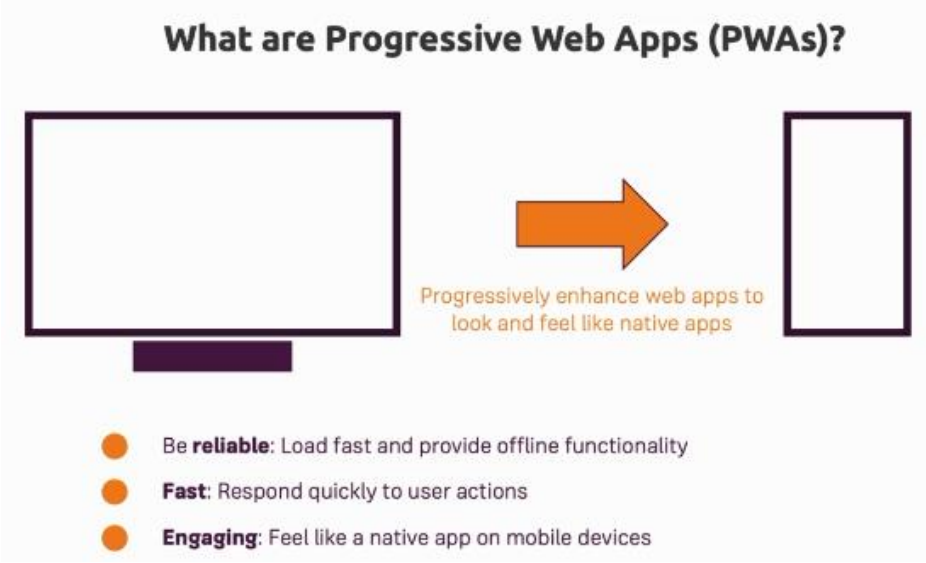
Ionic

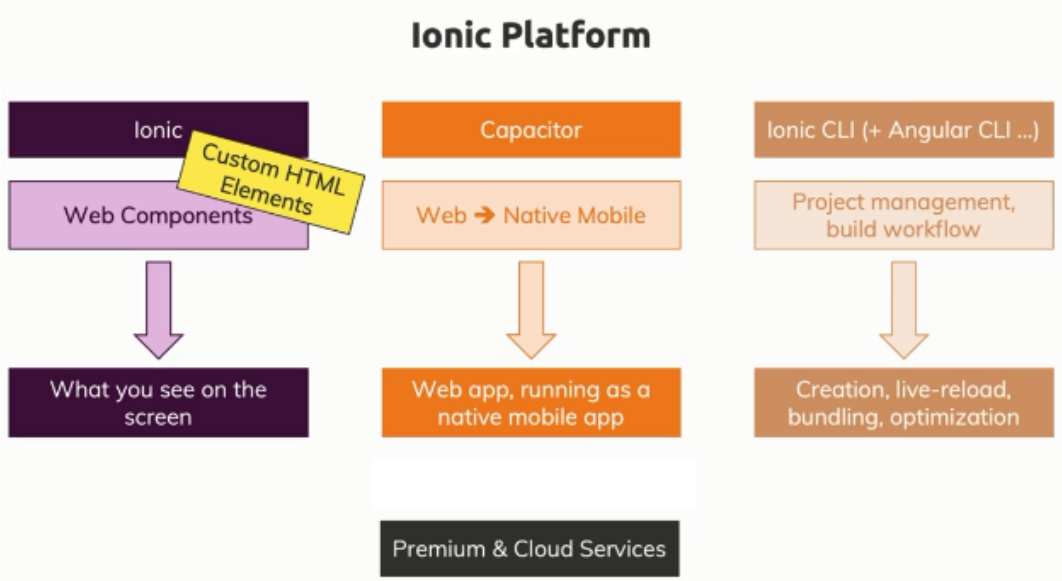
Build iOS, Android & Web Apps with Ionic & Angular

Contents

Getting Started

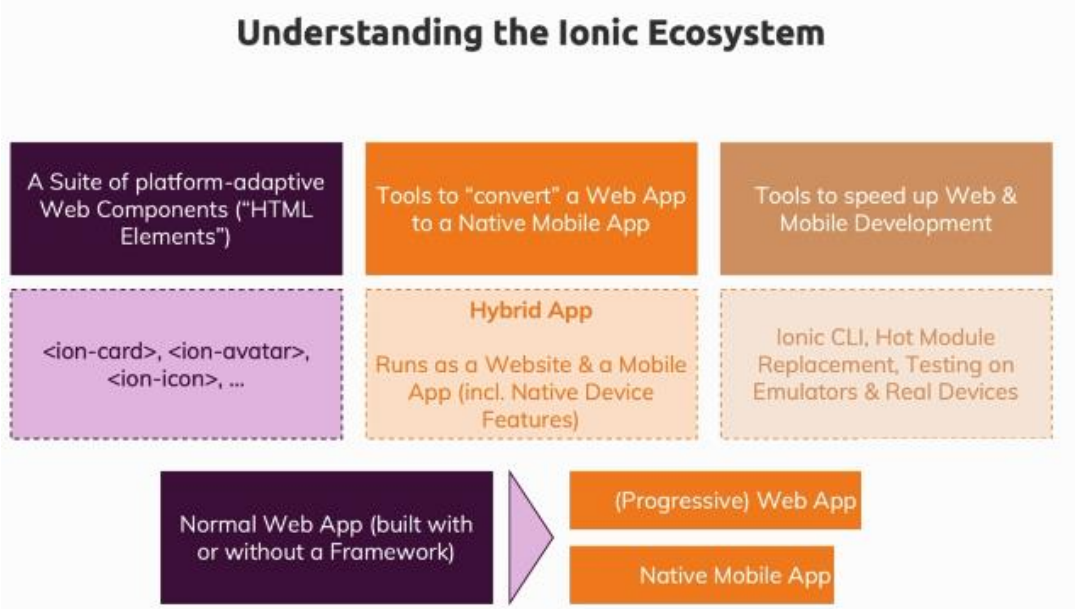
What Is Ionic?

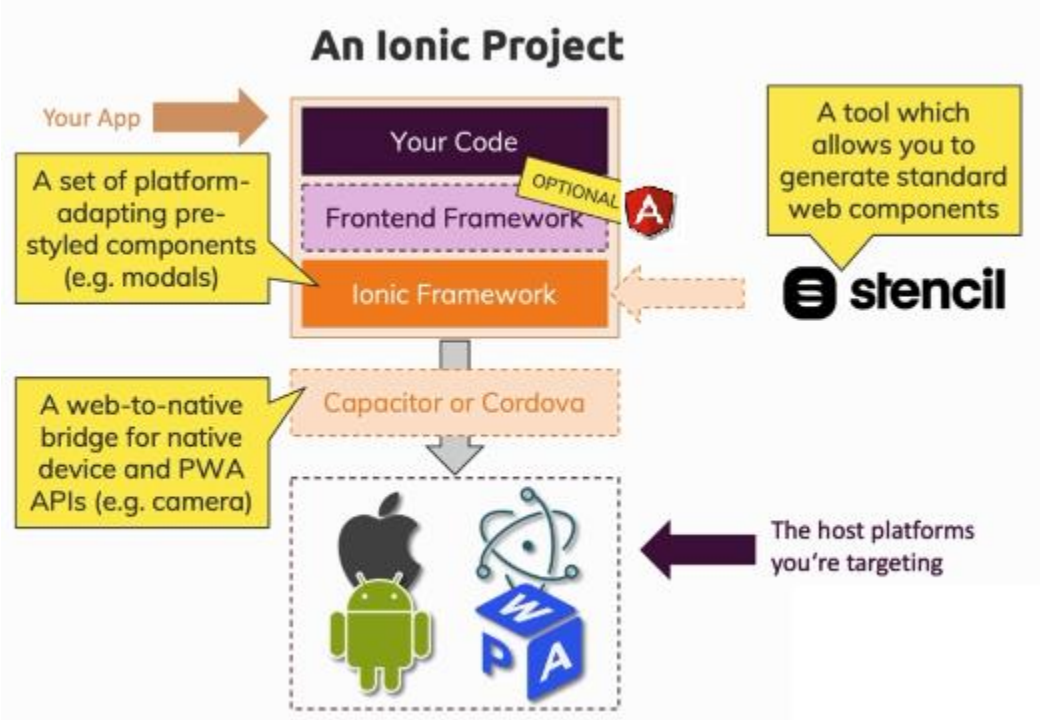




- You can build an app for different platforms with one and the same codebase, with minor adjustments.
- The Ionic platform is all about Ionic which in its core is a set of web components.
- Now web components on the other hand are a technique, a technology supported by modern browsers which allows you 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

