

## SQL Learn:

2 - 42

### Angular JS

- כתיבת האפליקציה
- MVC**
- Directives**
- => directives האזנה ותגובה**
- סיכון קצר
- Ajax**
- Spa**
- { Custom } Directives**
- שיתוף מידע**
- פילטרים
- טפסים
- שאלות ראיון

43 - 69

### Angular2 Intro

- מבוא
- קומפוננטות מקוננוות
- קומפוננטות - קלט ופלט
- [ Attribute ] Directive**
- קשרירה**
- קשרירה דו > = < כיוונית**
- Local Reference**
- Observable**

70 - 120

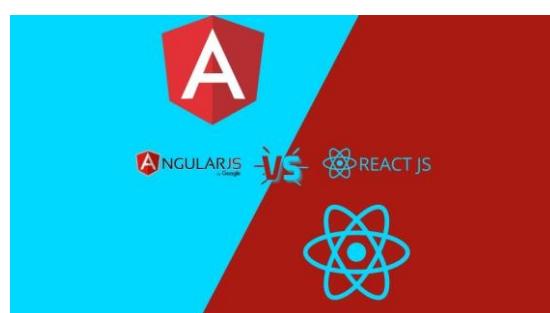
### Angular2 Tools

- SPA**
- מניעת גישה
- Pipe**
- פיתוח טפסים

121 - 150

### Angular2 API

- API**
- http - Client: Get**
- http - Client: Post**
- http - Client: Put**
- http - Client: Delete**





# Angular JS

## כתיבת האפליקציה :

### 2. מגדירים את האלמנט העוטף אפליקציה של Angular

מגדירים את האלמנט העוטף אפליקציה של Angular באמצעות הוסף **ng-app** לtagית הפונקציה של האלמנט. במקרה זה, האלמנט העוטף הוא DIV.

```
<div ng-app="">
</div>
```

### 3. מגדירים בתוך האפליקציה את מקור הנתונים באמצעות ng-model

במקרה זה, מקור הנתונים הוא שדה קלט אותו מזין המשתמש.

```
<div ng-app="">
  <input ng-model="name" type="text">
</div>
```

בהזדמנות זו ניתן למקור הנתונים (מכונה גם מודול) את השם "name".

### 4. נגדיר את האלמנט שמקבל את הנתונים מהמודול ומציג אותם באמצעות -פוך bind

```
<div ng-app="">
  <input ng-model="name" type="text">
  <span ng-bind="name"></span>
</div>
```

זהו, השלכנו את כתיבת האפליקציה הראשונה.

## (directives) והוראות Angular

Angular מרחיב את אוצר המילים של ה-HTML באמצעות הוראות (directives), שאוותם אמצעים נוספים לתיגיות ה-HTML.

באפליקציה הפשוטה אותה פיתחנו במדריך השתמשנו במספר directives של Angular, ובכללם:

directive שמנדרת שמסמן ה-HTML (או חלקים ממנו) לשימושו של Angular	<b>ng-app</b>
directive שמנדרת את מקור המידע לשימושו של האפליקציה.	<b>ng-model</b>
HTML אומר ל-angular להחליף את תוכנו של אלמנט בערך של ביטוי, ועדכן את ה-HTML כשער הביטוי משתנה.	<b>ng-bind</b>

במדריכים הבאים, נגשוו סוגים נוספים של directives.

## סיכום

אחרי שתבטו בו ידינו אפליקציה שקשורה בין מידע שמצוין המשתמש ובין הפלט שמוצג למשתמש בזמן אמיתי, במדריך הבא נלמד כיצד לארגן את האפליקציה שלנו, באופן יעיל יותר **בארქיטקטורת MVC**. מה שייתן לאפליקציה גמישות גדולה יותר, ואף יאפשר לנו לכתוב אפליקציות מורכבות בקלות רבה.

### כתיבת האפליקציה כוללת את הצעדים הבאים

1. נכלול את JS ב-head של החטם'ל

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title> מדריך Angular JS</title>
  <script
    src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0/angular.min.js"></script>
</head>
```

במקרה זה, הספרייה לא מאוחסנת אצלם באתר, אלא יודעת מ-cdn

עכשו כל המידע שemandין לטען שדה הקלט, יוצג באלמנט ה-span.

איך מתරחש הפלा הזה?

- שדה הקלט מוגדר כ **ng-model="name"** והוא מהוות את מקור הנתונים.
- מקור הנתונים מעדק את ה-span שמדובר ב **ng-bind=name**.

הסיבה שהמידע עובר מהמודול ומעדק את החטם'ל היא שם שניהם שייכים לאוותה האפליקציה (שניהם ישבים תחת אותו app-tag), ויש בינם databinding.

### נקבל נתונים מסוים שදות במקום אחד

ניתן לקשר יותר ממקור נתונים אחד (יותר מודול אחד). בדוגמה זו, נגדיר שני מודלים: name ו lastName ווציג את הערך שモקדל לתוכם בתחת החטם'ל.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title> מדריך Angular JS</title>
  <script
    src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0/angular.min.js"></script>
</head>
<body dir="rtl" ng-app="">
  <input ng-model="name" type="text">
  משפטה: <input ng-model="lastName" type="text">
  <span ng-bind="name"></span> <span ng-bind="lastName"></span>
</body>
</html>
```

דף ה-MVC הוא מאוד פופולרי בעולם המחשבים בגלל השימוש בו מעניק לתוכנה מבנה גמיש שקל להרחב אותו לאפליקציות גדולות וモותחנות.

נו בעצמכם את האפליקציה שנפתחה במדריך. הפעם האפליקציה כוללת ערך ברייט מודול, ואם תזינו את השם והכינוי תוכל לשנות את ההודעה שמצוגת לכם.

שם: ג'וֹן
כינוי: השועל
שלום, ג'וֹן המconeה השועל

## MVC(Model, View, Controller)

### 1. נגידיר את האפליקציה

נגידיר שהאפליקציה יושבת בתוך דיב מסוים על ידי הגדרת הדיב באמצעות app-ago בשם myapp.

```
<div ng-app="myapp">
</div>
```

- חובה לעטוף את האפליקציה ב-`ng-app`.
- דרך כלثنש אותו על האלמנט של החטטמ"ל שועטף את כל הדף.
- במדריך זה, שמו על הדיב העטוף את האפליקציה.

### 2. נגידיר את הקונטROLLER

בתוך האפליקציה, שהגדירנו ב-1, נגידיר קונטROLLER שיוטוף את הטופוף.

```
<div ng-app="myapp">
  <div ng-controller="HelloCtrl">
    <!--הטופפ...>
  </div>
</div>
```

### 4. נגידיר את ה-view

נגידיר את ה-view בתוך סוגרים מסוללים כפולים, ונגידיר שיקבל את ערכו מהפונקציה callMe שם היא שיכת לאובייקט user.

```
<div ng-app="myapp">
  <div ng-controller="HelloCtrl">
    שם: <input type="text" ng-model="user.name">
    כינוי: <input type="text" ng-
model="user.nickName">
  </div>
</div>
{{ user.callMe() }}
```

## AngularJS ב-MVC

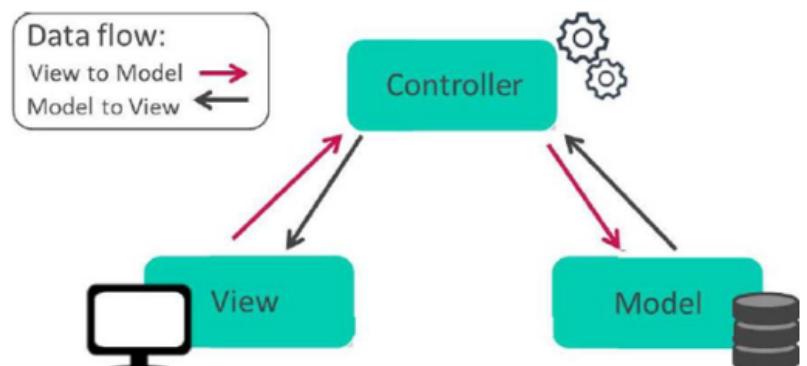
מחבר: יוסי בן הרוש בתאריך: 14.02.2016

דף ה- MVC הוא אחד מבני היסודות של AngularJS, ולכן חשוב להבין מה זה אומר כבר עכשיו. אפליקציה מבוססת MVC, מורכבת משלושה חלקים: Model, View, Controller

- **Model** - מקור המידע (לדוגמא, מסד נתונים או מידע שmagע מהמשתמש).
- **View** - הציגוהו שוראים המשתמשים.
- **Controller** - מתווך בין ה-Model וה-View.

הדרך שבה דף ה-MVC מיושם ב-Angular מאפשרת למידע לזרום ממוקור הנתונים במודול-`view`, והופך, באופן אוטומטי, ובלי שנ:center רעב לעבוד בשבייל זה.

הרטוט הבא מתרגם את כווני זרימת המידע ממקור הנתונים (המודול) ל-`view` (התצוגה) דרך הקונטROLLER, שמהווה את המתווך באפליקציה. המידע יכול לעשות גם את הדבר ההפוך מה-`view` אל האחסנה במודול כשהמתווך הוא הקונטROLLER.



- שמו של הקונטROLLER HelloCtrl, וזה לפ"י המוסכמה שנוהג לסיים את שמו של הקונטROLLER ב-`Ctrl`-ב-`Ctrl`-`model`.

### 3. נגידיר את מקורות המידע (המודולים)

נגידיר את מקורות המידע (המודולים) על ידי הגדרת שדות קלט באמצעות `ng-model`. שדה הקלט הראשון מקבל את ערכו מהתוכנה name של האובייקט `user`, ושדה הקלט השני מהתוכנה nickName של האובייקט `user`.

```
<div ng-app="myapp">
  <div ng-controller="HelloCtrl">
    שם: <input type="text" ng-model="user.name">
    כינוי: <input type="text" ng-
model="user.nickName">
  </div>
</div>
```

## 7. נוסיף לאובייקט תכונות

נוסיף לאובייקט `user` את התכונות: `.name`, `nickName`.

```
<script>
var myapp = angular.module("myapp", []);

myapp.controller("HelloCtrl", function($scope) {
  $scope.user = {
    name: "אלון",
    nickName: "השוול"
  };
})
</script>
```

- ננצל את ההזמנות כדי לחת ערכים בריית מחדל למשתנים.

## 8. נוסיף לאובייקט מתודות

נוסיף לאובייקט `user` את המתודה: `.callMe`.

```
<script>
var myapp = angular.module("myapp", []);

myapp.controller("HelloCtrl", function($scope) {
  $scope.user = {
    name: "אלון",
    nickName: "השוול",

    callMe: function() {
      var userObject = $scope.user;
      return userObject.name + " הידוע בכינוי " +
        userObject.nickName;
    }
})
</script>
```

וכשיין, בין תגיות סקריפט, נגדר את הקונטROLLER כשייר למודל `app` הבא:

```
<script>
var myapp = angular.module("myapp", []);

myapp.controller("HelloCtrl", function($scope) {
  // הערך הדיפולטי כפי שמיד נראה
})
</script>
```

- המודול מקבל בתור פרמטר ראשון את שמה של האפליקציה (במקרה זה, `app`), ופרמטר שני עם רשימת התלוויות (במקרה זה, אין תלויות).
- עבורו לקונטROLLER את האובייקט `$scope` שמצבע על (מחזק רפנס של) האפליקציה שאיתה עבדים, ומשמש כדי לקשר בין הקוד של הקונטROLLER לבין התוכן שמוצג ב-`view`.

## 6. נגדר את האובייקטים שאתם נowane בקונטROLLER

בתוך הקונטROLLER נגדר את האובייקט `user` כשייר ל- `$.scope`.

```
<script>
var myapp = angular.module("myapp", []);

myapp.controller("HelloCtrl", function($scope) {
  $scope.user = {};
})
</script>
```

- בתוך המתודה `callMe` הגדרנו `userObj` שلتוכו הצבנו את `user`.
- מה שמאפשר לנו להציג את התכונות של האובייקט `user`.

## איך זה עובד?

התצוגה (view) ניזונה ממוקור הנתונים (מודול) כשהמתווך הוא הקונטROLLER. או בקיצור, **MVC - model view controller**.

**הקשר הוא דו-כיווני.** מה שנמצא בתוך הקוד של הקונטROLLER מעדכן את ה-`view`, ומה שմזינים לתוך ה-`view`, מעדכן באופן אוטומטי את הקונטROLLER.

## שימוש לב לואפּן העבודה הנוכחי עם Angular

- תחכו את האפליקציה לדיב מסוים בלבד, כדי לאפשר למספר אפליקציות לעבוד במקביל באותו עמוד (כל אפליקציה על דיב או איזור אחר בדף).
- את הקונטROLLER הגדכנו בתוך פונקציה, כדי להימנע מזיהום ה-namespace (כי הקונטROLLER תחום רק לפונקציות מוגדרות ולא משותף לכל הקוד בדף). באופן זה, אם יוכלים להריץ מספר אפליקציות במקביל על אותו דף בלי שהם יפריעו האחת לרעותה.

כך נראה הקוד שכתבנו עד עכשיו:

```
<div ng-app="myapp">
  <div ng-controller="HelloCtrl">
    <input type="text" ng-model="user.name">
    <input type="text" ng-
model="user.nickName">
  </div>
</div>
{{ user.callMe() }}
```

```
<script>
var myapp = angular.module("myapp", []);

myapp.controller("HelloCtrl", function($scope) {
  $scope.user = {
    name: "ג'ון",
    nickName: "הושאיל",

    callMe: function() {
      var userObject = $scope.user;
      return userObject.name + " הידוע בכינוי " +
userObject.nickName;
    }
  };
}) ;
</script>
```

### 3 שלבים לכתיבת האפליקציה

1. נגידר את האפליקציה באמצעות `app`-`ng`

```
<div ng-app="">
</div>
```

2. נאתחל את המידע באפליקציה באמצעות מערך של אובייקטים באמצעות -`ng-init`

```
<div ng-app="" ng-init = "users=[{name:'ג'ו',nickName:'השועל'}, {name:'dag',nickName:'dag',nisim:'dag'}, {name:'tzion',nickName:'tzion',nisim:'tzion'}]">
</div>
```

בתוך ה-`ng-init` הגדרים מערך של אובייקטים של סוג שמכיל את השם והכינוי של המשתמשים. כך זה נראה:

```
users= [
  {name:'ג'ו',nickName:'השועל'},
  {name:'dag',nickName:'dag',nisim:'dag'},
  {name:'tzion',nickName:'tzion',nisim:'tzion'}
]
```

3. נתרגם את המערך לרישימה של `html` באמצעות `ng-repeat`

```
<div ng-app="" ng-init = "users=[{name:'ג'ו',nickName:'השועל'}, {name:'dag',nickName:'dag',nisim:'dag'}, {name:'tzion',nickName:'tzion',nisim:'tzion'}]">
<ul>
  <li ng-repeat = "user in users">
    {{ user.name + ' ' + user.nickName }}
  </li>
</ul>
</div>
```

## הוראות (directives) ב-JS (directives)

מחבר: יוסי בר הרש בתאריך: 21.02.2016

Angular מרחיב את אוצר המילים של HTML באמצעות הוראות (directives), שאוותם מושגים בתור שודת לתגיוט ה-HTML.

ה-directives אומרות לאפליקציה האנגלורית מה עלייה לعشות.

כבר השתמשנו במספר הוראות (directives) של Angular, ובכלל קרן:

- `app` הוראה שמגדירה שקובץ ה-HTML (או חלקים ממנו) שייכים לאפליקציה האנגלורית.

- `controller` שמחברת את הקוד של הקונטROLLER ל-view.

- `model` הוראה שמגדירה את מקור המידע לשימוש האפליקציה.

במדריך זה נזכיר עוד שני הוראות:

- `init` מאתחלת את המידע שישמש את האפליקציה.

- `repeat` מאפשר להציג כל אלמנט שמקורו באוסף באופן נפרד. לדוגמה, שימוש ב`repeat` מאפשר להציג מערך בתור רישמה של HTML.

האפליקציה שנספתח במדריך תציג רישימת שמות וכינויים שתתבסס על מערך.

זהו הקוד המלא:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title> מדריך angular JS</title>
  <script
src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0/angular.min.js">
</head>
<body dir="rtl">
<div ng-app="" ng-init = "users=[{name:'ג'ו',nickName:'השועל'}, {name:'dag',nickName:'dag',nisim:'dag'}, {name:'tzion',nickName:'tzion',nisim:'tzion'}]">
<ul>
  <li ng-repeat = "user in users">
    {{ user.name + ' ' + user.nickName }}
  </li>
</ul>
</div>
</body>
</html>
```

### כך נראה האפליקציה בפועל

האפליקציה מתרגמת את המערך לרישימה.

- ג'ו המcona השועל
- ניסים המcona הדג
- ציון המcona פצצת אסום

## אירועים אנגולריים

ניתן לחבר מזין לאירוע לאלמנט הtemp"ל על ידי שימוש באחד ה-directives להזמנה:

למה הוא משמש?	directive
כשמקליקים על אלמנט	ng-click
כשמקליקים הקלה כפולה על אלמנט	ng-dblclick
כשנכנים עם העכבר לתוכן אלמנט	ng-mouseenter
כשיוצאים עם העכבר מתוך אלמנט	ng-mouseleave
לחיצה על מקש במקלדת	ng-keypress
כשנכנים לשדה בטופס	ng-focus
כשיוצאים משדה בטופס	ng-blur

## האזנה ותגובה לEvents ב- AngularJS

מחבר: יוסף בן הרוש בתאריך: 25.02.2016

אחרי שבמדריך קודם טיפלנו ב- directives של AngularJS ממש על קצת המצלג, במדריך זה נלמד להשתמש ב-directives שמאזינים לאירועים. שימוש ב-directive של אירועים מאפשר לנו לכתוב קוד שmagic לאירועים שונים. לדוגמה, קוד שעשו המשהו בתגובה להקלקה על כפתור או בתגובה ליציאה משדה בטופס.

את ה-directive מכניסים לאלמנט, ובאותה ההזדמנות גם מגדרים את הפונקציה שmagic לאירוע. לדוגמה:

```
<a href="#" ng-click="addOne()"> לחץ עליי </a>
```

בלינק שבדוגמה, אנחנו משתמשים במאזין לאירוע **ng-click**, ומגדירים שהפונקציה **(()) addOne** תgive לאירוע.

## אפליקציה לדוגמה

במדריך זה נפתח מיני-אפליקציה שבסוגرتה אנחנו מוסיףים אחד או מורידים אחד מהערך בשדה טקסט בתגובה ללחיצה על לינקים.  
לחצו על הלינקים שמופיעים עליהם הכיתוב + ו- כדי לשנות את הערך של המספר המוצג בשדה.

## 1. ניצור את האפליקציה ונדיר את הקונטROLLER

- האפליקציה מוגדרת באמצעות **app**
- הקונטROLLER מוגדר באמצעות **ng-controller**

```
<div ng-app="myApp">
  <div ng-controller="mainController">
```

3. נוסיף הקוד את ה-**directive**

```
<div ng-app="myApp">
  <div ng-controller="mainController">
    <a href="#" ng-click="addOne($event)"> + </a>
    <input type="text" value="{{ count }}"/>
    <a href="#" ng-click="subOne($event)"> - </a>
  </div>
</div>
```

## 2. נוסיף את 2 הלינקים ואות שדה הטקסט

```
<div ng-app="myApp">
  <div ng-controller="mainController">
    <a href="#"> + </a>
    <input type="text" value="" />
    <a href="#"> - </a>
  </div>
</div>
```

- **ng-click** מORIZן לאיירע של הקלקה על הלינק, ומפעיל את הפונקציה בתגובהה
- המשתנה **count** הוא זה שישתנה בתגובה להקלוקות

## 5. נוסיף את הפונקציות

```
<script>
var myApp = angular.module("myApp", []);

myApp.controller("mainController", function($scope) {
  $scope.count = 0;

  $scope.addOne = function(event) {
    event.preventDefault();
    $scope.count++;
  }

  $scope.subOne = function(event) {
    event.preventDefault();
    $scope.count--;
  }
});
```

Angular 4. נוסיף את הקוד ה-**js**

```
<script>
var myApp = angular.module("myApp", []);

myApp.controller("mainController", function($scope) {
  $scope.count = 0;
});
```

- נDIR את האפליקציה בקוד.
- נDIR את הקונטROLLER.

• נDIR את המשתנה **count**, שערך מתעדכן בתגובה ללחיצות על הלינקים.

- הפונקציה **addOne()** מוסיפה 1 למשתנה **count**, והפונקציה **subOne()** מפחיתה 1 מערכו של המשתנה.
- כדי למנוע את התנהגות ברירת המחדל של פתיחת דף חדש בתגובה להקלקה על הלינק, אנחנו מعتبرים לפונקציה את האובייקט **event**. הפונקציה מונעת את פתיחת הדף באמצעות **event.preventDefault()**.

```
<html>
<head>
<meta charset="utf-8">
<title>Angular events</title>
</head>
<body>
<div ng-app="myApp">
<div ng-controller="mainController">
<a href="#" ng-click="addOne($event)"> + </a>
<input type="text" value="{{ count }}"/>
<a href="#" ng-click="subOne($event)"> - </a>
</div>
</div>

<script>
var myApp = angular.module("myApp", []);

myApp.controller("mainController", function($scope) {
    $scope.count = 0;

    $scope.addOne = function(event) {
        event.preventDefault();
        $scope.count++;
    }

    $scope.subOne = function(event) {
        event.preventDefault();
        $scope.count--;
    }
});
</script>
</body>
</html>
```

הקוד המלא

## רשימת דגמי רכבים ומחיר

- Audi - \$30,000
- Jaguar - \$60,000
- Sussita - \$2,500

מודל המכונית:

מחיר המכונית:

**הוסף****ה-HTML**

נדיר את האפליקציה באמצעות app-הו:

**ng-app="myApp"**

נדיר את הקונטROLLER באמצעות ng-controller:

**ng-controller="carListCtrl"**

מכיוון שאנו ממעוניינים להציג את המחיר בדולרים ובמספרים שלמים נשתמש בפילטר אנגלורי:

{{carModel.price | currency:"\$":0}}

נדיר את השדות שאליהם מzin את שמות הרכבים ומחייבם באמצעות ng-model:

```
<input type="text" ng-model="newCarModel">
<input type="text" ng-model="newPrice">
```

לחיצה על הכפתור (ng-click) תקרא לפונקציה addNewItem, ותעביר את הנתונים החדשניים שהזמננו לטופוף:

```
<button ng-click="addNewItem(newCarModel, newPrice)">+ Add model and price</button>
```

```
<ul>
<li ng-repeat="carModel in carModels"> {{carModel.name}} - {{carModel.price}}</li>
</ul>
```

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
    <meta charset="UTF-8">
    <title>Angular tutorial</title>
    <script
src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.min.js"></script>
</head>
<body dir="ltr">
    <div ng-controller="carListCtrl">

        <h3>List of car models</h3>

        <ul>
            <li ng-repeat="carModel in carModels">
                {{carModel.name}} - {{carModel.price | currency:"$":0}}</li>
        </ul>

        <p class="error">{{addModelError}}</p>

        <label>Car model name: <input type="text" ng-model="newCarModel"></label>

        <br /><br />

        <label>Car model price: <input type="text" ng-model="newPrice"></label>

        <br /><br />

        <button ng-click= "addItem(newCarModel, newPrice)">+
            Add model and price</button>

    </div>
</div>
```

## הקוד המלא

### קוד האנגולר

ניצור את האפליקציה, שנקרה לה :myApp

```
var myApp = angular.module('myApp', []);
```

בקונטROLLER carListCtrl, המערך carModels יכול את הנתונים שיוצגו ברשימה. הפקציה addItem עשויה ולידיה בסיסית לנתונים. במידה ותמצא שגיאה, ערך הודעת השגיאה יעדכן את המשתנה addModelError, שיזג באוטומטי.

ובמידה והנתונים שהזין המשתמש מספקים את הדרישות, יתווסף פריט חדש למערך carModels, והרשימה שמצוגת למשתמש תתעדכן באופן אוטומטי.

```
var myApp = angular.module('myApp', []);

myApp.controller('carListCtrl', function($scope){
    $scope.carModels = [
        {name: "Audi", price: "30000"},
        {name: "Jaguar", price: "60000"},
        {name: "Sussita", price: "2500"},
    ];

    $scope.addItem = function(newCarModel, newPrice){
        $scope.addModelError = "";

        if(newCarModel === undefined || newCarModel == "" || newPrice === undefined || newPrice == ""){
            $scope.addModelError = "Please enter new car model and price.";
        } else if(!parseInt(newPrice)) {
            $scope.addModelError = "Please provide a number as a price.";
        } else if(/[^a-zA-Z ]/.test(newCarModel) || newCarModel.length < 3) {
            $scope.addModelError = "Please provide a valid car model name.";
        } else {
            $scope.carModels.push({
                name : newCarModel,
                price: parseInt(newPrice)
            });
        }
    };
});
```

מחבר: יוסי בן הרוש בתאריך: 18.03.2016

**AJAX** היא טכניקה שמאפשרת לשנות את תוכנו של דף אינטרנט בלבד לרענן את הדף. מה שגורם לשיפור חווית המשתמש באתר, ולטיענה מהירה יותר של תוכן הדף. דוגמה לשימוש בא-AJAX ניתן למצוא באתר המפות Google Maps, שבו ניתן לבצע ממפה באופן חלק, בלי צורך לרענן את הדף.

## השירות `$http` של Angular

השירות `$http` מאפשר את העבודה עם AJAX באפליקציות של AngularJS. כדי להשתמש בשירות, علينا להזירק לקונטROLLER את האובייקט `$http`, שהוא שירות מובנה של אנגולר.

```
var ajaxExample = angular.module('ajaxExample', []);

ajaxExample.controller('mainController', function($scope,
$http){

});
```



AJAX פועל על ידי פקודות שזרמות באופן רציף מהדף אל השירות, ומוחזקות ממנו את המידע הנדרש. כשהמשתמש מבצע פעולה דוגמת הקלקה על אלמנט או גילת הדף, קוד java script שמאזין לפועלה, שולח קריאה לשרת, והשרת מעביד את הקריאה, ומחזיר את המידע הנדרש להציגו בדף.

ב-Angular משתמשים בשירות ששמו `$http`, שהופך את העבודה עם AJAX לקל ופיטוט במילוי.

במדריך נפתח אפליקציה מסווג אפלון שמכיל שמות פרטיים ושמות משפחה. האפליקציה מאפשרת צפיה בשמות, וגם הוספה ומחיקה של שמות. צד השירות כתוב ב-PHP.

במדריכים קודמים כבר נתקלנו בשירות של אנגולר שהוא `-eScope`, שבו השתמשנו כדי לבדוק בין הלוגיקה לבין ה-HTML.

כדי לעבוד עם השירות `$http` נדרש להגדיר מספר פרמטרים:

- הפרמטר `method` שמגדיר האם מקבלים מידע ב-GET או שולחים מידע ב-POST.
- הפרמטר `url` שמגדיר את הכתובת שבה נמצא המידע-
- קוד שבדוק את התוצאות (צד השירות של האפליקציה).
- קוד שציריך לרווח במקורה של הצלחה.
- קוד שציריך לרווח במקרה של כישלון.

## כך נראה קריאה של נתונים מקור המידע

כדי לקבל מידע, علينا להגדיר את המתודה `get`, כמו גם את `-url` של מקור המידע (`api/get.php`).

בקרה של הצלחה, יופעל התנאי `then`, ובמקרה של שגיאה תיראה הטענה השניה.

אחרי שהבנו את המבנה הבסיסי של שימוש בא-AJAX באפליקציות אנגולריות, נעבור לכתוב את הנטול'.

```
$http({
  method: 'GET',
  url: 'api/get.php'
}).then(function (response) {
  // code to run on success
}, function (response) {
  // code to run on error
});
```

```

<!doctype html>
<html ng-app="ajaxExample">
<head lang="he-IL">
<meta charset="utf-8">
<title>Angular Ajax</title>
</head>
<body>
    <form ng-controller="mainController">
        <h2>הוסף</h2>

        <p>מלא שם חדש והנש את הטופס</p>
        <div>
            <label>שם</label>
            <input type="text" ng-model="newName">
        </div>

        <div>
            <label>כינוי</label>
            <input type="text" ng-model="newNickName">
        </div>

        <input type="button" value="הוסף" ng-click="addUser()">
    </form>

    <h2>הרשימה</h2>
    <p>. ברשימה זו יוצגו השמות שהווסף באמצעות הטופס</p>

    <ul>
        <li ng-repeat="user in users">
            <button ng-click="deleteUser( user.id )">Delete</button> {{ user.name }} {{ user.nickName }}
        </li>
    </ul>

    <script
src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0/angular.min.js"></script>
    <script src="/assets/js/app.js"></script>

</body>
</html>

```

### הטמ"ל שישרת את האפליקציה

הקוד ימצא בקובץ index.html

הטמ"ל מתחולק לשניים. החלק הראשון הוא טופס שימושה להכנסת שם המשתמש והכינוי, והחלק השני הוא הרשימה שבה יציגו השמות. כל אחד מהשמות ניתן למחיקה בלחיצת כפתור.

- האפליקציה תוגדר באמצעות app-ו על התגית html. שם האפליקציה: ajaxExample
- נגידר את הקונטROLLER שלו mainController על הטופס (form).
- נגידר את שדות הקלט בתור ng-model: newName ו newNickName.
- newName לשדה שבו מגדרים את הاسم.
- newNickName לשדה שבו מגדרים את הכינוי.
- על האלמנט btn נגידר את btn-click-ו שיפעל את המתודה addUser().
- הרשימה מציגה את שמות המשתמשים שנוספו באמצעות הטופס, והוא עשוה זאת באמצעות ng-repeat.
- כל פריט ברשימה כולל את השם והכינוי וכפתור שלחיצה עליו מפעילה את הפונקציה deleteUser שמחזקת את הרשימה.

