Angular

Day 5

ITI – Assiut Branch
Eng. Hany Saad

Observables

More details:

- http://reactivex.io
- http://reactivex.io/rxjs

Observables (Cont.)

Observable	Promise
1.It Emits multiple value over a period of time	1.Emit only single value at a time
2.Lazy.Observable is not called untile we subscribe to the Observable	2.Not Lazy.It call the services with out .then and .catch
3.Can be cancelled by using the unscubscribe() method	3.Not possible to cancelled
4.Observable provides the map ,forEach, filter,reduce,retry,retryWhen operators	4.It not provides any operators

Observables (Cont.)

Defining observers

A handler for receiving observable notifications implements the `Observer` interface. It is an object that defines callback methods to handle the three types of notifications that an observable can send:

NOTIFICATION TYPE	DESCRIPTION
`next`	Required. A handler for each delivered value. Called zero or more times after execution starts.
`error`	Optional. A handler for an error notification. An error halts execution of the observable instance.
`complete`	Optional. A handler for the execution-complete notification. Delayed values can continue to be delivered to the next handler after execution is complete.

An observer object can define any combination of these handlers. If you don't supply a handler for a notification type, the observer ignores notifications of that type.

Observables - create


```
getScheduledAds(intervalInSeconds: number):Observable<string>
   {
      return new Observable <string>((observer)=>{
        // observer.next();
        // observer.error();
        // observer.complete();
        let counter=0;
        let adsTimer= setInterval(()=>{
          console.log('in Interval');
          if (counter==this.adsList.length)
          {
            observer.complete();
          }
          if(this.adsList[counter] =="")
          {
            observer.error("Error: Empty Ad.");// Will stop Observable
          }
          observer.next(this.adsList[counter]);
          counter++;
        },intervalInSeconds*1000);

        return {
          unsubscribe(){
            //Will be called:// 1- Error 2- Complete 3- unsubscribe()
            clearInterval(adsTimer);
            console.log("In Obs Unsubscribe...")
          }
        }
      });
   }
```

More details: https://reactivex.io/documentation/observable.html

Observables - Subscribe


```
  let observer={
      next:(data:string)=>{
        console.log(data);
      },
      error: (err:string)=>{
        console.log(err);
      },
      complete:()=>{
        console.log("Ads Finsihed!")
      }
    };
    let adsSubscription= this.promoAds.getScheduledAds(3).subscribe(observer);
    this.subscriptions.push(adsSubscription);
    // OR //
 let sub=this.promoAds.getSerialAds().subscribe(ad=>{
      console.log(ad);
    });
    this.subscriptions.push(sub);
```

```
ngOnDestroy(): void {
    // this.subscription.unsubscribe();
    for (let subscription of this.subscriptions) {
      subscription.unsubscribe();
    }
  }
```

More details: http://reactivex.io/documentation/operators/subscribe.html

Observables - Unsubscribe

Unsubscribing from an Observable

We need to unsubscribe to close the observable when we no longer require it. If not it may lead to memory leak & Performance degradation.

To Unsubscribe from an observable, we need to call the `Unsubscribe()` method on the subscription. It will clean up all listeners and frees up the memory.

To do that, first, create a variable to store the subscription

```
1
2 obs: Subscription;
3
```

Assign the subscription to the `obs` variable

```
1
2
3 this.obs = this.src.subscribe(value => {
4   console.log("Received " + this.id);
5 });
6
7
```

Call the unsubscribe() method in the ngOnDestroy method.

```
1
2 ngOnDestroy() {
3   this.obs.unsubscribe();
4 }
5
```

Observables (Cont.)

Full Example:

Observe geolocation updates

```
// Create an Observable that will start listening to geolocation updates
// when a consumer subscribes.
const locations = new Observable((observer) => {
  let watchId: number;

  // Simple geolocation API check provides values to publish
  if ('geolocation' in navigator) {
    watchId = navigator.geolocation.watchPosition((position: GeolocationPosition) => {
      observer.next(position);
    }, (error: GeolocationPositionError) => {
      observer.error(error);
    });
  } else {
    observer.error('Geolocation not available');
  }

  // When the consumer unsubscribes, clean up data ready for next subscription.
  return {
    unsubscribe() {
      navigator.geolocation.clearWatch(watchId);
    }
  };
});

// Call subscribe() to start listening for updates.
const locationsSubscription = locations.subscribe({
  next(position) {
    console.log('Current Position: ', position);
  },
  error(msg) {
    console.log('Error Getting Location: ', msg);
  }
});

// Stop listening for location after 10 seconds
setTimeout(() => {
  locationsSubscription.unsubscribe();
}, 10000);
```

Observables - Operators

Observable Operators

The Operators are functions that operate on an Observable and return a new Observable.

The power of observable comes from the operators. You can use them to manipulate the incoming observable, filter it, merge it with another observable, alter the values or subscribe to another observable.

You can also chain each operator one after the other using the pipe. Each operator in the chain gets the observable from the previous operator. It modifies it and creates a new observable, which becomes the input for the next observable.

The following example shows the filer & map operators chained inside a pipe. The filter operator removes all data which is less than or equal to 2 and the map operator multiplies the value by 2.