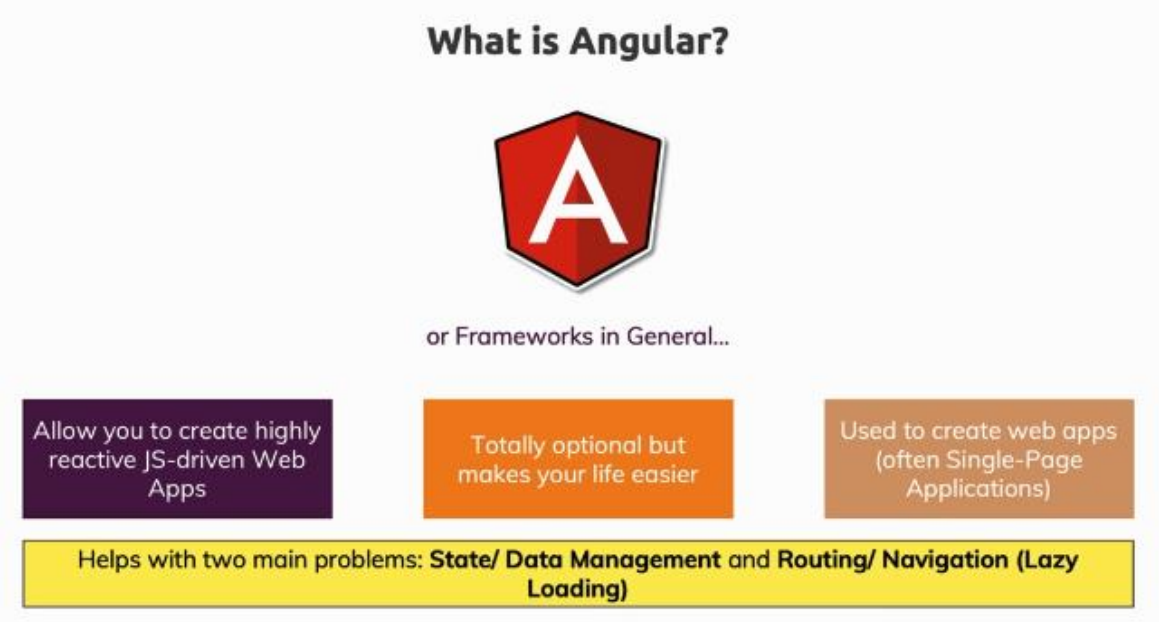




- The Ionic web components are built with a tool called **Stencil**.
- If you want to learn how the Ionic web components were built behind the scenes, or if you want to build your own web components which you can use in conjunction with your framework and with Ionic, then learning Stencil could be well 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

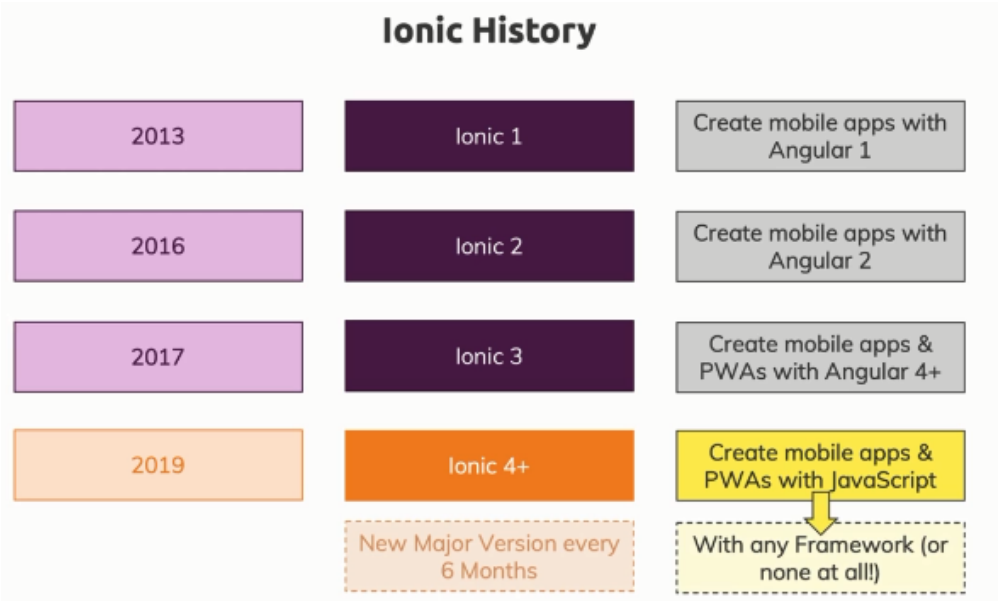
certain functions which then in the end trigger native code on your real device to for example open the camera, to get the user location, to show an alert or anything like that. This can then be done with real native code executed on your behalf without you needing to write it by tools like Capacitor or Cordova.

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

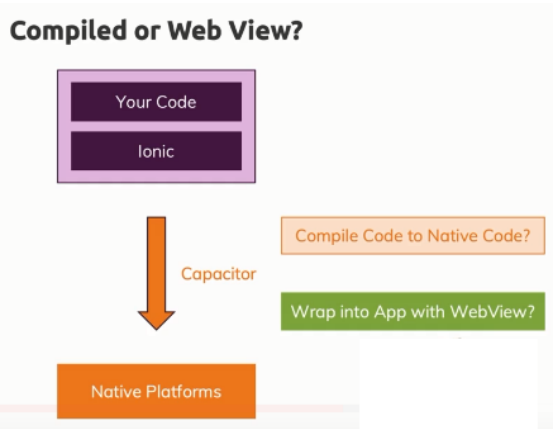


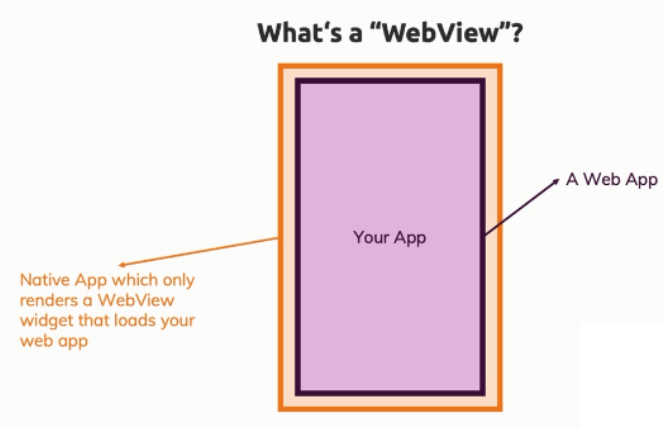
The History of Ionic

- 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





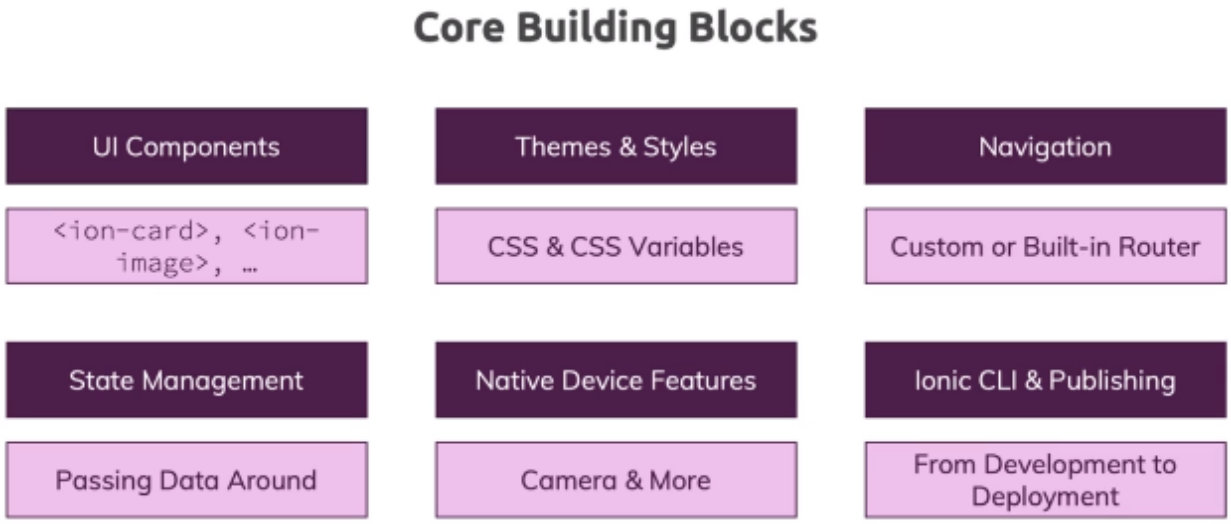
- On native mobile apps, you can launch a web view in the application to host a web page inside of that app.
- **Web view** in the end is a special widget you can use a native app development that is a fully fledged browser that doesn't look like one because you don't have a URL bar at the top and so 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/

