SYST35300
Hybrid Mobile App Development

Presentation created for Sheridan College by A. Pak

Agenda

- Ionic Introduction
- Software Installations
- First Ionic Project

Ionic Introduction

- Ionic Framework is an open source UI toolkit for building performant, high-quality mobile and desktop apps using web technologies (HTML, CSS, and JavaScript).
- Ionic Framework is focused on the frontend user experience, or UI interaction of an app (controls, interactions, gestures, animations).
- Ionic Framework has official integration with **Angular**, **React** and **Vue.** **In this course we'll focus on the Ionic-Angular integration**. And, we'll use **Angular standalone components** for the Ionic projects we build in this course.
- Ionic Framework has a library of UI Components, which are reusable elements that serve as the building blocks for an application. Ionic Components are built with web standards using HTML, CSS, and JavaScript.
- Ionic Native is a library of Capacitor and Cordova plugins, integrations that make it easy to add native functionalities to Ionic apps.

https://ionicframework.com/docs

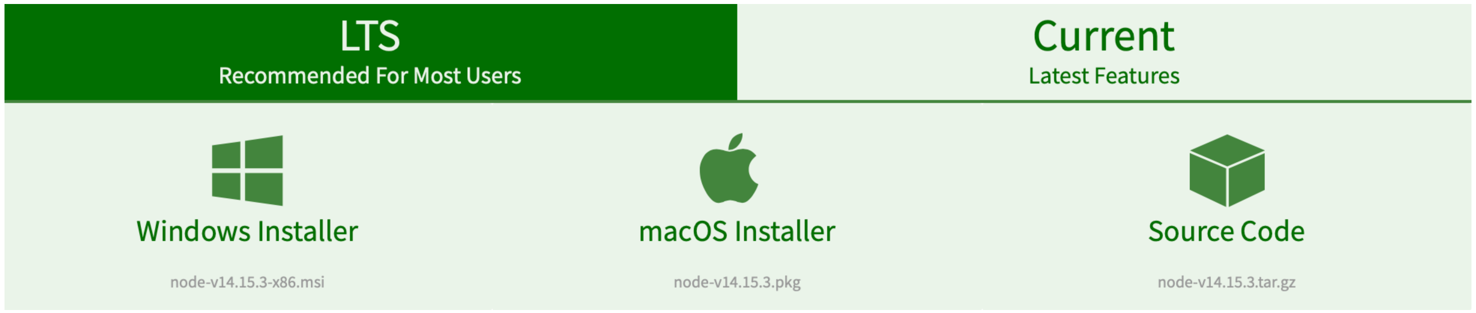
Ionic Introduction

- Ionic applications are generally considered as single-page applications (SPAs).
- In an Ionic app, navigation between different views or pages is managed within a single HTML file, typically the index.html. The app dynamically loads and updates content as users navigate through various sections, without requiring a full page reload. This dynamic updating is a key characteristic of SPAs, making Ionic particularly well-suited for building mobile and web applications with a smooth, app-like experience.
- In Ionic applications, the `<ion-router-outlet>` directive is used to manage and display routed views. It acts as a placeholder in the app where routed components are rendered according to the application's routing configuration.

Software Installations

- ❑ Node.js / Node Package Manager (NPM)
- ❑ Ionic CLI / Ionic Command Line
- ❑ IDE / Visual Studio Code

Node Install


LTS
Recommended For Most Users

Current
Latest Features

Windows Installer
node-v14.15.3-x86.msi

macOS Installer
node-v14.15.3.pkg

Source Code
node-v14.15.3.tar.gz

Node / NPM

Go to https://nodejs.org/en/

Download and install **"Recommended for Most Users"** version

To check Node version:

```
node --version
npm --version
```

Ionic Install

Install Ionic Command-Line Interface

- ❑ Open the command prompt or terminal window with ADMINISTRATOR privileges.
- ❑ To install Ionic CLI, execute the following command:

```
npm install -g @ionic/cli
```