קוד האנגולר כולל 3 פונקציות:

1. getUsers - שעובד AJAX עם סקריפט PHP שקורא את הרשומות ממסד הנתונים, ומוסיף את הרשומות לרשימה.
2. addUser - מקבל את השמות שנוספו באמצעות הטופס, ושולח אותו ב-AJAX לסקריפט PHP שמכניס את המידע החדש למסד הנתונים.
3. deleteUser - מקבל את id הרשומה, ושולח קריית AJAX לסקריפט PHP שמחוק את הרשומה מסד הנתונים.

```
var ajaxExample = angular.module('ajaxExample', []);

ajaxExample.controller('mainController', function($scope, $http) {
    $scope.users;

    $scope.getUsers = function() {
        $http({
            method: 'GET',
            url: '/api/get.php'
        }).then(function(response) { // on success
            // this callback will be called asynchronously
            // when the response is available
            $scope.users = response.data;
        }, function(response) { // on error
            // called asynchronously if an error occurs
            // or server returns response with an error status.
            console.log(response.data, response.status);
        });
    };

    $scope.addUser = function() {
        $http({
            method: 'POST',
            url: '/api/post.php',
            data: {newName: $scope.newName,
newNickName: $scope.newNickName }
        }).then(function(response) { // on success
            $scope.getUsers();
        }, function(response) { // on error
            console.log(response.data, response.status);
        });
    };
});
```

```
$scope.getUsers();

}, function(response) { // on error

console.log(response.data, response.status);
});

};

$scope.deleteUser = function(id) {

$http({
    method: 'POST',
    url: '/api/delete.php',
    data: { userId : id }
}).then(function(response) {

    $scope.getUsers();

}, function(response) {

console.log(response.data, response.status);
});

};

$scope.getUsers();
});
```

את קוד ה-PHP נשים בתקיית api, והוא כולל את הקבצים הבאים:

1. functions.php שכולל את הפונקציה connect שמקשרת עם מסד הנתונים
2. get.php ששולף את המידע מסד הנתונים
3. post.php יוסיף את המידע למסד הנתונים
4. delete.php ימחוק רשומות מסד הנתונים

### שימוש לתקשרות עם מסד הנתונים functions.php

```
<?php
// db credentials
define('DB_HOST', 'localhost');
define('DB_USER','root');
define('DB_PASS','');
define('DB_NAME','angular');

// connect with the database.
function connect()
{
    $connect = mysqli_connect(DB_HOST ,DB_USER ,DB_PASS
,DB_NAME);

    if (mysqli_connect_errno($connect))
    {
        echo "Failed to connect:" . mysqli_connect_error();
        die();
    }

    mysqli_set_charset($connect, "utf8");

    return $connect;
}
```

### שימוש לקריאת כל הרשומות מסד הנתונים get.php

```
<?php
require 'functions.php';

$connect = connect();

// Get the data
$users = array();
$sql = "SELECT id, name, nickName FROM users WHERE 1";

if($result = mysqli_query($connect,$sql))
{
    $count = mysqli_num_rows($result);

    $cr = 0;
    while($row = mysqli_fetch_assoc($result))
    {
        $users[$cr]['id']      = $row['id'];
        $users[$cr]['name']    = $row['name'];
        $users[$cr]['nickName'] = $row['nickName'];

        $cr++;
    }
}

$json = json_encode($users);
echo $json;
exit;
```

צריך לעשות לתוצאה .json\_encode

## משמש להוספה רשומות למסד הנתונים post.php

את המידע שנשלח מאנגולר בפואט, מקבלים באמצעות:

```
file_get_contents("php://input")
```

צרי לעשות decode\_json לנתונים.

```
<?php
require 'functions.php';

$connect = connect();

// Add the new data to the database.
$postdata = file_get_contents("php://input");
if(isset($postdata) && !empty($postdata))
{
    $request = json_decode($postdata);

    $newName = preg_replace('/[^a-zA-Zñ-ñ ]/','$request->newName');
    $newNickName = preg_replace('/[^a-zA-Zñ-ñ ]/','$request->newNickName');

    $newName =
    mysqli_real_escape_string($connect,$newName);
    $newNickName =
    mysqli_real_escape_string($connect,$newNickName);

    $sql = "INSERT INTO `users`(`name`, `nickname`) VALUES
    ('$newName','$newNickName')";

    mysqli_query($connect,$sql);
}
```

## משמש למחיקת רשומות מסד הנתונים delete.php

```
<?php
require 'functions.php';

$connect = connect();

// Delete single record by id.
$postdata = file_get_contents("php://input");
if(isset($postdata) && !empty($postdata))
{
    $request = json_decode($postdata);

    $userId =
(int)mysqli_real_escape_string($connect,$request->userId);

    $sql = "DELETE FROM `users` WHERE `id` = " . $userId;

    mysqli_query($connect,$sql);
}
```

# אפליקציות מבוססות דף יחיד באמצעות AngularJS

מחבר: יוסי בן הרוש בתאריך: 18.03.2016

במדריך זה הגענו לסייעת העיקרית שבגלאה אנחנו משתמשים באנגולר. כי אנגולר זו לא רק דרך לכתוב JavaScript בסגנון פרימיטיבי שמכיר אותו לכתוב קוד JavaScript מודרני, אלא עיקר ספריית קוד שמאפשרת לנו לכתוב אפליקציות SPA (single page application) בצורה ישרה ומודרנית.

```
<script>
var myApp = angular.module('myApp', ['ngRoute']);
myApp.config(function($routeProvider) {
  $routeProvider.
    when('/first', {
      templateUrl: 'pages/first.html',
      controller : 'firstController'
    }).
    when('/second', {
      templateUrl: 'pages/second.html',
      controller : 'secondController'
    }).
    when('/second/:num', {
      templateUrl: 'pages/second.html',
      controller : 'secondController'
    }).
    otherwise({
      redirectTo: '/first'
    });
});
```



## ה-URL באפליקציה

כשעושים אפליקציות של דף יחיד (spa) תוכן הדף משתנה בהתאם לשינוי בחלק הסולמיית (tag) של ה-URL.

אם תרגלו ל כתוב את ה-URLים באופן הבא:

```
yourwebsite>/first>//
yourwebsite>/second>//
yourwebsite>/second/1>//
```

אך בשבייל ה-single page application משתמשים ב-URLים עם סולמיות (#) במקום:

✖ ⓘ

yourwebsite>/#first>//
yourwebsite>/#second>//
yourwebsite>/#second/1>//

זה מאוד משמעותית מפני ששימוש בסולמיות גורם לנו להישאר באותו דף כל הזמן. זה אומר שהתוכן של האתר משתנה ביל' לטען את כל הדף מחדש, וכטזאה מכך נונענת התופעה של הצגת דפים לבנים בין דף לדף, וגםazon הטענה של התכנים מתקצר.

מהו זה index.html(index.html) מושך את שלד האפליקציה, והוא כולל את הקישורים שיאפשרו לדפסה בין התכנים (האלמנט nav).

תוכן הדיב view-וֹג יעדכן לפי התוכן הנבחר.

## הראוטר

נוסף לקוד ספרייה ה-`angular-route.js`, נכלל את קוד המודול האנגולרי שמספק את הרואוטר (נתב בעברית), שמאפשר להחליף את תוכן הדף בתגובה לשינוי ה-`url`.

## הטמפליטים

האפליקציה כוללת שני תכנים, `first.html` ו-`second.html`. את שני הטמפליטים שמשמשו את התכנים נוסיף לתוך התקייה `pages`.

`pages/first.html/`

```
<p>This page is called: {{name}}</p>
```

`pages/second.html/`

```
<p>This page is for {{name}}, and it got the number  
of {{ num }}.</p>  
  
<p><a href="#/second/1">1</a> | <a  
href="#/second/2">2</a> | <a href="#/second/3">3</a></p>
```

ערוך המשתנים `name` ו-`num` מתקובל מקוד JavaScript, כפי שנראה מיד.

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<meta charset="UTF-8">

<title>Single page application example with  
AngularJS</title>

<script

src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.min.js"></script>

<script

src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular-  
route.js"></script>

</head>

<body dir="ltr">

<nav><a href="#">Main</a> | <a

href="#/second">Second</a></nav>

<div ng-view></div>

</body>

</html>

## קוד ה-JavaScript

כשאנחנו יוצרים את האפליקציה, נגידר בראשית התוליות את המודול `ngRoute`.

```
var myApp = angular.module('myApp', ['ngRoute']);
```

כתבו את הקונטrollers. שנ-קונטrollers לשני תכנים.

בתוך הקונטroller 'firstController' נגידר את ערך המשתנה `name` שישתמש את הקונטroller.

```
myApp.controller('firstController', function($scope){  
    $scope.name = "the first";  
});
```

בתוך הקונטroller 'secondController' נגידר את ערך המשתנה `name` שישתמש את הקונטroller. בנוסף, נגידר את ערך המשתנה `num` שיקבל את ערכו מפרמטר שיעבורו בכתובת. קבלת ערך המשתנה מהכתובת מתאפשרתודות לשימוש באובייקט `$routeParams`. אותה נעביר כתלות לקונטROLLER.

```
myApp.controller('secondController',  
function($scope,$routeParams){  
    $scope.name = "the second";  
    $scope.num = $routeParams.num || 1;  
});
```

הmethod **when** מקבלת בתור פרמטר ראשון את ה-**url**, ולאחר כר אובייקט שכולל את ה-**templateUrl** ואות הקונטROLLER.

```
$routeProvider.  
when('/first', {  
    templateUrl: 'pages/first.html',  
    controller : 'firstController'  
})
```

נית להגדיר ב-**url** את הפרמטר שאנו רוצים להעביר לקונטROLLER. לדוגמה, הפרמטר **:num**.

```
when('/second/:num', {  
    templateUrl: 'pages/second.html',  
    controller : 'secondController'  
})
```

```
myApp.config(function($routeProvider) {
```

```
    $routeProvider.  
    when('/first', {  
        templateUrl: 'pages/first.html',  
        controller : 'firstController'  
    }).  
    when('/second', {  
        templateUrl: 'pages/second.html',  
        controller : 'secondController'  
    }).  
    when('/second/:num', {  
        templateUrl: 'pages/second.html',  
        controller : 'secondController'  
    }).  
    otherwise({  
        redirectTo: '/first'  
    });  
});
```

בקונטROLLER, יתקבל ערך הפרמטר **num** כתובות ה-**:num**, וערך זה יוצב למשתנה **num**. הקפידו לתת ערך ברירת מחדל.

```
myApp.controller('secondController',  
function($scope,$routeParams){  
    $scope.name = "the second";  
    $scope.num = $routeParams.num || 1;  
});
```

## סיכום

במדריך זה למדנו כיצד להשתמש אנגלולר כדי לכתוב אפליקציות מבוססות דף ייחודי, ובמדריך הבא נלמד כיצד להעביר מידע בין הדפים (הكونטROLLERים) באמצעות **.custom services**

## מדריך אנגולר: custom directives וסרטים

משתמשים ב- Custom directives כשביל להחליפם אלמנטים פשוטים בדף אינטרנט בתוכן מורכב יותר. השימוש בהם מושך במיוחד למיוחד שורצים לחזור על אותו אלמנט שוב ושוב. לדוגמה, אם רצזה להחליף את שלושת הדברים הבאים בתוכן עשיר הכלול את שם הסרט, שנה ותמונה.

```
<div movie-result></div>
<div movie-result></div>
<div movie-result></div>
```

### 1. בניית ה-HTML העיקרית

נתחל מיצירת בניית ה-HTML הבסיסית, שכוללת:

- הגדרת ng-app על כל דף ה-HTML.
- הגדרת ng-controller על כל מסויים שבתוכו יבוצע תהליך החלפת האלמנטים בתוכן של directive.custom
- בתוך הקונטROLLER, נגדיר אלמנט עם מייחד attribute movie-result, שיוחלף בהמשך בתוכן מלא שישר לסרט.

```
<div movie-result></div>
```

### начало разработки...

האפליקציה כוללת את 5 המרכיבים הבאים:

1. בניית ה-HTML העיקרית שבה נגידר את האלמנט שאנו ממעוניינים להחליף.
2. קוד ה-angular הבסיסי שכולל את הקונטROLLER.
3. קוד ה-custom directive.
4. הטמפליט שמחילף את האלמנט
5. אפשרות לטמפליט לגשת למידע בקונטROLLER

\* זכרו את שם השדה movie-result כי נתיחס אליו בהמשך כשניתן שם ל.directive

```
<html lang="en" ng-app="myApp">
<head>
  <title>Custom directives</title>
  <script
    src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.min.js"></script>
</head>
<body dir="ltr">
  <div ng-controller="mainCtrl">
    <h3>Movies</h3>
    <div movie-result></div>
  </div>
</body>
</html>
```

הגדרנו על האלמנט שאנו ממעוניינים להחליף את השדה movie-result. שם שמתאר האלמנט, ומציין לכלים שאנגולר מכתב לנו הכללים שם שמכיל אותיות קטנות בלבד, ומילים שמופיעות בכו מפורץ.

קוד האנגולר הבסיסי מחזק אובייקט movie הכלול מידע אודוט הסרט. קוד האנגולר כולל 3 מרכיבים:

- הגדרת האפליקציה שמה myApp.
- הגדרת הקונטROLLER שייר ל애�ליפקציה.
- בתוך הקונטROLLER נגדיר אובייקט movie שיש לו scope, ויכול את שם הסרט והשנה בה נוצר.

### 3. קוד ה-*directive* custom directive

נוסיף לקוד האנגולר גם קוד של directive, שיקבל את השם movieResult. שם זה נגזר מה-*directive*-*custom* שגדירתו על האלמנט שאנו רוצה להחליפה: result. כדי ליזכר את שם ה-*directive* מוחקמים את כל הנקודות המפרידים, ומתחילה כל מילה מלבד המילה הראשונה באות גזולה (lower camel case).

קוד ה-*directive* מחייב אובייקט שכולל את השדות הבאים:

- templateUrl**, שמגדיר את המקום שבו נמצא הטעמ"ל שיוהווה את הטמפליט.
- custom directive** של ה-*directive*.
- scope** שמנגדיר את מקור המידע שהוא נגיש לטמפליט (ועל זה עוד נרחב בהמשך).

```
myApp.directive("movieResult",function(){
  return {
    templateUrl: 'directives/movieresult.html',
    scope: {
      }
    });
});
```

```
var myApp = angular.module("myApp", []);
myApp.controller("mainCtrl",function($scope){
  $scope.movie = {
    name: 'Captain America',
    year: '2016'
  }
});
```

### 4. הטמפליט שמחליף את האלמנט

הטמפליט שמחליף את האלמנט נמצא בקובץ 'directives/movieresult.html'. הטמפליט כולל את השדות שאנו מזמינים שיוחלו בתוכן אמיתי.

אם נרים את מה שתבנו עד כה, לא נראה כלום מפני שאין לטמפליט גישה למידע באובייקט שנמצא ב-*scope* הקונטROLLER בעוד ל-*directive* יש scope משלה.

```
<div>
  <h4>{{ movieName }}</h4>
  <p>{{ movieYear }}</p>
</div>
<hr />
```

```
<div ng-controller="mainCtrl">
  <h3>Movies</h3>
  <div movie-result
    movie-name="{{ movie.name }}"
    movie-year="{{ movie.year }}></div>
</div>
```

המידע הקונקרטי שאמם מעוניינים להעביר מוגדר בין זוג שפמים ({{}}) מפני שמדובר במחזרות.

```
<html lang="en" ng-app="myApp">
<head>
  <title>Custom directives</title>
  <script
src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.min.js"></script>
</head>
<body dir="ltr">
  <div ng-controller="mainCtrl">
    <h3>Movies</h3>
    <div movie-result
      movie-name="{{ movie.name }}"
      movie-year="{{ movie.year }}>
    </div>
  </div>
</script>
var myApp = angular.module("myApp",[]);

myApp.controller("mainCtrl",function($scope){
  $scope.movie = {
    name: 'Captain America',
    year: '2016'
  };
});

myApp.directive("movieResult", function(){
  return {
    templateUrl: 'directives/movieresult.html',
    scope : {
      movieName: "@",
      movieYear: "@"
    }
  };
});

</script>
</body>
</html>
```

**הקוד המלא**

כדי לאפשר לטמפליט לקבל מידע מהקונטROLLER אנחנו צריכים להגדיר באמצעות custom attribute את המידע שאנו רוצים להעביר באמצעות custom attribute (index.html), נגדיר את המידע שאנו רוצים להעביר באמצעות custom attribute (לדוגמא, movie-name), וגם את המידע הקונקרטי שאנו מעוניינים שייעבור (לדוגמא, "{{ movie.name }}").

בטמפליט הראשי (index.html), נגדיר את המידע שאנו רוצים להעביר באמצעות custom attribute (movie-name), וגם את המידע הקונקרטי שאנו מעוניינים שייעבור (לדוגמא, "{{ movie.name }}").

נגדיר ב-scope של ה-custom directive שהגדכנו מקבל ערך של טקסט מהקונטROLLER באמצעות הסמל @. (למה? כהן)

המשמעות של @ זה קשריה חד-כיוונית של טקסט. כך שהמידע זורם מהקונטROLLER לטמפליט, ולא להיפך.

```
myApp.directive("movieResult", function(){
  return {
    templateUrl: 'directives/movieresult.html',
    scope : {
      movieName: "@",
      movieYear: "@"
    }
  });
});
```

directives/movieresult.html

```
<div>
  <h4>{{ movieName }}</h4>
  <p>{{ movieYear }}</p>
</div>
<hr />
```

עד כה רأינו כיצד ניתן לטעמפליט גישה למידע מסווג מחרוזת, אבל ישנו עוד שני טיפוסי מידע שהטמפליט צריך לקבל, שהם אובייקט ופונקציה.

```
<div ng-controller="mainCtrl">
  <h3>Movies</h3>
  <div movie-result movie-object="movie"></div>
</div>
```

## כיצד להעביר אובייקט?

כפי שניתן לתוך המידע מסווג מחרוזת לעבר מהקונטROLLER לטמפליט, ניתן גם להעביר אובייקט.

1. נגדיר את האובייקט שרצים להעביר בטמפליט הראשי. -`movie`

2. נגדיר ב-`scope` של ה-`directive` שה-`custom attribute` שהגדכנו מקבל אובייקט מהקונטROLLER.

המשמעות של "=" היא קשירה דו-כיוונית של האובייקט.

3. לשנות את הדירקטיב, כך שיידע להציג את המידע שמתקיים מהאובייקט.

```
<div>
  <h4>{{ movieObject.name }}</h4>
  <p>{{ movieObject.year }}</p>
</div>
<hr />
```

```
myApp.directive("movieResult", function(){
  return {
    templateUrl: 'directives/movieresult.html',
    scope : {
      movieObject: "="
    }
  );
});
```

## כיצד להעביר את הסוג הבוליאני?

שימוש לב שאות הסוג בוליאני (true/false) מעבירים כמו אובייקט.

צריך לשים לב שקשירה של אובייקט באנגלר היא קשירה דו-כיוונית, ולכן המידע יזרום מהקונטROLLER לטמפליט, וגם חזרה מהטמפליט לקונטROLLER, מה שעלול לגרום בעית אבטחה. לכן, צריך להיזהר שימושתמשים באופציה של קשירת אובייקט.

אחרי שלמדנו כיצד להעביר לטמפליט מחרוזות, אובייקטים ובויליאנס נלמד כיצד להעביר פונקציות.

1. נסיף את הפונקציה לקונטロולר.

```
myApp.directive("movieResult", function(){
    return {
        templateUrl: 'directives/movieresult.html',
        scope : {
            movieObject: "=",
            formatMovieTitleFunction: "&"
        }
    }
});
```

4. בטמפליט של ה-directive, נשלב את המחרוזת שמחזירה הפונקציה. נעביר לפונקציה מפת אובייקטים שmapsה על הפורטור שאמ מיפוי לו בפונקציה (movie) (movieObject).

```
myApp.controller("mainCtrl", function($scope){
    $scope.movie = {
        name: 'Captain America',
        year: '2016'
    }

    $scope.formatMovieTitle = function(movie){
        return movie.name + ' ' + movie.year;
    }
});
```

2. נגדיר את הפונקציה שהוציים להעביר בטמפליט הראשי.

```
< div>
    <h4>{{ movieObject.name }}</h4>
    <p>{{ formatMovieTitleFunction({movie:
        movieObject}) }}</p>
< /div>
< hr />
```

```
< div movie-result
      movie-object="movie"
      format-movie-title-function="formatMovieTitle(movie)">
</div>
```

3. נגדיר בקוד ה-element directive שאחנו מעבירים פונקציה באמצעות &.

## יכיז להציג מספר תוצאות במקומ רק אחת?

עד כה למדנו כיצד להחליף את האלמנט רק פעם אחת, אבל מה קורה כאשר צריכים להציג מספר אובייקטים ולא רק אחד (בדוגמה זו, נרצה להציג מספר סרטים במקומות אחד).

1. נגדיר בקונטロולר, מערך של אובייקטים בשם movies, והוא כולל מידע על מספר סרטים.

```
myApp.controller("mainCtrl", function($scope){
    $scope.movies = [
        {
            name: 'Captain America',
            year: '2016'
        },
        {
            name: 'Mad Max',
            year: '2015'
        },
        {
            name: 'Godzilla',
            year: '2014'
        }
    ]

    $scope.formatMovieTitle = function(movie){
        return movie.name + ' ' + movie.year;
    }
});
```

2. בטמפליט הראשי נשתמש repeat-ng כדי להציג את האלמנט כמה פעמים שציריך.

```
< div ng-repeat="movie in movies">
    < div movie-result
          movie-object="movie"
          format-movie-title-
          function="formatMovieTitle(movie)"> < /div>
    < /div>
```

שם מדריך על custom directives באנגלית אין יכול להיות שלם בלי למד על link directive רצה על כל האלמנטים שבתוך-h-templat. שינויים כוונקציה העברת מידע באמצעות משתנים והרצת פונקציות.

את הפונקציה link מעבירים כואפשרית לקוד-h-directive, ובתוך-h-link ניתן לבצע פעולה נוספת.

בדוגמה זו, נריץ בתוך הפונקציה link קוד שמוסיף קליאס לאלמנט רק במידה ושם הסרט הוא Mad Max.

### מה זה transclusion וכיצד להשתמש בה?

transclusion זו חתיכת מילה, אבל המשמעות היא פשוטה למדי, והוא שnitן לכלול טמפליט אחד בתוך טמפליט אחר.

בדוגמה הבאה, נוסיף לתצוגה אלמנט נוסף באמצעות .transclusion.

נתחיל מזה שנוסיף טקסט ישירות לתוך הטעמפליט הראשי שנמצא ב-.index.php. למרות שהוספנו את הטקסט לתמפליט לא נראה אותו מפני שככל האלמנט מוחלף בcustom directive על ידי אנגלול.

```
<div ng-repeat="movie in movies">
  <div movie-result
    movie-object="movie"
    format-movie-title-
    function="formatMovieTitle(movie)">
      * Not comprehensive.
    </div>
  </div>
```

2. בטמפליט של-h-transclusion ניצור אלמנט שייכל את-h-transclusion.

```
<div>
  <h4>{{ movieObject.name }}</h4>
  <p>{{ formatMovieTitleFunction({movie:
  movieObject}) }}</p>
  <p><ng-transclude></ng-transclude></p>
</div>
<hr />
```

```
myApp.directive("movieResult", function(){
  return {
    templateUrl: 'directives/movieresult.html',
    scope : {
      movieObject: "=",
      formatMovieTitleFunction: "&"
    },
    link: function(scope, elements, attrs){
      if(scope.movieObject.name == 'Mad Max'){
        elements.addClass("red");
      }
    }
  });
});
```

כדי שנוכל לצפות באלמנט שהוספה, علينا לבצע את שתי הפעולות הבאות:

1. ב-h-transclude directive true

```
myApp.directive("movieResult",function(){
  return {
    templateUrl: 'directives/movieresult.html',
    scope : {
      movieObject: "=",
      formatMovieTitleFunction: "&"
    },
    link: function(scope,elements,attrs){
      if(scope.movieObject.name == 'Mad Max'){
        elements.addClass("red");
      }
    },
    transclude: true
  };
});
```

## כיצד לשתף מידע באפליקציה אングולרית באמצעות custom services

מחבר: יוסי בן הרוש בתאריך: 28.10.2016

מדרך זה מסתמך על המדריך המקורי בסדרה, בו הסבכנו כיצד לכתוב אפליקציה מבוססת דף ייחד באמצעות AngularJS.

במדריך המקורי בסדרה, ראיינו שניתן לחלק את האפליקציה שלנו למספר קונטロלים, מה שיוצר בעיה כאשר נשאנו רצים לשתף קוד בין הקונטロלים כי לקונט롤ר אחד אין גישה לקוד בקונטROLר אחר.

הדרך הפשוטה ביותר שמאפשרת לנו לשתף קוד בין הקונטロלים היא באמצעות custom service-ה שימוש ב-

כדי ליצור custom service נשתמש בתחביר הבא:

```
myApp.service('serviceName', function(){
    // the code for the custom service, variables, functions
});
```

את ה-service אנחנו יוצרים על ידי שימוש בפונקציה האングולרית service, על האפליקציה הקיימת (App) עוז בדוגמה).

ה-service מקבל שני פרמטרים:

- שם ה-service
- פונקציה אטומית שמכילה את הקוד שאנו רוצים לשתף בין הקונטロלים.

הקוד ב-service יכול לשנותם ופונקצייתו שאנו רוצים לשתף בין הקונטロלים.

### צעד 1: הגדרת האפליקציה

נכתב את האפליקציה 'myApp', ונעביר את התלות ngRoute ( אם משה לא מובן נא לקרוא את המדריך בנושא אפליקציה אングולרית מבוססת דף ייחד ).

```
var myApp = angular.module('myApp', ['ngRoute']);
```

```
myApp.config(function($routeProvider) {

    $routeProvider.
    when('/first', {
        templateUrl: 'pages/first.html',
        controller : 'firstController'
    }).
    when('/second', {
        templateUrl: 'pages/second.html',
        controller : 'secondController'
    }).
    otherwise({
        redirectTo: '/first'
    });
});
```

## צעד 3: הקונטROLLERים

29

```
myApp.controller('firstController', function($scope,  
guessNumberService){  
    $scope.num = null;  
    $scope.activityState = false;  
  
    ($scope.init = function(){  
        $scope.num =  
guessNumberService.getUserSelectedNumber();  
        $scope.activityState =  
guessNumberService.getActivityState();  
    })();  
  
    $scope.chooseNumber = function(int, e){  
        e.preventDefault();  
  
        $scope.activityState = true;  
        $scope.num = int;  
  
        guessNumberService.setActivityState(true);  
  
        guessNumberService.setUserSelectedNumber(int);  
    }  
});
```

- הكونטROLLER הראשון מטפל בבחירה המשתמש.
- נעביר לו את ה-service שניצור בהמשך המדריך בתור פרמטר.
- המשתנה now יוכל את הערך רק בתנאי שהמשתמש בחר אחד המספרים.
- הfonקציה chooseNumber משנה את הערך של המשתנה now בהתאם לבחירת המשתמש, ובתגובה להקלקה של המשתמש על בחרתו.

- הfonקציה init קוראת לעצמה, ומפעלת בכל פעם שנכנסים או חוזרים לkonטROLLER, מציבה את ערך המשתנים, בהתאם לעריכים השמורים ב-.service.

הكونטROLLER השני אומר למשתמש אם המספר שבחר הוא המספר הנכון באמצעות .service isTheNumber מה-

```
myApp.controller('secondController', function($scope,  
guessNumberService){  
    $scope.result =  
guessNumberService.isTheNumber();  
});
```

## צעד 4: ה-custom service

ב哀יל'קציה שניצור במדריך נחק ב邏יק הינו שים. בדף הראשון, המשתמש יבקש לבחור באחד המספרים 1 עד 3, ובדף השני הוא יוכל לגלוות האם הינו שולחן.

זה שהוא אומר ברמת הקורס זה צריך ל Hebrew את המידע אודוות בחירת המשתמש מהkonטROLLER הראשוני לשני. כדי לשתוף קוד בין הקונטROLLERים נשימוש ב-.custom service.

נוסיף לקוד guessNumberService custom service

```
myApp.service('guessNumberService', function(){  
});
```

```

myApp.service('guessNumberService', function(){
    var thisController = this;

    this.number = 3;
    this.userSelectedNumber = null;
    this.activityState = false;

    this.setUserSelectedNumber = function( number ){
        var number = parseInt(number, 10);

        if( !isNaN(number) )
            thisController.userSelectedNumber =
number;
    };

    this.getUserSelectedNumber = function(){
        return this.userSelectedNumber;
    };

    this.isTheNumber = function(){
        return (thisController.number ===
this.userSelectedNumber)?
'זה המספר שחשבתי עליי'
:
'זה לא המספר. נסה שנית';
    };

    this.setActivityState = function(bool){
        this.activityState = bool;
    };

    this.getActivityState = function(){
        return this.activityState;
    };
});

```

- לתוך ה-service, נוסיף את המשתנים והפונקציות שאנו ממעוניינים לגשת אליהם מכל מקום, ובתנאי שמדובר באוטה האפליקציה.
- המשתנה number שמכיל את המספר, שאותו נגדיר מראש, והוא המשתמש צריך לנחש.
  - המשתנה userSelectedNumber, שיעודן במספר שהמשתמש ניחש.
  - הפונקציה setUserSelectedNumber שמקבלת את הערך שהמשתמש בחר. הפונקציה מציבה את הערך לערך המשתנה userSelectedNumber, רק בתנאי שמדובר במספר תקין.
  - והפונקציה isTheNumber שבודקת האם, בפועל, המשתמש ניחש את המספר הנכון. היא משווה את הערך שהמשתמש לערך שהוגדר מראש, ומחזירה הודעה שניית להציג למשתמש.

שים לב, לשימוש ב-this וב-thisController.

הפונקציה (isTheNumber) ומושדרים עם מילת המפתח this, והמשתנה (number) מוגדרים עם מילת המפתח this, המשיכת אותם לפונקציה שבתוכה הם נמצאים (שזה הקונטROLLER).

בתוך הפונקציה (isTheNumber) אנחנו צריכים גישה ל-number שמוגדר בקונטROLLER. אם ננסה לגשת אליו באמצעות this, נתקל בעיה, מפני ש-this בתוך הפונקציה 'չבע' על הפונקציה, ולא על הקונטROLLER.

כדי לפתור את הבעיה, יצרנו את המשתנה thisController, שמקבל את הערך של this שמצויב על הקונטROLLER. ואז, בתוך הפונקציה, אנחנו משתמשים ב- thisController כדי להציג על המשתנה number שמוגדר על הקונטROLLER.

