Due 11:59PM, October 22nd.

## Mobile Application Development with Ionic

Ionic is an Angular based tool for mobile development. You can use it to deploy and deploy to iOS, Android, Windows phone, even Blackberry and others. You can do a surprising amount with it without having to code in any native platform such as specific Android or iOS.

If you must have speed for things like games you might choose to develop in the native platform or in Unity but if you are performing simple data manipulation, then using Ionic can make you very productive.

You can develop in Ionic free and once you start publishing your builds commercially it costs $13.00 per user.

Ionic is designed to make it easy for you to create applications that look like native phone applications. It provides all kinds of templates and resources to make this task an easy one.

### Ionic Components

Ionic has lots of great components to make you app look and feel like an iOS or Android application.

https://ionicframework.com/docs/components/

### Ionic Icons

There are tonnes of ionic icons that can be added for free.

<https://ionicframework.com/docs/ionicons/>

There are many other great icon libraries on the web too if those are not enough.

### Installing the Ionic

In case you do not have Ionic, run these commands. If you are on a Mac run them using the **sudo** command to give yourself the proper permission:

**npm install -g @angular/cli**

**npm i -g reflect-metadata**

**npm install -g cordova ionic**

### Creating the Ionic Application

Example : Ionic Helloworld

To create an ionic application, create a directory with the **mkdir** command. Then navigate to this directory using Terminal. Then run the following command to create a new application.

**>ionic start myApp**

When prompted, choose the **tabs** project.

From the same directory as **package.json**, run the following command:

**>npm install**

### Running the Application

Next, from the application directory beneath the package.json file run the following command to launch your application in a browser. I recommend copying the URL and placing it in Chrome to obtain the best debugging experience:

**>ionic serve**

### Running the Application in the iOS Emulator

You can also test your application in Android and iOS emulators. To do this for iOS, from the application directory run the following scripts. Note it will take a large amount of time to run the application in the iOS emulator so I recommend the earlier example most of the time which involves running the application in the browser:

**ionic platform add ios**

**ionic build ios**

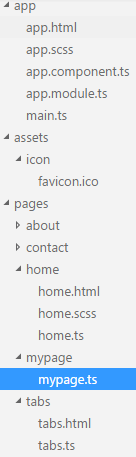
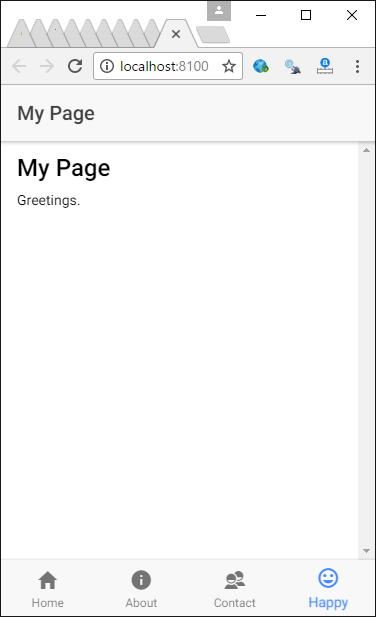
**ionic emulate ios**

Warning: The last three instructions will only work in OSX if you have XCode installed.

### Adding Tab Pages

Example : Adding a Tab View

This example demonstrates how to add a tab page into an Ionic project. When finished your project structure will have the same listing as shown on the left. When running the project, the application will have a new tab and associated view as shown on the right:

Starting with Example 1, add a new folder named **mypage** with a file named **mypage.ts** inside the **pages** directory. Place the following code inside the mypage.ts file. Notice how different ionic specific elements enable the header, navbar and view body:

|  |
| --- |
| import { Component } from '@angular/core';  import { NavController } from 'ionic-angular';  @Component({  selector: 'page-mine',  template: `<ion-header>  <ion-navbar>  <ion-title>My Page</ion-title>  </ion-navbar>  </ion-header>  <ion-content padding>  <h2>My Page</h2>  Greetings.  </ion-content>  `  })  export class MyPage {  constructor(public navCtrl: NavController) {  }  } |