- ❑ To verify the installation:

```
ionic --version  OR
ionic version
```

New Ionic Project

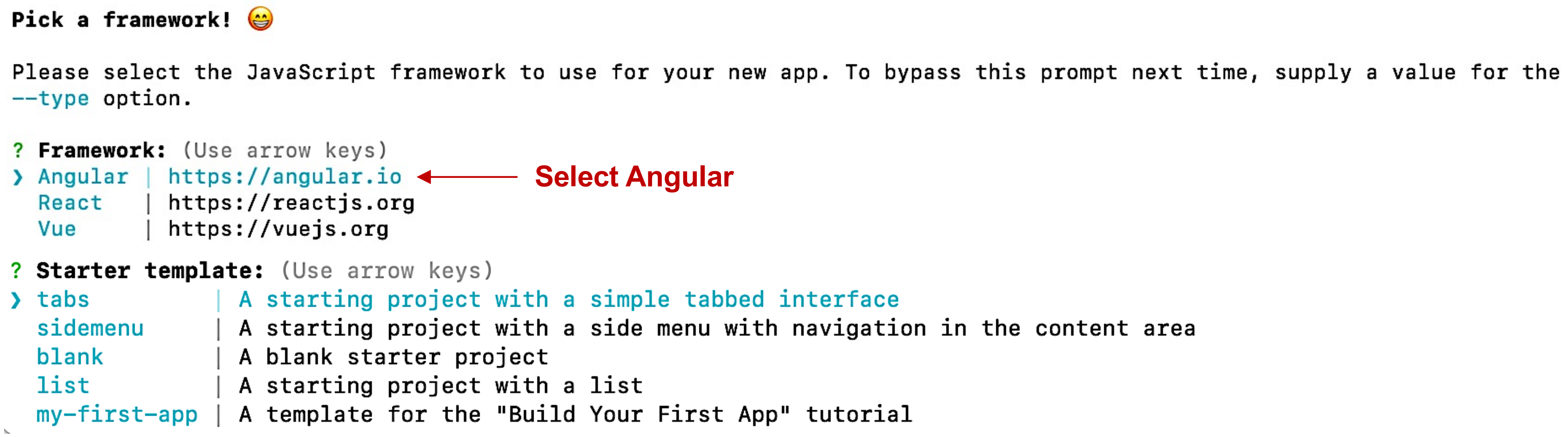
- Create a new folder to keep all your Ionic projects organized.
- Open the command prompt or terminal window with ADMINISTRATOR privileges.
- Navigate to the newly created folder and create Ionic projects.

New Ionic Project Cont…

In the newly created folder, utilize the command prompt or terminal window:

ionic start <App_Name>

ionic	Ionic CLI
start	To create a new project
<App_Name>	Name of the project



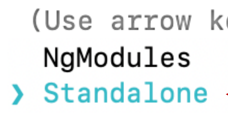
```
Pick a framework! 😁

Please select the JavaScript framework to use for your new app. To bypass this prompt next time, supply a value for the
--type option.

? Framework: (Use arrow keys)
❯ Angular | https://angular.io
  React   | https://reactjs.org
  Vue     | https://vuejs.org

? Starter template: (Use arrow keys)
❯ tabs         | A starting project with a simple tabbed interface
  sidemenu     | A starting project with a side menu with navigation in the content area
  blank        | A blank starter project
  list         | A starting project with a list
  my-first-app | A template for the "Build Your First App" tutorial
```

Select Angular



```
 (Use arrow keys)
  NgModules
❯ Standalone
```

Select Standalone

NOTE: Ctrl + C will terminate Node

Even in a standalone project, some tabs generated by the Ionic CLI may not be standalone by default; you may need to manually add the standalone property to true in the @Component decorator.

Ionic Standalone Projects

Angular Standalone Recap

- Angular standalone components eliminate the need for NgModules by allowing components, directives, and pipes to be imported directly within the @Component decorator.

Using Ionic Standalone Components

- Ionic supports the same approach, enabling the import of only required components from ***@ionic/angular/standalone*** instead of importing an entire module.

Using IonicModule for Convenience

- For projects that use many Ionic components, you can use IonicModule to avoid importing each component individually, offering a more convenient and centralized approach.

