**Ionic** is open source framework used for developing mobile applications. It provides tools and services for building Mobile UI with native look and feel. Ionic framework needs native wrapper to be able to run on mobile devices.

**Ionic** is a front-end HTML framework built on top of **AngularJS** and **Cordova**. As per their official document, the definition of this Ionic Open Source Framework is as follows −

Ionic is an **HTML5 Mobile App Development Framework** targeted at building hybrid mobile apps. Think of Ionic as the front-end UI framework that handles all the look and feel and UI interactions your app needs to be compelling. Kind of like "Bootstrap for Native", but with the support for a broad range of common native mobile components, slick animations and a beautiful design.

* Ionic Framework Features

Following are the most important features of Ionic −

* **AngularJS** − Ionic is using AngularJS MVC architecture for building rich single page applications optimized for mobile devices.
* **CSS components** − With the native look and feel, these components offer almost all elements that a mobile application needs. The components’ default styling can be easily overridden to accommodate your own designs.
* **JavaScript components** − These components are extending CSS components with JavaScript functionalities to cover all mobile elements that cannot be done only with HTML and CSS.
* **Cordova Plugins** − Apache Cordova plugins offer API needed for using native device functions with JavaScript code.
* **Ionic CLI** − This is NodeJS utility powered with commands for starting, building, running and emulating Ionic applications.
* **Ionic View** − Very useful platform for uploading, sharing and testing your application on native devices.
* **Licence** − Ionic is released under MIT license.
* Ionic Framework Advantages

Following are some of the most commonly known Ionic Framework Advantages −

* Ionic is used for Hybrid App Development. This means that you can package your applications for IOS, Android, Windows Phone and Firefox OS, which can save you a lot of working time.
* Starting your app is very easy since Ionic provides useful pre-generated app setup with simple layouts.
* The apps are built in a very clean and modular way, so it is very maintainable and easy to update.
* Ionic Developers Team have a very good relationship with the Google Developers Team and they are working together to improve the framework. The updates are coming out regularly and Ionic support group is always willing to help when needed.
* Ionic Framework Limitations

Following are some of the most important Ionic Framework Limitations −

* Testing can be tricky since the browser does not always give you the right information about the phone environment. There are so many different devices as well as platforms and you usually need to cover most of them.
* It can be hard to combine different native functionalities. There will be many instances where you would run into plugin compatibility issues, which leads to build errors that are hard to debug.
* Hybrid apps tend to be slower than the native ones. However, since the mobile technologies are improving fast this will not be an issue in the future.
* NodeJS : This is the base platform needed to create Mobile Apps using Ionic.
* Android SDK:

If you are going to work on a Windows platform and are developing your apps for the Android platform, then you should have Android SDK setup on your machine.

* cordova and Ionic

These are the main SDKs which is needed to start working with Ionic.

* Installing Cordova and Ionic

We will use the Windows command prompt for this tutorial. Open your command window to install Cordova and Ionic –

C:\Users\Username> npm install -g cordova ionic

**Creating Apps**

While creating apps in Ionic, you can choose from the following three options to start with −

* Tabs App
* Blank App
* Side menu app

In your command window, open the folder where you want to create the app and try one of the options mentioned below.

**Tabs App**

If you want to use the Ionic tabs template, the app will be built with the tab menu, header and a couple of useful screens and functionalities. This is the default Ionic template. Open your command window and choose where you want to create your app.

C:\Users\Username> cd Desktop

This command will change the working directory. The app will be created on the Desktop.

C:\Users\Username\Desktop> ionic start myApp tabs

Ionic Start command will create a folder named myApp and setup Ionic files and folders.

C:\Users\Username\Desktop> cd myApp

Now, we want to access the myApp folder that we just created. This is our root folder. Let us now add the Cordova project for the Android Platform and install the basic Cordova plugins as well. The following code allows us to run the app on the Android emulator or a device.

C:\Users\Username\Desktop\myApp> ionic platform add android

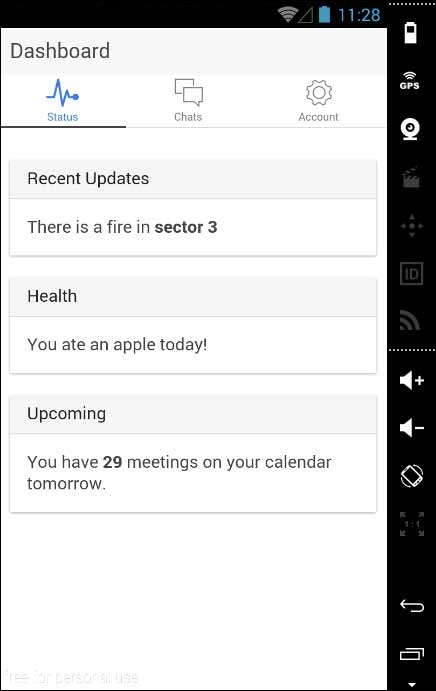
The next step is to build the app. If you have building errors after running the following command, you probably did not install the Android SDK and its dependencies.

C:\Users\Username\Desktop\myApp> ionic build android

The last step of the installation process is to run your app, which will start the mobile device, if connected, or the default emulator, if there is no device connected. Android Default Emulator is slow, so I suggest you to install [Genymotion](https://www.genymotion.com/#!/) or some other popular Android Emulator.

C:\Users\Username\Desktop\myApp> ionic run android

This will produce below result, which is an Ionic Tabs App.



**Blank App**

If you want to start from the scratch, you can install the Ionic blank template. We will use the same steps that have been explained above with the addition of ionic start myApp blank instead of ionic start myApp tabs as follows.

C:\Users\Username\Desktop> ionic start myApp blank

The Ionic Start command will create a folder named myApp and setup the Ionic files and folders.

C:\Users\Username\Desktop> cd myApp

Let us add the Cordova project for the Android Platform and install the basic Cordova plugins as explained above.

C:\Users\Username\Desktop\myApp>ionic platform add android

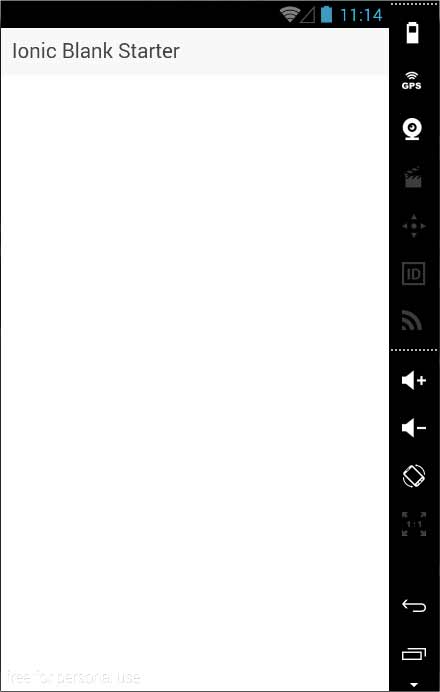
The next step is to build our app –

C:\Users\Username\Desktop\myApp> ionic build android

Finally, we will start the App as with the following code –

C:\Users\Username\Desktop\myApp> ionic run android

This will produce the following result, which is a Ionic Blank App.



