**angular-sweetalert module**

**AngularJS wrapper for SweetAlert**

AngularJS wrapper for **[SweetAlert](http://tristanedwards.me/sweetalert)**. Sweet Alert is a beautiful replacement for Javascript's "Alert".

## Demo

[**http://oitozero.github.io/ngSweetAlert/**](http://oitozero.github.io/ngSweetAlert/)

## Dependencies

* required:  
  [AngularJS](https://github.com/angular/angular)  
  [sweetalert](https://github.com/t4t5/sweetalert)

## Install

npm i angular-sweetalert --save

1. download the files
   1. Bower
      1. add "angular-sweetalert": "latest" to your bower.json file then run bower install OR run bower install angular-sweetalert
2. include the files in your app
   1. SweetAlert.min.js
   2. sweet-alert.js OR sweet-alert.min.js
   3. sweet-alert.css
3. include the module in angular (i.e. in app.js) - oitozero.ngSweetAlert

# ngSweetAlert

AngularJS wrapper for [SweetAlert](http://tristanedwards.me/sweetalert" \t "_blank). Sweet Alert is a beautiful replacement for Javascript's "Alert".

### Examples

A basic message

SweetAlert.swal("Here's a message");

A title with a text under

SweetAlert.swal("Here's a message!", "It's pretty, isn't it?");

A success message!

SweetAlert.swal("Good job!", "You clicked the button!", "success");

A warning message, with a function attached to the "Confirm"-button

SweetAlert.swal({  
   title: "Are you sure?",  
   text: "Your will not be able to recover this imaginary file!",  
   type: "warning",  
   showCancelButton: true,  
   confirmButtonColor: "#DD6B55",  
   confirmButtonText: "Yes, delete it!",  
   closeOnConfirm: false},   
function(){   
   SweetAlert.swal("Booyah!");  
});

... and by passing a parameter, you can execute something else for "Cancel".

SweetAlert.swal({  
   title: "Are you sure?",  
   text: "Your will not be able to recover this imaginary file!",  
   type: "warning",  
   showCancelButton: true,  
   confirmButtonColor: "#DD6B55",confirmButtonText: "Yes, delete it!",  
   cancelButtonText: "No, cancel plx!",  
   closeOnConfirm: false,  
   closeOnCancel: false },   
function(isConfirm){   
   if (isConfirm) {  
      SweetAlert.swal("Deleted!", "Your imaginary file has been deleted.", "success");  
   } else {  
      SweetAlert.swal("Cancelled", "Your imaginary file is safe :)", "error");  
   }  
});

A message with a custom icon

SweetAlert.swal({  
   title: "Sweet!",  
   text: "Here's a custom image.",  
   imageUrl: "https://avatars0.githubusercontent.com/u/4194490?s=400&v=4" });

**Angular Toaster**



## Features

* Toast Component Injection without being passed ViewContainerRef
* No use of \*ngFor. Fewer dirty checks and higher performance.
* AoT compilation and lazy loading compatible
* Component inheritance for custom toasts
* SystemJS/UMD rollup bundle
* Animations using Angular's [Web Animations API](https://angular.io/docs/ts/latest/guide/animations.html)
* Output toasts to an optional target directive

## Dependencies

Latest version available for each version of Angular

| **ngx-toastr** | **Angular** |
| --- | --- |
| 11.3.3 | 8.x |
| 12.1.0 | 9.x |
| 13.2.1 | 10.x 11.x |
| 14.3.0 | 12.x 13.x |
| current | >= 14.x |

## Install

npm install ngx-toastr --save

@angular/animations package is a required dependency for the default toast

npm install @angular/animations --save

Don't want to use @angular/animations? See [**Setup Without Animations**](https://www.npmjs.com/package/ngx-toastr#setup-without-animations).

## Setup

**step 1:** add css

* copy [toast css](https://github.com/scttcper/ngx-toastr/blob/HEAD/src/lib/toastr.css) to your project.
* If you are using sass you can import the css.

