**דברים חשובים שלמדתי בקורס שחשבתי שכדאי שאסכם**

**קומפילציה והרצה של אנגולר**

* Webpack : הוא כלי ש angular CLI משתמש בו.

הוא כלי אשר עושה build לקוד בצורה אוטומטית – build automation tool.

כשאנו עושים ng serve הwebpack מתחיל בקימפול שלו.

* + התהליך שלו : הוא לוקח את כל קבצי הjs שיש בתוכנה שלנו , ושם אותם ב-5 bundleים שונים ועושה להם minimize בשביל בצועים.

1. Pollyfills.bundle.js : מכיל קבצי js שמשלימים את הפער בין התמיכה שיש לדפדפן בגרסאת js מסוימת לבין מה שאנגולר צריך (במידה והדפדפן לא תומך בגרסא מספיק מתקדמת.)
2. **Main.bundle.js :** שהוא מכיל את כל קבצי הjs שהם הsource code של האפליקציה שלנו (כל הקומפוננטות , מודלים **, ואפילו כל הtemplates**.) מכניס אותם לבאנדל ועושה מיני פיקציה.
3. Styles.bundle.js – כל קבצי הcss
4. vendor – מכיל את כל הקבצים של ספריות צד שלישי.
   * עוד משו מגניב בwebpack זה שברגע שנשנה את אחד הקבצים הוא אוטומטית יקמפל מחדש וישנה את הbundleים בהתאם. ויעשה refresh אוטומטי לדפדפן.

* כאשר אנו נכנסים לאפליקציית הangular שלנו אנו ניגשים לעמוד index.html.
* **בזמן הרצה** הweb pack מכניס הפניה לbundleים שהוא יוצר לקובץ **index.html** ולכן בזכות זה – הselectorים של הcomponents שלנו מובנים ובכללי כל הframework של אנגולר עובד.
  + **לדוגמא אם נעשה view page source על הדפדפן אנו נראה את הindex html עם הפניות לbundleים של web pack.**

**Decorators**

* בתוך component יש decorator (@component()) שהוא הופך את הts class ל-component.
* הdecorator הזה מקבל אובייקט ובו מוגדר meta data לגבי אותו component:
  + Selector = זה השם של הcustom html tag שהקומפוננט תשתמש בו.
  + Template = הhtml שהcomponent ירנדר.
  + ועוד..
* בקובץ Index html ניתן לראות custom html tag בשם app-root
* זה למעשה הselector של ה-app. Component
* ואז ב-app component אנו נכניס selectorים של componentים שאנחנו יוצרים.
* Injectable decorator אומר שלקלאס הספציפי עליו הוא חל – ניתן להזריק תלויות.
* הcomponent decorator יש מימוש של inject able decorator בפנים לכן אין צורך להשתמש בdecorator הזV בcomponent למרות שכן מזריקים לו תלויות.
* לכן יש שימוש בinjectable decorator רק בserviceים (שיש בהם צורך להכניס תלויות )

Services,directives – בauthors list – וauthers.Service

וך

**Displaying and handling events**

**Property binding**

* תזכורת קלה – html זה שפת Markup בניגוד לDOM שזה אובייקטים ששמורים בזיכרון שמייצגים את הhtml .
  + בhtml לטקסט בתוף הtag קוראים attribute וב-DOM זה property.
* String interpolation מאחורי הקלעים עושה property bindning
* נשתמש בstring interpolation (השימוש ב –{{ field }}) בtagים שקשורים לטקסט.
* אך אם נרצה להביא מידע מהקלאס לattributes כמו נגיד <img src=>
  + נשתמש בproperty binding. כך : <img [src]="field\_name">
  + פשוט מוסיפים סוגריים מרובעים.
* יש לזכור ש property binding זה רק one way data binding מהcomponent class ל-**DOM ולא ל-HTML** ולא הפוך
* יש כמה attrubites של html שאין properties מקבילים אליהם ב-DOM (וגם הפוך)

מה שיכול ליצור בעיה בשימוש בproperty binding כי הוא מקשר ל-DOM.

* אז מה שעושים במקרה כזה זה : < <td [attr.colspan]="field\_name"

ככה אנחנו עושים property binding ל-html attribute ולא לDOM propery.

**הוספת bootstrap לפרויקט**

* עושים Npm install bootstrap –save
* הולכים ל-styles.css וכותבים : Import "~bootstrap/dist/css/bootstrap.css"
  + ה~ = מעביר אותנו לnode\_modules במקרה הזה.

**Class binding**

* במידה ונרצה לתת קלאס מסוים במקרה זה Pull right בהתאם לערך שיש לנו בקומפוננט נוכל להשתמש בclass binding כך :
  + [class.pull-right]="isPulled"
* Style binding אותו סגנון בדיוק – שים לב שניתן גם להשתמש בתנאיים נגיד :
  + (isPulled) ? blue : red style.backgroundColor]=]

Event bindning

* פשוט משתמשים ב – (click=method()).
* אם רוצים להעביר את הevent object : (click=method($event))
  + שהוא מכיל כל מיני מאפיינים על הevent כמו איפה העכבר היה וכאלה.
* Event bubling = אם יהיה לי div ובתוכו כפתור , ובשניהם יהיה event handler (מחכים לevent)
* גם אם אני אלחץ רק על ה-button ה-div גם יקפיץ event וזה לא להיט , בשביל לעצור את זה אפשר לקרוא לפונקציה – stoppropgatation() משו כזה על ה$event בפונקציה של הbutton click

Event filtering

* בעזרת אנגולר ניתן לעשות לדוגמא : <input (onkeyup.enter)="onEnter()"
  + ואז תרוץ המטודה רק כשיילחצו אנטר אחרי keyup

Template variables

* בשביל להעביר קלט מהUi לclass אפשר להשתמש בtemplate variables כך :

<input **#email** (onkeyup.enter)="onEnter(**email.value**)"

וזה יעביר לנו את הערך.

* אבל תכלס תמיד עדיף להשתמש בtwo way binding עם [(ngmodel)] , לא לשכוח לעשות Import בapp.module ל – formModule.