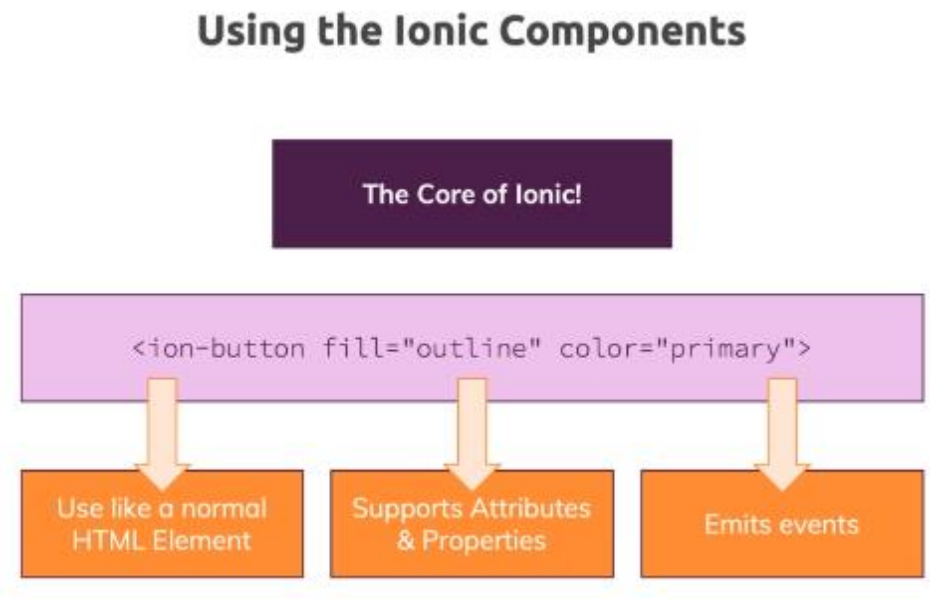
Ionic Component Basics

Core App Building Blocks



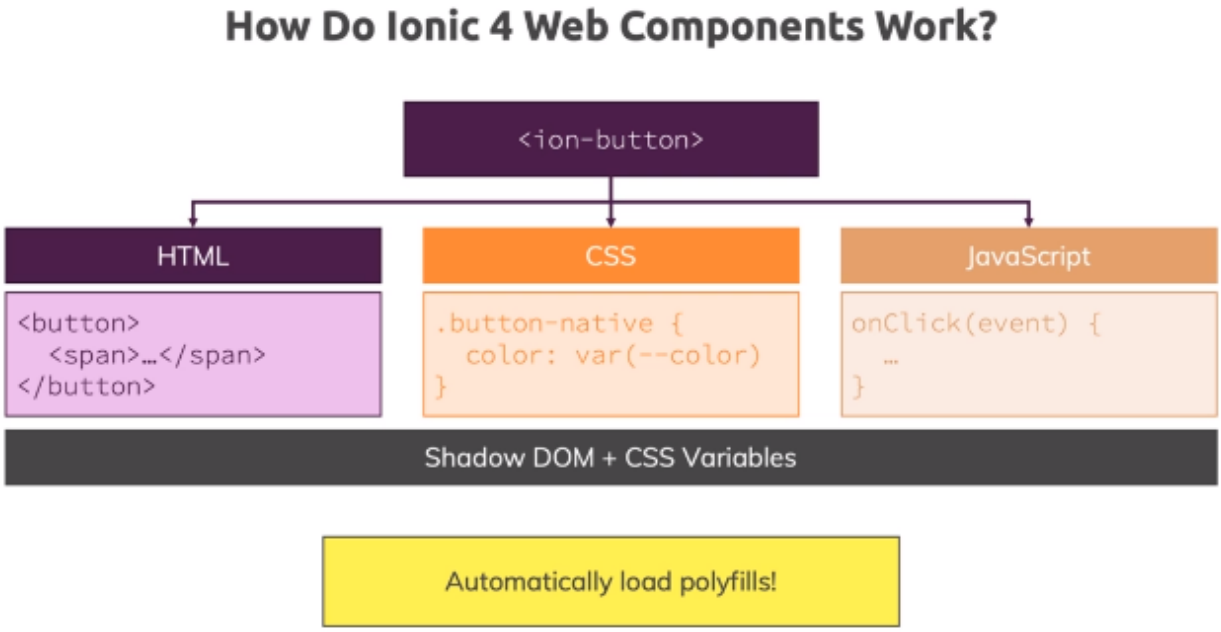
Under the Hood of Ionic Components

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.

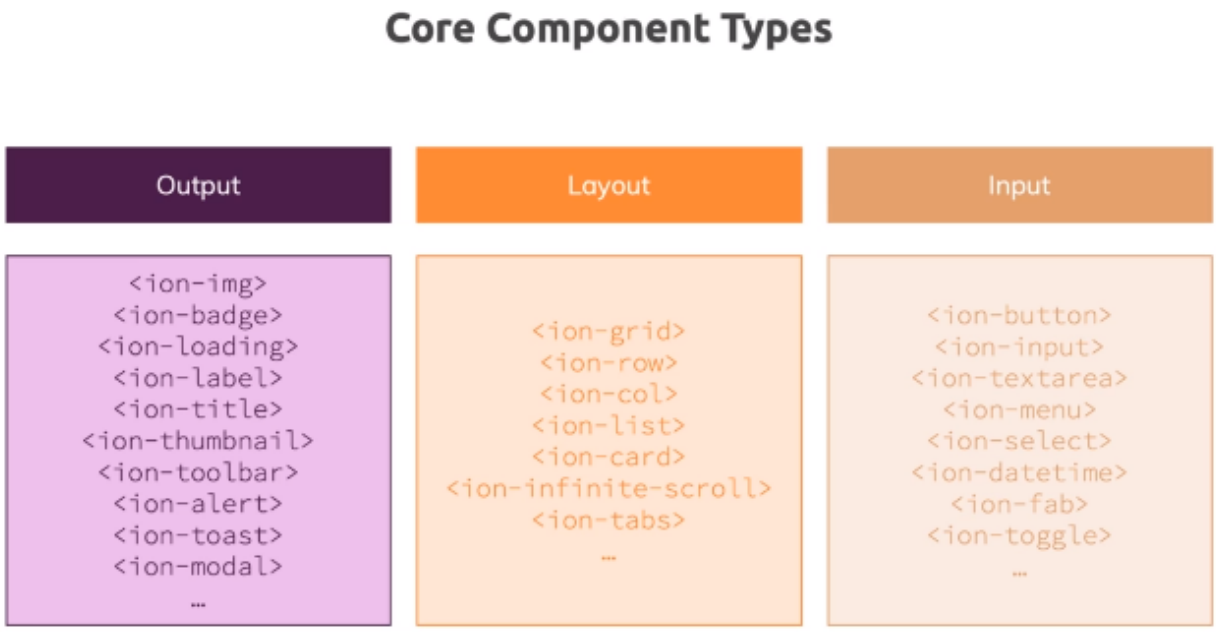
Setting Up a Non-Angular Ionic Project

- 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

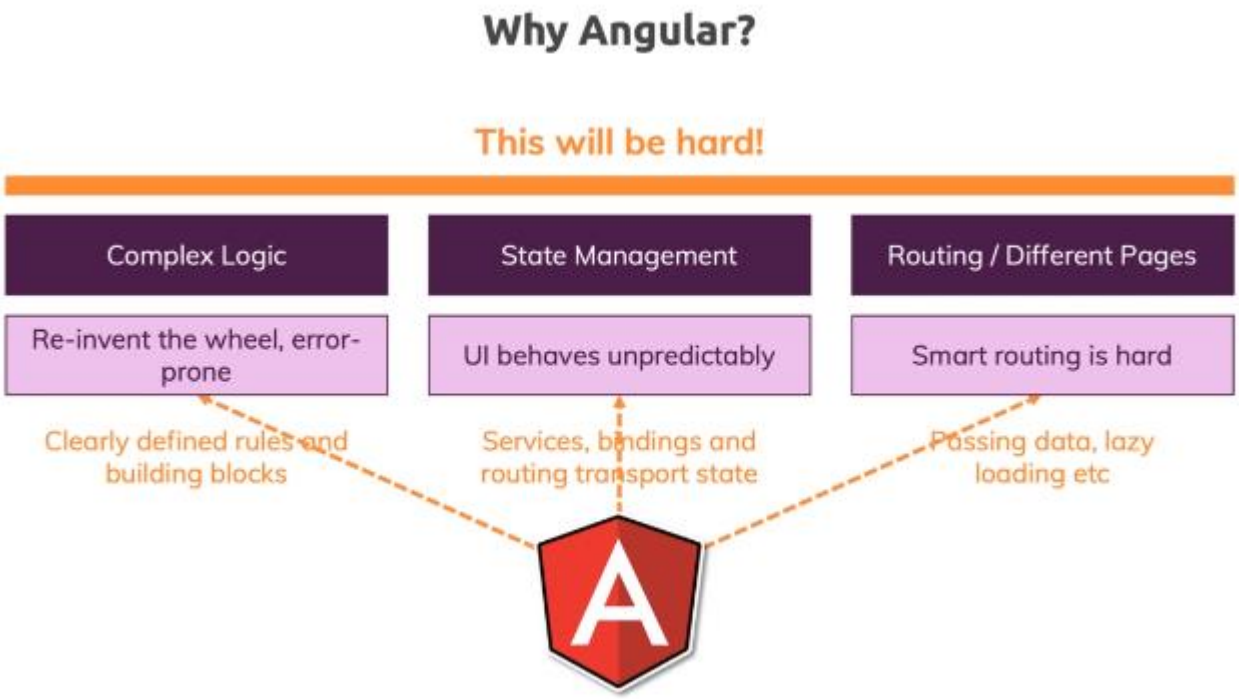
- Slots are default web component concept which basically allows web components to reserve certain places in their built-in markup where certain content should be rendered or can be targeted to be rendered.

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.



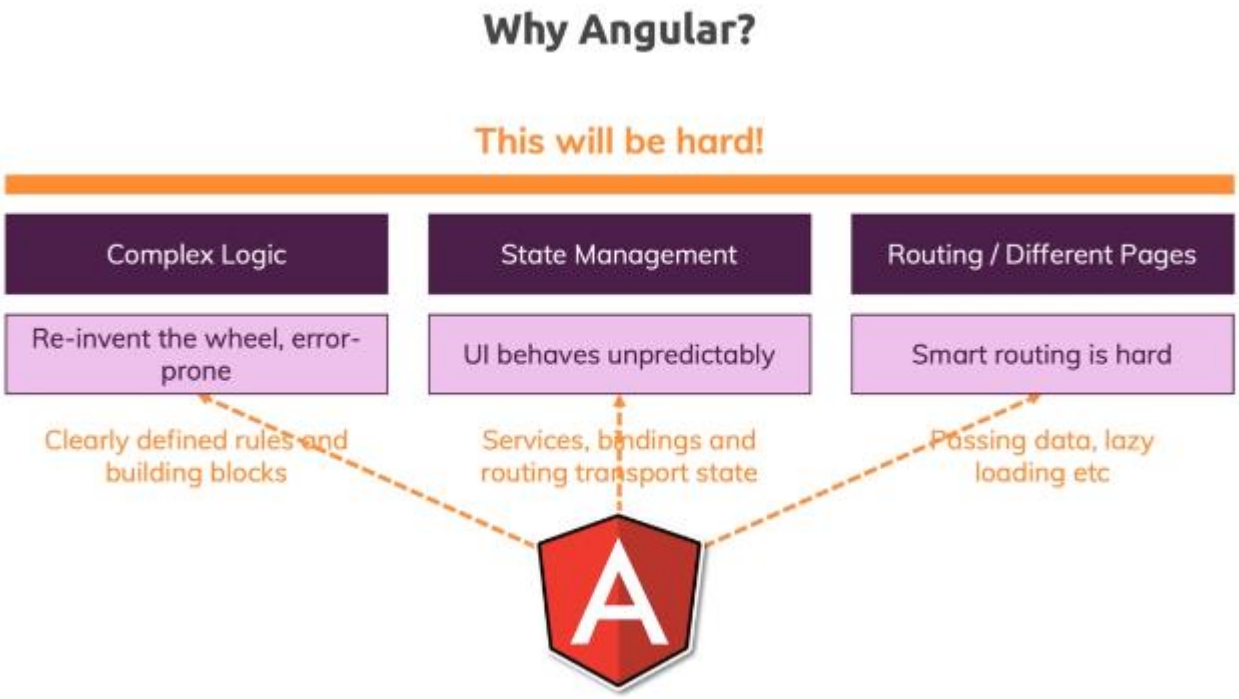
Notes

- Demo projects with vanilla JavaScript –
  - budget-planner
  - 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