// regular style toast

@import '~ngx-toastr/toastr';

// bootstrap style toast

// or import a bootstrap 4 alert styled design (SASS ONLY)

// should be after your bootstrap imports, it uses bs4 variables, mixins, functions

@import '~ngx-toastr/toastr-bs4-alert';

// if you'd like to use it without importing all of bootstrap it requires

@import '~bootstrap/scss/functions';

@import '~bootstrap/scss/variables';

@import '~bootstrap/scss/mixins';

@import '~ngx-toastr/toastr-bs4-alert';

* If you are using angular-cli you can add it to your angular.json

"styles": [

"styles.scss",

"node\_modules/ngx-toastr/toastr.css" // try adding '../' if you're using angular cli before 6

]

**step 2:** add ToastrModule to app NgModule, make sure you have BrowserAnimationsModule as well

import { CommonModule } from '@angular/common';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { ToastrModule } from 'ngx-toastr';

@NgModule({

imports: [

CommonModule,

BrowserAnimationsModule, // required animations module

ToastrModule.forRoot(), // ToastrModule added

],

bootstrap: [App],

declarations: [App],

})

class MainModule {}

## Use

import { ToastrService } from 'ngx-toastr';

@Component({...})

export class YourComponent {

constructor(private toastr: ToastrService) {}

showSuccess() {

this.toastr.success('Hello world!', 'Toastr fun!');

}

}

## Options

There are **individual options** and **global options**.

### Individual Options

Passed to ToastrService.success/error/warning/info/show()

| **Option** | **Type** | **Default** | **Description** |
| --- | --- | --- | --- |
| toastComponent | Component | Toast | Angular component that will be used |
| closeButton | boolean | false | Show close button |
| timeOut | number | 5000 | Time to live in milliseconds |
| extendedTimeOut | number | 1000 | Time to close after a user hovers over toast |
| disableTimeOut | boolean | 'timeOut' | 'extendedTimeOut' | false | Disable both timeOut and extendedTimeOut when set to true. Allows specifying which timeOut to disable, either: timeOut or extendedTimeOut |
| easing | string | 'ease-in' | Toast component easing |
| easeTime | string | number | 300 | Time spent easing |
| enableHtml | boolean | false | Allow html in message |
| newestOnTop | boolean | true | New toast placement |
| progressBar | boolean | false | Show progress bar |
| progressAnimation | 'decreasing' | 'increasing' | 'decreasing' | Changes the animation of the progress bar. |
| toastClass | string | 'ngx-toastr' | Class on toast |
| positionClass | string | 'toast-top-right' | Class on toast container |
| titleClass | string | 'toast-title' | Class inside toast on title |
| messageClass | string | 'toast-message' | Class inside toast on message |
| tapToDismiss | boolean | true | Close on click |
| onActivateTick | boolean | false | Fires changeDetectorRef.detectChanges() when activated. Helps show toast from asynchronous events outside of Angular's change detection |

#### Setting Individual Options

success, error, info, warning take (message, title, ToastConfig) pass an options object to replace any default option.

this.toastrService.error('everything is broken', 'Major Error', {

timeOut: 3000,

});

### Global Options

All [**individual options**](https://www.npmjs.com/package/ngx-toastr#individual-options) can be overridden in the global options to affect all toasts. In addition, global options include the following options:

| **Option** | **Type** | **Default** | **Description** |
| --- | --- | --- | --- |
| maxOpened | number | 0 | Max toasts opened. Toasts will be queued. 0 is unlimited |
| autoDismiss | boolean | false | Dismiss current toast when max is reached |
| iconClasses | object | [see below](https://www.npmjs.com/package/ngx-toastr#iconclasses-defaults) | Classes used on toastr service methods |
| preventDuplicates | boolean | false | Block duplicate messages |
| countDuplicates | boolean | false | Displays a duplicates counter (preventDuplicates must be true). Toast must have a title and duplicate message |
| resetTimeoutOnDuplicate | boolean | false | Reset toast timeout on duplicate (preventDuplicates must be true) |
| includeTitleDuplicates | boolean | false | Include the title of a toast when checking for duplicates (by default only message is compared) |