Pipe ,two-way binding וcustom pipes : ניתן לראות : title-casing. Component

**Building reusable components**

**Input properties to components**

* כמו שיש properties ל-dom של html tags , אם אנחנו יוצרים component ואנו רוצים שנוכל להשתמש בה מספר פ]עמים במקומות שונים בקוד שלנו כדי שגם לנו יהיה properties שנוכל לתת להם ערך מבחוץ.
* כמו לדוגמא : <favourites [isFavaourite]='true' – ככה הכוכב איך שנטען אותו יהיה מלא ,
* ואם נרצה לטעון את הקומפוננט הזאת שוב ממקום אחר בקוד כשהוא מתחיל בתור כוכב ריק גם נוכל לעשות זאת.
* הInput גורם לcomponents שלנו להיות יותר reusable.
* בנוסף כדאי שניתן alias לinput properties שלנו כדי שלא יהיו באותו השם של הfield ואז במידה ונשנה את שם הfield הקוד יישבר..

**Output properties**

* שזה תכלס האפשרות להקפיץ eventים לcomponent שאתה יצרת.
* סתכל על favourite component , וapp-component,html – ותראה את הevent שקורא כל פעם שמשתנה כוכב. (כותב לconsole).
* \*שים לב לעשות Importים מהמקומות הנכונים - הauto import לפעמים עושה אימפורטים הזויים ששוברים תקומפילציה.
* שים לב שניתן גם להעביר eventdata במטוד - סתכל על isfavourite component ועל appcomponent.ts + html
* וגם פה ניתן להשתמש בaliasים כמו ב- input properties
* שים לב שבמידה ויוצרים directive ונותנים לו output , הevent שנוצר יכול להיות ממומש על כל tag שמשתמש באותו ה-directive.

**View encapsulation**

* שים לב שכל עיצוב שנכניס לקובץ css של component מסוים יעבוד רק על אותו component , סתכל על favourite component איך הוא צובע בצהוב את ה glyphicon class אבל המעטפה לא נצבעת.
* זה עובד בעזרת אימולציה של אנגולר למימוש של shadow dom (כי לא רוב הדפדפנים תומכים בshadow dom)
  + זה מוגדר באופן דיפולטי בmetadata של הcomponent תחת הproperty : encapsulation : emulated
  + הוא עושה את האימולציה הזאת בעזרת הגדרת attribute יעודי לכל component

**Ng-content**

* מאפשר לי להזריק markup לקומפננטה שיצרתי מחוץ לקומפוננט בעזרת class names שאני מגדיר לדגומא סתכל על component – panel ועל – app.component.html

**Ng-container**

* במידה ואנחנו רוצים להכניס רק טקסט לng-content שלנו , במקום שנשתמש באיזה div פשוט נשתמש ב-ngcontainer ראה app-component.html , וסתכל על הheading .
* אם תעשה לזה Inspect תראה שהוא מקבל רק את הטקסט , וככה הקוד פחות noisy..

**סיכום על הפרק building reusable components ב-like component וapp.**

**ללמוד על change detection mechanisimשרץ ברקע וגורם לtemplates להיות בסנכרון עם הקבצי ts**

**Directives**

* Directiveים אחראים על שינוי הDOM.
  + Attributes directives : משנים properties ב-DOM כמו ngclass , ngstyle כל אלה .
  + Structure directives : אחראים על יצירה \ הסתרה של **אובייקטים בDOM ולא הproperties שלהם , הסוג הזה של directive יתחיל עם \* , לדוגמא :\*ngfor \ \*ngif**

**\*ng if**

* תסתכל על הcourses component – תראה שימוש בngif וng-template.
* שים לב שבמידה וקיימים קורסים יוצר הtemplate עם הtemplate variable – courseList ואם אין קורסים יוצג הtemplate עם הvariable noCourses.
* וכל התנאי הזה ייקרא רק במידה וה-viewmode יהיה courses בהתאם לswitchcase (שהוא משתנה בevent של לחיצה.)
* במילים אחרות במידה והngiif הוא truthy אז הוא יציג את הtemplate במידה והוא falsy הוא לא יציג.
* Turthy ו-falsey זה הערך של הדיקה בוליאנית לדוגמא : if(true) זה truthy בניגוד ל-if(false) שזה falsy.
* **שים לב שה ngif מתייחסים ל –variables ללא ה# בניגוד להצהרה שלהם.**
* שים לב שניתן לעשות property binding על hidden attribute ולהשווות אותו לביטוי בלויאני ואז יהיה כמעט אותו אפקט כמו ngif.
  + ההבדל בין השניים הוא שאם נשתמש בhidden attribute האלמנט שמוסתר עדיין יהיה ב-DOM בניגוד לngif שבו האלמנט לא יעלה בכלל ל-DOM , ולכן באפליקציות שיש בהם DOM גדול , עדיף להשתמש בngif בשביל ביצועים טובים יותר.
* שים לב שחובה להשתמש ב - \* כאשר נשתמש בngif – מה שהכוכבית עושה זה - היא יוצרת ng-templateים של הtagים אשר תלויים בתנאי.

ומוסיפה ng-ifים תואמים לתנאי ההצגה שלהם.

**Ngswitch**

* דוגמא ביחד עם ngif בcourse component

**ngFor**

* יש משתנים שניתן להשתמש בהם ב-ngfor כמו לדוגמא Index ו- even שיודעים להגיד לנו בהתאם לערכים שאנחנו עושים להם iteration – ראה table.component

**Change detection**

* כל פעם שיהיה בקשת ajax \ event \ timer ירוץ מאחורי הקלעים change detection אשר יזהה אם יש חוסר סנכרון בין הערכים בtemplate לבין הערכים בcomponent.ts ויעדכן את השינו – לדוגמא מחיקת שורה ב - table compenent

**ngFor and track by**

* נגיד ואנחנו נקבל את הטבלה שלנו משרת מסוים לפי לחיצת כפתורכמו שעשיתי בtable component.
* במידה ואני לא משתמש בtrack by method ליד ה-ngfor הDom objects שהngfor מייצר יווצר כל פעם מחדש כשאני יילחץ על הכפתור – למרות שהערכים תמיד יהיו זהים (ניתן לבדוק זאת לפי זה שב page source בדפדפן יש צבע סגול על הערכים שנוצרים כל פעם שנלחץ כפתור).
  + **הסיבה שהוא נוצר כל פעם מחדש היא שהngfor** מזהה את האובייקטים לפי ה-identity שלהם – שזה למעשה ה-כתובת שלהם בזכרון.
  + ובגלל שכל פעם שאנחנו לוחצים על הכפתור אנחנו יוצרים את המערך מחדש – אז הכתובת של המערך בזכרון כל פעם תיהיה שונה ולכן הngfor לא מודע בכלל שתוכן האלמנטים זהה והוא יוצר אותם מחדש כל פעם.
* לכן ניתן להשתמש ב-trackby לפי ערך של הdata.name במקרה של ה-component שלי , ואז הוא לא ירנדר את האלמנט מחדש כי הוא ייראה שהוא כבר קיים.
* **שים לב לא להשתמש בזה סתם !! רק אם באמת יש בעיות ביצועים באתר שלנו נשתמש בtrackby כי זה סתם מבלגן תקוד - preoptimization is the roots of all evuil**

**Ngclass**

* הוא attribute directive – תסתכל על favourite component על ה-heart.
* תשווה את זה לclass binding שיש לכוכב.
* זה יותר נוח
* הngclass מקבל אובייקט שבו יש key value pairs.
  + הkey הוא שם של css class – **שים לב לשים גרשיים סביבו** והvalue הוא ערך בוליאני שלפיו לelement יהיה באותו ה-class.

