

الـ Components (المكونات) 1

- الوحدة الأساسية لبناء التطبيق
- تحتوي على:
 - Class (TS) → المتنطق والبيانات
 - Template (HTML) → العرض
 - Styles (CSS) → التنسيق
- يمكن التواصل بين المكونات عن طريق:
 - استقبال البيانات من المكون الأب `Input@()`
 - إرسال الأحداث للمكون الأب `()Output@()`

الـ Module هو عبارة عن حاوية (Container) تجمع مجموعة من العناصر المرتبطة معاً في التطبيق، مثل:

- المكونات (Components)
- الخدمات (Services)
- التوجيهات (Directives)
- الأنابيب (Pipes)
- والوحدات الأخرى (Modules)

كل Module يمثل وحدة منطقية في التطبيق.
وأشهرها وأهمها هو:

↓ — وهو الموديول الرئيسي للتطبيق.

رابعاً: أقسام الـ NgModule@ *

الوظيفة	القسم
يحتوي على جميع الـ Pipes و Directives و Components التابعة لهذا الموديول	declarations
يحتوي على الموديولات الأخرى التي تحتاجها داخل هذا الموديول	imports
يُسجل الخدمات (Services) التي يمكن حقنها عبر الـ Dependency Injection	providers
يحدد المكون الذي يبدأ منه التطبيق (غالباً AppComponent)	bootstrap
يحدد العناصر التي يمكن مشاركتها مع موديولات أخرى	exports

أولاً: ما هو الـ Service (الخدمة)؟ *

الـ Service في Angular هو كلاس (class) يحتوي على منطق أو وظائف مشتركة يمكن استخدامها في أكثر من مكون.

بدل ما تكرر الكود في كل Component،
تحط المنطق داخل Service واحد وتستدعيه في أي مكان.

@Injectable

خامساً: ملاحظات مهمة

الشرح	النقطة
Angular ينشئ نسخة واحدة فقط من الخدمة (Singleton).	Singleton Service
يمكن تحديد مكان توفر الخدمة (في root أو في موديول محدد).	Scope
يتم تمرير الخدمة في الـ constructor تلقائياً.	Inject in Constructor
مثلاً خدمات Notifications, API, Auth, Logging.	Reusable Logic
المفهوم	الشرح المختصر
كود يُستخدم لتنفيذ منطق مشترك أو الوصول إلى بيانات	Service
آلية Angular لتوفير الخدمات تلقائياً للمكونات	Dependency Injection
تجعل الخدمة متاحة على مستوى التطبيق بالكامل	'providedIn: 'root'
يتم عبر الـ constructor داخل المكون	حقن الخدمة

ثالثاً: أشهر Lifecycle Hooks في Angular

الاستخدام الأساسي	متى يُستدعي	الـ Hook
لمراقبة التغييرات في القيم القادمة من مكون الألب	عند تغيير قيمة <code>@Input()</code>	<code>(ngOnChanges)</code>
لتهيئة البيانات (Initial setup)	بعد إنشاء المكون وقبل عرضه	<code>(ngOnInit)</code>
لمتابعة التغييرات اليدوية	عند كل عملية Change Detection	<code>(ngDoCheck)</code>
يستخدم غالباً مع <code>ng-content</code>	بعد إدخال المحتوى من مكون ألب	<code>(ngAfterContentInit)</code>
فحص إضافي بعد التغييرات	بعد كل فحص للمحتوى	<code>(ngAfterContentChecked)</code>
للوصول إلى عناصر DOM	بعد إنشاء وعرض عناصر الـ HTML للملون	<code>(ngAfterViewInit)</code>
فحص إضافي بعد التغييرات في العرض	بعد فحص الـ View	<code>(ngAfterViewChecked)</code>
لتنظيف الذاكرة أو إلغاء الاشتراكات (unsubscribe)	قبل تدمير المكون	<code>(ngOnDestroy)</code>

خامساً: الفرق بين `(ngOnInit)` و `(constructor)`

	<code>(ngOnInit)</code>	<code>(constructor)</code>	المقارنة
يُستدعي بعد تهيئة القيم و <code>Inputs</code>	يُستدعي أولاً عند إنشاء الكلاس		التوقيت
تحميل بيانات أو استدعاء API	تهيئة بسيطة للكائنات		الوظيفة
الأفضل لتهيئة البيانات	لا تتعامل مع Angular bindings		الأفضلية للاستخدام

يعني:

تستخدم `constructor()` لإنشاء الكائنات، وتستخدم `ngOnInit()` لتحميل البيانات أو استدعاء خدمات (Services).

سابعاً: الترتيب الزمني لأشهر الـ Hooks

Copy code

SCSS

```
constructor()
ngOnChanges()
ngOnInit()
ngDoCheck()
ngAfterContentInit()
ngAfterContentChecked()
ngAfterViewInit()
ngAfterViewChecked()
ngOnDestroy()
```

Data binding:

النوع	الصيغة	الاتجاه	الوصف	مثال	□
Interpolation	<code>{{ variable }}</code>	من Component → Template	عرض قيمة متغير أو نتيجة دالة في القالب	<code><h2>{{ name }}</h2></code>	
Property Binding	<code>"variable"=[property]</code>	من Component → Template	ربط خاصية عنصر HTML أو مكون بقيمة من الكود	<code><"img [src]ImagePath></code>	
Event Binding	<code>()handler=(event)</code>	من Template → Component	تنفيذ دالة عند وقوع حدث في العنصر (...click, input, change)	<code>button<"()click="changeName"><button/></code>	
Two-Way Binding	<code>"variable"=[(ngModel)]</code>	ثنائي (Component ↔ Template)	مزامنة متغير مع عنصر HTML بحيث تتغير البيانات في كلا الاتجاهين	<code><"input [(ngModel)]="username"></code>	

أولاً: ما هي Directives *

Directives هي تعليمات خاصة لـ Angular تتحكم في:

- شكل عناصر الـ DOM
- وجودها أو تكرارها
- سلوكها

💡 باختصار: Directives تغير DOM دون الحاجة لتغيير الـ HTML بدوياً.

⚡ ملاحظات مهمة عن Structural Directives

الشرح	الملاحظة
لأنها تغير البنية (DOM structure)	تبدأ بـ *
الشرطية	ngIf*
التكرار	ngFor*
اختيار عنصر بناء على قيمة	ngSwitch*
لإنشاء واجهات ديناميكية	يمكن دمجها مع Event Binding و Property Binding

✳️ أولاً: ما هي Attribute Directives

Attribute Directives تتحكم في مظهر وسلوك العناصر (CSS) أو خصائص العنصر دون تغيير بنية الـ DOM.

💡 باختصار: تغيير شكل العنصر وليس وجوده.

