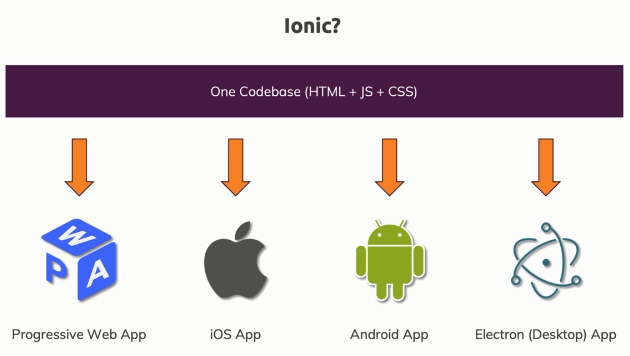
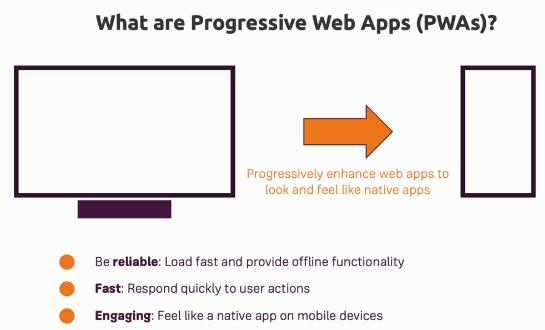
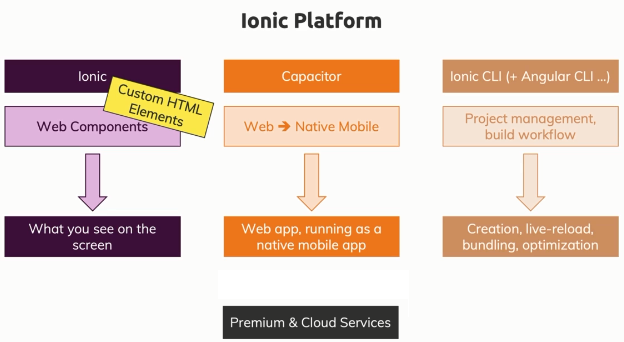
Ionic  
Build iOS, Android & Web Apps with Ionic & Angular