Next, the new component must be added into the module. In src/app/app.module.ts add the following highlighted code to load your new page into the library:

|  |
| --- |
| import { NgModule, ErrorHandler } from '@angular/core';  import { BrowserModule } from '@angular/platform-browser';  import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';  import { MyApp } from './app.component';  import { AboutPage } from '../pages/about/about';  import { ContactPage } from '../pages/contact/contact';  import { MyPage } from '../pages/mypage/mypage';  import { HomePage } from '../pages/home/home';  import { TabsPage } from '../pages/tabs/tabs';  import { StatusBar } from '@ionic-native/status-bar';  import { SplashScreen } from '@ionic-native/splash-screen';  @NgModule({  declarations: [  MyApp,  AboutPage,  ContactPage,  MyPage,  HomePage,  TabsPage  ],  imports: [  BrowserModule,  IonicModule.forRoot(MyApp)  ],  bootstrap: [IonicApp],  entryComponents: [  MyApp,  AboutPage,  ContactPage,  MyPage,  HomePage,  TabsPage  ],  providers: [  StatusBar,  SplashScreen,  { provide: ErrorHandler, useClass: IonicErrorHandler }  ]  })  export class AppModule { } |

**src/pages/my-page.html**

Replace the ion-content element with the following:

|  |
| --- |
| <ion-content padding>  This is my brand new page.  </ion-content> |

To initialize the new tab, add the highlighted lines to **src/pages/tabs/tab.ts**:

|  |
| --- |
| import { Component } from '@angular/core';  import { AboutPage } from '../about/about';  import { ContactPage } from '../contact/contact';  import { HomePage } from '../home/home';  import { MyPage } from '../mypage/mypage';  @Component({  templateUrl: 'tabs.html'  })  export class TabsPage {  tab1Root = HomePage;  tab2Root = AboutPage;  tab3Root = ContactPage;  tab4Root = MyPage;  constructor() {  }  } |

Next, set the icon for the tab by adding the highlighted ion-tab element to **src/pages/tabs/tabs.html**. I used the happy face icon. This icon can be found amongst many others at <https://ionicframework.com/docs/ionicons/>

|  |
| --- |
| <ion-tabs>  <ion-tab [root]="tab1Root" tabTitle="Home" tabIcon="home"></ion-tab>  <ion-tab [root]="tab2Root" tabTitle="About"  tabIcon="information-circle"></ion-tab>  <ion-tab [root]="tab3Root" tabTitle="Contact" tabIcon="contacts"></ion-tab>  <ion-tab [root]="tab4Root" tabTitle="Happy" tabIcon="happy"></ion-tab>  </ion-tabs> |

Running the app with this change shows the icon at the bottom.



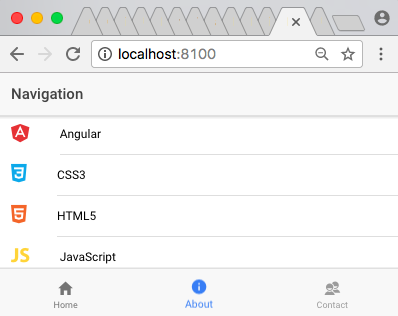
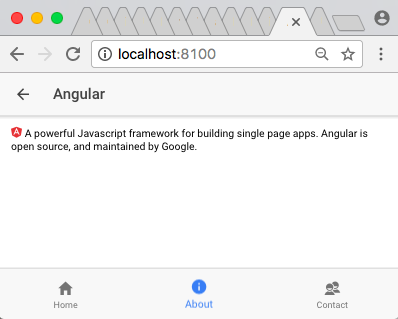
Exercise

Add a fifth tab, view and icon of your choice to the solution from Example 2. Show your new and revised code pages here:

|  |
| --- |
| ../../../../../Desktop/Screen%20Shot%202017-10-21%20at%206.00.48%20P../../../../../Desktop/Screen%20Shot%202017-10-21%20at%206.00.13%20P../../../../../Desktop/Screen%20Shot%202017-10-21%20at%206.00.07%20P |

### Adding a List and Detail View

Adding a detail and list view like the one below is easy too.

Example : Adding a List and Detail View

To begin start with a new example or with Example 2. After, replace the existing code with the following code to src/pages/about/about.ts. The code highlighted in green is for the list view and the code highlighted in yellow is for the detail view.