Scenario	**Best Practice**
Using only a few Ionic components	Import individual standalone components
Using many Ionic components	Import **IonicModule** for convenience

Run Ionic Project

Navigate to the newly created project folder and run the following command from the Terminal window to start the development server:

```
ionic serve
```

NOTE: This may take a minute … Be patient.

The application will automatically open in your default browser at localhost:8100.

Build Ionic Project

Build an Ionic project for web deployment.

```
ionic build <options>
```

```
ionic build    OR
```

```
ionic build -- --output-path XXX --base-href XXX
```

For example:

```
ionic build -- --output-path newApp --base-href ./
```

Build Ionic Project Cont…

We can automate the build process by setting up a hook script in the package.json file as follows:

```
"scripts:": {
  "ionic:build": "ng build --output-path newApp --base-href ./",
  ...
}
```

```
ionic build
```

Ionic Features

Use Ionic CLI to generate framework features such as pages, components, directives, services, etc.

```
ionic generate
```

or

```
ionic generate <feature> <name> [options]
```

We'll use Ionic CLI to generate the following features:

- ❑ Modules
- ❑ Components
- ❑ Pages
- ❑ Services

https://ionicframework.com/docs/cli/commands/generate

Ionic Module

Generate an Ionic Module:

`ionic generate module XXX` OR `ionic g module XXX`

A folder is created with one file:

XXX/XXX.module.ts

Ionic Component

Generate an Ionic Component:

`ionic generate component XXX` OR `ionic g component XXX`

Use `--spec false` to generate components without a spec file.

Use `--standalone` to declare a fully standalone component that doesn't need to be declared in any NgModule. This is optional.

A folder is created with these files:

- XXX/XXX.component.scss
- XXX/XXX.component.html
- XXX/XXX.component.spec.ts
- XXX/XXX.component.ts

Ionic Page

An Ionic page is a standalone component that serves as a ***complete view*** and may contain nested components. The Page is automatically added to app.routes.ts, enabling lazy loading.

Generate an Ionic Page:

`ionic generate page XXX` OR **`ionic g page XXX`**

Use **`--spec false`** to generate a page without a spec file.

An Ionic Page is created within a folder containing the following files:

- XXX/XXX.page.html
- XXX/XXX.page.scss
- XXX/XXX.page.spec.ts
- XXX/XXX.page.ts

Ionic Service

Generate an Ionic Service:

`ionic generate service XXX` OR `ionic g service XXX`

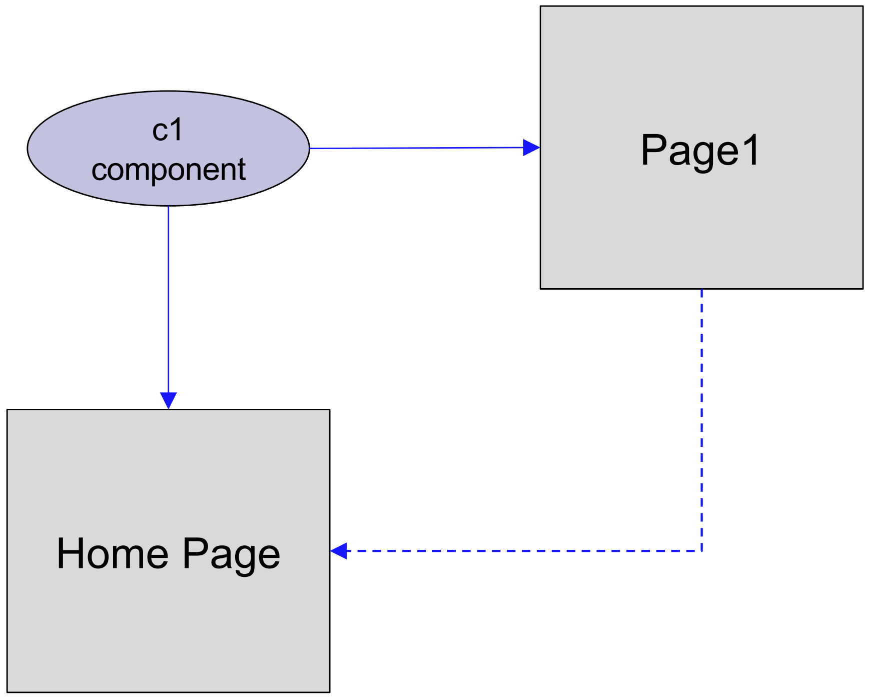
Use `--skip-tests` to generate a service without a spec file.