**Ngstyle**

* בדיוק אבל בדיוק אותו הגיון כמו ngclass רק ל-styleים.

**Safe traversal Operator – או במילים אחרות סימן שאלה(null able operator) בstring interpolation**

* תסתכל על הקומפוננט authors list
* במידה ותלך לauthor.service ותחזיר במערך Null (יש שם comment)
* **תופיע שגיאה בדפדפן כי הוא לא יכול להציג author.name כאשר author הוא Null**
* **אך אם נוסיף ? ליד author כמו שעשיתי בauthor list.html לא תיזרק שגיאה ופשוט יופיע לנו Null על הדפדפן.**
* **אנו נשתמש בזה כי לפעמים יש מצבים שאנחנו מביאים מידע מהשרת ולאיזה מאית שנייה האובייקטים המורכבים שאנחנו מייבאים הם Null – ולכן פשוט נוסיף את ה- Safe traversal Operator כדי למנוע זריקת שגיאה בruntime ואז כבר האובייקט לא יהיה null.**

**Custom directives**

* תסתכל על input-format.directive.
  + **Hostlistner – מאפשר לי לעשות event handeling על הdom object עליו חל הdirective.**
  + **ElementRef הוא service שנותן לי את גישה ל-Dom Object.**
  + **בשביל לתת אפשרות להכניס Input ל – directive פשוט מכניסים בInput alias את שם הdirective.**

**תרגיל מסכם directives ב-zippy component – (תסתכל גם על app.html)**

**Template Driven Forms**

* FormControl class
  + ניתן להצמיד לכל Input field בformים שלנו Inststance של –FormControl.
  + וכך בעזרת הInstance הזה נוכל לבדוק מה הערך בנמצא בInput field , האם עשו focus על הInput field , האם הערך שונה בעבר ועוד מלא אפשרויות validation.
* FormGroup class
  + מייצגת קבוצה של -control ים ב-form.
  + יש לאובייקט הזה את אותם הproperties (dirty, ועד מלא אפשרויות validation).
  + הformgroup יהיה valid רק אם כל הcontrollים שלו יהיו valid..
* יצירת controls
  + ישנם שני דרכים ליצור את אובייקטי הcontrol האלו.
    - Directives – template driven forms
      * טוב ל- formים פשוטים
      * Validation פשוט.
      * יותר קל לכתוב אותם.
    - Code - reactive forms
      * יתרון – יותר שליטה על לוגיקה של validation
      * אפשר לעשות על זה testים.
* יצירת template driven forms
  + נשתמש בdirective ngModel ללא two way binding syntax
    - והוא ייצור מאחורי הקלעים control object בשביל אותו Input
  + נצטרך לתת שם ל-input field כדי שהקונטרול יידע לזהות את אותו ה-input field.
  + לדגומא תסתכל על inputform.html
  + ה-ngModel משתמש הproperties של form control ומעתיק אותם אליו לצורך גישה מהירה יותר לproperties שלו – זה למה אני לא עושה firstname.control.valid אלא ישר ניגש למאפיין.
* יצירת validations
  + תסתכל על input forms.
  + שים לב שיש צורך בשימוש של template variables בשביל לשמור את האובייקט של הngModel למשתנה שנוכל לגשת אליו בmarkup הבא.
* שימוש בכמה validations והצגת שגיאת validation בהתאם לבעיה.
  + תסתכל על input forms.
  + שימוש ב-error object.
  + שים לב שבעת שינוי לInput field עשיתי שהוא גם ידפיס את הfirstNAme כדי שאוכל לראות את שמות ה-properties של אובייקטי השגיאות.
  + שים לב שיש מאפיין נוסף לכל שגיאה כמו לדוגמא ב-minLength – requiredLength
  + שימוש במאפיינים האלה מונעים מאיתנו לעשות hardcode ב-html שעלול להשתנות בעתיד.
* Styling invalid input fields
  + תסתכל אם אתה עושה Inspect על הinput field בזמן שהוא Invalid - אנגולר מוסיף class - ng-invalid , וגם לעוד כל מיני דברים כמו ng-touched (אם שיניתי שם משו) ועוד..
  + ניתן להיעזר בהוספת class הזה לעיצוב הInput fields לדגומא תסתכל על styles.css וה- input-form,
* Cleaner templates – שים לב שהtemplates שלך לא דורשים גלגול ימינה בגלל כל הattributes – לכן עדיף להפריד בשורות למרות שזה אותו אלמנט.
* ngForm
  + אם נשים את הinputים שלנו תחת form – בצורה אוטומטית מאחורי הקלעים לאותו ה-form יהיה directive בשם ngForm – שהוא מייצג אובייקט של formGroup (כמו שמוסבר למעלה)
  + שים לב של-directive הזה מוגדר גם output (שזה בעצם אומר שכל מי שיש לו את הדיירקטיב הזה יש event שתואם את הדיירקטיב) לדוגמא במקרה זה הoutput הוא ngSubmit – אז תכלס זה פשוט event שכל tag שמשתמש בdirective ng model יכול לממש.
    - הסבר חוזר על outputs אם לא הבנתי למעלה ויצירת directive לדגומא :favourite component.
  + ניתן לראות אותו בInput-form.component.html אחרי שנלחץ על submit
* ngModelGroup directive
  + הcontrol instaceים שאני אכניס לתוך div שמשתמש בdirective ngModelGroup ייגרום לכך שב-value property שב- FormGroupיהיה אובייקט JSON שמכיל את הערכים של הinputים בפנים.
  + לדגומא תסתכל על – input-form.component : תלחץ submit ותסתכל על ה-value של האובקייט ngForm – ותראה שם אובייקט בשם contact (כי זה השם שהבאתי לngModelGroup directive) והוא יכיל את הvalues של הInputים.
  + אפשר (לא חובה בכלל) נשתמש בזה אם נגיד בform מסוים יש לנו קבוצות של Inputים שקשורים אחד לשני , נגיד mail ופלאפון ושם אפשר ליצור לו modelGroup בשם contact info.
  + זה יכול לתרום במידה ואנו רוצים לשלוח בpost לאיזה webapi jsonים ולעשות עליהם איזה logic בצד השרת.
* תכלס ההבדל בין ngModel ל-ngform זה שלngform יש את ה-ngSubmit event.
* שים לב-disable שלה-submit button.
* Validating check boxes