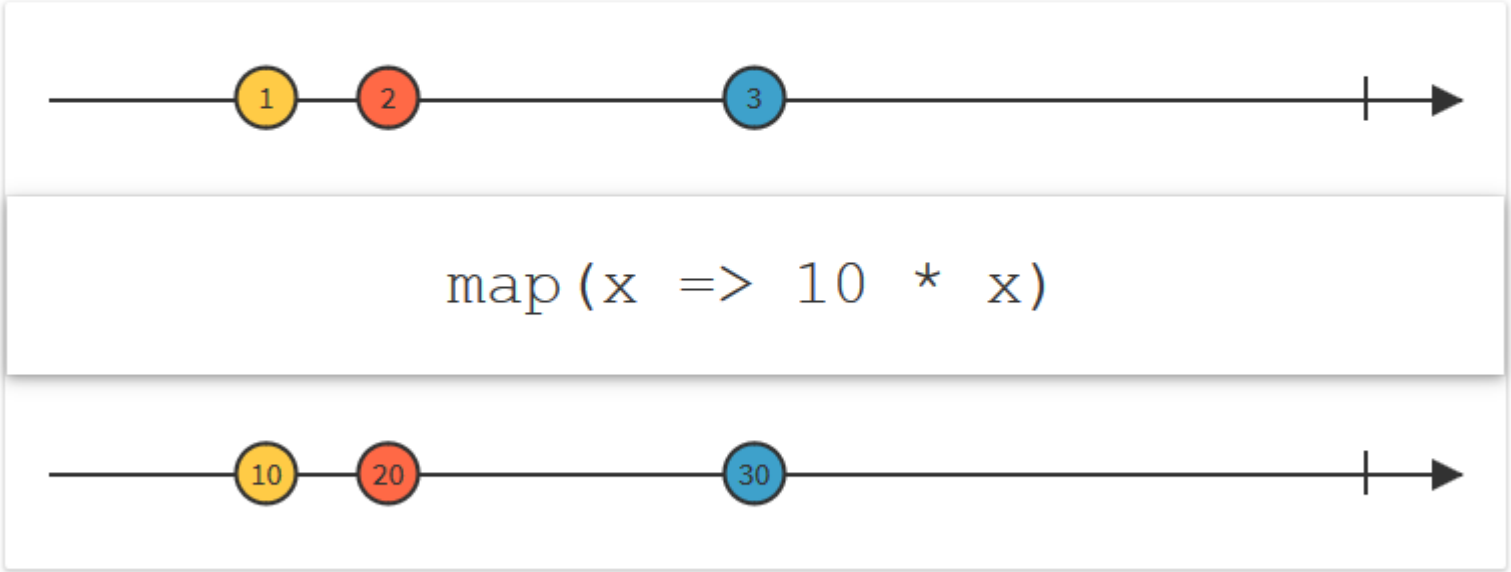
The input stream is [1,2,3,4,5] , while the output is [6, 8, 10].

```
1
2  obs.pipe(
3   obs = new Observable((observer) => {
4     observer.next(1)
5     observer.next(2)
6     observer.next(3)
7     observer.next(4)
8     observer.next(5)
9     observer.complete()
10   }).pipe(
11    filter(data => data > 2),   //filter Operator
12    map((val) => {return val as number * 2}), //map operator
13   )
14
```

Observables - Operators

Map

transform the items emitted by an Observable by applying a function to each item



The `Map` operator applies a function of your choosing to each item emitted by the source Observable, and returns an Observable that emits the results of these function applications.

Observables - Operators


```
// Operators for creating observables
getSerialAds(): Observable<string>
   {
    //  return of("ad1", "ad2", "ad3");
     return from(this.adsList)
   }
```

```
// Operators for filtering
    let filtersObservable = this.promoAds.getScheduledAds(3).pipe(
      filter(ad=>ad.includes("white Friday"))
      , map(ad=> "Ad: " + ad)
    );
    let adsSubscription=filtersObservable.subscribe(observer);
    this.subscriptions.push(adsSubscription);
```

```
// Operators for handling errors
    let filtersObservable = this.promoAds.getScheduledAds(3).pipe(
      retry(3),
      catchError((error: HttpErrorResponse) => {
        return throwError(
          () => new Error('Error occured, please try again')
        )
      })
    );
```

More details: https://reactivex.io/documentation/operators.html
https://rxjs.dev/guide/operators
https://www.learnrxjs.io/learn-rxjs/operators

Observables - Subjects

Subject

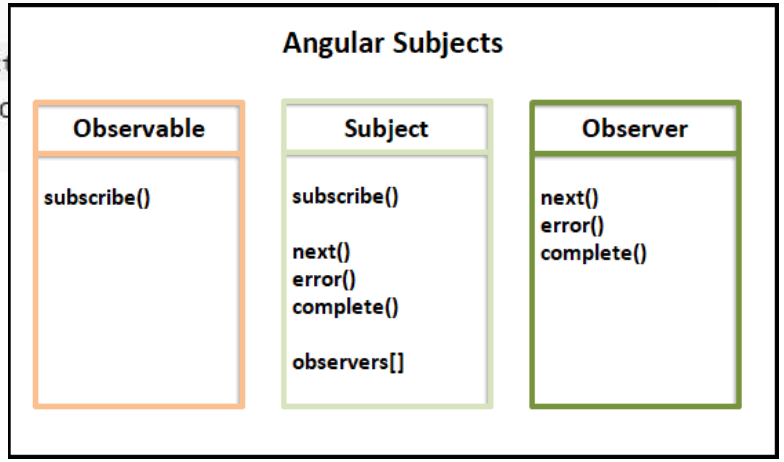
What is a Subject? An RxJS Subject is a special type of Observable that allows values to be multicasted to many Observers. While plain Observables are unicast (each subscribed Observer owns an independent execution of the Observable), Subjects are multicast.

> *A Subject is like an Observable, but can multicast to many Observers. Subjects are like EventEmitters: they maintain a registry of many listeners.*

Every Subject is an Observable. Given a Subject, you can `subscribe` to it, providing an Observer, which will start receiving values normally. From the perspective of the Observer, it cannot tell whether the Observable execution is coming from a plain unicast Observable or a Subject.

Internally to the Subject, `subscribe` does not invoke a new execution that delivers values. It simply registers the given Observer in a list of Observers, similarly to how `addListener` usually works in other libraries and languages.

