouterModule.forChild(adminRoutes)]

* נקרא ל RouteModule (מודול מובנה של ng איתו אנחנו בונים routes)
* נקרא לפונקציה forChild וניתן לה את adminRoutes.

1. נלך אל app-routing.module וניצור routeאל ה module שלנו

* במקום לייבא את ה comp נשתמש בפונקציה loadChildren

{ path: "admin", loadChildren: ()

* הפונקציה תקבל import עם ניתוב ל module אליו אנחנו מכוונים

{ path: "admin", loadChildren: () => import("./admin/admin.module")

* then – (אם הגעת לאותה תיקייה) – תייבא את AdminModule

{ path: "admin", loadChildren: () => import("./admin/admin.module").then(m => m.AdminModule) }, // Lazy Loading

1. נלך ל menu וניצור קישור ל admin

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

# Child Routes

* Comp שאנחנו רוצים שיהיו ניגשים רק תחת route מסויים
* מה שזה אומר זה שאותם-Components (ה-Childs) יופיעו במקום התגית router-outlet שנמצאת בתוך ה-Component של ה-Parent Route שלהם.

דוגמה בכיתה - יצירת child's routes ל admin comp

1. בקובץ admin.module ניגש למערך ה routes שכתבנו

const adminRoutes: Routes = [

    {

        path: "", component: AdminComponent, children: [

    },

];

* אחרי AdminComponent נכתוב את הפקודה children ונפתח מערך
* בתוך המערך נכניס אובייקט path לכל אחד מה comp שאנחנו רוצים שיהי child

         { path: "security", component: SecurityComponent },

         { path: "auth", component: AuthComponent }

        ]

1. נכנס לדף html של AdminComponent ונכתוב את התגית router-outlet

<router-outlet></router-outlet>

* אם נגלוש עכשיו ל route של security או auth נראה שהן מופיעים תחת AdminComponent

1. נוסיף ב admin.hmtl קישורים כדי שנוכל לנווט אל ה routes השונים

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

<span> | </span>

<a routerLink="/admin/security" routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}">Security</a>

<span> | </span>

<a routerLink="/admin/auth" routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}">Auth</a>

* השתמשנו ב [routerLinkActiveOptions]="{exact: true}" כדי לוודא שפקודת ה css של active צובעת רק את ה route שבו אנחנו נמצאים

# Costume Pipes

* במקרה ואנחנו רוצים להציג data ב html בפורמט מסויים שלא קיים

דוגמה בכיתה – במקום 0 אנחנו רוצים להציג ---

1. ניצור pipe - ng g p pipes/"name-of-pipe" --skipTests

* נבדוק ש pipe נמצא ב app.module

1. ה pipe הוא class שמכיל פונקציה מסוג transform

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

@Pipe({

    name: 'zeroSymbol' // This is the pipe name!

})

export class ZeroSymbolPipe implements PipeTransform {

     transform(value: unknown, ...args: unknown[]): unknown {

    return null;

  }

}

* name – ה selector של ה pipe

איך עובד pipe :

אנחנו רואים שהפונקציה ש pipe מכיל יכולה לקבל 2 פרמטרים :

1. value - המשתנה שאנחנו רוצים לשנות
2. args – מערך של arguments (לא חובה לתת)

* הערך שהפונקציה תחזיר יהיה הפורמט החדש בו אנחנו מעוניינים

כל מה שאנחנו צריכים זה לעדכן את הפונקציה:

1. נגדיר שה type של value הוא number(אנחנו לא מגדירים מה הפונקציה מחזירה וזה אוטומטית נחשב כ any)

  public transform(value: number) {

return value === 0 ? "---" : value;

}

* בתוך הפונקציה נגדיר תנאי – אם ה value שביא מקבלת שווה ל 0 תחזיר לי מחרוזת של שלושה קווים אם לא תשאיר את value כמו שהוא

בעיקרון כאן סיימנו את ה pipe אבל אנחנו רוצים להראות אפשרות לשלוח מידע לpipe

* כאשר השתמשנו בעבר ב pipe ראינו שאפשר לשלוח מחרוזת ל pipe ואז לקבל את ה data בפורמט מסויים