# Getting Started

## What Is Ionic?



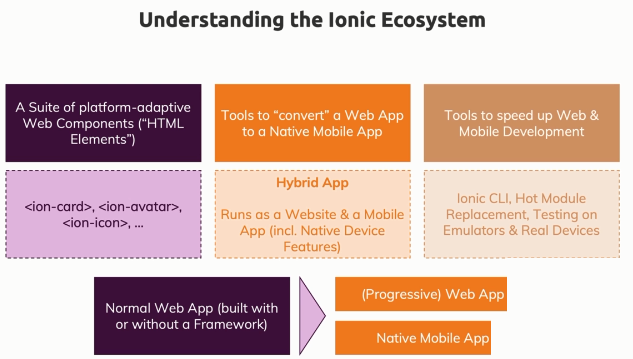


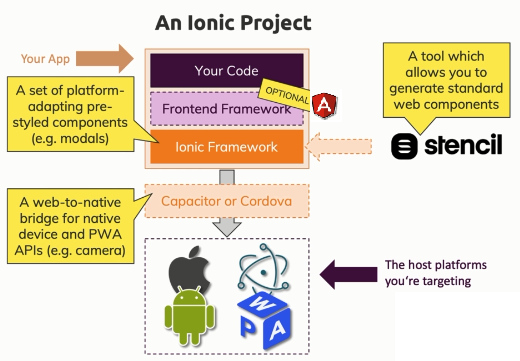


* You can build an app for different platforms with one and the same codebase, with minor adjustments.
* The Ionic platform is all about Ionic which in its core is a set of web components.
* Now web components on the other hand are a technique, a technology supported by modern browsers which allows you to basically build your own HTML elements that behind the scenes have more complex logic. E.g. something like a tabs component let's say where the user can toggle between different tabs.
* In the past, you could build something like this on your own by writing your own HTML, adding your own CSS and adding your own Javascript logic, Ionic gives you such a functionality packaged up in a finished, ready-to-use web component and it has lots of web components you can use.
* The core of ionic is this **suite of nicely styled, platform-agnostic and automatically adjusting (based on platform you're running on) web components** which you can dump into your web project.
* Ionic is a company which also works on a tool called **Capacitor** is essentially a tool that is capable of taking your existing web app and wrapping it into a so-called web view into a native mobile app.
* Ionic CLI can work with other CLIs, like the Angular CLI which it will use behind the scenes, which helps us create Ionic projects, manage them, use capacitor to convert our web app into a mobile app, so basically which helps us with the entire build workflow.

This makes developing simply easier, faster and allows us to finally build and package our app up.

## A Closer Look at the Ionic Platform

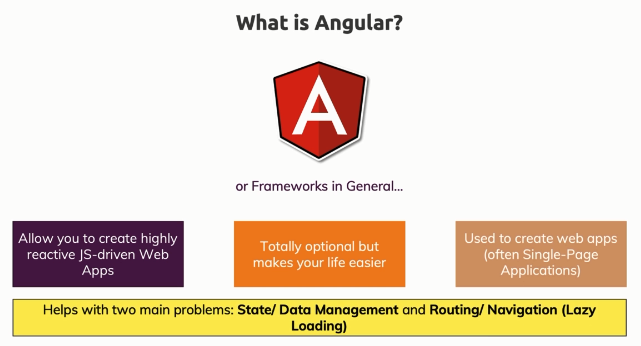




* The Ionic web components are built with a tool called **Stencil**.
* If you want to learn how the Ionic web components were built behind the scenes, or if you want to build your own web components which you can use in conjunction with your framework and with Ionic, then learning Stencil could be well worth it.
* **Capacitor** acts as a bridge between our web code (our web Javascript code) and the native platform we're running on. So that from our Javascript code, we can trigger certain functions which then in the end trigger native code on your real device to for example open the camera, to get the user location, to show an alert or anything like that.

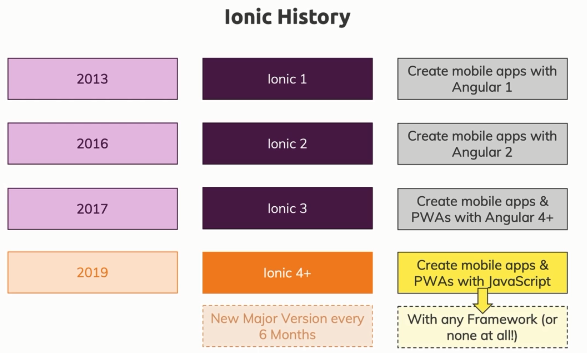
This can then be done with real native code executed on your behalf without you needing to write it by tools like Capacitor or Cordova.

## Why use Angular (or any other Frontend Framework) with Ionic

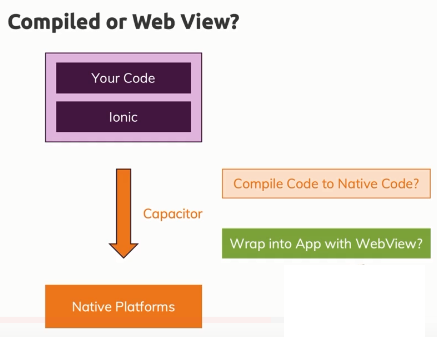
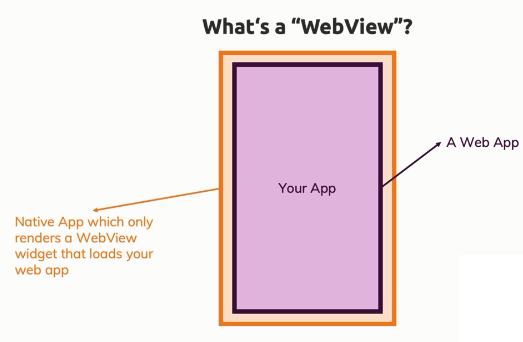


## The History of Ionic

* Ionic 4 and all the future versions of Ionic are based on web components. Web components is a browser specification that allows you to add your own HTML elements that run totally independent from any web framework you might be using.



