

An Introduction to Lists in Ionic 2

 joshmorony.com/an-introduction-to-lists-in-ionic-2/

Josh Morony

July 5, 2016

Lists are one of the most common interface elements in mobile applications. They are an efficient way to display lots of information in a small space and the act of scrolling through a list is basically second nature for most mobile users. Facebook uses a list for their news feed, as does Instagram and many others. All five apps I created for [Building Mobile Apps with Ionic 2](#) also use a list in one way or another.

Given the importance of lists, the Ionic team have put a lot of effort into creating an optimised list component that has smooth scrolling, inertia, acceleration and deceleration, and everything else that gives list that nice “native” feel. This is no easy feat, but since Ionic has done the hard yards for us we can do something as simple as this:

```
1 <ion-list>
2   <ion-item>I</ion-item>
3   <ion-item>am</ion-item>
4   <ion-item>a</ion-item>
5   <ion-item>list</ion-item>
6 </ion-list>
```

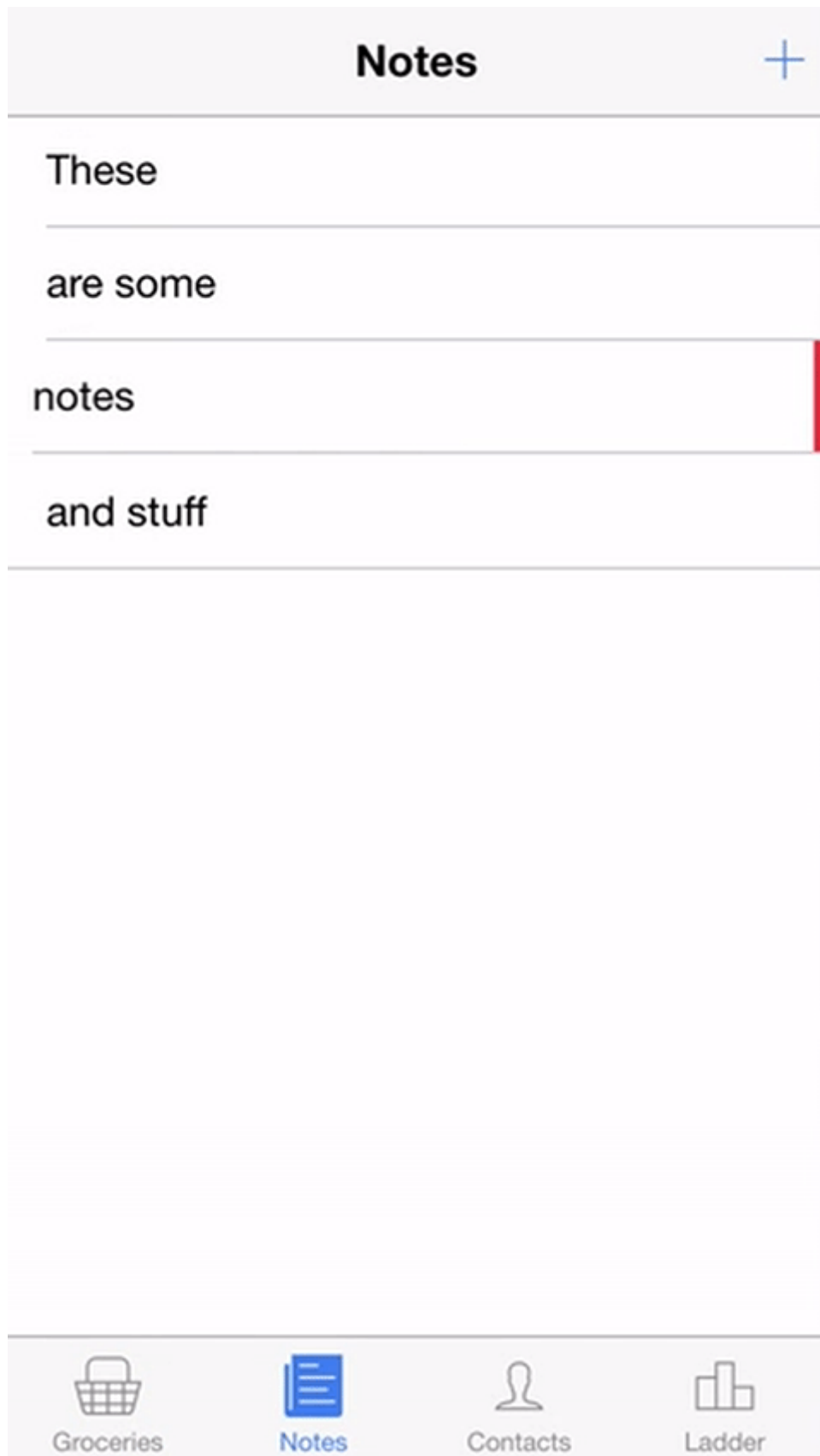


to create a list for our mobile apps. If you'd like to read a little more about the nitty gritty behind the scenes optimisations when it comes to lists, take a look at my more advanced article on [boosting scroll performance in Ionic 2](#).

It's pretty easy to create a basic list in Ionic, but in real life scenarios your implementation is likely going to require a little more work than the example I gave above. In this tutorial we are going to focus on going through the core concepts of how to use lists in Ionic 2, including:

- Creating a basic list with dynamic data
- Adding, deleting, and updating data in a list
- Creating a list with sliding actions
- Creating a list with grouping
- Reordering items in a list

We are going to do this by creating a little application with 4 different tabs, each one with a different style of list. At the end of this tutorial, the application will look something like this:



Before we Get Started

Before you go through this tutorial, you should have at least a basic understanding of Ionic 2 concepts. You must also already have Ionic 2 set up on your machine.