הטעמפליט הרשמי ב-[index.html](#) יכול את הדבר שאנו תוכנו נעדכן עם הטעמפליטים המשניים.

```
<h1></h1><h1>בבקשה למחוש את המספר</h1>

<ul class="pagination">
<li><a href="" ng-class="{active:(num==1 && activityState==true)}"
ng-click="chooseNumber(1, $event)">1</a></li>
<li><a href="" ng-class="{active:(num==2 && activityState==true)}"
ng-click="chooseNumber(2, $event)">2</a></li>
<li><a href="" ng-class="{active:(num==1 && activityState==true)}"
ng-click="chooseNumber(1, $event)">3</a></li>
</ul>

<p><a href="#/second">האם אתה זודק בניחוש?</a></p>
```

```
<div ng-view></div>
```

בסוף, ישנו שני טמפליטים משניים, כפי שהגדרנו בצעד השני.

### [pages/first.html](#)

בטמפליט זה המשתמש בוחר אחד משלושה מספרים, באמצעות לחיצה על הקישוריהם. הקישורים מכילים שתי הוראות:

- לחיצה על הקישור מעבירה לפונקציה `chooseNumber` את המספר הנבחר באמצעות `click`-`ng`.
- ה קישור מקבל כלואו `active` באמצעות `ng-class`.

### [pages/second.html](#)

בטמפליט השני, כל מה שנותר לעשות זה להציג את התוצאה.

```
<h1>התוצאה: {{ result }}</h1>
```

## 1. נגזר את האפליקציה והקונטROLLER

לאפליקציה נקרא app myApp, ונגזר אותה על התגים הפותחת של ה-.html, .html, ואת הקונטROLLER נגזר על הדיב שיעטוף את התצוגה.

```
<html ng-app="myApp">
<head>
  <title>AngularJS filters</title>
  <script
src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.9/angular.min.js"></script>
</head>
<body>
  <div ng-controller="carsCtrl">

  </div>
</body>
</html>
```

## 3. נציג את המידע מהמערך בתוך טבלה שתוצג בחלק הטעמ"ל

נזהור להטעמ"ל ונציג את המערך מהסעיף הקודם, בתוך טבלה הסטמ"ל. לשם כך, נעביר על פריטי המערך, פריט-פריט באמצעות repeat-ng, ונDéפֵס את הפריטים בתוך תיבות הטבלה.

```
<div ng-controller="carsCtrl">

<h3>Cars List</h3>

<table>
<thead>
  <tr>
    <th>Model</th>
    <th>Price</th>
    <th>Country</th>
    <th>Date</th>
  </tr>
</thead>
<tbody>
  <tr ng-repeat="car in cars">
    <td>{{ car.model }}</td>
    <td>{{ car.price }}</td>
    <td>{{ car.country }}</td>
    <td>{{ car.date }}</td>
  </tr>
</tbody>
</table>

</div>
```

# מדריך: פילטרים של AngularJS

מחבר: יוסי בן הרוש בתאריך: 16.07.2016

פילטרים (מסנן) של AngularJS משמשים אותנו כדי לסנן מידע וכי לפרט מידע (להציג את המידע בצורה הרצויה לנו). ה필טרים מתחלקים לשניים. פילטרים שהפדר'מוורק מספק לנו, ופילטרים מיוחדים שאנו חנו כתובים במידת הצורך.

## 2. נתחילה לכתוב את הסקריפט האנגולרי

בסקריפט האנגולרי, נגזר בשלב זה שני מרכיבים. המודול והקונטROLLER (את שניהם נגזר לפי קוד הטעמ"ל שככינו למטה). בתוך הקונטROLLER כתוב את המערך cars שמכיל מידע עבור המודלים השונים של המכוניות.

```
<script>
// Module
var myApp = angular.module('myApp', []);

// Controller
myApp.controller('carsCtrl', function($scope) {

  $scope.cars = [
    {model:'BMW',country:'Germany',price:160000,date:'2015-11-11'},
    {model:'Tesla',country:'USA',price:180000,date:'2014-10-08'},
    {model:'Jaguar',country:'GB',price:210000,date:'2016-09-01'},
    {model:'Audi',country:'Germany',price:150000,date:'2016-04-01'},
    {model:'Bentley',country:'USA',price:190000,date:'2014-12-28'},
    {model:'Ferrari',country:'Italy',price:1000000,date:'2016-07-16'}
  ];
});
```

## 5. נפעיל פילטרים ברמת הרשימה

נפעיל על הרשימה מספר פילטרים במטרה לברור אילו פריטים יוצגו, ומה הסדר שבו יוצגו.

בדוגמא הבאה נבחר להציג את המכוניות שמייצירות בארה"ב ונסדר את התאריכים בסדר יורד ואות הפוחרים בסדר עולה:

```
<tr ng-repeat="car in cars | orderBy: 'date' | orderBy: '-price' |
filter: { 'car.country': 'USA' }>
  <td>{{ car.model | uppercase }}</td>
  <td>{{ car.price | currency:'₪':0 }}</td>
  <td>{{ car.country | limitTo:3 | lowercase }}</td>
  <td>{{ car.date | ilDateFormat }}</td>
</tr>
```

נפעיל על פרטי הרשימה מספר פילטרים שיישנו את האופן (הפורם) שבו מוצגים הפריטים:

- uppercase - הופך את כל האותיות הקטנות לגדיות, רלבנטי באנגלית
- lowercase - הופך את כל האותיות הגדימות לקטנות, באנגלית
- limitTo - מגביל את מספר התווים בטקסט לפי הפורמט שמעבירים לפילטר.
- currency - מציג מספרים בפורמט של מטבע כולל סימן ה-\$, הפסיקים שמספריים בין האלפים, וערכיהם עשרוניים. ניתן לשנות את המטבע ואת מספר הספרות העשרוניות אם מעבירים פרמטרים נוספים.

```
<tr ng-repeat="car in cars">
  <td>{{ car.model | uppercase }}</td>
  <td>{{ car.price | currency:'₪':0 }}</td>
  <td>{{ car.country | limitTo:3 | lowercase }}</td>
  <td>{{ car.date }}</td>
</tr>
```

הפילטר filter מקבל מחוזת חסן שמכילה את הקriterיוונים לסינון אבל אם רצים סינון קצתי יותר מתקדם עדיף להשתמש ב-if-ng.

לדוגמא, מעוניינים לקבל מכוניות שעלוותם גבוהה מ-150000 ובתנאי שיוצרו בגרמניה או בארה"ב.

לעשות ש"ח

```
<tr ng-repeat="car in cars"
  ng-if="car.price > 150000 && (car.country == 'USA' ||
car.country == 'Germany')>
  <td>{{ car.model | uppercase }}</td>
  <td>{{ car.price | currency:'₪':0 }}</td>
  <td>{{ car.country | limitTo:3 | lowercase }}</td>
  <td>{{ car.date | ilDateFormat }}</td>
</tr>
```

## 6. פילטרים שאנו חזו כותבים

את הפילטר נשים לאפליקציה שלנו, ונעביר לו פונקציה אמצעית (שבמקורה זה מקבלת את השירות האנגולרי `$filter` כפרמטר).

לא תמיד הפילטרים שמספק Angular מסוימים לנו, ובמקרים אלה נוכל לכתוב את הפילטרים בעצמנו.

```
<script>
// Custom filter
myApp.filter('ilDateFormat', function ($filter){
  return function(text){
    var dateTxt = text.replace(/\-/g, "/");
    var dateTmp = new Date(dateTxt);

    return $filter('date')(dateTmp, "dd/MM/yyyy");
  }
});
</script>
```

הפונקציה שמחזיר הפילטר, משתמש בביטוי רגולרי כדי להחליף את הקווים המפרדים לקווי חיצונים, ולאחר מכן הופכת את המחרוזת שנמקבלת בתהליך לאובייקט תאריך, ולבסוף מוחזירה את התאריך במבנה הרצוי.

לדוגמה, פילטר ששמו `DateFormat` שהופך את פורמט התאריך מאמריקאי ליישראלי:

```
<html ng-app="myApp">
<head>
<title>AngularJS filters</title>
<script
src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.9/angular.mi
n.js"></script>
</head>
<body>
<div ng-controller="carsCtrl">

<h3>Cars List</h3>

<table>
<thead>
  <tr>
    <th>Model</th>
    <th>Price</th>
    <th>Country</th>
    <th>Date</th>
  </tr>
</thead>
<tbody>
  <tr ng-repeat="car in cars" ng-if="car.price > 150000 &&
(car.country == 'USA' || car.country == 'Germany')>
    <td>{{ car.model | uppercase }}</td>
    <td>{{ car.price | currency:'₪':0 }}</td>
    <td>{{ car.country | limitTo:3 | lowercase }}</td>
    <td>{{ car.date | ilDateFormat }}</td>
  </tr>
</tbody>
</table>
</div>
```

```
<script>
// Module
var myApp = angular.module('myApp', []);

// Filter
myApp.filter('ilDateFormat', function ($filter){
  return function(text){
    var dateTxt = text.replace(/\-/g, "/");
    var dateTmp = new Date(dateTxt);

    return $filter('date')(dateTmp, "dd/MM/yyyy");
  }
});

// Controller
myApp.controller('carsCtrl', function($scope) {

  $scope.cars = [
    {model:'BMW',country:'Germany',price:160000,date:'2015-11-11'},
    {model:'Tesla',country:'USA',price:180000,date:'2014-10-08'},
    {model:'Jaguar',country:'GB',price:210000,date:'2016-09-01'},
    {model:'Audi',country:'Germany',price:150000,date:'2016-04-01'},
    {model:'Bentley',country:'USA',price:190000,date:'2014-12-28'},
    {model:'Ferrari',country:'Italy',price:1000000,date:'2016-07-16'}
  ];
});

</script>
</body>
</html>
```

**הקוד המלא**

## טפויים בסגנון אנגולרי

מחבר: יוסי בן הרוש בתאריך: 24.06.2016

אתה הדריכם לתקשר עם המשתמשים באתר היא באמצעות טפויים. Angular מציע חווית משתמש מושפרת למשתמשים הודות לחוויה בזמן אמת, שיכל להציג למשתמש הוראות והודעות שגיאה בזמן שהוא מלא את הטופס, והודות לאפשרות למנוע את שליחת הטופס במידה והשודות לא מולאו כהלכה.

הטופס

הוסף אדם חדש לרישומה.

שם פרטי:

שם ח"ב להקל לפחות 2 אותיות

שם משפחה:

שם המשפחה צריך להקל לפחות 2 אותיות

כתובת:

שם העיר והרחוב בעברית בלבד

גיל:

נא למחור  הרשמה לניוזלטר

**AngularJS**  
by Google

### הكونטROLLER

הטופס תחום בטען קונטROLLER בשם `userController`.

### הטופס

הטופס נשלח לפונקציה `saveUser`, ומכל את האובייקט `userData`, שמכיל את המידע מהשודות בטופס.

```
<div ng-controller="userController">
</div>
```

```
<div ng-controller="userController">
  <form name="userForm" ng-
submit="saveUser(userData)">
    </form>
</div>
```

### ההודעה למשתמש

ההודעה מקבלת את הערך מהמשנה `msg` שיש לו-

הקלואס מוכתב על ידי המשנה `msgType`

```
<div ng-controller="userController">
<p ng-if="showMsg" class="alert" ng-class="{'alert-dnger':
msgType == 'danger', 'alert-success': msgType == 'success'}">
{{msg}}</p>
</div>
```

השדות ממלאים את האובייקט userData בגלל שהם מוחווים את המודול. לדוגמה,  
.firstName

```
<input type="text" name="firstName" ng-
model="userData.firstName"/>
```

נוסף לשדה הודעתת שגיאיה. מציין שהמשתמש ערך את השדה, והתנאי אומר שם המשתמש ערך את השדה, ולא מתקין התמי, אך יציג תוקן הודעתת השגיאיה.

```
<input type="text" name=="firstName"
autocomplete=="false" ng-model=="userData.firstName" ng-
required=="true" ng-minlength=="2" />
```

בاقפן דומה, נוסיף את שאר השדות בטופס.

```
<!-- Pass the user data to the function saveUser() -->
<form name="userForm" ng-submit="saveUser(userData)">

<div>

    <label>שם פרטי</label>
    <br />

    <!-- The value is required and it must have at least 2
characters -->
    <input type="text" name="firstName" ng-
model="userData.firstName"
    ng-required="true" ng-minlength="2" />

    <!-- $dirty is set to true if a user edited the input, then
we check if the field has the required error -->
    <span class="error-message" ng-
show="userForm.firstName.$dirty
    && userForm.firstName.$error.required">
        נא להזין שם פרטי
    </span>

    <!-- if the element was edited we check if it contains
at least 2 characters -->
    <span class="error-message" ng-
show="userForm.firstName.$dirty
    && userForm.firstName.$error.minlength">
        השם חייב להכיל לפחות 2 אותיות
    </span>

</div>
```

```
<div>

    <label>שם משפחה</label>
    <br />

    <input type="text" name="lastName" ng-
model="userData.lastName"
    ng-required="true" ng-minlength="2" />

    <span class="error-message" ng-
show="userForm.lastName.$dirty
    && userForm.lastName.$error.required">
        נא להזין שם משפחה
    </span>
```

```
<span class="error-message" ng-
show="userForm.lastName.$dirty
    && userForm.lastName.$error.minlength">
        שם המשפחה צריך להכיל לפחות 2 אותיות
    </span>
```

```
</div>

<div>

    <label>כתובת:</label>
    <br />
```

</div>

```

<!-- regex validation. only certain characters are
allowed -->
<input type="text" name="address" ng-
model="userData.address"
ng-required="true" ng-minlength="6" ng-pattern="/^[\u05d0-\u05e9]*
{2,}[\u05d0-\u05e90-9,- ]+$/i />

<span class="error-message" ng-
show="userForm.address.$dirty
&& userForm.address.$invalid">שם העיר והרחוב בעברית</span>
בבלר

</div>

<div>

  <label>גיל</label>

  <br />

  <select name="age" ng-model="userData.age" ng-
init="ages[0]" ng-options="option.name for option in ages">
</select>

</div>

<div>

  <label>ההרשמה לניוזלטר</label>

```

```

<!-- The value will change according to the choice of
the user -->
<input type="checkbox" name="subscribe" ng-
model="userData.subscribe" ng-true-value="yes"/>

</div>

<div>

  <!-- The button will be disabled until every input in the
form is valid -->
<input type="submit" value="המשך" ng-
disabled="userForm.$invalid"/>

</div>

</form>

```

בסוף הינה ליר מלאים הودעה שנותצג למשתמשים, ומציגים את ההודעה מעל לטופו.

```
var app = angular.module('myApp', []);

app.controller('userController', function($scope) {

    $scope.showMsg = false;
    $scope.msg      = "";

    // list of options for the select list.
    $scope.ages = [
        {value: 1, name: "18- מתחת ל-18"},
        {value: 2, name: "18- מעל ל-18"}
    ];
    // array of users
    $scope.users= [
        {
            firstName: "זאביק",
            lastName: "השועל",
            address: "הרכל 13 מוה עצילות",
            subscribe: "yes",
            age: $scope.ages[0] // the first item in the list
        }
    ];

    // function to save the user
    // by adding the new user to the users array
    $scope.saveUser = function(userData) {
        if($scope.userForm.$valid) {
            $scope.users.push({
                firstName:
                    userData.firstName,
                lastName:
                    userData.lastName,
                address: userData.address,
                subscribe:
                    userData.subscribe,
                age: userData.age
            });

            $scope.showMsg = true;
            $scope.msg      = 'המשתמש נשמר';
            $scope.msgType = 'success';

        } else {
            $scope.showMsg = true;
            $scope.msg      = 'בעה בשמירת המשתמש';
            $scope.msgType = 'danger';
        }
    }
});
```

כששולחים את הטופס, המידע מועבר כאובייקט לפונקציה `saveUser`, ומתווסף למשרket `users`.

```
var app = angular.module('myApp', []);

app.controller('userController', function($scope) {
    // array of users
    $scope.users= [
        {
            firstName: "זאביק",
            lastName: "השועל",
            address: "הרכל 13 מוה עצילות",
            subscribe: "yes",
            age: $scope.ages[1]
        }
    ];

    // function to save the user
    // by adding the new user to the users array
    $scope.saveUser = function(userData) {
        if($scope.userForm.$valid) {
            $scope.users.push({
                firstName:
                    userData.firstName,
                lastName:
                    userData.lastName,
                address: userData.address,
                subscribe:
                    userData.subscribe,
                age: userData.age
            });

            $scope.showMsg = true;
            $scope.msg      = 'המשתמש נשמר';
            $scope.msgType = 'success';

        } else {
            $scope.showMsg = true;
            $scope.msg      = 'בעה בשמירת המשתמש';
            $scope.msgType = 'danger';
        }
    }
});
```

את שדה הבחירה ages בטופס מלאים מהמערך .ages.

```
// list of options for the select list.  
$scope.ages = [  
    {value: 1, name: " מתחת ל-18"},  
    {value: 2, name: " מעל ל-18"}  
];
```

## שאלות ראיון

### - ? Angular: מהו \$ scope בJS

תשובה: ב AngularJS - הוא אובייקט שמתיחס למודל יישום. זהו אובייקט הקשור לתצוגה (אלמנט DOM) עם הבקר. בברק, ניתן לגשת לנוטוני המודול באמצעות אובייקט AngularJS \$ scope. תומך בתבנית \*, MV אובייקט \$ scope הופך למודל של \*. MV

### - ? Angular: מהו SPA (ישום עמוד יחיד) בJS

תשובה: יישומי עמוד יחיד (SPA) הם יישומי אינטרנט הטוענים דף HTML יחיד ומעדכנים את הדף באופן דינמי כאשר המשתמש מקיים אינטראקציה עם האפליקציה.

SPA משתמשים ב AJAX - וב HTML - ליצירת אפליקציות אינטרנט קולחות ומגיבות, ללא טעינה מתמדת של עמודים. עם זאת, המשמעות היא שחלק ניכר מהעבודות מתרחשות **בצד הלוקה**, ב JavaScript.

דף HTML יחיד פירשו בכך תגובת ממשק משתמש מהשרת. המקור יכול להיות ASP, ASP.NET, ASP.NET MVC, JSP וכן הלאה.

עם זאת, יישום אינטרנט של עמוד יחיד מועבר כדף אחד לדפפן ובדרך כלל אינם דורש טעינה חדש של הדף כאשר המשתמש מנווט לחלקים שונים ביחסום. התוצאה היא ניוט מהיר יותר, העברות רשות **יעילות יותר** וביצועים כוללים טובים יותר עבור משתמש הקצה.

### שאלת: מהי הזרקת תלות?

תשובה: הזרקת תלות (DI) היא **מבנה עיצוב תוכנה** העוסקת באופן שבו רכיבים משיגים תלותם של עצמם. לתה המערכת של מזرك AngularJS אחראית על **יצירת רכיבים**, **פתרון התלות** שלהם ומסירתם לרכיבים אחרים כנדרש.

### - ng-show / ng-hide בין If-else

תשובה: ההנחה If-else הופכת רק אלמנט DOM אם התנאי נכון. בעוד שההנחה ng-show / ng-hide הופכת את אלמנט DOM - אך משנה את המעמד של ng-show / ng-hide כדי לשמר על גראות האלמנט בדף.

## - שאלה: מהי ניתוב ב AngularJS ?

תשובה: **ניתוב** הוא **תוכנת ליבה** ב AngularJS - תכונה זו **שימושית לבניית SPA** ("שימושי לעמוד יחיד") עם מספר תצוגות. ב, **SPAs** - **כל התצוגות הן קבצי HTML** שונים ואנחנו משתמשים לבנייתן כדי **לטעתן חלקים** שונים ביישום.

כדי **חלוקת** את ה**הישום** באופן הגיוני ולהפוך אותו **לניהול**. במלילים אחרות, **ניתוב עוזר לנו לחלק** את ה**הישום** שלנו **لتצוגות** **לוגיות** ול**אגד** אותם **עם** **בקרים** שונים.

## - **ng-repeat** הסבר הוראה

תשובה: ההנחיה **ng-repeat** היא המאפיין הנפוץ ביותר ב AngularJS - זה **chodur ul** אוסף פריטים ויוצר אלמנטים של **DOM** זה **מפקח כל הזמן על** מקור הנתונים כדי **לעבד מחדש** תבנית בהתאם לשינוי.

## ? AngularJS פסק זמן עם **setTimeout**

תשובה: העטיפה של **AngularJS** על **window.setTimeout** אתה **מבטל** פסק זמן המפעיל את **הfonkzia**:

```
$timeout.cancel(function (){ // write your code. });
```

## - **ng-App** הסבר את הנחיתת ה **app**

תשובה: הוראת ה **ng-app** - **פעילה** **הישום** AngularJS הוא מגדיר את יסוד השורש. זה מאותחל אוטומטית או **מאתחל** את **הישום** כאשר **דף אינטרנט** המכיל את **הישום** AngularJS נטען.

הוא משמש גם **לטעינת מודולי** AngularJS שונים **בשימוש**.

## שאליה: הסבר את ההנחיה **init**

תשובה: הוראת **ng-init** מאותחתת את **נתוני** **הישום** AngularJS הוא **משמש** להצבת ערכים למשתנים שישמשו ביישום.

## שאלה: איך אתה משותף נתונים בין בקרים?

תשובה: צור שירות **AngularJS** שיחזק את הנתונים ויזרייק אותם לבקרים. **שימוש** בשירות הוא **הדרך הנקייה**, המהירה והקלת בioter לבדיקה.

עם זאת, ישן כמה  **דרכים אחרות** לישם שיתוף נתונים בין בקרים, כמו:

**שימוש באירועים**

שימוש ב **\$ parent, nextSibling, controllerAs** - וכן הלאה כדי **לגשת שירות לבקרים**

שימוש ב **\$ rootScope - כדי להוסיף את הנתונים עליהם** (לא נוהג טוב)

## - ng-show / hide בין ההוראות if- ng

תשובה **if-ng** : תיצור ותציג את אלמנט ה DOM רק כאשר מצבו נכון. אם התנאי שקרי או משתנה לשקר הוא **לא יוצר או יחרוס את המצב הנוכחי**.

**ng-show / hide** תמיד ייצור את אלמנט DOM אך הוא **יחיל את מאפיין התצוגה** של CSS **בהתבסס על הערכת המצב**.



## שלום עולם Angular

מחבר: יוסי בן הרוש בתאריך: 07.05.2021

**Angular** היא פרויקט שמיועד לרווח בעקבות הלקוח של אתרי אינטרנט. המערכת מפותחת על ידי Google, והוא מחליף את **AngularJS**.



Angular נכתבת מהיסוד, והיא שונה מאוד מ-**AngularJS**, ומציעה שיפור ניכר ביצועים, ותמכה טובה יותר בניידים. אחד השינויים המרכזים הוא שאט הקוד אנחנו כתבים בשפת **TypeScript**, ולא **JavaScript**. **TypeScript** מומדריכים באתר רשותן.

\$ ng new hello-world

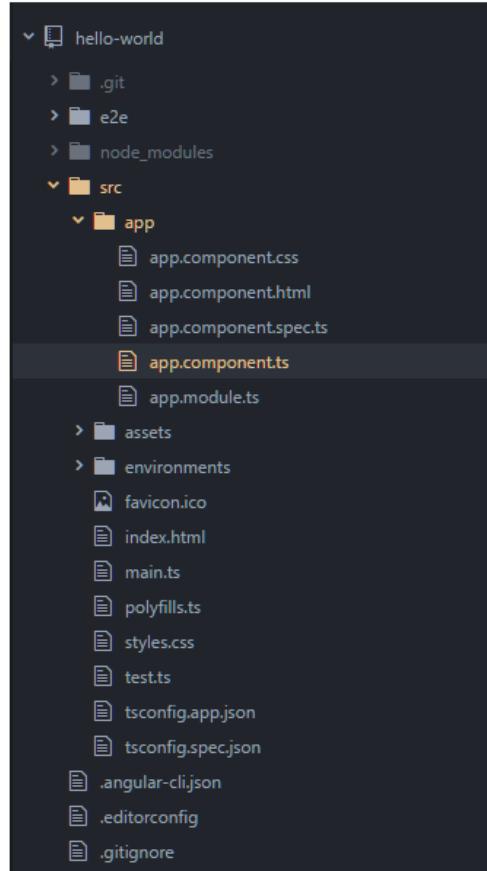
\$ ng serve

--open

### מבנה הקבצים והתיקיות של הפרויקט

נשתמש ב-IDE כדי לפתח את תיקיית הפרויקט. ננות לתיקייה **src**, שהיא התיקייה שבה נמצאים קבצי **TypeScript**, ובתוכה נמצאים רוב הקבצים שדרושים לנו לפיתוח הפרויקט.

זה מבנה התקיות שאתם אמורים לראות באפליקציה:



הקבצים הכלליים שאוטם צריך להכיר יושבים בתיקייה **src**:

- **src/index.html** – הוא קובץ HTML המרכזי המשותף לכל מרכיבי האפליקציה.

- **src/styles.css** – הוא קובץ ה- CSS המשותף לכל מרכיבי האפליקציה.

האפליקציה מתחלקת לרכיבים שמכונים קומפוננטות (**component**) ובשלב זה יש לנו קומפוננטה אחת של כל הקבצים שלה נמצאים בתיקייה **src/app**.

הקומפוננטה זו היא הרכיב המרכזי באפליקציה, וכל יתר הקומפוננטות משובצות לתוכה, באופן ישיר או עקיף.

אילו הקבצים שהכי חשוב להכיר בתיקייה **src/app**, ש כאמור, מכילה את הקומפוננטה המרכזית של האפליקציה.

- **src/app/app.module.ts**

הוא המודול המרכזי האפליקציה, ובתוכו נגדיר את המודולים והקומפוננטות בהם משתמש.

- **src/app/app.component.ts**

מכיל את הקוד עבור הקומפוננטה המרכזית של האפליקציה.

- **src/app/app.component.html**

בתוכו נמצא **html** של כל האפליקציה מכיוון שלתוכו נשבץ את **html** של יתר הקומפוננטות בצורה שתציג שיעיגו את הקומפוננטות השונות.

## נערוך את הקוד

נפתח לעריכה את הקובץ `app.component.ts`, שזה תוכנו.

`src/app/app.component.ts`

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

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

`src/app/app.component.html`

```
<h1>
  {{title}}
</h1>
```

השדה `title` מהמחלקה `AppComponent` מגרדר לתוך הסוגרים המסתוללים.  
בדוק שזה אכן כך על ידי שינוי ערך השדה `title` במחלקה `AppComponent`:

`src/app/app.component.ts`

מה שלנו חשוב בשלב זה של הלימוד זה שבשורזה השניה הנקורטור `Component` (ואם אתם לא יודעים מהذا דקוטורו אותם יכולים לקרוא לו רק `Component`) הוא לא "יעלב" מגדיר את קבצי `html` וה-`css` של הקומפוננטה.

בהמשך הקובץ יש לנו מחלוקת ששם `AppComponent` שמכילה את הקוד  
שנמצא ב-`HTML`. ובשלב זה היא מכילה שדה אחד, השדה `title`.

אבל איפה קובץ `HTML` שבתוכו מוצגת השדה `?title`?

נפתח את קובץ `HTML` שמוגדר ב-`@Component`, ונשנה את תוכן הקובץ כדי  
שירנדר (=יציג בדף) את ערכו של השדה `title`:

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

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

export class AppComponent {
  title = 'Hello world with Angular!';
}
```

ועכשיו, כשובזר לדף, נראה את השינוי. כך זה נראה אצלך:

localhost:4200

**Hello world with Angular!**

## הוספת קומפוננטות באמצעות -cli

קומפוננטות (`component`) הם הדרך העיקרית באנגולר להויסיף אלמנטים ולוגיקה  
לדף.

ניתור את הקומפוננטה הראשונה באמצעות ה-`-cli`, ונקרא לה `my-component`:

נכלייד בטרמינל:

```
$ ng generate component my-component
```

ה-`cli` יוסיף תיקייה לאפליקציה לפי שם הקומפוננטה: `src/app/my-component`.  
התיקייה תכלול 4 קבצים:

- `my-component.component.ts` יכול את הלוגיקה של הקומפוננטה.
- `my-component.component.html` יכול את ה-`html`.
- `my-component.component.css` קובץ CSS-শマשות את האפליקציה.
- `my-component.component.spec.ts` הוא קובץ לבדיקות.

את קוד הקומפוננטה נכתב אל תוך הקובץ `my-component.component.ts`

## src/app/my-component/my-component.component.ts/

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

@Component({
  selector: 'app-my-component',
  templateUrl: './my-component.component.html',
  styleUrls: ['./my-component.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class MyComponentComponent {
  name = 'Israel Israeli';

  sayHello() {
    console.log('Hello, ' + this.name)
  }
}
```

נערוך את קובץ ה-.html של הקומפוננטה ונוסיף כפטור שלחיצה עליו תקרא לפונקציה.

## src/app/my-component/my-component.component.html/

```
<p>
  <button (click)="sayHello()">Say Hello to {{name}}</button>
</p>
```

כדי לשלב את הקומפוננטה באפליקציה צריך ליבא אותה לקובץ המודול של האפליקציה.

- המשתנה name מוגדר בקומפוננטה וקרייה לפונקציה ()sayHello sayHello() תציג בקונסולה של דפדפן כרום את ההודעה.
- שימוש לביטולטור של הקומפוננטה 'app-my-component' כי אנחנו משתמשים בהmarsh כדי לשבץ את הקומפוננטה בקובץ ה-.html הראשי של האפליקציה.

נערוך את קובץ ה-.html של הקומפוננטה ונוסיף כפטור שלחיצה עליו תקרא לפונקציה.

## src/app/app.module.ts/

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

import { AppComponent } from './app.component';
import { MyComponentComponent } from './my-component/my-component.component';

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

- חשוב ליבא את הקומפוננטה גם להוציא אותה ל-declarations.
- אם השתמשנו ב-`cd` כדי להוציא את הקומפוננטה היא תעשה את העבודה בשביבם.

וכדי לצפות בקומפוננטה נשבץ את הסליקטור שלה בקובץ ה-.html של האפליקציה:

**src/app/app.component.html**

```
<h1>  
{{title}}  
</h1>  
  
<app-my-component></app-my-component>
```

כך פועלת האפליקציה מנקודת מבטו של המשתמש. לחיצה על הכפתור (1), תדפיס את ההודעה לקונסולה המפתחים של הדפדפן (2).

- בדף כרום צירוף המفاتחים Ctrl + Shift + F ופתח את הקונסולה של הדפדפן.