שים לב שזה עובד אותו דבר בcheckbox

* + ותראה טריק נחמד ה-p בסוף תמיד יראה לי את ה-values שיש ב-ngForm
  + **זה בעזרת הpipe json.**
* Drop down lists
  + שים לב שגם פה זה עובד תכלס אותו הדבר..
* ngValue
  + במידה והייתי שם attribute של ngValue על ה-option ככה: ngValue] = method]
  + הערך היה כל האובייקט ולא רק id או string..
* סיכום לפרק בcreateCourse component – שים לב לngsubmit שאני מדפיס אובייקט שמכיל את כל הערכים.

**Consuming HTTP Services**

Getting data

* תסתכל בpost component איך אני שולף מידע מ-api מסוים.
  + שים לב שאני משתמש ב ב class httpClient וב-app.module אני עושה Import ל –httpClientModule
  + שים לב שmosh מלמד להשתמש ב-http אבל זה ניהיה deprecated אז צריך להשתמש ב-httpClientModule.
  + **שים לב שה-subscirbe אני חייב להוסיף את ה<any[]> בשביל שאני יוכל לתת לposts את ה-ערך.**

Create data

* תסתכל post component
  + דבר ראשון שים לב לevent שנתתי לinput tag ואיך העברתי את הtemplate variable בשביל לגשת לערך שלו.
  + **שים לב ל-(keyup.enter) זה מאד שמיש וחשוב לזכור**
  + תסתכל על הקובץ ts איך מימשתי את הפונקציה של ה-create
  + הבאתי HTMLInputElementלפארמטר.
* עכשיו שים לב שהוספתי למערך של ה-posts את ה-post החדש.

למה עשיתי זאת ?

* תכלס אם זה post , והget שלי הוא observerable הוא אמור לקלוט תשינוי ולבד להציג לי את המידע ברשימה. **(לדעתי)**
* הסיבה שהוספנו ידנית היא : שהapi הזה שאנו משתמשים בו הוא לא אמיתי, הוא לא באמת מעדכן איזה DB ולכן אנו צריכים להוסיף בעצמנו כאילו נוסף ל-DB (אם נעשה refresh זה לא ייראה את ה-postים שלנו.)

Updating data

* תסתכל על ה-post component
  + ישנם שני אפשרויות לעדכן את המידע – בPut או ב-patch
  + Patch מעדכן רק כמה properties מסוים באובייקט בניגוד לput שמקבל אובייקט שלם שהוא הופך למעודכן. (ה-patch ב-comment)
  + שים לב שמצד השרת צריך לתמוך בrouteים המתאימים.
  + שים לב שבupdateים הנתיב יותר אקסקלוסיבי לדוגמא פה /posts/posted ולא פשוט /post – ככה יודעים איזה post לעדכן.

Deleting Data

* תסתכל על ה-post component
  + שים לב שגם פה הגישה היא לurl אקסקלוסיבי לאובייקט שאותו מוחקים.
  + הפונקציה לא מקבל object.

OnInitInterface

* באנגולר ל-component יש life cycle hooks לדוגמא :
  + יצירת component
  + רינדור של component
  + יצירת component ורינדור הילדים שלה
  + הריסת component
* לכל אחד מה life cycles events האלה יש interface שאם נממש אותו יהיה פונקציה שתרוץ בזמן אותו event.
* לדגומא ngOnInit רץ באתחול של ה-component.
* לכן כאשר נשתמש בhttp services נעדיף לשים אותם ב-ng on init interface ולא ב-constructor שאמור להיות קל ופשוט.

Separation of Concerns

* כרגע כתבתי את כל הקוד של הגישה לשרת ב-posts.component.
* מה ששובר את עקרון הseperation of concerns ב-OOP.
* משמעות העקרון בoop היא שלכל class יש concern אחד משלו.
  + כמו שלדוגמא במסעדה יש שף שאחראי על בישול ומנקה שאחרי על לנקות , לclass בקוד שלנו אמור להיות אחראיות אחת.
  + Class שיהיה לו יותר מידי concerns **הוא פחות maintainable וגם פחות testable.**
* לדגומא בPost.component שלנו יש שני concerns – גישה ל-web api והצגת המידע בview (נגיד מחיקת אובייקט מה-li וגם גישה ל-web api)
  + דוגמא לסיבה שהוא פחות maintainable :
    - נגיד ועכשיו נוסיף עוד component שגם זקוק לגישה ל-web api של –posts , נצטרך לכתוב שם שוב את כל הקוד הזה , נצטרך להגדיר שוב את הurl , בניגוד לאם נשתמש בservice שנממש אותו בשני הcomponents השונים.
  + דוגמא לסיבה שהוא פחות testable :
    - נגיד במידה ונרצה לעשות testים ל-update function שיש לנו ב-post component בtest שלנו תיהיה גישה ל-webapi מה שממש יאט את תהליך ה-test ולא נוכל להריץ מאות testים בשנייה.
    - בנוסף בשביל לבדוק שההתמודדות של הui עם עדכון אובייקט בposts טובה – ניהיה חייבים שגם הגישה לweb api טובה כי זה באותו הפונקציה.
    - בניגוד לאם נשתמש בservice שיהיה אחראי על הגישה ל-web api ,

כעת נוכל לבחון את הפונקציה update ולהיעזר באיזשהו fake service שיעשה כאילו הוא ניגש לweb api ככה נוכל לבדוק רק את הUI logic.

