Ionic Tutorial

Now that you have Ionic and its dependencies installed, you can build your first app! This section will guide you through the process of starting a new application, adding pages, navigating between those pages, and more. Let's get started!

> **Ionic uses TypeScript for its code. If you're unfamiliar with TypeScript, take a look at this page.**

Starting a New Ionic App

Starting a new app is easy! From your command line, run this command:

```
$ ionic start MyIonicProject tutorial
```

- start will tell the CLI create a new app.
- MyIonicProject will be the directory name and the app name from your project.
- tutorial will be the starter template for your project.

Along with creating your project, this will also install npm modules for the application, and get Cordova set up and ready to go.

If the tutorial template isn't something you want to use, Ionic has a few templates available:

- tabs : a simple 3 tab layout
- sidemenu: a layout with a swipable menu on the side
- blank: a bare starter with a single page
- super: starter project with over 14 ready to use page designs
- tutorial: a guided starter project

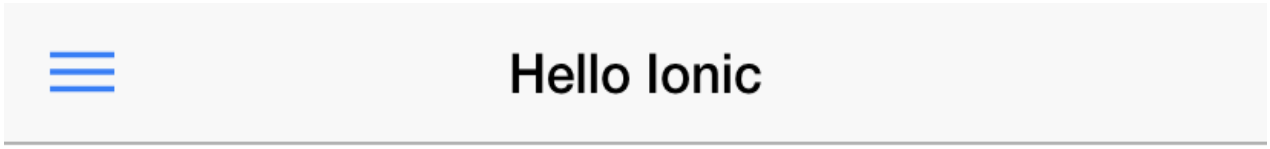
If you don't specify a template by running ionic start MyIonicProject, the tabs template will be used.

Viewing the app in a browser

Now, you can cd into the folder that was created. To get a quick preview of your app in the browser, use the serve command.

```
$ cd MyIonicProject/

$ ionic serve
```



You should see the welcome message shown above if everything was installed successfully.

In the next section, let's go over the project structure created by the ionic start command.

Project Structure

Let's walk through the anatomy of an Ionic app. Inside of the folder that was created, we have a typical Cordova project structure where we can install native plugins, and create platform-specific project files.

./src/index.html
src/index.html is the main entry point for the app, though its purpose is to set up script and CSS includes and bootstrap, or start running, our app. We won't spend much of our time in this file.

For your app to function, Ionic looks for the <ion-app> tag in your HTML. In this example we have:

```
<ion-app></ion-app>
```

And the following scripts near the bottom:

```
<script src="cordova.js"></script>
<script src="build/main.js"></script>
```

- build/main.js is a concatenated file containing Ionic, Angular and your app's JavaScript.
- cordova.js will 404 during local development, as it gets injected into your project during Cordova's build process.

./src/
Inside of the src directory we find our raw, uncompiled code. This is where most of the work for an Ionic app will take place. When we run ionic serve, our code inside

of src/ is transpiled into the correct Javascript version that the browser understands (currently, ES5). That means we can work at a higher level using TypeScript, but compile down to the older form of Javascript the browser needs.

src/app/app.module.ts is the entry point for our app.

Near the top of the file, we should see this:

```
@NgModule({
  declarations: [MyApp,HelloIonicPage, ItemDetailsPage, ListPage],
  imports: [ BrowserModule, IonicModule.forRoot(MyApp)],
  bootstrap: [IonicApp],
  entryComponents: [MyApp,HelloIonicPage,ItemDetailsPage,ListPage],
  providers: []
})
export class AppModule {}
```

Every app has a root module that essentially controls the rest of the application. This is very similar to ng-app from Ionic and Angular 1. This is also where we bootstrap our app using ionicBootstrap.

In this module, we're setting the root component to MyApp, in src/app/app.component.ts. This is the first component that gets loaded in our app, and typically it's an empty shell for other components to be loaded into. In app.component.ts, we're setting our template to src/app/app.html, so let's look in there.

./src/app/app.html
Here's the main template for the app in src/app/app.html:

```
<ion-nav id="nav" [root]="rootPage" #nav swipeBackEnabled="false"></ion-nav>
<ion-menu [content]="nav">
  <ion-header>
    <ion-toolbar>
      <ion-title>Pages</ion-title>
    </ion-toolbar>
  </ion-header>

  <ion-content>
    <ion-list>
```

```
      <button ion-item *ngFor="let p of pages" (click)="openPage(p)">
        {{p.title}}
      </button>
    </ion-list>
  </ion-content>
</ion-menu>
```

In this template, we set up an ion-menu to function as a side menu, and then an ion-nav component to act as the main content area. The ion-menu's [content] property is bound to the local variable nav from our ion-nav, so it knows where it should animate around.

Next let's see how to create more pages and perform basic navigation.

Adding Pages

Now that we have a basic understanding of the layout of an Ionic app, let's walk through the process of creating and navigating to pages in our app.

Taking a look at src/app/app.html, we see this line near the bottom:

```
<ion-nav [root]="rootPage" #content swipeBackEnabled="false"></ion-nav>
```

Pay attention to the [root] property binding. This sets what is essentially the first, or "root" page for the ion-nav component. When ion-nav loads, the component referenced by the variable rootPage will be the root page.