## קומפוננטות מקווננות ב-Angular

מחבר: יוסי בן הרוש בתאריך: 27.11.2017

אחרי שבמדריך קודם למדנו ליצור קומפוננטות Angular באמצעות שורת הפקודות (CLI), במדריך זה נלמד להויס קומפוננטות ידנית. כמו-כך, נלמד כיצד匿名 (לשם בטור) קומפוננטה אחת בתוך קומפוננטה אחרת. \* קיון מהמילה `kn` (לציפור). תרגום של nesting.

ב哀פליקציה שנכתוב במדריך:

- ניצור שתי קומפוננטות, `cat` ו-`cats`.
- נקבע את הקומפוננטה `cat` בתוך הקומפוננטה `cats`.
- ונשבץ את הקומפוננטה `cats` בתוך הקובץ הראשי של האפליקציה.

### 1. ניצור אפליקציה חדשה בשם cats

ניצור את האפליקציה מה-CLI כפי שהסבירנו במדריך "שלום עולם".

```
> ng new cats
```

נכון לティקית הפרויקט, ונ裏ץ את האפליקציה

```
> cd cats
> ng serve
```

גלווש לאפליקציה שיצרנו בכתובת

`localhost:4200//`

### 2. עריכה ראשונית של הקוד

שנה את ערכו של המשתנה `title` בקובץ הקומפוננטה הראשית:

`src/app/app.component.ts/`

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

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

נערוך את קובץ `html` של הקומפוננטה. נסיר את כל התוכן שיורד עם החיבור, ונשאר עם הקוד שמודפס את ה-`title`, שאות ערכו

`src/app/app.component.html/`

.`AppComponent` האדרנו במחילה

```
<div style="text-align:center">
<h1>
  Welcome to {{title}}!
</h1>
</div>
```

כך נראה האפליקציה שלנו בדף:

Welcome to cats!

### 3. נסיף ידנית את הקומפוננטה הראשונה שלנו ששםה cat

נסיף ידנית את הקומפוננטה `cat` בטור תי'יה שאגם שם `cat` שאוותה נסיף לטור התיק'יה `src/app/cat`

נסיף לטור תי'יה הקומפוננטה את קבצי `-ts` ו-`-html`:

`src/app/cat/cat.component.ts`  
`src/app/cat/cat.component.html`

נערוך את הקובץ `cat.component.ts`

`src/app/cat/cat.component.ts`

imp

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

@Component({
  selector: 'app-cat',
  templateUrl: './cat.component.html'
})
export class CatComponent {
  petSound = 'meow';
}
```

בשורה הראשונה אנו מיבאים את האובייקט `Component` מליבת אングולר.

בשורה הבאה אנו מגדירים את התכונות של הקומפוננטה באמצעות הדקורטור `@Component`. הגדרות אילו כוללות:

- משמש לדיהו: הקומפוננטה כנסחבע אותה לטור קומפוננטה אחרת. בדוגמה שלנו, הקומפוננטה מזוהה על ידי התגית `selector`

`<app-cat></app-cat>`

בכל מקום שבו נרצה לשbz אותה באפליקציה.

- תהיה תבנית הטעמ"ל שתשתמש את הקומפוננטה. `templateUrl`

שם המחלקה `CatComponent` הוא בהתאם לשם הקומפוננטה ובוסף מוסיפים את המילה `Component`.

בתוך המחלקה מגדירים תכונות ומתחות של המחלקה. במקרה זה, נגידיר את התכונה `petSound`.

```
export class CatComponent {
  petSound = 'meow';
}
```

כדי שנוכל ליצפות בערך המשטנה, נערוך את קובץ הטעמ"ל, ונשבץ את המשטנה בין סוגרים מסווגלים. מה שיגרום לאנגולר לרדדר את התוכן בהתאם לערכו של המשטנה.

`src/app/cat/cat.component.html`

`<p>My cat says {{petSound}}</p>`

## 4. נסיף את הקומפוננטה cats

לפי אוטם שלבים ששימושו אותו להוספה הקומפוננטה cat

[src/app/cats/cats.component.ts/](#)

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

@Component({
  selector: 'app-cats',
  templateUrl: './cats.component.html'
})
export class CatsComponent { }
```

לא לשכוח להוסיף את קובץ האטט"ל של הקומפוננטה

[src/app/cats/cats.component.html/](#)

## 5. נידע את האפליקציה על הקומפוננטות שהוספנו

נידע את האפליקציה על הקומפוננטות שהוספנו זהה שנייה את הקומפוננטות -ts.module.ts app ונוסיף את שם לרשימת ה-

declarations

[src/app/app.module.ts/](#)

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

import { AppComponent } from './app.component';
import { CatComponent } from './cat/cat.component';
import { CatsComponent } from './cats/cats.component';

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

## 6. נשבץ את הקומפוננטה cats בתחום הקומפוננטה app

51

כפי שהסבירנו בתחילת המדריך, ניתן לתקן (nesting) קומפוננטה אחת בתחום קומפוננטה אחרת. ומה שהකוד הבא עשו הוא לפחות את הקומפוננטה cats בתחום הקובץ הראשי של האפליקציה app.

[src/app/app.component.html](#)

```
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!
  </h1>
</div>
<app-cats></app-cats>
```

\* הקומפוננטה cats קיימת בתחום התגית app-cats app לפי שמו של הסלקטור (ראה צעדי מס' 4).

## 7. נשבץ את הקומפוננטה cat בתחום הקומפוננטה cats

[src/app/cats/cats.component.html](#)

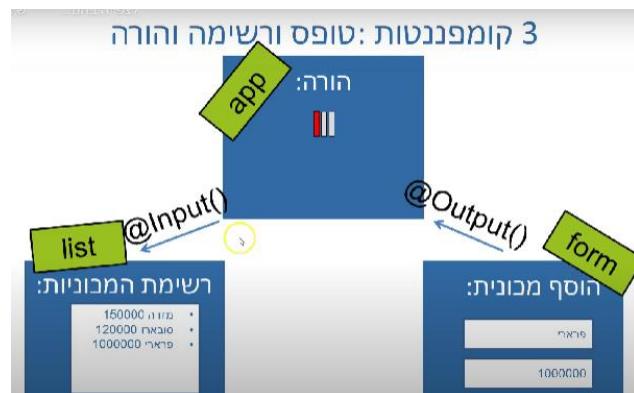
```
<p>
  The component "cats" works!
</p>
<p>And has the following sub components</p>
<app-cat></app-cat>
<app-cat></app-cat>
<app-cat></app-cat>
```

כפי שניתן לראות בדוגמה. ניתן לשבץ את הקומפוננטה יתר פעמיים אחת.

לසיכום זו התוצאה של התרגיל בקנון קומפוננטות שעשינו במדריך

Welcome to cats!

The component "cats" works!  
And has the following sub components  
My cat says meow  
My cat says meow  
My cat says meow



## הקומפוננטה ההורה (app) מכילה את המערך

הקומפוננטה ההורה יושבת בקומפוננטת השורש, כולל את מערך המכוניות (cars).

[src/app/app.component.ts](#)

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})

export class AppComponent {
  cars = [{model:"subaru",price:125000},
          {model:"sussita",price:12000}];
}
```

הערך cars כולל 2 פריטים. כל פריט כולל את שמו ומחירו של מודל מכונית.

## 2. הקומפוננטה הילד, list, מציגה את פרטי המערך

הקומפוננטה list מייבאת כל אחד מפריטי הרשימה בנפרד מההורה, ומציגו אותו ברשימה.

[src/app/list/list.component.ts](#)

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

@Component({
  selector: 'app-list',
  templateUrl: './list.component.html'
})

export classListComponent {
  @Input() car: {model:string, price:number};
}
```

- כל פריט נשמר בתור אובייקט car שהוצאה שלו.
- צרכי להגדיר לאובייקט את הדקטור (@Input) כדי לאפשר את 'Յוא' המידע מההורה לילד.
- את Input מייאים מפניהם מ@m.angular/core.

שימוש לבו! הקומפוננטות יושבות בקבצים נפרדים, בלי קשר ביניהם, והדרך לקשר בין ההורה והילד היא באמצעות קובץ החטמ'ל של הקומפוננטה ההורה.

```
<h1>Models list</h1>

<h2>Add a model</h2>
<app-form></app-form>

<h2>The list</h2>

<app-list *ngFor="let item of cars" [car]="item"></app-list>
```

החלק שבו נעשה הקישור בין הקומפוננטות הוא בדירקטיב app-list, שעובד בלולאה על המערך cars (מההוראה), וכל פריט (item) נקשר לתוכנה (car), שנמצאת בקדם ה car, הקומפוננטה של הרשימה.

את האובייקט car מציגים למשתמשים תוך תבנית שמצוינה כל פריט ברשימתו:

```
<p>{{car.model}} {{car.price}}</p>
```

### 3. הקומפוננטה הילד, form, שאלה מדינים מודלים חדשים

המשתמשים מדינים את המידע לטופס הבא, שכולל את השדות model ו-price בשבייל שם המודול והמחיר.

```
<div>
  <label>Model</label>
  <input type="text" #carModel>
  <br />

  <label>Price</label>
  <input type="text" #carPrice>
  <br />

  <input type="button" value="Add" (click)="onAddCar(carModel, carPrice)">
</div>
```

בתגובה להקלקה על הכפתור המידע מועבר לטיפול המחלקה באמצעות המתודה AddCar() שמעבירה את המידע באמצעות local references #carModel, #carPrice

**local references** מכילים את כל המידע על אלמנט ההטמ"ל שבתוכו הם מצויים, וזה כולל את הערך שאותו מדינים המשתמשים תוך השדה. ננצל עובדה זו, ונעביר אותן כמשתנים למתחודה שתפקידן מהם את ערך השדה.

```

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

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html'
})

export class FormComponent {

  @Output() addCar = new EventEmitter<{model:string, price:number}>();

  onAddCar(model, price){
    this.addCar.emit({
      model: model.value,
      price: price.value
    });
  }
}

```

- בתגובה להקלקה על הcptor, מופעל האירוע `onAddCar` שירוח את האירוע `addCar` באמצעות `emit`.
- בשליל זה אנחנו משתמשים ב-`EventEmitter` שמאחול את האירוע `addCar`, ומעבר לו את הסוג שהוא צריך לקבל (הצורה הכללית של האובייקט).
- הדקוטו (`@Output()` משמש כדי ליצא את המידע לקומפוננטה של ההורה.

**שים לב להבדל!!** העברת המידע מההורה ליד נועשית באמצעות קשרה לתוכנות בעוד העברת המידע מהילד להורה נעשית בתגובה לארחים (events).

כדי שההורה ידע להגיב לאירוע `addCar`, נעביר את המידע דרך התבנית הראשית:

```

<h1>Models list</h1>

<h2>Add a model</h2>
<app-form (addCar)="onCarAdded($event)"></app-form>

<h2>The list</h2>

<app-list *ngFor="let item of cars" [car]="item"></app-list>

```

האירוע `addCar` מהטופס מפעיל את המתודה `onCarAdded` בהורה, והמידע מועבר באמצעות `$event`

```

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

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html'
})

export class AppComponent {
  cars = [{model:"subaru",price:125000},
           {model:"sussita",price:12000}];

  onCarAdded(data:{model:string, price:number}){
    this.cars.push({
      model: data.model,
      price: data.price
    })
  }
}

```

הڪאה מובהקת – לאפשר משתנה ללא הקזאה  
מוגדרת

definite assignment assertion

```

7
8  export class FriendsComponent{
9    @Input() currentFriend!:{
10      name: string; sameLove: boolean; numCircle: number;
11    };
12  }
13

```

## Undefined & Null

The value ***undefined*** means **value is not assigned** & **you don't know its value**. It is an **unintentional absence of value**. It means that a variable has been declared but has not yet been assigned a value.

The value ***null*** indicates that **you know** that the field **does not have a value**. It is an **intentional absence of value**.

## דִּירְקָטִיבָת Attribute של Angular2

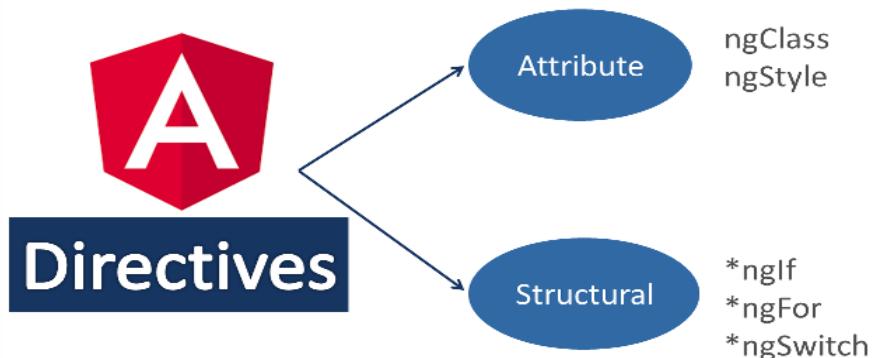
מחבר: יוסי בן הרוש בתאריך: 10.01.2018

דִּירְקָטִיבָת הַמִּיחִידָה הַבָּסִיסִית שֶׁל Angular. למעשה, גם הקומפוננטות, שביהם אנו מרבים להשתמש, הם דִּירְקָטִיבָת שִׁישׁ לְהַשְׁתָּמֵשׁ. במדרך אחר הראשו בסדרת האנגלור תוכל למדוד כיצד להשתמש בקומפוננטות אנגלוריות.

קיימים עוד 2 סוגי דִּירְקָטִיבָת:

- **דִּירְקָטִיבָת Attribute** - משנה את המראה וההתנהגות של אלמנטים בדף.
- **דִּירְקָטִיבָת מבניות** - יוצרות ו忽ירות אלמנטים של ה-HTML.

במדרך זה נסביר את הנושא של דִּירְקָטִיבָת Attribute.



### דִּירְקָטִיבָת Attribute

דִּירְקָטִיבָת Attribute משנה את המראה וההתנהגות של אלמנטים ב-HTML שעיליהם הם יושבות.

לדוגמה, ניתן לשנות את הקלאס של אלמנט באמצעות הוספה לוגיקה לדִּירְקָטִיבָת `:ngClass`:

```
<div [ngClass]="{'my-class': m==1, 'my-other-class': m==2}"></div>
```

- כשהמשתנה `m` מקבל את הערך 1 אז תן לאלמנט את הקלאס `.my-class`.
- כשהמשתנה `m` מקבל את הערך 2 אז תן לאלמנט את הקלאס `.my-other-class`.

בדִּירְקָטִיבָת `hidden` נשתמש להסתרת אלמנטים:

```
<p [hidden]="isHidden">Show if isHidden is false</p>
```

שימוש לבְּהַדִּירְקָטִיבָת `hidden` מאפשר להסתיר אלמנט באמצעות CSS, אבל האלמנט עדיין יטען ל-`DOM`.

את הדירקטיבה ש滥נו ששם my-highlight והוא נותנת צבע וקע לאלמנטים שעליים היא יושבת ניצור באמצעות ה-CLI, כלי שורט הנקודות של Angular:

> ng generate directive my-highlight

ng

נוצר את קובץ הדירקטיבה שאוטו יצר בשביבו ה-CLI:

app/my-highlight.directive.ts/

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

// Directive decorator
@Directive({
    selector: '[appMyHighlight]'
})

// Directive class
export class MyHighlightDirective {
    highlightColor: string = 'yellow';

    constructor(private el: ElementRef){
        this.el.nativeElement.style.backgroundColor = this.highlightColor;
    }
}
```

```
app > TS my-first-di.directive.ts > ...
import { Directive } from '@angular/core';

@Directive({
    selector: '[appMyFirstDi]'
})
export class MyFirstDiDirective {

    constructor() { }

}
```

- הדקוטור שיזה את הדירקטיבת Directive מזהה את הקלאס כדי רקטייה.
- הסלקטור שיזה את הדירקטיבת Directive שלנו ב-HTML הוא `appMyHighlight`, ואנחנו מקיפים אותו בסוגרים מרובעים כדי שנוכל לשימושו על האלמנט `<attribute>`, כפי שנראה בהמשך.
- באמצעות  `ElementRef` אנחנו יכולים לגשת לאלמנטים ב-DOM, ולשנות אותם.

כדי שהDIRECTIVE תעבד חיבים לייב אותה לאפליקציה, זה מה שהוא CLI עשה בשביבנו כשיצרנו את הדירקטיבת:

`app/app.module.ts.`

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
// Load our custom directive
import { MyHighlightDirective } from './my-highlight.directive';

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

אחרי שיידענו את האפליקציה על הדירקטיבת, יוכל להשתמש בה בכל קומponentה. כל מה שצריך זה לשימוש הסלקטור על האלמנט.  
לדוגמא:

`app/my-component/my-component.component.html.`

```
<h1>Welcome to my component</h1>
<!-- This element will be highlighted -->
<p appMyHighlight>My highlighted content</p>
<p>This content won't be highlighted</p>
```

באו נעשה קצת יותר מעניין עם הדירקטיבת שלנו, ונאפשר להגדיר את צבע הרקע מבוחוץ. לשם כך, נוצר ב-`input` שמספק לנו אנגלול, ומאפשר לנו לקבל ערכים מחוץ לדירקטיבת.

```

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

// Directive decorator
@Directive({
  selector: '[appMyHighlight]'
})

// Directive class
export class MyHighlightDirective implements OnInit {
  @Input() highlightColor: string = 'yellow';

  constructor(private el: ElementRef){
  }

  ngOnInit(){
    this.el.nativeElement.style.backgroundColor = this.highlightColor;
  }
}

```

- אנחנו משתמשים ב-`@Input` כדי לאפשר את הגדרת ערך המשנה `highlightColor` מוחוץ לדירקטיבה.
  - נשנה את מראה האלמנט בתוך `ngOnInit` כי הוא פועל לאחר שהאלמנט כבר קיים. אם נשים בקונסטרוקטור זה לא יעבד.
- נוסף עשינו את הערך של `backgroundColor` כ-`attribute` לאלמנט:

app/my-component/my-component.component.html/

```

<h1>Welcome to my component</h1>

<p appMyHighlight>Highlight me with the default color!</p>
<p appMyHighlight [highlightColor]=""green"">Highlight me with a different color that is set
in the template!</p>
<p appMyHighlight [highlightColor]="color">Highlight me with a different color that is set
in the class!</p>
<p>No highlight!</p>

```

- מכיוון שהגדכנו בתוך הדירקטיבה ערך בריית מחדל אנחנו לא חייבים להגדיר אותו על האלמנט.
- ניתן להעביר את הערך בתוך מחרוזת בתוך הטמפליט, ואז חשוב להזכיר את המחרוזת בגרשיים. כי זה לא ערך של משתנה אלא מחרוזת.
- את המשתנה `color` אנחנו יכולים להגדיר בתוך קוד הקומפוננטה:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-my-component',
  templateUrl: './my-component.component.html'
})
export class MyComponentComponent {
  color:string = 'green';
}
```

אנחנו יכולים לעשות משהו עוד יותר מעניין ולשנות את מראה הדירקטיבה בהתאם לאיורים. לדוגמה, להחליף את צבע הרקע כשעוברים עם העכבר. לשם כך, ניבא את ה-HostListener לדירקטיבה.

```
> app > My-Dog > My-Dog.component.html > p
1   @MyFirstDi [fontSize]=18 myHighLightColor="antiquewhite"
```

לא לרשום  
cash�וחים צבע

```
ngOnInit(){
  this.myElementRef.nativeElement.style.backgroundColor =
  this.myElementRef.nativeElement.style.fontSize= this.fontSize;
  this.myElementRef.nativeElement.style.color = this.myColor;
}
```

לשימם לבאות הראשונה  
של תכונות style היא  
קטנה

```
export class MyFirstDiDirective {

  @Input() myHighLightColor:string='pink';
  yellowHighLightColor:string = "yellow";
  currentHighLightColor:string = "";
  @Input() myColor:string='black';
  @Input() fontSize: number = 10;

  constructor(private myElementRef: ElementRef) {}

  ngOnInit(){
    this.myElementRef.nativeElement.style.backgroundColor = this.myHighLightColor;
    this.myElementRef.nativeElement.style.fontSize= this.fontSize+'px';
    this.myElementRef.nativeElement.style.color = this.myColor;
  }

  @HostListener('mouseenter') onMouseEnter() {
    this.currentHighLightColor = this.myHighLightColor;
    this.myHighLightColor = this.yellowHighLightColor
    this.ngOnInit();
  }

  @HostListener('mouseleave') onMouseLeave() {
    this.myHighLightColor=this.currentHighLightColor;
    this.ngOnInit();
  }
}
```

|| מחזור חיים  
שנקרא לאחר Angular  
אתחול את כל המאפיינים  
הקיימים  
לנתונים של הוראה.

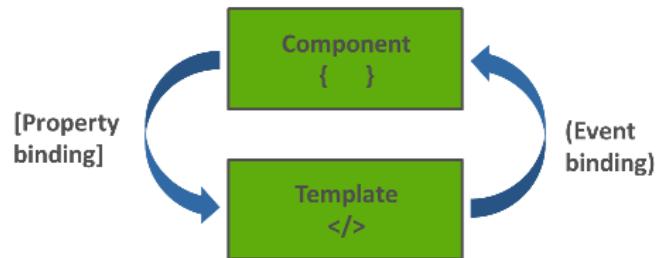
## קישור לתכונות ולאירועים ב-Angular

מחבר: יוסי בן הרוש בתאריך: 22.09.2017

קומפוננטה של Angular בונה מבנית ה-HTML וקוד typescript שישוב בתוך מחלקה, והבעיה היא כיצד להעביר את המידע בין שני החלקים. במדריך זה נזכיר שתי דרכים לטיפול בעיה:

- **קישור לתכונות** - מעבירה את המידע מקוד typescript אל תבנית ה-.html.
- **קישור לאירועים** - מאפשר למידע לעבור מה-.html אל קוד המחלקה.

התרשימים הבא ממחיש את כיווני זרימת המידע:



### קישור לתכונות

קישור לתכונות מעבירה את המידע מקוד typescript אל תבנית ה-.html. לדוגמה, אם נרצה להעביר את מצב הקפתור והכיתוב עליו מהמחלקה לתבנית ה-HTML, נשתמש בקישור לתכונות:

[src/app/app.component.ts](#)

```

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

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html'
})

export class AppComponent {
  text = "שלום";
  active = true;
}
  
```

[src/app/app.component.html](#)

```

<p>
  <button [disabled]="!active">{{text}}</button>
</p>
  
```

שימוש לב شأنנו קשורות בין תוכנה של המחלקה לבין תוכנה של אלמנט ה-HTML באמצעות סוגרים מרובעים. וקישורה מאפשר לנו לשלוח מידע מהמחלקה אל אלמנט ה-HTML.

62

```
[disabled]="!active"
```

\* הוספנו סימן קראיה (!) כדי להפוך את הערך של המשתנהaboliani. כי כשהתכונה active = true אנחנו מעוניינים שהכפטור יעבד.

להעברת ערכים של טקסט, נעדיף שימוש בסוגרים מסווגלים. {{text}}

אבל כדי להעביר ערכים שאינם מחרוזת (דוגמתaboliani) אנחנו חייבים להשתמש בתחביר עם הסוגרים המרובעים.

## 2. קישורה לאירועים

קישורה לאירועים מאפשרת לנו להפעיל מתודה בתוך המחלקה בתגובה לאירוע שתרחש בתבנית. לדוגמה, הקלה על כפטור (click) , לחיצה על מקש במקלדת (keydown,keyup,blur,focus) . וכי"ב אירוע של javascript .

בדוגמה להלן, לחיצה על הכפטור תגרום להפעלת המתודה () clicked שתגרום לשינוי הכיתוב על הכפטור וגם להעברתו במצב לא פעיל.

[src/app/app.component.ts/](#)

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

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html'
})

export class AppComponent {
  text = "שלום";
  active = true;

  clicked() {
    this.text = "נשלח";
    this.active = false;
  }
}
```

63

```
<p>
  <button [disabled]="!active" (click)="clicked()">{{text}}</button>
</p>
```

הקישור לאירוע נעשה על ידי הקפת שם האירוע בסוגרים עגולים וקרייה לפונקציה:

```
(click)="clicked()"
```

ניתן להבהיר מידע אודות האירוע באמצעות האובייקט `$event`

```
(click)="clicked($event)"
```

בדוגמת הקוד להלן המידע שמועבר עם האובייקט event מודפס לკונסולה:

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

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html'
})

export class AppComponent {
  text = "שלום";
  active = true;

  clicked($event) {
    console.log($event);
  }
}
```

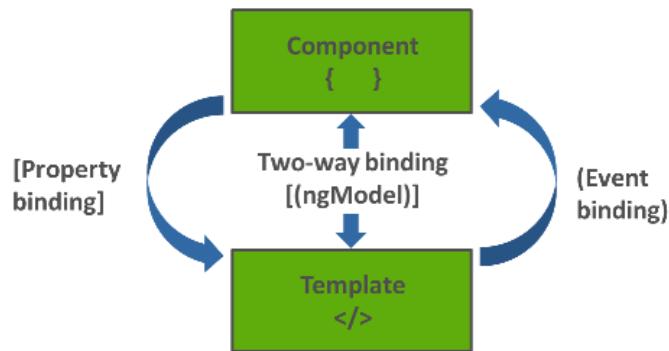
```
<p>
  <button [disabled]="!active" (click)="clicked($event)">{{text}}</button>
</p>
```

## קישור דו-כיוונית ב-Angular

מחבר: יוסי בן הרוש בתאריך: 23.09.2017

קישור דו-כיוונית מאפשרת לנו לעדכן את המידע המוצג למשתמש בקבוץ התבנית וגם את ערכה של התconaה בקומפוננטה בו-זמנית.

אחרי שבמזריך קודם למדנו  [קישור חד-כיוונית ב-Angular](#), שמאפשרת להעביר מידע מהקוד במחלקה לתבנית ההתמ"ל או הפוך, אבל לא ביחד, במדריך זה נלמד להעביר מידע בשני הכוונים באמצעות קישור דו-כיוונית, **two-way data binding**.



### קישור דו-כיוונית באמצעות FormsModule

כדי לישם קישור דו-כיווני אנחנו צריכים לייבא את FormsModule בקובץ app.module.ts

`src/app/app.module.ts`

```

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

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

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

זה לא מספיק ליבא את המודול, צריך גם להוסיף אותו למערך imports.

את המשתנה שמח נגזר ממשתנה דו-כיווני.

בתבנית ההטמ"ל, נגיד את המשתנה `name`

```
[(ngModel)]="name"
```

ובכך נגזר את המשתנה הקשור בקשר דו-כיוונית.

[src/app/app.component.html](#)

```
<label>הזן את שמי</label><br />
<input [(ngModel)] = "name" /><br />
<p>שלום {name}</p>
```

כשהמשתמש מזין את השם לתוכה, המשתנה בו בזמן הערך של `name` שעובר אינטראקטיבית בין הסוגרים المسؤولים. ולא זו בלבד, הערך של המשתנה בתוך המחלקה משתנה גם הוא. כדי לראות את השינוי במחלקה, נגיד את הערך של `name` בקוד המחלקה.

[src/app/app.component.ts](#)

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})

export class AppComponent {
  name = "ג'וני";
}
```

ערך של המשתנה בתוך המחלקה יהיה את הערך בירית המחלקה.

## אפליקציה אングולרית local reference

מחבר: יוסי בן הרוש בתאריך: 28.11.2017

משתנים מקומיים מספקים אפשרות לשדרה לאלמנט בתוך html. כדי להכריז על local reference מקדים למשזה סולמית (#):

```
<input #myLocalReference value="">
```

בדוגמה זו, myLocalReference הוא ה-reference שאנו שמים על השדה בוטופו.

המשנה המקומי מתייחס לשדה שבתוכו הוא יושב, ומשתמש בו כדי להעביר מידע:

1. לעדכן תבנית html שבתוכה הוא יושב
2. לעדכן הקוד בקומפוננטה

בדוגמה הבאה, בכל פעם שנקלידתו לערוך השדה, הערך שקיים בשדה יוצג בתוך הפסקה שמתחתיו.

[src/app/app.component.ts/](#)

```
<input #myLocalReference value="" (keyup)="changeVal()">
<p>{{ myLocalReference.value }}</p>
```

[src/app/app.component.html/](#)

```
changeVal(){}
```

שים לב, האירוע של keyup יגרום לעדכן הערך של ה-reference local reference ב-html באופן אוטומטי בלי שהמתודה מבצעת פעולה.

ניתן להשיג אותה התוצאה באמצעות קרייה למетодה שתעדכן את הערך המוצג ב-html:

```
myChange($event:any){
  console.log($event.target.value)
}
```

```
<input #myLocalReference value="" (keyup)="changeVal(myLocalReference.value)">
<p>{{ value }}</p>
```

```
value = "";
changeVal(val: string){
  value = val
}
```

בכל פעם שמקלידים או חדש לטור השדה, הערך נשלח למетодה changeVal שמעדכנת את הערך של המשתנה value שמצוג בפוסקה.

ניתן להאזין ליותר מאירוע אחד באמצעות אוטו :local reference

```
<input #myLocalReference value=""
       (keyup)="changeVal(myLocalReference.value)"
       (blur)="changeVal(myLocalReference.value)">
```

```
<input #myLocalReference value=""
       (keyup)="changeVal(myLocalReference.value)"
       (blur)="changeVal(myLocalReference.value); myLocalReference.value="">
```

- בדוגמה זו, האירוע `blur` עושה שני דברים. קורא לפונקציה `changeVal`, וגם משנה את הערך של השדה.
- בין שתי הפעולות מפheid הסימן נקודה פסיק.

## Observable באנגולר

מחבר: יוסי בן הרוש בתאריך: 08.06.2018

### [What is Angular Observable](#)

קיימת גם גרסה אנגלית של המדריך אותה תוכלו לקרוא אם תלחצו על הקישור [What is Angular Observable](#)

במדריכים הבאים אנו מטפל בנושאים של שימוש Ajax ובינו לבין דפים באפליקציה דף אחד. שני הנושאים מבוססים על קוד AngularJS, ולכן מאוד חשוב להבין את הדרך המיוחדת שבה Angular מטפל במקרה זה.

אנגולר משתמש ב- **Observable** לטיפול במקרה של promises ו-callbacks ב- JavaScript. למעשה Observable עדיף להתווסף לארסאות עתידיות של RxJS, אבל עד שזה יקרה הוא מיושם ב- Angular.