* לכן עדיף שניצור service שה-concern שלו יהיה גישה להweb api והpost יהיה אחראי על הLogic של הצגת המידע.
* תסתכל על post.service

Handling Errors

* ישנם שני סוגים של שגיאות - צפויות ולא צפויות
  + unexpected
    - server is offline
    - network is offline
    - unhandled exceptions - bug
  + expected
    - Not found (404)
    - Bad request (400)– לדוגמא לעשות register עם יוזר קיים.
* בסרטונים הראשונים מוש הראה לנו מימוש של טיפול בשגיאות בcomponent של ה-postים – לתפוס בsubscribe אובייקט מסוג response ואז לבדוק איזה status של שגיאה יש לו ולהציג בהתאם
  + אך לאחר מכן הסביר שזה חריגה של separation of concerns בגלל שהcomponent שלנו לא אמור להתחיל לבדוק סוגי שגיאות , זה יותר מתאים לאחראיות של ה- service שלנו.
* מה שאומר שצריך לתפוס את השגיאות מהservice , שזה אומר שצריך להחזיר אותם ב-observerable.
* תסתכל על post service.
  + שים לב שיש שימוש ב-catch operator (לו חייבים לעשות Import explicitly מספריית ה-RXJS)
  + שים לב שאני משתמש ב-catch על פונקצית Http שאמורה להחזיר observerable - ולכן אני חייב להחזיר שגיאה ב- observerable בעזרת שימוש בoperator - throw (שגם אותו אני חייב לעשות Import explicitly) אם לא תיהיה השגיאה הבאה : \_WEBPACK\_IMPORTED\_MODULE\_2\_rxjs\_Observable\_\_.a.throw is not a function
* עכשיו בשביל לממש את הלוגיקה של הui התואמת לסוג השגיאה, יש קודם ליצור סוגי שגיאות ב-domain model שלנו , שהcomponent שלנו יוכל להבין , לכן אני מחזיר observerable של שגיאות custom שלנו.
* כך הבדיקה בcomponent היא לפי ה-status של ה-httpResponseError אלא לפי domain-model errors מה שהרבה יותר הגיוני שיהיה מבחינת האחראיות של הcomponent.
* לסיכום :
* **טיפול של שגיאות של בקשות Http הוא אחראיות ה-service**
* **מימוש לוגיקת UI בהתאם לשגיאות הללו הוא אחראיות הcomponent.ts**
* **בעזרת domain model errors נוכל לגשת לסוג השגיאה ולממש לוגיקה תואמת בcomponent.ts.**
* **\*תזכור אם יש שגיאה כזאת :** \_WEBPACK\_IMPORTED\_MODULE\_2\_rxjs\_Observable\_\_.a.throw is not a function

**צריך לעשות Import explicitly.**

Global Error handling

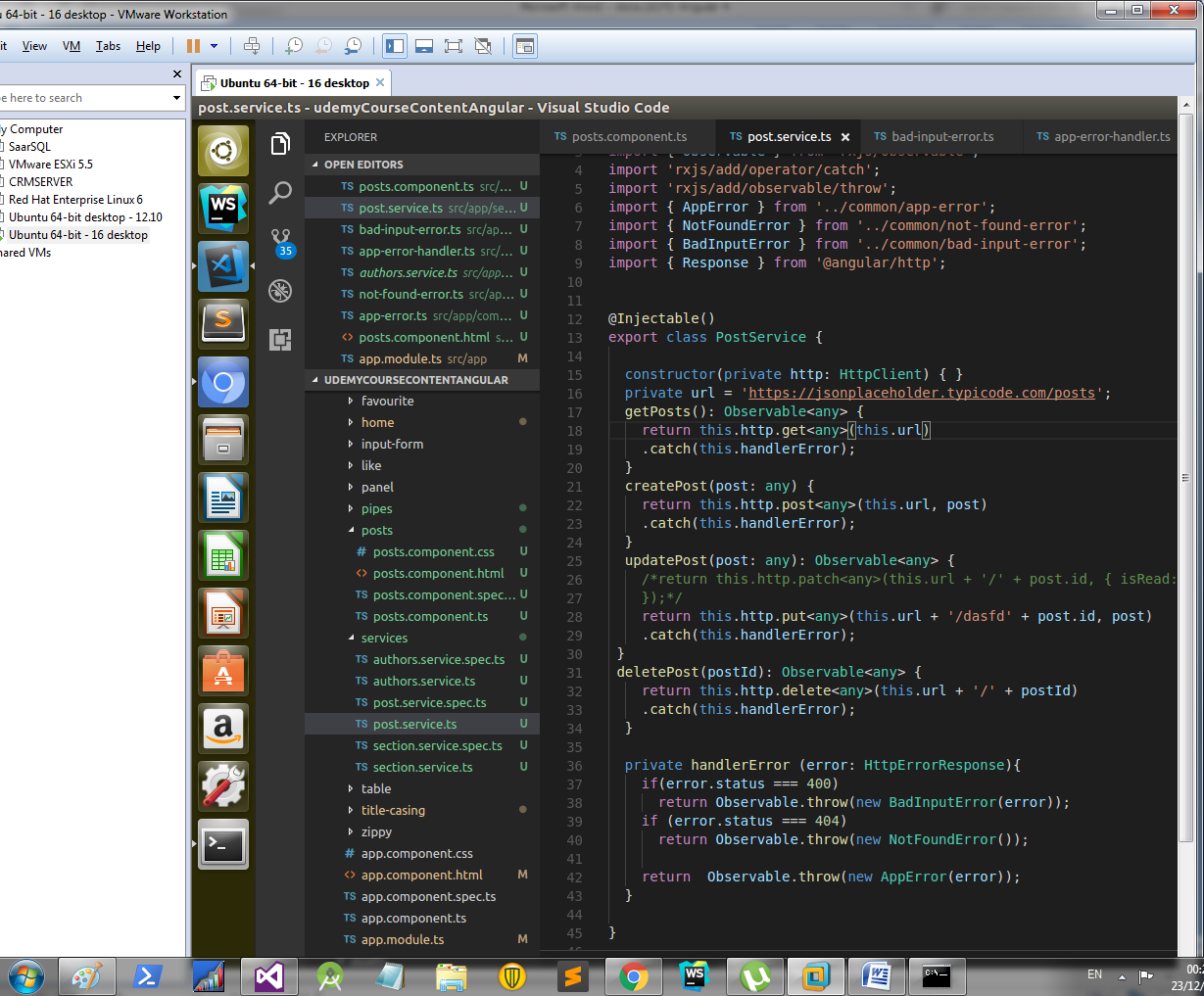
* דבר ראשון יש להכיר את הclass Error Handler הclass הזה נמצא ב@angular/core , וכל שגיאה שנזרקת באפליקציה שלנו תיהיה Handled בעזרת ה-class הזה.