Every Subject is an Observer. It is an object with the methods `next` new value to the Subject, just call `next(theValue)`, and it will be multic to the Subject.



Observables - Subjects

Subject

A Subject is a sort of bridge or proxy that is available in some implementations of ReactiveX that acts both as an observer and as an Observable. Because it is an observer, it can subscribe to one or more Observables, and because it is an Observable, it can pass through the items it observes by reemitting them, and it can also emit new items.

Because a Subject subscribes to an Observable, it will trigger that Observable to begin emitting items (if that Observable is "cold" — that is, if it waits for a subscription before it begins to emit items). This can have the effect of making the resulting Subject a "hot" Observable variant of the original "cold" Observable.

Since a Subject is an Observer, this also means you may provide a Subject as the argument to the `subscribe` of any Observable, like the example below shows:

```
1. import { Subject, from } from 'rxjs';
2.
3. const subject = new Subject<number>();
4.
5. subject.subscribe({
6.   next: (v) => console.log(`observerA: ${v}`)
7. });
8. subject.subscribe({
9.   next: (v) => console.log(`observerB: ${v}`)
10. });
11.
12. const observable = from([1, 2, 3]);
13.
14. observable.subscribe(subject); // You can subscribe providing a Subject
```

More details:
https://reactivex.io/documentation/subject.html
https://rxjs.dev/guide/subject
https://www.tektutorialshub.com/angular/subjects-in-angular/

Observables - Subjects


```
export class UserAuthService {
  private isloggedSubject: BehaviorSubject<boolean>;
  constructor() {
    this.isloggedSubject=new BehaviorSubject<boolean> (this.isUserLogged);
   }

  login(userName: string, password: string)
  {
    // Call login API, and get Access Token
    let usrToken='123456789';
    localStorage.setItem("token", usrToken);
    this.isloggedSubject.next(true);
  }

  logout()
  {
    localStorage.removeItem("token");
    this.isloggedSubject.next(false);
  }

  get isUserLogged(): boolean
  {
    return  (localStorage.getItem('token'))? true: false
  }

  getloggedStatus(): Observable<boolean>
  {
    return this.isloggedSubject.asObservable();
  }
}
```

More details: https://reactivex.io/documentation/subject.html

Observables (Cont.)

References:

https://angular.io/guide/observables

https://www.tektutorialshub.com/angular/angular-observable-tutorial-using-rxjs/#unsubscribing-from-an-observable

https://www.geeksforgeeks.org/angular-7-observables/

https://blog.angular-university.io/how-to-build-angular2-apps-using-rxjs-observable-data-services-pitfalls-to-avoid/

Httpclient

Angular 4.3 introduced a new HttpClient service, which is a replacement for the Http service from Angular 2. It works mostly the same as the old service, handling both single and concurrent data loading with RxJs Observables, and writing data to an API.

As of Angular 5.0, the older Http service still works, but it's deprecated and will be removed in a future release. The code samples in this post are compatible with Anguar 4.3, 5.x, and higher. If your project is still using Angular 4.2 or lower,

The `HttpClient` in `@angular/common/http` offers a simplified client HTTP API for Angular applications that rests on the `XMLHttpRequest` interface exposed by browsers. Additional benefits of `HttpClient` include testability features, typed request and response objects, request and response interception, `Observable` apis, and streamlined error handling.

However, data access rarely stays this simple. You typically post-process the data, add error handling, and maybe some retry logic to cope with intermittent connectivity.

The component quickly becomes cluttered with data access minutia. The component becomes harder to understand, harder to test, and the data access logic can't be re-used or standardized.

That's why it is a best practice to separate presentation of data from data access by encapsulating data access in a separate service and delegating to that service in the component, even in simple cases like this one.

Retrieving data with httpclient(cont.)

Setup

Before you can use the `HttpClient`, you need to import the Angular `HttpClientModule`. Most apps do so in the root `AppModule`.

app/app.module.ts (excerpt)

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

@NgModule({
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
  ],
```

Retrieving data with httpclient(cont.)

Having imported `HttpClientModule` into the `AppModule`, you can inject the `HttpClient` into an application class as shown in the following `ConfigService` example.

app/config/config.service.ts (excerpt)

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class ConfigService {
  constructor(private http: HttpClient) { }
}
```

HttpClient – get Method

In Products.Service.Ts:

```
getProducts():Observable<Array<Iproduct>>
{
    return
    this.httpClient.get<Array<Iproduct>>('http://localhost:8080/
    getproducts');
}
```

In Component (After injecting the service in constructor):

```
ngOnInit() {
    this.productService.getProducts()
    .subscribe(data=>{this.myProductList=data})
}
```

HttpClient – Using map

Using the `map()` Operator

The `map()` operator is similar to the `Array.map()` method. It lets you map observable responses to other values. For example:

```
 import { Observable} from 'rxjs';
import { map } from 'rxjs/operators';

getItems(): Observable<Array<any>> {

  return this.aService.getItems()
    .pipe(map(response => response.data));
}
```

The `getItems()` method returns an Observable. We're using the `map()` operator to return the *data* property of the response object. The operator enables us to map the response of the Observable stream to the *data* value.

More: https://www.techiediaries.com/angular-rxjs-tutorial/

HttpClient – Adding headers

Adding headers

Many servers require extra headers for save operations. For example, they may require a "Content-Type" header to explicitly declare the MIME type of the request body. Or perhaps the server requires an authorization token.