**Side Menu App**

The third template that you can use is the Side Menu Template. The steps are the same as the previous two templates; we will just add sidemenu when starting our app as shown in the code below.

C:\Users\Username\Desktop> ionic start myApp sidemenu

The Ionic Start command will create a folder named myApp and setup the Ionic files and folders.

C:\Users\Username\Desktop> cd myApp

Let us add the Cordova project for the Android Platform and install the basic Cordova plugins with the code given below.

C:\Users\Username\Desktop\myApp> ionic platform add android

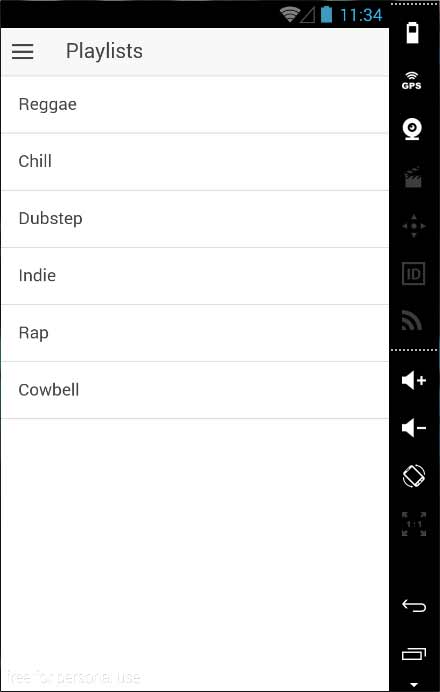
The next step is to build our app with the following code.

C:\Users\Username\Desktop\myApp> ionic build android

Finally, we will start the App with the code given below.

C:\Users\Username\Desktop\myApp> ionic run android

This will produce the following result, which is an Ionic Side Menu App.

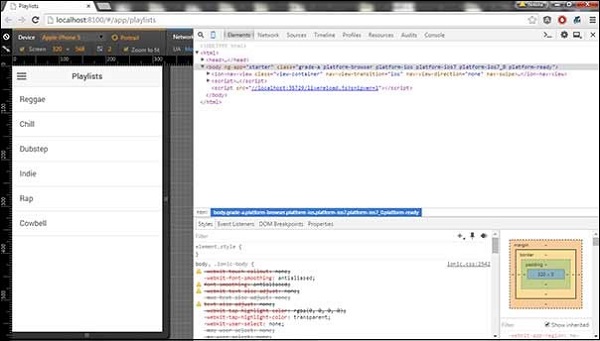


**Test App in Browser**

Since we are working with the JavaScript, you can serve your app on any web browser. This will speed up your building process, but you should always test your app on native emulators and devices. Type the following code to serve your app on the web browser.

C:\Users\Username\Desktop\myApp> ionic serve

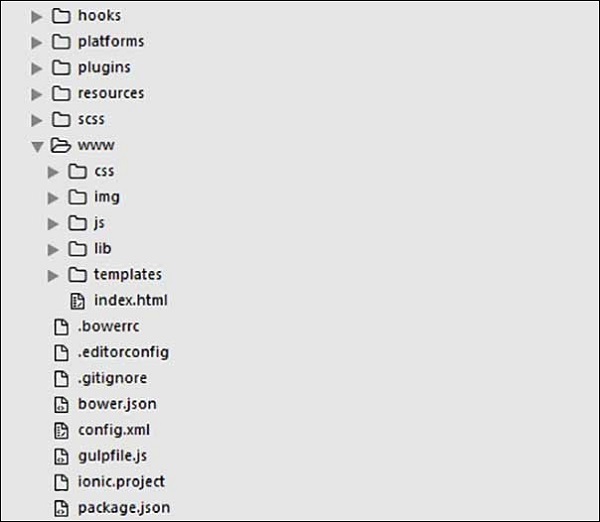
The above command will open your app in the web browser. Google Chrome provides the device mode functionality for mobile development testing. Press F12 to access the developer console.



The top left corner of the console window click has the "Toggle Device Mode" icon. The next step will be to click "Dock to Right" icon in the top right corner. Once the page is refreshed, you should be ready for testing on the web browser.

**Project Folder Structure**

Ionic creates the following directory structure for all type of apps. This is important for any Ionic developer to understand the purpose of every directory and the files mentioned below −



Let us have a quick understanding of all the folders and files available in the project folder structure shown in the image above.

**hooks** − Hooks are scripts that can be triggered during the build process. They are usually used for the Cordova commands customization and for building automated processes. We will not use this folder during this tutorial.

**platforms** − This is the folder where Android and IOS projects are created. You might encounter some platform specific problems during development that will require these files, but you should leave them intact most of the time.

**plugins** − This folder contains Cordova plugins. When you initially create an Ionic app, some of the plugins will be installed.

**resources** − This folder is used for adding resources like icon and splash screen to your project.

**scss** − Since Ionic core is built with Sass, this is the folder where your Sass file is located. For simplifying the process, we will not use Sass for this tutorial. Our styling will be done using CSS.

**www** − www is the main working folder for Ionic developers. They spend most of their time here. Ionic gives us their default folder structure inside 'www', but the developers can always change it to accommodate their own needs. When this folder is opened, you will find the following sub-folders −

The **css** folder, where you will write your CSS styling.

The **img** folder for storing images.

The **js** folder that contains the apps main configuration file (app.js) and AngularJS components (controllers, services, directives). All your JavaScript code will be inside these folders.

The **libs** folder, where your libraries will be placed.

The **templates** folder for your HTML files.

**Index**.html as a starting point to your app.

Other Files − Since this is a beginner’s tutorial, we will just mention some of the other important files and their purposes as well.

.**bowerrc** is the bower configuration file.

.**editorconfig** is the editor configuration file.

.**gitignore** is used to instruct which part of the app should be ignored when you want to push your app to the Git repository.

**bower**.**json** will contain the bower components and dependencies, if you choose to use the bower package manager.

**gulpfile**.js is used for creating automated tasks using the gulp task manager.

**config**.xml is the Cordova configuration file.

**package**.json contains the information about the apps, their dependencies and plugins that are installed using the NPM package manager.

**Ionic Color Classes**

Ionic framework gives us a set of nine predefined color classes. You can use these colors or you can override it with your own styling.