**src/pages/about/about.ts**

|  |
| --- |
| import { Component } from '@angular/core';  import { NavController, NavParams } from 'ionic-angular';  @Component({    templateUrl: 'navigation-details.html',  })  export class NavigationDetailsPage {    item;    constructor(params: NavParams) {      this.item = params.data.item;    }  }  @Component({    template: `  <ion-header>    <ion-navbar>      <ion-title>Navigation</ion-title>    </ion-navbar>  </ion-header>  <ion-content>    <ion-list>      <button ion-item \*ngFor="let item of items" (click)="openNavDetailsPage(item)" icon-left>        <ion-icon [name]="'logo-' + item.icon" [ngStyle]="{'color': item.color}" item-left></ion-icon>        {{ item.title }}      </button>    </ion-list>  </ion-content>  `  })  export class AboutPage {    items = [];    constructor(public nav: NavController) {      this.items = [        {          'title': 'Angular',          'icon': 'angular',          'description': 'A powerful Javascript framework for building single page apps. Angular is open source, and maintained by Google.',          'color': '#E63135'        },        {          'title': 'CSS3',          'icon': 'css3',          'description': 'The latest version of cascading stylesheets - the styling language of the web!',          'color': '#0CA9EA'        },        {          'title': 'HTML5',          'icon': 'html5',          'description': 'The latest version of the web\'s markup language.',          'color': '#F46529'        },        {          'title': 'JavaScript',          'icon': 'javascript',          'description': 'One of the most popular programming languages on the Web!',          'color': '#FFD439'        },        {          'title': 'Sass',          'icon': 'sass',          'description':  'Syntactically Awesome Stylesheets - a mature, stable, and powerful professional grade CSS extension.',          'color': '#CE6296'        },        {          'title': 'NodeJS',          'icon': 'nodejs',          'description': 'An open-source, cross-platform runtime environment for developing server-side Web applications.',          'color': '#78BD43'        },        {          'title': 'Python',          'icon': 'python',          'description': 'A clear and powerful object-oriented programming language!',          'color': '#3575AC'        },        {          'title': 'Markdown',          'icon': 'markdown',          'description':  'A super simple way to add formatting like headers, bold, bulleted lists, and so on to plain text.',          'color': '#412159'        },        {          'title': 'Tux',          'icon': 'tux',          'description': 'The official mascot of the Linux kernel!',          'color': '#000'        },      ]    }    openNavDetailsPage(item) {      this.nav.push(NavigationDetailsPage, { item: item });    }  } |

**src/app/app.module.ts**

|  |
| --- |
| import { NgModule, ErrorHandler } from '@angular/core';  import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';  import { MyApp } from './app.component';  import { AboutPage } from '../pages/about/about';  import { ContactPage } from '../pages/contact/contact';  import { HomePage } from '../pages/home/home';  import { TabsPage } from '../pages/tabs/tabs';  import { NavigationDetailsPage } from '../pages/about/about';  import { StatusBar } from '@ionic-native/status-bar';  import { SplashScreen } from '@ionic-native/splash-screen';  @NgModule({    declarations: [      MyApp,      AboutPage,      ContactPage,      HomePage,      TabsPage,      NavigationDetailsPage    ],    imports: [      IonicModule.forRoot(MyApp)    ],    bootstrap: [IonicApp],    entryComponents: [      MyApp,      AboutPage,      ContactPage,      HomePage,      TabsPage,      NavigationDetailsPage    ],    providers: [      StatusBar,      SplashScreen,      {provide: ErrorHandler, useClass: IonicErrorHandler}    ]  })  export class AppModule {} |

Finally add this template to display your details view.

**src/pages/about/navigation-details.html**

|  |
| --- |
| <ion-header>    <ion-navbar>      <ion-title>        {{ item.title }}      </ion-title>    </ion-navbar>  </ion-header>  <ion-content padding>    <ion-icon [name]="'logo-' + item.icon" [ngStyle]="{'color': item.color}"></ion-icon>    {{ item.description }}  </ion-content> |

Exercise

Add a new item to your list view in a manner that allows you to display unique content in the detail view if it is clicked. Show your code changes here:

|  |
| --- |
| **export class** AboutPage {  items = [];   **constructor**(**public** nav: NavController) {  **this**.items = [  {  'title': 'Angular',  'icon': 'angular',  'description': 'A powerful Javascript framework for building single page apps. Angular is open source, and maintained by Google.',  'color': '#E63135'  },  {  'title': 'CSS3',  'icon': 'css3',  'description': 'The latest version of cascading stylesheets - the styling language of the web!',  'color': '#0CA9EA'  },  {  'title': 'HTML5',  'icon': 'html5',  'description': 'The latest version of the web\'s markup language.',  'color': '#F46529'  },  {  'title': 'JavaScript',  'icon': 'javascript',  'description': 'One of the most popular programming languages on the Web!',  'color': '#FFD439'  },  {  'title': 'Sass',  'icon': 'sass',  'description':  'Syntactically Awesome Stylesheets - a mature, stable, and powerful professional grade CSS extension.',  'color': '#CE6296'  },  {  'title': 'NodeJS',  'icon': 'nodejs',  'description': 'An open-source, cross-platform runtime environment for developing server-side Web applications.',  'color': '#78BD43'  },  {  'title': 'Python',  'icon': 'python',  'description': 'A clear and powerful object-oriented programming language!',  'color': '#3575AC'  },  {  'title': 'Markdown',  'icon': 'markdown',  'description':  'A super simple way to add formatting like headers, bold, bulleted lists, and so on to plain text.',  'color': '#412159'  },  {  'title': 'Tux',  'icon': 'tux',  'description': 'The official mascot of the Linux kernel!',  'color': '#000'  },  // new item  {  'title': 'jap',  'icon': 'tux',  'description': 'The official CIT representative',  'color': '#62266f'   }  ] |

### Data Persistence

This document discusses the ionic endorsed method of storing data. Just in case you need to have the instructions handy this is information on how to set up an Ionic project.