If you're not familiar with Ionic 2 already, I'd recommend reading my [Ionic 2 Beginners Guide](#) first to get up and running and understand the basic concepts. If you want a much more detailed guide for learning Ionic 2, then take a look at [Building Mobile Apps with Ionic 2](#).

Subscribe to my mailing list to get a **free 7 day Ionic 2 email course** and a **free preview** of my [Building Mobile Apps with Ionic 2](#) book.

I won't send you spam. Unsubscribe at any time. [Powered by ConvertKit](#)

Generate a New Ionic 2 Application

We're going to start off by generating a new Ionic 2 application. Since we are going to be using a tabbed layout, you might assume we would base our application the the tabs starter template rather than the blank template. I think the tabs starter is great to show you how tabs work if you are a beginner, but since it creates a bunch of pre-named pages I generally find it's easier to just start from a blank template and add the tabs yourself.

Run the following command to generate the application:

```
1 ionic start ionic2-lists blank --v2
```

Now we're going to generate the different pages we will be using as our tabs.

Run the following commands to generate the pages:

```
1 ionic g page Groceries
```

```
1 ionic g page Contacts
```

```
1 ionic g page Ladder
```

```
1 ionic g page Notes
```

Since we are starting from the blank template, we are going to have to set up the tabs now as well. All we have to do is import the pages into our **HomePage**, make them member variables so that we can reference them from the template, and then set up the tabs in the template.

*Modify **home.ts** to reflect*

```
1  import {Component} from '@angular/core';
2  import {NavController} from 'ionic-angular';
3  import {GroceriesPage} from '../groceries/groceries';
4  import {ContactsPage} from '../contacts/contacts';
5  import {LadderPage} from '../ladder/ladder';
6  import {NotesPage} from '../notes/notes';
7
8  @Component({
9    templateUrl: 'build/pages/home/home.html'
10 })
11 export class HomePage {
12
13     tab1Root: any = GroceriesPage;
14     tab2Root: any = NotesPage;
15     tab3Root: any = ContactsPage;
16     tab4Root: any = LadderPage;
17
18     constructor(private navCtrl: NavController) {
19
20     }
21
22 }
```

As you can see above, we've imported each of the pages and set up member variables for `tab1Root`, `tab2Root`, `tab3Root`, and `tab4Root`.