الميزة	النوع	الوظيفة	ngStyle	ngClass
تعديل الأنماط (styles) مباشرةً ديناميكياً	Attribute Directive	إضافة أو إزالة فئات CSS (classes) ديناميكياً	Attribute Directive	
لا يغير DOM		لا يغير DOM		
الصيغة الأساسية			<pre>property': value,' }]=[ngStyle] {" 'property2.px': value2</pre>	<pre>class1': condition1,' }]=[ngClass] {" 'class2': condition2</pre>
يمكن أن يأخذ	Object كائن	كائن String Object, مصفوفة Array, نص		
مثال عملي 1	<pre>p [ngStyle]="{ 'color': color, 'font-> <p/>": fontSize }"</pre>	<pre>p [ngClass]="{ 'special': isSpecial,> <p/>": hasError }</pre>		
مثال عملي 2 (Array)	<pre><p>()</p> p [ngStyle]="getStyles"> <p>()</p>"<--> CSS classes</pre>	<pre>p [ngClass]=["class1",> <p>()</p>"<--> "class2"]</pre>		
الاستخدام النموذجي	التحكم في الأسلوب مباشرةً بدون CSS	التحكم في الأساليب المعدة مسبقاً في CSS		
أفضل استخدام	عندما تحتاج لتغيير الأنماط مباشرةً ديناميكياً	عندما تريدين إعادة استخدام CSS classes		

أولاً: ما هي Custom Directives

Custom Directives هي توجيهات يقوم المطور بإنشائها لتغيير سلوك أو مظهر عنصر HTML بطريقة مخصصة. يمكن أن تكون:

1. Attribute Directive → لتغيير سلوك أو مظهر عنصر

2. Structural Directive → لتغيير بنية DOM (أقل شيوعاً)

الفكرة: إذا لم يكن directive كافي، نصنع directive خاص بنا. 

ملاحظات مهمة

الشرح	الملاحظة
لتحديد Directive جديد	()Directive@
اسم التوجيه الذي سنستخدمه في HTML	selector
للوصول إلى عنصر DOM الذي يطبق عليه التوجيه	ElementRef
للاستماع للأحداث على العنصر	()HostListener@
لجعل التوجيه مرئياً وقابل للتخصيص	()Input@

خلاصة Custom Directives

- تسمح لك بإنشاء سلوك مخصص للعناصر
- يمكن أن تكون Attribute لتغيير المظهر أو السلوك
- تستخدم ()Input@ و Directive, ElementRef , @HostListener @
- تجعل الكود من وقابلاً لإعادة الاستخدام

النوع	التوجيهات	الوظيفة	أمثلة عملية
Structural Directives	ngIf*	عرض أو إخفاء عنصر بناء على شرط	<p *ngIf="isLoggedIn">مرحبا!
Attribute Directives	ngFor*	تكرار عنصر لكل عنصر في مصفوفة	<li *ngFor="let user of users">{{ user.name }}
Custom Directives	ngswitch*	اختيار عنصر للعرض بناء على قيمة	<div [ngSwitch]="role"><p>*ngSwitchCase="'admin'">Admin</p></div>
Structural Directives	[ngClass]	إضافة أو إزالة CSS classes ديناميكياً	<p [ngClass]="{ 'special': isSpecial }">{{ text }}</p>
Attribute Directives	[ngstyle]	تعديل الأنماط (CSS styles) مباشرةً	<p [ngStyle]="{ 'color': color, 'fontSize.px': fontsize }">{{ text }}</p>
Custom Directives	Directive@	إنشاء سلوك مخصص للعناصر	<p [appHighlight]="'lightgreen'">{{ text }}</p>

ملاحظات مهمة:

- Structural Directives تبدأ بـ * لأنها تغير بنية DOM.
- Attribute Directives لا تبدأ بـ * لأنها تغير مظهر أو سلوك العنصر فقط.
- يمكن أن تكون Custom Directives أو Attribute Directives حسب الحاجة.
- يمكن دمج جميع التوجيهات مع Data Binding لجعل التطبيق ديناميكي بالكامل.

أولاً: ما هي Pipes *

Pipes هي أدوات في Angular تستخدم لتحويل البيانات قبل عرضها في القالب (Template).

- تعمل مباشرة في الـ HTML
- تجعل البيانات أكثر قابلية للقراءة أو التنسيق
- باختصار: تغيير شكل البيانات دون تعديلها في الكود.

(Angular موجودة مسبقاً في Built-in Pipes ◆

يوفر مجموعة كبيرة من Pipes جاهزة للاستخدام: Angular

مثال	الوظيفة	Pipe
today }}	تنسيق التواريخ	date
name }}	تحويل النص إلى أحرف كبيرة	uppercase
name }}	تحويل النص إلى أحرف صغيرة	lowercase
price }}	عرض الأرقام كعملات	currency
ratio }}	تحويل الرقم إلى نسبة مئوية	percent
3.14159 }}	تنسيق الأرقام العشرية	decimal
user }}	عرض الكائنات كJSON	json
p>{{ observableData}}	التعامل مع Promises و Observables	async
text }}	تحويل النص إلى Title Case	titlecase

أولاً: ما هي Custom Pipes *

Custom Pipes تسمح لك بإنشاء طريقة تحويل بيانات مخصصة لا تتوفّر ضمن الـ Built-in Pipes.

- يمكن استخدامها لتحويل النصوص، الأرقام، التواريخ، أو أي نوع بيانات Component
- يجعل الكود أنظف لأن التحويل يحدث في القالب وليس في الـ

ملاحظات مهمة

الشرح	الملاحظة
لتحديد اسم الـ Pipe للاستخدام في القالب	<code>Pipe({ name: 'pipeName' })@</code>
واجهة الأساسية لكل Pipe	<code>PipeTransform</code>
دالة لتحويل البيانات، يمكن أن تأخذ أي عدد من المعاملات	<code>transform(value, ...args)</code>
يجب تسجيل الـ Pipe في AppModule ضمن declarations أو أي آخر	التسجيل
يمكن تمرير معاملات اختيارية لتخصيص التحويل	الاستخدام

خلاصة Custom Pipes

- لإنشاء تحويل بيانات مخصص
- يجعل القالب أنيق وأسهل لفهم
- يمكنها أخذ معاملات لتخصيص السلوك
- تُستخدم في القالب مع `{{ value | pipeName:arg1:arg2 }}`

أولاً: ما هو Routing

يسمح لك Routing :

- التنقل بين صفحات (Components) مختلفة في التطبيق بدون إعادة تحميل الصفحة
- إنشاء تطبيق (SPA)
- تمرير بيانات بين المكونات عبر الروابط

باختصار: Routing يجعل التطبيق ديناميكياً ويوفّر تجربة مستخدم سلسة.

Routes Array و RouterModule

RouterModule .1

- هو Module في Angular مسؤول عن إدارة التوجيهات
- يتم استيراده في Routes array مع AppModule

Routes array .2

- مصفوفة من الكائنات (Objects) تحدد:

• المسار `path`

• المكون الذي سيتم عرضه `component`

⚡ ملاحظات مهمة

الشرح	الملاحظة
تعريف التوجيهات على مستوى التطبيق	<code>RouterModule.forRoot(routes)</code>
المكان الذي يعرض فيه المكون المطابق للمسار	<code><router-outlet></code>
ربط المسار بالروابط في القالب	<code>routerLink</code>
التنقل برمجياً من الكود	<code>router.navigate(['/path'])</code>
لضمان تطابق المسار بالكامل	يمكن إضافة <code>'full'</code> للصفحة الرئيسية

خلاصة Routing💡

- تحدد المسارات والمكونات `Routes array`
- يدير التوجيهات في `RouterModule`
- يعرض المكون الحالي `<router-outlet>`
- للتحكم بالتنقل بين الصفحات `()router.navigate` و `routerLink`

أولاً: ما هي Route Parameters ❁

Route Parameters هي قيم ديناميكية تمرر في رابط المسار (URL)، مثلاً:

Copy code

bash

/user/123

- هنا `123` هو معامل ديناميكي (parameter) يمكن استخدامه في Component لعرض بيانات المستخدم المحدد.