Angular משתמש באופן נרחב observable לטיפול במקרה שלobservable. לדוגמה, לטיפול עם קוד Ajax, שימושים לארזים וబשיל ניוט בין דפי האפליקציה (routing). בשביל להבין את זה, מחשוב על קוד Ajax שמהכה לתגובה משרת מרוחק. אין אפשרות לדעת متى התגובה תחזור, והאם היא בכלל תחזור, וכן זה לא רעיון טוב לעצור את ביצוע הסקריפט עד לקבלת התגובה. קוד observable הוא פתרון טוב בהרבה, והדרך שבה אנגולר מעדיף לטפל במקרה שלobservable היא באמצעות observable.

ה- **observable** נוגע לפיה הדפואו התכונתי של **observer** הכלל שני שחקנים ראשיים:

Observable .1

Observer .2

ה- **observable** משגר מידע בעוד ה- **observer** גרשם (**subscribe**) אליו כדי לקבל את המידע.

ה- **observable** יורה את המידע בתגובה למשהו. לדוגמה, בתגובה לאיור, כאשר משתמש מקליק על כפתור, או בתגובה למידע שמאפשר משרת מרוחק.

ה- **observer** מכיל שלוש ידיות לשימוש במידע:

1. **onNext** מטפל במידע המבוקש,

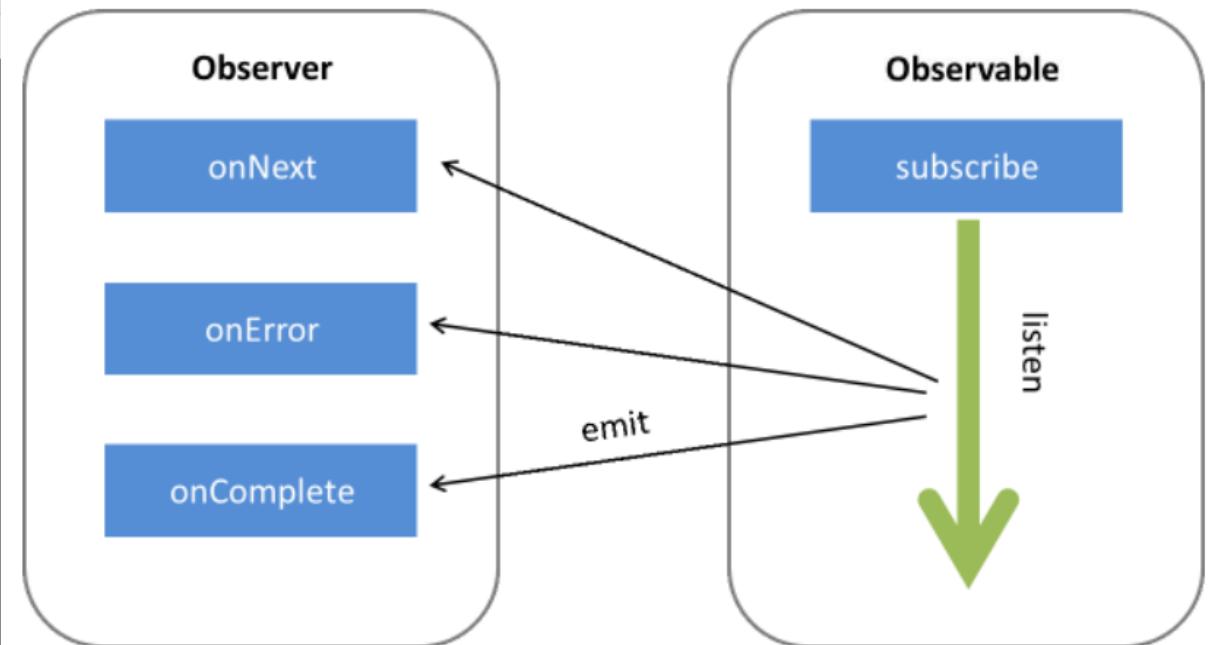
2. **onError** לטיפול בשגיאות,

3. **onComplete** כאשר התהליך מסתיים.

סינכרוני משמעוונו **פעולה** אחת פעמי אחת בכל זמן, הפעולה הבאה לא תתחיל לפני שהודמתה תסתיים.

איסינכרוני משמעוונו, מספר **פעולות** במקביל, כמו קריית API למשל, פקודה אחת לא "תוקעת" פקודה אחרת אלא הכל מתבצע במקביל.

אחרת אל הצל מתבצע במקביל. 29 באוקטובר 2020



כשהתכוות **observer** אתה יכול להחליט מה לעשות כאשרה מקבל את המידע שאתה מzystה לו, כיצד לנוהג בשגיאיה, ומה לעשות במקרה של סיום.

לא כל ה-observables מגיעים לסיום. לדוגמה, observable שמאזין לשינויים ב-URL לא יכול לסיים לעולם מכיוון שהוא משתמש תמיד עשוי לנוט לדף חדש.

## אפליקציה אングולרית מבוססת דף יחיד

מחבר: יוסי בר הרטש בתאריך: 21.10.2017

במדריך זה הוגם לחלק החשוב ביותר לימודי **Angular**. כי אングולר זו רק סביבת עבודה שמאפשרת לנו לכתוב JavaScript מתקדם, אלא בעיקר ספרית קוד שמיועד לכתיבת אפליקציות של דף יחיד (**spa = single page application**) שגראות ומונגהות כמו האפליקציות שאנו חנו רגילים אליוים בינוידים או על מחשבים שולחניים.



### ה-[url](#) באפליקציה

শচকটিবস্মী এপ্লিকেশন তুকন দফাসমূহ মন্তব্য করা হয়ে থাকে। URL ক্ষেত্রে শৈলী-ব-URL ক্ষেত্রে আইন্টেরেট রাগি।

לדוגמא:

/yourwebsite.com//

ציג את תוכן של דף הבית.

yourwebsite.com/users//

ציג מידע על המשתמשים.

החידוש באפליקציות של דף יחיד הוא שאנו חנו כל הזמן נשארים באותו דף, ב-index.html, ורק תוכן הדף משתנה בכל פעם שמחיליפים את הכתובת בשורת הכתובות. כתוצאה לכך, נמנעת התופעה של הבוהוב בזמן שבו עוברים בין הדפים, וגם זמן הטעינה של התכנים מתקצר משמעותית.



### הדף index.html

הוא הדף היחיד שהאפליקציה מציגה, והוא כולל את פרטי הקישורים הראשיים שיאפשר לנו לדפס בינהם (האלמנט nav).

אם איןכם יודעים כיצד להתקין את סביבת הפיתוח של Angular, נא להיכנס [למדריך הראשון בסדרת ה-\*\*Angular\*\*](#).

src/index.html/

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>HelloRouting</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

השדה הראשון ב-head צריך להיות **base href** והוא אומר לאפליקציה איפה כתובות משמשת להציג כל התכנים (היכן נמצא index.html). לדוגמה, אם דף האינדקס נמצא בתיקיית השורש של האפליקציה (src/index.html) נשתמש ב-

```
<base href="/">
```

## הקומפוננטות הבסיסיות

4 הקומפוננטות הבסיסיות שישרתו את האפליקציה מיעדות: לדף הבית, להציג המכוניות, להציג מכונית בודדת, ולדף 404.

יצור את הקומפוננטות באמצעות ה-`ng generate component`:

```
> ng generate component home
```

יצירת הקומponentה `homeComponent` משמשת להצגת דף הבית.

```
> ng generate component page-not-found
```

יצירת הקומponentה `pageNotFoundComponent` משמשת להצגת דף 404 ("דף לא נמצא").

```
> ng generate component cars
```

יצירת הקומponentה `carsComponent` שתשתמש אותו בשלב ראשון להציג את רשימת המכוניות.

אחרי יצירתו את הקומponentה `cars` ניצור בתוכה קומponentה בת, ששמה `car`. לשם כך, ננווט עם ה-`cd` לתוך התקייה `cars` וניצור בתוכה את הקומponentה `.car`.

קדם כל, ננווט באמצעות ה-`cd` לתוך הקומponentה `cars`:

```
> cd src/app/cars
```

ועכשיו, בתוך התקייה `cars`, ניצור את הקומponentה `car`:

```
> ng generate component car
```

## הראוטר

הנתב (ראוטר) משמש להפנות את כתובות ה-`url` לקומפוננטות שמתפלות בהם. ניצור את קובץ הרואוטר בתיקיית השורש של האתר, ונעניק לו את השם `app-routing.module.ts`, בהתאם להיותו קובץ מודול.

החלק העיקרי במודול הרואוטר הוא **מערך של אובייקטים** שמחברים בין כתובות האינטרנט לבין הקומפוננטות שמהוות את הדפים.

נוסיף את קובץ מודול הרואוטר בתיקיית השורש של האפליקציה (`:src/app`)

`app-routing.module.ts`

ונשתמש בקוד להלן בתור שיד ה-`router`:

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

import { HomeComponent } from './home/home.component';
import { CarsComponent } from './cars/cars.component';
import { CarComponent } from './cars/car/car.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

const appRoutes: Routes = [
  {path: "", component: HomeComponent },
  {path: 'cars', component: CarsComponent},
  {path: 'not-found', component: PageNotFoundComponent},
  {path: '**', redirectTo: '/not-found'}
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ],
  exports: [RouterModule]
})

export class AppRoutingModule { }
```

עדכון

- מכיוון שהארווטר יושב בתוך מודול עליון לייבא את NgModule מליבת אングולר:

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

- ניבא את האובייקטים RouterModule,RouterModule מפניהם שהקוד שלהם הוא ש账号 הוא אחראי ליכולות הניווט של האפליקציה.
- ניבא את כל הקומפוננטות שישמשו להצגת התכנים באתר. נתחיל מיבוא הקומפוננטות שכבר יצרנו:

```
, HomeComponent  
, CarsComponent  
, CarComponent  
PageNotFoundComponent
```

- החלק שמנפה את הכתובות על הקומפוננטות הוא מערך ששייך לסוג Routes (שאותו יbam בצעד השני):

```
const appRoutes: Routes = [ ];
```

- בתוך המערך ישם אובייקטים שבהם נעשה המיפוי, נראה את הדוגמה של דף הבית:

```
const appRoutes: Routes = [  
  {path: "", component: HomeComponent }  
];
```

- הנتيיב של דף הבית, שהוא מחוזת ריקה, גורם לטעינה של הקומפוננטה HomeComponent

```
const appRoutes: Routes = [  
  {path: "", component: HomeComponent},  
  {path: 'cars', component: CarsComponent}  
];
```

- ונוiot לכתובת cars יטعن את הקומפוננטה CarsComponent
- בהמשך נסיף נתיבים נוספים שיובילו כל אחד לקומפוננטה נפרדת.
- מה קורה כשmagיעים ל-path שאינו מוגדר בראוטר? נפנה את כל הכתובות שלא מוגדרות בראוטר לכתובת not-found שם תטפל בהם הקומפוננטה PageNotFoundComponent

```
const appRoutes: Routes = [  
  {path: "", component: HomeComponent },  
  {path: 'cars', component: CarsComponent},  
  {path: 'not-found', component: PageNotFoundComponent},  
  {path: '**', redirectTo: '/not-found'}  
];
```

האובייקט האחרון צריך תמיד להכיל את הנתיב **\*\*\*** שימושו כל נתיב אחר, והוא מפנה באמצעות RedirectTo לנטייב שבו נמצאת הקומפוננטה שמטפלת במקרים שבהם הדף לא נמצא.

```
@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ],
  exports: [RouterModule]
})

export class AppRoutingModule { }
```

- ש망דרים בשבייל ה-RouterModule שעליו להשתמש ב-`appRoutes` בהתאם להראות, וגם חשף את המודול לשאר האפליקציה באמצעות ייצואו (export).

כדי לידע את האפליקציה על מודול הנוכחי שהוא עתה פיתחנו, ניבא אותו לקובץ:

src/app/app.module.ts

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

import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';

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

import { HomeComponent } from './home/home.component';
import { CarsComponent } from './cars/cars.component';
import { CarComponent } from './cars/car/car.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    CarsComponent,
    CarComponent,
    PageNotFoundComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

באותה הzdמנות, לא נשכח ליבא את ארבעת הקומפוננטות שיצרנו:

,HomeComponent  
,CarsComponent  
,CarComponent  
PageNotFoundComponent

לבסוף, צריך מקום שבו יוצגו כל התכנים שבאים מהקומפוננטות, ומקום זה צריך להיות בקובץ html המרכז של האפליקציה, בתוך directive אングולרי `:router-outlet`:

`src/app/app.component.html/`

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

### נתיבים הורים ונtíבִים ילדים

קומפוננטות שיישבות בהן קומפוננטות אחרות הם קומפוננטות מקוונות (nested), ובהתאם גם הנתיב שモבייל אליום צריך להימצא בתוך נתיב קיים, ולא להיכתב בנפרד.

זה התחביר:

```
{path: 'parentPath', component: ParentComponent, children:[
  path: ':id', component: ChildComponent,
  path: ':id/edit', component: ChildComponentEdit,
  path: 'something' component: ChildComponentSomething
]}
```

- הנתיבים לילדיים מוקובצים ביחד בערך `children` שישוב בנתיב של ההורה.
- כשמגדירים את `path` לילדיים משמשים מה-path את חלקה של הכתובות שתורות ההורה.

אחרי שהסבירנו את העיקנון, נדגים את היחסים בין הורם לבין לילדם בראוטר באפליקציה שלנו.

באפליקציה שאנחנו מפתחים הקומפוננטה `car` נמצאת בתוך הקומפוננטה `cars`, ולפיכך צריך לקון אותה בראוטר:

`src/app-routing.module.ts`

```
const appRoutes: Routes = [
  {path: "", component: HomeComponent },
  {path: 'cars', component: CarsComponent, children:[
    {path: ':id', component: CarComponent }
  ]},
  {path: 'not-found', component: PageNotFoundComponent},
  {path: '**', redirectTo: '/not-found'}
];
```

- להגדרת `path` שטוען את הקומפוננטה הילד השתמשנו ב-`:id` ולא ב-`:id/cars` כי הכל הוא שוצריך להשミニ את החלק בכתובת שמאגי'ע מההורה

### יבור באמצעות קישורים

אחרי שהסבירנו קומפוננטות והגדירנו את הנתיבים שモביילים לקומפוננטות בתוך הראוטר אנחנו צריכים למדוד כיצד לנווט אליהם. ישנן שתי דרכים שימושות ליבור:

1. באמצעות קישורים
2. דרך תכניות

בחולק זה נלמד כיצד להשתמש בקישורים.

על גבי כל אחד מהקישורים נגידר את התוכנה `routerLink`, שהערך שלו הוא מערך שמויביל לנתייב:

לדוגמה, קישור לדף הבית:

```
<a [routerLink]="/">Home</a>
```

קישור לכתובת cars תהיה:

```
<a [routerLink]="/cars">Cars</a>
```

קישור להוספה מכונית:

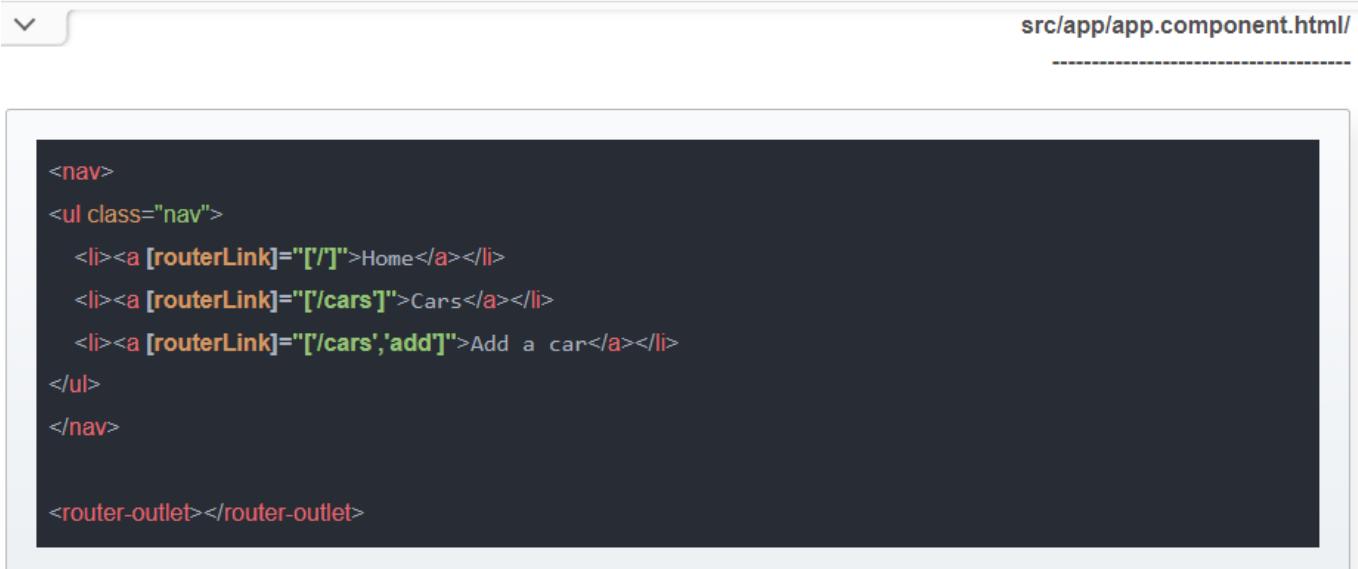
```
<a [routerLink]="/cars','add'">Add a car</a>
```

קישור לעדכנת מכונית שה-ip שלה הוא 3:

```
<a [routerLink]="/cars','3','edit'">Edit car #3</a>
```

השימוש במערך מאפשר כתיבת כתובות דינמיות בקלות רבה, כפי שנראה בהמשך.

נוסיף את תפריט הניווט הראשי לקובץ הנטמ"ל המרכדי של האפליקציה מעל ל router-outlet:



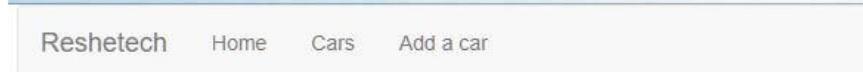
```

<nav>
  <ul class="nav">
    <li><a [routerLink]="/">Home</a></li>
    <li><a [routerLink]="/cars">Cars</a></li>
    <li><a [routerLink]="/cars','add'">Add a car</a></li>
  </ul>
</nav>

<router-outlet></router-outlet>

```

כך נראים הקישורים (העיצוב מבוסס על bootstrap).

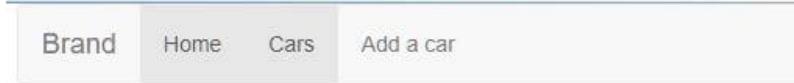


home works!

כדי לסמך את הקישורים הפעילים באמצעות הקלאס active, השערך שהוא מקבלת הוא שם הקלאס שמסמן את הקישורים הפעילים, שבאפליקציה שלנו הוא 'active'. כך זה נראה אצלו:

```
<nav>
<ul class="nav">
  <li><a [routerLink]="/" routerLinkActive="active">Home</a></li>
  <li><a [routerLink]="/cars'" routerLinkActive="active">Cars</a></li>
  <li><a [routerLink]="/cars','add'" routerLinkActive="active">Add a car</a></li>
</ul>
</nav>
```

שים לו בבעיה שניות לכתובת - cars יגרום להדגשה השגיה של הנטייה לדף הבית. כך זה נראה אצלך:



cars works!

וגם שניות לנטייב cars/add יגרום להדגשת הנטייבים לדף הבית ול-cars



cars works!

הסיבה לבעה היא שאנגולר מתחילה לבחש התאמת הצדזה השמאלי של הכתובת, ומספיק שתמצא התאמת בחילקה השמאלי ביותר של הכתובת, שהוא תמיד (/), כדי שהנתיב יוגדר כפועל.

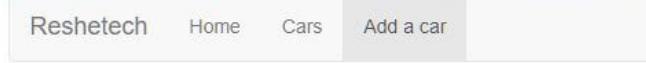
הפתרון הוא שימוש בתוכונה נוספת נסופת של ה-router:

```
[routerLinkActiveOptions]="{exact: true}"
```

נוסיף את התוכונה שתפקידו את בעית חוויס הספציפיות לשני הקישורים שביהם קיימת הטעיה:

```
<nav>
<ul class="nav">
<li><a [routerLink]="/" routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}">Home</a></li>
<li><a [routerLink]="/cars" routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}">Cars</a></li>
<li><a [routerLink]="/cars','add'" routerLinkActive="active" >Add a car</a></li>
</ul>
</nav>
```

בעקבות השימוש בתוכונה, כשרנווט לקישור add/cars/. יודגש באופן בלעדי הקישור לנטייב שבו נמצאים בתוכו.



cars works!

## ה-service שמתפלט במידע על המכוניות

בפרק הבא נלמד כיצד לנוטו באופן תכניabil קודם ליצור service אングורי שיירץ את הטיפול במידע עבור המכוניות. ובהתאם הוא יוכל את מערכת המכוניות וגם את המתודות שתפקידן לנוהל את המידע, ובכללן קר הצגה, הוספה, עדכון ומחיקה.

[src/app/car.service.ts](#)

```
export class CarService{
  private cars = [
    {id: 'xZw1Dg', model: 'Mercedes', price: 40000},
    {id: '2AbCdE', model: 'Tesla', price: 45000},
    {id: '2hXC9E', model: 'Jaguar', price: 65000},
    {id: 'ABvD75', model: 'Sussita', price: 4500},
  ];

  getAll(){
    return this.cars;
  }

  getById(id: string){
    for(let index in this.cars){
      if(this.cars[index].id == id)
        return this.cars[index];
    }
  }

  create(newModel: string, newPrice: number){
    this.cars.push({id: this.generateId(), model: newModel, price: newPrice});
  }
}
```

```

updateById(car: {id: string, model: string, price: number}){
    for(let index in this.cars){
        if(this.cars[index].id == car.id)
            this.cars[index] = car;
    }
}

deleteById(id: string){
    for(let index in this.cars){
        if(this.cars[index].id == id)
            this.cars.splice(parseInt(index), 1);
    }
}

private generateId(){
    let str = "";
    let possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_";

    for (var i = 0; i < 6; i++){
        str += possible.charAt(Math.floor(Math.random() * possible.length));
    }

    return str;
}
}

```

- הידוע על המכוניות מוחזק בתוך המערך cars. וכל פריט במערך יש את השדות id, model, price ו-i.
- הmethod getAll מחזירה את מערך המכוניות.
- הmethod getById מחזירה את הפריט במערך שיש לו את ה-id המבוקש.
- הmethod create מקבלת את מחיר ומודל המכונית החדשה, ומוסיף לה "חוד' באמצעות method generateId לפני שהיא מוסיף אותה למערך המכוניות.
- הmethod updateById מקבל את האובייקט car, ומעדכן את הפריט במערך לפי ה-id.
- הmethod deleteById מוחckaת את הפריט שיש לו את המזהה הספציפי.

כדי לידע את האפליקציה אודות ה-service נרשם אותה ב-`app.module.ts`

[src/app/app.module.ts/](#)

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

import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';

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

import { CarService } from './car.service';

import { HomeComponent } from './home/home.component';
import { CarsComponent } from './cars/cars.component';
import { CarComponent } from './cars/car/car.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    PageNotFoundComponent,
    CarsComponent,
    CarComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [CarService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

שימוש לב שלא רק יבואנו את ה-`CarService`, גם הוסיף אותו למערך ה-`providers`

## שימוש באמצעות קוד

ניתן לנוט באמצעות הקוד, ולשם כך נשתמש באובייקט האנגולרי Router שאותו ניבא לתוך הקומפוננטה cars.

`src/app/cars/cars.component.ts/`

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

import { Router } from '@angular/router';

import { CarService } from '../car.service';

@Component({
  selector: 'app-cars',
  templateUrl: './cars.component.html',
  styleUrls: ['./cars.component.css']
})
export class CarsComponent implements OnInit {
  cars: Array<{id:string, model:string, price:number}>;

  constructor(private router:Router, private carService: CarService) { }

  ngOnInit() {
    this.getCars();
  }

  getCars(){
    this.cars = this.carService.getAll();
  }

  loadCar(id: string){
    this.router.navigate(['/cars',id]);
  }
}
```

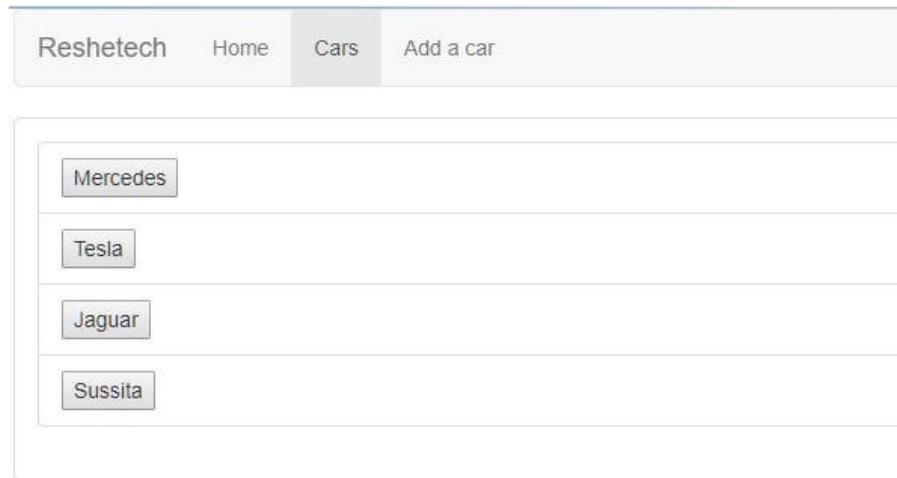


- ניבא את ה-Router, ונמש אוטו (בצורה ממנו אובייקט מקומי) בקונסטרוקטור.
- המתודה Car מקבלת id בטור פרמטר, ומשתמש בתוכנה router.navigate כדי להפנות לכתוב המבוקש.
- הפרמטר shNavigate מקבלת הוא מארך של חלקו הכתובת שאליה הוא מפנה.
- ה- id מועבר מהפרמטר שמקבלת המתודה ומושבץ במערך שמננו אנגולר יצר את הכתובת שאליה נרצה להגעה.
- המתודה getCards מיבאת את כל מערכ המכוניות מה-service, ומציבה אותו במשתנה המקומית cars שיכל מערך של מכוניות. כל מכונית היא מסוג אובייקט הכלול: id, model ו גם price.

בקובץ התבנית של הקומפוננטה cars, נדייס כפתורים שלחיצה עליהם תפנה לעירית המכוניות. ההדפסה תעשה בתחום לולאה שרצה על רשימת כל המכוניות שיבאו לתוך הקומפוננטה מה-service.

`src/app/cars/cars.component.html/`

```
<ul>
<li *ngFor="let car of cars">
  <button (click)="loadCar(car.id)">{{car.model}}</button>
</li>
</ul>
```



אנו מוצפים של חיציה על כל אחד מהכפרורים תגרום להציג המכונית המבוקשת, וכי לחשיג את מבוקשנו, נctrוך, קודם כל, לתפוא את המידע  
שמועבר באמצעות ה-[API](#).

## כיצד לתרפס את המידע ש מגיע מה-Url?

האובייקט האנגולרי ActivatedRoute משמש אותנו כדי לקלוט מידע מהכתובת. המידע נקלט על ידי האזנה לכתובת וקליטת כל שינוי שימושו בה.

[src/app/cars/car/car.component.ts](#)

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

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

import { CarService } from '../car.service';

@Component({
  selector: 'app-car',
  templateUrl: 'car.component.html'
})
export class CarComponent implements OnInit {

  car: {id:string, model:string, price:number};
  id : "";

  constructor(private carService:CarService, private activatedRoute:ActivatedRoute) { }

  ngOnInit() {

    this.activatedRoute
      .params
      .subscribe(params => {

        this.id =params['id'] || "";

        if(this.id != ""){
          this.car = this.carService.getById(this.id);
        }
      });
  }
}
```

- ייבנו את האובייקט `.ActivatedRoute`.
- הפקם אותו למשתנה מקומי ב-`constructor`.
- בתוך `ngOnInit` נרשמו (**subscribe**) לכל אירוע של הגדרה או שינוי של ערך ה-`car`, וכשהקומפוננטה נטענת או אחר כך ניקח מהכטובה את ערך `id` המכוניות. אחרי שנקבל את ה-`id`: נשתמש ב-`-p` כדי לשולף את המידע אודות המכוניות, וכאן אנו שמים את הפונקציה בהמתנה בתוך קוד.
- אנחנו נרשמים, **subscribe**, לאירוע הניגוט לכתובות מפני שאיןנו יודעים מתי הוא יתרחש, ולכן אנו שמים את הפונקציה בהמתנה בתוך קוד אסינכרוני עד שיתרחש האירוע, ורק אז הקוד יזע.
- ההרשמה (**subscribe**) נעשית בתוך `ngOnInit` כיוון שמיד כשהקומפוננטה נטענת היא צפיה לקבל פרמטרים שימושיים בכתובות ה-URL.

את המידעណ דפיו להובץ התבנית של הקומפוננטה `car`:

`src/app/cars/car/car.component.html`

```
<h1>The current car</h1>
Car model: {{car.model}}
<br />
Car price: {{car.price}}
```

אבל אם ננסה לנוט לכתובות נקבל שגיאה כי אין מקום שבו הקומפוננטה הילך `car` מציגה את המידע.

## ה-`router-outlet` שטוח directive

הDIRקטיבה `router-outlet` משמשת להציג ה-`html` של הקומפוננטות הבנות בתוך קובץ התבנית של הקומפוננטה ההורה. כדי להציג את ערכי המשתנה `car`, וכל מידע אחר שמקורו הקומפוננטות הבנות בתוך קובץ התבנית של האלמנט ההורה (`cars`), נוסיף את ה-`directive` `router-outlet` ל-`html` של הקומפוננטה ההורה:

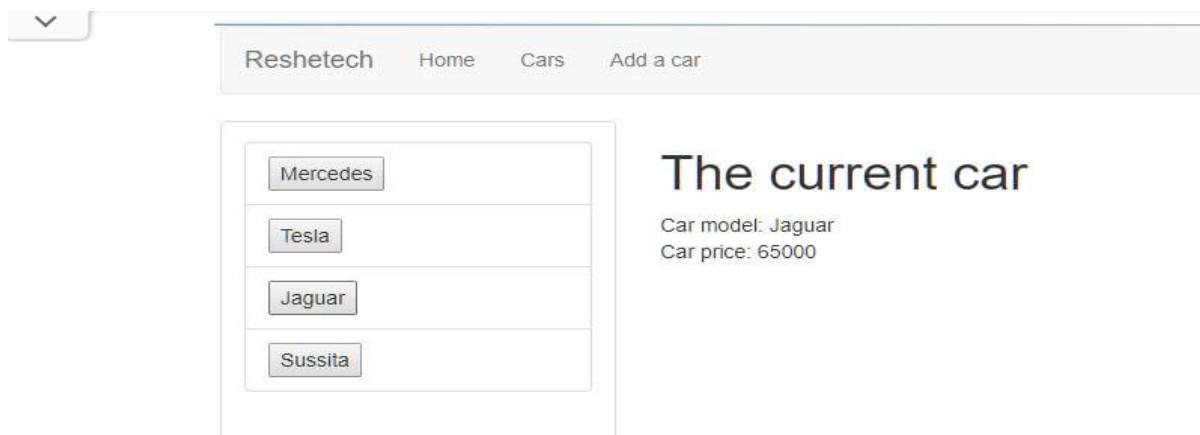
`src/app/cars/cars.component.html`

```
<ul>
<li *ngFor="let car of cars">
    <button (click)="loadCar(car.id)">{{car.model}}</button>
</li>
</ul>

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

**תזכורת:** זו אינה הפעם הראשונה שבה אנו נתקלים בDIRקטיבה `router-outlet`. ראיינו DIRקטיבה זו שמשמשת את כל האפליקציה. במקרה זה, DIRקטיבה משמשת רק את הקומפוננטות הבנות של הקומפוננטה `cars`.