The following table shows the default set of nine colors provided by Ionic. We will use these colors for styling different Ionic elements in this tutorial. For now, you can check all the colors as shown below −

|  |  |  |
| --- | --- | --- |
| Class | Description | Result |
| Light | To be used for white color |  |
| Stable | To be used for light grey color |  |
| positive | To be used for blue color |  |
| Calm | To be used for light blue color |  |
| balanced | To be used for green color |  |
| energized | To be used for yellow color |  |
| assertive | To be used for red color |  |
| Royal | To be used for violet color |  |
| Dark | To be used for black color |  |

* **Button**

<!-- Colors -->

<button ion-button>Default</button>

<button ion-button color="secondary">Secondary</button>

<button ion-button color="danger">Danger</button>

<button ion-button color="light">Light</button>

<button ion-button color="dark">Dark</button>

<!-- Shapes -->

<button ion-button full>Full Button</button>

<button ion-button block>Block Button</button>

<button ion-button round>Round Button</button>

<!-- Outline -->

<button ion-button full outline>Outline + Full</button>

<button ion-button block outline>Outline + Block</button>

<button ion-button round outline>Outline + Round</button>

<!-- Icons -->

<button ion-button icon-start>

<ion-icon name="star"></ion-icon>

Left Icon

</button>

<button ion-button icon-end>

Right Icon

<ion-icon name="star"></ion-icon>

</button>

<button ion-button icon-only>

<ion-icon name="star"></ion-icon>

</button>

<!-- Sizes -->

<button ion-button large>Large</button>

<button ion-button>Default</button>

<button ion-button small>Small</button>

| Attr | Type | Details |
| --- | --- | --- |
| block | boolean | If true, activates a button style that fills the available width. |
| clear | boolean | If true, activates a transparent button style without a border. |
| color | string | The color to use from your Sass $colors map. Default options are: "primary", "secondary", "danger", "light", and "dark". For more information, see [Theming your App](https://ionicframework.com/docs/theming/theming-your-app). |
| default | boolean | If true, activates the default button size. Normally the default, useful for buttons in an item. |
| full | boolean | If true, activates a button style that fills the available width without a left and right border. |
| large | boolean | If true, activates the large button size. |
| mode | string | The mode determines which platform styles to use. Possible values are: "ios", "md", or "wp". For more information, see [Platform Styles](https://ionicframework.com/docs/theming/platform-specific-styles). |
| outline | boolean | If true, activates a transparent button style with a border. |
| round | boolean | If true, activates a button with rounded corners. |
| small | boolean | If true, activates the small button size. |
| solid | boolean | If true, activates a solid button style. Normally the default, useful for buttons in a toolbar. |
| strong | boolean | If true, activates a button with a heavier font weight. |

* **Checkbox (ion-checkbox)**

A checkbox is an input component that holds a boolean value. Checkboxes are no different than HTML checkbox inputs. However, like other Ionic components, checkboxes are styled differently on each platform. Use the checked attribute to set the default value, and the disabled attribute to disable the user from changing the value.

The Checkbox is a simple component styled based on the mode. It can be placed in an ion-item or used as a stand-alone checkbox.

<ion-list>

<ion-item>

<ion-label>Pepperoni</ion-label>

<ion-checkbox [(ngModel)]="pepperoni"></ion-checkbox>

</ion-item>

<ion-item>

<ion-label>Sausage</ion-label>

<ion-checkbox [(ngModel)]="sausage" disabled="true"></ion-checkbox>

</ion-item>

<ion-item>

<ion-label>Mushrooms</ion-label>

<ion-checkbox [(ngModel)]="mushrooms"></ion-checkbox>

</ion-item>

</ion-list>

<!-- Call function when state changes -->

<ion-list>

<ion-item>

<ion-label>Cucumber</ion-label>

<ion-checkbox [(ngModel)]="cucumber" (ionChange)="updateCucumber()"></ion-checkbox>

</ion-item>

</ion-list>

<ion-item>

<ion-label>Daenerys Targaryen</ion-label>

<ion-checkbox color="dark" checked="true"></ion-checkbox>

</ion-item>

<ion-item>

<ion-label>Arya Stark</ion-label>

<ion-checkbox disabled="true"></ion-checkbox>

</ion-item>

| Attr | Type | Details |
| --- | --- | --- |
| checked | boolean | If true, the element is selected. |

* **RadioButton (ion-radio)**

Like the [checkbox](https://ionicframework.com/docs/components/#checkbox), a radio is an input component that holds a boolean value. Under the hood, radios are no different than HTML radio inputs. However, like other Ionic components, radios are styled differently on each platform. Unlike checkboxes, radio components form a group, where only one radio can be selected at a time. Use the checked attribute to set the default value, and the disabled attribute to disable the user from changing to that value.

A radio button is a button that can be either checked or unchecked. A user can tap the button to check or uncheck it. It can also be checked from the template using the checked property.

Use an element with a radio-group attribute to group a set of radio buttons. When radio buttons are inside a [radio group](https://ionicframework.com/docs/api/components/radio/RadioGroup), exactly one radio button in the group can be checked at any time. If a radio button is not placed in a group, they will all have the ability to be checked at the same time.

**RadioGroup**

A radio group is a group of [radio buttons](https://ionicframework.com/docs/api/components/radio/RadioButton). It allows a user to select at most one radio button from a set. Checking one radio button that belongs to a radio group unchecks any previous checked radio button within the same group.

**Ex-1**

<ion-list radio-group>

<ion-list-header>

Language

</ion-list-header>

<ion-item>

<ion-label>Go</ion-label>

<ion-radio checked="true" value="go"></ion-radio>

</ion-item>

<ion-item>

<ion-label>Rust</ion-label>

<ion-radio value="rust"></ion-radio>

</ion-item>

<ion-item>

<ion-label>Python</ion-label>

<ion-radio value="python" disabled="true"></ion-radio>

</ion-item>

</ion-list>

**Ex-2**

<ion-list radio-group [(ngModel)]="relationship">

<ion-item>

<ion-label>Friends</ion-label>

<ion-radio value="friends" checked></ion-radio>

</ion-item>

<ion-item>

<ion-label>Family</ion-label>

<ion-radio value="family"></ion-radio>

</ion-item>

<ion-item>

<ion-label>Enemies</ion-label>

<ion-radio value="enemies" [disabled]="isDisabled"></ion-radio>

</ion-item>

</ion-list>

**Ex-3**

<ion-list radio-group [(ngModel)]="autoManufacturers">

<ion-list-header>

Auto Manufacturers

</ion-list-header>

<ion-item>

<ion-label>Cord</ion-label>

<ion-radio value="cord"></ion-radio>

</ion-item>

<ion-item>

<ion-label>Duesenberg</ion-label>

<ion-radio value="duesenberg"></ion-radio>

</ion-item>

<ion-item>

<ion-label>Hudson</ion-label>

<ion-radio value="hudson"></ion-radio>

</ion-item>

<ion-item>

<ion-label>Packard</ion-label>

<ion-radio value="packard"></ion-radio>

</ion-item>

<ion-item>

<ion-label>Studebaker</ion-label>

<ion-radio value="studebaker"></ion-radio>

</ion-item>

</ion-list>

| Attr | Type | Details |
| --- | --- | --- |
| checked | boolean | If true, the element is selected, and other buttons in the group are unselected. |
| color | string | The color to use from your Sass $colors map. Default options are: "primary", "secondary", "danger", "light", and "dark". For more information, see [Theming your App](https://ionicframework.com/docs/theming/theming-your-app). |
| disabled | boolean | If true, the user cannot interact with this element. |
| value | any | The value of the radio button. Defaults to the generated id. |

|  |  |  |
| --- | --- | --- |
| disabled | boolean | If true, the user cannot interact with any of the buttons in the group. |

* **Range**

A Range is a control that lets users select from a range of values by moving a slider knob along the slider bar or track.