The `HeroesService` defines such headers in an `httpOptions` object that will be passed to every `HttpClient` save method.

app/heroes/heroes.service.ts (httpOptions)

```
import { HttpHeaders } from '@angular/common/http';

const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type':  'application/json',
    'Authorization': 'my-auth-token'
  })
};
```

HttpClient – post method

Making a POST request

Apps often POST data to a server. They POST when submitting a form. In the following example, the `HeroesService` posts when adding a hero to the database.

app/heroes/heroes.service.ts (addHero)

```
/** POST: add a new hero to the database */
addHero (hero: Hero): Observable<Hero> {
  return this.http.post<Hero>(this.heroesUrl, hero, httpOptions)
    .pipe(
      catchError(this.handleError('addHero', hero))
    );
}
```

The `HeroesComponent` initiates the actual POST operation by subscribing to the `Observable` returned by this service method.

app/heroes/heroes.component.ts (addHero)

```
this.heroesService.addHero(newHero)
  .subscribe(hero => this.heroes.push(hero));
```

HttpClient – post example

In Products.Service.Ts:

```
insertProduct(prd:Product){
        const httpOptions = {headers: new HttpHeaders({
            'Content-Type': 'application/json','Accept':' */*'
            //,'Authorization': 'my-auth-token'
        })};
    return this.httpClient.post('http://localhost:8080/insert',
    prd,httpOptions)
}
```

In Component (After injecting the service in constructor):

```
insertProduct(){
    this.prdService.insertProduct(this.newProduct)
    .subscribe(data=>{
        console.log(data);
        this.router.navigate(['/home']);
    });
}
```

HttpClient – Delete method

app/heroes/heroes.service.ts (deleteHero)

```
/** DELETE: delete the hero from the server */
deleteHero (id: number): Observable<{}> {
  const url = `${this.heroesUrl}/${id}`; // DELETE api/heroes/42
  return this.http.delete(url, httpOptions)
    .pipe(
      catchError(this.handleError('deleteHero'))
    );
}
```

The HeroesComponent initiates the actual DELETE operation by subscribing to the Observable returned by this service method.

app/heroes/heroes.component.ts (deleteHero)

```
this.heroesService.deleteHero(hero.id).subscribe();
```

> You must call *subscribe()* or nothing happens. Just calling HeroesService.deleteHero() **does not initiate the DELETE request.**

HttpClient – Error Handling

You *could* handle in the component by adding a second callback to the `.subscribe()`:

app/config/config.component.ts (showConfig v.3 with error handling)

```
showConfig() {
  this.configService.getConfig()
    .subscribe(
      (data: Config) => this.config = { ...data }, // success path
      error => this.error = error // error path
    );
}
```

Getting Error Details:
https://angular.io/guide/http#error-handling

HttpClient – Error Handling

You can use *pipes* to link operators together. Pipes let you combine multiple functions into a single function.
The pipe() function takes as its arguments the functions you want to combine and returns a new function that, when sequence.

app/config/config.service.ts (getConfig with retry)

```
getConfig() {
  return this.http.get<Config>(this.configUrl)
    .pipe(
      retry(3), // retry a failed request up to 3 times
      catchError(this.handleError) // then handle the error
    );
}
```

```
this.httpClient
      .post<IProduct>(`${environment.APIURL}/products`, JSON.stringify(newPrd),
this.httpOption)
      .pipe(
        retry(2),
        catchError((err: HttpErrorResponse) => {
          return throwError(
            () => new Error('Error occured, please try again')
          )
        })
      // catchError(this.handleError)
    );
```

Retrieving data with http - Tips (cont.)

It’s recommended to make a file that contains API links, to not be hard-coded:

```
//in envrionment.ts
export const environment = {
  production: false,
  API_URL: 'http://localhost:8080'
};
```

More about http in Angular:

https://angular.io/tutorial/toh-pt6#http
https://angular.io/guide/http
https://angular.io/guide/rx-library#the-rxjs-library
https://www.metaltoad.com/blog/angular-5-making-api-calls-httpclient-service
https://codecraft.tv/courses/angular/http/http-with-observables/
https://thinkster.io/tutorials/angular-2-http

HTTP Interceptor

Most interceptors transform the outgoing request before passing it to the next interceptor in the chain, by calling next.handle(transformedReq)

https://angular.io/api/common/http/HttpInterceptor

https://www.digitalocean.com/community/tutorials/how-to-use-angular-interceptors-to-manage-http-requests-and-error-handling

Create a class `HttpConfigInterceptor` and implement the interface `HttpInterceptor`. This is an example:

```
@Injectable()
export class HttpConfigInterceptor implements HttpInterceptor {
    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
      // ...
    }
}
```

You will set `token`, `Content-Type`, `Accept` type for an API request. Here is an example:

```
const token: string = localStorage.getItem('token');

request = request.clone({ headers: request.headers.set('Authorization', 'Bearer ' + token) });

request = request.clone({ headers: request.headers.set('Content-Type', 'application/json') });

request = request.clone({ headers: request.headers.set('Accept', 'application/json') });
```

Angular Forms

- **Forms are the mainstay of business applications. You use forms to log in, submit a help request, place an order, book a flight, schedule a meeting, and perform countless other data-entry tasks.**

- **Angular forms supports two-way data binding, change tracking, validation, and error handling,**

- **There are 2 types of forms in Angular:**
  - Template-driven forms.
  - Model-driven forms (Reactive Forms)

- **Reactive forms Vs. Template forms:**
  - Template-driven forms are created using html tags, but Reactive forms are created in the component class using form control.
  - Reactive forms are synchronous. Template-driven forms are asynchronous.

Angular Forms (Cont.)

Key differences

The table below summarizes the key differences between reactive and template-driven forms.

	REACTIVE	TEMPLATE-DRIVEN
Setup (form model)	More explicit, created in component class	Less explicit, created by directives
Data model	Structured	Unstructured
Predictability	Synchronous	Asynchronous
Form validation	Functions	Directives
Mutability	Immutable	Mutable
Scalability	Low-level API access	Abstraction on top of APIs

Immutability is a design pattern where something can't be modified after being instantiated.
More: https://angular.io/guide/forms-overview#introduction-to-forms-in-angular

Angular Forms – Template forms

- **You need to import FromsModule, and in add it to imports in the app.module.**

Track control state and validity with *ngModel*

Using `ngModel` in a form gives you more than just two-way data binding. It also tells you if the user touched the control, if the value changed, or if the value became invalid.