לחיצה על אחד הcptוריהם ישיג לנו את המידע עבור אחד מדגמי המכוניות (לצורך עיצוב האפליקציה השתמשתי ב-`bootstrap`).



## העברת מידע באמצעות האש (hash) ופרמטרים בכתבota (queryParams)

עד כה רأינו כיצד להעביר מידע באמצעות ה-path, אבל ניתן להעביר מידע גם באמצעות פרמטרים (query parameters) שבאים אחרי סימן השאלה בכתבota וגם באמצעות hash, שבו המחרוזת שבאה אחרי הסולמית, לדוגמה:

`mywebsite.com/path/child-path?query=1#something//`

את הפרמטרים ואת ה-hash ניתן לבדוק בכתבota המבוקשת על גבי הקישורים, וגם בדרך תכונית.

## כיצד להוכיח האש (hash) ופרמטרים בכתבota ישרות על גבי הקישורים

כדי להוציא פרמטרים בכתבota על גבי הקישורים נשתמש בתוכונה queryParams שמקבלת אובייקט בו שמות הפרמטרים וערךם:

```
<a [routerLink]="'[my-path]" [queryParams]="{a:'asdf,b:'123'}">Link to /my-path?a=asdf&b=123</a>
```

הקישור יוביל לנתיב: `my-path?a=asdf&b=123`

כדי להוציא hash נכתבת (ניתן להוציא רק אחד), נשתמש בתוכונה fragment שמקבלת מחרוזת:

```
<a [routerLink]="'[my-path]" fragment="myhash">Link to /my-path#myhash
```

ה קישור יוביל לנתיב: `my-path#myhash`

## הקומפוננטה carsEditComponent

כדי להציג את הוספה המידע עבור האש (hash) והפרמטרים בכתבota, נוסיף את הקומפוננטה carsEditComponent, שכך ישמה מעיך עליה היא קומפוננטה בת שמה בתחתן הקומפוננטה ההורה cars.

קודם ננות בamusot ה-path לתקיה שבתוכה נמצאת הקומפוננטה cars:

כדי להוסיף פרמטרים לכטובת על גבי הקישורים נשתמש בתוכנה queryParams שמקבלת אובייקט ובו שמות הפרמטרים וערךם:

```
<a [routerLink]=["[my-path]" [queryParams]="{a:'asdf,b:'123'}">Link to /my-path?a=asdf&b=123</a>
```

הקישור יוביל לנטייב: my-path?a=asdf&b=123

כדי להוסיף hash לכטובת (ניתן להוסיף רק אחד), נשתמש בתוכנה fragment שמקבלת מחרוזת:

```
<a [routerLink]=["[my-path]" fragment="myhash">Link to /my-path#myhash
```

**בכתובת URL, סימן גיבוב, סימן מספר או סימן לירה (#)**  
**מןנים דפדן לנקודה ספציפית** בדף או באתר. הוא **משמש**  
**להפרדה בין URL** של אובייקט **למזהה קטע**. כאשר אתה  
**משתמש בכתובת URL עם #**, היא **לא תמיד עוברת לחلك**  
**הນכון של הדף או האתר**

ה קישור יוביל לנטייב: my-path#myhash

## carsEditComponent

כדי להציג את הוספה המידע עבור ההאש (hash) והפרמטרים לכטובת, נוסיף את הקומפוננטה carsEditComponent, שlify' ששםה מעיד עליה היא קומפוננטה בת שמה בתוכה הוראה cars.

קודם ננות בamusot ה-hi לתקן שבתוכה נמצאת הקומפוננטה cars:

```
> cd src/app/cars
```

ובתוכה הקומפוננטה cars ניצור את הקומפוננטה cars-edit:

```
> ng g c cars-edit
```

שים לב שהשתמשנו בקיצורים שמחיליפים את הפקודה הבאה:

```
> ng generate component cars-edit
```

קובץ התבנית של האפליקציה מכיל מבנה דומה לטופס, שלוינו מין את הערכים עבור המוכנית:

src/app/cars/cars-edit/cars-edit.component.html/

```
<h2>Edit a car</h2>
<div>
  <label for="model">Model:</label>
  <input [ngModel]="car.model" type="text" #carModel>
</div>
<div>
  <label for="price">Price:</label>
  <input [ngModel]="car.price" type="text" #carPrice>
</div>
<button type="submit" (click)="onSave(carModel, carPrice)">Save</button>
</div>
```

הקלקה על הכפתור Save לנקחת את הנתונים carPrice ו-carModel מהשדות, ושולחת אותם כפרמטרים למетодה onSave.

וישנו קוד ה-typescript שמשרת את הקומפוננטה:

[src/app/cars/cars-edit/cars-edit.component.ts/](#)

```

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

import { Router, ActivatedRoute } from '@angular/router';

import { CarService } from '../../car.service';

@Component({
  selector: 'app-cars-edit',
  templateUrl: './cars-edit.component.html'
})

export class CarsEditComponent implements OnInit {

  private id: string;

  car: {id:string, model:string, price:number};

  constructor(private carService:CarService, private router:Router, private activatedRoute:ActivatedRoute) { }

  ngOnInit() {
    this.activatedRoute.params.subscribe(
      (params)=> {
        this.id = params.id;
        this.car = this.carService.getById(this.id);
      }
    );
  }

  onSave(carModel, carPrice){
    this.car.model = carModel.value;
    this.car.price = carPrice.value;

    this.carService.updateById(this.car);
  }
}

```

- ניבא את האובייקטים Router-ActivatedRoute ו גם את ה-CarService .
- בטור ngOnInit נרשמו (subscribe) ל-פרמטרים שמוגעים בכתובת מה שמאפשר לנו לקלוט את ה-id, ולהשתמש בה זה כדי לשולף את המידע עבור CarService מה-
- המתודה onSave מקבלת את המידע על מודל המכונית וממחיר ומודכנת את המידע עבור המכונית בגרסה המקומית של car שמצותקת בתוך CarService .service
- הקליאן, וגם בגרסת שמצותקת ב-

src/app-routing.module.ts

```

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

import { HomeComponent } from './home/home.component';
import { CarsComponent } from './cars/cars.component';
import { CarComponent } from './cars/car/car.component';
import { CarsEditComponent } from './cars/cars-edit/cars-edit.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

const appRoutes: Routes = [
  {path: "", component: HomeComponent },
  {path: 'cars', component: CarsComponent, children:[
    {path: ':id', component: CarComponent },
    {path: ':id/edit', component: CarsEditComponent }
  ]},
  {path: 'not-found', component: PageNotFoundComponent},
  {path: '**', redirectTo: '/not-found'}
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ],
  exports: [RouterModule]
})

export class AppRoutingModule { }

```

עדין לא סימנו. כי אנחנו צריכים לנות לטופס לעדכנת המכוניות. ולשם כך, נוסיף כפתור עריכה לקומפוננטה car:

src/app/cars/car/car.component.html

```

<h1>The current car</h1>
<p>Car model: {{ car.model }}</p>
<p>Car price: {{ car.price }}</p>
<p><button (click)="loadEdit()">Edit</button></p>

```

Reshetech Home Cars Add a car

Mercedes
Tesla
Jaguar
Sussita

## The current car

Car model: Mercedes

Car price: 40000

Edit

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

import { ActivatedRoute, Router } from '@angular/router';

import { CarService } from '../car.service';

@Component({
  selector: 'app-car',
  templateUrl: 'car.component.html'
})
export class CarComponent implements OnInit {

  car: {id:string, model:string, price:number};
  id : "";

  constructor(private carService:CarService, private activatedRoute:ActivatedRoute, private router:Router) { }

  ngOnInit() {
    this.activatedRoute
      .params
      .subscribe(params => {
        this.id = params['id'] || "";
        if(this.id != ""){
          this.car = this.carService.getById(this.id);
        }
      });
  }

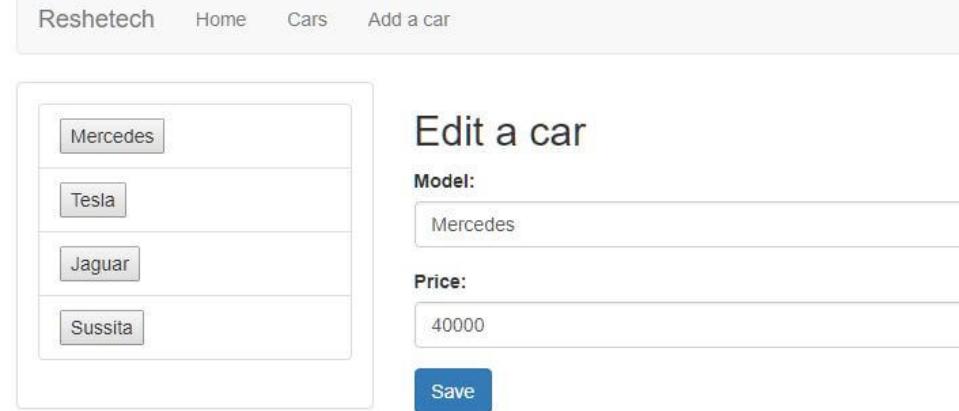
  loadEdit(){
    this.router.navigate(['edit'], {relativeTo: this.activatedRoute});
  }
}
```

שיםו לב! הכווית נעשה באמצעות המתוודה `navigate`, אבל הפעם הנתיב הוא 'חס' ולא 'אבסולוטי', ולכן, נדרש להעביר פרמטר נוסף:

```
{relativeTo: this.activatedRoute}
```

שאומר לאנגולר לבדוק את ה-`edit` בסוף הנתיב הנוכחי.

לחיצה על הקישור, תוביל לטופס ערך.



ב חלק הבא נלמד כיצד להעביר באופן תכונתי מידע מסווג פרמטרים והאש לכתובות.

## כיצד להדיביך האש (hash) ופרמטרים לכתובות באמצעות תכונות

כדי להדיביך לכתובות פרמטרים באופן תכונתי נעביר ל-`router.navigate` אובייקט שבו נגדיר את ה-`queryParams`:

```
router.navigate(['/cars'], { queryParams: {msgCode: '1', msgName: 'success'}})
```

זה יאפשר לנו לכתובת:

`cars?msgCode=1&msgName=success/`

כדי להדיביך לכתובות האש נוסיף לאובייקט :

```
router.navigate(['/cars'], {fragment: myhash})
```

זה יגרום לנוסיף לכתובות:

`cars#myhash/`

ונitinן לשלב פרמטרים והאש. לדוגמה:

```
router.navigate(['/cars'], {queryParams: {msg:1}, fragment: myhash})
```

מה שיאפשר לנו לנוסיף לכתובות:

`cars?msg=1#myhash/`

```
onSave(carModel, carPrice){
  this.car.model = carModel.value;
  this.car.price = carPrice.value;

  this.carService.updateById(this.car);

  this.router.navigate(['/cars'], {queryParams: {messageCode:'1'}, fragment: 'success'});
}
```

הפעלת הקוד שהוספנו תגרום לניפוי כתובות -

`cars?messageCode=1#success`

פרק זה למדנו כיצד להעביר פרמטרים והASH באמצעות הכתובת, ובפרק הבא נלמד כיצד לתפוא את המידע שהועבר בכתובת.

## תפיסת המידע שעבר באמצעות האש (hash) ופרמטרים בכתובת (query params)

אחרי שלמדנו כיצד להעביר פרמטרים בכתובת ובאמצעות האש, הגיע הזמן ללמידה כיצד לתפוא את המידע הוא בא-  
`activatedRoute` של הקומponentה מקבלת, לצורך זה נדרש `OnInit` כרך נראית "תפיסה" של הפרמטר `messageCode`:

```
this.activatedRoute
  .queryParams
  .subscribe((params: Array) => {
    this.messageType = +params['messageCode'] || 0;
  });

```

- נרשמים ל-`activatedRoute.queryParams`, ומצפים לקבל פרמטר שהוא מערך של מחוזות (אם אינם מעבירים מספר דרך הכתובת הוא הופך למחוזות).
- אם מעבירים מספר דרך הכתובת, הוא יփוך למחוזות, וכן כשנקלוט אותו נוסיף + לפני כדי להפוך אותו למספר. וכך:

```
this.messageType = +params['messageType'];
```

- נוסיף אפשרות בירית מחדל שווה לאפס באמצעות:

```
this.messageType = +params['messageType'] || 0;
```

כך נראה תפיסה של האש ששתן :messageClass

```
this.activatedRoute
  .fragment
  .subscribe((fragment: string) => {
    this.messageClass = fragment || 'info';
  });
}
```

נרשמים ל-activatedRoute.fragment, ומצפים לקבל פרמטר שהוא מחוץ ל-  
'info' אפשרות ברירת מחדל שהוא info בדרך הבאה:

```
this.messageClass = fragment || 'info';
```

ועכשיו, לחבר את הכל לתוך קוד ה-`ts` של הקומפוננטה `cars`:

`src/app/cars/cars.component.ts/`

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

import { Router, ActivatedRoute } from '@angular/router';

import { CarService } from './car.service';

@Component({
  selector: 'app-cars',
  templateUrl: './cars.component.html'
})

export class CarsComponent implements OnInit {

  cars: Array<{id:string, model:string, price:number}>;
}
```



```
export class CarsComponent implements OnInit {

  cars: Array<{id:string, model:string, price:number}>;
  msgs = {0: "", 1: 'A car was edited', 2: 'A car was created', 3: 'A car was deleted'};
  msgCode = 0;
  msgClass = 'info';
  msg = "";

  constructor(private carService:CarService, private router:Router, private activatedRoute:ActivatedRoute) { }

  ngOnInit() {
    this.getCars();

    this.activatedRoute
      .queryParams
      .subscribe((params: Array<string>) => {
        this.msgCode = +params['messageCode'] || 0;
        this.msg = this.msgs[this.msgCode];
      });

    this.activatedRoute
      .fragment
      .subscribe((fragment: string) => {
        this.msgClass = fragment || 'info';
      });
  }

  getCars(){
    this.cars = this.carService.getAll();
  }

  loadCar(id: string){
    this.router.navigate(['/cars',id]);
  }
}
```

src/app/cars/cars.component.html

```

<div class="row">
  <div>
    <ul class="list-group">
      <li *ngFor="let car of cars">
        <a (click)="loadCar(car.id)">{{car.model}}</a>
      </li>
    </ul>
  </div>
  <div>
    <div *ngIf="msg != ''" class="alert" [ngClass]="'alert-' + msgClass"><p>{{msg}}</p></div>

    <router-outlet></router-outlet>
  </div>
</div>

```

באמצעות הדריקטיבה `if`ogl אוחנו מודדים שקיים `msg` שהוא אינו מחוץ ריקה, ואת הקלאס נקבע על ידי קשרה לתוכנה `msgClass`.

כשתסינו את ערך הטופס, האפליקציה תעביר אתכם לדף `cars` בו תראו את ההודעה הבאה, שתוכנה והקלאס שלה נקבעים על ידי הכתובת.

Reshetech Home Cars Add a car



```

9 const routes: Routes = [
0   {path: '', component: HomeComponent},
1   {path: 'cars', component: CarsComponent, children:[
2     {path: ':Myid', component: CarComponent},
3     {path: ':Myid/edit', component: CarsEditComponent}
4   ]},
5   {path: 'not-found', component: PageNotFoundComponent},
6   {path: '**', redirectTo: '/not-found'}
7 ];
8

```

סימון נתיב  
динמי הוא ->

## מניעת גישה מנתיבים באפליקציה של canActivate בAngular

מחבר: יוסי בן הרוש בתאריך: 28.10.2017

במדריך הקודם למדנו לעבד עם נתיבים (דף) באפליקציה האנגולרית באמצעות **routing**, אבל כיצד נמנע מהמשתמשים באפליקציה כניסה לדפים מסוימים? להגמה, כיצד נטיר גישה לדפים מסוימים רק למשתמש מזוההalogin. לצורך כך, משתמש ב- **guard**. **CanActivate Guard** ה- **service**.

### עד 1: נוסיף את קובץ auth-guard.ts

יצור קובץ auth.guard בתיקיית השורש של האפליקציה:

/src/app/auth.guard.ts

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

export class AuthGuard implements CanActivate {
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return true;
  }
}
```

- הקלאס מיישם את שמו **CanActivate** שאוות אוממיות מ-interface angular/router.
- כדי לישם את ה-interface, הקלאס חייב לכלול מתודה canActivate שמקבלת שני פרמטרים: **ActivatedRouteSnapshot** ו- **RouterStateSnapshot**.
- מהין מגיעים הפרמטרים? Angular טוען את ה-guard לפני שנכנסים לניב, ומספק לו את הפרמטרים.
- המתודה צפיה להחזיר בוליאני. תחזיר true במקרה שהמשתמש רשאי להיכנס, ו-false שמנע כניסה של המשתמש.

## צעד 2: נוסיף את ה-guard לרשימה providers של האפליקציה

app.module.ts

```
providers: [CarService,AuthGuard],  
bootstrap: [AppComponent]  
})
```

מכיון שהוא service ניבא אותו לתוך ה-providers.

## צעד 3: שארחיו לדיזיינו המשתמשים

ניצור service כדי לנהל את המשתמשים, ולהבחן בין משתמש שנכנס למערכת לבין מי שאינו מזוהה.

/src/app/auth.service.ts

```
export class AuthService {  
  loggedIn = false;  
  
  isAuthenticated(){  
    return loggedIn;  
  }  
}
```

הmethodה isAuthenticated מוחזירה את הערך של המשתנה loggedIn שאומר לנו האם המשתמש מזוהה במערכת.

## צעד 4: לחבר את ה-services עם האפליקציה

src/app/app.module.ts

```
providers: [CarService,AuthService,AuthGuard],  
bootstrap: [AppComponent]  
})  
export class AppModule { }
```



לא מספיק ליבא את ה-services, צריך להוסיף אותם לרשימה providers.

## עד 5: נזיריק ל-guard את הקלאס שמננו הוא מקבל מידע

ארך ה-guard ידע להבחן בין משתמש מזווה ולא מזווה? הוא צריך לקבל מידע מ AuthService. لكن נזיריק את ה-guard.

[/src/app/auth.guard.ts](#)

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from
  '@angular/router';

import { AuthService } from './auth.service';
@Injectable()
export class AuthGuard implements CanActivate {
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return true;
  }
}
```

הdkorutor Injectable מאפשר לקלאס לקבל מידע מקרים אחרים. באפליקציה שלנו, אנחנו מעוניינים לדעת האם המשתמש מזווה, אך אנחנו צריכים ליבא את ה AuthService אל תוך ה-guard.

אחרי שייבנו את ה AuthService אנחנו יכולים לבדוק באמצעותו האם המשתמש מזווה. אם המשתמש מזווה נחזיר true, ואם אין מזווה נפנה אותו לדף הבית.

[/src/app/auth.guard.ts](#)

```
import { CanActivate } from '@angular/router';

import { Injectable } from '@angular/core';
import { AuthService } from './auth-service';
import { Router } from '@angular/router';

@Injectable()
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) {}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    if(this.authService.isAuthenticated()) {
      return true;
    } else {
      router.navigate("/");
      return false;
    }
  }
}
```

אם המשתמש מזווה באמצעות ה AuthService הוא יכול להיכנס לנútבי כי המתודה canActivate תחזיר true. ואם אינו מזווה, תפתחו לדף הבית באמצעות router.navigate("/").

## צעד 6: נגדיר את הנתיבים שעלייהם נשמר באמצעות canActivate

בקובץ app-routing.module.ts, נגדיר canActivate על הנתיבים שאחננו מעוניינים להגן עליהם.

**app-routing.module.ts**

```
import { AuthGuard } from './auth.guard';

const appRoutes: Routes = [
    {path: "", component: HomeComponent },
    {path: 'cars', component: CarsComponent,
canActivate: [AuthGuard],
children:[
        {path: 'id', component: CarComponent },
        {path: 'id/edit', component: CarsEditComponent }
    ],
    {path: 'not-found', component: PageNotFoundComponent},
    {path: '**', redirectTo: '/not-found'}
];

```

- canActivate ב途וך הנתיב מקבל מערך של כל ה-guards שבhem הוא צריך להשתמש.
  - הוא מונע גישה משתמשים שאינם מורשים גם לנתיב שעליו הוא יושב וגם לכל הנתיבים הילדיים.
- כמפורט מהשימוש ב-guard canActivate משמש שאינו מזוהה במערכת שינסה להיכנס לדף יפונה לדף הבית. רק אחרי שעשה לוגאין, הוא יוכל להיכנס לדף.

## שימוש ב-עוקב קיימים ומשלנו ב-Angular

מחבר: יוסי בן החוש בתאריך: 11.10.2017

(צינורות) של JavaScript המשמשים אותנו כדי לسان מידע וכדי לפרט מידע (להציג את המידע בצורה הרצויה לנו). ה-pipes **Pipe** מתחלקים לשניים. כאשרה שפה ייימורק מספק לנו, ו-pipes שנחנכו כתובים. את שני הסוגים נסביר במדריך.

רוצים דוגמאות ל-pipes? בבקשה.

- התאריכים שמתקבלים מממד הנתונים הם בפורמט שאינו ידידותי למשתמשים, ולכן נרצה לשנות את אופן הציגתם לאולש.
- נרצה לשנות את המחרירים ולהוסיף פסיקים בין האלפים, ואית סימן המטבח (ש"ח, דולר או ביטקוין).
- נרצה לסנן פריטים מתוך רשימה של פריטים מסווגים לנו במערך.

נתחיל מהסביר על **pipe** מובנים שמספק אנגולר, ואח"כ נלמד כיצד לכתוב סוקון משלנו.

### לא פתרונות Pipe

את העוקב מוסיף למשתנה ב-.html בתוך הסוגרים המסתולסים. לדוגמה, נוסיף למשתנה title את ה-**pipe** שנותו lowercase.

`src/app/app.component.ts`

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'List Of Cars';
}
```

`src/app/app.component.html`

```
<h2>{{ title | lowercase }}</h2>
```

כך זה נראה כשרץ על הדפדפן:

list of cars

- המשתנה title יוצג על גבי המסך כשל האותיות קטנות (uppercase).
- את ה-**pipe** מוסיףם לערך שמודפס ב-.html.
- בין העוקב והמשתנה שעליו הוא עובד מפheid קוו מפheid אנכי (זוקן אנגלית).

## 2. Pipe שמקבלים פרמטרים Pipe.

דוגמא ל-`pipe` שמקבל פרמטר הוא `currency`, משמש כדי להוציא פסיקים של אלפיים וגם את סימן המטבע. בסוף לדוגמה האנגלורית מערך שמכיל מודלים ומחירים של מכוניות. ואח"כ מדפיס את המחירים בתוך לולאה ונפעיל את ה- `pipe` ששמו `currency`.

The screenshot shows a code editor with two tabs:

- src/app/app.component.ts**
- src/app/app.component.html**

**Content of app.component.ts:**

```

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'List Of Cars';

  cars = [
    {model: 'Tesla', price: 30000},
    {model: 'BMW', price: 47000},
    {model: 'Jaguar', price: 200000},
    {model: 'Ferrari', price: 400000},
    {model: 'Sussita', price: 4000}
  ];
}

```

**Content of app.component.html:**

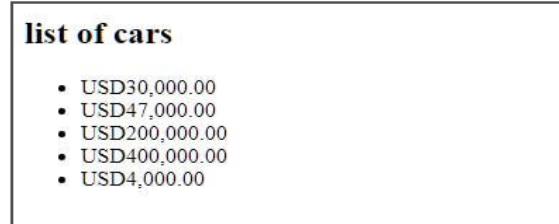
```

<h2>{{ title | lowercase }}</h2>

<ul>
  <li *ngFor="let car of cars">
    {{car.price | currency}}
  </li>
</ul>

```

כך זה נראה:



הערך בירית המחדל שמציג ה-`pipe` הוא בדולרים.  
אם נרצה להוציא סימן אחר, דוגמת ש"ח, אנו יכולים לעשות זאת על ידי העברת פרמטר.

```
<h2>{{ title | lowercase }}</h2>

<ul>
  <li *ngFor="let car of cars">
    {{car.price | currency:'NIS'}}
  </li>
</ul>
```

כך זה נראה כשרירות על הדף:

### list of cars

- NIS30,000.00
- NIS47,000.00
- NIS200,000.00
- NIS400,000.00
- NIS4,000.00

\* העברת פרמטר דוגמת סימן ה-₪ תעבור ב-5 Angular אבל לא בגרסאות מוקדמות יותר שעבורם נסתפק בסימן הבינ"ל של מטבעות באמצעות 3 אותיות.

כדי לקבוע את מספר הספרות אחרי הנקודה ניתן להעביר פרמטר שלישי:

```
<ul>
  <li *ngFor="let car of cars">
    {{car.price | currency:'NIS':'1.2-2'}}
  </li>
</ul>
```

והתוצאה לפניכם:

### list of cars

- NIS30,000.00
- NIS47,000.00
- NIS200,000.00
- NIS400,000.00
- NIS4,000.00

המשמעות של 1.2-2 היא:

<b>Currency – מחיר</b>
<b>uppercase – אותיות גדולות</b>
<b>lowercase – אותיות קטנות</b>

- לפחות ספרה אחת לפני הנקודה
- לא פחות משתי ספרות אחרי הנקודה
- לא יותר משתי ספרות אחרי הנקודה

ניתן לצרף מספר sedopin ביחד לשרשרת של הוראות. לדוגמה:

```
{{ car.price | currency:'NIS' | lowercase }}
```

זו התוצאה:

### list of cars

- nis30,000.00
- nis47,000.00
- nis200,000.00
- nis400,000.00
- nis4,000.00

```
{{ car.price | lowercase | currency:'NIS' }}
```

Error

מן סדר ישים sedopin על ידי אנגולר הוא משמאלי לימין.

## 4. ה-Pipe שמצויג ערכים ממקור א-סינכרוני

כשננסח להציג משתנה שמקורו א-סינכרוני, דוגמת ערך שmagiu מ-promise או מ-observable , נתקל בבעיה. בדוגמה להלן, ערך המשתנה welcomeMsg מוגדר באופן א-סינכרוני באמצעות promise שיוורה רק 3 שניות אחרי שהדף כבר מופיע.



src/app/app.component.ts/

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  welcomeMsg = new Promise((resolve, reject)=>{
    setTimeout(()=>{
      resolve("Hello, car owner!");
    }, 3000)
  })
}
```

src/app/app.component.html/

```
<p>{{ welcomeMsg }}</p>
```

קוד זה לא ידפיס למסך את ערך המשתנה בגלל שהוא מגיע מ-promise באחור לזמן יצירת הדף, ולכן אין זמן כשההאפליקציה יוצרת את הדף. במקום ערך המשתנה יודפס למסך:

[object Promise]

זו בעיה שנוצרה בגלל שההאפליקציה לא ידעה שעלייה לחכות לשינוי בערך המשתנה בזמן קלשוו בעתיד כי המשתנה הוא א-סינכרוני. כדי לפתור את הבעיה נשתמש ב-עוקן ששם **async**.

src/app/app.component.html/

```
<p>{{ welcomeMsg | async }}</p>
```

כך זה נראה אחרי שפתרנו את הבעיה, וה-promise סיפק את ערך המשתנה:

Hello, car owner!

## JavaScript Promise Object

A JavaScript Promise object contains both the producing code and calls to the consuming code:

### Promise Syntax

```
let myPromise = new Promise(function(myResolve, myReject) {
  // "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject(); // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

When the producing code obtains the result, it should call one of the two callbacks:

Result	Call
Success	myResolve(result value)
Error	myReject(error object)

### Example

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;

  // The producing code (this may take some time)

  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});

myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

### Waiting for a Timeout

#### Example Using Callback

```
setTimeout(function() { myFunction("I love You !!!"); }, 3000);

function myFunction(value) {
  document.getElementById("demo").innerHTML = value;
}
```

[Try it Yourself »](#)

#### Example Using Promise

```
let myPromise = new Promise(function(myResolve, myReject) {
  setTimeout(function() { myResolve("I love You !!!"); }, 3000);
});

myPromise.then(function(value) {
  document.getElementById("demo").innerHTML = value;
});
```

אנחנו יכולים לכתוב סקופ משלנו. ובחלק זה של המדריך נפתח סקופ שיקוצר את מספר התווים במחוץ לאותו גדי.

כדי להוסיף סקופ חדש במשחק שורת הפקודות זו בפקודה שזה התחביר שלו:

```
> ng generate pipe [pipeName]
```

כדי להוסיף סקופ חדש ששמו shortString, נשתמש בפקודה:

```
> ng generate pipe shortString
```

הרצת הפקודה תגרום להוספת הקובץ:



`short-string.pipe.ts`

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

@Pipe({
  name: 'shortString'
})
export class ShortStringPipe implements PipeTransform {
  transform(value: any, args?: any): any {
    return null;
  }
}
```

- בדקוטורו מגדרים את שם ה-`pipe`.
- ה-`pipe` מיישם את ה-`interface PipeTransform`, ולכן הוא חייב שתהיה לו מетодה `transform`, שמקבלת ערך של משתנה ופרמטרים נוספים שאינם הכרחיים. ה-`pipe` חייב להחזיר ערך.