לclass הזה יש פונקציה שמקבל את השגיאה , ופשוט מדפיסה אותה ל-console.

* במידה ואנו רוצים לטפל בשגיאות שלנו בצורה שונה , אנו יכולים ליצור class משלנו שיוריש מה-error Handler class ובמימוש שלנו של אותו הפונקציה נוכל לעשות דברים נוספים (כמו לדגומא לזרוק את השגיאה ב-alert).
* בשביל לגרום לזה שהclass handler שאנו יצרנו יחליף את הhandler הדיפולטי של אנגולר , צריך ללכת ל-app.module ב-providers ולהוסיף אובייקט **כמו בapp.module אצלי בקוד.**
* כעת אותו הclass שיצרנו הוא זה שיטפל בכל השגיאות שיקפצו לנו.
* **שים לב** שיכול להיות שאנחנו עדיין רוצים לעשות handle שונה בהתאם לשגיאות שונות בכל component. **– תסתכל על post.component**
  + לדגומא במידה ותקפוץ לי שגיאת Not found על Post מסוים אני ארצה להציג alert שכתוב בו – post already been deleted.
  + אבל אם נגיד אני גם מרנדר תמונות מאיזה api , ואני רוצה למחוק תמונה אני ארצה להציג alert שקוראים לו image has already been deleted.
  + לכן עשית שב-error handler class שיצרתי – אטפל בשגיאות גנריות, ובכל component שעלולים לצוץ בו שגיאות יותר ספציפיות , אצור private handleError method שתטפל באותם שגיאות.
  + אם לא אעשה זאת – ואממש הכל ב-error handler תדמיין שיש לך 1000 סוגים שונים של שגיאות – כל פעם שתקפוץ שגיאה היא תלך ל-errorHandler ושם יהיה איזה מליון if ו-elseים כדי לזהות את סוג השגיאה ולהציג את הhandle המתאים לאותו שגיאה.
  + **לא להתבלבל הhandle של השגיאות האלה הוא כן אחראיות של ה-component כי זה בסופו של דבר לבחור איזה UI logic לעשות בהתאם לשגיאה – זה לא אחראיות הservice , ולכן ה-handler הזה הופך את הקוד לטוב יותר.**
* תסתכל על posts.service
  + גם לו כדאי שיהיה handler כדי שייזרוק את הdomain-model errors המתאימות – ושלא יהיה code duplication בכל catch operator.
  + שים לב שכשנשתמש ב-handler עם catch operator נעביר הפנייה לפונקציה ולא Invocation.

**Exctracting a reusable Data Service**

* תסתכל איך הpost.service נראה כרגע :

****

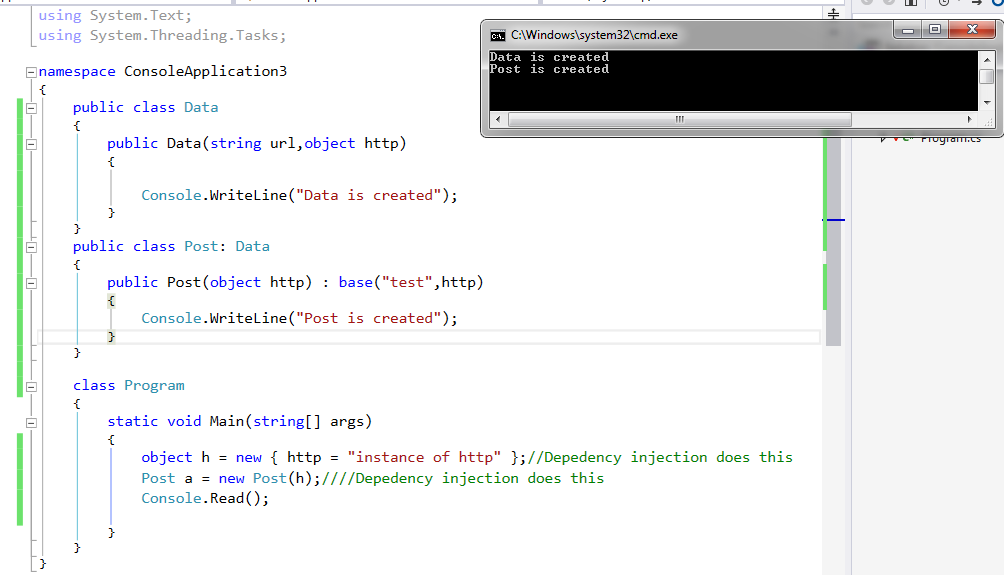
* הבעיה פה היא : שאם נרצה עכשיו להוסיף עוד service שלוקח data מ-api אחר **או עוד 10 serviceים** ?
  + מה גם בשבילם נממש את כל הפונקציות של ה-crud האלה?
  + יהיה **מלא** **duplication** of code.
  + וגם בhandling של השגיאות יהיה דופליקציות
* לכן עדיף שניצור את ה-data service – תסתכל על data service.
  + שים לב שכרגע ה-posts service מוריש מה-data service מה שאומר :
    - שכל ה – methods / properties /fields שיש לאבא , יהיה גם לילד.
    - ובנוסף ניתן להוסיף תכונות לילד.
    - שים לב שאי אפשר לדרוס תכונות מסוימות – לדגומא אני לא יוכל ליצור עכשיו create method ב-posts service , אלא רק להוסיף דברים חדשים – אלא אם כן יש במקרה של c# virtual – קיצר בשביל לממש מחדש field מסוים זה תלוי באיך הוא מוגדר ב-אבא.
  + **שים לב קצת סדר במה שקורה פה :**

1. אני קורא ל-post.service ב-constructor של – post.component.ts – בזכות העובדה שהpost.service נמצא ב-providers array שב-app.module נוצר Instance של - post.service.
2. בעת היווצרות הInstance של post.service הוא יגיע ל-constructor שלו (ככה נוצר Instance) – וב-constructor שלו – מוגדר משתנה http עכשיו גם לו יווצר instance

כעת נכנסנו ל-constructor ורצה הפונקציה super – שהיא יוצרת instance של הbase class (משמע מריצה את ה-constructor של ה-base class)– במקרה הזה – ה-super יוצר Instance של ה-data service עם הInstance של הhttp והurl שמוגדר כפרמטר.