The *NgModel* directive doesn't just track state; it updates the control with special Angular CSS classes that reflect the state. You can leverage those class names to change the appearance of the control.

State	Class if true	Class if false
The control has been visited.	`ng-touched`	`ng-untouched`
The control's value has changed.	`ng-dirty`	`ng-pristine`
The control's value is valid.	`ng-valid`	`ng-invalid`

Angular Forms – Template forms


```
{{prdNameInp.className | json}}
```

Dr IQ

TODO: remove this: form-control ng-untouched ng-pristine ng-valid

Untouched

Dr IQ

TODO: remove this: form-control ng-pristine ng-valid ng-touched

Touched

Dr IQ////

TODO: remove this: form-control ng-valid ng-touched ng-dirty

Changed

TODO: remove this: form-control ng-touched ng-dirty ng-invalid

Invalid

Angular Forms – Template forms

Using ngFor to fill list

src/app/hero-form/hero-form.component.html (powers)

```
<div class="form-group">
  <label for="power">Hero Power</label>
  <select class="form-control" id="power" required>
    <option *ngFor="let pow of powers" [value]="pow">{{pow}}</option>
  </select>
</div>
```

Two-way binding:

src/app/hero-form/hero-form.component.html (excerpt)

```
<input type="text" class="form-control" id="name"
       required
       [(ngModel)]="model.name" name="name">
TODO: remove this: {{model.name}}
```

Angular Forms – Template forms

src/app/hero-form/hero-form.component.html (excerpt)

```
<form #heroForm="ngForm">
```

The variable `heroForm` is now a reference to the `NgForm` directive that governs the form as a whole.

The *NgForm* directive

What `NgForm` directive? You didn't add an NgForm directive.

Angular did. Angular automatically creates and attaches an `NgForm` directive to the `<form>` tag.

The `NgForm` directive supplements the `form` element with additional features. It holds the controls you created for the elements with an `ngModel` directive and `name` attribute, and monitors their properties, including their validity. It also has its own `valid` property which is true only *if every contained control* is valid.

Angular Forms – Template forms

src/app/hero-form/hero-form.component.html (excerpt)

```
<label for="name">Name</label>
<input type="text" class="form-control" id="name"
       required
       [(ngModel)]="model.name" name="name"
       #name="ngModel">
<div [hidden]="name.valid || name.pristine"
     class="alert alert-danger">
  Name is required
</div>
```

You need a template reference variable to access the input box's Angular control from within the template. Here you created a variable called `name` and gave it the value "ngModel".

> Why "ngModel"? A directive's exportAs property tells Angular how to link the reference variable to the directive. You set `name` to `ngModel` because the `ngModel` directive's `exportAs` property happens to be "ngModel".

You control visibility of the name error message by binding properties of the `name` control to the message `<div>` element's `hidden` property.

src/app/hero-form/hero-form.component.html (hidden-error-msg)

```
<div [hidden]="name.valid || name.pristine"
     class="alert alert-danger">
```

Angular Forms – Template forms

Difference between angular submit and ngSubmit events?
ngSubmit ensures that the form doesn't submit when the handler code throws and causes an actual http post request.

submit: It is html default form submit event, it will call underlying method when form gets submitted.
ngSubmit: Is host binding situated on form element. Basically it prevent default submit event of browser(which can be form post) by returning false. Eventually you can prevent traditional PostBack calls or page reload due to form load. This way you can validate your form & submit it to server by manual ajax from Component code

Angular Forms – Validation


```
<form #prdFrm="ngForm" (ngSubmit)="addProduct()">
  <div class="mb-3">
    <label for="pName" class="form-label">Product Name</label>
    <input #prdNameInp="ngModel" type="text" class="form-control"
        id="pName" name="prdname"
        [(ngModel)]="newPrd.name"
        required pattern="[A-Za-z]{3,}"
        [class.is-invalid]="prdNameInp.invalid"
        [class.is-valid]="prdNameInp.valid">
    <div class="alert alert-warning" [hidden]="prdNameInp.valid ||
prdNameInp.untouched">
      <small *ngIf="prdNameInp.errors?.['required']"> Name is required</small>
      <small *ngIf="prdNameInp.errors?.['pattern']"> Name must be 3 letters at
least</small>
    </div>
  </div>

  <button [disabled]="prdFrm.invalid" type="submit"
        class="btn btn-primary">Submit</button>
```

Angular Forms – Reactive forms

Compare using the form builder to creating the instances manually.

src/app/profile-editor/profile-editor.component.ts (instances) | src/app/profile-editor/profile-editor.

```
profileForm = new FormGroup({
  firstName: new FormControl(''),
  lastName: new FormControl(''),
  address: new FormGroup({
    street: new FormControl(''),
    city: new FormControl(''),
    state: new FormControl(''),
    zip: new FormControl('')
  })
});
```

Compare using the form builder to creating the instances manually.