## How to Build Native Mobile Apps with Ionic

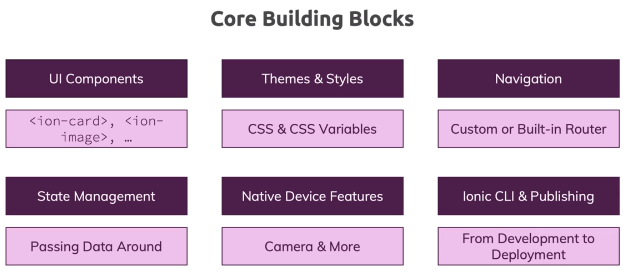
* On native mobile apps, you can launch a web view in the application to host a web page inside of that app.
* **Web view** in the end is a special widget you can use a native app development that is a fully fledged browser that doesn't look like one because you don't have a URL bar at the top and so on. It just is a fullscreen browser and that is what Ionic uses.
* With tools like Capacitor or Cordova, you in the end get a mobile app shell that has such a web view in it and also then has some capabilities of launching a simple web server, running mobile on the device that hosts your Ionic web app inside of that web view. And then Cordova or Capacitor, also give you a bridge so to say through which you can tap into real native device features from inside your web app running in web view.
* Ionic takes this approach of wrapping your app into a web view no matter if you using Capacitor or Cordova.
* Now that **web view** allows you to run your normal web app inside of a native app that renders a full screen browser.
* Now you could say that this has to be slower than a compiled app where you work with the real native widgets and technically that would be true, such an app will be a little bit smaller because there is this extra wrapper and it is just a web page but it is super important to stress here that modern devices are so fast and an Ionic app typically uses so little performance that you will absolutely not see any difference and then you would just have the advantage of being able to build a cross-platform app with almost no effort at all, that looks and feels like a native app, that also is technically a native app and where you can tap into all the native device features like the camera, etc.

## Comparing Ionic to Alternatives

* Ionic vs React Native vs Flutter vs etc.  
  <https://academind.com/learn/flutter/react-native-vs-flutter-vs-ionic-vs-nativescript-vs-pwa/>

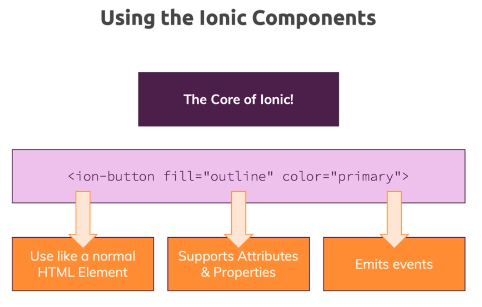
# Ionic Component Basics

## Core App Building Blocks



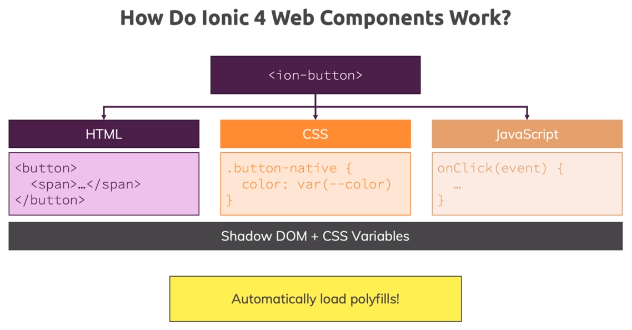
## Under the Hood of Ionic Components

### Using Ionic Components



* We use ionic components like a normal HTML elements. It supports attributes and properties and these elements can also emit events.
* These web components are not just about pre-styled elements, they do add JavaScript logic to them as well.