Two files are created:

- XXX.service.spec.ts
- XXX.service.ts

In-Class Exercise 1

1. Create an Ionic standalone project "w2-inclass1" using a blank template
2. Create a component "c1" and add it to the home page
3. Create a page "page1" in a folder called "pages" and include "c1" in "page1"



In-Class Exercise 1 Cont…

1. Create an Ionic standalone project "w2-inclass1" using a blank template
2. Create a standalone component "c1" and add it to the home page
3. Create a page "page1" in a folder called "pages" and include "c1" in "page1"

1. `ionic start w2-inclass1` OR `ionic start w2-inclass1 blank`
   a) `cd w2-inclass1`
2. a) `ionic g component c1 --standalone`
   b) Import c1 component into the home page
   - Update home.page.ts:

```
import { C1Component } from '../c1/c1.component';
imports: […, C1Component]
```

   c) Add the following to home.page.html

```
<app-c1></app-c1>
```

In-Class Exercise 1 Cont…

1. Create an Ionic standalone project "w2-inclass1" using a blank template
2. Create a standalone component "c1" and add it to the home page
3. Create a page "page1" in a folder called "pages" and include "c1" in "page1"

3. a) `ionic g page pages/page1`

   b) Import c1 component into the page1

   - Update page1.page.ts:

     ```
     import { C1Component } from 'src/app/c1/c1.component';
     imports: […, C1Component]
     ```

   - Update page1.page.html:

     ```
     <app-c1></app-c1>
     ```

Try this:

Navigate to page1 from the home page

- Import RouterModule into the home page:

  ```
  import { RouterModule } from '@angular/router';
  imports: […, RouterModule]
  ```

- Update home.page.html:

  ```
  <ion-button [routerLink]="['/page1']">Page 1</ion-button>
  ```

In-Class Exercise 1 Cont…

1. Create an Ionic standalone project "w2-inclass1" using a blank template
2. Create a standalone component "c1" and add it to the home page
3. Create a page "page1" in a folder called "pages" and include "c1" in "page1"
4. Pass data from "page1" to "c1"

4. a) Input data in "page1"

- Update page1.page.ts:

Define an Interface

```
interface Man { fName: string; lName: string; nName: string; }
```

Define and initialize a variable called 'man' of the type Man.

```
import { IonItem, IonLabel, IonInput } from
'@ionic/angular/standalone';
```

```
imports: […, IonItem, IonLabel, IonInput]
```

In-Class Exercise 1 Cont…

1. Create an Ionic standalone project "w2-inclass1" using a blank template
2. Create a standalone component "c1" and add it to the home page
3. Create a page "page1" in a folder called "pages" and include "c1" in "page1"
4. Pass data from "page1" to "c1"

4. a) Input data in "page1" (cont.)
   - Update page1.page.html (inside ion-content):

```
<ion-item>
  <ion-label position="stacked">Personal Profile</ion-label>
  <ion-input placeholder="First Name"
  [(ngModel)]="man.fName"></ion-input>
  <ion-input placeholder="Last Name"
  [(ngModel)]="man.lName"></ion-input>
  <ion-input placeholder="Nick Name"
  [(ngModel)]="man.nName"></ion-input>
</ion-item>
<app-c1 [c1Info]="man"></app-c1>
```

Update this

In-Class Exercise 1 Cont...

