



Dr. Babasaheb Ambedkar Open University

(Established by Government of Gujarat)

PGDMAD-202

Cross Platform Mobile Application Development

Cross Platform Mobile Application Development



Post Graduate Diploma in Mobile Application Development

2019

Cross Platform Mobile Application Development

Dr. Babasaheb Ambedkar Open University



Expert Committee

Prof. (Dr.) Nilesh Modi Professor and Director, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad	(Chairman)
Prof. (Dr.) Ajay Parikh Professor and Head, Department of Computer Science Gujarat Vidyapith, Ahmedabad	(Member)
Prof. (Dr.) Satyen Parikh Dean, School of Computer Science and Application Ganpat University, Kherva, Mahesana	(Member)
Prof. M. T. Savaliya Associate Professor and Head, Computer Eng. Department Vishwakarma Engineering College, Ahmedabad	(Member)
Dr. Himanshu Patel Assistant Professor, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad	(Member Secretary)

Course Writer

Dr. Ankit R. Bhavsar Assistant Professor, Faculty of Computer Application & Information Technology GLS University, Ahmedabad
Mr. Nirav Suthar Assistant Professor, Faculty of Computer Application & Information Technology GLS University, Ahmedabad
Mr. Pratik Maniar

Subject Reviewer

Dr. Himanshu Patel Assistant Professor, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad
--

Editors

Prof. (Dr.) Nilesh Modi Professor and Director, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad
Dr. Himanshu Patel Assistant Professor, School of Computer Science, Dr. Babasaheb Ambedkar Open University, Ahmedabad

June 2019, © Dr. Babasaheb Ambedkar Open University

ISBN-978-81-940577-6-5

All rights reserved. No part of this work may be reproduced in any form by mimeograph or any other means, without written permission from the Dr. Babasaheb Ambedkar Open University.

Printed and published by: Dr. Babasaheb Ambedkar Open University, Ahmedabad



Cross Platform Mobile Application Development

Block-1: Scenario of Mobile Application Development

UNIT-1 Cross Platform Mobile Application Development	02
Unit-2 Basic of Development Environment-Angular	16
Unit-3 Basic of Development Environment-IONIC	29

Block-2: Working with Angular

UNIT-1 Introduction to Angular	41
Unit-2 The Basic of Angular	57
Unit-3 Introduction to MVC	77
Unit-4 Angular Directives	96
Unit-5 Working with Forms	112

Block-3: Working With IONIC

UNIT-1 Setting Up the Environment for IONIC	134
Unit-2 Developing First Mobile Application	146
Unit-3 Typescript	167

Block-4: Advance of IONIC

UNIT-1 Ionic UI Controls	183
Unit-2 Advanced Components	203
Unit-3 Advanced Topics in IONIC	223

Block-1

**Scenario of Mobile Application
Development**

Unit 1: Cross Platform Mobile Application Development

1

Unit Structure

- 1.1 Learning Objectives
- 1.2 Introduction
- 1.3 Mobile Application Development
- 1.4 Native Mobile Application Development
- 1.5 Cross Platform (Hybrid) Mobile Application Development
- 1.6 Basic Requirement For The Cross Platform Mobile Application Development
- 1.7 Let us sum up
- 1.8 Check your Progress
- 1.9 Check your Progress: Possible Answers
- 1.10 Further Reading
- 1.11 Assignments

1.1 LEARNING OBJECTIVES

After studying this chapter, students should be able to understand:

- Android OS app development environment
- iOS app development environment
- Window Phone app development environment
- Native mobile application development concept
- Concept of BYOD
- Cross platform (Hybrid) mobile application development concept
- Tools / Framework available for the cross platform mobile application development

1.2 INTRODUCTION

As per the survey by the leading statistics portal (sttista.com), in 2019 the number of smart phone users is forecasted to reach 4.68 billion and will cross cross 5 billion at the end of the year 2019. This number represents 66 percentage of world population. It shows the use of smart phone is growing rapidly. The applications play a prominent role in smart phone's growth.

The increasing popularity of smart phone is rooted with mobile applications. The massive number of mobile applications is available for different mobile operating systems. By using applications, the smart phone helps their user to do many things. They can set alarms, get reminders, find location, do online shopping, booking and many more. More than five million apps are available on leading app store till the end of the year 2018. This statistics declare by the leading statistics portal – statista.com. As per the portal total 2.1million apps available on android play store and 2 million apps available on apple's app store. Remaining apps are available on window store, Amazon store and BlackBerry World.

When creating apps for different smart phone's operating systems, more time is required for programming because each mobile platform is based on a different programming language. This means that companies must use different experts to develop applications for each platform. To overcome the said problem, we need to develop hybrid mobile application. The hybrid mobile application development

supports developers to build applications for multiple mobile platforms at the same time. In this book we will learn hybrid mobile application development using IONIC. This unit discuss about the basic of mobile platform environment. We will also learn the native mobile application development and cross platform (hybrid) mobile application environment.

1.3 MOBILE APPLICATION DEVELOPMENT

1.3.1 ANDROID

1.3.1.1 Development Environment

Android is an open source operating system. It is based on Linux operating system. Android was developed by Open Handset Alliance (OHA) led by Google. The vision is to provide a robust and open source environment for wireless platform. Android provide a uniform approach for its developer. It means android developer needs to develop mobile applications for the android and their applications able to run on any device powered by Android.

The Android platform is itself complete, open and free for the mobile platform.

Complete: The android developers take a comprehensive approach to develop secure operating system called Android. On the top of this they provide robust software framework that allows for rich mobile application development environment.

Open: The android is an open source platform. Neither application developers nor device manufactures need to take any license for the android platform.

Free: The android applications are free to develop. There is no licensing, membership, testing fees for the android platform.

Android applications are developed in Java language using the Android Software Development Kit (SDK). The first beta version of the SDK was released by Google in 2007. The first commercial version, Andorid 1.0 was released in 2008. There after various versions like Cupcake, Donut, Éclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean,

KitKat, Lollipop, Marshmallow, Nougat, Android Oreo were published. The latest version of Android is “Android 9.0: Android Pie”.

Android application development is supported by the following operating system:

- Microsoft Window XP and later version
- Mac OS X 10.5.8 and later version with Intel chip
- Linux including GNU C library 2.7 and later

1.3.1.2 Tools and Technology for Android App Development

The tools that required developing Android applications are freely available on internet. We can download the tools and set up the Android application development environment. The detail installation steps are not scope of this book so we are not discussing it in detail. The following tools we need to install before starting the Android application programming.

- Java JDK 6
- Android SDK
- Android Studio or Eclipse
- Android Development Tools (ADT)

1.3.2 iOS

1.3.2.1 Development Envirement For iOS

The iOS is the mobile operating system developed by the Apple Company. It is developed to support company’s own devices like iPhone, iPad, iPod Touch and Appel TV. The iOS is derived from the OS X. The OS X is the operating system used in Apple computers.

The iOS, earlier known as iPhone OS, is a mobile operating system for Apple mobile. The first version of the iOS was released in 2007 with the name “iPhone OS”. It was officially renamed as “iOS” in 2010. The most recent stable release, iOS 12.2 was released on March 2019.

The iOS SDK (Software Development Kit) is used to develop iOS applications. The iOS SDK was developed by Apple Company. The iOS SDK

is a free download for the only Mac operating system users. It is not available for the Microsoft Windows operating system users. The SDK contains sets that give developers access to various functions and services of iOS devices, such as hardware and software attributes. The iOS SDK clubbed with Xcode IDE.

1.3.2.2 Tools And Technology For iOS

The tools that required developing iOS application is Xcode. The Xcode is an integrated development environment (IDE) for iOS. It contains software development tools developed by Apple for developing application for iOS. The Xcode helps developers to write iOS applications using programming languages such as Objective-C and Swift.

Objective-C is a general-purpose and object-oriented programming language. It was the main programming language supported by Apple for the iOS operating systems. The Objective –c was originally developed in the 1980 then Apple used it for its NeXTSTEP operating system, from where iOS are derived.

The Swift is the general purpose and compile programming language. It was developed by the Apple Company for iOS in 2014. The Swift is designed to work with Apple's Cocoa touch frameworks. The Cocoa touch is a user interface framework for building applications that run on iOS. The latest version of Swift is "Swift 4.1" released in March 2018. Swift is an alternative to the Objective-C language that employs modern programming-language theory concepts with simpler syntax.

1.3.3 WINDOW PHONE

1.3.3.1 Development Environment For Windows Phone

The Windows phone is an operating system developed by the Microsoft. The latest version of Windows phone is "Windows 10 Mobile" released in 2015. The Windows 10 Mobile is the successor of "Windows Phone 8.1". The Windows 10 Mobile operating system is available commercially with Lumia brand smart phones from 2015. The aim of Windows 10 Mobile is to provide new universal

application platform that allows one application to run on multiple Windows 10 devices such as personal computers and mobile

The Windows Phone platform never achieved any significant degree of popularity or market share in comparison to Android or iOS. By 2017, Microsoft had already begun to depreciate Windows 10 Mobile, having discontinued active development due to a lack of user and developer interest in the platform. Windows 10 Mobile will be deemed to end-of-life on December 2019.

1.4 NATIVE MOBILE APPLICATION DEVELOPMENT

1.4.1 DEFINITION

A native mobile application is “a smart phone application this is coded in a specific programming language for specific mobile operating system”. A native mobile application development is specific for a mobile operating system or device. That means before actual development of mobile application, developer needs to finalize that on which mobile operating system or device it will be executed. Native application is built for the use on a particular mobile operating system, it has the ability to use device specific hardware or software. Native apps work with the device's OS in the way that enables them to perform faster and more flexibly than alternative application types. Native applications can be either installed on the specific mobile by default or downloaded from Mobile OS specific app store. Figure-1 shows the native mobile application development approach.

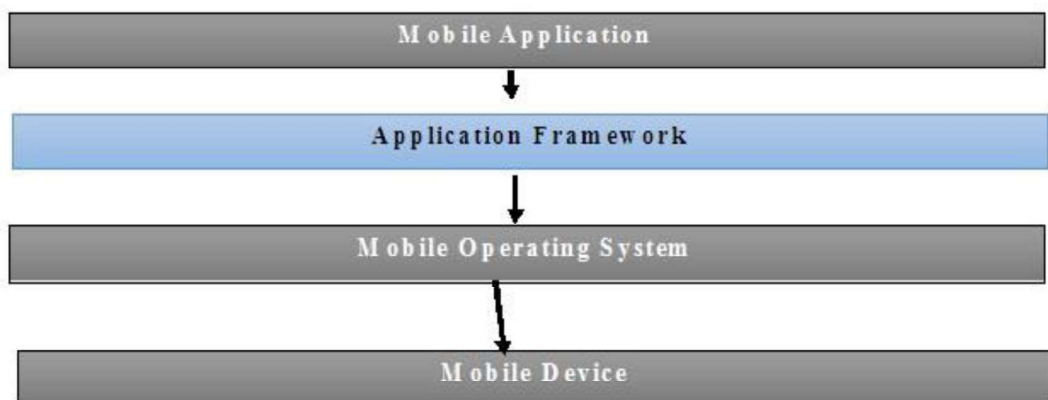


Figure-1 Native Mobile Application Development Approach

A native app is specially made and coded for a specific mobile platform in its native programming language. The Swift and Objective –C language are used to develop mobile applications that are supported by the iOS (Apple) phone. The Java or Kotlin language is used to develop mobile applications that run on the android OS based mobile. The C# language is for Window 10 Phone. These all are the example of native mobile application development.

1.4.2 BYOD STRATEGY

The native mobile application development bring the trend of BYOD (Bring Your Own Device). The BYOD refers to the policy of permitting employees to bring their own mobile device to their workplace. The company needs to provide access privilege to access company information and application on their mobile. We can find many companies that provide mobile application to their employ through which they can share company information. This kind of mobile applications mostly runs on specific mobile operating system. Here it is compulsory that company employees have mobile devices that have same mobile operating system.

The BYOD has some advantages. Employees can be more productive by adopting BYOD strategy in companies. The BYOD increases employee's satisfaction and job satisfaction. The main disadvantage of adopting the BYOD in the company is that they need to develop same application for the multiple mobile operating system. The illegal way of information discloser is second disadvantage to adopting BYOD strategy.

1.4.3 ADVANTAGES AND DISADVANTAGES OF NATIVE MOBILE APPLICATION

Advantages

- The developers code the application for the specific mobile OS. So it runs smoothly over it.
- Native applications achieve higher marks in speeds and performance.
- In native apps, the look n feel and experience are much than other types of mobile app.
- Native apps offer fast access to inbuilt device utilities like the camera, GPS, calendar, microphone, and other functions of the smartphone.

Disadvantages

- Native apps need more time to develop. Creating and implementing the design for every device takes more development time.
- Developers usually have specialization in a single platform. To develop a native app, we need as many development teams as the platforms on which we want the app to be created. Multiple development teams imply multiple budgets.

1.5 CROSS PLATFORM (HYBRID) MOBILE APPLICATION DEVELOPMENT

1.5.1 DEFINATION

“A mobile application which can be developed for multiple platform by using single codebase is called cross platform mobile application.” Here multiple platforms stand for the any mobile operating system or devices. A cross platform mobile application also called Hybrid mobile application. Hybrid mobile application development is required some specialized code.

A hybrid application is created as a single app for the use on multiple platforms like Android, iPhone, and Windows phone. It is a single product that works on many operating systems like iOS, Android, Windows phone etc. Hybrid applications perform differently compare to native apps in several ways.