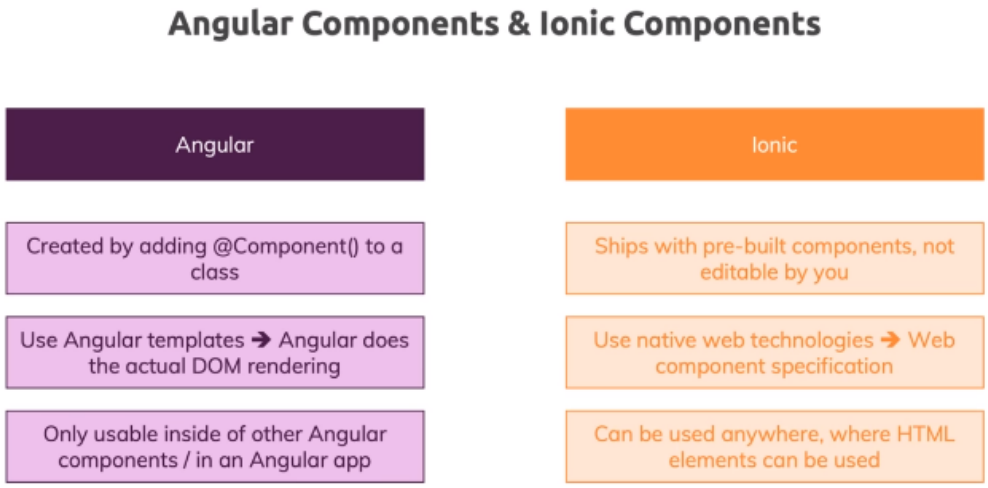
- We can use ng to generate new components but ionic provides better options with "ionic generate" command. E.g. page, service, component, etc.
- With "ionic generate" command and selecting "page", ionic will create a new lazy loading module and will also adjust app routing module. This in the end will 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

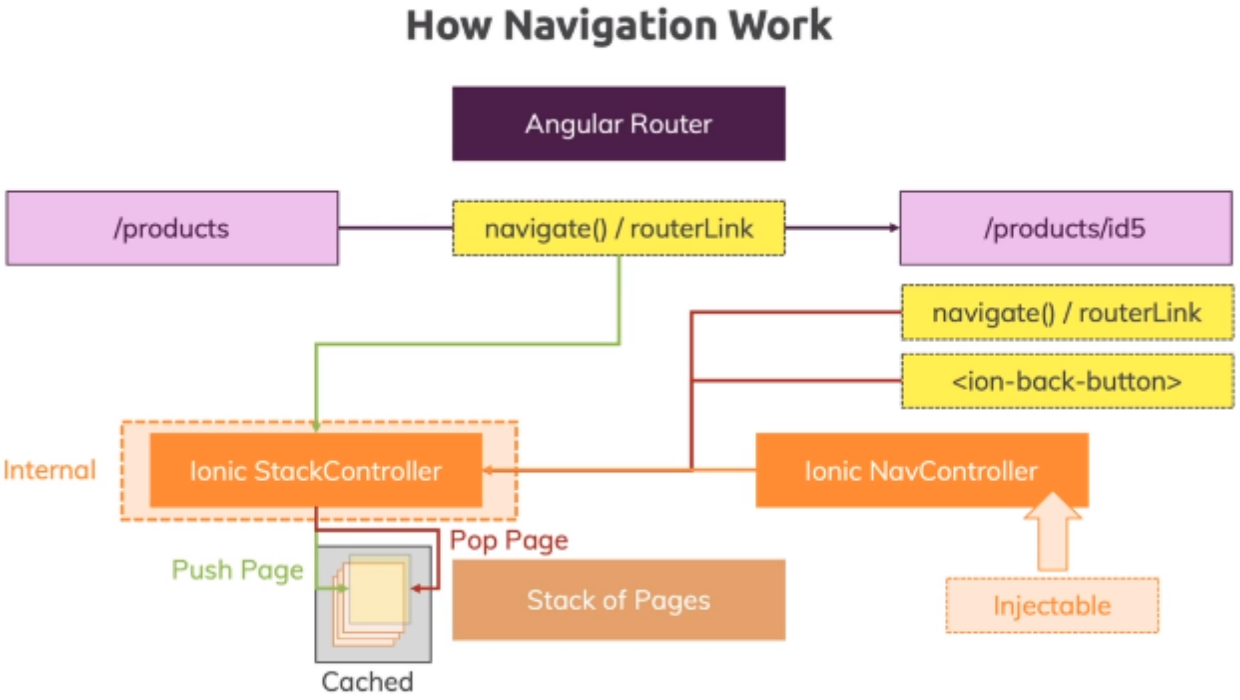
- To see logs, Android Studio -> Run tab
- To debug, chrome://inspect

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 Deep dive Overview

Attributes & Slots

Attributes

- Attributes allow us to place information on HTML elements or on web components that are then consumed by these components and that do something in these components.
- E.g.

```
<ion-item
      *ngFor="let place of loadedPlaces.slice(1)"
       [routerLink]="['/', 'places', 'tabs', 'discover', place.id]"
       detail>
```

  Here 'detail' is simply a property for which Ionic looks out for and when you add it, it knows that it needs to add a tiny arrow icon on the ion-item.
- Though the terms **attribute and property** are used interchangeably, they are not entirely the same. Whatever you place on your element selector, is an attribute. An attribute is typically internally bound to a property of that component and a property is simply like a variable inside of a class.
- Web components and native HTML elements also are based on such classes in the end you could say, they are based on Javascript objects and these objects do have properties and the attributes are just the HTML way of assigning values to these properties. You can also assign values to them by directly selecting the element and then assigning a property in Javascript or in Typescript.

Slots

- Slots are a web component feature and web components is a term of specifications that are built into the browser, so native web features.
- A slot allows you to define a place in your web component where external content can be rendered in.
- E.g.

```
<ion-item
   *ngFor="let place of loadedPlaces.slice(1)"
   detail>
   <ion-thumbnail slot="start">
     <ion-img [src]="place.imageUrl"></ion-img>
   </ion-thumbnail>
```

  Slots are basically your way of telling Ionic where in this item to put a certain content and this ensures that the thumbnail content gets placed all the way on the start of this item to be precise.

Ionic Grid Basics

- IMP Refer – https://ionicframework.com/docs/layout/grid
- Ionic grid is a crucial component that helps you with ordering and lay outing your complex user interfaces.
- The Ionic grid layout uses the CSS flexbox specification under the hood.
- The Ionic grid layout is all about here components working together – ion-grid, ion-row, ion-col
- The height is determined by the content that goes into the rows and each row is just as tall as its highest child.
- Columns can be sized by you. The default number of columns in ionic grid are 12.

Controlling Grid Column Sizes

- Using size and offset attributes.
- E.g.

```
<ion-grid>
    <ion-row>
      <ion-col size="4" offset="2">Row 1 Col 1</ion-col>
      <ion-col size="4">Row 1 Col 2</ion-col>
    </ion-row>
  </ion-grid>
```