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

@Pipe({
  name: 'shortString'
})
export class ShortStringPipe implements PipeTransform {
  transform(str: string, len: number): any {
    let newStr = str.substr(0, len);

    if(str.length > len){
      newStr += '...';
    }

    return newStr;
  }
}
```

- ה-אפקט מקבל שני פרמטרים. מחוזת (str), ואורך רצוי של ה-`output`.
- במידה ואורך המחרוזת ארוך יותר מהאורך הרצוי, ה-אפקט יוסיף 3 נקודות.

כדי לידע את האפליקציה האנגולרית אודות ה-**code** שהוסףנו צריך ליבא אותו לקובץ המודול.

[src/app/app.module.ts](#)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { ShortStringPipe } from './short-string.pipe';

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



- צריך ליבא את ה-**code** וגם להוסיף לרשימה ה-**declarations**.
- במידה והשתמשנו ב-**zo** כדי ליצור את ה-**pipe**, הוא יבצע את הייבוא בשילומו.

נשתחם ב-**pipe** ב-**html**

[src/app/app.component.html/](#)

```
<h2>{{ title | lowercase | shortString:5 }}</h2>
```

זו התוצאה:

welco...

Pipes יכולים לבצע סינון של מידע לפי הקритריונים שאנו מגדירים.

באפליקציה שנפתחה בחלק זה של המדריך, נרצה לסנן מידע מסוים מתוך רשימת מכוניות את אילו מכוניות נמוך מהמחיר שהזין המשתמש.

נתחיל מהוספת מערך המכוניות לקומפוננטה:

src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'List Of Cars';

  welcomeMsg = new Promise((resolve, reject)=>{
    setTimeout(()=>{
      resolve("Hello, car owner!");
    }, 3000)
  })

  cars = [
    {model: 'Tesla', price: 30000},
    {model: 'BMW', price: 47000},
    {model: 'Jaguar', price: 200000},
    {model: 'Ferrari', price: 400000},
    {model: 'Sussita', price: 4000}
  ];
}
```

נשתמש ב-`<ng>` כדי ליצור את ה-pipe שמו less-than שבו נשתמש כדי לסנן את המכוניות שמחירם נמוך מהמספר שיזין המשתמש.

> `ng generate pipe lessThan`

זה קוד ה-`es6`:

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

@Pipe({
  name: 'lessThan'
})
export class LessThanPipe implements PipeTransform {
  transform(arr: any, number: number, param: string): any {
    let filteredArray = [];

    if(arr.length === 0 || number < 1 || typeof param !== 'string') {
      return arr;
    }

    for(var i in arr){
      if(arr[i][param] <= number){
        filteredArray.push(arr[i]);
      }
    }

    return filteredArray;
  }
}
```

- ה-`es6` קיבל מערך ושי פרמטרים. `number` שהוא המחיר המקורי, ו-`param` שהוא שם השדה לפיו אנו מעוניינים לסנן.
- ולאלה, שעובדת על כל אחד מפריטי המערך, מסננת את כל הפריטים שבهم ערך השדה נמוך מהמוקבב, ומחזקת אותם למערך, שהפונקציה מחזירה.

מוסיף לקובץ ה-`html` את רישימת המכוניות ואת השדה שלתוכו יזין המשתמש את המחיר הגבוה ביותר שהוא מוכן לשלים תמורה מכונית.

`src/app/app.component.html/`

```
<h2>{{ title | lowercase | shortString:5 }}</h2>

<p>{{ welcomeMsg | async }}</p>

<p><label>Prices less than</label> <input type="text" [(ngModel)]="filterByPrice"></p>
<ul>
  <li *ngFor="let car of cars | lessThan:filterByPrice:'price'">
    <p>{{car.model}} <i>{{car.price}}</i></p>
  </li>
</ul>
```

- הערך שאנו מזין המשמש מועבר לעיבוד של אנגולר באמצעות `>this.filterByPrice` למשתנה `filterByPrice`.
- הפילטר מסנן את הרשימה, והוא מקבל שני פרמטרים. את `filterByPrice` ואת שם השדה `price`, שלו מותבצע הסינון.

```
@Pipe({
  name: 'getCarsLowerPrice'
})
export class GetCarsLowerPricePipe implements PipeTransform {

  transform(arr:any, maxPrice:number , param_nameProp:string): any {
    let filteredArray = [] ;
```

לשים לב **שלא יחזיר**  
unknown

## פיתוח טפסים באפליקציה מבוססת אングולר

מחבר: יוסי בן הרוש בתאריך: 08.12.2018

[Angular form, NgForm and two-way data binding](#)

אחד המרכיבים החשובים ביותר בכל אפליקציה הוא טפס html שמאפשר למשתמשים אינטראקטיבית עם האתר. במדריך זה נלמד את הדרך של Angular ליצר טפסים חכמים ו互動-אקטיביים במיוחד.

הדוגמה במודול היא טפס שלתוכו אנחנו מזינים את הפרטים של מודל מכונית, דוגמת שם המודל, והאם יש למוכנית גג פתוח.

### 1. ניצור מחלקה CarModel

נתחיל מיצירת המחלקה CarModel, שמכילה את השדות של מודל המכונית, ולצורך כך נעזר ב-CLI. לחצו על הקישור כדי ללמידה כיצד להשתמש ב-CLI באפליקציה אングולרית.

> ng generate class CarModel

```
export class CarModel {
    constructor(public id : number,
                public name: string,
                public motor: string,
                public hasSunroof : boolean
    ){}
}
```

המחלקה מכילה את השדות אתכם נעבד בטופס. מזהה המודל (id), שם המודל, סוג המנוע, והאם המודל מצויד בגג פתוח.

כדי לעבוד עם טפסים אנגולריים באפליקציה ניבא את ה FormsModule למודול השורש של האפליקציה.

#### app.module.ts

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

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

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

לא מספיק רק לייבא את המודול, צריך גם להויסף אותו למערך imports.

### 3. נכתוב את הקוד שיישמש את הקומפוננטה

נוסיף את הקוד הבסיסי לקובץ המחלקה של הקומפוננטה.

**app.component.ts**

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

import { CarModel } from './car-model';

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

  motors = ['petrol', 'electric', 'unknown'];

  carModel = new CarModel(1, "", "", false);

  submitted = false;

  onSubmit(){
    this.submitted = true;
  }

  getCurrentModel() {
    return JSON.stringify(this.carModel);
  }
}
```

- המערך `motors` כולל סוגי מנועים שונים (דלק, חשמל או לא יודע).
- האובייקט `carModel` מרכיב את הklass `CarModel`, והוא ריק כרגע. כי ריק תחילה.
- שימושים את הטופס המתודה `onSubmit()` משנה את ערכו של המשתנה `submitted` מ-`false` ל-`true`.
- המתודה `getCurrentModel()` מאפשרת לנו לעקוב אחרי המידע במודול בכל רגע.



```
<form>
  <label>Name</label>
  <input name="name" required>

  <br />
  <select name="motor" required>
    <option *ngFor="let motor of motors" [value]="motor">{{motor}}</option>
  </select>

  <br />
  <label><input type="radio" name="hasSunroof" value="true">Has sunroof</label>
  <label><input type="radio" name="hasSunroof" value="false">No sunroof</label>

  <br />
  <input type="submit" value="Save">
</form>
```

- נקבע את הטופס בתגיiot form כדי שאנגולר יזהה את הטופס.
- שם של השדות בטופס ה-`html` הוא בהתאם לשם השדות במודול (`name`, `model`, `hasSubroof`).
- את רישימת הבחירה ניצור בתוך לולאת `ngFor`.
- השדות הכרחיים וכן הוסףו להם `required`. זה מספיק כדי שאנגולר יטפל בולידציה של השדות.

## 5. נסיף קישירה דו-כיוונית

כדי לגשת למידע בשדות נשתמש בקישור דו-כיוונית. קישירה דו-כיוונית מאפשרת לשינויים בקוד להראות באופן פלאי בטופס, ולמידע שמיון המשתמש לטופס לעדכן מיד את קוד המחלקה.

תחביר הקשירה הדו-כיוונית הוא:

```
[ (ngModel) ]="fieldName"
```

כח-`name` `fieldName` הוא שמו של השדה בקוד הקומפוננטה.

כדי שה קישירה תעבור חובה להוסיף לשדה שם "יחוד".

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

שם הייחודי חינוי מפני שהוא מזהה את הקונטROL שסמן מגיע המידע.

נסיף את הקשירה הדו-כיוונית לטופס בכך שנוסיף `name` ו-`ngModel` לכל שדה:

```
<form>
  <label>Name</label>
  <input name="name" [(ngModel)]="carModel.name" required>

  <br />
  <select name="motor" [(ngModel)]="carModel.motor" required>
    <option *ngFor="let motor of motors" [value]="motor">{{motor}}</option>
  </select>

  <br />
  <label><input type="radio" name="hasSunroof" [(ngModel)]="carModel.hasSunroof" [value]="true">Has sunroof</label>
  <label><input type="radio" name="hasSunroof" [(ngModel)]="carModel.hasSunroof" [value]="false">No sunroof</label>

  <br />
  <input type="submit" value="Save">
</form>
```

## 6. עדכון המודול בזמן אמת

נשתמש בMETHOD getcurrentModel כדי לעקוב אחר מצב המידע במודול. לשם כך, נשבץ קרייה לפונקציה בתוך קובץ ה-.html של הקונטROLLER.

```
<pre>{{ getcurrentModel() }}</pre>
```

שיםו לב, שבו בזמן שאנחנו מזינים את הנתונים לשדות בטופס, מתעדכן המידע שמציגת הפונקציה getcurrentModel כי הקשירה הדזו-כיוונית משנה מיד את המידע במודול.

## Form Car Model

Name

Tesla

petrol

electric

unknown

NO sunroof

SAVE

```
{"id":1,"name":"Tesla","motor":"","hasSunroof":false}
```

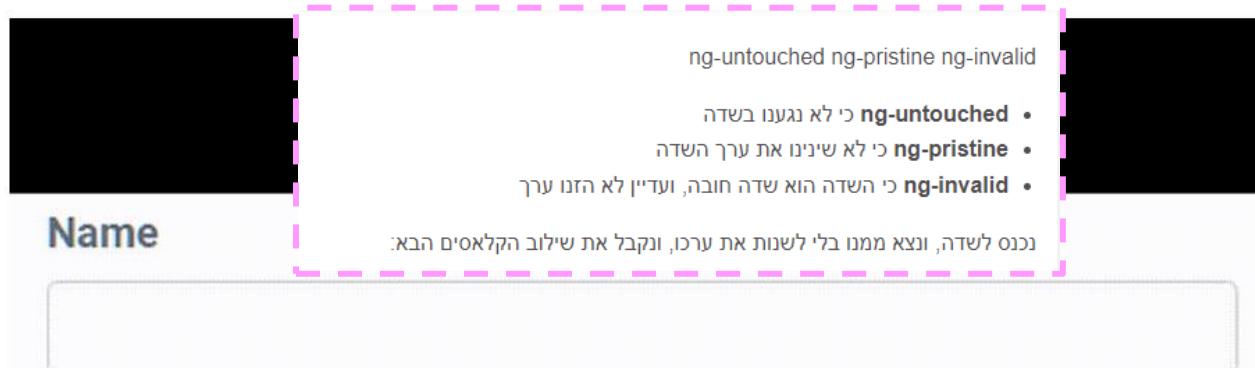
מכיוון שייבנו את NgForm למודול השורש של האפליקציה בצעד השני, והקפדנו שהטופס ימצא בין תגיוט form, אנגולר הלביש באופן אוטומטי את הדירקטיבה NgForm על הטופס. הדירקטיבה מחזיקה את המידע בתופס, ועקבות אחריו מציב הזנת השדות, ומציב הולידציה של השדות ושל הטופס.

אנחנו יכולים לראות את השינוי במצב השדות אם נעקוב אחר השינוי בשמות הקלאסים שהאפליקציה הצמידה לכל שדה.

כדי לעקוב אחר המצב של אחד השדותemossoif עליו local reference על השדה name. נdfsit את trackName.local reference ששםו trackName על השדה name. נdfsit את הקלאסים מיד מתוך שדה באופן הבא:

```
<input type="text" name="name" [(ngModel)]="carModel.name" required #trackName>
{{trackName.className}}
```

נרענן את הדף, ונראה את שמות הקלאסים שאנגולר צירף לשדות:



ng-untouched ng-pristine ng-invalid



- ng-touched ng-dirty ng-valid
- **ng-touched** כי נכנסנו לתוך השדה
  - **ng-dirty** כי שינוינו את ערך השדה
  - **ng-valid** כי הערך עבר את הולידציה של required

- ng-pristine ng-invalid ng-touched
- **ng-pristine** כי לא שינוינו את ערך השדה
  - **ng-invalid** כי השדה הוא שדה חובה, ועדין לא הזנו ערך
  - **ng-touched** כי נכנסנו לתוך השדה

קלידי מספר תווים לתוך השדה, ואילו הקלאסים שנראתה:

כפי שניתן לעקוב אחר מצבם של השדות בטופס באמצעות local reference (local reference), ניתן לעקוב אחר כל הטופס באותו השיטה.

מוסיף את ה-`carModelForm` ששמו local reference על התגית הפתוחת של הטופס:

```
<form #carModelForm="ngForm">
```

הצבת ה-`NgForm` כערך של ה-`local reference` שמתיחס לטופס כולו מאפשרת לנו לעקוב אחר מצב הטופס באמצעות תכונות של המודול `FormsModule`.  
לדוגמה, בואו נבדוק מה יקרה כאשר נזיף את הקוד הבא לטופס, שמאזין לתוכנה `value` של ה-`NgForm`:

```
{{carModelForm.value | json}}
```

אנו יכולים לראות את ערכי השדות בטופס משתנים בזמן שנחנכו מזמן את הטופס.  
אפשר גם לבדוק האם הטופס כולו ולידי על ידי האזנה לתוכנה `valid`:

```
{{carModelForm.valid}}
```

מלמד אותנו האם הטופס ולידי. בהתאם לעובדה שכל השדות דרושים, נראה שככל עוד לא מילאנו את כל שלושת השדות מצב הטופס יהיה לא ולידי (ערוך). רק אחרי שנמלא את כל שלושת השדות יփוך מצב הולידציה ל-`true` (`false`).

## Form Car Model

Name

Tesla

petrol  
electric  
unknown  
 No sunroof

SAVE

**CarModelForm.value:**

```
{
  "name": "Tesla",
  "motor": "",
  "hasSunroof": "false"
}
```

**CarModelForm.valid:**

```
false
```

מיד נראה כיצד להשתמש במצבים השונים שונים לו הטופס האנגולי בשבי להנחות את המשתמש בזמן مليו הטופס, וגם כדי למנוע את הגשת הטופס במידה ואיתם ולידי.

```
<label><input type="radio" name="hasSunroof"
  [(ngModel)]="carModel.hasSunroof" [value]="true"
  [checked]="carModel.hasSunroof">
  Has sunroof
</label>
<label><input type="radio" name="hasSunroof"
  [(ngModel)]="carModel.hasSunroof" [value]="false"
  [checked]="!carModel.hasSunroof">
  No sunroof
</label>
```

## 8. אינדיקציה והוראות למשתמש בזמן אמת

נוסף פסקה שתציג הודעה שגיאה בתנאי שהשדה ריק, וכן אם וליד, כי הוא מוגדר כ-`required`, ורק בתנאי שנכנסנו לשדה.

```
<input type="text" name="name" [(ngModel)]="carModel.name" required #trackName="ngModel">
<p class="alert" *ngIf="!trackName.valid && trackName.touched">The name is required</p>
```

כדי שהודעות השגיאה בזמן אמת יעבדו, הוספנו את הדירקטיבה `ngModel` על ה-`input`, שיש לה תכונות דוגמת `valid`, `touched`, `pristine` local reference, הוספנו את הדירקטיבה `ngModel` על ה-`input`, שיש לה תכונות דוגמת `valid`, `touched`, `pristine` local reference, שאנחנו יכולים להיעזר בהם כדי להנחות את המשתמש באופן מדויק.

ניתן גם להווסף כללי CSS שיגרם לשינוי מראה הטופס לפי מצבו. לדוגמה, הוספה קו אדום סביב השדות שאינם ולידיים ובתנאי שנגעו בהם.

```
.ng-invalid.ng-touched:not(form) {
  border: 1px solid red;
}
```

## Form Car Model

Name

The name is required

גיש את הטופס באמצעות קשירת האירוע `ngSubmit` לMETHOD `Submit` בתוך הקוד.

```
<form (ngSubmit)="onSubmit()" #carModelForm="ngForm">
```

## 10. כיצד למנוע את הגשת הטופס אם אין לו לידי

ניתן למנוע הגשת טופס שאין לו לידי באמצעות קשירת התכונה `disabled` של הטופס למצוות `disabled`. ניתן לדעט אותו כי אנחנו משתמשים ב-`carModelForm`, local reference לטופס.

```
<input type="submit" value="Save" [disabled]="!carModelForm.form.valid">
```

## 11. איפוס המודל והטופס

אפשר לאפס את הטופס, כולל מחיקת תוכן שהוזן לשדות, איפוס מצב הטופס, ומיחיקת השדות במודל באמצעות המethode `reset`.

```
<form (ngSubmit)="onSubmit(); carModelForm.reset()" #carModelForm="ngForm">
```

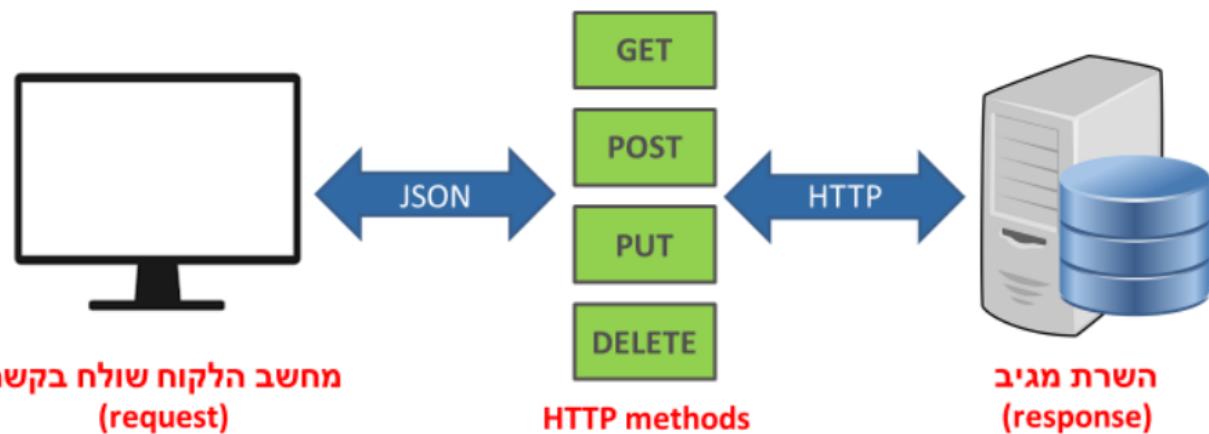
בהתגובה לאירוע `ngSubmit` יופעלו שתי מתודות. השנייה היא המethode `reset` שתאפס את הטופס והמודל בתגובה להגשת הטופס.

## מה זה REST API?

מחבר: יוסי בן הרוש בתאריך: 28.07.2018

הגרסתה האנגלית של המאמר [What is REST API? in plain English](http://PHPenthusiast.com/What_is_REST_API_in_plain_English)

(Application Program Interface) **API** הוא דרך מוסכמת לשילוח ולקבל מידע בין מחשבים. לדוגמה, אם אתה מעוניין להציג באתר שלך מפות גוגל (לצערך) הנמצאות על השרת שלך אלא על השירותים של גוגל. אז הדרך לבקש מגוגל לשולח לך את המפה המבוקשת היא באמצעות **API** שמספק נתן השירות (גוגל) שאומר לאילו כתובות אינטרנט צריך לփנות את הבקשות כדי לקבל את המידע. הבקשה היא ה-**request** שהולח האתר שלו, והתגובה המוחזרת מהשירותים של נתן השירות היא ה-**response**.



**REST** (REpresentational State Transfer) הוא API שמאגד רטט של פונקציות שמתכוונים יכולם להשתמש בהם כדי לשולח בקשות ולקבל תשובות באמצעות המתודות של פרוטוקול HTTP דוגמת GET ו-POST.

**REST API** יכול לשמש כל אתר או אפליקציה ולא משנה באיזו שפה היא כתובה מכיוון שהבקשות מבוססות על הפרוטוקול האוניברסלי של HTTP והמידע מוחזר בדרך כלל בפורמט של JSON לכל השפות השימושיות בעולם יודעות לקרוא.

**HTTP** הוא הפרוטוקול שעליינו מבוסס האינטרנט. הוא מאפשר למחשב מכל מקום בעולם לשולח בקשה לשרת מרוחק, ולקבל חזרה תגובה שאותה ניתן להציג בדפנינו.

HTTP מספק מתודות שבאמצעותם מבצעת התקשורת באינטרנט, וועלוי מtaboo REST.

לדוגמה, כדי ליצור מחדש בבלוג המחשב שלנו צריך לשולח בקשה לשרת המרוחק באמצעות Method POST שמספק HTTP. כדי להציג מאמר בודד או רשימה של מאמרים נשתמש בmethod GET. המethod PUT יכולת לשמש לעדכנת מאמר קיימ, והmethod DELETE למחיקה.

הטבלה הבאה מציגה את 4 המתודות השימושיות ביותר של פרוטוקול HTTP:

מבקש מידע משרת מרוחק	GET
שולח מידע למחשב מרוחק	POST
עדכן את המידע בשרת המרוחק	PUT
מוחק מידע מהשרת המרוחק	DELETE

את הבקשות צריך לשלוח ל-URL בשורת המרוחק. לדוגמה:

כדי לצפות ברשימה המאמרים נשלח בקשה ב-GET ל-URL:

`//something.com>/api/articles>//`

אם נרצה לקבל מאמר שה-ID שלו 1, נשלח בקשה גם כ-`GET` ל-URL:

`//something.com>/api/articles/1>//`

כדי ליצור מאמר חדש נשלח את המידע עם המethode POST לכתובת:

`//something.com>/api/articles>//`

ואל טענות. הכתובת היא אותה אחת שמשמשת להבאת רשימת המאמרים. אבל מכיוון שהmethode היא POST התוצאה היא אחרת.

כדי לעורוך את המאמר שה-ID שלו 1 נעדיף להשתמש בmethode PUT של HTTP, ונשלח בקשה ל:



`//something.com>/api/articles/1>//`

כדי למחוק מאמר משתמש בmethode DELETE, ונציין בבקשתה לשרת את ה-ID:

`//something.com>/api/articles/1>//`

המידע שמחזר מצד השירות מכיל שני חלקים:

payload – הוא גוף התגובה, לרבות בפורמט JSON. הוא מכיל את המידע שאנו יכולים להציג למשתמשים לדוגמה את רשימת המאמרים או מאמר בודד. גוף התגובה אינו חובה. לדוגמה, כשמוחקים מאמר אין טעם להחזיר את המידע שלו שכבר לא קיים על השירות.

החלק השני הוא head שכולל את המידע על הבקשתה, דוגמת כתובת ה-URL-ו-IP, זמן התגובה ועוד.

הכי חשוב מביחינטנו כפתחים הוא קוד התגובה (status code). לדוגמה, קוד 404 למצב שהמידע אינם קיימים. קוד 200 למצב שהמידע קיימים ומוחזר בגוף התגובה.

את הקודים של התגובה המוחזרת מהשרת ניתן לחלק לקבוצות:

קבוצת ה-200 (200, 201, 200) מבטאים את הצלחת העברת המידע.

קבוצת ה-300 (302, 301) מפנה את הבקשת לכתובת אחרת שמספקת את השירות המבוקש.

קבוצת ה-400 (400, 401, 403) כשהמידע לא נמצא או שהפונה אליו רשאי לקבלו.

200	העברת המידע הצל'חה
201	פריט המידע החדש נוצר על השרת, והתגובה מחייבת את כל הפרטים שלו כולל ה-pid
204	הבקשה מולאה, אבל לא מוחזר מידע בגין התגובה. לדוגמה, כשמותקים פריט מידע הוזע לצמימות - הפניה הבקשה זו והבאות בעתיד לכתובת חדשה על השרת המרוחק
301	שגיאה בבקשתה שהגיש צד הלקו ולבסוף הבקשה לא תטופל
402	נדרש תשלום
403	הבקשה תקינה, אבל השרת לא יוציא אותה אל הפעול בגלל שללקיות אין רשות המתודה אסורה. לדוגמה, כשרוצים לאסור על מחיקה של מידע.
404	לא נמצא
405	שגיאה פנימית בשרת המרוחק
500	שגיאה פנימית בשרת המרוחק

<https://github.com/waldemarnt/http-status-codes>

## מדריך 1 : httpClient באנגולר

מחבר: יוסי בן הרוש בתאריך: 11.06.2021

Learn to code Angular app with PHP backend: part 1

מדריך זה קיימת גרסה אנגלית מומלצת מאוד [HttpClientModule](#) שבא מותקן עם אנגולר.

להורד קוד האנגולר אותו נפתח במדריך

בסדרת המדריכים הזה נלמד כיצד להשתמש במודול לבניית אפליקציות Angular שכוללת זאת משתמש ודף שרת. הקוד שנפתח במדריך הראשון בסדרה מביא את המידע כיצד השרת באמצעות GET, ואחר' כמציג אותו באפליקציה האנגולרית.

במדריכים הבאים בסדרה נלמד כיצד להוסיף פריטים, לעדכן, ולמחוק.



### 1. התקנת אנגולר

אם איןכם יודעים כיצד להקים פרויקט Angular, אז כדאי שתקרוואו את המדריך [אודות התקנת אנגולר ספרותית](#) באתר.

### 2. הוספת המודול HttpClientModule לפרויקט

כדי שנitin יהיה להשתמש במודול HttpClientModule צריך ליבא אותו למודול השורש של האפליקציה:

[src/app/app.module.ts](#)

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

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

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

### 3. הגדרת הסוג הכללי `interface`

מחייב כל קוד שימושו לכתוב מethodות או שדות מסוימים. **interface** שגדיר את הסוג מכנית עבור האפליקציה שלו בתוך קובץ חדש `car.ts`:

[src/app/car.ts](#)

```
export interface Car {
  model: string;
  price: number;
  id?: number;
}
```

בdziיר, בכל פעם שנזדקק לשוג Car נהיה מחויבים לספק מידע אודות השדות `model` ו-`price`. את השדה `id`, לעומת זאת, אנחנו לא חייבים לספק כי הוא אופציוני כי שמעיד סימן השאלה לימין.

### 4. ריצוף הפונקציות שמת谦שות עם השירות ב- `service`

אומנם ניתן להתקשר ב-AJAX עם צד השירות שירות מהקומפננטות אבל זו אינה הדרך המומלצת מכיוון שהיא שורשת שימוש באמצעות הפונקציות במוקומות שונים באפליקציה. אך, עדיף להפריד את הפונקציות-`service` ייעוד. ובקרה שלנו, יהיה זה service שייעודו להעיבר מידע אודות מכניות, ושם בהתאם:

`car.service`

וניצור את ה-`service` באמצעות הקולדת הפקודה הבאה ל-CLI:

```
ng generate service car --flat
```

הפקודה מורה על יצירת ה-`service`. הדגל `flat` – מאפשר ליצור את הקבצים בתוך התיקייה הנוכחית בלי לפתח תיקייה נוספת.

בתוך קובץ ה- `service` נמצא את הקוד הבא:

[src/app/car.service.ts](#)

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

@Injectable({
  providedIn: 'root'
})
export class CarService {
```

הקורטטור `Injectable` מאפשר `service` לשתתף במערכת הזרקת התלוויות (dependency injection). זה אומר שמצד אחד הוא יכול לקבל מילויות (לדוגמה את המודול `HttpClientModule`), ומצד שני הוא יכול להיות מזמין כמלhot, לדוגמה לתוך הקומפננטות.

אבל מה זה תלות?

תמלות היא המצביע שקיים אחד תלוי בקהלואס אחר לתקופתו.

הדרך של אングולר לנוהל את התלותות באפליקציה היא באמצעות **Dependency Injection** או בקיצור **DI**, שתפקידו ליצור בשביבו את האובייקטים מוקלאים כשאנחנו זקוקים להם.

הדרך המקובלת בתוכנות מונחה עצמים ליצור אובייקטים מוקלאים היא באמצעות מילת המפתח **new**. לדוגמה, אם אנחנו מעוניינים באובייקט של **HttpClient**, אז הדרך המקובלת היא לעשות את הדבר הבא:

```
private http = new HttpClient;
```

אבל זו לא הדרך האנגולרית כי אングולר מעדיף להשתמש ב- **Dependency Injection**, שהוא זה שמנוהל בשביבנו את ייצור האובייקטים מוקלאים. כדי שה-service יוכל לבצע את תפקידו הוא נדרש להזקוף התלות של **HttpClient**, וליבוא של רכיבי קוד שונים שדרושים לניהול התקשרות עם השרת.

[src/app/car.service.ts](#)

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { map } from 'rxjs/operators';

import { Car } from './car';

@Injectable({
  providedIn: 'root',
})
export class CarService {
  constructor(private http : HttpClient) {}
}
```

הזרקת התלות נעשית ב-**constructor**, על ידי שאנחנו מגדירים את המשתנה הפרטי **http** כשייך לסוג **HttpClient**, ואז אングולר מטפל ביצירת האובייקט של **HttpClient** בשביבנו באמצעות ה-**DI**.

היבוא מספרי תץ' מאפשר לנו לעבוד עם متודות של **observable** שהוא המנוף שאנגולר עוטף באמצעותה את המידע שmagiu מהשרת. השימוש ב-**observable** במקום **call back** כדי לטפל במקרה א-סינכרוני מעניק מספר יתרונות, בהםם: ריבוי אופרטורים שמקלים על הטיפול במידע, כמו גם האפשרות להאזין למידע שהשרת פולט שוב ושוב לאחר זמן.