The hybrid application is of two types:

- A Web View – Based Hybrid Mobile Applications
- Cross – Compiled Hybrid Mobile Applications

A native mobile application which runs the web applications by using a Web View is called Web View- Based Hybrid Mobile Applications. Generally Web View based hybrid applications are developed using HTML5, Java Script and CSS3. It has numbers of Java Script based gesture detection library to handle touch interaction on screen. It look and feel like a native application but is actually run by website, It is basically a web-based program to be put in a native app shell and connected to the device hardware. The IONIC, Cordova, Angular are some of the example of Web

View –based hybrid application development. Figure-2 shows the Web-View based hybrid mobile applications development approach.

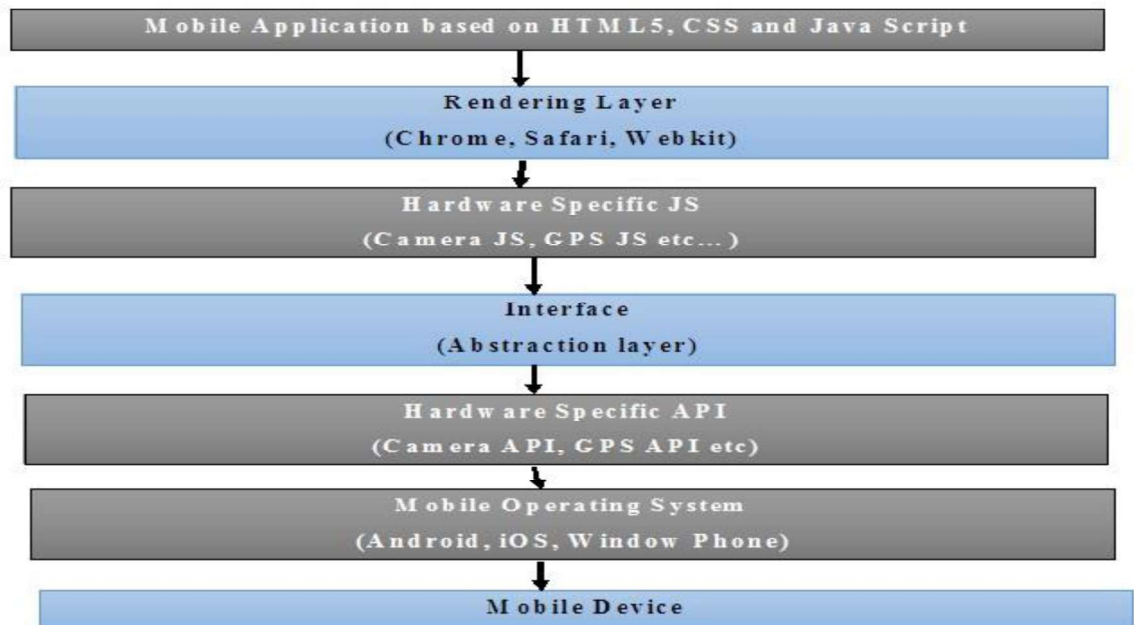


Figure-2 Web-View based hybrid mobile applications development approach

A Cross Compiled Hybrid Mobile Applications development allows developers to convert a specific single language into native language component at either compile time or during run time. It acts as an interface between native components and constructor in the concern programming language.

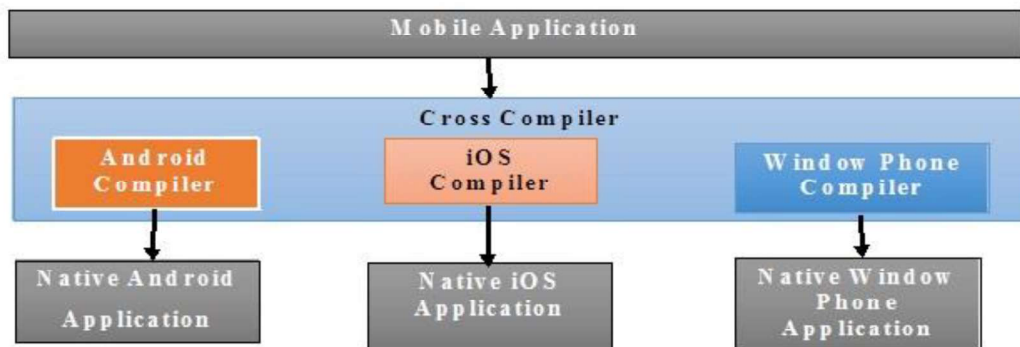


Figure-3 Cross-Compiled hybrid mobile applications development approach

Figure-3 shows the Cross-Compiled hybrid mobile applications development approach. Here cross compiler tool binds their API with native API, therefore the

performance and user experience can be achieved almost the same as native mobile application. The Xamarin with C#, Corona with C and Kony with Java Script are the example of Cross – Compiled Hybrid Mobile Applications.

1.5.2 MOBILE APPLICATION TECHNOLOGY STACK

Having a next-generation mobile app is the need of every small to large-scale companies. Before the actual working model development of mobile application, mobile app developers and management together to define the design, structure, architecture, features, and functions of the mobile app. Before starting the development, company management and developers have to select the most suitable technologies, platforms, frameworks, and tools for the mobile application. Ideally, it's a job of the mobile app developers. So it is necessary that company's management and app developers have to participate in the process of selecting an appropriate technology stack.

To build a robust mobile app in terms of scalability and performance, we need to make smarter decision related to the technology aspects. We need to select any one mobile application technology stack approach based on requirement. Now we know that mobile application can be develop using any one of the given technology stack.

- Using native mobile application development approach
- Using web-view based hybrid mobile applications development approach
- Using cross-compiled hybrid mobile applications development approach

When a high performance is necessary, native mobile application development approach should be selected. It is a bit expensive approach but the user experience and security is high compare to other approaches. In the competitive age, each company reaches to customer speedily. The web-view based hybrid mobile application development approach provides the speedy mobile application development environment. In this approach we need to code once and run on the multiple mobile platforms with less cost. When we want to take advantage of native mobile application and web-view mobile application, we need to select cross-compiled hybrid mobile applications development approach. As we know that

in this approach, mobile application developed in single specific language and at compile time (or run time) it converts it in to mobile specific native code.

1.5.3 ADVANTAGES AND DISADVANTAGES OF CROSS PLATFORM MOBILE APPLICATION

Advantages

- Hybrid mobile applications are developed once for all platforms. So we do not require to hire different programmer. Due to that the development cost is very low.
- As we know hybrid apps are web apps incorporated in a native shell, so its content can be updated as many times as you need or want. So, hybrid apps have a low maintenance.
- Hybrid mobile applications need to develop once, so we require short time to develop and place it quickly on app store.

Disadvantages

- Hybrid apps add an extra layer between the source code and the target mobile platform, layer is call “mobile framework”. It can result in loss of performance.
- The extra layer from hybrid development framework also makes debugging a bigger task.
- As compared to native app development, it is difficult to maintain a proper user experience between the Android and iOS app. If you focus more on iOS, the user experience will worsen for Android users and vice a versa.

1.6 BASIC REQUIREMENT FOR THE CROSS PLATFORM MOBILE APPLICATION DEVELOPMENT

Today, many tools and frameworks are available to develop cross platform mobile applications. Below Table-1 shows the list of most popular tools used for the cross platform mobile application development.

Tools /	Approach	Manufacturer
----------------	-----------------	---------------------

Framework		
Ionic	Web View	Drifty Co.
React JS	Web View	Facebook
jQuery Mobile	Web View	jQuery Project
Cordova	Mobile development Framework	Apache
PhoneGap	Mobile development Framework	Nitobi
Xamarin	Cross Compiled	Xamarin
Robo VM	Cross Compiled	Robo VM AB

Table-1 Tools / Framework for Cross Platform Mobile Application Development

Here, in this book we learn cross platform mobile application development using Ionic. The Ionic is the web view based open source framework for the cross platform mobile application development. Ionic offers mobile application development based on Angular that is a Java Script frame work. So to learn Ionic, user have to know of HTML5, Cascading Style Sheet (CSS) and Java Script.

1.7 LET US SUM UP

This unit we learnt about the types of mobile application development environment. Let's quickly review the main points of the unit.

- At the end of 2019, number of smart phone user will be cross 5 billion.
- More than five million mobile applications available on leading app store.
- A mobile app that is coded for specific mobile OS called native app.
- Native application development require more time and cost.
- A mobile app which can be developed for multiple platform by using single codebase is call cross platform mobile application. It also call hybrid application.
- Hybrid app can be two type: Web view based or Cross compile base
- A mobile app which run the web applications by using web view is called web view based hybrid app.
- Web view based app develop using HTML5, CSS and Java Script.

- A mobile app which developed using specific language, will convert into mobile's OS specific native language at run time or compile time is call Cross compile hybrid app.
- Ionic, React JS, jQuery, Xamarin etc. are the tools / framework to develop cross platform mobile app.

1.8 CHECK YOUR PROGRESS

Give the answer of the following MCQ.

1. Android was developed by _____ ?

A. Google	B. Samsung
C. Apple	D. Microsoft
2. Android is based on _____ operating system

A. Window	B. Unix
C. Mac	D. Linux
3. iOS was developed by _____ ?

A. Google	B. Samsung
C. Apple	D. Microsoft
4. Latest version of iOS is _____

A. 10.2	B. 12.2
C. 11.2	D. 13.2
5. To develop iOS app _____ tool is require.

A. Y code	B. X code
C. Visual Studio	D. Eclipse
6. A mobile app that developed for specific mobile platform is called _____ app

A. Native	B. Cross platform
C. Hybrid	D. None of all
7. BYOD stand for _____

A. Best your Over Device	B. Buy your Own Device
C. Bring your Own Device	D. Bring why Odd Device
8. A mobile app developed using one codebase and run on any mobile is called _____.

A. Cross platform mobile app	B. Native mobile app
C. iOS mobile app	D. Android mobile app

Unit 2: Basic Of Development Environment - Angular

2

Unit Structure

- 2.1 Learning Objectives
- 2.2 Introduction
- 2.3 Angular Framework
- 2.4 History of Angular
- 2.5 Features of Angular
- 2.6 Prerequisites for Angular
- 2.7 Let us sum up
- 2.8 Check your Progress
- 2.9 Check your Progress: Possible Answers
- 2.10 Further Reading
- 2.11 Assignments

2.1 LEARNING OBJECTIVES

After studying this chapter, students should be able to understand:

- Angular framework.
- SPA (Single Page Application)
- Angular for mobile applications development.
- Angular architecture
- Advantages and disadvantages of Angular
- Necessary component to setup Angular environment.
- History of Angular
- Features of Angular4
- Perquisite of Angular4

2.2 INTRODUCTION

In the previous unit we learnt about the cross platform mobile application development environment. Various tools and frame works are available for the same. Here in this book we will learn the cross platform mobile applications development using Angular and Ionic framework. We also know that Ionic work is based on Angular framework. So before learning Ionic framework we need to get basic knowledge about Angular framework. This unit gives you insight of Angular. We start with Angular framework and architecture follow by history of it. We also discuss Angular environment and features of Angular4. This unit is explaining the concepts of Angular with version of “Angular4”.

2.3 ANGULAR FRAMEWORK

Angular is the framework for building web applications and mobile applications using HTML, Java Script and CSS. Angular is open source Java Script framework which binds the Java Script object and HTML UI elements. Angular was developed by the Google. It not just Java Script library but it is a complete framework that provides guideline in writing a proper architected, maintainable and testable client side code.

Angular provides inbuilt support for the animation, http services and many more things. Angular empower developer who develops applications which live on the web, mobile or the desktop. Angular is written in typescript.

2.3.1 EVOLUTION OF ANGULAR

Angular is basically invented to provide a good support to develop web applications. Angular is based on Java Script framework. We know that JavaScript is handling dynamic content on web. Sometimes, web sites are open in the mobile device, the Java Script need to decide that whether or not to render the mobile version of the requested website. As sometime as a dynamic content manager, Java Script needs to provide interface of the website on computer desktop or mobile devices. To support on both kind of device, developers need to write thousands of lines of code.

In late 2013, web developer wanted to have their own custom JavaScript libraries for reducing the number of code lines and implement complex functionalities easily. A jQuery is the small and feature rich Java Script library that make things easy for the web developer. But the main problem with jQuery is not real structure which makes lot of confusion in larger projects. Here Angular comes into picture and help a developer by providing structured environment. Angular is Java Script framework that was specifically designed to help developer to build SPAs (Single Page Applications) for web. A single-page application (SPA) is a web application or website that interacts with the user by dynamic rewriting the current page rather than loading entire new pages from a server. Figure-4 shows the different bewteen traditional web page life cycle and SPA life cycle.

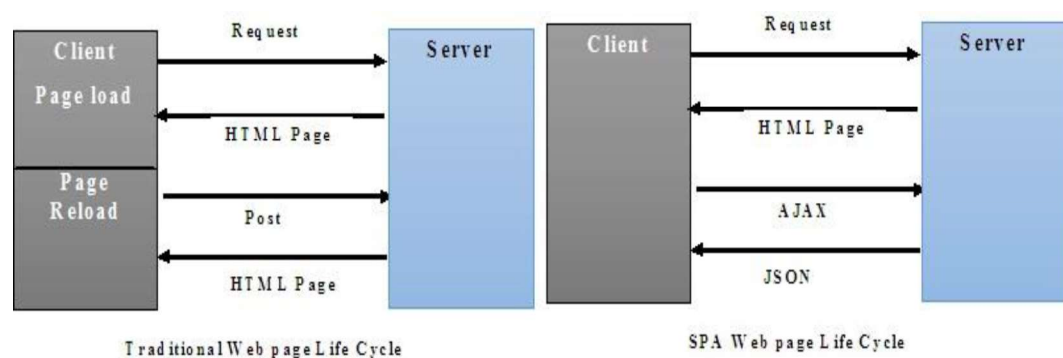


Figure-4 Traditional web v/s SPA web page life cycle

2.3.2 FEATURES OF ANGULAR

Before going into the detailed study of Angular, we need to learn the feature of the Angular:

- It provides a cross platform support. Angular applications can be run on any OS platforms.
- It gives high performance with easy way to write code.
- Using Angular, we can build native mobile application using Ionic framework.
- We can create desktop application version for Mac, Windows and Linux operating system with the use of Angular.
- Angular provide support to use any technology along with it. We can use node.js, PHP etc along with Angular framework.
- Angular apps load quickly with the new Component Router, which delivers automatic code-splitting, so users only load code required to render the view they request.
- It provides templates through which user can create UI views quickly.
- Using Command Line Interface (CLI) we can easily and quickly build and add components. We can also test and deploy them easily with the help of CLI.