1. Create an Ionic standalone project "w2-inclass1" using a blank template
2. Create a standalone component "c1" and add it to the home page
3. Create a page "page1" in a folder called "pages" and include "c1" in "page1"
4. Pass data from "page1" to "c1"

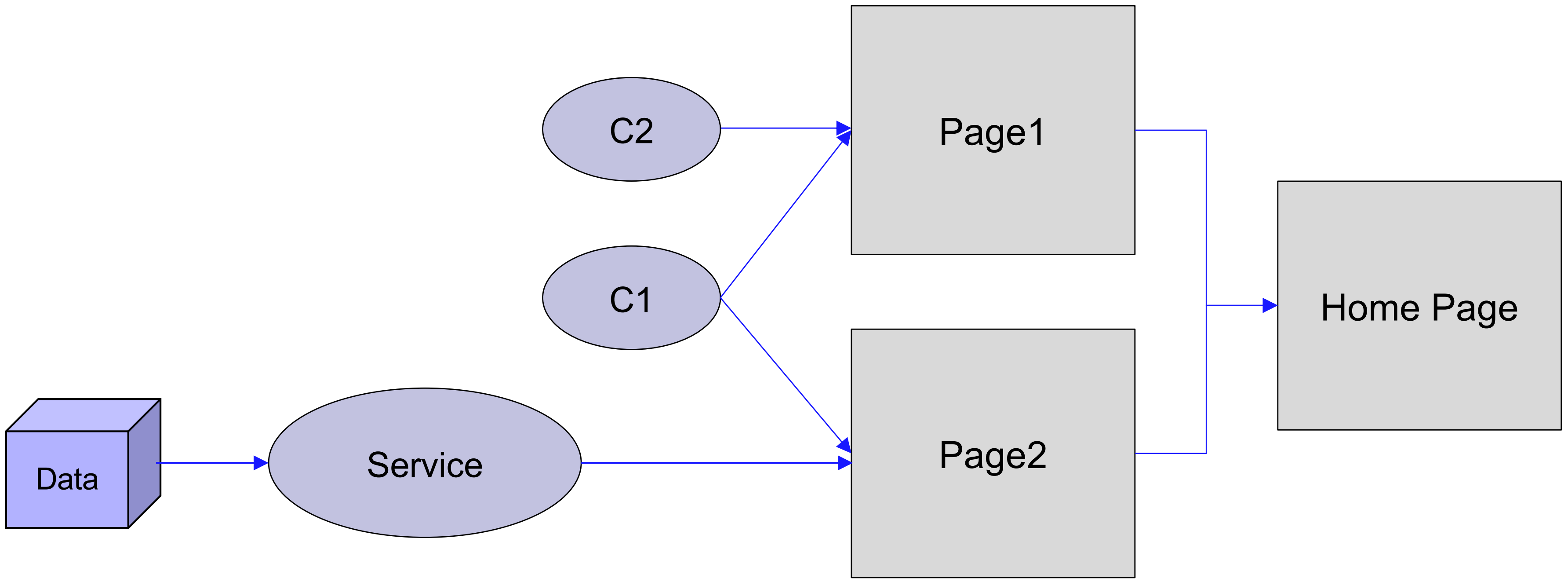
4. b) Setup @Input decorator in "c1"
   - Update c1.component.ts:
     ```
     import { Component, OnInit, Input } from '@angular/core';

     @Input() c1Info: any;
     ```
   - Update c1.component.html:
     ```
     {{c1Info.fName}} {{c1Info.lName}} {{c1Info.nName}}
     ```

*** Now that we're passing data to c1, REMOVE its selector from the homepage view to prevent errors.

In-Class Exercise 2

1. Create an Ionic standalone project "w2-inclass2" using a blank template
2. Create two pages "page1", "page2" in a folder called "pages"
3. Create two standalone components "c1", "c2"
4. Include "c1" in "page1" and "page2"; and "c2" in "page1"
5. Read data into "page2"



In-Class Exercise 2 Cont…

1. Create an Ionic standalone project "w2-inclass2" using a blank template
2. Create two pages "page1", "page2" in a folder called "pages"
3. Create two standalone components "c1", "c2"
4. Include "c1" in "page1" and "page2"; and "c2" in "page1"
5. Read data into "page2"

1. `ionic start w2-inclass2` OR `ionic start w2-inclass2 blank`
2. Create two pages "page1" and "page2"

   ```
   ionic g page pages/page1    ionic g page pages/page2
   ```
3. Create two components "c1" and "c2"