### Debugging

To debug your applications, wherever possible use the Chrome browser to view errors and to step through your code. Some plugins will require that you run the application using the native mobile platform with an emulator. If you have to use an emulator for iOS you will also notice there is an XCode project file which you can launch so you can view logging in the debug area.

**ionic cordova platform add browser**

### Basic Ionic Storage

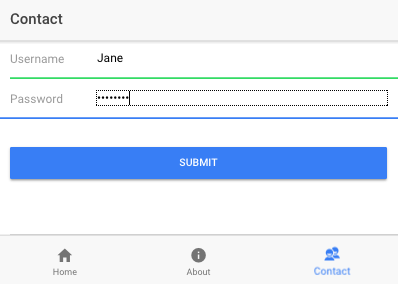
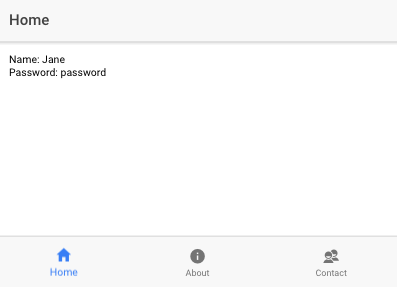
Ionic version 2 appears to be moving towards using ionic storage as the preferred way to manage persistent data using key value pairs. The data can be stored using indexedDB storage in the browser. When SQLite is installed with either iOS and Android it can be stored to SQLite too.

This repository contains more information about ionic-storage:

<https://github.com/driftyco/ionic-storage>

Example

This demonstration shows how to create a basic ionic application that allows a user to store data on the contact page and view it on the home page.

#### Installing Ionic Storage

To use this in your Ionic 2/Angular 4 apps, either start a fresh Ionic project which has it installed by default, or run:

**npm install @ionic/storage**

If you'd like to use SQLite as a storage engine, install a SQLite plugin (only works while running in a simulator or on device):

**cordova plugin add cordova-sqlite-storage --save**

Once you have installed ionic/storage at least, run the following commands to build your app.

>ionic start myApp (remember to choose the tabs option)

>npm install

>ionic serve

>npm install @ionic/storage

**src/app/module.ts**

|  |
| --- |
| import { NgModule, ErrorHandler } from '@angular/core';  import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';  import { MyApp } from './app.component';  import { AboutPage } from '../pages/about/about';  import { ContactPage } from '../pages/contact/contact';  import { HomePage } from '../pages/home/home';  import { TabsPage } from '../pages/tabs/tabs';  import { StatusBar } from '@ionic-native/status-bar';  import { SplashScreen } from '@ionic-native/splash-screen';  import { IonicStorageModule } from '@ionic/storage';  import { Storage } from '@ionic/storage';  @NgModule({    declarations: [      MyApp,      AboutPage,      ContactPage,      HomePage,      TabsPage    ],    imports: [      IonicModule.forRoot(MyApp),      IonicStorageModule.forRoot({          name: '\_\_mydb',          driverOrder: ['indexeddb', 'sqlite', 'websql']     })      ],    bootstrap: [IonicApp],    entryComponents: [      MyApp,      AboutPage,      ContactPage,      HomePage,      TabsPage    ],    providers: [      StatusBar,      SplashScreen,      {provide: ErrorHandler, useClass: IonicErrorHandler}    ]  })  export class AppModule {} |

The contact page allows users to add new users through entry in input boxes that can be submitted on completion. This is the component:

**src/pages/contacts/contact.ts**

|  |
| --- |
| import { Component } from '@angular/core';  import { Storage } from '@ionic/storage';  import { NavController } from 'ionic-angular';  @Component({    selector: 'page-contact',    templateUrl: 'contact.html'  })  export class ContactPage {    pwd:string;    username:string;    input:string;    constructor(public navCtrl: NavController, public storage: Storage) {    }    // User clicked button.    setPerson() {        this.storage.set('name', this.username)        this.storage.set('pwd', this.pwd)        this.input = this.username + " " + this.pwd;    }  } |

This is the HTML containing the input boxes and submit button in an Ionic template.