In src/app/app.component.ts, the MyApp component specifies this in its constructor:

```
...
import { HelloIonicPage } from '../pages/hello-ionic/hello-ionic';
...
export class MyApp {
  ...
  // make HelloIonicPage the root (or first) page
  rootPage: any = HelloIonicPage;
  pages: Array<{ title: string, component: any }>;
  constructor(private platform: Platform, private menu: MenuController, ...) {
    ...
```

```
    }

  ...

}
```

We see that `rootPage` is set to `HelloIonicPage`, so `HelloIonicPage` will be the first page loaded in the nav controller. Let's take a look at it.

Creating a Page

Next, let's check out the `HelloIonicPage` that we are importing. Inside the `src/pages/hello-ionic/` folder, go and open up `hello-ionic.ts`.

> **You may have noticed that each page has its own folder that is named after the page. Inside each folder, we also see a `.html` and a `.scss` file with the same name. For example, inside of `hello-ionic/` we will find `hello-ionic.ts`, `hello-ionic.html`, and `hello-ionic.scss`. Although using this pattern is not required, it can be helpful to keep things organized.**

Below, we see the `HelloIonicPage` class. This creates a Page - an Angular component with all Ionic directives already provided, to be loaded using Ionic's navigation system. Notice that because Pages are meant to be loaded dynamically, they don't need to have a selector. However, the selector is useful in order to override the default styles on a specific page (see `hello-ionic.scss`):

```
import { Component } from '@angular/core';

@Component({
  selector: 'page-hello-ionic',
  templateUrl: 'hello-ionic.html'
})
export class HelloIonicPage {
  constructor() {
  }
}
```

All pages have both a class, and an associated template that's being compiled as well. Let's check out `src/pages/hello-ionic/hello-ionic.html` - the template file for this page:

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
```

```
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Hello Ionic</ion-title>
  </ion-navbar>
</ion-header>


<ion-content padding>
  <h3>Welcome to your first Ionic app!</h3>
  <p>
    This starter project is our way of helping you get a functional app running in record time.
  </p>
  <p>
    Follow along on the tutorial section of the Ionic docs!
  </p>
  <p>
    <button ion-button color="primary" menuToggle>Toggle Menu</button>
  </p>
</ion-content>
```

The <ion-navbar> is a template for the navigation bar on this page. As we navigate to this page, the button and title of the navigation bar transition in as part of the page transition.

The rest of the template is standard Ionic code that sets up our content area and prints our welcome message.

Creating Additional Pages

To create an additional page, we don't need to do much beyond making sure we correctly configure the title and anything else we want the navigation bar to display.

Let's check out the contents of src/pages/list/list.ts. Inside, you will see a new page is defined:

```
import {Component} from "@angular/core";
import {NavController, NavParams} from 'ionic-angular';
import {ItemDetailsPage} from '../item-details/item-details';
@Component({
  templateUrl: 'list.html'
```

```
})
export class ListPage {
  selectedItem: any;
  icons: string[];
  items: Array<{ title: string, note: string, icon: string }>;
  constructor(public navCtrl: NavController, public navParams: NavParams) {
    // If we navigated to this page, we will have an item available as a nav param
    this.selectedItem = navParams.get('item');
    this.icons = ['flask', 'wifi', 'beer', 'football', 'basketball', 'paper-plane',
      'american-football', 'boat', 'bluetooth', 'build'];
    this.items = [];
    for (let i = 1; i < 11; i++) {
      this.items.push({
        title: 'Item ' + i,
        note: 'This is item #' + i,
        icon: this.icons[Math.floor(Math.random() * this.icons.length)]
      });
    }
  }
  itemTapped(event, item) {
    this.navCtrl.push(ItemDetailsPage, {
      item: item
    });
  }
}
```

This page will create a basic list page containing a number of items.

Overall, this page is very similar to the `HelloIonicPage` we saw earlier. In the next section, we will learn how to navigate to a new page!

Navigating to Pages

Recall from the previous section we had a function inside our ListPage class that looked something like this:

```
itemTapped(event, item) {
    this.navCtrl.push(ItemDetailsPage, {
      item: item
    });
}
```

You might have noticed we are referencing ItemDetailsPage. This is a page included in the tutorial starter. Let's import it in app/pages/list/list.ts so we can use it:

```
...
import { ItemDetailsPage } from '../pages/item-details/item-details';
```

After saving the file, you will notice the ionic serve process will recompile your app with the new changes, and reload the browser. Let's revisit our app in the browser, and when we tap an item, it will navigate to the item details page! Notice that the menu-toggle is replaced with a back button instead. This is a native style that Ionic follows, but can be configured.

How It Works

Navigation in Ionic works like a simple stack, where we push new pages onto the top of the stack, which takes us forwards in the app and shows a back button. To go backwards, we pop the top page off. Since we set this.navCtrl in the constructor, we can call this.navCtrl.push(), and pass it the page we want to navigate to. We can also pass it an object containing data we would like to pass to the page being navigated to. Using push to navigate to a new page is simple, but Ionic's navigation system is very flexible. Check out the navigation docs to see more advanced navigation examples.

> **When it comes to URLs, the latest Ionic works a bit differently than Ionic v1.x. Instead of using URLs to navigate, we use them to make sure we can always come back to a page (on app launch, for example). This means we aren't limited to using href to navigate around. However, we still have the option to use a URL to navigate to a page when necessary.**

Next Steps

Nice job! You've made it through the tutorial and are on your way towards Ionic mastery! If you're looking for an overview on what else is included with Ionic, check out the Component docs. To learn about using device APIs, head over to the Native section. If at any point you need help, check out our developer resources section, or ask a question on the forums.