**ion-range**

The Range slider lets users select from a range of values by moving the slider knob. It can accept dual knobs, but by default one knob controls the value of the range.

**[Range Labels](https://ionicframework.com/docs/api/components/range/Range/" \l "range-labels)**

Labels can be placed on either side of the range by adding the range-left or range-right property to the element. The element doesn't have to be an ion-label, it can be added to any element to place it to the left or right of the range. See [usage](https://ionicframework.com/docs/api/components/range/Range/#usage) below for examples.

**[Minimum and Maximum Values](https://ionicframework.com/docs/api/components/range/Range/" \l "minimum-and-maximum-values)**

Minimum and maximum values can be passed to the range through the min and max properties, respectively. By default, the range sets the min to 0 and the max to 100.

**[Steps and Snaps](https://ionicframework.com/docs/api/components/range/Range/" \l "steps-and-snaps)**

The step property specifies the value granularity of the range's value. It can be useful to set the step when the value isn't in increments of 1. Setting the step property will show tick marks on the range for each step. The snaps property can be set to automatically move the knob to the nearest tick mark based on the step property value.

**[Dual Knobs](https://ionicframework.com/docs/api/components/range/Range/" \l "dual-knobs)**

Setting the dualKnobs property to true on the range component will enable two knobs on the range. If the range has two knobs, the value will be an object containing two properties: lower and upper.

**Ex-1**

<ion-item>

<ion-range [(ngModel)]="brightness">

<ion-icon range-left small name="sunny"></ion-icon>

<ion-icon range-right name="sunny"></ion-icon>

</ion-range>

</ion-item>

**Ex-2**

<ion-list>

<ion-item>

<ion-range [(ngModel)]="singleValue" color="danger" pin="true"></ion-range>

</ion-item>

<ion-item>

<ion-range min="-200" max="200" [(ngModel)]="saturation" color="secondary">

<ion-label range-left>-200</ion-label>

<ion-label range-right>200</ion-label>

</ion-range>

</ion-item>

<ion-item>

<ion-range min="20" max="80" step="2" [(ngModel)]="brightness">

<ion-icon small range-left name="sunny"></ion-icon>

<ion-icon range-right name="sunny"></ion-icon>

</ion-range>

</ion-item>

<ion-item>

<ion-label>step=100, snaps, </ion-label>

<ion-range min="1000" max="2000" step="100" snaps="true" color="secondary" [(ngModel)]="singleValue4"></ion-range>

</ion-item>

<ion-item>

<ion-label>dual, step=3, snaps, </ion-label>

<ion-range dualKnobs="true" [(ngModel)]="dualValue2" min="21" max="72" step="3" snaps="true"></ion-range>

</ion-item>

</ion-list>

[**ratio**](https://ionicframework.com/docs/api/components/range/Range/#ratio)

Returns the ratio of the knob’s is current location, which is a number between 0 and 1. If two knobs are used, this property represents the lower value.

* **Select (ion-select)**

The ion-select component is similar to an HTML <select> element, however, Ionic’s select component makes it easier for users to sort through and select the preferred option. When users tap the select component, a dialog will appear with all of the options in a large, easy to select list.

The select component takes child ion-option components. If ion-option is not given a value attribute then it will use its text as the value.

If ngModel is bound to ion-select, the selected value will be based on the bound value of the model. Otherwise, the selected attribute can be used on ion-option components.

By default, the ion-select uses the [AlertController API](https://ionicframework.com/docs/api/components/alert/AlertController) to open up the overlay of options in an alert. The interface can be changed to use the [ActionSheetController API](https://ionicframework.com/docs/api/components/action-sheet/ActionSheetController) or [PopoverController API](https://ionicframework.com/docs/api/components/popover/PopoverController) by passing action-sheet or popover, respectively, to the interface property. Read on to the other sections for the limitations of the different interfaces.

<ion-list>

<ion-item>

<ion-label>Gaming</ion-label>

<ion-select [(ngModel)]="gaming">

<ion-option value="nes">NES</ion-option>

<ion-option value="n64">Nintendo64</ion-option>

<ion-option value="ps">PlayStation</ion-option>

<ion-option value="genesis">Sega Genesis</ion-option>

<ion-option value="saturn">Sega Saturn</ion-option>

<ion-option value="snes">SNES</ion-option>

</ion-select>

</ion-item>

</ion-list>

multiple selections can be made with <ion-select> by adding multiple="true" to the component.

<ion-list>

<ion-item>

<ion-label>Toppings</ion-label>

<ion-select [(ngModel)]="toppings" multiple="true" cancelText="Nah" okText="Okay!">

<ion-option value="bacon" selected="true">Bacon</ion-option>

<ion-option value="olives">Black Olives</ion-option>

<ion-option value="xcheese" selected="true">Extra Cheese</ion-option>

<ion-option value="peppers">Green Peppers</ion-option>

<ion-option value="mushrooms">Mushrooms</ion-option>

<ion-option value="onions">Onions</ion-option>

<ion-option value="pepperoni">Pepperoni</ion-option>

<ion-option value="pineapple">Pineapple</ion-option>

<ion-option value="sausage">Sausage</ion-option>

<ion-option value="Spinach">Spinach</ion-option>

</ion-select>

</ion-item>

</ion-list>

[**Single Value: Radio Buttons**](https://ionicframework.com/docs/api/components/select/Select/#single-value-radio-buttons)

The standard ion-select component allows the user to select only one option. When selecting only one option the alert interface presents users with a radio button styled list of options. The action sheet interface can only be used with a single value select. If the number of options exceed 6, it will use the alert interface even if action-sheet is passed. The ion-select component's value receives the value of the selected option's value.

<ion-item>

<ion-label>Gender</ion-label>

<ion-select [(ngModel)]="gender">

<ion-option value="f">Female</ion-option>

<ion-option value="m">Male</ion-option>

</ion-select>

</ion-item>

**[Multiple Value: Checkboxes](https://ionicframework.com/docs/api/components/select/Select/" \l "multiple-value-checkboxes)**

By adding the multiple="true" attribute to ion-select, users are able to select multiple options. When multiple options can be selected, the alert overlay presents users with a checkbox styled list of options. The ion-select multiple="true" component's value receives an array of all the selected option values. In the example below, because each option is not given a value, then it'll use its text as the value instead.

Note: the action-sheet and popover interfaces will not work with a multi-value select.

