Ionic 3 Lazy Loading

Ionic 3.0.0 introduces some optional changes to the file structure of your application in order to speed up your application. To get started with lazy loading in Ionic see the steps to upgrade section below.

Please create an issue on the Ionic repository if you find any problems using lazy loading or upgrading: https://github.com/driftyco/ionic/issues

Steps to Upgrade to Ionic 3

Follow the steps to upgrade under the 3.0.0 release:
https://github.com/driftyco/ionic/blob/master/CHANGELOG.md#300-2017-04-05

Lazy Loading Steps

This will improve your applications startup time, reduce the bundle size, and easily set up Ionic routing.

Important: the following steps will override any `DeepLinkConfig` defined in the `IonicModule.forRoot`. You should remove this config in favor of setting the config in the `IonicPage` decorator of each individual page.

> Let's take an app based on the blank starter template:
>
> Right now, our `app.module.ts` file has `HomePage` imported and declared in the `declarations` as well as the `entryComponents`
>
> ```
> @NgModule({
>  declarations: [
>  MyApp,
>  HomePage
>  ],
>  ---
>  entryComponents: [
>  MyApp,
>  HomePage

```
  ],
  ---
})
```

The goal is to reduce this so we're only loading the main `app.component.ts`, and lazy-loading the HomePage component everywhere else.

So we'll remove `HomePage` from the declarations, entryComponents, and remove the import statement as well.

In our `src/pages/home` directory, we should have something close to this.

```
└home
  ├─ home.html
  ├─ home.scss
  └─ home.ts
```

What we'll want to do is create a new file here, called `home.module.ts`, similar to our `app.module.ts`

```
//home.module.ts
import { NgModule } from '@angular/core';
import { HomePage} from './home’;
import { IonicPageModule } from 'ionic-angular';

@NgModule({
  declarations: [HomePage],
  imports: [IonicPageModule.forChild(HomePage)],
})
export class HomePageModule { }
```

Now, in our `home.ts` file, we can add the `@IonicPage` decorator to the HomePage class

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

```
import { IonicPage } from 'ionic-angular';

@IonicPage()
@Component({
    ---
})
export class HomePage {
```

Now the last bit of plumbing we'll need to do is back in our `app.component.ts`.

Since our HomePage component is now lazy loaded, we do not want to import it directly and reference it anywhere. Instead, we can pass a string that matches up with the component.

So This:

```
import { HomePage } from '../pages/home/home';
@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage:any = HomePage;
```

Would become:

```
@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage:any = 'HomePage';

```

During the build process, the deeplinks for the HomePage component will be generated that know how to handle that string.
The string is actually a reference to the `name` property of the `@IonicPage` decorator, which defaults to the class name as a string. If we change that name property to something else, we'll also need to update the reference we use elsewhere

```
//home.ts
import { Component } from '@angular/core';
import { IonicPage } from 'ionic-angular';

@IonicPage({
  name: 'home'
})
@Component({
    ---
})
export class HomePage {


// app.component.ts
export class MyApp {
  rootPage:any = 'home';

```

The `IonicPage` decorator supports several fields and options for passing data. Please review the documentation for more information: http://ionicframework.com/docs/v2/nightly/api/navigation/IonicPage/

This same concept can and should be applied to pages presented using the Modal or Popover components, since they are just Components.

Handling Components/Providers/Pipes

The changes above are geared towards pages in your apps, that is, full screen UI that users can navigate to. An app may also contain custom components for widgets, pipes, or providers that can load data. We suggest the following pattern for a more straightforward approach to load the additional pieces.

Components

One module that imports all component classes, no individual modules for each component
`components/`

- components.module.ts (ComponentsModule)

        - imports and exports all components the user creates
    - `component1/`
        - component1.ts
        - component1.html
    - `component2/`
        - component2.ts
        - component2.html

Creating a page using generators automatically imports ComponentsModule

Pipes

One module that imports all pipe classes, no individual modules for each pipe
`pipes/`

- pipes.module.ts (PipesModule)
    - imports and exports all pipes the user creates
- pipe1.ts
- pipe2.ts

Providers

No NgModule for providers, import each provider in main app NgModule
`providers/`

- provider1.ts
- provider2.ts

A lot of this work will be handled by the Ionic generators, so it will be automatic for the most part. If a developer wanted to modify this for more fine-grained control, the can do so without any issues. But things setup provides people with a very straightforward and easy path.