تعريف Routes array في Route Parameter ◆

app-routing.module.ts

Copy code

typescript

```
const routes: Routes = [
  { path: 'user/:id', component: UserComponent }
];
```

- اسم المعامل الديناميكي `:id`
- أي رابط مثل `user/5/` يمرر القيمة `5` للمكون `UserComponent`

⚡ ملاحظات مهمة

الشرح	الملاحظة
تعريف معامل ديناميكي في المسار	paramName:
لقراءة القيمة مرة واحدة عند تحميل المكون	ActivatedRoute.snapshot.paramMap.get('paramName')
لقراءة القيمة ومتابعة أي تغييرات ديناميكية أثناء التنقل	(...)ActivatedRoute.paramMap.subscribe
'path: 'user/:id/:action' مثال:	يمكن استخدام أكثر من معامل

💡 خلاصة Route Parameters

- URL → تمرير قيم ديناميكية في Route Parameters
- ActivatedRoute → لقراءة هذه القيم داخل المكون
- يمكن استخدام subscribe أو snapshot حسب الحاجة
- يمكن دمجها مع Routing & Navigation لعرض صفحات مخصصة لكل قيمة

✳️ أولاً: ما هي Guards

Guards هي خدمات (Services) تتحكم في إمكانية الوصول إلى المسارات أو الخروج منها في Angular.

أنواع Guards الأساسية:

- .1 → قبل الدخول إلى المسار CanActivate
 - .2 → قبل الخروج من المسار CanDeactivate
 - .3 → قبل الدخول إلى مسارات فرعية CanActivateChild
 - .4 → قبل تحميل Module كسيتاريو CanLoad
 - .5 → لتحميل البيانات قبل تفعيل المسار Resolve
- 💡 باختصار: Guards تساعد على حماية المسارات والتحكم بسلوك التنقل.

◆ ملاحظات مهمة

Guard	متى يستخدم	مثال
CanActivate	قبل الدخول إلى المسار	حماية الصفحات الحساسة (Dashboard)
CanDeactivate	قبل الخروج من المسار	التأكد على حفظ البيانات
CanActivateChild	قبل الدخول إلى المسارات الفرعية	حماية المجلدات الفرعية
CanLoad	قبل تحميل Module كسيناريو Lazy Loading غير مصرح بها	منع تحميل Modules منع تحميل
Resolve	لتحميل البيانات قبل تفعيل المسار	تحميل بيانات المستخدم قبل فتح الصفحة

Guards خلاصة💡

- تحكم بالوصول و التنقل بين المسارات → Guards
- منع الدخول → CanActivate
- منع الخروج إذا لم تُحفظ البيانات → CanDeactivate
- سيناريوهات متقدمة لإدارة الوصول و تحميل البيانات → CanActivateChild, CanLoad, Resolve

؟Lazy Loading *

Lazy Loading هو أسلوب في Angular لتأخير تحميل Modules حتى يحتاجها المستخدم، بدلاً من تحميل كل شيء عند بدء التطبيق.

- يقلل حجم Bundle الأولي → تحميل أسرع للتطبيق
 - يستخدم غالباً للمسارات الكبيرة أو صفحات نادرة الاستخدام
- * باختصار: تحميل Module عند الطلب بدل تحميله مسبقاً.

⚠ ملاحظات مهمة

الشرح	الملاحظة
تحميل Module عند الطلب	<code>loadChildren</code>
يستخدم داخل الـ Module الذي يتم تحميله Lazy	<code>()RouterModule.forChild</code>
التطبيق يصبح أسرع عند البداية	يقلل حجم Bundle
مثل <code>canLoad</code> لمنع تحميل Module غير مصرح بها	يمكن دمج مع Guards

💡 خلاصة Lazy Loading

- تحسين الأداء عن طريق تحميل الـ Modules عند الطلب
- يستخدم `()loadChildren + RouterModule.forChild`
- يمكن دمجه مع `Route Parameters` و `Guards`
- مهم جدًا للتطبيقات الكبيرة التي تحتوي على صفحات متعددة

المفهوم	الوظيفة	الصيغة / الاستخدام	مثال عمل
Routes Array	تعريف المسارات والمكونات	<pre>const routes: Routes = [{ path: 'home', component: HomeComponent }]</pre>	صفحة HomeComponent تظهر عند home/
RouterModule	إدارة التوجيهات	<pre>imports: [RouterModule.forRoot(routes)]</pre>	استيراد التوجيهات في AppModule
routerLink	الربط بين الروابط والقوالب	<pre>About</pre>	تنقل المستخدم إلى about/ عند الضغط
router-outlet	مكان عرض المكون المطابق للمسار	<pre><router-outlet></router-outlet></pre>	يعرض HomeComponent أو AboutComponent حسب المسار
Navigation برمجيا	التنقل بين المسارات من الكود	<pre>this.router.navigate(['/about']);</pre>	زر ينقالك لصفحة AboutComponent
Route Parameters	تمرير قيم ديناميكية في URL	<pre>path: 'user/:id', component: { UserComponent }</pre>	ـ يمكن قراءة id=123 بـ user/123/ ActivatedRoute
ActivatedRoute	قراءة Route Parameters داخل Component	<pre>this.route.snapshot.paramMap.get('id')</pre>	عرض المعرف في القالب: {{ userId }}
Guards - CanActivate	منع الدخول إلى مسار غير مسموح	<pre>canActivate: [AuthGuard]</pre>	حماية صفحة Dashboard من المستخدمين غير المسجلين
Guards - CanDeactivate	منع الخروج من مسار إذا كانت هناك تغييرات غير محفوظة	<pre>canDeactivate: [UnsavedGuard]</pre>	عرض رسالة تأكيد عند محاولة مغادرة صفحة Edit
Guards - CanLoad	منع تحميل Module كسيناريو Loading	<pre>canLoad: [AuthGuard]</pre>	منع تحميل AdminModule إذا لم يسجل المستخدم الدخول
Lazy Loading	تحميل Modules عند الحاجة فقط لتحسين الأداء	<pre>loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule)</pre>	تحميل AdminModule فقط عند زيارة admin/

ما هو Dependency Injection (DI) ◆

Dependency Injection يعني "حقن التبعيات"، وهي طريقة في Angular لإرسال الكائنات (مثل الـ Services) إلى الكلاسات أو المكونات بدلاً ما تنشئها بنفسك.