2.3.3 ARCHITECTURE OF ANGULAR

Angular architecture consists of eight blocks. Any Angular app is made up of eight essential constituents. The eight blocks are Modules, components, Templates, Metadata, Data Binding, Directives, Services and Dependency Injection. Figure-5 shows the architectural view of Angular. Let us understand each block one by one.

➤ **Modules**

Modules are logical boundaries of the application. Instead of writing everything into one application, we can build separate modules for each functionality of the application. Angular apps are modular and to maintain modularity, we have Angular modules or we can say NgModules. Every Angular app contains at least one Angular module called root module. Module is made of three parts called Bootstrap Array, Export Array and Import Array. The details of its functionality can be seen learn in the next block of this book.

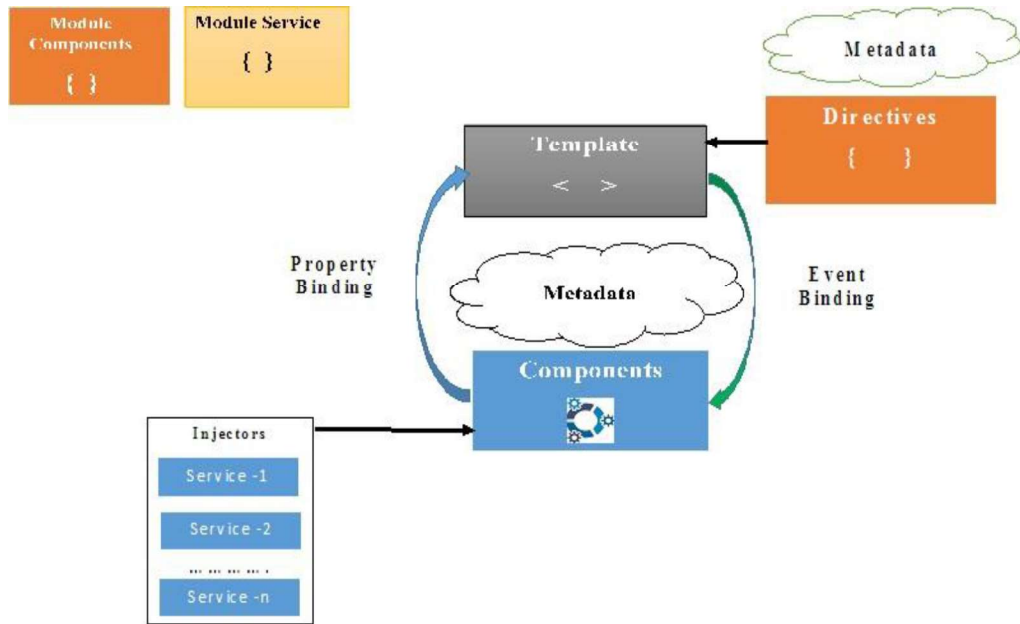


Figure-5 Architecture of Angular

➤ **Components**

Each application consists of Components. Each component is a logical boundary of functionality for the application. Each application is made up of modules. Each Angular application needs to have one Angular Root Module. Each Angular Root module can then have multiple components to separate the functionality. Each component consists of Class, Template and Metadata. It is represented by the Figure-6.

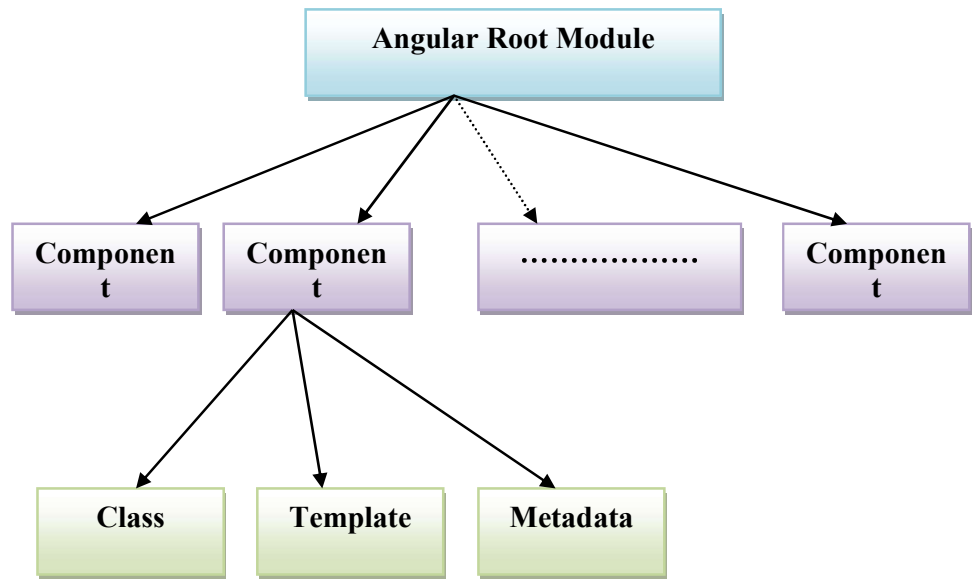


Figure-6 Module Component Relationships

➤ **Templates**

As the name suggests, templates are elements of Angular applications that combine HTML with Angular markup, which are able to modify HTML elements before displaying them on the screen. A template looks like regular HTML, except for a few differences. Templates make use of pipes for improving the user experience. Templates have two parameters like HTML code and Class Properties.

➤ **Metadata**

The information related to class is provided by metadata. This has an extra data defined for the Angular class. It is defined with a decorator. A class decorator is used for attaching metadata to a class. For providing the necessary information required by Angular to create the component, class Decorator makes use of configuration objects. Some of the configuration options are directives, selector, and template URL.

➤ **Data Binding**

If we are not using a framework, we have to push data values into the HTML controls and turn user responses into some actions and value updates. Angular supports data binding, a mechanism for coordinating parts of a template with parts of a component. We should add binding markup to the template HTML to tell Angular how to connect both sides. Binding markup is responsible for connecting application data with the DOM. There are two types of data binding, namely:

- **Event Binding** – Allows the application to respond to user input in the target environment. It does so by updating application data.
- **Property Binding** – Allows interpolation of values, which are computed from application data into the HTML.

➤ **Directives**

Angular templates are dynamic. When Angular renders them, it transforms the DOM according to the instructions given by directives. A directive is a

custom HTML element that is used to extend the power of HTML. Two kinds of directives exist, structural and attribute directives.

- **Structural directives** alter layout by adding, removing, and replacing elements in DOM.
- **Attribute directives** alter the appearance or behavior of an existing element.

➤ **Services**

Service is a broad category around any value, function, or feature which application needs. A service is typically a class with a well-defined purpose. Anything can be a service. It is a part of component. Examples include: logging service, data service, and business logic and application configuration.

➤ **Dependency Injection**

Dependency injection is the ability to add the functionality of components at runtime. Most dependencies are services. Angular uses dependency injection to provide new components with the services to the existing components.

2.3.4 ADVANTAGES AND DISADVANTAGES OF ANJULAR

Advantages

➤ **Consistency**

Code consistency is an important goal to strive for any code base system. Angular is framework is based on components and services. In components based structure always look same, we can add additional thing in to component but overall structure is always look same. All the components services start on the same way. By component and service's structure angular maintain consistency.

➤ **Productivity**

With greater consistency, we get the benefit of productivity. When we learn how to write one component we can write another component by same general guidelines and code structure. Once we learn how to create a service class it's easy to create another

one. So development of application possible on fast track and sufficient productivity.

➤ **Maintainability**

Angular code can be built using TypeScript (it is improved JavaScript) which provide lots of benefits along with easy maintenance of the app.

➤ **Modularity**

Angular is all about organizing code into modules. Everything you create whether it's components, services, pipes, or directives has to be organized into one or more modules.

➤ **Catch Errors Early**

Angular is built using TypeScript provide mechanism to find out errors easily if any.

Disadvantages

- Angular manipulates actual DOM (Document Object Model) which make it slower and less efficient.
- Angular is difficult to learn. If we don't know about TypeScript then need require more time to learn.
- Data Binding concept is difficult to implement in real world applications.
- It is difficult to implements server templates.
- Testing is difficult. End to end tests are simplified only with Angular CLI.

2.3.5 ANGULAR ENVIRONMENT

To start with Angular framework, we need to install several tools that Angular required. The detail installations of each components will be learned in block two of the book. To setup the Angular environment we need the following component:

➤ **Node js**

Node.js is a cross-platform runtime library and environment for running JavaScript applications outside the browser. This is a free and open source tool used for creating server-side JS applications. Node.js is useful

to build fast and scalable server-side networking applications. This framework is best suited for building single-page client-side web applications.

➤ **Npm (Node Package Manager)**

Npm is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js. NPM is a package manager for Node.js packages. The Angular Framework, Angular CLI, and components used by Angular applications are packaged as npm packages and distributed via the npm registry.

➤ **Angular CLI (Command Line Interface)**

Angular CLI makes it easy to start with any Angular project. Angular CLI comes with commands that help us to create and start on our project very fast. It provide command to create project, component and services.

➤ **IDE for writing code.**

It provide integrated development environment to write Angular code.

2.4 HISTORY OF ANGULAR

Angular is the most well-known framework for SPA (Single – page application) development. Angular allows the developer to interact with both the frontend and the backend. The first version of the framework – AngularJS started back in 2009. AngularJS was a outcome of side project, by two developers Misko Hevery and Adam Abrons at Google.

Misko Hevery eventually began working on a project at Google called Google Feedback. Misko Hevery and two other developers wrote 17,000 lines of code over the period of 6 months for Google Feedback. However, the code size increased, Misko Hevery began to grow frustrated with how difficult it was to test and modify the code which the team had written. So Misko Hevery made bet with his manager that he could rewrite the entire application using his side and get Angular project in two weeks. Hevery lost the bet. Instead of two weeks, it took him three weeks to rewrite the entire application, but he was able to cut the application from 17,000 lines to 1,500 lines. Because of Hevery's success, his manager predicted that Angular would

be powerful framework for the web development. Thus Angular.js development began to accelerate.

There are three major releases of Angular. The first version was released is AngularJS, which is also called Angular1. Angular1 is followed by Angular2, which came with a lot of changes compared to Angular1. AngularJS is totally different from Angular2. Some of the differences between AngularJS and Angular2 are mentioned below:

- The architecture of an Angular application is different from AngularJS. AngularJS is completely based on controllers and the view communicates using `$scope`. Whereas Angular is based on modules, components, templates, metadata, data binding, directives, services and dependency injection.
- Angular is a complete rewrite of AngularJS.
- Angular does not have a concept of “scope” or controllers yet it uses a hierarchy of components as its main architectural concept.
- Angular has a simpler expression syntax for event binding compared to AngularJS
- Mobile development and Desktop development is much easier.
- Angular follows modularity. Similar functionalities are kept together in same modules. This gives Angular a lighter & faster core compared to AngularJS.

We know that the structure of Angular is based on the components/services architecture. Angular is based on the model view controller. Angular 4 was released in March 2017 which proved to be a major breakthrough. Angular 4 is almost the same as Angular 2. It has a backward compatibility with Angular 2. Angular4 has advanced features compared to Angular2. Due to this, Angular framework becomes more stable.

The latest version of Angular is “Angular 7” which was released in October, 2018. However for the cross platform mobile applications development using Ionic, we will learn Angular4.

2.5 FEATURES OF ANGULAR4

Angular4 comes up with advance feature as compared to Angular2. The features are:

- Angular2 supported only “if” condition, however Angular4 supports the “if else” condition as well. It provide support of “if else” using “ng-template”
- Angular4 is smaller and faster. It reduces bundle file size upto 60% which improves the application speed.
- Angular4 is compatible with newer version of TypeScript2.2
- Majority of Angular Universal code has been merged into Angular core.
- Animations taken from the Angular core and set within their own package “Animation Package”. It means if we don’t use animations, the excess code won’t end up in app.

2.6 PREREQUISITIES FOR ANGULAR

Next block of this book will discuss Angular basics. The block will teach you, how to write Angular code. Before moving to the next block, you should have a basic understanding of HTML, CSS, Java Script and Document Object Model (DOM).

2.7 LET US SUM UP

In this unit we learnt the fundamentals of Angular framework. Let’s quickly review the main points of the unit.

- Angular is the framework based on Java Script.
- Angular is use to develop web view based hybrid mobile applications.
- Angular is used to execute web application outside browser.
- Angular provide structure environment to develop large application.
- Angular held developer to build SPA (Single Page Application) for web.
- Using Angular CLI developer can easily and quickly build applications.
- Angular architecture is made of eight components.