##### iconClasses defaults

iconClasses = {

error: 'toast-error',

info: 'toast-info',

success: 'toast-success',

warning: 'toast-warning',

};

#### Setting Global Options

Pass values to ToastrModule.forRoot()

// root app NgModule

imports: [

ToastrModule.forRoot({

timeOut: 10000,

positionClass: 'toast-bottom-right',

preventDuplicates: true,

}),

],

### Toastr Service methods return:

export interface ActiveToast {

/\*\* Your Toast ID. Use this to close it individually \*/

toastId: number;

/\*\* the title of your toast. Stored to prevent duplicates if includeTitleDuplicates set \*/

title: string;

/\*\* the message of your toast. Stored to prevent duplicates \*/

message: string;

/\*\* a reference to the component see portal.ts \*/

portal: ComponentRef<any>;

/\*\* a reference to your toast \*/

toastRef: ToastRef<any>;

/\*\* triggered when toast is active \*/

onShown: Observable<any>;

/\*\* triggered when toast is destroyed \*/

onHidden: Observable<any>;

/\*\* triggered on toast click \*/

onTap: Observable<any>;

/\*\* available for your use in custom toast \*/

onAction: Observable<any>;

}

### Put toasts in your own container

Put toasts in a specific div inside your application. This should probably be somewhere that doesn't get deleted. Add ToastContainerModule to the ngModule where you need the directive available. Make sure that your container has an aria-live="polite" attribute, so that any time a toast is injected into the container it is announced by screen readers.

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

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

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { ToastrModule, ToastContainerModule } from 'ngx-toastr';

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

@NgModule({

declarations: [AppComponent],

imports: [

BrowserModule,

BrowserAnimationsModule,

ToastrModule.forRoot({ positionClass: 'inline' }),

ToastContainerModule,

],

providers: [],

bootstrap: [AppComponent],

})

export class AppModule {}

Add a div with toastContainer directive on it.

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

import { ToastContainerDirective, ToastrService } from 'ngx-toastr';

@Component({

selector: 'app-root',

template: `

<h1><a (click)="onClick()">Click</a></h1>

<div aria-live="polite" toastContainer></div>

`,

})

export class AppComponent implements OnInit {

@ViewChild(ToastContainerDirective, { static: true })

toastContainer: ToastContainerDirective;

constructor(private toastrService: ToastrService) {}

ngOnInit() {

this.toastrService.overlayContainer = this.toastContainer;

}

onClick() {

this.toastrService.success('in div');

}

}

## Functions

##### Clear

Remove all or a single toast by optional id

toastrService.clear(toastId?: number);

##### Remove

Remove and destroy a single toast by id

toastrService.remove(toastId: number);

## SystemJS

If you are using SystemJS, you should also adjust your configuration to point to the UMD bundle.

In your SystemJS config file, map needs to tell the System loader where to look for ngx-toastr:

map: {

'ngx-toastr': 'node\_modules/ngx-toastr/bundles/ngx-toastr.umd.min.js',

}

## Setup Without Animations

If you do not want to include @angular/animations in your project you can override the default toast component in the global config to use ToastNoAnimation instead of the default one.

In your main module (ex: app.module.ts)

import { ToastrModule, ToastNoAnimation, ToastNoAnimationModule } from 'ngx-toastr';

@NgModule({

imports: [

// ...

// BrowserAnimationsModule no longer required

ToastNoAnimationModule.forRoot(),

],

// ...

})

class AppModule {}

That's it! Animations are no longer required.

## Using A Custom Toast

Create your toast component extending Toast see the demo's pink toast for an example [**https://github.com/scttcper/ngx-toastr/blob/master/src/app/pink.toast.ts**](https://github.com/scttcper/ngx-toastr/blob/master/src/app/pink.toast.ts)

import { ToastrModule } from 'ngx-toastr';

@NgModule({

imports: [

ToastrModule.forRoot({

toastComponent: YourToastComponent, // added custom toast!