   ```
   ionic g component c1 --standalone  (c2)
   ```
4. Include "c1" in "page1" and "page2"; "c2" in "page1"
   - Update page1 and page2 accordingly.
   - page1.page.ts:

     ```
     import { C1Component } from 'src/app/c1/c1.component';
     import { C2Component } from 'src/app/c2/c2.component';
     imports: […, C1Component, C2Component]
     ```
   - Include c1and c2 selectors in page1.page.html

In-Class Exercise 2 Cont…

1. Create an Ionic standalone project "w2-inclass2" using a blank template
2. Create two pages "page1", "page2" in a folder called "pages"
3. Create two standalone components "c1", "c2"
4. Include "c1" in "page1" and "page2"; and "c2" in "page1"
5. Read data into "page2"

4. Navigation to page1 and page2 from the home page
   - Import RouterModule into the home page:
     ```
     import { RouterModule } from '@angular/router';
     imports: […, RouterModule]
     ```
   - Update home.page.html:
     ```
     <ion-button [routerLink]="['/page1']">Page 1</ion-button>
     <ion-button [routerLink]="['/page2']">Page 2</ion-button>
     ```

In-Class Exercise 2 Cont…

1. Create an Ionic standalone project "w2-inclass2" using a blank template
2. Create two pages "page1", "page2" in a folder called "pages"
3. Create two standalone components "c1", "c2"
4. Include "c1" in "page1" and "page2"; and "c2" in "page1"
5. Read data into "page2"

5. Read data into "page2"
   - Create interface file "pdata.ts" in the app folder

```
export interface Pdata {
  fName: string;
  lName: string;
  nName: string;
}
```

   - Create data file "pData.ts" in assets/data

```
import { Pdata } from '../../app/pdata';

export const PDATA: Pdata = {
  fName: 'Andy',
  lName: 'Pak',
  nName: 'Prof'
};
```

In-Class Exercise 2 Cont…

1. Create an Ionic standalone project "w2-inclass2" using a blank template
2. Create two pages "page1", "page2" in a folder called "pages"
3. Create two standalone components "c1", "c2"
4. Include "c1" in "page1" and "page2"; and "c2" in "page1"
5. Read data into "page2"

5. Read data into "page2" (cont.)
   - Generate service "getdata"

     ```
     ionic g service getdata
     ```

   - Update getdata.service.ts

     ```
     import { Pdata } from '../app/pdata';
     import { PDATA } from '../assets/data/pData';

     loadData(): Pdata {
       return PDATA;
     }
     ```

In-Class Exercise 2 Cont…

1. Create an Ionic standalone project "w2-inclass2" using a blank template
2. Create two pages "page1", "page2" in a folder called "pages"
3. Create two standalone components "c1", "c2"
4. Include "c1" in "page1" and "page2"; and "c2" in "page1"
5. Read data into "page2"

5. Read data into "page2" (cont.)
   - Inject the service into page 2 - update "page2.page.ts"

```
import { Pdata } from '../../pdata';
import { Getdata } from '../../getdata';

myData!: Pdata;
constructor(private ldData: Getdata) { }
loadMyData(): void {
  this.myData = this.ldData.loadData();   }

ngOnInit()
  { this.loadMyData(); }
```

   - Update "page2.page.html"

```
{{myData.fName}} {{myData.lName}} {{myData.nName}}
```

Ionic Web Storage

- Web storage enables web applications to store data locally within the user's browser.
- Dive Into HTML5 / Explanation of localStorage

  (http://diveintohtml5.info/storage.html)
- Access to the web storage is facilated through the special built-in objects - localStorage() and sessionStorage().
- Both objects have a relatively small number of methods that are highly powerful and easy to use.
- Unlike cookies, web storage data will NOT be automatically transmitted with every request to the server.

Ionic Web Storage

The localStorage object stores the data without an expiration date, making it **persistent**. This means the data will not be deleted when the session ends or when the browser is closed. Instead, they remain available the next day, week, or year until manually deleted.

The sessionStorage object functions similiarly to the localStorage object, with the distinction that it stores data for only one session. The data is automatically deleted when the session ends or when the user closes the browser window.