- The eight components are Modules, components, Templates, Metadata, Data Binding, Directives, Services and Dependency Injection.
- Modules are logical boundaries of the application.
- Each application consists of Components. Each component is a logical boundary of functionalities.
- Templates are elements of Angular applications that combine HTML with Angular markup, which are able to modify HTML elements.
- Metadata is a information related to class used by Angular.
- Data binding is a mechanism for coordinating parts of a template with parts of a component.
- Angular templates are dynamic. A service is typically a class with a well-defined purpose. And Dependency injection is the ability to add the functionality of components at runtime.
- To develop application using Angular, we should have a knowledge of HTML5, CSS, JavaScript and DOM architecture.

2.8 CHECK YOUR PROGRESS

Fill in the Blanks

1. Angular is frame work to develop mobile app using _____, _____, _____.
2. Angular was developed by _____ company.
3. Angular provide _____ environment.
4. Angular CLI stand for _____.
5. _____ is the logical boundaries of the application.
6. _____ is the logical boundaries of the functionality.
7. Each component consist of _____, _____, _____.
8. _____ modify HTML elements before displaying them on the screen.
9. _____ information related to class.
10. NPM stands for _____.
11. First version of Angular is called _____.
12. Angular is based _____ architecture.

2.9 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

- | | | |
|---------------------------------|----------------|---------------|
| 1. HTML5, CSS, Java Script | 2. Google | 3. Structured |
| 4. Command Line Interface | 5. Module | 6. Component |
| 7. Class, Template,
Metadata | 8. Template | 9. Metadata |
| 10. Node Package Manager | 11. Angular JS | 12. MVC |

2.10 FURTHER READING

- Angular web site : <https://angular.io/guide/quickstart>
- Oswald Campesato, "Angular 4 Pocket Primer"

2.11 ASSIGNMENTS

Write answer of the following Questions

1. Explain the features of Angular.
2. Write a short note on Angular Architecture.
3. List the advantages and disadvantages of Angular.
4. Write a short note on history of Angular.
5. List the features of Angular4.

Unit 3: Basic Of Development Environment - IONIC

3

Unit Structure

- 3.1 Learning Objectives
- 3.2 Introduction
- 3.3 Ionic Framework
- 3.4 History of Ionic
- 3.5 Understanding the Ionic Stack
- 3.6 Prerequisites for Ionic
- 3.7 Let us sum up
- 3.8 Check your Progress
- 3.9 Check your Progress: Possible Answers
- 3.10 Further Reading
- 3.11 Assignments

3.1 LEARNING OBJECTIVES

After studying this chapter, students should be able to understand:

- Ionic framework.
- Features of Ionic
- Advantages and disadvantages of Ionic
- Ionic Environment
- History of Ionic
- Ionic Stack
- Perquisite of Ionics

3.2 INTRODUCTION

In this unit we will discuss about the basic fundamental of Ionic. Ionic framework provides a facility to the mobile developers to develop cross platform mobile application. Ionic based mobile application need to code once and run on any mobile operating system like Android, iOS and Window phone. Ionic is an HTML5 mobile app development framework targeted at building hybrid mobile apps. Hybrid apps are essentially small websites running in a browser shell in an app that have access to the native platform layer. Hybrid apps have many benefits over pure native apps, specifically in terms of platform support, speed of development, and access to 3rd party code.

Ionic as the front-end UI framework handles all of the look and feel and UI interactions that app needs in order to be compelled. Ionic provide a native style mobile UI components and layouts that are the same as we get with native SDK on iOS or Android. Here we will learn about the basic of the Ionic framework and evolution of Ionic. After that we will talk about Ionic stack and prerequisites for Ionic learning.

3.3 IONIC FRAMEWORK

Ionic framework is the one of the popular frameworks for the cross platform mobile application development. Using this framework, mobile app developer can

create native-looking mobile applications. Ionic framework is based on web technologies such as HTML5, CSS and JavaScript. All these three technologies are open source. Ionic developer can build and upload mobile app on market place without any cost because Ionic is also open source. Developers and users should not pay any charge to use it.

The developers of Ionic believe that HTML5 would rule on mobile over a time, exactly as it has on desktop. Ionic is an HTML5 mobile app development framework targeted at building hybrid mobile apps. Hybrid apps are essentially small websites running in a browser shell in an app that have been accessed to the native platform layer. Hybrid apps have many benefits over pure native apps, especially in terms of platform support, speed of development, and access to 3rd party code.

Those who are familiar to web development, will find the structure of an Ionic app straightforward. At its core, it's just a web page running in a native app shell. That means we can use any kind of HTML, CSS, and JavaScript we want. The only difference is, instead of creating a website that others will link to, we are building a self-contained application experience. The bulk of an Ionic app will be written in HTML, JavaScript, and CSS.

Ionic framework is built with Angular, a widely used and well tested framework. The combination of the two framework allows Ionic's developer to create SPA (Single Page Application) based mobile application that is easier to organize. Each mobile operating system has their requirement for UI components. Ionic provides a readymade UI mobile component and feature of automatic implementation based on the mobile operating system it built for. So mobile application users fill the UI same as the native application provide. Ionic use SASS (Syntactically awesome style sheets) to generate CSS that visualize the Ionic application. SASS is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets (CSS). SASS provides several advantages over writing CSS directly. Detail will be shown later in this book.

Ionic also provides the JavaScript features for app developers for developing Ionic applications. All the JavaScript features are built on Angular. App developers use the JavaScript feature through the Angular.

3.3.1 FEATURE OF IONIC

Ionic help to build cross platform mobile applications using HTML5 with the use of Angular framework. Ionic provides a great range of tools and service using the framework. Let us see key features of Ionic framework.

- Ionic uses Angular MVC architecture for building SPA (Single Page App) specifically for mobile devices.
- SASS provides plenty of UI components for creating robust and rich apps.
- It provides JavaScript components. These components are extending CSS components with JavaScript functionalities that are specifically made for the mobile elements, which is not possible only with HTML and CSS.
- Development of the app is very vital only once as well as it would be compatible with all the mobile devices. Also, it needs very limited use of time, resources and efforts, and helps in giving an integrated look and feel. It provides easy and feasible cross platform mobile application environment
- Ionic has CLI. It is nodeJS utility which is the command for starting, building, running and emulating Ionic applications.
- Ionic View is very useful platform for uploading, sharing testing mobile application on native devices.
- Ionic is released under MIT license. Ionic is free and open-source.

3.3.2 ADVANTAGES AND DISADVANTAGES OF IONIC FRAMEWORK

Following are the advantages and disadvantages of Ionic framework:

Advantages

- Ionic framework provide cross platform mobile application environment. It means we can build mobile application that run on iOS, Android, Window Phone or other mobile operating system.
- To start mobile application is easy with the help of pre-generated app setup with simple layout.

- The mobile application is built in modular way, so it is easy to be maintained and updated.
- Ionic framework is built with Angular framework, which is a product of Google. So updates comes regularly that help fast and speedy mobile development.

Disadvantages

- Testing of it is little tricky as browser does not always give right information about mobile environment.
- Today lots of mobile devices and operating system are available in market. Usually all need to cover them for Ionic framework support.
- It will be hard to combine different native functionalities.
- Sometimes compatibility issues arises, which leads to build errors that are hard to debug.
- Cross platform applications are slower than native applications.

3.3.3 IONIC ENVIRONMENT

To start with Ionic framework, we need to install several tools that Ionic requires. It is little challenging task to install it. The detail installations of each component will be shown in block three of the book. We need to setup two kinds of components: The base of Ionic installation and platform specific SDK installation. The base installation requires tools that need to build mobile applications and preview in browser. The platform specific installation requires to setup native development environment. As we know our mobile application will be built based on web technologies, the platform specific installation gives access of emulators to test the applications on mobile device.

- **The base of Ionic Installation :**

This section discusses the tools required to setup the Ionic environment.

- **NodeJS**

The foundation for Ionic is built on NodeJS. It is a platform that enables you to run JavaScript outside the browser. Based on NodeJS (Simple sometime called Node) developer can build applications that are

written in JavaScript that can be run anywhere. Both Cordova CLI and Ionic CLI are written using Node.

- **Git**

Ionic CLI holds Git for the management of templates. Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

- **Apache Cordova CLI**

Cordova provides a set of JavaScript APIs which enables a developer to build a application using HTML5, CSS3 and JavaScript and through Cordova's APIs access native-specific functions like GPS, camera and network.

The installation of Cordova CLI uses the Node package manager (npm) to perform the installation. This is the SDK.

- **Ionic CLI**

Ionic compatibility starts at iOS 6 and Android 4.1, older versions than that will not be officially supported. The Ionic Framework also provides a useful command line interface (CLI) that makes it easy to start, create, compile and export mobile applications. The Ionic framework provides some useful functions such as `ionic.Platform.isIOS()`, `ionic.Platform.isAndroid()` and `ionic.Platform.isWindowsPhone()` which can be used to detect on what OS the current application is running on.

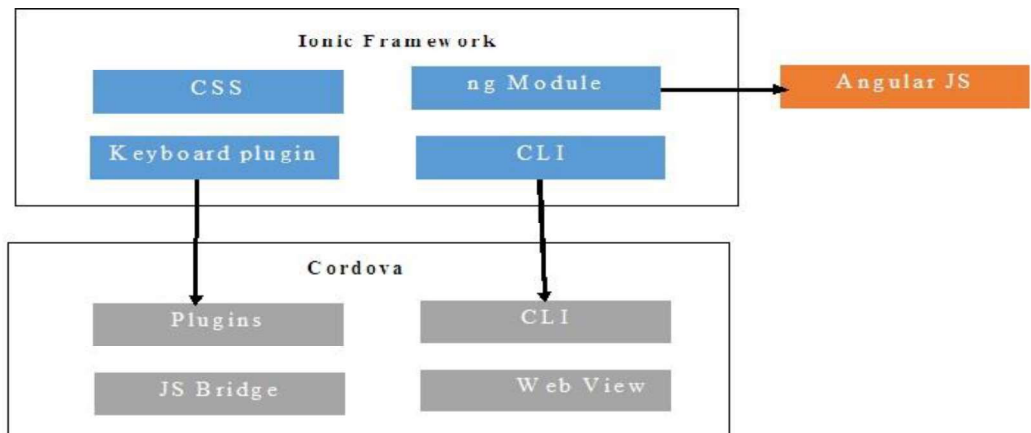


Figure-7 Ionic Framework

➤ **Platform Specific SDK Installation:**

By setting up above component we can start developing applications and test it in browser. But if we want to test it on either device emulator or actual device we need to set up platform specific SDK. Currently Ionic officially supports iOS, Android and Window Universal platform.

- **iOS**

If we want to build application for iOS, we need to install Xcode for emulation and distribution of app. Xcode is available for Mac os.

- **Android**

If we want to build application for Android, Android can be done on Window, Mac or Linux system. We need to install Android SDK tools. If we require IDE then we should install Android Studio.

- **Window Universal**

Window universal supports only on window machine. We need to install visual studio 2015 with the option of “Tools for Cross Platform Development” and SDK for Window Universal app.

3.4 HISTORY OF IONIC

Ionic is an open-source SDK framework for hybrid mobile app development. It was created by Max Lynch, Ben Sperry and Adam Bradley of Drifty Company in 2013. The first version was built on top of AngularJS and Apache Cordova. Ionic V1 is focused on building native mobile apps rather than mobile websites. Ionic 1x supported by iOS 7+ and Android 4.1 and up. Ionic 2 is focused on building both native/hybrid apps through Cordova, as well as adding the ability for Progressive Web Apps and Electron. Ionic 2 supported by iOS8+, Window 10 and Android 4.4 and up versions of mobile operating system.

The Ionic 3 or simply "Ionic", are built on Angular (web framework). However, The Ionic3 release allows you to choose your User interface framework from Angular (web framework), React (JavaScript library) and Vue.js. The latest release Ionic 4

was come up in January, 2019. Every release come up with new components and updates in framework.

3.5 UNDERSTANDING THE IONIC STACK

We are clear about the fundamental of mobile application development. It is time to look deeper into Ionic framework. The Ionic mobile application is built as part of three layers of technology.

- **Cordova**

The Cordova is used as an interface between the web view and the device's native layer. It's library provide a framework to bridge the gap between two technologies, web technology and native web view. Cordova is provide mobile platform support form iOs, Android, Window Phone, Blackberry and FireOS. It is an open source framework developed in 2009.

- **Angular**

As we have discussed a lot about Angular in last unit, there is no need to discuss it here in detail about Angular. Basically Angular provide SPA (Single Page App) development with MVW (Model – View – Whatever) to build complex web applications. Ionic team decides to take advantage of this feature of Angular and they build the application upon it.

- **Ionic Framework**

Ionic framework is an open source provided under MIT license. The primary feature of the Ionic Framework is to provide the user interface components that are not available to web-based application development. For example, a tab bar is a common UI component found in many mobile applications. But this component does not exist as a native HTML element. The Ionic Framework extends the HTML library to create one. These components are built with a combination of HTML, CSS, and JavaScript, and each behaves and looks like the native controls it is recreating. Ionic also provides some additional tooling to help build mobile applications.

- **Ionic CLI**

The Ionic CLI (Command Line Interface) is a command line tool that is used to manage Ionic applications. It allows you to create an Ionic application easily, and provides tooling for serving your application throughout development, and building your application for production.

- **Capacitor**

Capacitor is a separate project to Ionic (it is still created by the Ionic team), but it is used in conjunction with Ionic. Capacitor provides a common API for accessing native functionality across different platforms. This means that if you want to access functionality like the camera, you can use the same code for iOS and Android without worrying about the underlying native implementation on each platform. Capacitor also allows you to build your Ionic application as a native application for iOS/Android/Desktop.

- **Appflow**

Ionic Appflow is an optional platform which is also provided by the Ionic team that you can use in conjunction with your Ionic applications. This is a paid solution that provides functionality like continuous deployment and automatic application builds.