לדוגמה – כאשר השתמשנו ב pipe שנקרא currency :

h4>

    {{minPrice | currency:"ILS"}}

</h4>

* ה pipe קיבל מחרוזת (ILS) והחזיר לנו את ה value עם הסמל של שקל

אותו דבר אנחנו רוצים לעשות ב pipe שיצרנו:

1. המחרוזת שאנחנו שולחים נכנסת לפונקציה transform ולכן היא מקבלת אותה כפרמטר נוסף – symbol

public transform(value: number, symbol = "---") {

        return value === 0 ? symbol : value;

    }

* לתוך symbol נגדיר כערך דיפולטיבי את ה string של שלושת הקווים
* בתנאי נשנה את ה string אל symbol

מה שקיבלנו זו הפונקציה הבאה – אם לא הוגדר שום ערך, תציג במקום 0 שלושה קווים, אם כן ניתן ערך (במקרה הזה "חסר במלאי") תציג את הערך

* אפשר גם להכניס מספר ערכים תחת מערך

# Costume Directives

* directive הם פקודות אותם אנחנו שמים על תגית והם משנים את ההתנהגות שלהם (מבחינת css, פונקציונאליות וכו')
* כמו הרעיון של costume pipes אנחנו רוצים לבנות directive שלא קיים
* שימוש ב custom directive נעשה כאשר אנחנו רוצים לבצע שינוי שמצריך קוד js

דוגמה בכיתה – directive שצובע טקסט בצבע רנדומלי

1. ניצור קובץ directive – ng g d directive/"name-of-file"--skip Tests
2. בהתחלה זה מה שאנחנו רואים

@Directive({

    selector: '[appHighlight]'

})

export class HighlightDirective {

}

* אנחנו רואים שהרכיב מוגדר עם annotation שנקראת @Directive
* ה selector נקרא appHighlight (בו נשתמש כדי לקרוא ל directive)

1. אנחנו רוצים להשפיע על צבע הטקסט

* נגדיר משתנה ב constructor שנקרא color

    public color: string;

1. נקרא ל annotation שנקראת @HostBinding – בתוכה נציין על איזה property אנחנו רוצים להשפיע (לעשות binding)

 @HostBinding("style.color")

* כיוון שאנחנו רוצים להשפיע על הצבע נקרא ל style.color
* Host – התגית בה אנחנו שמים את ה directive שלנו

1. נכתוב את הלוגיקה לשינוי הצבע

private getRandomColor(): string {

        const r = Math.floor(Math.random() \* 256);

        const g = Math.floor(Math.random() \* 256);

        const b = Math.floor(Math.random() \* 256);

        const color = `rgb(${r},${g},${b})`;

        return color;

    }

1. נקרא ל ngOnInit ונאתחל את המשתנה

 ngOnInit(): void {

        this.color = this.getRandomColor();

    }

1. שמנו את ה directive בתוך כותרת home

    <h1 appHighlight>Northwind Website</h1>

* אנחנו רואים שכל רפרוש משנה את צבע התגית

**Extra – Udemy Course**

# ngStyle

* מאפשר לעשות מניפולציה על css
* כדי להתשמש בו צריך לבצע property Binding

<div [ngStyle]="{'background-color' : 'red'}" class="row">

* את ngStyle נשים בתוך סוריים מרובעים
* נפתח גרשיים ובתוכם סוגרים מסולסלות "{}"
* ב style נכתב כמו בcss רגיל איך עם גרשיים ''
* אפשר גם להכניס פונקציה לתוך ה {} שתחזיר string וכך לקבל עיצוב דינמי

# ngClass

* מאפשר להכניס class בכל מיני צורות

<element [ngClass]="['first', 'second']"></element>

* Class שניתן כמערך

<element [ngClass]="{'first': true 'third': false}"></element>

* Class שניתנים כאובייקט, אפשר להגדיר true ו false (או לחלופין לתת תנאי שנון true או false) וכך להפוך את ה class למודולרי

# ngFor

<button  \*ngFor="let recipe of recipes"

* אפשר לחלץ את i מתוך הלולאה שיוצרים

app-recipe-item \*ngFor="let recipe of recipes; let i = index">

# viewChild & contentChild()

שתי פונקציות שמאשפרות לנו לגשת לאלמנטים ב DOM

* viewChild() - נותנת ref ל html-Element

@ViewChild(elment, {static: true}) elmennt: ElementRef;

