Lesson 1: Setup

In this lesson we discuss how to set up Ionic on your machine, how to generate a new Ionic application, and what the purpose of the different files and folders are.

Updates and Errors

- Ionic now includes its own error handler by default, so you may notice some difference in the **app.module.ts** file in your own projects. It is not required that you include this, but it does make error reporting nicer. The **app.module.ts** file now also needs to include the BrowserModule, and IonicStorageModule (if you are using Storage). The appropriate **app.module.ts** file for this project now looks like this:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, ErrorHandler } from '@angular/core';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';
import { HomePage } from '../pages/home/home';
import { EditTodo } from '../pages/edit-todo/edit-todo';
import { Data } from '../providers/data';
import { IonicStorageModule } from '@ionic/storage';

@NgModule({
  declarations: [
    MyApp,
    HomePage,
    EditTodo
  ],
  imports: [
    BrowserModule,
    IonicModule.forRoot(MyApp),
    IonicStorageModule.forRoot()
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    EditTodo
  ],
  providers: [{provide: ErrorHandler, useClass: IonicErrorHandler}, Data]
})
export class AppModule {}
```

Key Points

- To create a new project you can use the `ionic start blank MyApp --v2` command. You can replace `blank` with different values, including `tabs` and `sidemenu` depending on what type of project you want to create. You do not *have* to use these template. If you want to create a tabs style application, you can still just use the `blank` template and add tabs yourself.
- You don't need to worry about most of the files and folders in the generated project, the most important folder is `src` and that is where you will do most of your coding
- Do **not** edit code inside of the `www` folder. This folder is for the transpiled code that is automatically built by Ionic. If you make changes in this folder, they will be overwritten by new builds.
- The **app.scss** file can be used to apply styles globally to your application
- The **variables.scss** file can be used to set up SASS variables that can be used throughout your application
- The **app.component.ts** file contains the **root component** and it is used to set up the **root page** for your application

- Any pages or providers you create need to be added to the **app.module.ts** file
- Each page in your application will have it's own folder, which will consist of a **.ts** file for the class, a **.html** file for the template, and a **.scss** file for styling

Resources

- Ionic Installation Guide

Lesson 2: Getting Ready

In this lesson we use the generate commands to set up the pages and providers our application needs.

Updates and Errors

- When running `ionic g page EditTodo` it now auto generates with lazy loading by default. For this simple tutorial, you should remove the @IonicPage decorator, and delete the edit-todo.module.ts file. We cover what lazy loading is and how to use it properly in the book.

Key Points

- The `ionic g` command can be used to generate pages with `ionic g page MyPage` and providers with `ionic g provider MyProvider`
- For a full list of things you can generated with `ionic g` run `ionic g --list`
- After creating a page or provider, it must be added to the **app.module.ts** file
- If there are any errors in your application, they will be displayed in either the **browser console** or the **terminal / command prompt**

Lesson 3: Decorators

In this lesson we quick cover the concept of a **decorator**.

Key Points

- A decorator is a block of code above the class that defines some metadata, or extra information, about the class. This commonly includes the name of the `selector` for the component, and a `templateUrl` which defines where the template file can be found.
- The `@Component` decorator is used for components, and the `@Injectable` decorator is used for providers. There are also other decorators available such as `@Pipe` .
- The `@Component` decorator allows us to specify the `selector` for the component, and it links to the component's template file
- The decorators are the same for just about every class, and will rarely require any editing, so you can almost just ignore them.

Lesson 4: Classes

In this lesson we start implementing our class definitions, talk briefly about the role of a class, and how we can add dependencies, variables, and functions to the class.

Key Points

- A class is a concept from object oriented programming. It allows us to define a "blueprint" for creating "objects". In the case of Ionic, our class defines the behaviour for our page components.
- We can import other classes by using the `import` keyword, and we can make our own class available for importing elsewhere by using the `export` keyword.
- We can import our own classes, and we can also import from other existing libraries as well (like the Ionic and Angular libraries)
- A constructor function is a function that runs immediately and can be used to setup the class. It is also used to inject any dependencies into the class. We can inject a dependency through the constructor by suppling it with a paramter, and giving that parameter a type of whatever it is that we want to inject.
- The `ionViewDidLoad` function is similar to the `constructor` in that it runs immediately.
- A member variable can be added to the class by placing it above the constructor. Member variables are accessible anywhere throughout the class by using `this.variableName`. Member variables are also accessible from within a components template file.
- A normal variable, for example if we declared `let myVar` inside of a function, is only accessible within that function, not the entire class.
- Types define the type of data that can be stored on a variable. For example, a variable that can only be a string might look like this `let myString: string;`. You can use the `any` type to allow any type of data to be stored on a variable. Attempting to store the wrong type of data in a variable will result in a build error.
- We can add functions to our class that are accessible to other functions in our class, and also to our template.

Lesson 5: Templates

In this lesson we start implementing the template files for the home page and for the edit todo page.

Key Points

- Data defined in a member variable in the class can be accessed in the template for that component
- `<ion-content>` stores the main content for a page, the `<ion-header>` contains the navigation bar, and `<ion-footer>` can be used to add content at the bottom of the page
- Ionic provides a wide range of components like `<ion-list>` that we can use in templates. These components mimic the native UI of whatever platform the app is running on.
- The `*ngFor` directive can be used to repeat the element it is attached to in the template for every element that is present in an array of data
- Expressions can be redered out in the template using double curly braces `{{}}`. You could render out the value of a variable like this `{{todo.title}}` or you could even render out a mathematical expression like this `{{1+1}}`. Whatever is inside of the curly braces will be evaluated and then displayed.
- `(click)` listeners should only be added to `<button>` and `<a>` to avoid tap delays on mobile. If you must attach the click listener to anything else, you should also add `tappable` to the element.
- Using `start` or `end` with `<ion-buttons>` will place the button in the navbar in the position that is normal for whatever platform the app is running on.

- `[(ngModel)]` is used to set up two-way data binding between an input in the template and a member variable in the class. If the value is changed in the class, it will be immediately reflected in the template. If the value is changed in the template, it will be immediately reflected in the class.

Lesson 6: Navigation

In this lesson we set up navigation so that we can navigate between our two pages, as well as pass data between those pages.

Key Points

- The **NavController** can be used for navigation in the application. Usually we will inject it into the constructor and set up a reference to it using `navCtrl`.
- We can use `this.navCtrl.push(MyPage)` to push a a page onto the navigation stack (making it the current page), and we can use `this.navCtrl.pop()` to remove the current page from the navigation stack (which will take the user back to the previous page). You can call `pop` manually, or you can use the automatically generated back button to go back instead.
- **NavParams** can be used to pass data from one page to another. When pushing the page, we supply an object like this: `this.navCtrl.push(MyPage, {data: 'my data'});` and then we can grab that data in the page we pushed to by using `this.navParams.get('data');`.

Lesson 7: Saving Data

In this lesson we set up our data provider to save and retrieve data from storage.

Updates & Errors

- The way in which the storage module is included in Ionic has changed. It is now included as an `import` in **app.module.ts** instead of as a provider. Your **app.module.ts** file should look like this:

```
import { NgModule, ErrorHandler } from '@angular/core';
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
import { MyApp } from './app.component';
import { HomePage } from '../pages/home/home';
import { EditTodo } from '../pages/edit-todo/edit-todo';
import { Data } from '../providers/data';
import { IonicStorageModule } from '@ionic/storage';

@NgModule({
  declarations: [
    MyApp,
    HomePage,
    EditTodo
  ],
  imports: [
    IonicModule.forRoot(MyApp),
    IonicStorageModule.forRoot()
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    EditTodo
  ],
  providers: [{provide: ErrorHandler, useClass: IonicErrorHandler}, Data]
})
export class AppModule {}
```

Key Points

- Ionic provides a generic storage API called **Storage**. This will use the best storage available to the application, whether that is the browsers local storage, or a native SQLite database (if it is installed).
- Data can be saved using `set('myData', data)` and it can be retrieved using `get('myData')`.
- Promises can be used to handle asynchronous operations. Asynchronous operations are operations that may take some time to complete, and we don't want to block the application from running whilst we wait for the response. Loading data is a good example of an asynchronous operation, but not the only one.
- If the data provider is injected into the the home page, we can also access the data from the data provider directly in our home page's template.
- It's important to check data when you load it in from storage, if it is the first time the user is using the app the load may not return any data, and we need to make sure to handle that case properly.

Lesson 8: Styling

In this lesson we take our application from being boring and black and white, to something a little more pleasing to the eye.

Updates and Errors

- Occasionally the generators for pages are changed. To ensure that your styles are applied correctly, make sure that the `selector` in your **my-page.ts** file matches up with the selector you are using in your **my-page.scss** file.

Key Points

- We can use the `colors` defined in **theme/variables.scss** on elements by adding the `color="secondary"` attribute. Other color names include `primary`, `danger`, `light`, and `dark`, but you can also define our own custom colors if you wish. Using this method, rather than manually adding colors with CSS, makes the theme easier to change in future.
- Ionic provides some directives you can add to elements which will effect the styling, such as adding `no-lines` to a list, or `full` to a button (which will make it full width). These are for convenience, but you could also just apply your own CSS styles.
- The background color can be changed globally by changing the `$background-color: #fff;` variable in **theme/variables.scss**
- You can isolate CSS rules to a specific component, by adding the rules to that components `.scss` file **and** placing the rules inside of a selector that is the same as the components selector in the **@Component** decorator, e.g:

```
page-something {
    // rules go here
}
```

- If you do not place CSS rules inside of the components own selector, the rules will apply globally to the application
- You can also surround rules in the `.ios` or `.md` classes. This will allow you to specifiy which styles will apply to what platform, and it also increases the "specicifity" of your rules, meaning they likely won't be overridden by Ionic's own styling.

Lesson 9: Native Functionality

In this lesson we install the native SQLite plugin and talk a little bit about how Cordova plugins and Ionic Native work.

Key Points

- Cordova plugins provide access to Native APIs
- The SQLite plugin provides access to a native SQLite database, which has greater storage capacity and is persistent (i.e. your data won't be wiped)
- Ionic Native is a library provided by Ionic that wraps Cordova plugins, this makes them much easier and nicer to use in Ionic project
- You do not **have** to use Ionic Native, if a Cordova plugin is not available in Ionic Native you can just use the plugin directly
- All plugins, whether you are using Ionic Native or not, need to be installed through the command line
- One example of accessing a native API through the use of a plugin is the Camera plugin, which allows you to launch the users camera and retrieve a photo