**src/pages/contacts/contact.html**

|  |
| --- |
| <ion-header>    <ion-navbar>      <ion-title>        Contact      </ion-title>    </ion-navbar>  </ion-header>  <ion-content>    <ion-list>      <ion-item>        <ion-label fixed>Username</ion-label>        <ion-input [(ngModel)]="username" type="text"></ion-input>      </ion-item>      <ion-item>        <ion-label fixed>Password</ion-label>        <ion-input [(ngModel)]="pwd" type="password"></ion-input>      </ion-item>    </ion-list>    <div padding>      <button ion-button color="primary" (click)="setPerson()" block>Submit</button>    </div>    <ion-item>    {{input}}    </ion-item>  </ion-content> |

The home component contains code to read and display the data:

**src/pages/home/home.ts**

|  |
| --- |
| import { Component  } from '@angular/core';  import { NavController } from 'ionic-angular';  import { Storage } from '@ionic/storage';  @Component({    selector: 'page-home',    templateUrl: 'home.html'  })  export class HomePage {    name:string;    pwd:string;    constructor(public navCtrl: NavController, public storage: Storage) {        // Prepare storage.        storage.ready().then(() => {            // Store some data if none exists.            if(!this.storage.get('pwd')) {                console.log("Storing default data!");                this.storage.set('name', 'This is the default name.');                this.storage.set('pwd',  'This is the default password.');            }        });    }    // Executes every time page is viewed.    ionViewWillEnter(){      this.displayData();    }    // Uses a promise to get data.    displayData() {        this.storage.get('name').then((name) => {            this.name = name;        });        this.storage.get('pwd').then((pwd) => {            this.pwd = pwd;        });    }  } |

Here is the home view which shows the data that is retrieved from storage.

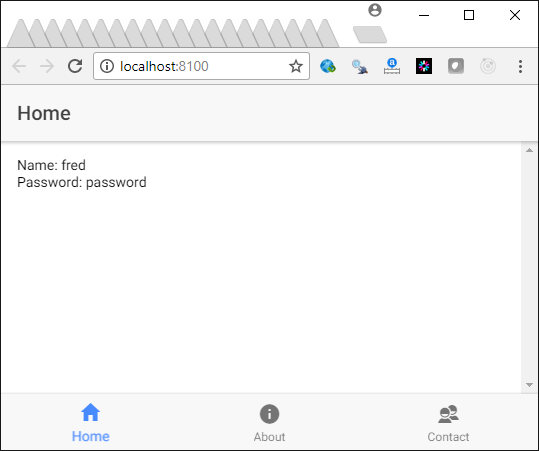
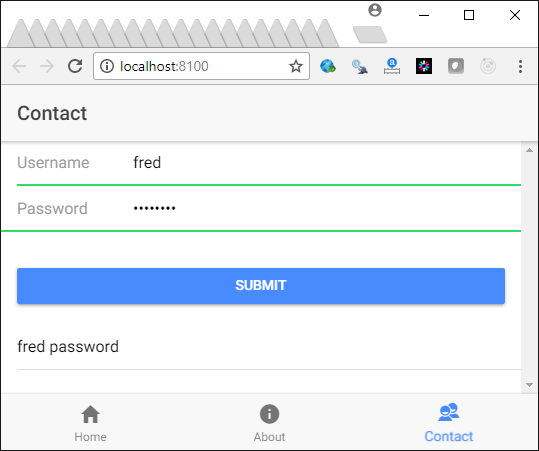
**src/pages/home/home.html**

|  |
| --- |
| <ion-header>    <ion-navbar>      <ion-title>Home</ion-title>    </ion-navbar>  </ion-header>  <ion-content padding>    Name: {{name}}<br/>    Password: {{pwd}}  </ion-content> |

Build and run the ionic application:

**ionic serve**

Once the application runs you will be able to save data using ionic storage from the contact page and view it on the home page.



### Storing and Retrieving Multiple Rows of Data

Example : Storing and Retrieving Multiple Rows of Data

This application shows how to store and retrieve multiple rows of data. To begin, start with . Then, create a models folder and add this class to it:

**src/models/usermodel.ts**

|  |
| --- |
| export class UserModel {      pwd:string;      username:string;      constructor() {      }  } |

Replace the code in contact.ts so the application adds a new object to an array that is stored when a new contact is submitted.