* + במידה ולא הייתי קורא לפונקציה super יש שגיאות קומפילציה – מכיוון שלא ניתן ליצור instance של ה-post service מבלי ליצור instance של אבא שלו.
  + **שים לב**  שב-Inheritance **כשיוצרים Instance של ה-derived type חייבים ליצור קודם Instance של ה-base type.**
    - תחשוב על זה ככה – אם לא ניצור Instance בזיכרון של כל הfields שיש לאבא , איך הילד יוכל להשתמש בהם ? האבא חייב להיות Instantiated קודם.
  + לכן חייבים לקרוא לפונקציה super ב-constructor של הילד – שים לב גם שלפני השימוש בthis אם לא בא super יהיה שגיאה על ה-this.
  + \*בעקבות לעובדה שאני לא שם בשום constructor בקוד שלי dependency של –data Service – הdependency injection לא יוצר instance של אותו ה-class,

לכן גם אין באמת צורך להוסיף את הclass הזה ל-providers array.

* שים לב שככה זה גם ב-c# לדגומא : ****
* **אני חייב לקרוא להרצה של הbase constructor אחרת יש שגיאות קומפילציה.**

**Map operator**

אופרטור של RXJS – אשר עושה מיפוי בין ה-respons ל-datatype אחר – בדרך כלל זה:

this.http.get<any>(this.url).**map(response ) => {respone = respone.json()}**

ואז כשעושים subscribe ב-component מקבלים את אובייקט ה-json ולא את ה-response ואז זה better seperation of concerns

אבל תכלס עם האובייקט החדש של – httpClient הMap הזה קורא באופן אוטומטי – ה-response שמקבלים הוא מייד json.

**Optimistic vs pessimistic updates**

* שני גישות למימוש create -delete
* הראשונה קודם לראות שנוסף הdata ל –שרת – ואז להוסיף ל-ui – שיטה פסימית
* השנייה קודם להוסיף ל-ui ואז להוסיף לשרת – במידה ויהיה שגיאה בשרת , נוריד את הרשימה מהui (זה ייקרה ממש מהר ולא נשים לב) – השיטה האופטימית
  + היתרון של השיטה הזאת היא שהui יהיה טיפה יותר מהיר וחלק.
* דוגמא למימוש השיטה האופטימית בפונקציות create ו-delete ב-posts.component.ts.

**Observable vs. promise**

* הגדרה יפה שאני אוהב לpromise – אובייקט שנותן לי את האפשרות להריץ פעולה בצורה אסינכרונית ולהשתמש בפלט שלה בצורה סינכרונית. – לדוגמא להריץ שאילתה ל-db בצורה א-סינכרונית ולטפל בפלט של השאילתה ברגע שהיא חוזרת עם then function.
* הגדרה יפה ל-observable – אובייקט שנותן לי את האפשרות להריץ **stream של פעולות**  בצורה אסינכרונית ולהשתמש בפלט שלהן בצורה סינכרונית .
* **Observables are lazy - promises are eager**
  + הוא ירוץ רק מתי שנעשה לו subscribe.
  + לדוגמא אם אני עכשיו מוחק את ה-subscribe מה-delete method ב-post.component – אריץ את הקוד- ואכנס ל-network tab – לא אראה שום ניסיון גישה לשרת.
  + לכן גם ניתן לשרשר כל כך הרבה אופרטורים לobservables כמו לדוגמא Map ו-catch וretry וכל מיני בלי בעיות – כי הכל ירוץ רק אחרי subscribe.
* סתם טוב לדעת שיש operator ב-rxjs שהופך Observable ל-promise – שאם נגיד נעשה את הdelete שיחזיר promise גם אם לא נשתמש ב-then function אנחנו כן נראה נסיון גישה לשרת – משמע promise הוא Eager ולא Lazy כמו Observable.
* דבר שmosh לא אמר והוא מאד חשוב זה – promise הוא פיסת מידע אחת שרצה בצורה אה סינכרונית – וברגע שהיא מסתיימת ניתן להשתמש במידע הזה בצורה סינכרונית עם הפונקציה then.
* אבל Observable הוא stream של מידע- משמע לא רק פיסת מידע אחת – לא משנה כמה בקשות אה סינכרוניות יהיו – ה-subcscribe ירוץ אחרי כל אחת מהן , בניגוד לpromise שאחרי שירוץ then פעם אחת – זה לא ייקרה יותר.

**Router and Navigation**

* כדי להשתמש ב-Routing חובה לעשות Import ל -router Module
  + בmodule הזה יש directiveים ו-serviceים שנוכל להשתמש בהם כדי לממש navigation באתר שלנו.
* ישנם מספר שלבים ביצירת routeים

1. לקנפג את הrouteים **– המיפוי בין נתיב לבין -component**
2. להוסיף router outlet **–הוספה של איפה יעלה הcomponent ב-markup**
3. להוסיף לינקים **– לחיצה על כפתורים שתוביל ל-path מסוים שיוביל ל-component.**

* בשביל לקנפג routeים ניתן לראות ב-פרוייקט hello-world (נמצא תחת תקיית downloads או dev\_project) תכנס ל-app.module
  + שים לב שעשיתי import ל –routerModule ובdecorator של ה-appModule – בimports הוספתי routerModule.forRoot() שם מכניסים מערך שמקבל אובייקטים (שלכל אחד מהם יש נתיב וcomponent) – שם מתבצע המיפוי בין הpath ל-component
  + שים לב שלסדר של האובייקטים במערך יש חשיבות רבה – לדוגמא אם הייתי שם את /followers ראשון , בחיים לא הייתי יכול להגיע ל-component של profile.
* לאחר מכן חייבים להוסיף את ה-router outlet – ניתן לראות אצלי ב-app.html.
* שים לב שהcomponent שתואמת ל-route תכנס tag אחד אחרי הrouter outlet ולא בתוכו

(יכול לבדוק עם inspect).