<ion-item>

<ion-label>Toppings</ion-label>

<ion-select [(ngModel)]="toppings" multiple="true">

<ion-option>Bacon</ion-option>

<ion-option>Black Olives</ion-option>

<ion-option>Extra Cheese</ion-option>

<ion-option>Mushrooms</ion-option>

<ion-option>Pepperoni</ion-option>

<ion-option>Sausage</ion-option>

</ion-select>

</ion-item>

**[Select Buttons](https://ionicframework.com/docs/api/components/select/Select/" \l "select-buttons)**

By default, the two buttons read Cancel and OK. Each button's text can be customized using the cancelText and okText attributes:

<ion-select okText="Okay" cancelText="Dismiss">

...

</ion-select>

The action-sheet and popover interfaces do not have an OK button, clicking on any of the options will automatically close the overlay and select that value.

[**Select Options**](https://ionicframework.com/docs/api/components/select/Select/#select-options)

Since ion-select uses the Alert, Action Sheet and Popover interfaces, options can be passed to these components through the selectOptions property. This can be used to pass a custom title, subtitle, css class, and more. See the [AlertController API docs](https://ionicframework.com/docs/api/components/alert/AlertController/#create), [ActionSheetController API docs](https://ionicframework.com/docs/api/components/action-sheet/ActionSheetController/#create), and [PopoverController API docs](https://ionicframework.com/docs/api/components/popover/PopoverController/#create) for the properties that each interface accepts.

For example, to change the mode of the overlay, pass it into selectOptions.

<ion-select [selectOptions]="selectOptions">

...

</ion-select>

this.selectOptions = {

title: 'Pizza Toppings',

subTitle: 'Select your toppings',

mode: 'md'

};

**[Object Value References](https://ionicframework.com/docs/api/components/select/Select/" \l "object-value-references)**

When using objects for select values, it is possible for the identities of these objects to change if they are coming from a server or database, while the selected value's identity remains the same. For example, this can occur when an existing record with the desired object value is loaded into the select, but the newly retrieved select options now have different identities. This will result in the select appearing to have no value at all, even though the original selection in still intact. Using the compareWith Input is the solution to this problem

<ion-item>

<ion-label>Employee</ion-label>

<ion-select [(ngModel)]="employee" [compareWith]="compareFn">

<ion-option \*ngFor="let employee of employees" [value]="employee"></ion-option>

</ion-select>

</ion-item>

compareFn(e1: Employee, e2: Employee): boolean {

return e1 && e2 ? e1.id === e2.id : e1 === e2;

}

| Attr | Type | Details |
| --- | --- | --- |
| debounce | number | How long, in milliseconds, to wait to trigger the ionChange event after each change in the range value. Default 0. |
| dualKnobs | boolean | Show two knobs. Defaults to false. |
| max | number | Maximum integer value of the range. Defaults to 100. |
| min | number | Minimum integer value of the range. Defaults to 0. |
| pin | boolean | If true, a pin with integer value is shown when the knob is pressed. Defaults to false. |
| snaps | boolean | If true, the knob snaps to tick marks evenly spaced based on the step property value. Defaults to false. |
| step | number | Specifies the value granularity. Defaults to 1. |

| Attr | Type | Details |
| --- | --- | --- |
| cancelText | string | The text to display on the cancel button. Default: Cancel. |
| compareWith | Function | The function that will be called to compare object values |
| interface | string | The interface the select should use: action-sheet, popover or alert. Default: alert. |
| multiple | boolean | If true, the element can accept multiple values. |
| okText | string | The text to display on the ok button. Default: OK. |
| placeholder | string | The text to display when the select is empty. |
| selectOptions | any | Any additional options that the alert or action-sheet interface can take. See the [AlertController API docs](https://ionicframework.com/docs/api/components/alert/AlertController/#create) and the [ActionSheetController API docs](https://ionicframework.com/docs/api/components/action-sheet/ActionSheetController/#create) for the create options for each interface. |
| selectedText | string | The text to display instead of the selected option's value. |

* **Tab (**ion-tab)

The Tab component, written <ion-tab>, is styled based on the mode and should be used in conjunction with the [Tabs](https://ionicframework.com/docs/api/components/tabs/Tabs/) component.

Each ion-tab is a declarative component for a [NavController](https://ionicframework.com/docs/api/navigation/NavController/). Basically, each tab is a NavController.

To add a basic tab, you can use the following markup where the root property is the page you want to load for that tab, tabTitle is the optional text to display on the tab, and tabIcon is the optional [icon](https://ionicframework.com/docs/api/components/icon/Icon/).

<ion-tabs>

<ion-tab [root]="chatRoot" tabTitle="Chat" tabIcon="chat"></ion-tab>

</ion-tabs>

Then, in your class you can set chatRoot to an imported class:

import { ChatPage } from '../chat/chat';

export class Tabs {

// here we'll set the property of chatRoot to

// the imported class of ChatPage

chatRoot = ChatPage;

constructor() {

}

}

You can also pass some parameters to the root page of the tab through rootParams. Below we pass chatParams to the Chat tab:

<ion-tabs>

<ion-tab [root]="chatRoot" [rootParams]="chatParams" tabTitle="Chat" tabIcon="chat"></ion-tab>

</ion-tabs>

export class Tabs {

chatRoot = ChatPage;

// set some user information on chatParams

chatParams = {

user1: 'admin',

user2: 'ionic'

};

constructor() {

}

}

And in ChatPage you can get the data from NavParams:

export class ChatPage {

constructor(navParams: NavParams) {

console.log('Passed params', navParams.data);

}

}

Sometimes you may want to call a method instead of navigating to a new page. You can use the (ionSelect) event to call a method on your class when the tab is selected. Below is an example of presenting a modal from one of the tabs.

<ion-tabs>

<ion-tab (ionSelect)="chat()" tabTitle="Show Modal"></ion-tab>

</ion-tabs>pop

export class Tabs {

constructor(public modalCtrl: ModalController) {

}

chat() {

let modal = this.modalCtrl.create(ChatPage);

modal.present();

}

}

| **Attr** | **Type** | **Details** |
| --- | --- | --- |
| enabled | boolean | If true, enable the tab. If false, the user cannot interact with this element. Default: true. |
| root | Page | Set the root page for this tab. |
| rootParams | object | Any nav-params to pass to the root page of this tab. |
| show | boolean | If true, the tab button is visible within the tabbar. Default: true. |
| tabBadge | string | The badge for the tab button. |
| tabBadgeStyle | string | The badge color for the tab button. |
| tabIcon | string | The icon for the tab button. |
| tabTitle | string | The title of the tab button. |
| tabUrlPath | string | The URL path name to represent this tab within the URL. |
| tabsHideOnSubPages | boolean | If true, hide the tabs on child pages. |

* **Tabs (ion-tabs)**

Tabs make it easy to navigate between different pages or functional aspects of an app. The Tabs component, written as <ion-tabs>, is a container of individual [Tab](https://ionicframework.com/docs/api/components/tabs/Tab/) components. Each individual ion-tab is a declarative component for a [NavController](https://ionicframework.com/docs/api/navigation/NavController/).