الهدف: ➔

تسهيل إدارة الكود، تقليل التكرار، وتحسين الاختبار .(Testing)

العناصر الأساسية في DI

العنصر	الوصف	مثال
Service	كود يحتوي على منطق مشترك يمكن استخدامه في أكثر من مكون	<code>UserService, AuthService</code>
<code>() Injectable @</code>	تزود Angular بالمعلومات إنها ممكن "تحقق" هذا الـ Service في أماكن أخرى	<code>Injectable({ providedIn: 'root' }) export @ {} class AuthService</code>
Provider	تعرف لـ Service داخل النظام حتى يعرف كيف ينشئه ويوفره Angular	يتم إضافته في providers داخل Module أو Component
'providedIn: 'root'	معناها أن الخدمة متوفرة على مستوى التطبيق كامل (Singleton)	<code>Injectable({ providedIn: 'root' }) @</code>

(Service) ✓

الخدمة هي كلاس يحتوي على منطق مشترك (مثل جلب بيانات، تسجيل الدخول، إلخ) يمكن استخدامه في أكثر من مكون .(Component)

Observables ✓

هي جزء من RxJS تُستخدم لمعالجة البيانات //غير متزامنة (asynchronous) مثل:

- جلب بيانات من API
- انتظار رد السيرفر
- التعامل مع تغييرات الوقت الحقيقي

.Promises بدل الـ HttpClient تستخدم الـ Observable Angular

HttpClient .2 ◆

تُستخدم HttpClient للتعامل مع APIs (الواجهة الخلفية) عبر بروتوكول HTTP.

• 3. الأوامر الأساسية HTTP

نوع الطلب	الوصف	مثال في الكود
GET	جلب بيانات من السيرفر	this.http.get('https://api.com/users')
POST	إرسال بيانات جديدة للسيرفر	this.http.post('https://api.com/users', newUser)
PUT	تعديل بيانات موجودة	this.http.put('https://api.com/users/1', updatedUser)
DELETE	حذف بيانات من السيرفر	this.http.delete('https://api.com/users/1')

ملخص سريع في جدول ◆

مثال	الغرض	العنصر
<code>this.http.get(url)</code>	للتعامل مع REST APIs	HttpClient
<code>()subscribe</code>	لاستقبال البيانات بشكل غير متزامن	Observable
<code>http.get(url)</code>	جلب البيانات	GET
<code>http.post(url, body)</code>	إرسال بيانات جديدة	POST
<code>http.put(url/id, body)</code>	تحديث بيانات	PUT
<code>http.delete(url/id)</code>	حذف بيانات	DELETE

أولاً: ما هي RxJS ◆

RxJS (Reactive Extensions for JavaScript)

هي مكتبة مبنية على مفهوم Stream of data — يعني التعامل مع البيانات "كتيار متدفق" (Reactive Programming) بدلاً من قيم ثابتة.

Angular Routing, Forms, RxJS, HttpClient, State Management داخلي تستخدم

Observable .1 ◆

هي الأساس في RxJS.
تمثل تدفق بيانات يمكن الاشتراك فيه (subscribe).
الـ Observable ما يرسل بيانات إلا لما "نشتري فيه" (subscribe).

Subject .2 ◆

الـ Subject هو نوع خاص من الـ Observable، لكنه يسمح بإرسال البيانات يدوياً للمشتركين.

يُصدر بيانات فقط = Observable
يُصدر ويستقبل بيانات = Subject

الفرق الأساسي:

جميع المشتركين يستقبلون نفس البيانات في نفس اللحظة.

BehaviorSubject .3 ◆

هو نوع خاص من الـ Subject، لكنه يحتفظ بآخر قيمة تم إصدارها.
يعني أي مشترك جديد يدخل لاحقاً، سيحصل على آخر قيمة فوراً.

:BehaviorSubject ميزة ◆

- يحتفظ بآخر قيمة دائماً
- مناسب لتخزين "الحالة" (state) مثل المستخدم الحالي أو اللغة المختارة

◆ مقارنة في جدول

العنصر	يرسل بيانات	يستقبل بيانات	يحتفظ بآخر قيمة	يستخدم عادة في
Observable	✓	✗	✗	استدعاءات HTTP وعمليات async
Subject	✓	✓	✗	التواصل بين المكونات
BehaviorSubject	✓	✓	✓	تخزين الحالة ومشاركة القيم الحالية

ما هي RxJS Operators ◆

ال Operators هم دوال تُستخدم لمعالجة البيانات أثناء مرورها في ال Observable. يتسم كل تعلم عمليات مثل:

- تحويل البيانات (map)
- تصفية البيانات (filter)
- التعامل مع Observables متداخلة (switchMap)
- معالجة الأخطاء (catchError)

أنواع ال Operators ✨

- Observables → لإنشاء Creation Operators
- لتحويل البيانات Transformation Operators
- لتصفية البيانات Filtering Operators
- لمعالجة الأخطاء Error Handling Operators

وإحنا الآن رح نغطي أهم 4 منهم 👉

.)map .1 ◆

الوظيفة: تحويل كل قيمة صادرة من ال Observable إلى قيمة جديدة.
زي map في JavaScript، لكنها تعمل على تيار بيانات)

.)filter .2 ◆

الوظيفة: تمرير القيم التي تحقق شرط معين فقط.
(تبهه Array.filter في JavaScript)

.)switchMap .3 ◆

الوظيفة: تُستخدم للتعامل مع Nested Observables داخل Observable. يتحول كل قيمة صادرة من Observable إلى Observable جديد، وتستبدل القديم بالجديد.
مهم جدًا في HTTP Requests لتجنب تداخل الطلبات. !

(catchError .4)

♦ الوظيفة: لمعالجة الأخطاء الناتجة عن Observable (خاصة في HTTP Requests).

المشغّل (Operator)	الوظيفة	مثال عمل
()map	تحويل القيم	تحويل أرقام إلى أصنافها
()filter	تصفية القيم حسب شرط	إظهار الأرقام الزوجية فقط
()switchMap	استبدال Observable بآخر جديد	تنفيذ طلب HTTP جديد عند تغيير ID
()catchError	معالجة الأخطاء في التيار	إرجاع قيمة بديلة عند فشل الطلب

ما هو ال State Management *

المقصود بـ "الحالة" (State) هو البيانات الحالية للتطبيق، مثل:

- المستخدم الحالي
- سلة المشتريات
- الإعدادات
- العناصر المعروضة في الصفحة

إدارة الحالة = الطريقة اللي بتحافظ فيها على البيانات وتشاركها بين المكونات (Components) بدون تكرار أو فوضى.

1. المشكلة بدون إدارة حالة

في تطبيق بسيط، ممكن تنقل البيانات من Component إلى آخر باستخدام:

- من الأب إلى الابن () → Input@
- من الابن إلى الأب () → Output@

لكن لما يصير المشروع كبير...

ويصير عندك مكونات بعيدة عن بعض، الطريقة هاي بتصير صعبة وغير فعالة.

2. الحل: استخدام Services + Observables

الخدمات (Services) في Angular تكون المكان الأفضل لتخزين البيانات ومشاركتها بين المكونات.

مع استخدام RxJS BehaviorSubject، بنقدر نعمل حالة (state) مشتركة وسهلة التحديث.