*Modify **home.html** to reflect the following:*

```
1  <ion-tabs>
2    <ion-tab [root]="tab1Root" tabTitle="Groceries" tabIcon="basket"></ion-tab>
3    <ion-tab [root]="tab2Root" tabTitle="Notes" tabIcon="paper"></ion-tab>
4    <ion-tab [root]="tab3Root" tabTitle="Contacts" tabIcon="person"></ion-tab>
5    <ion-tab [root]="tab4Root" tabTitle="Ladder" tabIcon="podium"></ion-tab>
6  </ion-tabs>
```

and now we're referencing those here in our tabs layout, along with a snazzy little icon to go along with it. Now let's get into building some lists!

Basic List

We're going to start off with a really simple list. We've already covered how to create a basic list:

```
1 <ion-list>
2   <ion-item>I</ion-item>
3   <ion-item>am</ion-item>
4   <ion-item>a</ion-item>
5   <ion-item>list</ion-item>
6 </ion-list>
```

but it's not often that you'll want to manually define a list like this. Typically, you'll have some data source in your class definition that is being used to populate the list. So let's do that with some dummy data.

*Modify **groceries.ts** to reflect the following:*

```
1 import { Component } from '@angular/core';
2 import { NavController } from 'ionic-angular';
3
4 @Component({
5   templateUrl: 'build/pages/groceries/groceries.html',
6 })
7 export class GroceriesPage {
8
9   groceries: any;
10
11   constructor(private nav: NavController) {
12
13     this.groceries = [
14       'Bread',
15       'Milk',
16       'Cheese',
17       'Snacks',
18       'Apples',
19       'Bananas',
20       'Peanut Butter',
21       'Chocolate',
22       'Avocada',
23       'Vegemite',
24       'Muffins',
25       'Paper towels'
26     ];
27
28   }
29
30 }
```

We've defined a member variable `groceries` that contains an array of shopping items. Any member variables we define like this will be available in that components template. Now we will make use of that data to populate the list automatically in the template.

Modify **groceries.html** to reflect the following:

```
1  <ion-header>
2
3    <ion-navbar>
4      <ion-title>Groceries</ion-title>
5    </ion-navbar>
6
7  </ion-header>
8
9
10 <ion-content>
11
12   <ion-list>
13     <ion-item *ngFor="let grocery of groceries">{{grocery}}</ion-item>
14   </ion-list>
15
16 </ion-content>
```

Now we're using the `*ngFor` directive to loop through every grocery item in our `groceries` array and stamp out an `<ion-item>` with each particular item. The result will look like this:

Groceries

Bread

Milk

Cheese

Snacks

Apples

Bananas

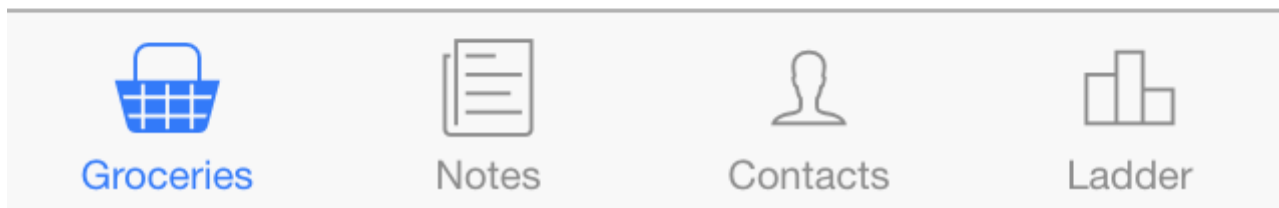
Peanut Butter

Chocolate

Avocada

Vegemite

Muffins



This is about as basic as it gets, but in a lot of cases this is all you will need to do. Maybe you have the data array being loaded in from some external source, or calculated in some way, but in the end it's just a matter of looping through those items and stamping out the items in the list.

Adding, Deleting and Updating Data in a List

There will probably come a time where you want the user to be able to modify the data in a list, so in this example we are going to take a look at how to do just that.


```
1  <ion-header>
2
3    <ion-navbar>
4      <ion-title>Notes</ion-title>
5      <ion-buttons end>
6        <button (click)="addNote()"><ion-icon name="add"></ion-icon></button>
7      </ion-buttons>
8    </ion-navbar>
9
10 </ion-header>
11
12 <ion-content>
13
14   <ion-list>
15
16     <ion-item-sliding *ngFor="let note of notes">
17
18       <ion-item>
19         {{note.title}}
20       </ion-item>
21
22       <ion-item-options>
23         <button (click)="editNote(note)" light>
24           <ion-icon name="paper"></ion-icon>
25         </button>
26         <button (click)="deleteNote(note)" danger>
27           <ion-icon name="trash"></ion-icon>
28         </button>
29       </ion-item-options>
30
31     </ion-item-sliding>
32
33   </ion-list>
34
35 </ion-content>
```