The position of the tabs relative to the content varies based on the mode. The tabs are placed at the bottom of the screen for iOS and Android, and at the top for Windows by default. The position can be configured using the tabsPlacement attribute on the <ion-tabs> component, or in an app's [config](https://ionicframework.com/docs/api/components/config/Config/). See the [Input Properties](https://ionicframework.com/docs/api/components/tabs/Tabs/#input-properties) below for the available values of tabsPlacement.

[Layout](https://ionicframework.com/docs/api/components/tabs/Tabs/#layout)

The layout for all of the tabs can be defined using the tabsLayout property. If the individual tab has a title and icon, the icons will show on top of the title by default. All tabs can be changed by setting the value of tabsLayout on the <ion-tabs> element, or in your app's [config](https://ionicframework.com/docs/api/components/config/Config/). For example, this is useful if you want to show tabs with a title only on Android, but show icons and a title for iOS. See the [Input Properties](https://ionicframework.com/docs/api/components/tabs/Tabs/#input-properties) below for the available values of tabsLayout.

[Selecting a Tab](https://ionicframework.com/docs/api/components/tabs/Tabs/" \l "selecting-a-tab)

There are different ways you can select a specific tab from the tabs component. You can use the selectedIndex property to set the index on the <ion-tabs> element, or you can call select() from the Tabs instance after creation. See [usage](https://ionicframework.com/docs/api/components/tabs/Tabs/#usage) below for more information.

You can add a basic tabs template to a @Component using the following template:

<ion-tabs>

<ion-tab [root]="tab1Root"></ion-tab>

<ion-tab [root]="tab2Root"></ion-tab>

<ion-tab [root]="tab3Root"></ion-tab>

</ion-tabs>

Where tab1Root, tab2Root, and tab3Root are each a page:

@Component({

templateUrl: 'build/pages/tabs/tabs.html'

})

export class TabsPage {

// this tells the tabs component which Pages

// should be each tab's root Page

tab1Root = Page1;

tab2Root = Page2;

tab3Root = Page3;

constructor() {

}

}

By default, the first tab will be selected upon navigation to the Tabs page. We can change the selected tab by using selectedIndex on the <ion-tabs> element:

<ion-tabs selectedIndex="2">

<ion-tab [root]="tab1Root"></ion-tab>

<ion-tab [root]="tab2Root"></ion-tab>

<ion-tab [root]="tab3Root"></ion-tab>

</ion-tabs>

Since the index starts at 0, this will select the 3rd tab which has root set to tab3Root. If you wanted to change it dynamically from your class, you could use [property binding](https://angular.io/docs/ts/latest/guide/template-syntax.html#!#property-binding).

Alternatively, you can grab the Tabs instance and call the select() method. This requires the <ion-tabs> element to have an id. For example, set the value of id to myTabs:

<ion-tabs #myTabs>

<ion-tab [root]="tab1Root"></ion-tab>

<ion-tab [root]="tab2Root"></ion-tab>

<ion-tab [root]="tab3Root"></ion-tab>

</ion-tabs>

Then in your class you can grab the Tabs instance and call select(), passing the index of the tab as the argument. Here we're grabbing the tabs by using ViewChild.

export class TabsPage {

@ViewChild('myTabs') tabRef: Tabs;

ionViewDidEnter() {

this.tabRef.select(2);

}

}

You can also switch tabs from a child component by calling select() on the parent view using the NavController instance. For example, assuming you have a TabsPage component, you could call the following from any of the child components to switch to TabsRoot3:

switchTabs() {

this.navCtrl.parent.select(2);

}

* **Toggle (ion-toggle)**

A toggle technically is the same thing as an HTML checkbox input, except it looks different and is easier to use on a touch device. Toggles can also have colors assigned to them, by adding any color attribute.

A toggle is an input component that holds a boolean value. Like the [checkbox](https://ionicframework.com/docs/components/#checkbox), toggles are often used to allow the user to switch a setting on or off. Attributes like value, disabled, and checked can be applied to the toggle to control its behavior.

**Ex-1**

<ion-list>

<ion-item>

<ion-label>Pepperoni</ion-label>

<ion-toggle [(ngModel)]="pepperoni"></ion-toggle>

</ion-item>

<ion-item>

<ion-label>Sausage</ion-label>

<ion-toggle [(ngModel)]="sausage" disabled="true"></ion-toggle>

</ion-item>

<ion-item>

<ion-label>Mushrooms</ion-label>

<ion-toggle [(ngModel)]="mushrooms"></ion-toggle>

</ion-item>

</ion-list>

**Ex-2**

<ion-item>

<ion-label> Sam</ion-label>

<ion-toggle disabled checked="false"></ion-toggle>

</ion-item>

| Attr | Type | Details |
| --- | --- | --- |
| checked | boolean | If true, the element is selected. |

* **Grid ion-grid, [ion-grid]**

The grid is a powerful mobile-first flexbox system for building custom layouts. It is heavily influenced by [Bootstrap's grid system](http://v4-alpha.getbootstrap.com/layout/grid/).

The grid is composed of three units — a grid, row(s) and column(s). Columns will expand to fill their row, and will resize to fit additional columns. It is based on a 12 column layout with different breakpoints based on the screen size. The number of columns and breakpoints can be fully customized using Sass.

**How it works**

The grid is a mobile-first system made up of any number of rows and columns. It is built with flexbox making it extremely responsive. The components that make up the grid can be written as an element (e.g., <ion-grid>) or added as an attribute to any element (e.g., <div ion-row>).

Grids act as a container for all rows and columns. Grids take up the full width of their container, but adding the fixed attribute will specify the width per screen size, see [grid size](https://ionicframework.com/docs/api/components/grid/Grid/#grid-size) below. Rows are horizontal groups of columns that line the columns up properly. Content should be placed within columns, and only columns may be immediate children of rows. Grid columns without a specified width will automatically have equal widths. For example, four instances of col-sm will each automatically be 25% wide for small breakpoints. Column attributes indicate the number of columns to use out of the default 12 per row. So, col-4 can be added in order to have three equal-width columns. Column widths are set as a percentage, so they’re always fluid and sized relative to their parent element. Columns have padding between individual columns, however, the padding can be removed from the grid and columns by adding no-padding on the grid.

There are five grid tiers by default, one for each responsive breakpoint: all breakpoints (extra small), small, medium, large, and extra large. Grid tiers are based on minimum widths, meaning they apply to their tier and all those larger than it (e.g., col-sm-4 applies to small, medium, large, and extra large devices). Grids can easily be customized via Sass variables. See [customizing the grid](https://ionicframework.com/docs/api/components/grid/Grid/#customizing-the-grid).