}),

],

bootstrap: [App],

declarations: [App, YourToastComponent], // add!

})

class AppModule {}

## FAQ

1. ExpressionChangedAfterItHasBeenCheckedError: Expression has changed after it was checked  
   When opening a toast inside an angular lifecycle wrap it in setTimeout

ngOnInit() {

setTimeout(() => this.toastr.success('sup'))

}

1. Change default icons (check, warning sign, etc)  
   Overwrite the css background-image [**https://github.com/scttcper/ngx-toastr/blob/master/src/lib/toastr.css**](https://github.com/scttcper/ngx-toastr/blob/master/src/lib/toastr.css)
2. How do I use this in an ErrorHandler? See: [**https://github.com/scttcper/ngx-toastr/issues/179**](https://github.com/scttcper/ngx-toastr/issues/179)
3. How can I translate messages See: [**https://github.com/scttcper/ngx-toastr/issues/201**](https://github.com/scttcper/ngx-toastr/issues/201)
4. How to handle toastr click/tap action?

showToaster() {

this.toastr.success('Hello world!', 'Toastr fun!')

.onTap

.pipe(take(1))

.subscribe(() => this.toasterClickedHandler());

}

toasterClickedHandler() {

console.log('Toastr clicked');

}

## Previous Works

**Angular wrapper for SweetAlert2**

This is a simple wrapper to use **[SweetAlert](https://limonte.github.io/sweetalert2/)** in Angular projects, it works with angular-cli also

### Install

npm install --save angular-sweetalert-service

or if you prefer Yarn

yarn add angular-sweetalert-service

### Include in your app

In your app.module just include the service

**...**

**import** { SweetAlertService } **from** 'angular-sweetalert-service';

**...**

Then add the service as a provider

@NgModule({

  declarations**:** [

**...**

  ],

  imports**:** [

**...**

  ],

  providers**:** [

**...**

    SweetAlertService,

**...**

  ],

  bootstrap**:** [

**...**

  ]

})

Now you have the service available across the application. Now you need to call the service in your component and that's it.

*// myComponent.ts*

**...**

**import** { SweetAlertService } **from** 'angular-sweetalert';

@Component({

  selector**:** 'app-foobar',

  styles**:** [],

  templateUrl**:** './foobar.html'

})

**export** class MyDummyClass {

  constructor(

    private alertService**:** SweetAlertService

  ) {}

**...**

### Available methods

SweetAlert.confirm SweetAlert.prompt SweetAlert.alert SweetAlert.question SweetAlert.success SweetAlert.warning SweetAlert.error

You can extend the default options by just passing a configuration object into the method, like this:

const options **=** {

  title**:** 'Are you sure?',

  text**:** "You won't be able to revert this!",

  type**:** 'warning',

  showCancelButton**:** true,

  confirmButtonColor**:** '#3085d6',

  cancelButtonColor**:** '#d33',

  confirmButtonText**:** 'Yes, delete it!'

};

SweetAlert.confirm(options);

### Chaining & Promises

    this.alertService.confirm({

      title**:** 'Delete account?'

    })

    .then(() => {

      this.alertService.success({

        title**:** 'Account deleted'

      });

    })

    .catch(() => console.log('canceled'));

### Typescript

If you use Typescript, if you don't use it you should, you can access the types since the service has the d.ts file available.

**export** declare class SweetAlertService {

    constructor();

    swal(): any;

    confirm(options: any): any;

    prompt(options: any): any;

    alert(options: any): any;

    question(options: any): any;

    success(options: any): any;

    warning(options: any): any;

    error(options: any): any;

    info(options: any): any;

}

**sweetalert2**

11.5.0 • Public • Published on 10-10-2022

A beautiful, responsive, customizable, accessible (WAI-ARIA) replacement  
for JavaScript's popup boxes. Zero dependencies.

## Sponsors

For all questions related to sponsorship please contact me via email [**limon.monte@protonmail.com**](mailto:limon.monte@protonmail.com)