ملخص في جدول

الطريقة	الاتجاه	المزايا	العيوب
()Input@	من الأب إلى الابن	بسيط وسريع	لا يعمل بين مكونات بعيدة
()Output@	من الابن إلى الأب	تواصل مباشر	لا يمكن بين مكونات غير مرتبطة
Service + BehaviorSubject	في كل الاتجاهات	مشاركة عالمية وسهلة	يحتاج RxJS Observable ومعرفة بـ RxJS

متى تستخدم كل طريقة؟

الحالات	الحل المناسب
تمرير بيانات من أب إلى ابن	()Input@
تمرير بيانات من ابن إلى أب	()Output@
مشاركة بيانات بين مكونات غير مرتبطة	Service + BehaviorSubject
تطبيق كبير بعدة صفحات	(NgRx / Signal Store) State Management

ما هو Web Storage؟

عبارة عن آلية في المتصفح لتخزين البيانات بشكل محلي على جهاز المستخدم، تُستخدم لحفظ معلومات مثل:

- بيانات تسجيل الدخول
- إعدادات المستخدم
- عناصر سلة المشتريات
- التوكن (JWT Token)

أنواع Web Storage

النوع	المدة	تحذف متى؟	السعة التقريبية
Local Storage	دائمة	فقط عند مسح بيانات المتصفح	MB 10-5
Session Storage	مؤقتة	عند إغلاق التبويب أو المتصفح	MB 5

ملخص المقارنة

Session Storage	Local Storage	العنصر
مؤقتة (حتى إغلاق التبويب)	دائمة	مدة التخزين
حوالى 5MB	حوالى 10MB-5	السعة
لكل تبويب	لكل موقع	النطاق
بيانات الجلسة المؤقتة	حفظ الإعدادات أو التوكن	الاستخدام الشائع

ما هو الـ JWT؟

JSON Web Token (JWT) هو نوع من الرموز (tokens) يُستخدم للتحقق من هوية المستخدم (Authentication). يُصدره السيرفر بعد ما المستخدم يسجل الدخول، وبعدها يٌتخزن عادة في Session Storage أو Local Storage داخل المتصفح.

- يكون الـ JWT من 3 أجزاء:

Copy code

css

```
xxxxx.yyyyy.zzzzz  
Header.Payload.Signature
```

- : نوع التشفير (مثلاً HS256) **Header**
- : يحتوي بيانات المستخدم (id, email, role) **Payload**
- : توقيع التحقق من صحة الرمز **Signature**

كيف يعمل نظام الـ Authentication في Angular

- المستخدم يدخل بياناته (email + password) 1
- الـ Angular frontend يرسلهم إلى API (باستخدام HttpClient) 2
- السيرفر يتتحقق ويرجع **JWT Token** 3
- التطبيق يخزن التوكن في **Local Storage** 4
- عند أي طلب لاحق (GET / POST / DELETE...) يتم إرسال التوكن في **Header** للتحقق من الصلاحية 5

ملخص في جدول

المكان	الوصف	العملية
<code>()AuthService.login</code>	يرسل البريد وكلمة المرور للسيرفر	تسجيل الدخول
<code>()subscribe</code>	السيرفر يرجع JWT	استلام التوكن
<code>()saveToken</code>	في Local Storage	تخزين التوكن
<code>AuthInterceptor</code>	تقاضياً عبر Interceptor	إضافة التوكن للطلبات
<code>()logout</code>	حذف التوكن	تسجيل الخروج

نصائح أمنية

- لا تخزن معلومات حساسة داخل الـ token نفسه
- استخدم HTTPS دائمًا
- امسح التوكن عند تسجيل الخروج
- استخدم Guards (CanActivate) لحماية الصفحات الخاصة

Angular 18

(المكونات المستقلة) Standalone Components

ما هي؟

.NgModules هي مكونات يمكن استخدامها بدون الحاجة إلى

بساطة، بدل ما تسجّل المكون داخل AppModule ، يتعلّم عليه مباشرةً إنه .standalone . هذا يجعل الكود أبسط وأسرع في التحميل، خصوصاً مع التطبيقات الحديثة.

الفكرة الأساسية

في السابق، أي مكون لازم تسجّله في declarations داخل NgModule@

الفوائد

الفائدة	التوضيح
تقليل التعقيد	ما في حاجة لإنشاء Modules لكل جزء
أسرع في التحميل	يدعم إلـ tree-shaking (يحذف الكود غير المستخدم)
أسهل في المشاركة	يمكن استيراده مباشرة في أي مكان
تبسيط الهيكلاية	الكود يصبح أوضح وأسهل لفهم

Angular 18 – نظام إدارة الحالة الجديد في Signals

ما هي Signals

Signals هي طريقة جديدة في Angular لإدارة وتحديث البيانات (state) داخل المكونات بدل الاعتماد على RxJS أو Change Detection التقليدي.

كلمات بسيطة:

هي متغيرات "تنقايق تلقائياً" مع أي تغيير بالقيمة، وينتخد إلـ `EventEmitter` أو `Input@()` أو `.ngOnChanges` أو `Component` يستخدم هذا إلـ Signal.

الفكرة الأساسية

عندك `Signal` = متغير قابل للمراقبة
أي `Signal` يستخدم هذا إلـ Component، بيتخذ تلقائياً لما تتغير قيمته

دوال مهمة في Signals

المثال	الوظيفة	الدالة
<code>count = signal(0)</code>	إنشاء Signal جديد	<code>signal(initialValue)</code>
<code>count.set(5)</code>	تغيير القيمة مباشرة	<code>set(value)</code>
<code>count.update(v => v + 1)</code>	تحديث القيمة بناءً على القيمة السابقة	<code>update(fn)</code>
<code>double = computed(() => count() * 2)</code>	إنشاء Signal تعتمد على أخرى	<code>computed(fn)</code>
<code>effect(() => console.log(count()))</code>	تنفيذ كود عند تغيير Signal	<code>effect(fn)</code>

RxJS مزايا Signals مقارنة بـ 🔥

RxJS	Signals	المقارنة
Observable/subscribe ✕ يحتاج	أبسط بكثير ✓	سهولة الاستخدام
يعتمد على كشف التغييرات العام ✕	يحدث تغييرات دقيقة فقط ✓	أداء أفضل
Subscription يحتاج	مباشر وسهل	تعامل مع الواجهة
أعلى	منخفض	تعقيد

الخلاصة 📊

- Angular Signals = طريقة حديثة وسريعة لإدارة الحالة في Angular
- تستخدم بدل RxJS في الحالات البسيطة والمتوسطة
- يمكن دمجها مع RxJS في المشاريع الكبيرة

جدول المقارنة 📈

الميزة الجديدة	الصيغة الجديدة	الصيغة القديمة	الميزة
ng-template وبدون	<code>} if (cond) { ... } @else@ { ...</code>	<code>"ngIf="cond"; else block"</code>	If/Else
empty@ ويدعم	<code>for (x of items; track@ x.id)</code>	<code>ngFor="let x of items;" "trackBy: fn</code>	Loop
أوضح وأقصر	<code>switch (val) { @case() ...@ }</code>	<code><"div [ngSwitch]="val></code>	Switch

الخلاصة 🎯

- switch@ , if, @for @ = جيل جديد من بناء الجمل في القوالب
- تجعل الكود أنظف وأسهل في القراءة
- أسرع في الأداء
- لا تحتاج * أو ng-template *

Change Detection Strategy (OnPush) .1 ⚡

ما هو Change Detection ◆

- يقوم بشكل دوري بفحص التغييرات في البيانات داخل المكونات (Components) لتحديث واجهة المستخدم.
- الوضع الافتراضي: Default → كل تغيير في أي مكان في التطبيق يؤدي لفحص كل المكونات (قد يكون بطء في التطبيقات الكبيرة).