(From W3Schools …)

Ionic Web Storage

- Data is stored in key-value pairs.
- Data is retrieved by specifying the corresponding key.
- Data is stored as strings.
- LocalStorage maintains persistence across browser sessions.
- SessionStorage is cleared when a browser session concludes.

Key	Value
studID	"991222333"
studProg	"Computer Programmer"
numYears	"2"

Ionic Web Storage

- `localStorage.setItem(key, value):`
  Stores a value associated with a key.
- `localStorage.getItem(key):`
  Retrieves the value associated with the key.
- `localStorage.removeItem(key):`
  Removes an item from storage.
- `localStorage.length:`
  Returns the number of key/value pairs in the storage.
- `localStorage.key(i):`
  Given an integer i, this method finds the corresponding key.
- `localStorage.clear():`
  Clears all key/value pairs from localStorage.

Ionic Web Storage

- `sessionStorage.setItem(key, value):`
  Stores a value associated with a key.
- `sessionStorage.getItem(key):`
  Retrieves the value associated with the key.
- `sessionStorage.removeItem(key):`
  Removes an item from storage.
- `sessionStorage.length:`
  Returns the number of key/value pairs in the storage.
- `sessionStorage.key(i):`
  Given an integer i, this method finds the corresponding key.
- `sessionStorage.clear():`
  Clears all key/value pairs from sessionStorage.

Ionic Web Storage

Examples:

```
// Store an item
setData(key:string, value:any) {
   localStorage.setItem(key, value);
}

// Retrieve an item
data: any;
getData(key:string) {
   this.data = localStorage.getItem(key);
   return this.data;
}
```

Ionic Web Storage

Examples:

```
// Remove an item
removeData(key:string) {
   localStorage.removeItem(key);
}

// Retrieve all the values
getAllData() {
   for (let i=0; i<localStorage.length; ++i) {
      const myKey: any = localStorage.key(i);
      console.log(myKey, localStorage.getItem(myKey)); }
}

// Clear localStorage
clearData() {
   localStorage.clear();
}
```

In-Class Exercise 3

Use "w2-inclass2"

1. Create a page "page3" in the "pages" folder
2. Add, retrieve, and delete items in localStorage

1. Create page "page3" in the "pages" folder

   ```
   ionic g page pages/page3
   ```

   - Add the following to home.page.html

     ```
     <ion-button [routerLink]="['/page3']">Page 3</ion-button>
     ```

2. Add, retrieve, and delete items in localStorage
   - Update getdata.ts

     ```
     // Store an item
     setData(key:string, value:any) {
         localStorage.setItem(key, value);
     }
     // Retrieve an item
     data: any;
     getData(key:string) {
         this.data = localStorage.getItem(key);
         return this.data;
     }
     ```

In-Class Exercise 3 Cont…

Use "w2-inclass2"

1. Create a page "page3" in the "pages" folder
2. Add, retrieve, and delete items in localStorage

2. Add, retrieve, and delete items in localStorage (cont.)
   - Update getdata.ts

```
// Remove an item
removeData(key:string) {
    localStorage.removeItem(key);
}
```

   - Update page3.page.html

```
<ion-button (click)=addItem()>Add Item</ion-button>
<ion-button (click)=retrieveItem()>Retrieve Item</ion-button>
<ion-button (click)=deleteItem()>Delete Item</ion-button>
```

   - Update page3.page.ts

```
import { …, IonButton } from '@ionic/angular/standalone';
imports: […, IonButton]
```

In-Class Exercise 3 Cont…

Use "w2-inclass2"

1. Create a page "page3" in the "pages" folder
2. Add, retrieve, and delete items in localStorage

2. Add, retrieve, and delete items in localStorage (cont.)
   - Update page3.page.ts

```
import { Getdata } from '../../getdata';

constructor(private ldData: Getdata) { }

addItem() {
   this.ldData.setData('a1','This is a1');
   console.log(`Added: ${this.ldData.getData('a1')}`);
}

retrieveItem() {
   console.log(`Retrieved: ${this.ldData.getData('a1')}`);
}

deleteItem() {
   this.ldData.removeData('a1')
    console.log(`Deleted: ${this.ldData.getData('a1')}`);
}
```