We're starting off with the template this time because I want to point out a few things. This is similar to our last list in that we are using the `*ngFor` directive and an `<ion-list>`, but this time we are using `<ion-item-sliding>`. This will create items in our list that we can swipe to reveal some buttons which are defined with `<ion-item-options>`. We use these buttons to attach an edit and delete button to each specific note.

We've also added a button in the navbar to add new notes to the list, and all of the functions we are calling through our `(click)` handlers we will define in our class now.

*Modify **notes.ts** to reflect the following:*

```
1  import { Component } from '@angular/core';
2  import { NavController, Alert } from 'ionic-angular';
```

```
3
4  @Component({
5    templateUrl: 'build/pages/notes/notes.html',
6  })
7  export class NotesPage {
8
9    notes: any = [];
10
11    constructor(private nav: NavController) {
12
13    }
14
15    addNote(){
16
17      let prompt = Alert.create({
18        title: 'Add Note',
19        inputs: [{
20          name: 'title'
21        }],
22        buttons: [
23          {
24            text: 'Cancel'
25          },
26          {
27            text: 'Add',
28            handler: data => {
29              this.notes.push(data);
30            }
31          }
32        ]
33      });
34
35      this.nav.present(prompt);
36    }
37
38    editNote(note){
39
40      let prompt = Alert.create({
41        title: 'Edit Note',
42        inputs: [{
43          name: 'title'
44        }],
45        buttons: [
46          {
47            text: 'Cancel'
48          },
49          {
50            text: 'Save',
51            handler: data => {
52              let index = this.notes.indexOf(note);
53            }
54          }
55        ]
56      });
57
58      this.nav.present(prompt);
59    }
60  }
61}
```

```
54             if(index > -1){
55                 this.notes[index] = data;
56             }
57         }
58     }
59 ]
60 });
61
62     this.nav.present(prompt);
63
64 }
65
66 deleteNote(note){
67
68     let index = this.notes.indexOf(note);
69
70     if(index > -1){
71         this.notes.splice(index, 1);
72     }
73 }
74
75 }
```

Things are starting to look a little more complicated now, so let's break it down. First of all, rather than defining any data for our list to use, we simply create an empty notes array as a member variable. This is because we are going to be adding notes through the app rather than manually defining them.

The rest of the class contains the various functions that our template calls: `addNote`, `editNote`, and `deleteNote`. The add and update functions both make use of the **Alert** component to grab data from the user. In the case of `addNote` it is simple as grabbing that data and pushing it into the notes array.

For the `editNote` function we pass in a reference to the note that was clicked. If you take a look at the template again, you will notice the edit button looks like this:

```
1  <button (click)="editNote(note)" light>
2      <ion-icon name="paper"></ion-icon>
3  </button>
```

This is inside of our `*ngFor` directive, so note here is a reference to the current note that is being stamped out. So this time when the user enters the note into the prompt, rather than pushing it into the array we find the existing note in the array and update that instead.

The same basic concept applies to `deleteNote` as well, except this time we find it and remove it from the array. Try creating some notes of your own and playing around with the edit and delete functions.

Notes

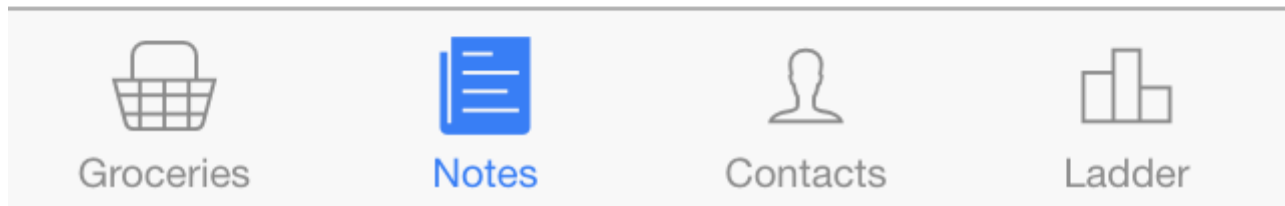


These

are some



and stuff



Reordering Lists

We've covered quite a lot so far but we still have some more tricks left in the bag. Another common feature in lists is the ability to reorder them by dragging items around. In a todo list or grocery list you might want to change the order of items based on their importance for example.