OnPush ◆

- استراتيجية OnPush تقلل عمليات الفحص غير الضرورية.
- يقوم بتحديث المكون فقط إذا:
 - تم تمرير Input جديد للمكون.
 - حدث داخلي في المكون (... , click, input).
 - تغير Observable مستخدم داخل المكون.

Lazy Loading Modules .2 ⚡

ما هو Lazy Loading ◆

- تحميل الوحدات (Modules) فقط عند الحاجة لها، بدلاً من تحميل كل شيء عند بدء التطبيق.
- يقلل حجم الـ Bundle ويزيد سرعة تحميل الصفحة الرئيسية.

ngFor في TrackBy .3* ⚡

ما المشكلة؟ ◆

- بدون trackBy ، Angular يعيد رسم جميع العناصر عند أي تغيير في القائمة → أداء ضعيف للقوائم الكبيرة.

الحل: استخدام trackBy لتحديد مفتاح فريد لكل عنصر:

ملخص Performance Optimization 📈

المزايا	الهدف	التقنية
أداء أفضل، تحديث فقط عند تغيير البيانات	تقليل عمليات إعادة الرسم غير الضرورية	ChangeDetectionStrategy.OnPush
تقليل حجم الـ bundle وسرعة تحميل الصفحة	تحميل وحدات التطبيق عند الحاجة فقط	Lazy Loading Modules
أداء أفضل للقوائم الكبيرة	تبين العناصر لتحديث فقط المتغير	ngFor في TrackBy*

جدول مقارنة بين Reactive Forms و Template-driven

Reactive Forms	Template-driven	الخاصة
في الـ Component (TypeScript)	في الـ HTML	تعريف الـ Form
عالية	منخفضة	المرونة
كامل ومرن	محدود	التحكم بالـ validation
مشاريع كبيرة ومعقدة	مشاريع صغيرة	المناسب

Developer Skills

Angular CLI (Command Line Interface)

ما هو ⚡

- أداة سطر أوامر لتسهيل إنشاء مشاريع Angular وإدارتها.
- توفر أوامر جاهزة لإنشاء:

مشروع جديد •

مكونات (Components) •

خدمات (Services) •

واجهات (Interfaces) •

وحدات (Modules) •

Routing, Guards, Pipes, Directives •

أهم أوامر Angular CLI ◆

الأمر	الوصف	مثال
<code>ng new <project-name></code>	إنشاء مشروع Angular جديد	<code>ng new my-app</code>
<code>ng serve</code>	تشغيل المشروع على السيرفر المحلي	<code>ng serve --open</code>
<code>ng g c <name></code> أو <code><ng generate component <name><><name></code>	إنشاء Component جديد	<code>ng g c user</code>
<code>ng g s <name></code> أو <code><ng generate service <name><><name></code>	إنشاء Service جديد	<code>ng g s auth</code>
<code>ng g m <name></code> أو <code><ng generate module <name><><name></code>	إنشاء Module جديد	<code>ng g m admin</code>
<code><ng generate directive <name></code>	إنشاء Directive جديد	<code>ng g d highlight</code>
<code><ng generate pipe <name></code>	إنشاء Pipe جديد	<code>ng g p uppercase</code>
<code><ng generate guard <name></code>	إنشاء Guard جديد	<code>ng g guard auth</code>
<code>ng build --prod</code>	بناء المشروع للإنتاج	
<code>ng test</code>	تشغيل اختبارات الوحدة (Unit Tests)	
<code>ng lint</code>	فحص الكود وتحسين الجودة	
<code><ng add <package></code>	إضافة مكتبة خارجية	<code>ng add @angular/material</code>
<code>ng update</code>	تحديث إلى نسخة أحدث	<code>ng update @angular/core @angular/cli</code>



Chrome DevTools .1 ⚡

ما هي؟ ◆

- أداة مدمجة في متصفح Chrome لفحص التطبيقات.
- تساعدك على:

مشاهدة DOM وال Components ◆

- فحص Network requests
- مراقبة logs للأخطاء وال Console
- فحص Memory و Performance
- اختبار Responsive design

أهم تبويبات DevTools ◆

الاستخدام	التبوب
فحص وتعديل CSS و DOM مباشرة	Elements
مشاهدة الأخطاء وكتابة أوامر JavaScript مباشرة	Console
تتبع طلبات HTTP وفحص الاستجابات	Network
وضع نقاط توقف (Breakpoints) وتصحيح الكود	Sources
قياس سرعة التطبيق وأداءه	Performance
فحص Local Storage, Session Storage, Cookies	Application
تحليل استهلاك الذاكرة واكتشاف التسريريات	Memory

نصائح استخدام Chrome DevTools في Angular ◆

- استخدم `console.log()` أو `console.table()` لفحص البيانات.
- استخدم `Breakpoints` داخل `sources` لتنبيه تنفيذ الـ TypeScript.
- تحقق من الـ `Network` لتتبع استدعاءات `HttpClient`.
- استخدم `Local Storage & Session Storage` من تبوب Application لفحص البيانات المخزنة.

Augury .2 ⚡

ما هي؟ ◆

.Angular Debugging خاصة بـ Augury •

متاحة كإضافية لمتصفح Chrome / Edge •

تساعد على: •

فحص Component Tree للمكونات (Hierarchy) •

رؤية Inputs/Outputs و State Properties لكل مكون •

فحص Services و Routes •

Observables و Change Detection تتبع •

استخدام Augury ◆

1. تثبيت الإضافة من Chrome Web Store •

2. فتح DevTools → تجد تبويب Augury. •

3. ستظهر كل المكونات ومقدار Change Detection cycles لكل مكون. •

4. يمكن تعديل القيم مباشرة وتجربة النتائج في الوقت الحقيقي. •

ملخص مقارنة بين Augury و DevTools 📈

المزايا	الاستخدام الأساسي	الأداة
متاحة دائمة، قوية لجميع المواقع	فحص الكود، Network Performance	Chrome DevTools
متخصص بـ Angular، يعرض هيكل التطبيق بالكامل	فحص مكونات، Angular: Inputs/Outputs، Services	Augury

نصائح عامة للتصحيح في Angular ◆

1. استخدم console.log() بحكمة لتبين المتغيرات.

2. استغل Step Over / Step Into و Breakpoints في DevTools.

3. تحقق دائمًا من Subscriptions و Observables عند استخدام RxJS.

4. افحص OnPush strategy عند مشاكل الأداء باستخدام Augury و Change Detection.

1.

1. ما هو Unit Testing 🔥

- اختبار أصغر جزء من التطبيق (Function, Component, Service) للتأكد من عمله بشكل صحيح.
- الهدف: التأكد من أن كل وحدة تعمل بمفردها قبل دمجها مع باقي التطبيق.
- يأتي مهياً مسبقاً لاختبار الوحدة باستخدام Jasmine → Framework.
- لكتابية الاختبارات على المتصفح Karma → Test Runner.