3.6 PREREQUISITIES FOR IONIC

To develop Ionic application we need to have some additional technical skills that are not covered in this book. While you do not need to use an expert in these skills, you should have general knowledge of these technical terms to understand the concepts of Ionic.

Ionic applications are built using HTML5, CSS and JavaScript. So you should have basic knowledge of these technologies and how to implement these technologies to build web applications. Here in this book Ionic application development code used JavaScript based on Angular4, this means you should know fundamental of Angular4. However we will learn the Angular4 in block2 in detail. Ionic is all about cross platform mobile application development, so probably you should have basic knowledge for the operating the iOS and Android devices.

In short Ionic is built on top of AngularJS and Apache Cordova, you will need to have basic knowledge about web technologies. You should familiar with HTML, CSS and JavaScript, if you want to understand all the information provided by the book.

3.7 LET US SUM UP

In this unit we learnt the fundamentals of Ionic framework. Let's quickly review the main points of the unit.

- Ionic is the framework for developing hybrid mobile application.
- Ionic framework is based on web technologies such as HTML5, CSS and JavaScript.
- Ionic is used to develop Single Page Application (SPA)
- Ionic environment consist of NodeJS, Git, Cordova CLI and Ionic CLI.

3.8 CHECK YOUR PROGRESS

Fill in the Blanks

1. Ionic framework is based on _____ technology.
2. Using Ionic developer can built _____ mobile application.
3. Ionic use _____ architecture.
4. Ionic is release under _____ license.
5. The foundation of Ionic is built on _____.
6. CLI stands for _____.
7. Ionic developed by _____ company.
8. Latest version of Ionic is _____.
9. _____ provide a common API for accessing native functionality.

3.9 CHECK YOUR PROGRESS: POSSIBLE ANSWER

- | | | |
|-----------|-----------|---------------------------|
| 1. Web | 2. Hybrid | 3. MVC |
| 4. MIT | 5. NodeJS | 6. Command Line Interface |
| 7. Drifty | 8. Ionic4 | 9. Capacitor |

3.10 FURTHER READING

- Ionic web site : <https://ionicframework.com/>
- Chris Griffith, “Mobile Appl Development with Ionic” , Oreilly

3.11 ASSIGNMENTS

Write answer of the following Questions

1. Explain Ionic framework.
2. List the features of Ionic.
3. List the advantages and disadvantages of Ionic.
4. Write a short note on Ionic environment.
5. Write a short note on Ionic Stack.

Block-2

Working with Angular

Unit 1: Introduction to Angular

1

Unit Structure

- 1.1 Learning Objectives
- 1.2 Basic Introduction of Script
- 1.3 About Angular
- 1.4 General Features
- 1.5 Advanced Features
- 1.6 Advantages and Disadvantages of Angular
- 1.7 Difference between Angular and AngularJS
- 1.8 Let Us Sum Up
- 1.9 Check Your Progress
- 1.10 Check Your Progress: Possible Answers
- 1.11 Assignments

1.1 LEARNING OBJECTIVES

After studying this chapter, students should be able to understand:

- The history of SCRIPT and Why SCRIPT
- The Different available SCRIPTING languages
- Difference between AngularJS and Angular with syntax
- General and Core features of Angular
- Advantages and Disadvantages of Angular

1.2 BASIC INTRODUCTION OF SCRIPT

All scripting languages are known as programming languages. The scripting language is basically a language where instructions are written for a run time environment. Scripting languages do not require the compilation step and interpreted. It brings new functions to applications and attach complex system together. A scripting language is a programming language designed for integrating and communicating with other programming languages.

Scripting languages are becoming more popular. Scripting languages are intended to be very fast to learn and write in, either as short source code files or interactively in a read–eval–print loop (REPL, language shell). This generally implies relatively simple syntax and semantics; typically a "script" is executed from start to finish, as a "script", with no explicit entry point. There are list of scripting languages are available but among them JavaScript is known as popular scripting language.

JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for interaction of a user with the webpage. You can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and Mobile application development.

You should place all your JavaScript code within <script> tags (<script> and </script>) if you are keeping your JavaScript code within the HTML document itself.

This helps your browser distinguish your JavaScript code from the rest of the code. As there are other client-side scripting languages (Example: VBScript), it is highly recommended that you specify the scripting language you use. You have to

use the type attribute within the <script> tag and set its value to text/ JavaScript like this:

```
<script type="text/javascript">
```

➤ **Hello World Example using JavaScript:**

```
<html>
<head>
<title>My First JavaScript Example</title>
<script type="text/javascript">
alert("Hello World...");
</script>
</head>
<body>
</body>
</html>
```

There are many scripting languages some of them are discussed below:

- **Ruby:** It is a scripting language which is widely used for web development.
- **Python:** It is easy, free and open source. Python is an interpreted language with dynamic and huge lines of code are scripted.
- **bash:** It is a scripting language to work in the Linux interface.
- **Node js:** It is a framework to write network applications using JavaScript. Popular users of Node.js are IBM, LinkedIn, Microsoft, Netflix, PayPal, Yahoo for real-time web applications.
- **PHP (Hypertext Preprocessor):** PHP is a server-side scripting language designed for Web development. PHP code may be embedded into HTML code, or it can be used in combination with various web template systems, web content management systems, and web frameworks.
- **Angular JS:** AngularJS is a JavaScript framework. AngularJS extends HTML attributes with Directives, and binds data to HTML with Expressions. It can be added to an HTML page with a <script> tag.

Scripting languages, on other hand, solves different problems:

- Building applications from 'off the shelf' components
- Controlling applications that have a programmable interface
- Writing programs where speed of development is more important than run-time efficiency.

1.3 ABOUT ANGULAR

Successive versions of Angular are simply called Angular. Angular is one of the most popular JavaScript framework. *Angular* is a complete rewrite of the *Angular* framework. Angular is a platform that makes it easy to build an applications with the web. Angular empowers developers to build applications that live on the web, mobile, or the desktop. Angular is a faultless framework for developing Single Page Applications. A SPA is web application that requires only a single page load in web browser.

Angular was created by Google and released in 2010 as AngularJS the 1.x version . The initial release for the Angular 2.x version was on September 14, 2016. The second major revision was initially referred to as AngularJS 2 or 2.0 but was later rebranded to just “Angular” for release 2.0 and higher.

Angular allows us to build applications for all platforms. It is free and an open-source platform that uses *TypeScript*. *TypeScript* is a strict syntactical superset of JavaScript, adds optional static typing to the language and maintained by Microsoft.

Angular Version	Release Date
Version 2	September 14, 2016
Version 4	March 23, 2017
Version 5	November 1, 2017
Version 6	May 4, 2018
Version 7(Latest)	October 18, 2018

Table-2 Transparent and Incremental evolution of Angular

1.4 GENERAL FEATURE

Developers prefer this framework for their front-end development for a number of reasons. It's simple to use and has some powerful tools that make the development process much more efficient. These tools also seamlessly work together in tandem providing you with an efficiently compiled program. Given below are five features of Angular that make it the best for web development as framework.

➤ The Perfect MVC Architecture

MVC stands for Model View Controller Architecture which is a software pattern to develop applications. The model layer manages the application data, the view layer is responsible for the display of data while the controller is what connects the model and the view. in Angular, all you have to do is split the application into MVC and it does the rest. This saves huge amount of time in coding.

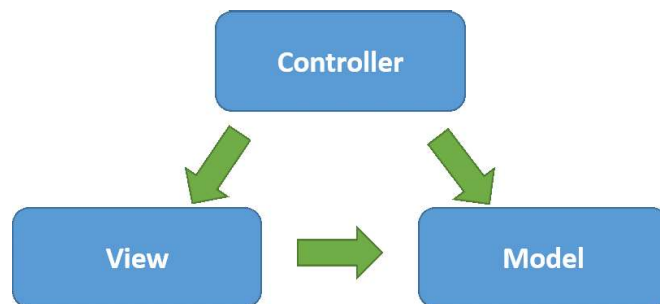


Figure-8 MVC Architecture

Unit Testing assures Quality Code

One of the best and popular features of angular is the fact that it makes use of a unit testing technique that helps developers to produce high quality apps. The code is divided into the smallest testable parts i.e. units. This also helps you easily detect any flaws or mistakes in each line of the code.



Using Angular's module that provides mocking for tests, mock units are injected and tested to see if the request is returned with the expected data. This improves you make sure that each and every component of your application works exactly as required.

Data binding is Efficient

Data binding in Angular is a two way road. This means that the view layer of the architecture is always an exact representation of the model. Unlike in other apps, the model and view layers are continuously updated to remain in sync with one another.

So any changes you make in your model layer will automatically be reflected in your view layer and vice versa. This again saves a significant amount of time in coding the connection between the two whenever a change is made.



Figure-9 Data binding in Angular

Requires Writing Less Code



Figure-10 Less Code

Angular needs a lot less coding than others. You do not have to write code to connect the MVC layers, you don't have to write separate codes for the view manually, directives are separate from the app code and can be written parallelly etc. All of these collectively decrease the amount of coding that is required, significantly.

Developed by Google



Figure-11 Angular

Google is the apex of the internet age and you know when there's something that Google develops, it will be great. Angular is maintained by a dedicated team of highly skilled engineers who are readily available to solve any issues related to the framework.

1.5 ADVANCED FEATURE

Angular release is mainly focused on making Angular framework smaller, faster, and easier to use. So for that Google introduced new advanced features of Angular as below.

Progressive Web Application

This feature of angular on application makes feel like native apps with mobile web apps along with add-ons like offline experience and push notifications. This is made possible as Angular can create code and configuration on its own with Angular-CLI and it offers service workers through the `@angular/service-worker`.

Following command is used to activate PWA support in your application:

```
$ ng set apps.0.serviceWorker=true
```

Build Optimizer tool(Default tool)

This tool is by default applied. The build optimizer tool makes the application faster and lighter by removing additional parts and runtime code as well. This tool decrease the size of script and enhance the speed of the application.

Angular Universal State Transfer API and DOM

Angular team added the domino to the platform-server.it enhance the more DOM manipulations are supported within server-side contexts. Team also added the BrowserTransferModule and ServerTransferStateModule. Both modules enable you to transfer information between the server and client-side versions of the application. This is helpful for developers when their application accessing data over HTTP. This means there is no need to make another HTTP request once the application reached client-side.

HttpClient

Using the HTTP Client developers can replace the HttpModule with HttpClientModule from `@angular/common/http`, inject the HttpClient service, and remove the `map(res => res.json())` calls that is no longer required.

Compiler and Typescript improvements

This new feature brought a lot of improvements in Angular Compiler to make re-builds of the applications faster, mainly for AOT and production builds. And the

TypeScript is also upgraded to the latest version of TypeScript , which allows connecting to the standard TypeScript compilation pipeline

You can use this by running:

```
ng serve
```

Multiple Export Alias

This allows exporting, you can give multiple names to your directives and components. Exporting a component/directive with multiple names can help users to migrate smoothly without breaking changes.

Animation

Angular now came with some updates in Animations, where you can animate by using `:increment` and `:decrement` based on numerical value changes. you can also activate and deactivate the animations using values that are associated with data binding. The `.disabled` attribute of the trigger value is constrained to do this.

1.6 ADVANTAGES AND DISADVANTAGES OF ANGULAR

The advantages of Angular are:

- It provides the capability to create Single Page Application.
- Angular code is well structured
- Angular uses the Typescript.
- It provides data binding capability to HTML.
- It gives user a rich and responsive experience.
- Angular code is unit testable and easy to debug.
- Angular uses dependency injection and make use of separation of concerns.
- Angular provides reusable components.
- With Angular, the developers can achieve more functionality with short code.
- In Angular, views are pure html pages, and controllers written in JavaScript do the business processing.

Though Angular comes with a lot of merits, here are some points of concern as demerits:

- **Not Secure** – Being JavaScript only framework, application written in Angular are not safe. Server side authentication and authorization is must to keep an application secure.
- **Not degradable** – If the user disables JavaScript, then nothing would be visible except the basic page.

1.7 DIFFERENCE BETWEEN ANGULAR AND ANGULARJS

Angular was a ground-up note of AngularJS.

- Instead of scope or controller, Angular uses a hierarchy of components as its primary architectural characteristic.
- As compare to AngularJS, Angular has a different expression syntax, focusing on "[]" for property binding, and "()" for event binding.
- Modularity is a much core functionality has moved to modules.
- Angular recommends the use of Microsoft's TypeScript language, which introduces the features like Class-based Object Oriented Programming, Static Typing, Generics.
- TypeScript is a superset of ECMAScript 6 (ES6), and is backwards compatible with ECMAScript 5 (i.e.: JavaScript). Angular also includes ES6: Lambdas, Iterators, For/Of loops, Python-style generators, Reflection, Dynamic loading
- Asynchronous template compilation
- Iterative callbacks provided by RxJS. RxJS limits state visibility and debugging, but these can be solved with reactive add-ons like ngReact or ngrx.
- Support Angular Universal, a technology that runs your Angular application on the server
- Has its own suite of modern UI components that work across the web, mobile and desktop, called Angular Material

1.8 LET US SUM UP

- All scripting languages are known as programming languages.
- Scripting languages do not require the compilation step and interpreted.
- JavaScript is a very powerful client-side scripting language. JavaScript is used mainly for interaction of a user with the webpage.
- There are many(Ruby,Python,bash,Nodejs,PHP) scripting languages
- Angular was created by Google and released in 2010 as AngularJS the 1.x version
- Angular tools make it the best for web development as framework.
- Angular release is mainly focused on making Angular framework smaller, faster, and easier to use.
- Build optimizer is by default tool.
- New feature brought a lot of improvements in Angular Compiler to make re-builds of the applications faster, mainly for AOT and production builds.
- Some times angularbecome the not secure and not degradable.
- Modularity is a much core functionality has moved to modules.