**src/pages/contacts/contact.ts**

|  |
| --- |
| import { Component } from '@angular/core';  import { Storage } from '@ionic/storage';  import { NavController } from 'ionic-angular';  import { UserModel } from '../../models/usermodel';  @Component({    selector: 'page-contact',    templateUrl: 'contact.html'  })  export class ContactPage {    pwd:string;    username:string;    input:string;    constructor(public navCtrl: NavController, public storage: Storage) {    }    // User clicked button.    setPerson() {        this.storage.set('name', this.username)        this.storage.set('pwd', this.pwd)        this.input = this.username + " " + this.pwd;        this.addToTable();    }    addToTable() {        this.storage.get('myusers').then((myusers) => {            var userModel      = new UserModel;            userModel.pwd      = this.pwd;            userModel.username = this.username;            var allusers =  myusers;  // Get table.            // Create empty array if it does not exist.            if(!allusers) {              allusers = [];            }            allusers.push(userModel); // Insert new row in array.            this.storage.set('myusers',allusers); // Store revision.        });    }  } |

This modified version of the home.ts component retrieves the array from storage every time the page is viewed.

**src/pages/home/home.ts**

|  |
| --- |
| import { Component  } from '@angular/core';  import { NavController } from 'ionic-angular';  import { Storage } from '@ionic/storage';  @Component({    selector: 'page-home',    templateUrl: 'home.html'  })  export class HomePage {    \_myusers:any[];    constructor(public navCtrl: NavController, public storage: Storage) {    }    // Executes every time page is viewed.    ionViewWillEnter(){      this.displayData();    }    // Uses a promise to get data.    displayData() {        this.storage.get('myusers').then((myusers) => {            this.\_myusers = myusers;        });    }  } |

The home view now needs a loop to iterate through all items that were retrieved in the array.

**src/pages/home/home.html**

|  |
| --- |
| <ion-header>    <ion-navbar>      <ion-title>Home</ion-title>    </ion-navbar>  </ion-header>  <ion-content padding>    <li \*ngFor="let \_user of \_myusers">      {{\_user.username}} {{\_user.pwd}}    </li>  </ion-content> |

# Vue

Vue is a client side framework that functions a lot like Angular. However, it is comparatively light weight.

## Installation

To install vue, be sure to download and install the latest version of NodeJS. Then, run the following command from the command prompt:

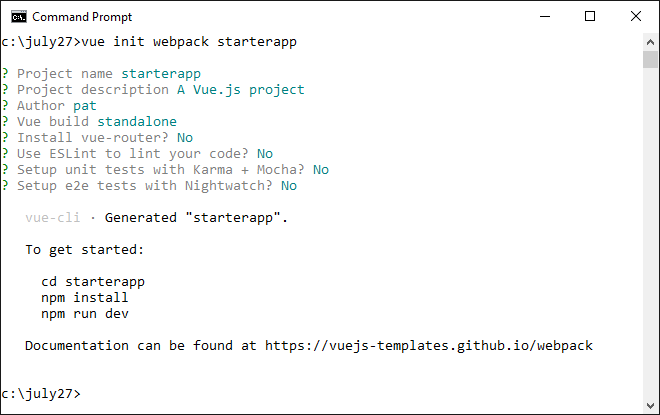
npm install -g vue-cli

## Creating and Running an Introductory Vue Project

To create a Vue project, create a project folder and cd into it. Then, run the command:

>vue init webpack starterapp

The project generation process will prompt you for the project name, description, author. For now, when prompted to choose a router, linting, unit testing and e2e tests choose no.



As you can see from the command prompt screenshot, the project generator gives helpful hints on steps to take. Once the project has been generated, cd into your project folder and run the following command to download the necessary node\_modules packages:

>npm install

After, run the following command to launch the Vue project:

>npm run dev

## Overview of Vue Components

A Vue application is built using components. Vue components contain three sections:

1. Template
2. Script
3. Style

Example

Build your project using the steps described above. Then, replace the code inside HelloWorld.vue with the following code which is a simplified version of the original default code:

**src/components/HelloWorld.vue**

|  |
| --- |
| <template>  <div class="hello">  <h1>{{ msg }}</h1>  </div>  </template>  <script>  export default {  name: 'HelloWorld',  data () {  return {  msg: 'Welcome to Your Vue.js App'  }  }  }  </script>  <!-- Add "scoped" attribute to limit CSS to this component only -->  <style>  #app {  color: #00ff00;  }  </style> |

Next, replace the App.vue code with this code.