Once again, Ionic's got our backs with this as well with some in built functionality, and we're going to use some Australian Football League teams to help demonstrate how reordering works. Let's start off with our template again.

*Modify **ladder.html** to reflect the following.*

```
1  <ion-header>
2
3  <ion-navbar>
4    <ion-title>Ladder</ion-title>
5  </ion-navbar>
6
7 </ion-header>
8
9 <ion-content>
10
11   <ion-list reorder="true" (ionItemReorder)="reorderItems($event)">
12     <ion-item *ngFor="let team of ladder; let i = index;">{{i+1}}. {{team}}</ion-
13 item>
14   </ion-list>
15
16 </ion-content>
```

Again, we have a pretty similar looking list, but this time we set the `reorder` property to `true`, and we are also listening for the `ionItemReorder` event. Setting the `reorder` property to `true` will enable us to drag items around, but we need to reflect that change in our data as well. This is why we use the `ionItemReorder` event, it will pass through the original and the new order of indexes for our list.

Also note that I've added a bit of extra syntax to `*ngFor` here. We can also capture the current index of the loop by adding `let i = index`. This will allow us to display the numerical position of each item on the list (since it is a ladder), which will also change as we swap items around. When displaying the index we have to use `i+1` since the index starts at 0, and we want the first position to be 1.

*Modify **ladder.ts** to reflect the following:*

```
1  import { Component } from '@angular/core';
2  import {NavController, reorderArray} from 'ionic-angular';
3
4  @Component({
5    templateUrl: 'build/pages/ladder/ladder.html',
6  })
7  export class LadderPage {
8
9    ladder: any;
10
11    constructor(private nav: NavController) {
12
13      this.ladder = [
14        'Adelaide',
15        'Collingwood',
16        'Essendon',
17        'Hawthorn',
18        'Carlton',
19        'Sydney',
20        'Western Bulldogs',
21        'West Coast',
22        'Fremantle',
23        'North Melbourne',
24        'Richmond',
25        'Greater Western Sydney',
26        'St Kilda',
27        'Geelong',
28        'Brisbane',
29        'Melbourne',
30        'Port Adelaide',
31        'Gold Coast'
32      ];
33
34    }
35
36    reorderItems(indexes){
37      this.ladder = reorderArray(this.ladder, indexes);
38    }
39
40  }
```

This is actually a pretty simple class. We have our dummy data defined, and then the `reorderItems` function which is called whenever we receive the `ionItemReorder` event in the template. As I mentioned, we need to reflect the change in our data, and fortunately Ionic makes this super easy with their `reorderArray` function. This needs to be imported, but then you can simply call the function with your current array of data, and the indexes passed in from the reorder event, to update your data to reflect the new order.

Now you should be able to drag and drop to your hearts content:

Ladder

1. Adelaide

2. Hawthorn

3. Collingwood

4. West Coast

5. Sydney

6. Carlton


7. Essendon

8. Western Bulldogs


9. Fremantle

10. North Melbourne


11. Richmond




Groceries



Notes



Contacts



Ladder

Grouping Lists

Ionic also provides the ability to “group” list items, and we are going to have a bit of fun with that now. We will be recreating a typical contact style list where names are grouped alphabetically. We are going to do this dynamically

as well, so we will take an array filled with random data and get it displaying alphabetically and grouped automatically.