شرح الكود 🔘

الوظيفة	الجزء
(suite) مجموعة من الاختبارات لنفس الوحدة	()describe
تنفيذ إعدادات قبل كل اختبار	()beforeEach
اختبار محدد (test case)	()it
التحقق من القيم (assertion)	()expect
تهيئة البيئة لإنشاء المكون	()TestBed.configureTestingModule

5. نصائح لكتابة Unit Tests 🔥

1. اختبر كل وظيفة مستقلة قبل دمجها.
2. استخدم `spyOn` لتتبع الاستدعاءات (خصوصاً مع `Services`).
3. لا تختبر إطار Angular نفسه، فقط تطبيقك.
4. ركز على `Component logic` و `Pipes` و `Services` و `Directives`.
5. اجعل كل `test case` قصيرة وواضحة.

Folder Structure .1 (هيكلية المجلدات)

الهدف:

- تنظيم المشروع بطريقة واضحة وقابلة للتوسيع
- تسهيل صيانة الكود وإيجاد الملفات بسرعة

مثال هيكلية Angular متقدمة:

Copy code 

ruby

```
src/
  └── app/
    |   ├── core/           # Services,  مكونات
    |   |   ├── services/
    |   |   ├── guards/
    |   |   └── interceptors/
    |   ├── shared/         # مشتركة
    |   |   ├── components/
    |   |   ├── directives/
    |   |   └── pipes/
    |   ├── features/       # Modules
    |   |   ├── auth/
    |   |   ├── dashboard/
    |   |   └── user/
    |   └── app-routing.module.ts
    └── app.component.ts
  └── assets/
  └── environments/
└── main.ts
```

مزايا هذا النظام:

- فصل الخدمات العامة عن المكونات الخاصة
- تسهيل إعادة استخدام الكود
- تبسيط إضافة ميزات جديدة (Feature Modules)

Naming Conventions .2 🔥 (معايير التسمية)

مكونات Angular 🔶

النوع	الصيغة الموصى بها	مثال
Component	camelCase + .component.ts	user-profile.component.ts
Service	camelCase + .service.ts	auth.service.ts
Module	PascalCase + .module.ts	UserModule → user.module.ts
Directive	camelCase + .directive.ts	highlight.directive.ts
Pipe	camelCase + .pipe.ts	date-format.pipe.ts
Guard	camelCase + .guard.ts	auth.guard.ts

مكونات Template & Class 🔶

- أسماء المكونات يجب أن تكون واصفة (LoginComponent , DashboardComponent)
- المتغيرات والـ TypeScript: camelCase في functions
- الثوابت: UPPER_CASE

٣.٤ (إعادة الاستخدام وقابلية التوسيع) Reusability and Scalability

إعادة الاستخدام:

- ضع المكونات المشتركة في `shared/components`
- ضع `shared/directives` و `shared/pipes` عامة في `Directives` و `Pipes`
- استخدم `Services` بدل تكرار الكود في المكونات

قابلية التوسيع:

- استخدم `Feature Modules` لكل ميزة (AuthModule, UserModule)
- استخدم `Lazy Loading` لتحسين الأداء عند التوسيع
- حافظ على `Separation of Concerns` → فصل الـ UI عن المنطق والخدمات

مثال:

- مكون `ButtonComponent` يمكن استخدامه في أي صفحة:

[Copy code](#)

html

```
<app-button label="Save" (click)="save()"></app-button>
```

Best Practices ملخص

النقطة	التوضيح
Folder Structure	core, shared, features, assets → تنظيم واضح
Naming Conventions	... Component → <code>.component.ts</code> , Service → <code>.service.ts</code>
Reusability	استخدم shared modules, services, components
Scalability	Feature modules, lazy loading, separation of concerns

أسئلة شائعة

♦ الفرق الأساسي في جدول

الخاصية	()Output@	()Input@
الاتجاه	من الابن إلى الأب	من الأب إلى الابن
نوع البيانات	أحداث (<code>EventEmitter</code>)	قيم أو <code>objects</code>
الهدف	التواصل أو إرسال إشارات للأب	مشاركة البيانات

Behavior	Subject	الخاصية
يوجد (initial value) <input checked="" type="checkbox"/>	لا يوجد <input type="checkbox"/>	قيمة ابتدائية
نعم، يحصل على آخر قيمة <input checked="" type="checkbox"/>	لا <input type="checkbox"/>	المشتركون الجدد يحصلون على القيم السابقة؟
حالة الحالة (state) أو آخر قيمة لـ stream	إرسال بيانات جديدة فقط	الاستخدام الشائع

Agile / Scrum Teamwork .1 ⚡

ما هو؟ ◆

- = طريقة عمل مرنة لتطوير البرمجيات خطوة خطوة
- = إطار عمل Agile يستخدم Sprints قصيرة (عادة أسبوعين) لتسلیم الميزات Scrum

أهم النقاط: ◆

- . المشاركة اليومية في Daily Standups → مشاركة ما تم عمله وما سيُعمل وما المشاكل.
- . احترام Roles: → تحديد الأولويات Product Owner •
→ إزالة العوائق Scrum Master •
→ تنفيذ المهام Development Team •
- . استخدام أدوات إدارة المشاريع مثل Jira, Trello, Azure DevOps
- . العمل بشكل تعاوني ومراجعة الكود مع الزملاء (Code Reviews).

2. التعامل مع العملاء (Client Communication) ⚡

نصائح مهمة: ◆

- . الاستماع الجيد لفهم متطلبات العميل بدقة.
- . توضیح ما يمكن وما لا يمكن تنفيذه بشكل صريح.
- . تقديم تحديقات دورية حول تقدم المشروع.
- . استخدام لغة واضحة وبسيطة عند شرح التقنية.
- . التعامل مع الملاحظات بشكل إيجابي ومرن.

Agile / Scrum Teamwork .1 🔥

ما هو؟ ◆

- = طريقة عمل مرنة لتطوير البرمجيات خطوة خطوة
- = إطار عمل Agile يستخدم Sprints قصيرة (عادة أسبوعين) لتسلیم المیزات

أهم النقاط: ◆

- . المشاركة اليومية في Daily Standups → مشاركة ما تم عمله وما سيُعمل وما المشاكل.
- . احترام Roles:
 - Product Owner
 - إزالة العوائق
 - تنفيذ المهام
- . استخدام أدوات إدارة المشاريع مثل Jira, Trello, Azure DevOps
- . العمل بشكل تعاوني ومراجعة الكود مع الزملاء (Code Reviews)

2. التعامل مع العملاء (Client Communication) 🔥

نصائح مهمة: ◆

- . الاستماع الجيد لفهم متطلبات العميل بدقة.
- . توضیح ما يمكن وما لا يمكن تنفیذه بشكل صريح.
- . تقديم تحديات دورية حول تقدم المشروع.
- . استخدام لغة واضحة وبسيطة عند شرح التقنية.
- . التعامل مع الملاحظات بشكل إيجابي ومرن.

◆ الفرق بين Agile و Waterfall

Waterfall	Agile	الخاصية
خطية ومتسلسلة	تدريجية وتكرارية	طريقة التنفيذ
صعب ومعقد	سهل ومرن	التعامل مع التغيير
في البداية والنهاية فقط	مستمرة	مشاركة العميل
بطيئة، بعد اكتمال المشروع	سريعة	سرعة التسلیم

ما فائدة الـ `async pipe`؟ 1

- يشتغل مع Observables و Promises لتحديث الواجهة تلقائياً عند وصول القيم الجديدة.
- لا يحتاج `unsubscribe` → يقلل مشاكل الذاكرة.