-editor.component.ts (instances) | src/app/profile-editor/profile-editor.component.ts (form builder)

```
profileForm = this.fb.group({
  firstName: [''],
  lastName: [''],
  address: this.fb.group({
    street: [''],
    city: [''],
    state: [''],
    zip: ['']
```

Reactive forms - Example


```
this.userRegFrm = fb.group({
      fullName: ['', [Validators.required, Validators.pattern('[A-Za-z]{3,}')]],
      email: ['', [Validators.required)]],
      phoneNo: fb.array([this.fb.control('')]),
      address: fb.group({
        city: [''],
        postalCode: [''],
        street: ['']
      }),
      password: ['', [Validators.required]],
      confirmPassword: ['', [Validators.required]],
      referral: [''],
      referralOther: ['']
    });
```

```
 <form [formGroup]="userRegFrm" class="m-3" (ngSubmit)="submit()">
       <input type="text" formControlName="fullName" class="form-control" id="fName" name="userFname"
            [class.is-invalid]="fullName?.invalid" [class.is-valid]="fullName?.valid">

      <div formGroupName="address">
            <div class="mb-3">
              <label for="uCity" class="form-label">City</label>
              <input type="text" formControlName="city" class="form-control" id="uCity" name="usercity">
            </div>
            <div class="mb-3">
              <label for="uPostalCode" class="form-label">Postal Code</label>
              <input type="number" formControlName="postalCode" class="form-control"
                        id="uPostalCode" name="userpostalcode">
            </div>
      </div>
</  form  >
```

Reactive forms - Example


```
  get fullName() {
    return this.userRegFrm.get('fullName');
  }

ngOnInit(): void {
    // Check for path params, to specify user reg. or Edit profile
    // In case of EditProfile

    // Call API to get user profile
     this.userRegFrm.setValue({ // Must provide all properties
      fullName: 'ITI',
      email: 'info@iti.gov.eg',
      address:
      {
        city: 'Assiut',
        postalCode: 111,
        street: 'street 1'
      }
     });

    //  this.userRegFrm.get('fullName')?.setValue('Test');

    this.userRegFrm.patchValue({ // can provide some properties
      fullName: 'ITI',
      email: 'info@iti.gov.eg',
      address:
      {
        city: 'Assiut',
        postalCode: 111,
        street: 'street 1'
      }
     });
  }
```

Reactive forms – Dynamic forms


```
this.userRegFrm = fb.group({
      fullName: ['', [Validators.required, Validators.pattern('[A-Za-z]{3,}')]],
      phoneNo: fb.array([this.fb.control('')]),
     );
```

```
get phoneNumbers() {
    return this.userRegFrm.get('phoneNo') as FormArray;
  }

addPhoneNo(event: any) {
    this.phoneNumbers.push(this.fb.control(''));
    event.target?.classList.add('d-none');
  }
```

```
<div formArrayName="phoneNo">
      <div class="mb-3 ms-3" *ngFor=" let phoneNo of phoneNumbers.controls; let i=index">
        <label for="{{'uPhone-' + i}}" class="form-label">Phone No. #{{i+1}}</label>
        <input type="text" [formControlName]="i" class="form-control" id="{{'uPhone-' + i}}" name="{{'userPhone-'
+ i}}">
        <button type="button" class="btn btn-primary" (click)="addPhoneNo($event)"> + </button>
      </div>
    </div>
```

Reactive forms – Conditional validation


```
  updateReferralValidators() {
    if (this.referral?.value == "other") {
      this.userRegFrm.get('referralOther')?.addValidators([Validators.required]);
    }
    else {
      this.userRegFrm.get('referralOther')?.clearValidators();
    }
    this.userRegFrm.get('referralOther')?.updateValueAndValidity();
  }
```

```
    <div class="btn-group" role="group" aria-label="Basic radio toggle button group">
      <input type="radio" class="btn-check" formControlName="referral" name="referral" id="btnradio1"
autocomplete="off" value="SocialMedia" (change)="updateReferralValidators()">
      <label class="btn btn-outline-primary" for="btnradio1">Social media</label>

      <input type="radio" class="btn-check" formControlName="referral" name="referral" id="btnradio2"
autocomplete="off" value="Friend" (change)="updateReferralValidators()">
      <label class="btn btn-outline-primary" for="btnradio2">from a Friend</label>

      <input type="radio" class="btn-check" formControlName="referral" name="referral" id="btnradio3"
autocomplete="off" value="other" (change)="updateReferralValidators()">
      <label class="btn btn-outline-primary" for="btnradio3">other</label>
    </div>
```

Reactive forms – Custom validator


```
import { AbstractControl, ValidationErrors, ValidatorFn } from "@angular/forms";
// https://angular.io/guide/form-validation#defining-custom-validators
/*Validator functions
  Validator functions can be either synchronous or asynchronous.

  Sync validators: Synchronous functions that take a control instance and immediately return either a set of validation errors or null. Pass these in as the second argument when you instantiate a FormControl.

  Async validators: Asynchronous functions that take a control instance and return a Promise or Observable that later emits a set of validation errors or null. Pass these in as the third argument when you instantiate a FormControl.*/
  /* It's not recommended to use this implementation to send the email list,
     INSTEAD, use Async validator to call API,that takes the email value and returns boolean
     https://angular.io/guide/form-validation#creating-asynchronous-validators
     https://www.concretepage.com/angular-2/angular-custom-async-validator-example#AsyncValidatorFn
     https://www.tektutorialshub.com/angular/angular-async-validator-example/
     https://www.thisdot.co/blog/using-custom-async-validators-in-angular-reactive-forms
     https://codinglatte.com/posts/angular/how-to-add-async-validators-to-an-angular-reactive-form/ */
export function existEmailValidator(existEmails: string[]): ValidatorFn {
  return (control: AbstractControl): ValidationErrors | null => {
    let emailVal: string = control.value;
    if(emailVal.length==0 && control.untouched)
      return null;
    console.log(existEmails)
    let validationError = {'existEmail': { 'value': emailVal }}
    let foundEmail= existEmails.includes(emailVal);
    return foundEmail? validationError : null;
    // return (emailVal.includes('@')) ? null : validationError;
  }
}
```

```
email: ['', [Validators.required, existEmailValidator(this.existUserEmails)]],
```