1.9 CHECK YOUR PROGRESS

Give the answer of the following MCQ.

1. In AngularJS, JS stands for _____
A. Java Server
B. Java Script
C. Java Servlet
D. JSON Script
2. All scripting languages are known as _____ language.
A. Presentation
B. Process
C. Programming
D. POST JS
3. _____ is a scripting language to work in the Linux interface.
A. Ruby
B. Perl
C. Bash
D. PHP
4. The latest version of Angular is _____.
A. Angular 7
B. Angular 6
C. Angular 5
D. Angular 6.9

5. Angular follows the _____ architecture.
- A. MVV
B. MVC
C. MMC
D. MV
6. In MVC, C stands for _____
- A. Controller
B. Center
C. Core
D. Co-operative
7. Latest version of Angular comes with _____
- A. Animation
B. Http Client
C. Only A
D. Both A and B
8. Angular is Developed by _____
- A. Facebook
B. Alibaba
C. Google
D. Amazon
9. Angular uses the _____ script for the core programming.
- A. Java
B. Type
C. Ruby
D. Paython
10. To compile the Angular Application _____ command is used.
- A. ng open
B. ng run
C. ng compile
D. ng serve

1.10 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

1. Java Script
2. Programming
3. Bash
4. Angular 7
5. MVC
6. Controller
7. Both A and B
8. Google
9. Type
10. ng serv

1.11 ASSIGNMENTS

Write the answer for the following questions.

1. What is Script? Explain the different scripting programming language.
2. Explain the Angular and list out the available versions for angular.
3. Explain the general features of Angular.
4. List and Explain the advanced features of angular.
5. Explain the advantages and disadvantages of angular.
6. Differentiate Angular and Angularjs

Unit 2: The Basic of Angular

2

Unit Structure

- 2.1 Learning Objectives
- 2.2 What and Why Framework
- 2.3 Environment setup
- 2.4 Hello World in Angular
- 2.5 Directives and String interpolation
- 2.6 Angular Events
- 2.7 Let Us Sum Up
- 2.8 Check Your Progress
- 2.9 Check Your Progress: Possible Answers
- 2.10 Assignments

2.1 LEARNING OBJECTIVES

After studying this chapter, students should be able to understand:

- The nature of framework and what is the important of framework
- Why only Angular and settingup the environment of Angular
- The Compilation and Running process of angular
- Entire folder structure of Angular
- Basic use of the events and event handling

2.2 WHAT AND WHY FRAMEWORK

Angular is a modern web application platform. That promises to provide developers with a comprehensive set of tools and capabilities to build large, complex and robust applications. The core advantage of Angular is to make it possible to build applications that work for any platform like mobile, web, or desktop. The Angular team has focused on building much more than a robust application framework.

2.2.1 WHY ANGULAR

To build web applications that can meet the needs of users is not a small task. The quality and complexity of applications is ever increasing, and so are users expectations for quality and capabilities. Angular exists to help developers deliver applications to meet these demands.

Angular is inspired by web standards, enhanced by modern capabilities

Angular tries to design its framework and the development process around common standards like the latest JavaScript language features, using modern capabilities like Typescript.

Development tooling included, customizations available

Angular provides a common developer experience through its CLI tooling which includes the generating, building, testing and deploying apps.

Powerful ecosystem with a large community

Angular supports number of third-party libraries, UI libraries, blog posts, and events. Angulars active community provides a great foundation on which to learn and should instill confidence that it will remain a valuable technology.

Sponsored by Google, open source community driven

The Google has a team of engineers, managers, and evangelists only dedicated to bringing Angular to the rest of Google and the entire web community.

2.2.2 ANGULAR: A PLATFORM, NOT A FRAMEWORK

Some important distinctions between a framework and a platform are a frame-work is usually just the code library used to build an good application, whereas a platform is more holistic and includes tooling and support beyond a framework. Angular was focused solely on building web applications in the browser and was clearly a frame-work. It had a large ecosystem of third-party modules that could be easily used to add features to your application, but at the heart of it all, it simply built web applications in the browser. Angular comes with a leaner core library and makes additional features available as separate packages that can be used as needed. It also has many tools that push it beyond a simple framework, including the following features:

- Dedicated CLI for application development, testing, and deployment
- Offline rendering capabilities on many back-end server platforms
- Desktop-, mobile-, and browser-based application execution environments
- Comprehensive UI component libraries, such as Material Design

2.3 DOWNLOADING AND INSTALLING ANGULAR

Angular helps to you build dynamic applications for mobile, web and desktop. The following steps shows you how to install and build you app in angular. AngularJS is based on MVC, whereas Angular 2 is based on component structure. But the version Angular 4 works on the same structure as Angular2 but is faster when compared to Angular2.

Angular4 uses TypeScript 2.2 version whereas Angular 2 uses TypeScript version 1.8. This brings a lot of difference in the performance. To install Angular 4, the Angular team came up with Angular CLI which eases the installation. You need to run through a few commands to install Angular 4.

Follow the Following steps to install Angular 4.

Step 1: Install the nodejs and npm

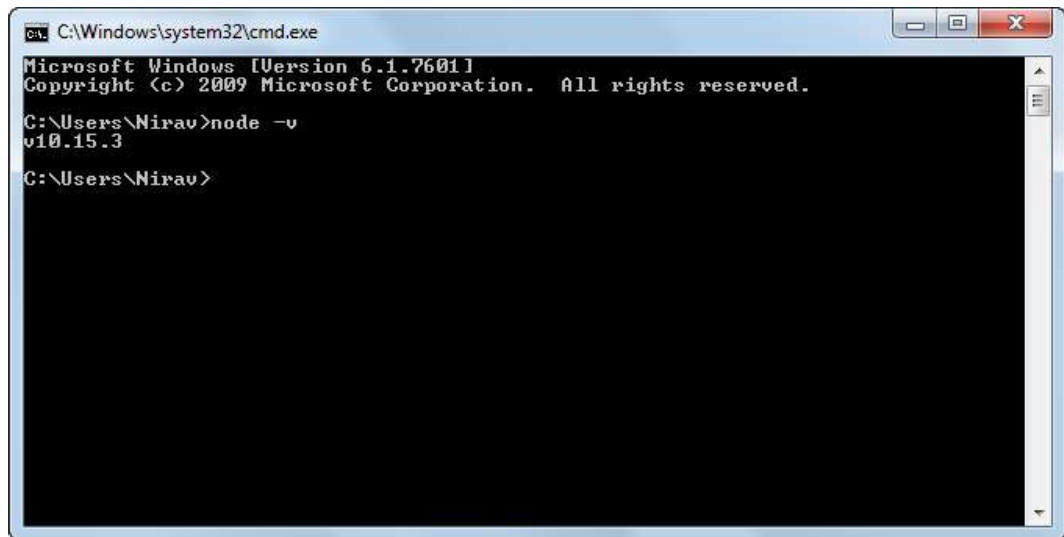
We first need to install nodejs and npm with latest version to get started installation of Angular. The npm package gets installed along with nodejs.

To download node js go to the nodejs site

<https://nodejs.org/en/>

Angular requires nodejs version 8.x or 10.x.

- To check your version, run `node -v` in a terminal/console window.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Nirav>node -v
v10.15.3

C:\Users\Nirav>
```

Figure-12 Console

To check the version of npm, type command `npm -v` in the consol. It will display the version of npm as shown below.

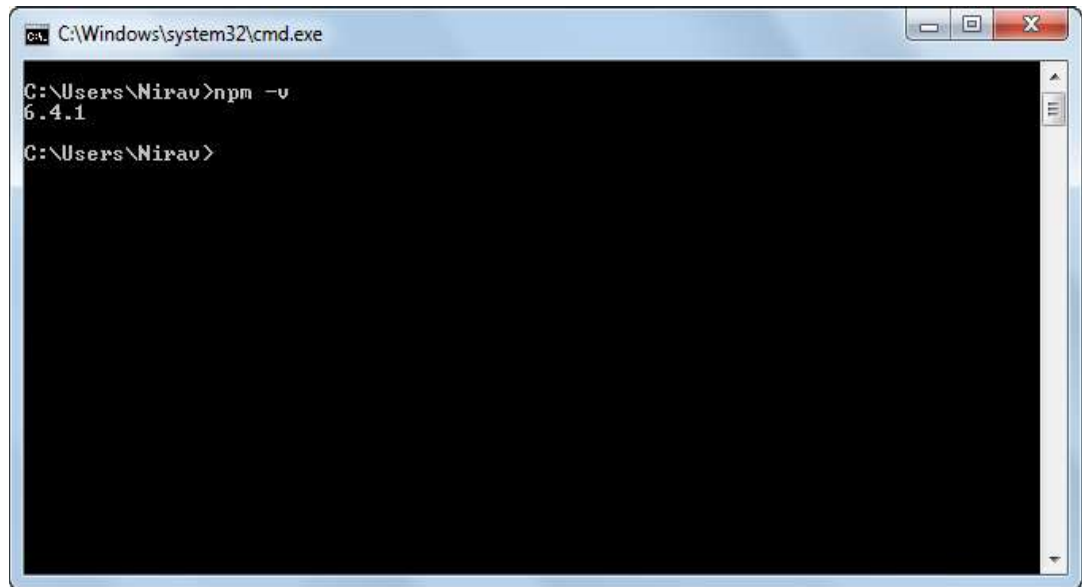
A screenshot of a Windows command prompt window. The title bar reads 'C:\Windows\system32\cmd.exe'. The command prompt shows the user's current directory as 'C:\Users\Nirav'. The user has entered the command 'npm -v', and the output is '6.4.1'. The prompt is now 'C:\Users\Nirav>'.

Figure-13 Check Version

Step 2: Install the Angular CLI(Command Line Interface)

Now that we have nodejs and npm installed as step 1 shown, let us run the angular cli commands to install Angular 4. You use the Angular CLI to create projects, generate application and library code, and perform a variety of ongoing development tasks such as testing, bundling, and deployment. Install the Angular CLI globally.

To install the CLI using npm, open a terminal/console window and enter the following command:

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

Step 3: Create a workspace and initialize application

Workspace contains the files structure for one or more projects. A project is the set of files that comprise an app, a library tests.

To create a new workspace and initial app project:

Run the CLI command ng new and provide the name my-app, as shown here:

```
ng new first
```


It also creates the following workspace and starter project files:

- A new workspace, with a root folder named first
- An initial skeleton app project, also called first (in the src subfolder)
- An end-to-end test project (in the e2e subfolder)
- Related configuration files

Step 4: Serve the application

Angular includes a server, so that you can easily build and serve your app locally.

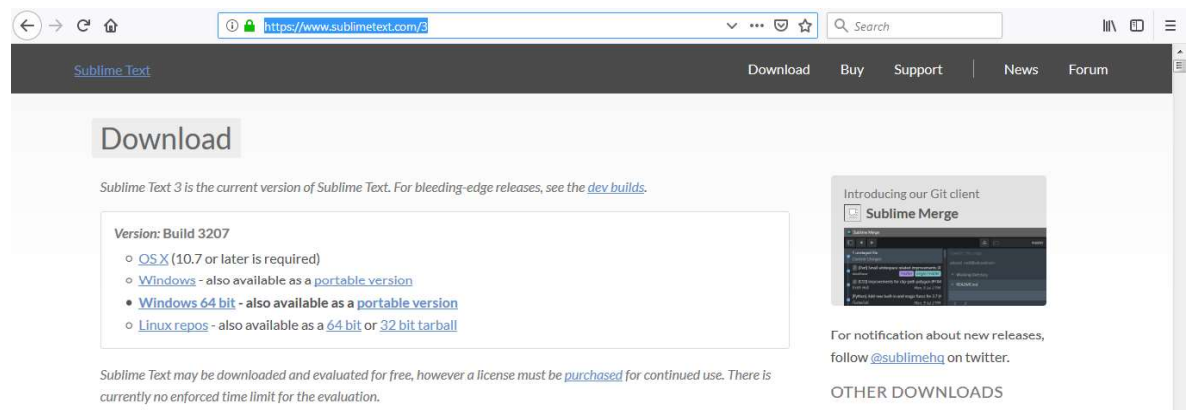
1. Go to the workspace folder (my-app).
2. Launch the server by using the CLI command `ng serve`, with the `--open` option.

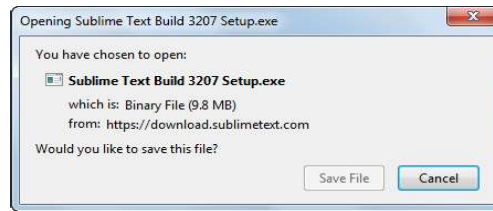
```
cd first
ng serve --open
```

Step 5: Edit the application

The project first is created successfully. Internally its install all the package which are required to Angular 4. Now we have to change the view and behavior of app so for the need a one smart editor. You can use the IDE like Visual Studio Code IDE, Atom, Webstrom, Sublime Text, etc.

To download the Sublime Text go to <https://www.sublimetext.com/3> and click on windows and download.





2.4 FIRST EXAMPLE IN ANGULAR

To create an application in Angular, Angular CLI is an awesome tool to start. It creates fully functional well-structured project which we can take forward.

Only few steps are required to follow for first “Hello Wolrd” Example in Angular.

1. Check angular version using **ng -v**.
2. Create our first app using command `ng new hello-world` It will take a while and create a fully-fledged app for you.
3. Now you will have your folder created called **hello-world**. Navigate to it using **cd hello-world** and then do **ng serve**. You can see server is started.
4. Open your browser and visit localhost:4200 You will see “App works” printed on screen.

After performing this steps open your project in sublime or visual studio code to use and understand the project structure.

The project structure of Angular 4 application which Angular CLI created for us. We have already created HELLO-WORLD project, so we will use that project to get idea about project structure. When we open the Angular 4 project in editor, we can see three main folders `e2e`, `node_modules`, `src` and different configuration files.

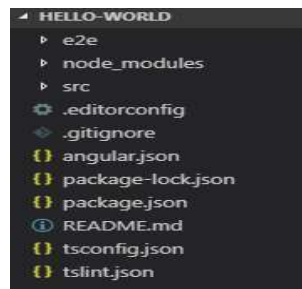
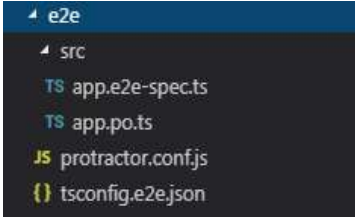
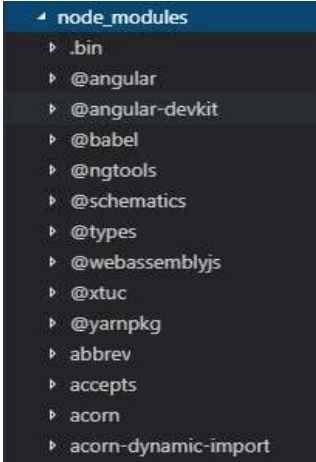
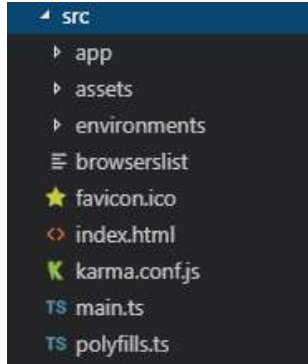


Figure-14 project structure of Angular 4 application

Following table describe the basic use of this files and folders.

File Name	Purpose
E2e/	<p>This folder contain the test cases from live the End-to-End cycle covered.This must be a separate to test your app.</p>  <pre>├─ e2e ├─ src │ ├── app.e2e-spec.ts │ ├── app.po.ts │ ├── protractor.conf.js │ └── tsconfig.e2e.json</pre>
Node_modules	<p>Node.js creates this folder and puts all third party modules listed in package.json .</p>  <pre>├─ node_modules ├── .bin ├── @angular ├── @angular-devkit ├── @babel ├── @ngtools ├── @schematics ├── @types ├── @webassemblyjs ├── @xtuc ├── @yarnpkg ├── abbrev ├── accepts ├── acorn └── acorn-dynamic-import</pre>
Src/	<p>This folder contain different three folders: app, assets , environments</p> <p>It also has other configuration files for src directory.</p>  <pre>├─ src ├── app ├── assets ├── environments ├── browserslist ├── favicon.ico ├── index.html ├── karma.conf.js ├── main.ts └── polyfills.ts</pre>

editor.config	It contains the setting of your editor. It has parameter like style, size of character, line length.It works for UI.
.gitignore	We can define all the folders and files which we want to exclude from our repository in our git.
karma.conf.js	It has configuration for writing unit tests. karma is the test runner and it uses jasmine as framework. These both tester and framework are developed by angular team for writing unit tests.
package.json	The Jason based file contains all the dependency modules which are required for our application. if you want to use _js library or any other library just add name and version of that dependency library in package.json and execute command npm install. It will execute all the dependencies and download in node_modules folder.
README.md	Is contains basic documentation for your project, pre-filled with CLI command information. Just make sure to enhance it with project documentation so that anyone checking out the reputation can build your application.
protector.conf.js	It contains testing configurations.
tsconfig.json	The ts stands for typescripts. Typescripts are used for developing angular applications since Angular 2 came out. It contains the configurations for typescripts.
tslint.json	Used for building application with consistent code style. We can change the configuration that defines how our application should be build.
Src/favicon.ico	Its favicon icon for your website or an application.
Src/index.html	It contains html code with head, and body section. It is starting point of your application.
Src/main.ts	It is starting point of typescript file in your angular application. It contains library which are imported by your angular project.
Src/polyfill.ts	It is used for browser compatibility.
Src/style.css	It has all the styles and css for your angular 4 project.

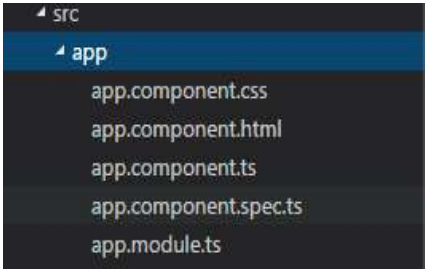
Src/test.ts	This file is used to write unit tests.
Src/tsconfig.app.json	It contains the configuration about how your application should compile.
App/*	<p>This folder contains component and ts files.</p> 
app /app.module.ts	It contains the entire library which are imported and used in your angular 4 application. It's a root module that tells Angular how to assemble the application. Currently it declares only the App_componenet.
app/app.component. {ts,html,css,spec.ts}	It has AppComponent with an HTML template, CSS style sheet, and a unit test. It's the root component of what will become a tree of nested components as the application evolves.
assets/*	In this folder you can put images and anything else to be copied wholesale when you build your application.
environments/*	In this folder one file for each of your destination environments, each exporting simple configuration variables to use in your application. The files are replaced on-the-fly when you build your app. You might use a different API endpoint for development than you do for production or maybe different analytics tokens. You might even use some mock services. Either way, the CLI has you covered.

Table-3 Use of Angular 4 application files and Folder

2.5 ANGULAR DIRECTIVES AND STRING INTERPOLATION

2.5.1 DIRECTIVES

Directives are the most fundamental unit of Angular applications. As a matter of fact, the most used unit, which is a component, is actually a directive. Components are high-order directives with templates and serve as building blocks of Angular applications. Directives in Angular are a js class, which is declared as `@directive`. We have 3 directives in Angular.

1. Components—directives with a template.
2. Structural directives—change the DOM layout by adding and removing DOM elements.
3. Attribute directives—change the appearance or behavior of an element, component, or another directive.

Component Directives

These form the main class having details of how the component should be processed, instantiated and used at runtime.

Structural Directives

A structure directive basically deals with manipulating the dom elements. Structural directives have a `*` sign before the directive. For example, `*ngIf` and `*ngFor`.

Attribute Directives

Attribute directives deal with changing the look and behavior of the dom element. You can create your own directives as shown below.

2.5.2 STRING INTERPOLATION

String interpolation is a part of data binding. Data biding is robust feature of Angular, which allow us to communicate between the component and its view. Data biding can be either one way or two way biding.

In **one-way data binding**, the value of the **Model** is inserted into an **HTML (DOM)** element and there is **no way** to update the **Model** from the **View**. In **two-way binding** automatic synchronization of data happens between the **Model** and the **View**.

String Interpolation also known as Angular Interpolation, uses template expressions in double curly `{{ }}` braces to display data from the component, the special syntax `{{ }}`, also known as **moustache syntax**. The `{{ }}` contains JavaScript expression which can be run by Angular and the output will be inserted into the HTML.

Say if we put `{{ 2 + 2 }}` in the template `4` will be inserted into the HTML

String Interpolation Uses

Display main properties – Interpolation can be used to display and evaluate strings into the text between HTML element tags and within attribute assignments.

Sample Example:

```
<h1>Hello {{ name }}.....! </h1>
```

Evaluate arithmetic expressions – Another usage of interpolation is to evaluate arithmetic expressions present within the curly braces.

Sample Example:

```
<h2>{{3 + 3}}</h2> //outputs 6 on HTML browser
```

Display array items – We can use interpolation along with **ngFor** directive to display an array of items.

DomainObject.ts

```
export class DomainObject
{
  constructor(public id: number, public name: string) {
```

```

    //code
  }
}

app.component.ts

import { DomainObject } from './domain';

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>The name is : {{domainObjectItem.name}}</h2>
    <p>Data Items:</p>
    <ul>
      <li *ngFor="let d of domainObjects">
        {{ d.name }}
      </li>
    </ul>
  `
})
export class AppComponent
{
  title = 'App Title';

  domainObjects = [
    new DomainObject(1, 'A'),
    new DomainObject(2, 'B'),
    new DomainObject(3, 'C'),
    new DomainObject(4, 'D')
  ];

  domainObjectItem = this.domainObjects[0];
}

```

2.6 ANGULAR EVENT

Angular event is known as Angular event binding. Event binding is used to build interactive web application with the flow of data. The flow of data varies from component to element and from element to component.

In some cases user will not only just view the information or data on web application, but also would like to interact with these application using different user action like clicks, keystrokes and change event.

To define event binding syntax will have a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.

Syntax: (name of event)

Example: onclick()

Lets consider an example where we are binding an event Onlick() on button element. When user click on button, event binding listens to the button's click event and calls the components onClick() method.

File name: ex.componet.ts

```
import { Component } from "@angular/core";
```

```
@Component({
```

```
  selector: 'app-ex',
```

```
  template: `
```

```
    <div>
```

```
      <button (click)="onClick()">Click Here!</button>
```

```
    </div>
```

```
    })  
  
    export class ExComponent {  
  
        onClick(){  
  
            alert("You Clicked on button!");  
  
        }  
  
    }  
  
}
```

Angular also provides the different way to handle events.

Target Event Binding

The target event is identified by the name within the parenthesis, ex: (click), which represents the click event. In the example, above we saw the target click event bound to the 'onClick()' method, which will listen to the button's click event.

```
<button (click) = "onClick()">Click me!</button>
```

We can also use the prefix on-, in event binding this is known as canonical form.

```
<button on-click = "onClick()">Click me!</button>
```

If the name of the target event does not match with the element's event, then Angular will throw an error "unknown directive".

2.7 LET US SUM UP

- Angular is a modern web application platform. That promises to provide developers with a comprehensive set of tools and capabilities to build large, complex and robust applications
- The Angular quality and complexity of applications is ever increasing, and so are users expectations for quality and capabilities

- Angular provides a common developer experience through its CLI tooling which includes the generating, building, testing and deploying apps.
- Angular supports number of third-party libraries, UI libraries, blog posts, and events
- Angular comes with a leaner core library and makes additional features available as separate packages that can be used as needed.
- Angular4 uses TypeScript 2.2 version whereas Angular 2 uses TypeScript version 1.8
- To setup the Angular 4 need to install nodejs and npm.
- Angular works on command thatswhy need the CLI.
- Angular project is known as workspace.
- The description knowledge of files and folder of angular project is important.
- All the files and folder have a meaning ful files
- Angular supports the three types of directives.
- String Interpolation also known as Angular Interpolation, uses template expressions in double curly {{ }} braces to display data from the component,
- String Interpolation also evaluated the string expression.

2.8 CHECK YOUR PROGRESS

Give the answer of the following MCQ.

1. _____ is a modern web application platform.

A. JS	C. ANGULARJS
B. ANGULAR	D. JAVA
2. Angular sponsored by the _____.

A. Yahoo	C. Google
B. Bing	D. Amazon
3. Angular CLI need the nodejs and _____.

A. HTTP	C. NPM
B. MMP	D. DSN

4. To serve the angular _____ command is used.

A. ng serve	C. ng
B. serve	D. ds Server

5. Angular events known as angular event _____.

A. Binding	C. Serve
B. Lunching	D. Start

6. _____ can be used to display and evaluate strings into the text between HTML element tags and within attribute assignments.

A. String	C. Binding
B. Interpolation	D. Event

7. A _____ directive basically deals with manipulating the dom elements.

A. Attribute	C. Structure
B. DOM	D. Property

8. Angular Supports the _____ types of directives.

A. 3	C. 2
B. 4	D. 5

9. _____ folder contain different three folders: app, assets , environments
It also has other configuration files for src directory.

A. Src	C. Main
B. App	D. Project

2.9 CHECK YOUR PROGRESS: POSSIBLE ANSWER

Give the answer of the following MCQ.

- | | | |
|------------|------------------|--------------|
| 1. ANGULAR | 4. ng serve | 7. Structure |
| 2. Google | 5. Binding | 8. 3 |
| 3. NPM | 6. Interpolation | 9. Src |

2.10 ASSIGNMENTS

Write the answer for the following questions.

1. What is Framework?
2. State the reason: Angular is a platform not a framework.
3. Write down the steps for angular setup.
4. Explain the file structure of angular application.
5. Explain the string interpolation with example.
6. Explain the event with sample code.

Unit 3: Introduction to MVC

3

Unit Structure

- 3.1 Learning Objectives
- 3.2 Design Pattern
- 3.3 The Model View Controller
- 3.4 Introduction to the Pipes
- 3.5 Custom Pipes
- 3.6 Event binding
- 3.7 Let Us Sum Up
- 3.8 Check Your Progress
- 3.9 Check Your Progress: Possible Answer
- 3.10 Assignments

3.1 LEARNING OBJECTIVES

After studying this chapter, students should be able to understand:

- The complete design pattern of MVC.
- Architecture of MVC in detail.
- Various use of pipes in angular applications.
- How to create the custom pipes.
- How to handle the user interaction on screen means event handling.

3.2 DESIGN PATTERN

Angular (Angular 2, 4, 5, 6...) is a new framework completely rewritten from the ground up, replacing the now well-known AngularJS framework (Angular 1.x). More than just a framework, Angular should be considered as a whole *platform* which comes with a complete set of tools like its own [CLI](#), [debug utilities](#) or [performance tools](#).

Angular was designed for the use of design patterns you may not be accustomed to, like **reactive programming**, **unidirectional data flow** and **centralized state management**.

Reactive programming

The Angular is now a reactive system by design. Although you are not forced to use reactive programming patterns, they make the core of the framework and it is definitely recommended to learn them if you want to leverage the best of Angular. Angular uses RxJS to implement the Observable pattern. An Observable is a stream of asynchronous events that can be processed with array-like operators.

Unidirectional data flow

The AngularJS where one of its selling points was two-way data binding which ended up causing a lot of major headaches for complex applications, Angular now enforces unidirectional data flow. It means that change detection cannot cause cycles, which was one of AngularJS problematic points. It also helps to maintain

simpler and more predictable data flows in applications, along with substantial performance improvements.

Centralized state management

As applications grow in size may be complex, keeping track of the all its individual components state and data flows can become tedious, and tend to be difficult to manage and debug. The main goal of using a centralized state management is to make state changes predictable by imposing certain restrictions on how and when updates can happen, using unidirectional data flow.

3.3 THE MODEL VIEW CONTROLLER

Generally AngularJS follows the MVC architecture, the diagram of the MVC framework as shown below.

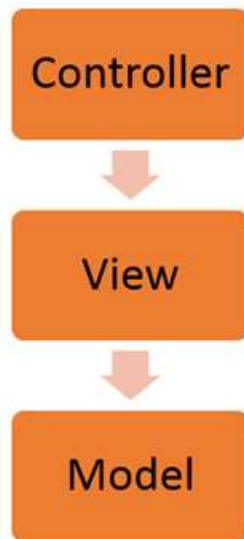


Figure-15 MVC Framework

Models are used to represent your real data. The data in your model can be as simple as just having primitive declarations. If you are maintaining a employee application, your data model could just have a empid and empname.

Views are used to represent the presentation layer which is provided to the end users.

The Controller represents the layer that has the business logic. User events trigger the functions which are stored inside your controller. The user events are part of the controller.

Angular Architecture

Depends on the Component classes. The Architecture of an Angular Application is based on the idea of Components. An Angular application starts with a Top level component called Root Component. Every Angular application has at least one component, the *root component* that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML *template* that defines a view to be displayed in a target environment.

The [@Component\(\)](#) decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Main part of the development with Angular 4 is done in the components. Components are basically classes. That classes are interact with the .html file of the component, which gets displayed on the browser. The file structure has the app component and it consists of the following files

- app.component.css
- app.component.html
- app.component.spec.ts
- app.component.ts
- app.module.ts

if we open the app.module.ts which shows some libraries which are imported like

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

The declaration becomes the parent component. This include the AppComponent variable. This is in build component. If you would like to create a component form command line then follows this syntax.

ng g component new-cmp

When you run the above command in the command line, you will receive the following output

installing component