كيف تحسن أداء التطبيق؟ 2

- استخدام `ChangeDetectionStrategy.OnPush`
- (Modules) Lazy Loading
- استخدام `*ngFor` في `trackBy`
- تجنب العمليات الثقيلة داخل القوالب

ما الفرق بين `constructor` و `ngOnInit`؟ 3

الخاصية	constructor	ngOnInit
التوقيت	عند إنشاء الكائن	بعد إنشاء المكون وتهيئة Inputs
الهدف	إعداد المتغيرات الأساسية	تنفيذ الكود الذي يحتاج <code>Inputs</code> أو بيانات من الألب
استدعاء	تلقائي من Angular	تلقائي من <code>constructor</code> بعد <code>Angular</code>

ما هو مفهوم `Angular Signals` في 18؟ 4

- متغيرات تتفاعل تلقائياً عند تغيير قيمتها.
- تستخدم لإدارة الحالة (state) بدون الحاجة إلى `Input()`, `@Output()` أو `RxJS`.
- كل تغيير في Signal يحدث تحديث مباشر للواجهة.

كيف تعمل `Routing Guards`؟ 5

- تنمنع أو تسمح بالوصول إلى Route معين بناءً على شروط محددة.
- أنواع شائعة:
 - لمنع الوصول قبل الدخول للصفحة `CanActivate`
 - لمنع الخروج إذا لم يتم حفظ البيانات `CanDeactivate`
 - لمنع تحميل Module كامل `CanLoad`
 - ↓ `Observable/Promise` أو `true/false` ترجع عادةً `true/false`

ما الفرق بين Template-driven Forms و Reactive Forms ٦

Reactive Forms	Template-driven	الخاصة
في الـ Component باستخدام FormGroup/FormControl	في الـ HTML باستخدام ngModel	تعريف النموذج
عالية	منخفضة	المرونة
كامل ومرن	بسيط	التعامل مع validation
مشاريع كبيرة ومعقدة	مشاريع صغيرة	مناسب

الإجابة المختصرة	السؤال
يربط Observables/Promises بالواجهة مباشرة، ويحدث التحديث تلقائياً بدون الحاجة لـ <code>.unsubscribe</code> .	ما فائدة الـ <code>async pipe</code> ؟
استخدم <code>OnPush</code> , <code>Lazy Loading</code> , <code>trackBy</code> , <code>ngFor*</code> ، وتجنب العمليات الثقيلة في القوالب.	كيف تحسن أداء التطبيق؟
لإنشاء الكائن وإعداد المتغيرات، <code>ngOnInit</code> للتنفيذ بعد تهيئة <code>constructor</code> .	ما الفرق بين <code>ngOnInit</code> و <code>constructor</code> ؟
متغيرات تتفاعل تلقائياً عند تغيير قيمتها، لتحديث الواجهة مباشرة بدون <code>RxJS</code> أو <code>.Input/@Output@</code> .	ما هو مفهوم <code>Signals</code> في Angular 18؟
تنع أو تسمح بالوصول إلى <code>Route</code> حسب شروط محددة (<code>CanActivate()</code> , <code>(CanDeactivate, CanLoad</code>)	كيف تعمل <code>Routing Guards</code> ؟
Reactive Forms: <code>Template-driven</code> : HTML + <code>ngModel</code> بسيط وصغير؛ <code>TypeScript + FormGroup/FormControl</code> مرن وكبير.	ما الفرق بين <code>Template-driven</code> و <code>Reactive Forms</code> ؟
تمرير بيانات من الألب إلى الابن، <code>@Input()</code> = إرسال أحداث من الابن للألب.	ما الفرق بين <code>@Input()</code> و <code>@Output()</code> ؟
Subject: لا يحمل قيمة ابتدائية، المشترك الجديد لا يحصل على القيم السابقة. BehaviorSubject: يحمل قيمة ابتدائية، المشترك الجديد يحصل على آخر قيمة مباشرة.	ما الفرق بين <code>Subject</code> و <code>BehaviorSubject</code> ؟
ReplaySubject: يخزن <code>n</code> قيمة ويرسلها للمشترك الجديد. AsyncSubject: يرسل فقط آخر قيمة عند اكتمال الـ <code>stream</code> .	ما الفرق بين <code>ReplaySubject</code> و <code>AsyncSubject</code> ؟
<code>ngIf</code> = directive عادي، <code>ngIf* = shorthand</code> لإنشاء <code>ng-template</code> تلقائياً.	ما الفرق بين <code>ngIf</code> و <code>ngIf*</code> ؟
الطريقة التقليدية، <code>@for</code> = <code>Angular 18+ syntax</code> <code>for</code> = <code>Control Flow</code> ، <code>track</code> و <code>empty</code> .	ما الفرق بين <code>ngFor*</code> و <code>ngFor</code> ؟
CSS classes ديناميكياً، <code>ngStyle</code> لتبديل <code>CSS styles</code> ديناميكياً.	ما هو الفرق بين <code>ngClass</code> و <code>ngStyle</code> ؟
Lazy Loading: تحميل الوحدة عند الحاجة فقط، Eager Loading: تحميل كل شيء عند بدء التطبيق.	ما الفرق بين <code>Eager Loading</code> و <code>Lazy Loading</code> ؟

ما الفرق بين ReplaySubject و BehaviorSubject؟	ReplaySubject = آخر قيمة فقط، BehaviorSubject = يخزن عدة قيم ويرسلها للمشترك الجديد.
ما الفرق بين OnPush و Default Change Detection؟	Default: يفحص كل المكونات عند أي تغيير. OnPush: يفحص فقط عند تغيير Observable أو حدث داخلي Input.
كيف يمكن تحسين الأداء مع *ngFor؟	استخدام trackBy لتحديد مفتاح فريد لكل عنصر للتحديث العنصر المتغير فقط.
كيف تتعامل مع API باستخدام HttpClient؟	استخدام pipe + Observables مع GET, POST, PUT, DELETE. يمكن استخدام catchError لمعالجة خطأ.
ما الفرق بين Observables و Promises؟	Promise: قيمة واحدة، لا يمكن إلغاؤها، تنفيذ مرة واحدة. Observable: قيم متعددة، يمكن إلغاؤه، يمكن الاشتراك مرات متعددة.
كيف تعامل Event Binding؟	النقر على المكون "handler"=(event) لتنفيذ دالة في المكون.
كيف تعامل Property Binding؟	.value=[property] لتمرير قيمة من الـ Component إلى DOM element.
كيف تعامل Two-way Binding؟	value=[[ngModel]] يربط بين Event و Property تلقائيا.
ما الفرق بين Attribute Directives و Structural؟	ngClass, ngStyle تغير هيكل DOM. Attribute (ngIf, *ngFor) تغير سلوك أو شكل العناصر فقط.
ما الفرق بين Custom Pipes و Built-in Pipes؟	Built-in: date, uppercase, async, Custom حسب الحاجة.
ما الفرق بين Routes array و RouterModule؟	RouterModule = استيراد نظام التوجيه، Routes = مصفوفة تحدد المسارات وعلاقتها بالمكونات.
ما هو الفرق بين CanActivate و CanDeactivate؟	route. CanActivate = منع الدخول للـ route. CanDeactivate = منع الخروج من route إذا لم تُحفظ البيانات.