**Grid size**

By default, the grid will take up 100% width. To set a maximum width based on the screen size add the fixed attribute. The maximum width of the grid for each breakpoint is defined in the $grid-max-widths Sass variable. For more information, see [customizing the grid](https://ionicframework.com/docs/api/components/grid/Grid/#customizing-the-grid).

| Name | Value | Description |
| --- | --- | --- |
| xs | auto | Don't set the grid width for xs screens |
| sm | 540px | Set grid width to 540px when (min-width: 576px) |
| md | 720px | Set grid width to 720px when (min-width: 768px) |
| lg | 960px | Set grid width to 960px when (min-width: 992px) |
| xl | 1140px | Set grid width to 1140px when (min-width: 1200px) |

**Grid attributes**

The grid takes up full width and has padding added to it based on the screen size. There are two attributes that can be used to adjust this behavior.

| Property | Description |
| --- | --- |
| no-padding | Removes padding from the grid and immediate children columns. |
| fixed | Set a max width based on the screen size. |

**Default breakpoints**

The default breakpoints are defined by the $grid-breakpoints Sass variable. It can be customized to use different values for the breakpoint, rename and add/remove breakpoints. For more information, see [customizing the grid](https://ionicframework.com/docs/api/components/grid/Grid/#customizing-the-grid).

| Name | Value | Width Prefix | Offset Prefix | Push Prefix | Pull Prefix | Description |
| --- | --- | --- | --- | --- | --- | --- |
| xs | 0 | col- | offset- | push- | pull- | Set columns when (min-width: 0) |
| sm | 576px | col-sm- | offset-sm- | push-sm- | pull-sm- | Set columns when (min-width: 576px) |
| md | 768px | col-md- | offset-md- | push-md- | pull-md- | Set columns when (min-width: 768px) |
| lg | 992px | col-lg- | offset-lg- | push-lg- | pull-lg- | Set columns when (min-width: 992px) |
| xl | 1200px | col-xl- | offset-xl- | push-xl- | pull-xl- | Set columns when (min-width: 1200px) |

Note: the first breakpoint must have the value set to 0 and all breakpoint values must be in ascending order.

Auto-layout columns

[**Equal-width**](https://ionicframework.com/docs/api/components/grid/Grid/#equal-width)

By default, columns will take up equal width inside of a row for all devices and screen sizes.

<ion-grid>

<ion-row>

<ion-col>

1 of 2

</ion-col>

<ion-col>

2 of 2

</ion-col>

</ion-row>

<ion-row>

<ion-col>

1 of 3

</ion-col>

<ion-col>

2 of 3

</ion-col>

<ion-col>

3 of 3

</ion-col>

</ion-row>

</ion-grid>

**[Setting one column width](https://ionicframework.com/docs/api/components/grid/Grid/" \l "setting-one-column-width)**

Set the width of one column and the others will automatically resize around it. This can be done using our predefined grid attributes. In the example below, the other columns will resize no matter the width of the center column.

<ion-grid>

<ion-row>

<ion-col>

1 of 3

</ion-col>

<ion-col col-8>

2 of 3 (wider)

</ion-col>

<ion-col>

3 of 3

</ion-col>

</ion-row>

<ion-row>

<ion-col>

1 of 3

</ion-col>

<ion-col col-6>

2 of 3 (wider)

</ion-col>

<ion-col>

3 of 3

</ion-col>

</ion-row>

</ion-grid>

**[Variable-width](https://ionicframework.com/docs/api/components/grid/Grid/" \l "variable-width)**

Using the col-{breakpoint}-auto attributes, the column can size itself based on the natural width of its content. This is extremely useful for setting a column width using pixels. The columns next to the variable-width column will resize to fill the row.

<ion-grid>

<ion-row>

<ion-col>

1 of 3

</ion-col>

<ion-col col-auto>

Variable width content

</ion-col>

<ion-col>

3 of 3

</ion-col>

</ion-row>

<ion-row>

<ion-col>

1 of 4

</ion-col>

<ion-col>

2 of 4

</ion-col>

<ion-col col-auto>

<ion-input placeholder="Variable width input"></ion-input>

</ion-col>

<ion-col>

4 of 4

</ion-col>

</ion-row>

</ion-grid>

Responsive attributes

**[All breakpoints](https://ionicframework.com/docs/api/components/grid/Grid/" \l "all-breakpoints)**

To customize a column's width for all devices and screens, add the col-\* attribute. These attributes tell the column to take up \* columns out of the available columns.

<ion-grid>

<ion-row>

<ion-col col-4>

1 of 4

</ion-col>

<ion-col col-2>

2 of 4

</ion-col>

<ion-col col-2>

3 of 4

</ion-col>

<ion-col col-4>

4 of 4

</ion-col>

</ion-row>

</ion-grid>

**[Stacked to horizontal](https://ionicframework.com/docs/api/components/grid/Grid/" \l "-stacked-to-horizontal)**

Use a combination of width and breakpoint attributes to create a grid that starts out stacked on extra small screens before becoming horizontal on small screens.

<ion-grid>

<ion-row>

<ion-col col-12 col-sm>

1 of 4

</ion-col>

<ion-col col-12 col-sm>

2 of 4

</ion-col>

<ion-col col-12 col-sm>

3 of 4

</ion-col>

<ion-col col-12 col-sm>

4 of 4

</ion-col>

</ion-row>

</ion-grid>

Reordering

**[Offsetting columns](https://ionicframework.com/docs/api/components/grid/Grid/" \l "offsetting-columns)**

Move columns to the right by adding the offset-\* attributes. These attributes increase the margin start of the column by \* columns. For example, in the following grid the last column will be offset by 3 columns and take up 3 columns:

<ion-grid>

<ion-row>

<ion-col col-3>

1 of 2

</ion-col>

<ion-col col-3 offset-3>

2 of 2

</ion-col>

</ion-row>

</ion-grid>

Offsets can also be added based on screen breakpoints. Here's an example of a grid where the last column will be offset by 3 columns for md screens and up:

<ion-grid>

<ion-row>

<ion-col col-md-3>

1 of 3

</ion-col>

<ion-col col-md-3>

2 of 3

</ion-col>

<ion-col col-md-3 offset-md-3>

3 of 3

</ion-col>

</ion-row>

</ion-grid>

**[Push and pull](https://ionicframework.com/docs/api/components/grid/Grid/" \l "push-and-pull)**

Reorder the columns by adding the push-\* and pull-\* attributes. These attributes adjust the left and right of the columns by \* columns making it easy to reorder columns. For example, in the following grid the column with the 1st col description will actually be the last column and the 2nd col will be the first column.

<ion-grid>

<ion-row>

<ion-col col-9 push-3>

1 of 2

</ion-col>

<ion-col col-3 pull-9>

2 of 2

</ion-col>

</ion-row>

</ion-grid>

Push and pull can also be added based on screen breakpoints. In the following example, the column with the 3rd column description will actually be the first column for md screens and up:

<ion-grid>

<ion-row>

<ion-col col-md-6 push-md-3>

1 of 3

</ion-col>

<ion-col col-md-3 push-md-3>

2 of 3

</ion-col>

<ion-col col-md-3 pull-md-9>

3 of 3

</ion-col>

</ion-row>

</ion-grid>

Alignment

**[Vertical alignment](https://ionicframework.com/docs/api/components/grid/Grid/" \l "vertical-alignment)**

All columns can be vertically aligned inside of a row by adding different attributes to the row. For a list of available attributes, see [row attributes](https://ionicframework.com/docs/api/components/grid/Row#row-attributes).