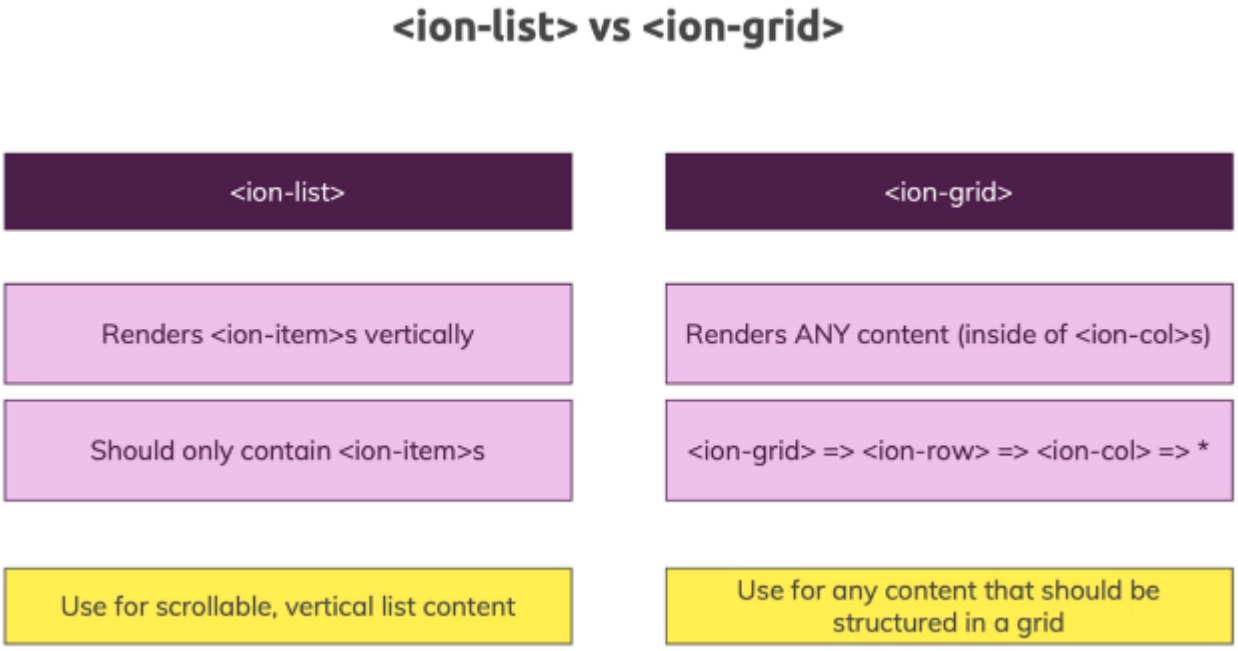
Controlling Grid Alignment

- The items in the grid are as high as their content requires them to be unless you set an explicit height for the row.
- We can control how this should be positioned vertically. The default is stretch.
- How to Set?
    - Vertical Alignment: https://ionicframework.com/docs/layout/grid#vertical-alignment
    - Horizontal Alignment: https://ionicframework.com/docs/layout/grid#horizontal-alignment
- These in the end just sets CSS Flexbox properties.
- **If you are building a grid that should look and behave differently on different screen sizes, you'll have an easier time when using the offset attribute compared to using the CSS utility classes for horizontal alignments**.

Responsive Grid Sizing

- Ionic Responsive Grid: https://ionicframework.com/docs/layout/grid
- Grid Size: https://ionicframework.com/docs/layout/grid#grid-size
- Default breakpoints: https://ionicframework.com/docs/layout/grid#default-breakpoints
- ion-grid 'fixed' attribute means the grid will have a fixed width based on the screen size. This can be helpful if you have a page where at some point, you don't want to take the full available width.

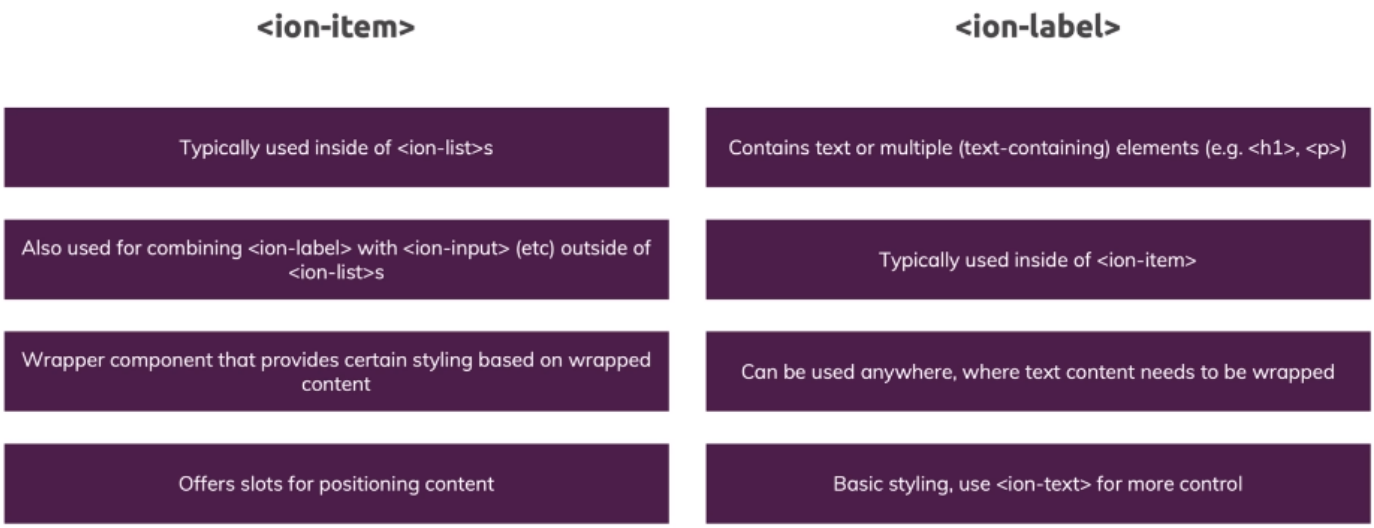
ion-list vs ion-grid



- ion-list also helps you layout items on a page.
- Ion-list has a very narrow purpose. It's there to help you render ion-item, so really this one component which in turn might hold any kind of content **vertically** (from top to bottom).
- **When to use ion-list?** Whenever you have a scenario that you simply want to structure content from top to bottom and you don't need specific control across its width, then you can use an ion-list.
- You can wrap your ion-list with your grid because the list always takes the full width of its container and if the container is the grid or a grid column to be precise, then you can control the sizing of the list through the grid.
- Inside ion-col, you can have any content you want.

ion-label & ion-item

- You use ion-label typically inside of ion-item to wrap any text related content, or text wrapped into other elements. It's just a good practice to use an ion-label and it will also optimize some stylings.
- You can use ion-item inside or outside of an ion-list. ion-item can be used along with ion-label and ion-input for forms.
- ion-item a wrapper component that provides certain styling based on the wrapped content.



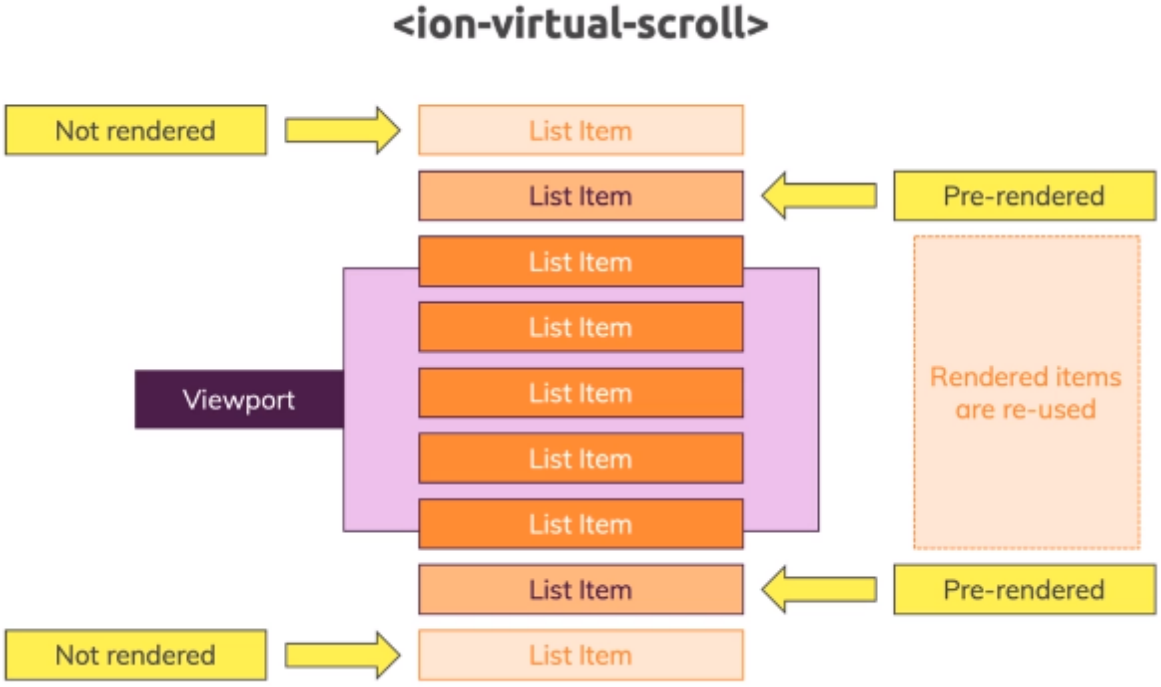
ion-text

- It is a good practice to use ion-text whenever you are simply having the goal of styling some text.
- If you just want to change the color of a text, use ion-text.
  If you want to have multiple text elements, multiple h1 and paragraphs in an ion-item or working with form controls, use ion-label instead.

Swipeable List Items

- These three – ion-item-sliding, ion-item-options and ion-item-option are related to making the ion-item swipeable so that we can expose extra options to choose from, like editing or deleting.
- ion-item-divider and ion-item-group just as you also have ion-list-header are components you can use to split your list into multiple sections.

Understanding Virtual Scrolling (Infinite Scrolling)



- The concept of virtual scrolling is you basically don't render items that are far away (e.g. 3-4 items away from viewport) from being visible.
- You also have certain items which are a kind of pre-rendered, so they are fully rendered even though they're not visible yet but you're likely to scroll to them soon because they're close to your viewport and therefore they are rendered so that when you do scroll down, they are already there and you don't see them pop in.
- And then whenever you scroll, items that are not visible are basically removed and items that are visible are added and they are added in advance to provide smooth scrolling and to not hit the DOM too much and render and remove too many items at the same time.
- overall, it's a performance optimization which can actually be a disadvantage if you use it on very short lists because then you have all that extra logic running behind the scenes that checks whether you're scrolling and what items need to be changed.
- If you have long lists of unpredictable length, so where you don't know if it'll be 50 or 100 items long, then virtual scrolling can really be helpful.
- Refer Details - https://ionicframework.com/docs/api/virtual-scroll

Adding image elements

- **ion-img** element is a special Ionic component that in the end will wrap the normal HTML img image tag but will add extra optimizations to it.
- Inside of a virtual scrolling list, if you have items with images, you should always use ion-img because ion-virtual-scroll and ion-img will work together as such that images are loaded efficiently in advance as well. However even outside of a virtual scrolling list, it is strongly recommend to use ion-img element. Why? Because this element loads images lazily, which means only when they are really needed. It will not load images that are not visible or about to become visible.
- **ion-thumbnail** renders a square image for example and **ion-avatar** simply renders a rounded image. So this is simply just some styling that makes your image look good.