כדי לקבל את רשימת המכוניות שmagieha מהשרת, נוסיף את הקוד הבא לתוך ה-service:

[src/app/car.service.ts](#)

```
baseUrl = 'http://localhost/api/';

constructor(private http : HttpClient) { }

getAll() {
    return this.http.get(`${this.baseUrl}list`).pipe(
        map((res: any) => {
            return res['data'];
        })
    );
}
```

- הmethod getAll מזכה להחזיר את רשימת המכוניות עטופה בתוך Observable.
- הmethod של HttpClient שבה אנחנו משתמשים כדי להביא את המידע מה-URL בצד השרת היא get.
- המידע שוחזר מצד השרת הוא לא רשימת המכוניות, שלא אנו מצפים, אלא מערך המכוניות בתור הערך של מפתח data (מקרה נפוץ בצריכה של URL מקור חיצוני). ככה נראה הקוד שוחזר מהשרת:

```
{
  data: [
    [
      {
        id: "1",
        model: "subaru",
        price: "120000"
      },
      {
        id: "2",
        model: "mazda",
        price: "160000"
      }
    ]
  ]
}
```

- אבל אנחנו מעוניין רק החלק הפנימי, ללא המעטפת של ה-data. כדי למצות את המידע, נשתמש באופרטור map של rxjs. וכך להשתמש באופרטורים אנחנו צריכים לשגרר למתודה get של httpClient את המתודה pipe של סדרית rxjs.
- בתוך map נמזה את מערך המכוניות כדי להחזיר אותו.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { map } from 'rxjs/operators';

@Injectable({
  providedIn: 'root',
})
export class CarService {
  baseUrl = 'http://localhost/api/';

  constructor(private http : HttpClient) {}

  getAll() {
    return this.http.get(`${this.baseUrl}list`).pipe(
      map((res: any) => {
        return res['data'];
      })
    );
  }
}
```

בקומפוננטה, ח"ב להיות קוד שנרשם מיד שמווחר מהשרת באמצעות ה-service, אך בוואו נוסיף אותו:

[src/app/app.component.ts](#)

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

import { Car } from './car';
import { CarService } from './car.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  cars: Car[] = [];
  error = "";
  success = "";

  constructor(private carService: CarService) {}

  ngOnInit() {
    this.getCars();
  }

  getCars(): void {
    // here we'll write the method code
  }
}
```

- המשתנה cars מכיל את מערך המכוניות שמאյע מהשרת.
- בקונסטורקטור נזכיר את התולות של ה-service, carService, כדי שהשירות יהיה זמין לכל הפונקציות של הקונטROLLER כבר ברגע שהאובייקט נוצר מהקלאס.
- בתוך **hook** מסוג **ngOnInit** נקרא לפונקציה `getCars()`.
- **ngOnInit** היא פונקציה מיוחדת של אגולר שרצה מיד אחרי שהקונסטורקטור מסיים להזיריק את התוליות, ולכן מקום טוב לקרוא בו להבאת מערך המכוניות.

טוויף לתוכה המתודה `getCars` את הקוד שבסמצעותו אנחנו נרשמים להבאת מערך המכוניות או לקבלת שגיאה מצד השירות:

```
getCars(): void {
  this.carService
    .getAll()
    .subscribe(
      (data: Car[]) => {
        this.cars = data
        this.success = 'Operation success'
      },
      (err) => {
        console.log(err)
        this.error = err.message;
      }
    );
}
```

ה-call back הראשון בתוך הורשמה (`subscribe`) מטפל במקרה שהמידע מתקבל בהצלחה, ומציב את רשימת המכוניות במשתנה `cars`.

-call back השני מיועד לטיפול בשגיאות מצד השירות.

כשאנו נרשמים **ל-observable** באמצעות `subscribe` אנחנו מוצאים לפחות מ-3 תוצאות:

- התוצאה הראשונה היא קבלת המידע הדרוש מצד השירות (בדוגמה שלנו, מערך המכוניות),
- השנייה, שגיאה מצד השירות,
- והשלישית סיום הבאת המידע.

רוצים להעמק בהבנת **Observable**? קראו את המדריך [שמסביר את Observable](#) בשפה שווה לכל נפש.

במקרה שלנו, אנחנו מתעניינים רק בשתי האפשרויות הראשונות. זאת מכיוון שהפעולה לא צפיה להסתיימות הרשימה יכולה להתעדכן בכל זמן נתון בפריטים שונים באמצעות הטופס שהוא ייצור בהמשך המדריך.

עכשו כל מה שנשאר לנו הוא להציג את ה-`html` עם רשימת המכוניות, ועם הודעות ההצלחה והשגיאה.

```
<div *ngIf="error">{{error}}</div>
<div *ngIf="success">{{success}}</div>

<div id="theList">
  <h2>The list</h2>
  <ul>
    <li *ngFor="let car of cars">{{car.price}} {{car.model}}</li>
  </ul>
</div>
```

- הרשימה מודפסת למסך בתוך `for` של `let` `car of cars` שאוות נקבל מהשרת.

את הקוד בצד השרת כתבתי בסקריפט PHP שישוב על מסד נתונים MySQL . היכי פשוט שאפשר. העקרונות נכונים לכל שפה.

להלן קוד ה-sql לtáבלה cars שימושה לאחסון המידע באפליקציה:

```
CREATE TABLE IF NOT EXISTS `cars` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `model` varchar(255) NOT NULL DEFAULT '',
  `price` int (10) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

הקובץ הוא קובץ הקונפיגורציה של שרת ה-Apache, בו נגדיר:

1. כל להסרת סימנת php משמות הקבצים
2. headers שיאפשרו לנו להעביר מידע ולבצע פעולות על השרת למחרות שהחלק האנגלי של האפליקציה יושב על שרת נפרד בכתובת נפרדת.

### htaccess

```
# Remove the php extension from the filename
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^([^\.]*)$ $1.php [NC,L]

# Set the headers for the restful api
Header always set Access-Control-Allow-Origin
//localhost:4200
Header always set Access-Control-Max-Age "1000"
Header always set Access-Control-Allow-Headers "X-
Requested-With, Content-Type, Origin, Authorization, Accept,
Client-Security-Token, Accept-Encoding"
Header always set Access-Control-Allow-Methods
"POST, GET, OPTIONS, DELETE, PUT"
```

```
Header always set Access-Control-Allow-Origin "localhost:4200"
```

מורא לשרת לאפשר גישה למידע ולפעולות מהכתובת שעליה רצה האפליקציה האנגלרית במחשב האישי שלו.

טור ה-headers מתיירים החלפת מידע בין האפליקציה האנגלרית ועד השרת באמצעות המתוות של פרוטוקול HTTP:

לשמירת מידע חדש בצד השרת, דוגמת מכונית חדשה.	<b>POST</b>
לקבל מידע אודות פרייטי בודד או רשימת פרייטים (מכונית או מערכת מכונית).	<b>GET</b>
לעריכת מידע שכבר נשמר על השרת.	<b>PUT</b>
למחיקה.	<b>DELETE</b>

הקובץ **connect.php** מכיל קוד להתקשרות עם מסד הנתונים.

[connect.php](#)

```
<?php

// db credentials
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASS', '');
define('DB_NAME', 'angular_db');

// Connect with the database.
function connect()
{
    $connect = mysqli_connect(DB_HOST ,DB_USER ,DB_PASS ,DB_NAME);

    if (mysqli_connect_errno($connect)) {
        die("Failed to connect:" . mysqli_connect_error());
    }

    mysqli_set_charset($connect, "utf8");

    return $connect;
}

$con = connect();
```

עד השרת מכיל את הקובץ **list.php** שמקבל בקשה ב GET מצד האפליקציה האנגלרית, ומחזיר את המידע שהוא שולף מסד הנתונים.

```
<?php

/**
 * Returns the list of cars.
 */

require 'connect.php';

$cars = [];
$sql = "SELECT id, model, price FROM cars";

if($result = mysqli_query($con,$sql))
{
    $cr = 0;
    while($row = mysqli_fetch_assoc($result))
    {
        $cars[$cr]['id'] = $row['id'];
        $cars[$cr]['model'] = $row['model'];
        $cars[$cr]['price'] = $row['price'];
        $cr++;
    }

    echo json_encode(['data'=>$cars]);
}
else
{
    http_response_code(404);
}
```

## מדריך 2 : httpClient – postanganolir - post

מחבר: יוסי בן הרוש בתאריך: 11.06.2021

לארסמו האנגלית של המדריך [Angular app with PHP backend: part 2 \(POST\)](#)

מדריך זה הוא השני בסדרת המדריכים שמלמדת כיצד להשתמש ב-**HttpClient** של Angular כדי לתקשר ב-AJAX עם צד השירות של האפליקציה. במדריך נפתח טופס אングולר שלתוכו יזום המשתמשים את שם הדגם של מכונית ואות מחירה, ונשלח את המידע באמצעות POST לצד השירות.

### 1. עבודה עם טפסים – יבוא התלוויות

כדי לעבוד עם טפסים ניבא את המודול **FormsModuleModule** למודול השרוש של האפליקציה:

[src/app/app.module.ts](#)

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

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

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

וגם נוסיף את התלוות **NgForm** לרשימה התלוויות של הקומפוננטה:

135

```

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

import { Car } from './car';
import { CarService } from './car.service';

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

export class AppComponent implements OnInit {

  constructor(private carService: CarService) {}

  ngOnInit() {
    this.getCars();
  }

  getCars(): void {
    this.carService.getAll().subscribe(
      (data: Car[]) => {
        this.cars = data;
        this.success = 'Success in retrieving the list';
      },
      (err) => {
        console.log(err);
        this.error = err.message;
      }
    );
  }
}

```



## service-ה .2

ל-**service**, שהחלנו לפתח במדריך הקודם, נוסיף את המתודה **store** שקולעת את המידע מהקומפוננטה, ומשדרת אותו לצד השרת.

[src/app/car.service.ts](#)

```

store(car: Car) {
  return this.http.post(`${this.baseUrl}/store`, { data: car }).pipe(
    map((res: any) => {
      return res['data'];
    })
  );
}

```

- המתודה שמשמשת לשידור המידע לצד השרת היא **post** שמקבלת את נתיב ה-**url** בתור פרמטר ראשון וגם את האובייקט **car** שמכיל את המידע שהתקבל מהקומפוננטה.
- המידע שמוחזר מצד השרת כולל את כל המידע על המכונית החדשה שיצרמו (שם, מחיר, id) או שגיאה. מהtagובה המוחזרת צריך למצות את המידע הרלוונטי.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { map } from 'rxjs/operators';
import { Car } from './car';

@Injectable({
  providedIn: 'root',
})

export class CarService {
  baseUrl = 'http://localhost/api/';

  constructor(private http : HttpClient) {}

  getAll() {
    return this.http.get(`${this.baseUrl}list`).pipe(
      map((res: any)=> {
        return res['data'];
      })
    );
  }

  store(car: Car) {
    return this.http.post(`${this.baseUrl}/store`, { data: car }).pipe(
      map((res: any)=> {
        return res['data'];
      })
    );
  }
}
```

הקוד  
המלא



במדריך הקוד פיתחנו את המתודה `getCars` שמביאה את רשימת המכוניות. במדריך זה, נסיף לקומפוננטה את המשתנים `error` ו-`success`, שיקלטו את הודעות השגיאה וההצלחה.

[src/app/app.component.ts](#)

```
error = "";
success = "";
```

אתחל את הטופס באמצעות אתחול האובייקט `car`:

```
car: Car = {model:"", price:0};
```

שימושם את ה-`interface` `Car` הגדרכו במדריך הקוד בסדרת [Angular HttpClient](#)

[src/app/car.ts](#)

```
export interface Car {
  model: string;
  price: number;
  id?: number;
}
```

נסיף לקוד הקומפוננטה את המתודה `addCar` שקולtot את הערכים שמזינים לטופס, ושולחת את הערכים ל-`service`.

```
addCar(f: NgForm) {
  this.resetAlerts();

  this.carService.store(this.car).subscribe(
    (res: Car) => {
      // Update the list of cars
      this.cars.push(res)

      // Inform the user
      this.success = 'Created successfully';

      // Reset the form
      f.reset();
    },
    (err) => (this.error = err.message)
  );
}
```

במקרה של הצלחה, מופעלת המתודה `reset`, שמאתחלת את הטופס.

המתודה נרשמת למתודה `store` של ה-`service`, ושולחת לו את הערכים שהמשתמשים הדינו לטופס.

ההרשמה מתבצעת באמצעות `subscribe` באירוע `store` ה-`observable`.

- במקרה הראשוני, הצלחה. עליהណד למשתמש הקצה וגם לנקה את הטופס מערכים ונתיפויים
- במקרה השני, שגיאה. אותה האפליקציה מציגה למשתמש.

138

```

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

import { Car } from './car';
import { CarService } from './carservice';

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

export class AppComponent implements OnInit {
  cars: Car[] = [];
  car: Car = {model:"", price:0};

  error = "";
  success = "";

  constructor(private carService: CarService) {}

  ngOnInit() {
    this.getCars();
  }

  getCars(): void {
    this.carService.getAll().subscribe(
      (data: Car[]) => {
        this.cars = data;
        this.success = 'Success in retrieving the list';
      },
      (err) => {
        console.log(err);
        this.error = err.message;
      }
    );
  }

  addCar(f: NgForm) {
    this.resetAlerts();

    this.carService.store(this.car).subscribe(
      (res: Car) => {
        // Update the list of cars
        this.cars.push(res)

        // Inform the user
        this.success = 'Created successfully';

        // Reset the form
        f.reset();
      },
      (err) => (this.error = err.message)
    );
  }

  resetAlerts() {
    this.error = "";
    this.success = "";
  }
}

```

הקוד המלא של הקומפוננטה:  
<src/app/app.component.ts>

את הטופס נסיף מתחת לרשימה שפיתחנו, במדריך הקודם, והוא כולל שיבת חלקים:  
א. דיבים שמציגים את הודעות השגיאה והצלחה, במידה והם קיימות:

[src/app/app.component.html](#)

```
<div class="alert alert-danger" *ngIf="error">{{error}}</div>
<div class="alert alert-success" *ngIf="success">{{success}}</div>
```

ב. הטופס עצמו:

[src/app/app.component.html](#)

```
<div id="theForm">
  <h2>The form</h2>
  <form #f="ngForm" name="theForm" (submit)="addCar(f)">
    <div class="form-group">
      <label>Model</label>
      <input type="text"
        class="form-control"
        name="model"
        [(ngModel)]="car.model"
        #carModel="ngModel"
        required
        pattern="^[a-zA-Z ]+$">
      <small class="form-text text-danger" *ngIf="carModel.errors?.required && carModel.touched">
        The model name is required
      </small>
      <small class="form-text text-danger" *ngIf="carModel.errors?.pattern && carModel.touched">
        The model name can only contain the letters a-z or A-Z
      </small>
    </div>

    <div class="form-group">
      <label>Price</label>
      <input type="number"
        class="form-control"
        name="price"
        required
        [(ngModel)]="car.price"
        #carPrice="ngModel">
      <small class="form-text text-danger" *ngIf="carPrice.errors?.required && carPrice.touched">
        The price is required
      </small>
    </div>

    <button
      class="btn btn-primary btn-sm"
      [disabled]="f.invalid">Add</button>
  </form>
</div>
```



\* הטופס הוא אנגלי, ואם לא ברור איך הוא עובד אז אני ממליץ בחום לקרוא את המדריך על טפסים אנגלרים.

- ערכי השדות בטופס קשורים בקשרה דו-כוונית לאובייקט Car אשר בקומפוננטה בעזרת **.ngModel**.
- הוסףתי כליל ולידzie להשdot, kr שבמידה והמידע שזמן המשמש איתן תקין, השדה מקבל את הקלאס "danger".
- המשתמש יכול לשלוח את הטופס רק אם הוא מילא את כל השדות בהתאם לכללים. אם הטופס אינו ולידי, השילוח לא מתאפשרת. לדוגמה, כפטור השילוח יהיה במצב **disabled**.
- הגשת הטופס מעבירה את המידע מהטופס לקומפוננטה באמצעות **#f** אשר מועבר כפרמטר לפונקציה **addCar()**.

זה קוד ה-PHP שמתפל באחסון המידע בצד השרת:

[store.php](#)

```
<?php
require 'connect.php';

// Get the posted data.
$postdata = file_get_contents("php://input");

if(isset($postdata) && !empty($postdata))
{
    // Extract the data.
    $request = json_decode($postdata);

    // Validate.
    if(trim($request->data->model) === "" || (int)$request->data->price < 1)
    {
        return http_response_code(400);
    }

    // Sanitize.
    $model = mysqli_real_escape_string($con, trim($request->data->model));
    $price = mysqli_real_escape_string($con, (int)$request->data->price);

    // Store.
    $sql = "INSERT INTO `cars`(`id`, `model`, `price`) VALUES (null,'{$model}', '{$price}')";
}
```

```
if(mysqli_query($con,$sql))
{
    http_response_code(201);
    $car = [
        'model' => $model,
        'price' => $price,
        'id'   => mysqli_insert_id($con)
    ];
    echo json_encode(['data'=>$car]);
}
else
{
    http_response_code(422);
}
```

במקרה של שאיה מוחזר קוד ממושחתת ה-400.

במידה והMiami מאוחסן בהצלחה במאד הנתונים מוחזר קוד תגובה 201 גם מערך אשר כולל את ה-id של הרשומה החדשה במאד הנתונים.

מחבר: יוסי בן הרוש בתאריך: 11.06.2021

Angular app with PHP backend: part 3 (PUT)

תוכו למצוין גרסה אנגלית טוביה לא פחות של המדריך בכתבובת [httpClient](#) של Angular כדי לתקשר ב-AJAX עם צד השירות של האפליקציה. מדריך זה הוא השלישי בסדרת המדריכים שממלמדת כיצד להשתמש ב-[httpClient](#) עם צד השירות של האפליקציה.

**service .ts**

בסוף service את המתודה update שקופה את המידע ממקורוונטה, ומשדרת אותו לצד השירות באמצעות המתודה `put` של `httpClient`.

[src/app/car.service.ts](#)

```
update(car: Car) {
  return this.http.put(`${this.baseUrl}/update`, { data: car });
}
```

- המתודה שמשמשת לשידור המידע לצד השירות היא `put` שמקבלת את נתיב ה-`url` בתווך פרמטר ראשון ואת האובייקט `car` שמכיל את המידע שהתקבל מהקומפונטה.
- המידע שמוחזר מצד השירות כולל מידע ב-`header` על הצלחת או כישלון הפעולה. את החלק של השירות נקבע סוף המדריך.

הקוד המלא של ה-`service`:[src/app/car.service.ts](#)

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { map } from 'rxjs/operators';
import { Car } from './car';

@Injectable({
  providedIn: 'root',
})
export class CarService {
  baseUrl = 'http://localhost/api/';

  constructor(private http : HttpClient) {}

  getAll() {
    return this.http.get(`${this.baseUrl}list`).pipe(
      map((res: any) => {
        return res['data'];
      })
    );
  }

  store(car: Car) {
    return this.http.post(`${this.baseUrl}/store`, { data: car }).pipe(
      map((res: any) => {
        return res['data'];
      })
    );
  }

  update(car: Car) {
    return this.http.put(`${this.baseUrl}/update`, { data: car });
  }
}
```

## 2. קוד הקומפוננטה

נוסיף לקומפוננטה את המетодה updateCar שקופה את הערךם שמוצנים לטופס, וגרשמת למתודה update אשר ב-.service

[src/app/app.component.ts](#)

```
updateCar(name: any, price: any, id: any) {
  this.resetAlerts();

  this.carService
    .update({ model: name.value, price: price.value, id: +id })
    .subscribe(
      (res) => {
        this.success = 'Updated successfully';
      },
      (err) => (this.error = err)
    );
}
```

ההרשמה מטבחת באמצעות subscribe בסיסי של updateCar. הינה updateCar().observable

הצפ' הוא לאחת משלטי אפשרויות של מידע שיוחזר מה- observable:

- במקרה של שגיאה, נדוח למשתמש על שגיאה שארעה על ידי אכלאס המשתנה error.
- במקרה של הצלחה, נדוח למשתמש על הצלחת העברת הנתונים באמצעות המשתנה success.

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

import { Car } from './car';
import { CarService } from './car.service';

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

הקוד  
Ts

```

export class AppComponent implements OnInit {
  cars: Car[] = [];
  car: Car = { model: "", price: 0 };

  error = "";
  success = "";

  constructor(private carService: CarService) {}

  ngOnInit() {
    this.getCars();
  }

  getCars(): void {
    this.carService.getAll().subscribe(
      (data: Car[]) => {
        this.cars = data;
        this.success = 'Success in retrieving the list';
      },
      (err) => {
        this.error = err.message;
      }
    );
  }

  addCar(f: NgForm) {
    this.resetAlerts();

    this.carService.store(this.car).subscribe(
      (res: Car) => {
        // Update the list of cars
        this.cars.push(res);

        // Inform the user
        this.success = 'Created successfully';

        // Reset the form
        f.reset();
      },
      (err) => (this.error = err.message)
    );
  }
}

```

```

updateCar(name: any, price: any, id: any) {
  this.resetAlerts();

  this.carService
    .update({ model: name.value, price: price.value, id: +id })
    .subscribe(
      (res) => {
        this.success = 'Updated successfully';
      },
      (err) => (this.error = err)
    );
}

resetAlerts() {
  this.error = "";
  this.success = "";
}

```

### 3. ה-HTML של הטופס

זכרים את הרשימה שמציגה את שם המכונית ומהירה (אם איןכם יודעים بما מדובר אז נא להתחיל במדריך הראשוני בסדרה על [Angular HTTP](#) ו-[HTML](#)), את הרשימה זהו נערך כדי להסביר כיצד ניתן ל轻松ן הcoliות: אפשרות לעדכון המחיר והשם של כל פריט בנפרד. נוסף לכך שלחיצה עליו שולחת את המידע העורף להמשך טיפול בצד הקלאס של הקומפוננטה.

<src/app/app.component.html>

```
<div id="theList">
  <h2>The list</h2>

  <div class="container">
    <div *ngFor="let item of cars;let i = index;" class="list-group-item row">
      <div class="col-4">
        <input type="text"
          [(ngModel)]="cars[i].model"
          class="form-control"
          required
          pattern="^[a-zA-Z ]+$"
          #model="ngModel"
          [ngClass]="{ 'is-invalid': model.touched && model.invalid }">
      </div>
      <div class="col-4">
        <input type="number"
          [(ngModel)]="cars[i].price"
          class="form-control"
          required
          #price="ngModel"
          [ngClass]="{ 'is-invalid': price.touched && price.invalid }">
      </div>
      <div class="col-4">
        <input type="button"
          value="Update"
          class="btn btn-success btn-sm"
          [disabled]="model.invalid || price.invalid"
          (click)="updateCar(model, price, item.id)">
      </div>
    </div>
  </div>
</div>

<div class="col-4">
  <input type="button"
    value="Update"
    class="btn btn-success btn-sm"
    [disabled]="model.invalid || price.invalid"
    (click)="updateCar(model, price, item.id)">
</div>
</div>
</div>
</div>
```

זה קוד ה-PHP שמטפל בעדכון כל אחד מהפריטים בצד השירות.

```

is-
<?php
require 'connect.php';

.loc{
    // Get the posted data.
$postdata = file_get_contents("php://input");

if(isset($postdata) && !empty($postdata))
{
    // Extract the data.
$request = json_decode($postdata);

    // Validate.
if ((int)$request->data->id < 1 || trim($request->data->model) == "" || (int)$request->data->price < 1) {
    return http_response_code(400);
}

    // Sanitize.
$id   = mysqli_real_escape_string($con, (int)$request->data->id);
$model = mysqli_real_escape_string($con, trim($request->data->model));
$price = mysqli_real_escape_string($con, (int)$request->data->price);

    // Update.
$sql = "UPDATE `cars` SET `model`='$model',`price`='$price' WHERE `id` = '{$id}' LIMIT 1";

if(mysqli_query($con, $sql))
{
    http_response_code(204);
}

else
{
    return http_response_code(422);
}
}

```



- במקרה של שגיאה מוחזר קוד ממישפטה ה-400.
- במקרה של הצלחה מוחזר קוד תגובה 204 , שאומר שהפעולה הצלילחה אבל ללא החזרת מידע בגוף התגובה .(payload)

# מדריך 4 : httpClient באנגולר - delete

מחבר: יוסי בן הרוש במתאריך: 11.06.2021

Angular app with PHP backend: part 4 (DELETE)

מדריך זה הוא הרביעי בסדרה של מלמדת כיצד להשתמש ב-**httpClient** של Angular כדי לתקשר עם צד השירות של האפליקציה. במדריך זה, נמחק פרטי מידע באמצעות המethode **delete**.

## 1. ה-service האנגולרי

נוסף ל-service delete שקובלתת את המידע מהקומפוננטה, ומשדרת אותו לצד השירות על גבי המethode delete של `.httpClient`.

[src/app/car.service.ts](#)

```
delete(id: any) {
  const params = new HttpParams()
    .set('id', id.toString());

  return this.http.delete(`${this.baseUrl}/delete`, { params: params });
}
```

- המethode שמשמשת לשידור המידע לצד השירות היא `delete` שמקבלת את נתיב ה-`baseUrl` בתווך פרמטר ראשון ואת הפרמטר `params` אשר מכיל את ה-`id` של הפריט שמיועד למחיקה.
- המידע שמחוזר מצד השירות כולל מידע ב-`header` על הצלחת או כישלון הפעולה.

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpParams } from '@angular/common/http';

import { map } from 'rxjs/operators';
import { Car } from './car';

@Injectable({
  providedIn: 'root',
})
export class CarService {
  baseUrl = 'http://localhost/api/';

  constructor(private http : HttpClient) {}

  getAll() {
    return this.http.get(`${this.baseUrl}list`).pipe(
      map((res: any) => {
        return res['data'];
      })
    );
  }

  store(car: Car) {
    return this.http.post(`${this.baseUrl}/store`, { data: car }).pipe(
      map((res: any) => {
        return res['data'];
      })
    );
  }

  update(car: Car) {
    return this.http.put(`${this.baseUrl}/update`, { data: car });
  }

  delete(id: any) {
    const params = new HttpParams()
      .set('id', id.toString());

    return this.http.delete(`${this.baseUrl}/delete`, { params: params });
  }
}
```

במדריך זה, מוסיף לקומפוננטה את המתודה deleteCar שקולעת את ה-id של הפליט שמעוניינים למחוק, וורשתת למתודה delete שמספק ה-.service

[src/app/app.component.ts](#)

```
deleteCar(id: number) {
  this.resetAlerts();
  this.carService.delete(id).subscribe(
    (res) => {
      this.cars = this.cars.filter(function (item) {
        return item['id'] && +item['id'] !== +id;
      });

      this.success = 'Deleted successfully';
    },
    (err) => (this.error = err)
  );
}
```

- השימוש ב-.subscribe באגל שהמתודה delete היא observable.
- הџי' הוא לאחת מlest' אפשרויות של מידע שייחזיר ה-.observable. שגיאה או הצלחה.

הקוד המלא של הקומפוננטה הרו' הוא לפניכם:

#### 4. צד השרת

זה הקוד ה-PHP שמטפל במחיקת הפליטים בצד השרת:

```
<?php
require 'connect.php';

// Extract, validate and sanitize the id
$id = ($_GET['id'] != null && (int)$_GET['id'] > 0)? mysqli_real_escape_string($con, (int)$_GET['id']): false;

if(!$id)
{
  return http_response_code(400);
}

// Delete.
$sql = "DELETE FROM `cars` WHERE `id` ='{$id}' LIMIT 1";

if(mysqli_query($con, $sql))
{
  http_response_code(204);
}
else
{
  return http_response_code(422);
}
```

```

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

import { Car } from './car';
import { CarService } from './car.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent implements OnInit {
  cars: Car[] = [];
  car: Car = { model: "", price: 0 };

  error = "";
  success = "";

  constructor(private carService: CarService) {}

  ngOnInit() {
    this.getCars();
  }

  resetAlerts() {
    this.error = "";
    this.success = "";
  }

  deleteCar(id: number) {
    this.resetAlerts();
    this.carService.delete(id).subscribe(
      (res) => {
        this.cars = this.cars.filter(function (item) {
          return item['id'] && +item['id'] !== +id;
        });

        this.success = 'Deleted successfully';
      },
      (err) => (this.error = err)
    );
  }
}

```

```

getCars(): void {
  this.carService.getAll().subscribe(
    (data: Car[]) => {
      this.cars = data;
      this.success = 'Success in retrieving the list';
    },
    (err) => {
      this.error = err.message;
    }
  );
}

addCar(f: NgForm) {
  this.resetAlerts();

  this.carService.store(this.car).subscribe(
    (res: Car) => {
      // Update the list of cars
      this.cars.push(res);

      // Inform the user
      this.success = 'Created successfully';

      // Reset the form
      f.reset();
    },
    (err) => (this.error = err.message)
  );
}

updateCar(name: any, price: any, id: any) {
  this.resetAlerts();

  this.carService
    .update({ model: name.value, price: price.value, id: +id })
    .subscribe(
      (res) => {
        this.success = 'Updated successfully';
      },
      (err) => (this.error = err)
    );
}

```

זכרים את הרשימה שמציגה את שם המוכנית ומחירה ומאפשרת עריכה (אם אייכם בטוחים בכך מזכיר מהمدير הראשי בסדרה), את הרשימה הזאת נשנה כדי לאפשר מחיקה של כל פריט בנפרד. לשם כך, נוסיף לטופס כפטור שלחיצה עליו שולחת את המידע על הפריט שיש למחוק לטיפול הקלאס של הקומוננטה.

<src/app/app.component.html>

```
<div id="theList">
  <h2>The list</h2>

  <div class="list-group">
    <div *ngFor="let item of cars;let i = index;" class="list-group-item row">
      <div class="col-md-4">
        <input type="text"
          [(ngModel)]="cars[i].model"
          class="form-control"
          required
          pattern="^[a-zA-Z ]+$"
          #model="ngModel"
          [ngClass]="{ 'is-invalid': model.touched && model.invalid }">
      </div>
      <div class="col-md-4">
        <input type="number"
          [(ngModel)]="cars[i].price"
          class="form-control"
          required
          #price="ngModel"
          [ngClass]="{ 'is-invalid': price.touched && price.invalid }">
      </div>
      <div class="col-md-4">
        <input type="button"
          value="Update"
          class="btn btn-success btn-sm"
          [disabled]="model.invalid || price.invalid"
          (click)="updateCar(model, price, item.id)">
        <input type="button"
          value="Delete"
          class="btn btn-danger btn-sm"
          (click)="deleteCar(item.id? item.id:0)">
      </div>
    </div>
  </div>
</div>
```

- כל פריט של הרשימה מודפס בשורה נפרדת.
- לחיצה על הכפטור מעבירה את המידע אודות ה-`id` של הפריט לקומוננטה להמשך טיפול באמצעות המתודה `deleteCar`.