*Modify **contacts.html** to reflect the following:*

```
1  <ion-header>
2
3  <ion-navbar>
4    <ion-title>Contacts</ion-title>
5  </ion-navbar>
6
7 </ion-header>
8
9 <ion-content>
10
11   <ion-item-group *ngFor="let group of groupedContacts">
12
13     <ion-item-divider light>{{group.letter}}</ion-item-divider>
14     <ion-item *ngFor="let contact of group.contacts">{{contact}}</ion-item>
15
16   </ion-item-group>
17
18 </ion-content>
```

This is pretty straight forward, we've just introduced a couple new bits of syntax with `<ion-item-group>` which is used to group chunks of a list together, and `<ion-item-divider>` which provides a nice divider between the groups. In our case we want to have a divider that displays the letter for the contacts being grouped, and then display all of the contacts in that group. The tricky part is structuring our data to play nicely with this format.

*Modify **contacts.ts** to reflect the following:*

```
1  import { Component } from '@angular/core';
2  import { NavController } from 'ionic-angular';
3
4  @Component({
5    templateUrl: 'build/pages/contacts/contacts.html',
6  })
7  export class ContactsPage {
8
9    contacts;
10    groupedContacts = [];
11
12    constructor(private nav: NavController) {
13
14      this.contacts = [
15        'Kate Beckett',
```



```
16      'Richard Castle',
17      'Alexis Castle',
18      'Lanie Parish',
19      'Javier Esposito',
20      'Kevin Ryan',
21      'Martha Rodgers',
22      'Roy Montgomery',
23      'Jim Beckett',
24      'Stana Katic',
25      'Nathan Fillion',
26      'Molly Quinn',
27      'Tamala Jones',
28      'Jon Huertas',
29      'Seamus Dever',
30      'Susan Sullivan'
31  ];
32
33  this.groupContacts(this.contacts);
34
35  }
36
37  groupContacts(contacts){
38
39      let sortedContacts = contacts.sort();
40      let currentLetter = false;
41      let currentContacts = [];
42
43      sortedContacts.forEach((value, index) => {
44
45          if(value.charAt(0) !== currentLetter){
46
47              currentLetter = value.charAt(0);
48
49              let newGroup = {
50                  letter: currentLetter,
51                  contacts: []
52              };
53
54              currentContacts = newGroup.contacts;
55              this.groupedContacts.push(newGroup);
56
57          }
58
59          currentContacts.push(value);
60
61      });
62
63  }
64
65  }
```

We have our initial contacts data which is random and not sorted, but we need to group these alphabetically before we can display them in our list. So we create a `groupContacts` function which creates a new array for us to use. It loops through each of the contacts and creates a new “group” for each letter, and then each contact that starts with that letter is added to the group. The end result is an array that contains an object for each letter that is represented in the list, and those objects contain an array of all the contacts that fit in that group.

If you run that code now you should see something like this:

Contacts

A

Alexis Castle

J

Javier Esposito

Jim Beckett

Jon Huertas

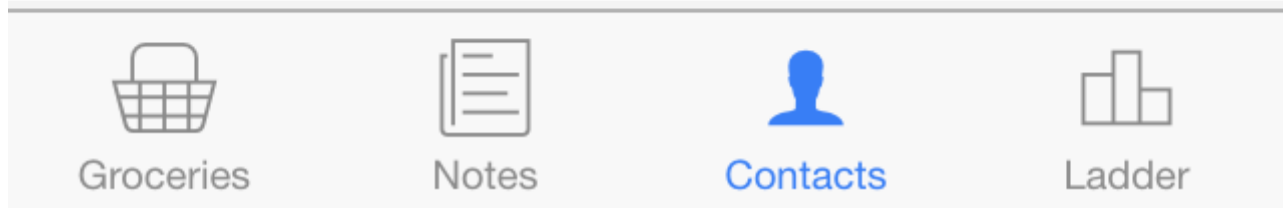
K

Kate Beckett

Kevin Ryan

L

Lanie Parish



BONUS CONTENT: Download the source code for this tutorial by entering your email address below:

Summary

Ionic 2 has great support for most things you would want to do with lists straight out of the box, and I haven't even covered all of that here. It's easy to put together a list by manually defining the data, but hopefully this tutorial has given you some insight on how you can use and manipulate lists in real life situations for a range of different scenarios.

What to read next...

- [Offline Syncing in Ionic 2 with PouchDB & CouchDB](#)
- [Advanced Forms & Validation in Ionic 2](#)
- [Building a Review App with Ionic 2, MongoDB & Node](#)
- [How to Unit Test an Ionic 2 Application](#)