Segmented Buttons

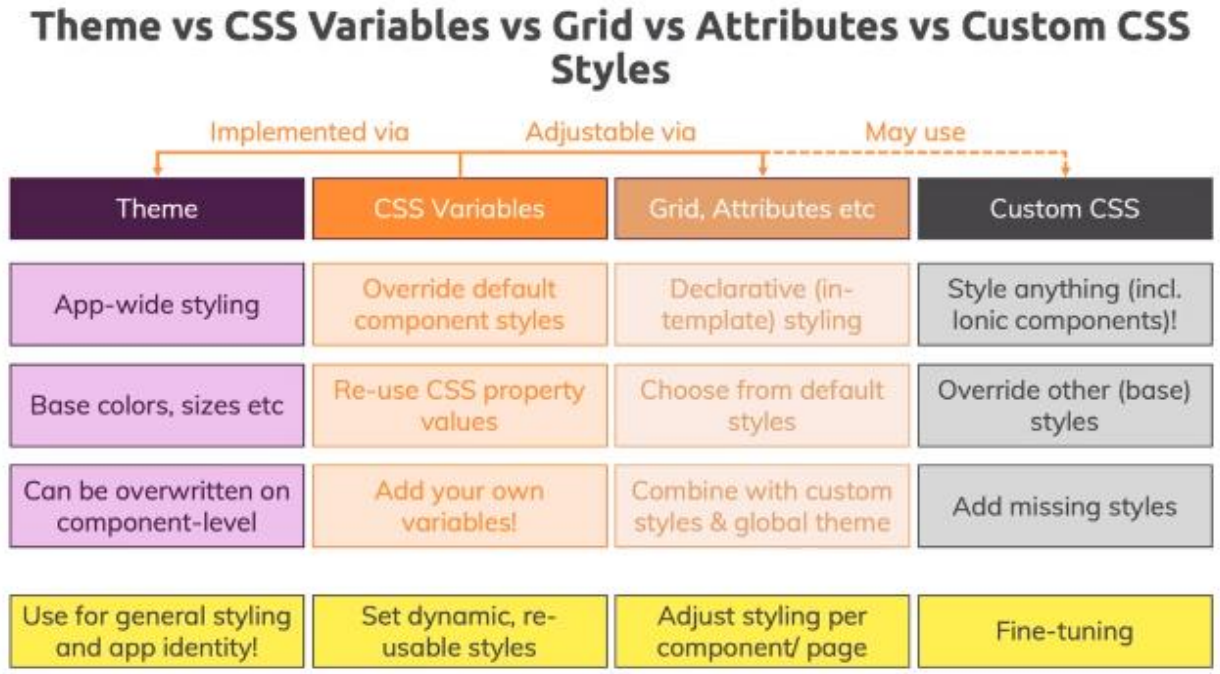
- ion-segment component allows you to add segmented buttons, which means buttons were only one button of the set of buttons can be active at a time.

Controllers

- LoadingController
    - Inject and use LoadingController to have more control over loading spinner. E.g. blocking user interaction till loading finishes.
- ActionSheetController –
    - An actionsheet is basically a set of options that slides up from the bottom of the page.

Styling & Theming Ionic Apps

How Styling & Theming Works in Ionic Apps



- The difference to Sass variables is that CSS variables are baked into modern browsers and therefore you don't need to compile them and you could then even change them at runtime which is not possible with Sass variables.

Docs & Utility Attributes

- Ionic Theming docs – https://ionicframework.com/docs/theming/basics
- Ionic CSS Utilities – https://ionicframework.com/docs/layout/css-utilities

Setting Global Theme Variables

- The default global theme variables are at your codebase **theme/variables.scss**, you can override these as per your requirements.
- So you set up your global variables at **theme/variables.scss**, and Ionic consumes them automatically.
- Ionic Theming Advanced – https://ionicframework.com/docs/theming/advanced
- You can not only set global colors, but also other things like global margin, padding, font, etc.

Setting Global Styles

- **theme/variables.scss** should be used for global CSS variables.
- For other gloval stylings, use **global.scss** file. For component level stylings, use component specific .scss file.

Setting All Colors at Once

- Use Ionic Color Generator tool – https://ionicframework.com/docs/theming/color-generator
- Using this tool, you can select the basic 9 colors (primary, secondary, danger, dark, light, etc.) and then it will generate all other variations of these colors which you can then copy and paste in your theme/variables.scss.
- E.g. primary-contrast, primary-shade, etc. as these are required to style the elements in different situations. E.g. a button may have 'primary' color, but it should have different color (lighter/darker) if you hover over it or click it, etc.

Setting Platform-Specific Styles

- What if you need one these on iOS and other theme on Android?
- The good thing about Ionic is it by default adjusts the styles based on the platform it's running on, which is a big selling point of Ionic because it makes it easy for us to build cross-platform applications with one codebase that still look and feel native-like.
- However sometimes you just want to overwrite something or you want to have your own main color set or a different primary color for a different platform.
- On the root html element, there are some classes applied by Ionic automatically based on the mode or platform it detects. It does this simply by request headers. E.g.
  `<html mode="md"` -> means material design. So Android or desktop.
  `<html mode="ios"` -> means iOS design
- Also ionic sets platform specific CSS classes to this html element. E.g. md, ios, plt-android, plt-phablet, plt-mobile, plt-mobileweb, etc. which you can use to set up our styles. E.g. in theme/variables.scss, you can habe something like –

```
:root {
… // general styles
}
.ios {
… // ios specific styles
}
md {
… // material design specific styles
}
```

- Not only in variables.scss, we can use it at global level or component level as well.

Styling Core Components with Variables

- How to style a specific component application-wide. For example the ion-toolbar should always have our primary background color.
- Global Application Variables – https://ionicframework.com/docs/theming/advanced#application-variables

Component-specific CSS Variables

- For each ionic components, you can see which different CSS variables you can set. Just refer to the docs for that component and scroll to the CSS Custom Properties section. e.g. For ion-button, https://ionicframework.com/docs/api/button#css-custom-properties
- You can override it globally in the theme/variables.scss file or component specific .scss file. E.g.

```
ion-button {
   --background: blue, // supported CSS variable for ion-button
   color: white
}
```

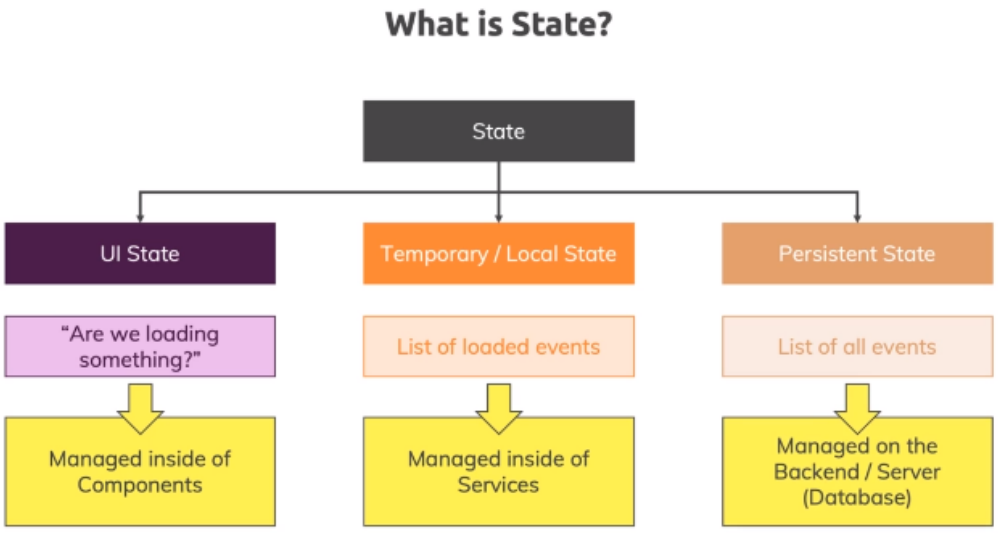
Handling User Input

- ion-input element supports all the form related directives supported by normal input elements. So all the Angular specific behaviors of normal input elements (e.g. highlight on error, etc.) is also supported by ion-input elements.

Managing State

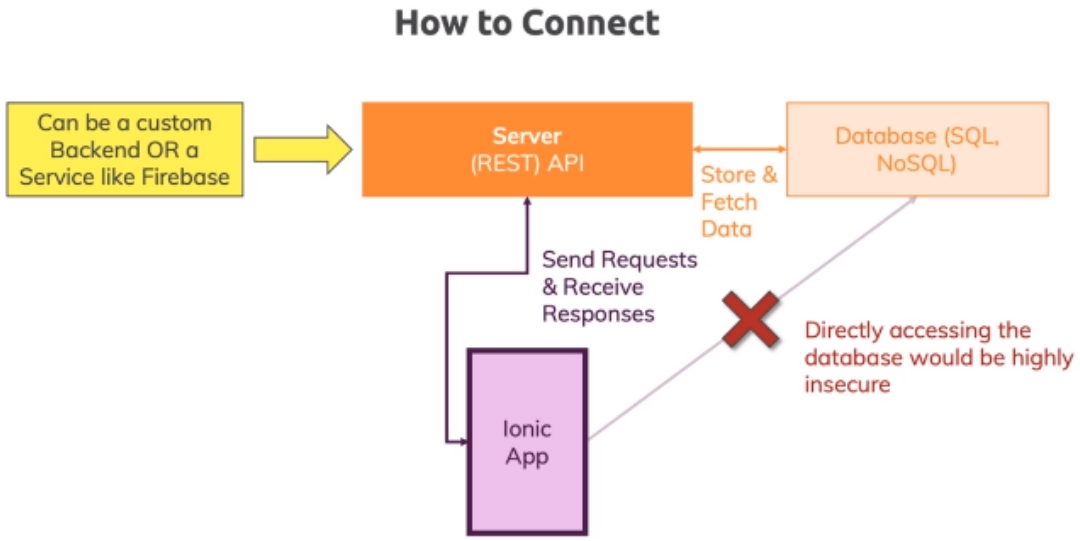
What is State?