Reactive forms – custom validator


```
<div class="mb-3">
  <label for="uEmail" class="form-label">Email</label>
  <input type="email" formControlName="email" class="form-control" id="uEmail" name="useremail">
  <div class="alert alert-warning" [hidden]="email?.valid || email?.untouched">
    <small *ngIf="email?.errors?.['required']">
      Email is required
    </small>
    <small *ngIf="email?.errors?.['existEmail']">
      Entered Email is used before, <b>{{email?.errors?.['existEmail'].value}}</b> can't be used
    </small>
  </div>
</div>
```

Reactive forms – Cross-field validator


```
// If validator has no parameters
export const passwordMatch: ValidatorFn =
  (frmGroup: AbstractControl): ValidationErrors | null => {
    let passControl= frmGroup.get('password');
    let confirmPassControl= frmGroup.get('confirmPassword');
    if(!passControl || !confirmPassControl || !passControl.value || !confirmPassControl.value)
      return null;

    let valErr={'UnmatchedPassword': {'pass': passControl?.value, 'confrim': confirmPassControl?.value}}
    return (passControl?.value==confirmPassControl?.value)? null : valErr;
  }

 // If validator has parameters
export function passwordMatchAdv(complexPassword: boolean=false): ValidatorFn
{
  //If complextPassword?, check fullname not included in password
  return (control: AbstractControl) : ValidationErrors | null=>{
    let passControl= control.get('password');
    let confirmPassControl= control.get('confirmPassword');
    if(!passControl || !confirmPassControl || !passControl.value || !confirmPassControl.value)
      return null;

    let valErr={'UnmatchedPassword': {'pass': passControl?.value, 'confrim': confirmPassControl?.value}}
    return (passControl?.value==confirmPassControl?.value)? null : valErr;
  }
}
```

```
this.userRegFrm = fb.group({
      fullName: ['', [Validators.required, Validators.pattern('[A-Za-z]{3,}')]],
      email: ['', [Validators.required, existEmailValidator(this.existUserEmails)]],
    }, {validators: passwordMatch});
```

Reactive forms – Cross-field validator


```
    <div class="mb-3">
      <label for="uCPass" class="form-label">Confirm Password</label>
      <input type="password" formControlName="confirmPassword" class="form-control" id="uCPass" name="usercpass">
      <!-- <div class="alert alert-warning" *ngIf="(confirmPassword?.touched && password?.touched)
&&  (confirmPassword?.invalid || userRegFrm?.errors?.['UnmatchedPassword'])"> -->
        <div class="alert alert-warning" *ngIf="confirmPassword?.errors?.['required'] &&
confirmPassword?.touched">
          Confirm Password is required
        </div>
        <div small class="alert alert-warning" *ngIf="userRegFrm?.errors?.['UnmatchedPassword']">
          Confirm password must match password.
        </div>
      <!-- </div> -->
    </div>
```

Angular Forms – More Reading

Template driven forms:
https://angular.io/guide/forms#template-driven-forms

Reactive Forms:
https://angular.io/guide/reactive-forms

Dynamic Form:
https://angular.io/guide/dynamic-form

Angular Deployment

For the simplest deployment, build for development and copy the output directory to a web server.

1. Start with the development build

```
ng build
```

2. Copy *everything* within the output folder (`dist/` by default) to a folder on the server.

3. If you copy the files into a server *sub-folder*, append the build flag, `--base-href` and set the `<base href>` appropriately.

   For example, if the `index.html` is on the server at `/my/app/index.html` , set the *base href* to `<base href="/my/app/">` like this.

```
ng build --base-href=/my/app/
```

   You'll see that the `<base href>` is set properly in the generated `dist/index.html` .

   If you copy to the server's root directory, omit this step and leave the `<base href>` alone.

   Learn more about the role of `<base href>` below.

4. Configure the server to redirect requests for missing files to `index.html` . Learn more about server-side redirects below.

Angular Deployment (Cont.)

Optimize for production

Although deploying directly from the development environment works, you can generate an optimized build with additional CLI command line flags, starting with `--prod` .

Build with *--prod*

```
ng build --prod
```

Deprecated: Use: ng build --configuration production instead of –prod.

The `--prod` *meta-flag* engages the following optimization features.

- Ahead-of-Time (AOT) Compilation: pre-compiles Angular component templates.
- Production mode: deploys the production environment which enables *production mode*.
- Bundling: concatenates your many application and library files into a few bundles.
- Minification: removes excess whitespace, comments, and optional tokens.
- Uglification: rewrites code to use short, cryptic variable and function names.
- Dead code elimination: removes unreferenced modules and much unused code.

The remaining copy deployment steps are the same as before.

Angular Deployment (Cont.)

Angular compilation

An Angular application consists largely of components and their HTML templates. Before the browser can render the application, the components and templates must be converted to executable JavaScript by an *Angular compiler*.

Angular offers two ways to compile your application:

1. *Just-in-Time* (JIT), which compiles your app in the browser at runtime
2. *Ahead-of-Time* (AOT), which compiles your app at build time.

JIT compilation is the default when you run the *build-only* or the *build-and-serve-locally* CLI commands:

```
ng build
ng serve
```

For AOT compilation, append the `--aot` flags to the *build-only* or the *build-and-serve-locally* CLI commands:

```
ng build --aot
ng serve --aot
```

> The `--prod` meta-flag compiles with AOT by default.
>
> See the CLI documentation for details, especially the `build` topic.

Angular Deployment (Cont.)

Why compile with AOT?

Faster rendering

With AOT, the browser downloads a pre-compiled version of the application. The browser loads executable code so it can render the application immediately, without waiting to compile the app first.

Fewer asynchronous requests

The compiler *inlines* external HTML templates and CSS style sheets within the application JavaScript, eliminating separate ajax requests for those source files.

Smaller Angular framework download size

There's no need to download the Angular compiler if the app is already compiled. The compiler is roughly half of Angular itself, so omitting it dramatically reduces the application payload.

Detect template errors earlier

The AOT compiler detects and reports template binding errors during the build step before users can see them.