<ion-grid>

<ion-row align-items-start>

<ion-col>

1 of 4

</ion-col>

<ion-col>

2 of 4

</ion-col>

<ion-col>

3 of 4

</ion-col>

<ion-col>

4 of 4 <br>#<br>#<br>#

</ion-col>

</ion-row>

<ion-row align-items-center>

<ion-col>

1 of 4

</ion-col>

<ion-col>

2 of 4

</ion-col>

<ion-col>

3 of 4

</ion-col>

<ion-col>

4 of 4 <br>#<br>#<br>#

</ion-col>

</ion-row>

<ion-row align-items-end>

<ion-col>

1 of 4

</ion-col>

<ion-col>

2 of 4

</ion-col>

<ion-col>

3 of 4

</ion-col>

<ion-col>

4 of 4 <br>#<br>#<br>#

</ion-col>

</ion-row>

</ion-grid>

Columns can also align themselves differently than other columns by adding the alignment attribute directly to the column. For a list of available attributes, see [column attributes](https://ionicframework.com/docs/api/components/grid/Col#column-attributes).

<ion-grid>

<ion-row>

<ion-col align-self-start>

<div>

1 of 4

</div>

</ion-col>

<ion-col align-self-center>

<div>

2 of 4

</div>

</ion-col>

<ion-col align-self-end>

<div>

3 of 4

</div>

</ion-col>

<ion-col>

<div>

4 of 4 <br>#<br>#<br>#

</div>

</ion-col>

</ion-row>

</ion-grid>

**[Horizontal alignment](https://ionicframework.com/docs/api/components/grid/Grid/" \l "horizontal-alignment)**

All columns can be horizontally aligned inside of a row by adding different attributes to the row. For a list of available attributes, see [row attributes](https://ionicframework.com/docs/api/components/grid/Row#row-attributes).

<ion-grid>

<ion-row justify-content-start>

<ion-col col-3>

1 of 2

</ion-col>

<ion-col col-3>

2 of 2

</ion-col>

</ion-row>

<ion-row justify-content-center>

<ion-col col-3>

1 of 2

</ion-col>

<ion-col col-3>

2 of 2

</ion-col>

</ion-row>

<ion-row justify-content-end>

<ion-col col-3>

1 of 2

</ion-col>

<ion-col col-3>

2 of 2

</ion-col>

</ion-row>

<ion-row justify-content-around>

<ion-col col-3>

1 of 2

</ion-col>

<ion-col col-3>

2 of 2

</ion-col>

</ion-row>

<ion-row justify-content-between>

<ion-col col-3>

1 of 2

</ion-col>

<ion-col col-3>

2 of 2

</ion-col>

</ion-row>

</ion-grid>

Customizing the grid

Using our built-in grid Sass variables and maps, it’s possible to completely customize the predefined grid attributes. Change the number of breakpoints, the media query values, the number of columns, and more.

**[Number of columns and padding](https://ionicframework.com/docs/api/components/grid/Grid/" \l "number-of-columns-and-padding)**

The number of grid columns and their padding can be modified via Sass variables. $grid-columns is used to generate the widths (in percent) of each individual column. $grid-padding-width is used for the padding on the grid, while $grid-padding-widths allows breakpoint-specific widths that are divided evenly across padding-left and padding-right as well as padding-top and padding-bottom of the grid and columns.

$grid-columns: 12 !default;

$grid-padding-width: 10px !default;

$grid-padding-widths: (

xs: $grid-padding-width,

sm: $grid-padding-width,

md: $grid-padding-width,

lg: $grid-padding-width,

xl: $grid-padding-width

) !default;

[Grid tiers](https://ionicframework.com/docs/api/components/grid/Grid/" \l "grid-tiers)

To customize the breakpoints and their values, override the values of $grid-breakpoints and $grid-max-widths. For example, to only use 3 breakpoints, the following could be written:

$grid-breakpoints: (

sm: 0,

md: 768px,

lg: 1024px

);

$grid-max-widths: (

sm: 420px,

md: 720px,

lg: 960px

);

* **Icon (ion-icon)**

Icons can be used on their own, or inside of a number of Ionic components. For a full list of available icons, check out the [Ionicons docs](https://ionicframework.com/docs/ionicons).

One feature of Ionicons in Ionic is when icon names are set, the actual icon which is rendered can change slightly depending on the mode the app is running from. For example, by setting the icon name of alarm, on iOS the icon will automatically apply ios-alarm, and on Material Design it will automatically apply md-alarm. This allows the developer to write the markup once while Ionic applies the appropriate icon based on the mode.

[Usage](https://ionicframework.com/docs/api/components/icon/Icon/#usage)

<!-- automatically uses the correct "star" icon depending on the mode -->

<ion-icon name="star"></ion-icon>

<!-- explicity set the icon for each mode -->

<ion-icon ios="ios-home" md="md-home"></ion-icon>

<!-- always use the same icon, no matter what the mode -->

<ion-icon name="ios-clock"></ion-icon>

<ion-icon name="logo-twitter"></ion-icon>

| Attr | Type | Details |
| --- | --- | --- |
| ios | string | Specifies which icon to use on ios mode. |
| isActive | boolean | If true, the icon is styled with an "active" appearance. An active icon is filled in, and an inactive icon is the outline of the icon. The isActive property is largely used by the tabbar. Only affects ios icons. |
| md | string | Specifies which icon to use on md mode. |
| name | string | Specifies which icon to use. The appropriate icon will be used based on the mode. For more information, see [Ionicons](https://ionicframework.com/docs/ionicons/). |

To use an icon, populate the name attribute on the ion-icon component:

<ion-icon name="heart"></ion-icon>

**Active / Inactive Icons**

All icons have both active and inactive states. Active icons are typically full and thick, where as inactive icons are outlined and thin. Set the isActive attribute to true or false to change the state of the icon. Icons will default to active if a value is not specified.

<ion-icon name="heart"></ion-icon> <!-- active -->

<ion-icon name="heart" isActive="false"></ion-icon> <!-- inactive -->

**Platform Specific Icons**

Many icons have both Material Design and iOS versions. Ionic will automatically use the correct version based on the platform.

To specify the icon to use for each platform, use the md and ios attributes and provide the platform specific icon name.

<ion-icon ios="logo-apple" md="logo-android"></ion-icon>

Variable Icons

To set an icon using a variable:

<ion-icon [name]="myIcon"></ion-icon>

export class MyFirstPage {

// use the home icon

myIcon: string = "home";

}