* element מציין templet-reference שנמצא ב dom על ה element אותו אנחנו רוצים לייבא
* ממנו אפשר לגשת ל nativeElment ולקבל גישה ל html element
* contentChild() - נותנת התוכן של html Element

@ContentChild('p', {static: true}) : ElementRef;

# Custom Directives

* אפשרות נוספת שקיימת ב directive היא לקשור את השינוי לאירוע

לדוגמה – directive שפותח וסגור dropdown

1. יוצרים directive – ng g c d directive/dropDown –skipTests

@Directive({

  selector: '[appDropDownDirective]'

})

export class DropDownDirective {

}

1. משתמשים ב @HostBinding כדי לקשור property

@HostBinding('class.open') isOpen: boolean = false

* במקרה הזה ה property שאנחנו רוצים לקשור הוא class open שמשפיע על הסגירה והפתיחה של האלמנט
* המשתנה שהוא יקבל יהיה מסוג boolean

1. כדי לקשור אירוע משתמשים ה @HostListener

@HostListener('click') toggleOpen() {

    this.isOpen = !this.isOpen

  }

* בתוך הסוגריים ניתן את ה event שאנחנו רוצים לקשור (במקרה הזה click)
* נותנים method שתופעל בעקבות האירוע – תשנה את הערך של isOpen

# Reactive Forms - Base

* בניית של form דרך ts
* צריך לייבא את FormReactiveModule במקום FormModule

דוגמה –

1. בדף ts יוצרים משתנה מסוג FormGroup

public profileForm : FormGroup

* FormGroup יוצר לנו אובייקט שייקבץ את כל ה input שנשים בתוכו

1. יוצרים מקרה חדש של FormGroup, האובייקט מקבל string שיצביע על FormControl

  profileForm = new FormGroup({

    firstName: new FormControl(''),

    lastName: new FormControl(''),

  });

* כל FormControl מייצג input שאנחנו רוצים לשים ב form

1. בדף ה hmtl יוצרים form – צריך לקשור אותו ל form שייצרנו ב ts

    <form [formGroup]="myForm" #formDirective="ngForm">

* formGroup - קושר ל FormGroup הראשי

<app-my-input  [(ngModel)]="ingredient.amount" formControlName="amount"

* formControlName – קושר ל control שבתוך formGroup
* אם ה control שלנו לא נמצא בתוך FormGroup נשתמש ב formControl

# Reactive Forms - FormBuilder

צורה יותר נוחה לכתוב את זה היא על ידי שימוש באובייקט שנקרא FormBuilder

1. מבצעים DI לאובייקט

constructor(private fb: FormBuilder,) {}

1. יוצרים form חדש בצורה הבאה

profileForm = this.fb.group({

    firstName: [''],

    lastName: [''],

  })

* כל field בתוך group הוא אוטומטית FormControl אלה אם מגדירים אחרת

אובייקט FormControl מקבל מערך שמכיל שלושה דברים:

* ערך ראשוני ל value – לרוב string ריק או null
* מערך ולידציה ל control (אופציונלי) – צריך לקרוא לאובייקט של ng שנקרא Validators ולבחור ולדיצה
* מערך ולדיצה ל data אה-סינכרוני (אופציונאלי)

  profileForm = this.fb.group({

    firstName: ['', Validators.required],

    lastName: [''],

  })

1. כדי לאתחל את ה form צריך להגדיר פונקציה מסוג get לכל control

  get name() {

    return this.profileForm.get(firstName)

  }

  get amount() {

    return this.profileForm.get(lastName)

  }

# Reactive Forms - FormArray

* דרך ליצור control בצורה דינמית

לדוגמה – במידה ויש לנו form של כתובת, אנחנו לא יודעים כמה כתובות המשתמש ירצה

להכניס ולכן נרצה ליצור את ה form בצורה דינמית

  aliases: this.fb.array([

    this.fb.control('')

  ])

* מייבאים את FormArray לדף הts
* מתוך אובייקט fb קוראים לאובייקט array

נושאים שלא עברתי עליהם

* Async validation
* Custom Validation