Better security

AOT compiles HTML templates and components into JavaScript files long before they are served to the client. With no templates to read and no risky client-side HTML or JavaScript evaluation, there are fewer opportunities for injection attacks.

Angular Deployment – More Reading

https://angular.io/guide/setup

https://angular.io/guide/setup-systemjs-anatomy

https://angular.io/guide/npm-packages

https://angular.io/guide/typescript-configuration

https://angular.io/guide/aot-compiler

https://angular.io/guide/deployment

Angular Apps Debugging

https://blog.angularindepth.com/everything-you-need-to-know-about-debugging-angular-applications-d308ed8a51b4

https://angularfirebase.com/lessons/methods-for-debugging-an-angular-application/

https://augury.angular.io/pages/guides/

More Reading…

More about components & Templates:
- https://angular.io/guide/component-styles
- https://angular.io/guide/elements
- https://angular.io/guide/dynamic-component-loader
- https://angular.io/guide/animations

Bootstrapping: https://angular.io/guide/bootstrapping

Angular NgModules:
- https://angular.io/guide/ngmodules
- https://angular.io/guide/ngmodule-vs-jsmodule
- https://angular.io/guide/frequent-ngmodules
- https://angular.io/guide/module-types
- https://angular.io/guide/entry-components
- https://angular.io/guide/feature-modules
- https://angular.io/guide/providers
- https://angular.io/guide/singleton-services
- https://angular.io/guide/lazy-loading-ngmodules
- https://angular.io/guide/sharing-ngmodules
- https://angular.io/guide/ngmodule-api

More about Dependency Injection in Angular:
- https://angular.io/guide/dependency-injection-pattern
- https://angular.io/guide/dependency-injection
- https://angular.io/guide/hierarchical-dependency-injection
- https://angular.io/guide/dependency-injection-in-action

More about Observables, RxJS:
- https://angular.io/guide/observables
- https://angular.io/guide/rx-library
- https://angular.io/guide/observables-in-angular
- https://angular.io/guide/practical-observable-usage
- https://angular.io/guide/comparing-observables
- https://blog.angular-university.io/rxjs-higher-order-mapping
- https://www.c-sharpcorner.com/blogs/rxjs-operators-in-angular
- https://indepth.dev/posts/1114/learn-to-combine-rxjs-sequences-with-super-intuitive-interactive-diagrams

Unit tests for angular apps: https://angular.io/guide/testing

Angular Internationalization: https://angular.io/guide/i18n

Angular Language Service: https://angular.io/guide/language-service

Angular service worker: https://angular.io/guide/service-worker-intro

Security guidelines: https://angular.io/guide/security

Angular cheat sheet: https://angular.io/guide/cheatsheet

More Reading…

- Angular Lazy-loading:
  - https://angular.io/guide/lazy-loading-ngmodules
  - https://www.youtube.com/watch?v=NWZ6K-dV2uw
  - https://www.youtube.com/watch?v=ecMmo0PRUKA

- Authentication using Angular:
  - https://angular.io/guide/router#milestone-5-route-guards
  - https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3
  - https://blog.angular-university.io/angular-jwt-authentication/
  - https://code.tutsplus.com/tutorials/jwt-authentication-in-angular--cms-32006
  - https://fullstackmark.com/post/13/jwt-authentication-with-aspnet-core-2-web-api-angular-5-net-core-identity-and-facebook-login
  - http://jasonwatmore.com/post/2018/10/29/angular-7-user-registration-and-login-example-tutorial
  - https://www.youtube.com/watch?v=e8BlURn6SFk
  - https://www.youtube.com/watch?v=ozXGkqpzo_A&list=PLC3y8-rFHvwg2RBz6UplKTGIXREj9dV0G
  - https://www.youtube.com/watch?v=rajjv-0bSps&list=PLYxzS__5yYQlqCmHqDyW3yo5V79C7eaTe&index=15

- Localization (Multi-language) in Angular:
  - https://angular.io/guide/i18n
  - https://www.youtube.com/watch?v=74OCrD6Ckgg
  - https://www.youtube.com/watch?v=TTZ7IWJO6pk
  - https://www.youtube.com/watch?v=5-6dwYtfRO4

- Angular Deployment for Production:
  - https://angular.io/guide/deployment
  - https://www.youtube.com/watch?v=AvV0BwAyxes

- Angular and SEO (Angular Universal):
  - https://angular.io/guide/universal
  - https://www.youtube.com/watch?v=hORh4TQL0zw
  - https://www.youtube.com/watch?v=lncsmB5yfzE
  - https://www.youtube.com/watch?v=hxG9nuvnh-A

More Reading…

Angular Unit testing:
https://blog.logrocket.com/angular-unit-testing-tutorial-examples/
https://medium.com/swlh/angular-unit-testing-jasmine-karma-step-by-step-e3376d110ab4
https://angular.io/guide/testing

Subject, BehaviorSubject, ReplaySubject:
https://stackoverflow.com/questions/34376854/delegation-eventemitter-or-observable-in-angular/35568924#35568924
https://dev.to/alfredoperez/angular-service-to-handle-state-using-behaviorsubject-4818
https://www.c-sharpcorner.com/article/how-to-use-behaviorsubject-in-angular/
https://medium.com/@weswhite/angular-behaviorsubject-service-60485ef064fc
https://dev.to/muhammadawaisshaikh/subject-rxjs-in-angular-3dg5

Subject:
http://reactivex.io/rxjs/manual/overview.html#subject

BehaviorSubject:
http://reactivex.io/rxjs/manual/overview.html#behaviorsubject

ReplaySubject:
http://reactivex.io/rxjs/manual/overview.html#replaysubject

Angular & Firebase App:
https://www.youtube.com/playlist?list=PLL2zWZTDFZzh2WEmc3fH_O4y4N05ZCqB2