**src/App.vue**

|  |
| --- |
| <template>  <div id="app">  Parent  <hello-world/>  </div>  </template>  <script>  import HelloWorld from './components/HelloWorld'  export default {  name: 'app',  components: {  HelloWorld  }  }  </script>  <style>  #app {  color: #00ff00;  }  </style> |

After, run the following command to launch the Vue project:

>npm run dev

Exercise

A Vue component has three distinct sections. The HelloWorld.vue file from Example 6 has these three sections. Paste the code from HelloWorld.vue for Example 6 here but also include comments to clearly denote each of the three sections:

|  |
| --- |
| <template>  <!--this section has the html code of the project-->  <div class="hello">  <h1>{{ msg }}</h1>  </div> </template>  <script> // code similar to the ts file  // the logic of the app is here  **export default** {  name: 'HelloWorld',  data () {  **return** {  msg: 'Welcome to Your Vue.js App'  }  }  } </script>  <!-- Add "scoped" attribute to limit CSS to this component only --> <style>  */\*the CSS of the app \*/* #app {  color: #00ff00;  } </style> |

Exercise

Starting with Example 6, replace the style element with the following element inside HelloWorld.vue.

|  |
| --- |
| <style>  .hello {  color: #0000ff;  }  </style> |

Show a screenshot of your output after making and saving this change:

|  |
| --- |
| ../../../../../Desktop/Screen%20Shot%202017-10-21%20at%206.53.02%20P |

Exercise

In Example 6, inside **main.js**, **el** takes the value **#app**. This allows us to display the output of the Vue application at the **<div id="app"></div>** element in index.html.

|  |
| --- |
| import Vue from 'vue'  import App from './App'  Vue.config.productionTip = false  /\* eslint-disable no-new \*/  new Vue({  el: '#app',  template: '<App/>',  components: { App }  }) |

Change the value of el: to #myapp. Then adjust the id attribute of the div tag inside index.html so the application still works properly.

Show the adjusted contents of main.js here:

|  |
| --- |
| ../../../../../Desktop/Screen%20Shot%202017-10-21%20at%206.56.27%20P |

Show your index.html page contents here:

|  |
| --- |
| ../../../../../Desktop/Screen%20Shot%202017-10-21%20at%206.56.24%20P |

Exercise

If you change the name option inside HelloWorld.vue from:

name: 'HelloWorld',

to:

name:’HelloMyWorld’,

What three changes will you need to make inside App.vue to make the code work again? Show your revised App.vue file with these three changes included here. Note the HTML element will need to be all lower case and each word must be separated by a hyphen:

|  |
| --- |
| * In main.js * Change the import to HelloMyWorld * Change components to: HelloMyWorld * Change <hello-world> to <hello-my-world> |

Exercise

Add a second data property called *subtitle* to the **HelloWorld.vue** file and assign it the value of your name. Display this value of *subtitle* in addition to the *msg* data property. Show your revised HelloWorld.vue component here:

|  |
| --- |
| ../../../../../Desktop/Screen%20Shot%202017-10-21%20at%207.01.10%20P |

### Looping

Looping is enabled through a *v-for* directive.

Example

This example shows how to implement a v-for directive to iterate through an array of playing cards. To begin, start with Example 6. Then, at this JSON array declaration to the data() function in the *HelloWorld* component:

|  |
| --- |
| cards: [  {cardNum: 'A', suit: 'Clubs'},  {cardNum: 'Two', suit: 'Hearts'},  {cardNum: 'Three', suit: 'Diamonds'}  ], |

Next, add this code to the template in the *HelloWorld* component. It must be placed inside the div tag inside the template element:

|  |
| --- |
| <ul>  <li v-for="card in cards">  {{card.cardNum}} {{card.suit}}  </li>  </ul> |

The output will show the contents of the cards array in an unordered list:



### Two Way Data Binding

Just like with Angular 4, Vue has two way data binding.

* + v-model works as the attribute in an input element.
  + v-text works as the attribute in a display element like a span tag.

Example : 2-Way Binding

You can create a template, start with Example 7 and replace the contents of HelloWorld.vue with this version:

|  |
| --- |
| <template>  <div class="hello">  <h1>{{ msg }} </h1>  <input type="text" v-model="myContents"><br/>  Input Value: <span v-text="myContents"></span><br/>  </div>  </template>  <script>  export default {  name: 'HelloWorld',  data () {  return {  msg: 'Welcome to Your Vue.js App!',  myContents: 'change me',  }  }  }  </script>  <!-- Add "scoped" attribute to limit CSS to this component only -->  <style>  .hello {  color: #0000ff;  }  </style> |

The output shows how any text entered in the input with the **v-model** attribute is immediately bound to output tags that contain **the v-text** attribute.



|  |
| --- |
| <template>  <div>  <h1>{{ msg }}</h1>  <input type='text' v-model='someContent'>  <button v-on:click="showContent(someContent)">Show the content</button>  </div>  </template>  <script>  export default {  name: 'hello',  data () {  return {  msg: 'Hello world from Vue!',  }  },  methods: {  showContent: function (message) {  alert(message)  }  }  }  </script> |