### How do Ionic 4 Web Components work



* Such a web component is basically like a wrapped up piece of pre-structured HTML code.
* The ionic component also has CSS styling which often use CSS variables so that the component can be styled from outside as well.
* We also have a JavaScript portion in there which adds certain functionalities to that component, which exposes properties that can be set, which controls things like that we can set the color or the fill mode of that button.
* This is all packaged up together and basically wrapped into a Javascript object which we can add to the HTML/DOM code with the component selector e.g. <ion-button>.
* Now under the hood, this also uses a technique called The Shadow DOM and CSS variables which help with encapsulating the styles of a particular component, so that the styling applied to the elements in a component doesn't spill over to your app or to other components and the Ionic web components actually also automatically load any polyfills that might be required to make the components run on older browsers.
* Modern browsers support all the web component features by default, older browsers don't and therefore Ionic actually make sure that these components work on older browsers as well by automatically polyfilling everything that is required.

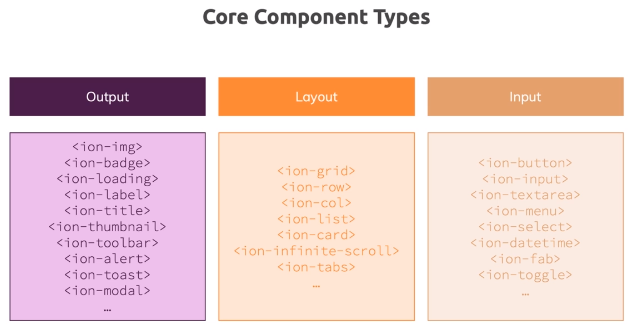
## Setting Up a Non-Angular Ionic Project

* Include Ionic CDN packages from <https://ionicframework.com/docs/intro/cdn>

## Where to Learn all about Ionic Components

* Ionic UI Components – <https://ionicframework.com/docs/components>
* Refer to the docs for a component to learn more about that. E.g. <https://ionicframework.com/docs/api/button>
* There you will see all the details about a component like how to use it, how it will look on mobile devices, how to style it, different events, etc.

## Component Categories



## Adding Icons & Using Slots

* Slots are default web component concept which basically allows web components to reserve certain places in their built-in markup where certain content should be rendered or can be targeted to be rendered.

## Using CSS Utility Attributes

* To control general things like a margin around elements or padding inside of elements or positioning inside of a container of a box, you can use some utility features provided by Ionic, basically couple of utility classes or attributes, which you can add two elements to have some default access kick in.
* Refer – <https://ionicframework.com/docs/layout/css-utilities>

## Why Angular (or any other framework)?

* Doing everything on our own will be hard.
* Difficult to implement routing. Ionic actually has its own router component but the Angular router is way more powerful than that, as it gives us a lot of advanced functionalities.
* Difficult to implement smart routing. So routing correctly where we also can work with things like query params in the URL and so on is actually quite difficult, so we definitely want to use a finished and tested and battle-proven routing solution like the Angular router to have efficient, fast and powerful feature-rich routing we can add in our app to switch pages and navigate around.



## Notes

* Demo projects with vanilla JavaScript –
  + [budget-planner](https://github.com/sameerbhilare/Ionic/tree/main/Workspace/budget-planner)
  + [courses-ratings](https://github.com/sameerbhilare/Ionic/tree/main/Workspace/courses-ratings)

# Angular + Ionic

## Why Angular (or any other framework)?

* Doing everything on our own will be hard.
* Difficult to implement routing. Ionic actually has its own router component but the Angular router is way more powerful than that, as it gives us a lot of advanced functionalities.
* Difficult to implement smart routing. So routing correctly where we also can work with things like query params in the URL and so on is actually quite difficult, so we definitely want to use a finished and tested and battle-proven routing solution like the Angular router to have efficient, fast and powerful feature-rich routing we can add in our app to switch pages and navigate around.



## Creating a New Ionic Angular Project

* Ionic installation - <https://ionicframework.com/docs/intro/cli>

## How Angular & Ionic Work Together

* The @ionic/angular package in the end is a wrapper package around the Ionic components suite.
* It makes the usage of these components in Angular easier and actually also more efficient, especially when we talk about things like the alert controller, modal controller, etc.
* To run ionic Angular project,

>ionic serve

It is same as ng serve, but just opens a specific port 8100. So essentially ng serve will also work.