- State basically means data.



Sending Http Requests

How To Connect to a Backend



Adding Google Maps

API Setup

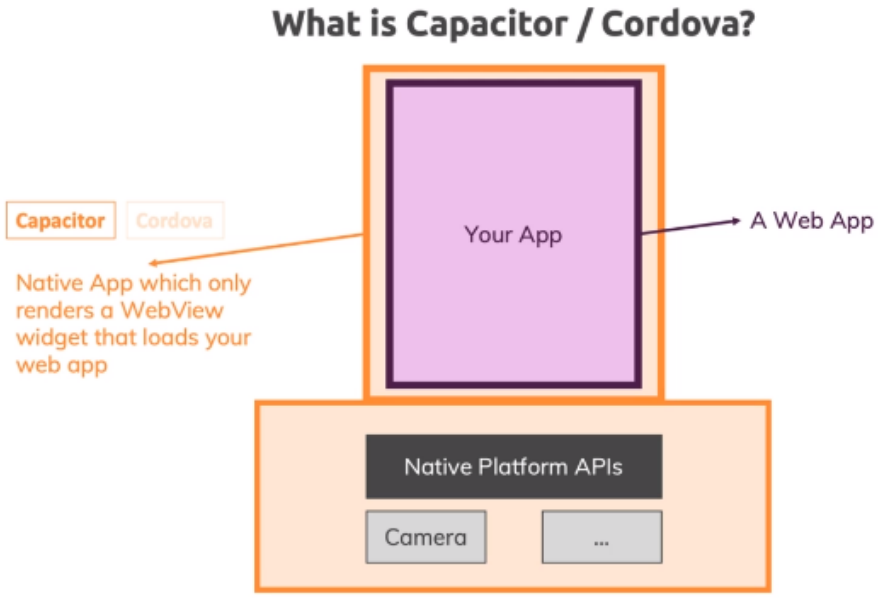
- Google Maps Javascript SDK and API https://developers.google.com/maps/documentation/javascript/overview
- The **SDK** is basically the toolset that allows you to write code that displays and manages the map in your application and the **API** is what works behind the scenes and to which your Google Maps Javascript SDK connects automatically.
- For relatively free SDK, we can explore https://www.mapbox.com/ instead of Google Maps.
- **To use Google Maps for free, pass API Key as empty string. This will load the map but you will see the text over the map as "For Development Purposes Only".**
  e.g. *https://maps.googleapis.com/maps/api/js?key=*

Notes

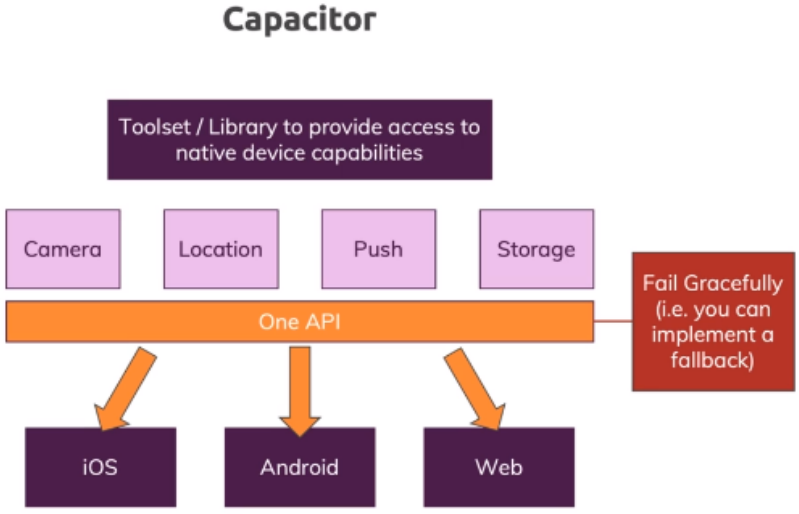
- There are some free Maps API available from some providers. E.g.
  - https://positionstack.com/
  - https://locationiq.com/
  - https://developer.mapquest.com/
  - https://www.mapbox.com/

Using Native Device Features (Camera & Location)

Understanding Capacitor & Cordova



- Capacitor is a tool developed and maintained by the Ionic team and therefore, we can pretty much rely on this being up-to-date and only growing in popularity and feature richness over the time.
- Cordova nonetheless is an alternative to Capacitor and historically, Ionic did use Cordova.
- These (Capacitor, Cordova) tools basically give you a native app shell, take your web app, wrap it into that and then build a real native app out of this shell plus web app thing.
- Capacitor and Cordova also provide a bridge to the native platform APIs. They give us Javascript methods we can call which under the hood will basically translate to native API calls that allow us to tap into that camera and so on.



Using the Docs

- Official Website: https://capacitorjs.com/
- Capacitor Docs: https://capacitorjs.com/docs
- Capacitor Plugins: https://capacitorjs.com/docs/plugins
- Capacitor APIs: https://capacitorjs.com/docs/apis

Using Capacitor Plugins

- The `Platform` service from `@ionic/angular` is a really helpful service that allows you to detect the platform on which your app runs is ready.
- With Capacitor3, we need to import `Plugins` from '`@capacitor/core`' and then to retrieve specific plugins (like the `Camera`), you need to install an extra package and import the plugin from there.
- E.g.

```
>npm install --save @capacitor/splash-screen

import { Capacitor } from '@capacitor/core';
import { SplashScreen } from '@capacitor/splash-screen';
// other code ...
SplashScreen.hide()
```

Getting the User Location

- Refer docs for installation, usage and device permissions – https://capacitorjs.com/docs/apis/geolocation
- Sample code -

```
if (!Capacitor.isPluginAvailable('Geolocation')) {
      // Geolocation feature not available, may be due to the platform not supporting it or permisson denied.
      return;
    }
    // it means Geolocation feature available
    try {
      // Gets the current GPS location of the device
      const geoPosition: Position = await Geolocation.getCurrentPosition();
      const coordinates: Coordinates = {
        lat: geoPosition.coords.latitude,
        lng: geoPosition.coords.longitude,
      };
      // create place
    } catch (error) {
      // error handling
    }
```

Using Camera Plugin

- Refer docs for installation, usage and device permissions – https://capacitorjs.com/docs/apis/camera
- Sample code -

```
async onPickImage() {
    // check if Camera feature is available
    if (!Capacitor.isPluginAvailable('Camera')) {
      return;
    }

    try {
      // Camera feature is available
      const photo: Photo = await Camera.getPhoto({
        // image quality. max 100, min 1
        quality: 50,
        // CameraSource.Prompt = will ask use to open camera or gallary
        source: CameraSource.Prompt,
        correctOrientation: true,
        // choose appropriate dimensions
        height: 320,
        width: 200,
        // means that the image is encoded into a string which we then can
convert to a file if we want to, or just use like that.
        resultType: CameraResultType.Base64,
      });

      // set the image base64 representation
      this.selectedImage = photo.base64String;
    } catch (error) {
      console.log(error);
      return false;
    }
  }
```

Detecting the Platform Correctly

- `this.platform.is('hybrid');`
- hybrid is an indicator for whether we're really running the app on a native mobile device or not.
- For Desktop (even if you use mobile simulator) - hybrid = false
- For Real Mobile - hybrid = true

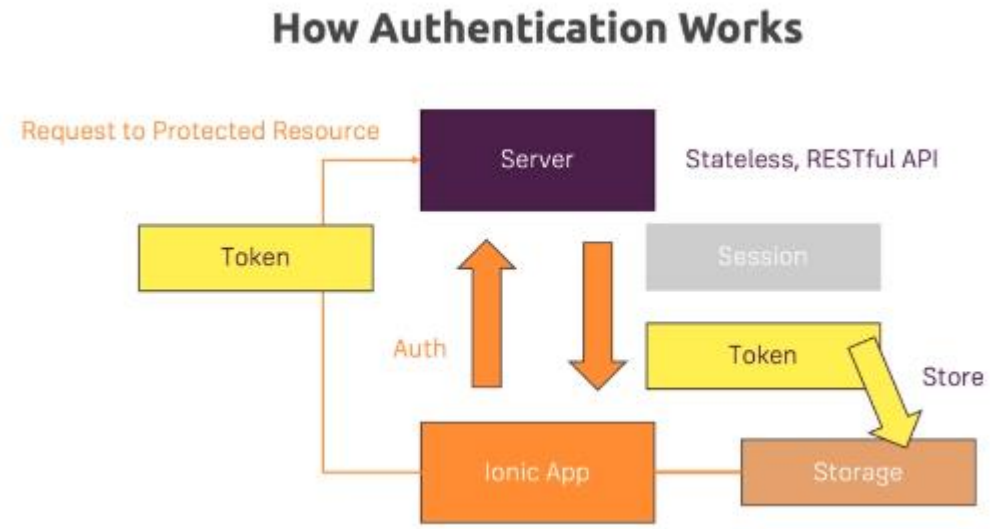
PWA Elements

- Some Capacitor plugins, such as `Camera` or `Toast`, have web-based UI available when not running natively. For example, calling `Camera.getPhoto()` will load a responsive photo-taking experience when running on the web. That UI is implemented using web components. Due to the magic of Shadow DOM, these components should not conflict with your own UI.
- To enable such controls, you must add `@ionic/pwa-elements` to your app.
- Refer all details for PWA Elements – https://capacitorjs.com/docs/web/pwa-elements

Adding Authentication

How Authentication Works

- An Ionic app typically is a single page application.