* **Router links**
* ב-routerModule קיים directive בשם – routerlink.
  + ולו ניתן ערך של הpathים שיצרנו להם מיפוי ל-component.
  + **לא לתת נתיב עם href!**
* הסיבה שלא ניתן נתיב עם href אלא אם routerLink היא :
  + אם נשתמש ב-href כל פעם שניגש לנתיב כל האפליקציה שלנו תרד מחדש , אפשר לראות זאת ב-network tab אם נעשה פילטר ל-js scripts – נראה שכל פעם שאנחנו נלחץ על לינק מסוים (שמשתמש בhref) יורדים לנו הסקריפטים הדיפולטים של אנגולר שהwebpack יוצר באופן אוטומטי – (שהם מייצגים את כל הקוד של האפליקציה שלנו.
* בניגוד לאם נשתמש ב-routerLink **ככה ירד לנו רק התוכן של אותה ה-component.**
* **דוגמא פשוט לשימוש בrouter link - navbar.component.html**
* שים לב שיש פעמים בהם נרצה להוביל לנתיב דינאמי – **כמו לדוגמא ב-followers.component**
  + **שים לב שאני עושה property binding על ה-routerLink ואני נותן לו מערך שהתא הראשון שלו זה הנתיב.**
  + **התא השני של המערך של הפארמטר שיבוא בנוסף לנתיב**
  + **במקרה הזה יש אובייקט follower לאותה component (הוא לא נמצא כfield בTS כנראה שזה משו שקשור לservice שמביא אותו בצורה מסוימת שאני לא מכיר.)**
* **routerLinkActive directive**
  + directive שמוגדר בrouterModule.
  + תסתכל על navbar.component.html
  + מה שקורה זה שאנחנו נותנים לdirective הזה classes של css (במקרה הזה bootstrap) במידה והroute שכרגע פעיל זה זה , אז הוא יגדיר את ה-classes האלה על אותו li.
* **Getting the Route parameters**
  + תסתכל על github.profile.component
  + שים לב שעושה Import ל-activated route
  + יש אובייקט שנקרא paramMap שהוא observable של route parameters
  + לאובייקט param יש key property שם נמצא השם של הנתיב
  + ו-עוד מאפיין בשם param שבו יש את הid וה-value שלו.
  + ניתן להשתמש בפונקציה get כדי לקבל את הערך  **(נותנים לו את שם הparameter).**
* **מה זה observable**
  + בפשוטת זהו stream של 0 או יותר eventים.
  + בעזרת הפונקציה subscribe נעשה callback ברגע שevent מסוים יעשה emit.
  + לדגומא אם יש לי כפתור שאני עוטף אותו ב-observable **כל פעם שאלחץ על הכפתור ייכנס event ל-stream ובעזרת ה-subscribe method אני יוכל לגשת למידע שהevent מקפיץ.**
* **כעת נסביר על Lifecycle של component**
  + איך ש-component נטען יש את הon init
  + וכשעוברים לcomponent אחר / יוצאים מהאתר יש on destroy
  + ניתן לכתוב קוד שיקרה בדיוק בזמן האלה בעזרת מימוש הinterfaceים onInit ו-onDestroy

**Why route parameters are observable ?**

* עכשיו השאלה היא למה ה-paramMap מחזיר את הפארמטרים של הcomponent בObservable ולא בצורה רגילה ?
* בוא נראה אפשרות ובה הוא לא observable :
  + תסתכל על profile.Component.ts על שימוש בsnapshot property.
  + הוא אינו observable ואכן כאשר אני עובר מן הfollowers component ל –profile של אחד המשתמשים מודפס לי הid שלו – משמע הגישה ל-route parameters מוצלחת.
  + **למה זה עובד ?**
    - בגלל שכל פעם שאני עובר מה-followers ל-profile אני מגיע ל-ngOnInit ושם אני ניגש למאפיין של הsnapshot ומדפיס אותו.
  + **מתי זה לא עובד?**
    - **כאשר אני לוחץ על הכפתור next , אני בכלל לא ניגש ל – ngOnInit ולכן זה לא מדפיס**
  + **כאשר אני משתמש בobservable** 
    - **נכון , גם במקרה הזה הngOnInit לא מופעל , אבל בגלל שobservable מתפקד כ-stream של מידע ,שכל הזמן רץ ברקע וברגע שיוחלף הפרמטר (שזה תכלס הevent של אותו observable) ה-subscribe method יקלוט את זה וירוץ.**

**Routes with multiple parameters**

* תסתכל על זה שהוספתי עוד פארמטר של login ל-followers , פשוט מוסיפים /:param\_name וככה עושים עם כמה פארטמרטים.

**Query parameters**

* הם פרמטרים **אופציונאלים** שאנחנו מוסיפים אחרי ?
* לדוגמא אם אלך ל-facebook ואוסיף ? והרבה lachtongים זה עדיין יוביל אותי ל-facebook הרגיל.
* אבל עדיין אפשר להיעזר בparameterים האלה כי ניתן לגשת אליהם מה-.ts ולעשות איתם Logic מסוים.
* תסתכל בnvabar component.htmll איך אני מוסיף שני query parameters ל-followers - ככה שכל פעם שאני לוחץ על הfollowers ב-navbar זה מוביל אותי ל-followers עם פרמטרים אופציונאלים האלה.
* ותסתכל ב-followers.ts איך אני ניגש ומדפיס אותם – כמעט אותו דבר כמו פרמטרים רגילים. (גם פה יש אפשרות לעשות עם snapshot).

**Subscribing to multiple observables**

* כעת יש לנו subscribe לparamMap observable שבאמצעותו ניגש ל-id parameter.
  + **תכלס מה שבד"כ נעשה אחרי שנשיג את ה-id זה : נשתמש בservice שלנו כדי לגשת לweb api שייתן לנו profile עם id שתואם ל-required parameter שלנו.**
  + **מה שאומר שאנחנו נקרא לפונקציה של הservice בתוך הsubscribe שמקבל את ה-parameter.(אחרת לא נוכל לשלוף את הprofile בהתאם ל-id)**
* אבל אם יש לנו גם query parameter שאנחנו צריכים להתחשב בו(לדוגמא Page) כשאנו ניגשים ל-web api **איך נעשה זאת ?**
* **איפה נשים את הsubscribe של ה-service כך שהוא יוכל להשתמש בשני הפארמטרים ?**
* בשביל זה נאחד את הobserverables שלנו (paramMap ו -queryParamMap) ל- observable אחד.
  + ומשם נוכל לגשת לשני הפארמטרים שלנו ולקרוא לservice בפנים.
* אפשר לראות דוגמא ב-followers.ts אבל שים לב שאין לו באמת שימוש בid והservice לא באמת דורש את שני הפארמטרים – זה סתם בשביל לראות איך מחברים בין שני observerables.

**Programmatic navigation**

* תסתכל על github profile component .ts ו-html
* בליחצה על submit אני עובר עמוד – המימוש הוא בקובץ ts ולא בhtml כמו מקודם עם הdirective של ה-routerLink.
* וגם אני מוסיף quryPrameters.
* במידה והייתי רוצה להוסיף required parameters אז פשוט בתא הבא של המערך שבו אני מגדיר את ה-path.

**תרגול של הsection הזה ב-router-nav-proj תחת dev\_projects**