## Adding & Loading a New Page

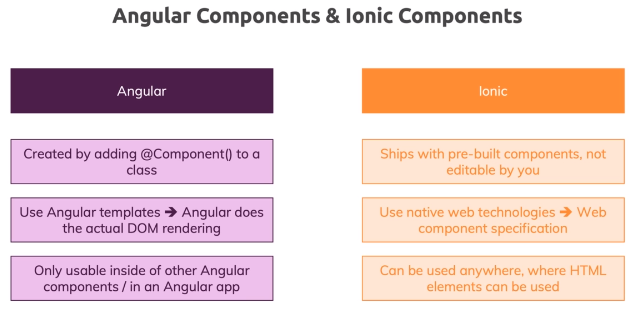
* We can use ng to generate new components but ionic provides better options with “ionic generate” command. E.g. page, service, component, etc.
* With “ionic generate” command and selecting “page”, ionic will create a new lazy loading module and will also adjust app routing module. This in the end will create a new route.
* ion-router-outlet is directive added by @ionic/angular. ion-router-outlet wraps the angular router-outlet and adds extra stylings and features in order to properly view the ionic pages.

## Managing State with Services

* >ionic generate service

## Angular Components vs Ionic Components

* We can create normal Angular components in an ionic app.
* >ionic generate component



# Building Native Apps with Capacitor

## General Information

* Android Native app Development –

<https://ionicframework.com/docs/developing/android>

* iOS Native App Development –

<https://ionicframework.com/docs/developing/ios>

* You can't build iOS apps on Windows unless you're using Ionic's paid service – AppFlow.
* Android apps can be built on both MacOS and Windows systems as well as Linux.

## Creating an Android App

* Android Native app Development –

<https://ionicframework.com/docs/developing/android>

* To build your app for Android or iOS, we need to use **Capacitor**.
* Capacitor Official Docs – <https://capacitorjs.com/docs>
* Before below step, make sure you first build your Angular app with ng build.
* Project Setup - <https://ionicframework.com/docs/developing/android#project-setup>

# Debugging

## Error Messages & console.log()

* Google the error message
* Use console.log() statements.

## Using the Browser DevTools & Breakpoints

* IF you have source maps, then you can add breakpoint to your TS code in browser and debug your code.

## Using VS Code for Debugging

* <https://code.visualstudio.com/docs/nodejs/angular-tutorial#_debugging-angular>

## Debugging the UI & Performance

* Using chrome’s Elements and Network tabs.
* Also use Performance and Memory tabs of chrome browser.

## Debugging Android Apps

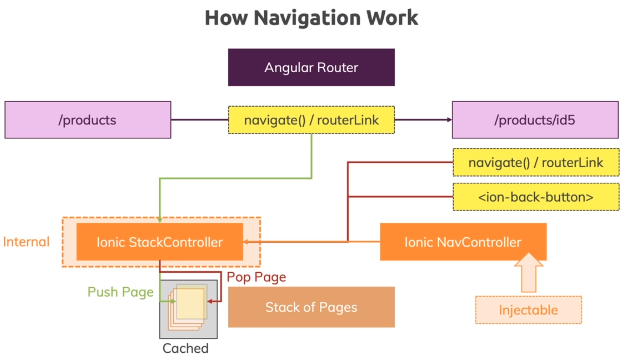
* To see logs, Android Studio -> Run tab
* To debug, chrome://inspect

## Useful Resources & Links

* Learn more about the Chrome dev tools: <https://developers.google.com/web/tools/chrome-devtools/>