Authentication using Firebase

- Enable Authentication for you firebase app from firebase console and use the Firebase Auth APIs.
- Refer – Firebase Auth REST API docs – https://firebase.google.com/docs/reference/rest/auth

Storing Auth Data in Device Storage

- We want to store some information like Auth token and we want to store it either in in-browser storage on a Mac or on a PC but on a real device, we want to store it on that real device because if we would use the in-browser storage, that would be available in that wrapped web app the Ionic app is, but it might be cleared by the wrapping app shell and it's less reliable than our on device storage and therefore on the native device, we want to use that on-device storage and Capacitor conveniently gives us access to that storage using the **Storage** plugin.
- Mobile OS's may periodically clear data set in window.localStorage, so this API should be used instead. This API will fall back to using localStorage when running as a Progressive Web App.
- This plugin will use UserDefaults on iOS and SharedPreferences on Android. Stored data is cleared if the app is uninstalled.
- Must Refer – https://capacitorjs.com/docs/apis/storage

Publishing the Apps

Preparing App Configs

- Build the app for prod
  >ng build --prod
- Android: Sync with android folder in the project
  >ionic capacitor sync android
- Open project in Android Studio
  >ionic capacitor open android
- Update configuration as mentioned here
  https://capacitorjs.com/docs/android/configuration

Notes

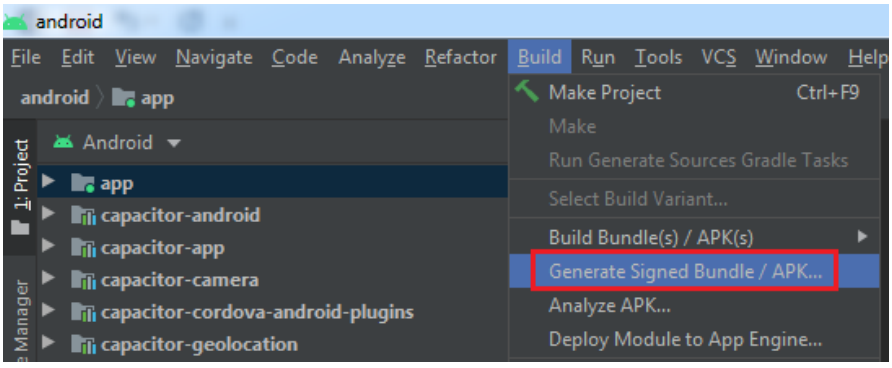
- Refer for Android – https://capacitorjs.com/docs/android
- Refer Android Configuration - https://capacitorjs.com/docs/android/configuration
- Refer for iOS – https://capacitorjs.com/docs/ios
- Refer iOS Configuration – https://capacitorjs.com/docs/ios/configuration

Custom Icons & Splash Screens

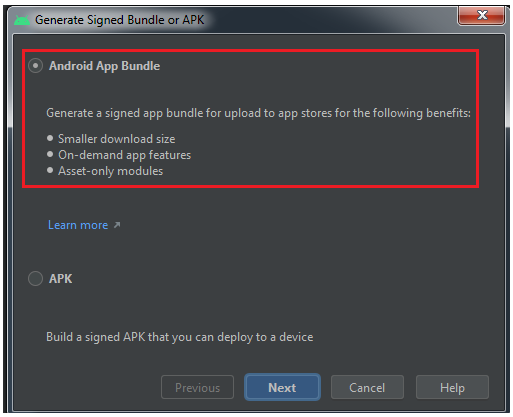
- We need to have App icons and splash screens for different types of devices and orientations.

Android Deployment to Google Play Store

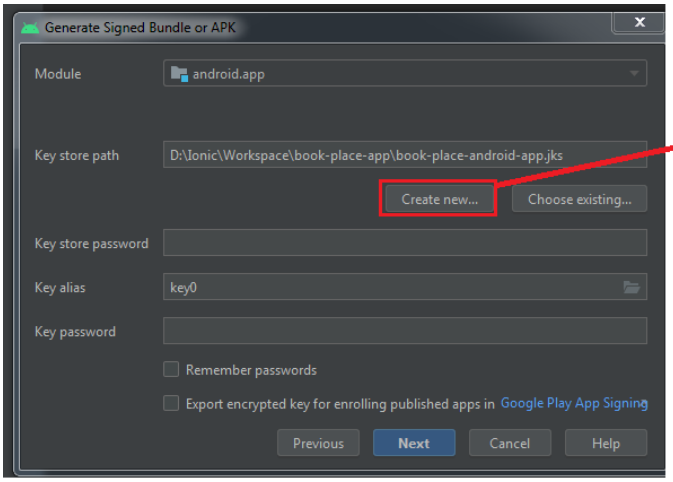
- Make sure you complete the steps in "Preparing App Configs" above.
- Open project (android folder) in Android Studio.
- Open Build -> Generate Signed Bundle/APK...

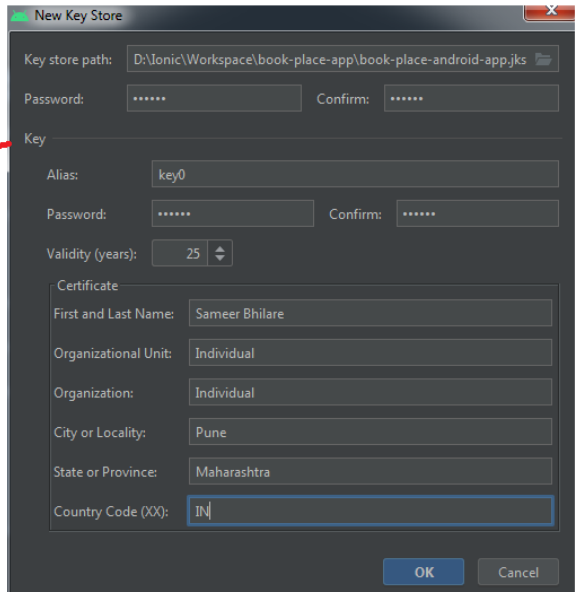


- Select as shown

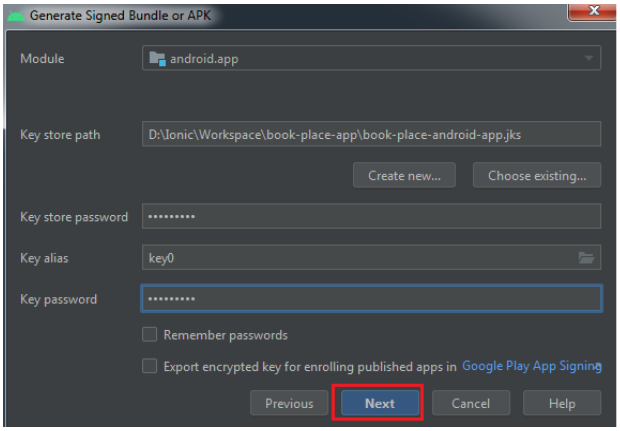


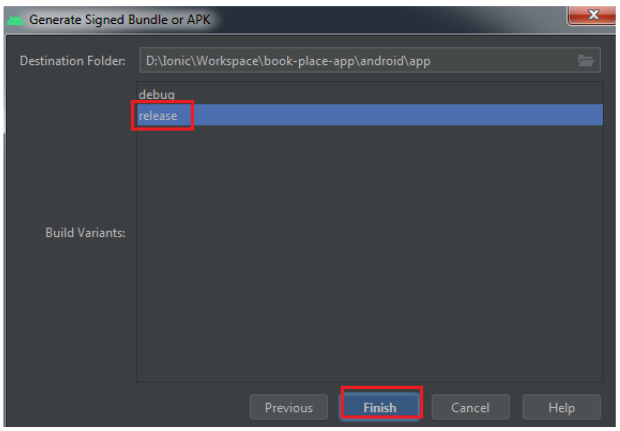
- Fill information as below





- Follow as below





- Now the deployable (.AAB) will be generated under android/app/release/release folder.
- Go to Google Play Console to deploy your app – https://play.google.com/apps/publish
    - You need to pay on time fee for publishing app to Play Store.

Android Deployment and APK file generation (without publishing)

- If you want to just create APK file and install the app directly (without publishing to Play Store) on native device, then just generate APK file as Build -> Generate Signed Bundle/APK... -> Select 'APK' option and then follow as usual as shown above.
- This will generate the APK file in android/app/release/release folder which you can use to install your app on native Android device.

(PWA) Web Deployment

- Run
  > ng add @angular/pwa
- This configures the Angular project such that it automatically generates such a service worker that is responsible for that caching and so on and basically sets up everything you need to have a progressive web app, nothing else needed.
- Now build the app again for prod
  >ng build --prod

Useful Resources

- Ionic iOS Publishing Docs:
  https://ionicframework.com/docs/publishing/app-store
- Ionic Android Publishing Docs:
  https://ionicframework.com/docs/publishing/play-store
- Ionic PWA Publishing Docs:
  https://ionicframework.com/docs/publishing/progressive-web-app

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.
- .cancat() is default array method which adds an item to the array and returns a new array.
- To use **Google Maps for free**, pass API Key as empty string. This will load the map but you will see the text over the map as "For Development Purposes Only".
  e.g. *https://maps.googleapis.com/maps/api/js?key=*
-