### Click Handling

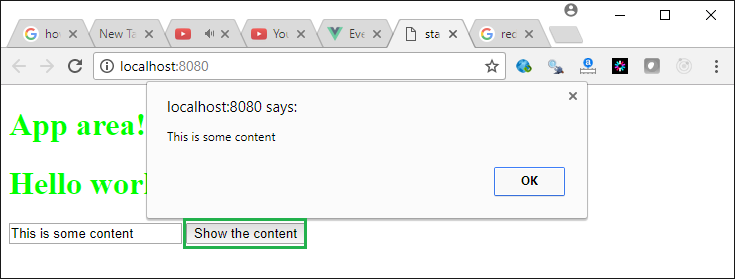
The click handler can be handled with the **v-on:click** directive and additional functions can be included in the methods section.

Example : Click Handling

The following example shows how click handlers and functions in the methods section work together.

|  |
| --- |
| <template>  <div>  <h1>{{ msg }}</h1>  <input type='text' v-model='someContent'>  <button v-on:click="showContent(someContent)">Show the content</button>   <div>  <input type='text' v-model='someContentt'>  <button v-on:click="assign(someContentt)">Show the content</button>  <p v-if="text"> {{text}}</p>  </div>  </div> </template>  <script>  **export default** {  name: 'HelloWorld',  data () {  **return** {  msg: 'Hello world from Vue!',  text:**null**,  }  },  methods: {  showContent: **function** (message) {  alert(message)  },  assign:**function** (mesg) {  **this**.text=mesg;  }  }  } </script> |

Once finished the page output functions as shown:



Exercise

Inside HelloWorld.vue create a second text input in addition to the existing one and add a second button in addition to the first one. When the user clicks the second button display the contents of the second input in a paragraph tag within the web page. Show the contents of your modified HelloWorld.vue file here after these changes:

|  |
| --- |
| <template>  <div>  <h1>{{ msg }}</h1>  <input type='text' v-model='someContent'>  <button v-on:click="showContent(someContent)">Show the content</button>   <div>  <input type='text' v-model='someContentt'>  <button v-on:click="assign(someContentt)">Show the content</button>  <p v-if="text"> {{text}}</p>  </div>  </div> </template>  <script>  **export default** {  name: 'HelloWorld',  data () {  **return** {  msg: 'Hello world from Vue!',  text:**null**,  }  },  methods: {  showContent: **function** (message) {  alert(message)  },  assign:**function** (mesg) {  **this**.text=mesg;  }  }  } </script> |

### Conditionals

To show and hide content or to dynamically change it the **v-if** directive is helpful.

Example

This example displays content if the property *someContent* has data. This demonstration starts with Example 9. Replace the contents of HelloWorld.vue with the following.

|  |
| --- |
| <template>  <div>  <h1>{{ msg }}</h1>  <input type='text' v-model='someContent'>  <div>  <span v-if="someContent!=''">Now you see me</span>  </div>  </div>  </template>  <script>  export default {  name: 'HelloWorld',    data () {  return {  msg: 'Hello world from Vue!',  someContent:'',  }  },  }  </script> |

When you run the program content will appear and disappear depending on whether the user enters content or not.

### Form Inputs

Excellent documentation for simple form inputs can be found at:

<https://vuejs.org/v2/guide/forms.html>

Exercise

Create a brand new Vue application using the steps described at the beginning of the section in this Word document. Then use the code from the radio, select and ‘multiple checkbox’ control demonstrations at <https://vuejs.org/v2/guide/forms.html> so you can add them into the HelloWorld.vue page. Show your revised working HelloWorld.vue page after these changes.

|  |
| --- |
| <template>   <div>  <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">  <label for="jack">Jack</label>  <input type="checkbox" id="john" value="John" v-model="checkedNames">  <label for="john">John</label>  <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">  <label for="mike">Mike</label>  <br>  <span v-if="checkedNames">Checked names: {{ checkedNames }}</span>  <br></br>   <input type="radio" id="one" value="One" v-model="picked">  <label for="one">One</label>  <br>  <input type="radio" id="two" value="Two" v-model="picked">  <label for="two">Two</label>  <br>  <span>Picked: {{ picked }}</span>   <select v-model="selected">  <option disabled value="">Please select one</option>  <option>A</option>  <option>B</option>  <option>C</option>  </select>  <span>Selected: {{ selected }}</span>  </div> </template>  <script>  **export default** {  name: 'HelloWorld',  data () {  **return** {  msg: 'Hello world from Vue!',  checkedNames:[],  picked:"",  selected: ''  }  },  methods: {  }  } </script> |

test