# Navigation & Routing in Ionic Apps

## How Routing Work In An Ionic + Angular App



* @angular/ionic package actually kind of wraps that Angular routing functionality and it does so to basically add all the nice transitions.
* Ionic thinks of navigation or the different pages as a stack of pages.
* Because if you think about a mobile app, you typically see one page at a time and you can go to a new page or press the back button and go back. You can essentially do the same in the browser of course using browser back button. So you can think of navigation as a stack of pages and you always view the page which is on top of this stack.
* Ionic controls these stack of pages with a **StackController**. (This class is internal to Ionic)
* When you use the Angular router for navigating, Ionic basically watches the Angular router, it has a listener to your routing actions. When you go forward, it basically pushes a new page onto that stack and when you go back or use the ion–back-button, it pops that topmost page off.
* How does the StackController know whether a navigate call is meant to go forward or backward? It basically has a look at the internal ID of your navigation action and it turns out that by that ID you can determine whether the new route is basically one step ahead of the old route or behind it. That in turn is important for playing the right animation for transitioning from pages.
* These stack of pages is also cached for you by Ionic and that is different than your normal Angular web app.
* In Angular web app, what happens is if you go to a new page and you go back, all these pages which in the end are just components are essentially destroyed when you leave them, no matter if you're going forward or backward.
* Now with Ionic, that's actually not the case. When you are going forward to a new page, that old page which is still in that stack of pages is kept in a cache in memory. So this whole stack of pages is actually cached and when you pop a page off, then this is removed from the cache because it's removed from the stack.
* We primarily use the **Angular router** for navigating around which is the recommended tool for routing in an Ionic Angular app. We also got the Ionic **NavController**. That is an injectable service which you can inject into any component or a service you might have and that essentially gives you some utility methods that will also interact with that StackController. For example it will give you a pop method which allows you to manually pop off the latest page on the stack of pages and which will therefore trigger a back navigation. This is good to know but we primarily use Angular router.

## Ionic Page Caching & Extra Lifecycle Hooks

* **ionViewWillEnter** will execute right before the content of the page has been loaded and is displayed on the screen. And **ionViewDidEnter** will be called right after that.
* Both (ionViewWillEnter and ionViewDidEnter) are called whenever a page becomes visible.
* That is important to understand because with caching, if a page is still in cache and you are just not seeing it because another page is on top of that stack of pages, so it will actually never be destroyed so ngOnDestroy will never be called and ngOnInit will also never be called when you go back to that page which is still on the stack of pages.
* ngOnDestroy doesn’t gets called till the page is in the stack, but you might still want to do some cleanup work, for that you also get **ionViewWillLeave** and **ionViewDidLeave** hooks. And these are called whenever the page becomes invisible so to say, i.e. whenever the page is not on top of stack. **ionViewWillLeave** gets called before our leaving animation starts and **ionViewDidLeave** gets called when the leaving animation finishes.
* ngOnDestroy gets called when the topmost page is popped off the stack of pages. And what's removed from the stack is removed from the cache as well.
* **Bottom-line** - You have to be aware of the fact that ngOnInit and ngOnDestroy will not run on every page every time you leave it. It depends on whether you are popping the page or you're pushing a new page on top of it and therefore you should rely on Ionic specific lifecycle hooks (ionView\*) if you need to do some state or data updating whenever a page becomes visible or invisible.

## Understanding Ionic Tabs

* Tabs are a common navigation concept you see in mobile apps, you can of course use them on web apps just as well but you especially see them in mobile apps and tabs basically allow you to press the different tabs in the bottom tab bar and then load different pages based on which tab you pressed.
* One important takeaway is that you will have **separate navigation page stacks for each tab**.
* That of course is very useful for making sure that navigation is stored and users don't reset their navigation by using tabs.

### Adding Tabs to the App

* The component where you have your ion-tabs element must have some child routes matching 'tab' attribute of ion-tab so that the tabs will remain sticked to the page.
* Refer – <https://ionicframework.com/docs/api/tabs>

# Ionic Components Overview

# Tips and Tricks

* Ionic Glossary – <https://ionicframework.com/docs/reference/glossary>
* Ionic is the modern version of cordova.
* **Stencil** is a tool developed by the Ionic company, by the Ionic team that they use internally to build all their ionic web components. It's in the end a tool that just makes the creation of web components easier, it still spits out normal vanilla web components.
* Ionic caches pages.
* For authentication guard on lazy loaded modules, we need to implement both canLoad and canActivate guards. Because canLoad executes once before loading the module. Once the module is loadedit doesn’t run again. However canActivate runs on each page load.