```

create src\app\new-cmp\new-cmp.component.css
create src\app\new-cmp\new-cmp.component.html
create src\app\new-cmp\new-cmp.component.spec.ts
create src\app\new-cmp\new-cmp.component.ts
update src\app\app.module.ts

```

The following files are created in the new-cmp folder

- new-cmp.component.css – css file for the new component is created for the style.
- new-cmp.component.html – html file is created for the Interface.
- new-cmp.component.spec.ts – this can be used for unit testing.
- new-cmp.component.ts – used to define the module, properties, etc.

3.4 INTRODUCTION TO PIPES

Pipes were earlier called filters in Angular1 and called pipes in Angular 2 and 4+. Sometimes application starts out with what seems like a simple task: get data, transform them, and display them to users. Getting data could be as simple as creating a local variable or as complex as streaming data over a WebSocket.

Once data arrives, you could push their raw `toString` values directly to the view, but that rarely makes for a good user experience.

For example, in most use cases, users prefer to see a date in a simple format like April 15, 2019 rather than the raw string format Fri Apr 15 2019 00:00:00 GMT-0700 (Pacific Daylight Time).

Angular pipes is used display-value transformations that you can declare in your HTML. The `|` (pipe sign) character is used to transform data.

Example:

```
{{ Hello World | lowercase}}
```

Using Pipes

Pipes takes any integers, strings, arrays, and date as input separated with `|` to be converted in the format as required and display the same in the browser.

In the **app.component.ts** file, we have defined the title variable

```
export class AppComponent {
  title = 'hello-world';
}
```

The following line of code goes into the **app.component.html** file.

```
<h1>
  Welcome to {{ title | uppercase }}!
</h1>
<h1>
  Welcome to {{ title | lowercase }}!
</h1>
```

Will convert the title variable in uppercase and lowercase.

A pipe takes in data as input and transforms it to a desired output. In next example pipes to transform a component's birthday property into a human-friendly date.

In component.ts file

```
export class AppComponent {  
  birthday = new Date(2019, 3, 15); // April 15, 2019  
}
```

Component.html will be look like

```
<h1>  
  Todays date is {{birthday | date}}!  
</h1>
```

Built-in Pipes

Angular comes with a stock of pipes such as [DatePipe](#), [UpperCasePipe](#), [LowerCasePipe](#), [CurrencyPipe](#), and [PercentPipe](#). They are all available for use in any template.

Pipe	Usage	Example
DatePipe	date {{ dateObj date }}	output is 'Jun 15, 2015'
UpperCasePipe	uppercase {{ value uppercase }}	output is 'SOMETEXT'
LowerCasePipe	lowercase {{ value lowercase }}	output is 'sometext'
CurrencyPipe	currency {{ 31.00 currency:'USD':true }}	output is '\$31'
PercentPipe	percent {{ 0.03 percent }}	output is %3

Table-4 Usage of Angular Pipes

Example of Built in pipes

Component.ts file

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.less']
})

export class AppComponent {
  public fName: string = 'Nirav';
  public lName: string = 'Suthar';
  public reservationMade: string = '2016-06-22T07:18-08:00'
  public reservationFor: string = '2025-11-14';
  public cost: number = 99.99;
}
```

Component.html file

```
<div style="text-align:center">
  <h1>Welcome back {{fName | uppercase}} {{lName | lowercase}}</h1>
  <p>
    On {reservationMade | date} at {reservationMade | date:'shortTime'} you
    reserved room 205 for {reservationDate | date} for a total cost of
    {cost | currency}.
  </p>
</div>
```

Output

Welcome back Nirav Suthar

On Jun 26, 2016 at 7:18 you reserved room 205 for Nov 14, 2025 for a total cost of \$99.99.

Parameterizing a pipe

A pipe can accept any number of optional parameters to output. To add parameters to a pipe, follow the pipe name with a colon (:) and then the parameter value (such as currency:'EUR'). If the pipe accepts multiple parameters, separate the values with colons (such as slice:1:5)

Component.html file

```
<p>Your birthday is {{ birthday | date:"MM/dd/yy" }} </p>
```

The parameter value can be any valid template expression, such as a string literal or a component property. In other words, you can control the format through a binding the same way you control the birthday value through a binding.

Chaining Pipe

Pipes may be chained. The combination of more than one pipe is called chaining pipe. we can use chain pipes together in potentially useful combinations.

Component.html

```
<p>your birthday is  
{{ birthday | date | uppercase}}</p>
```

This example—which displays **FRIDAY, APRIL 15, 2019** chains the same pipes as above with passes in a parameter to date as well.

```
<p>your birthday is  
{{ birthday | date:'fullDate' | uppercase}}</p>
```

3.5 CUSTOM PIPE

You can write your own custom pipes. To create a custom pipe, we have to create a new **ts** file.

Steps to create a new custom pipe:

1. Create a new ts file which you want to use as pipe.
2. Add custom created pipe in to module.ts file
3. Use pipe name in component.html file.

Below example shows how to create a custom pipe.

Here, we want to create the **cube** custom pipe. We have given the same name to the file and it looks as follows

app.cube.ts

```
import {Pipe, PipeTransform} from '@angular/core';
@Pipe ({
  name : 'cube'
})
export class cubePipe implements PipeTransform {
  transform(val : number) : number {
    return Math.cbrt(val);
  }
}
```

To create a custom pipe, we have to import Pipe and Pipe Transform from Angular/core. In the @Pipe directive, we have to give the name to our pipe, which will be used in our .html file. Since, we are creating the cube pipe, we will name it cube.

As we proceed further, we have to create the class and the class name is **cubePipe**. This class will implement the **PipeTransform**.

we need to add the same in **app.module.ts**. This is done as follows –

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

import { NewCmpComponent } from './new-cmp/new-cmp.component';
import { ChangeTextDirective } from './change-text.directive';
import { cubePipe } from './app.cube';
```

```

@NgModule({
  declarations: [
    cubePipe,
    AppComponent,
    NewCmpComponent,
    ChangeTextDirective
  ],

  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

now to call cube file in app.component.html file

```

<h1>Custom Pipe</h1>
<b>Cube root of 64 is: {{64 | cube}}</b>
<br/>

```

Will give the output as Cube root of 64 is 4

3.6 EVENT BINDING

Event binding refers to the data binding. Data binding is very important feature of Angular. Data binding is used for front-end framework, Interpolation and property binding in Angular.

Data can be bind in form of Event binding and two-way binding.

Event Binding in Angular

A user expects a UI to respond to her/his actions on the single page. Every such action would trigger an event on the page and the page has to respond by

listening to these events. The event binding system provides us the way to attach a method defined in a component with an event. Event binding is built on top of the events defined in the DOM objects.

With Angular, there are two ways to handle an event on an HTML element. The following line of code shows both the ways for handling a click event on a button:

```
<button class='btn' (click)='save()'>Save</button>  
<button class='btn' on-click='save()'>Save</button>
```

The method `save` has to be defined in the component class. Any DOM event can be either prefixed with `on-` or can be enclosed inside parentheses to bind it with a method in the component class.

Functionally there is no difference in the two ways of handling events, you can choose the one you like, and use it. As the process of handling events doesn't involve any abstractions which has to be written for any event on the page, this model is extensible. Any new events added to a DOM element can be bound to functions in the component using this syntax without writing any piece of new code. Events can be used to run a piece of logic based on the action taken by the user. It may include changing values of a few fields in the component, posting data to a REST API, moving to a different page or anything else. As the events directly correspond to the browser's events, every event adds an entry to the event loop.

All of this means Angular's change detection cycle runs whenever an event is triggered. So, any values modified by the event handler will be detected by the change detection system and the changes are applied on the page.

The following component shows an example of event binding:

```
@Component({  
  selector: 'app-demo',  
  template: `

{{submitText}}</div>  
  <button class="btn" (click)='save()'>Save</button>`


```

```

})
export class DemoComponent {
  public submitText: string = 'Not submitted yet.';

  save(){
    this.saveText = 'Submitted successfully!';
  }
}

```

The above component has a *div* element and a button. The *div* element has an interpolation applied, text of the interpolated expression is modified in the click event handler of the button. You will see that the text inside the expression is modified after clicking the button.

If an event has to perform a single action like changing a simple value, it can be done in the HTML instead of writing a separate method. The following snippet shows an example:

```
<button class='btn' (click)='color="green"'>Save</button>
```

But if the event involves a few more lines of logic, it should be kept outside HTML to separate the concerns. This way, the code in the event handler can be unit tested as well. The event handling method gets access to the event object, which is same as the object passed into any DOM event handler. This object gives us access to the information about the event. The following snippet shows how to pass the event object from HTML to the event handling object:

```
<button class='btn' (click)='save($event)'>Save</button>
```

The *\$event* object is same as the object that the browser sends when an event is triggered on an HTML element. We can get details of the event like source of the event, target element, type of the event, co-ordinates on the page and screen where the event triggered, as well as many other details.

The following snippet shows a component handling two events on a button:

```
@Component({
  selector: 'app-demo',
  template: `<div>{{saveText}}</div>
  <button class="btn" (click)='save($event)'
(mousemove)="mouseMove($event)">Save</button>
  <div>{{x}} {{y}}</div>`
})
export class DemoComponent {
  public submitText: string = 'Not submitted yet.';
  public x: number;
  public y: number;

  submit($event: Event){
    this.saveText = 'Submitted successfully!';
    console.log($event);
  }

  mouseMove($event: MouseEvent) {
    this.x = $event.x;
    this.y = $event.y;
  }
}
```

The component in the above snippet has a button, on which the click and mouse move events are handled using methods in the component. The click event changes the text displayed in the div element and the mouse move event updates values of the co-ordinates displayed on the screen.

Two-way Binding in Angular

The feature two-way binding in Angular is derived from the property and event bindings. The property and event bindings are directed one way with the former receiving data into view from the component object and the later sending data from

the view to the component. The two-way binding is a combination of these two bindings; it gets the data from the component object to the view and sets the data from view to the component object.

The following line of code shows an example of a directive, *ngModel* to show how two-way binding can be used:

```
Name: <input type="text" [(ngModel)]="name" />
<div>{{name}}</div>
```

The field name is two-way bound on the input box. When it is rendered on a page, it shows the existing value of the field and when the value is modified on the screen, it updates the value in the field. The change in the value of the field is immediately reflected in the interpolation in the div element next to the input control. To use the *ngModel* directive, the *FormsModule* has to be added to the application's module. The following snippet shows the application module using the *FormsModule*:

```
import { NgModule }    from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }  from '@angular/forms'; // Importing forms module to this
file

import { AppComponent }    from './app.component';
import { DemoComponent }  from './demo.component';
@NgModule({
  imports: [
    BrowserModule,
    FormsModule // Importing forms module to application module
  ],
  declarations: [
    AppComponent,
    DemoComponent
  ],
  bootstrap: [ AppComponent ]
})
```

```
})  
export class AppModule { }
```

The following component uses the *ngModel* directive:

```
@Component({  
  selector: 'app-demo',  
  template: `  
    Name: <input type='text' [(ngModel)]='name' />  
    <div>Name is: {{name}}</div>  
  `,  
})  
export class DemoComponent {  
  public name: string = "Virat";  
}
```

The textbox is bound to the field *name* in the template. When you type something in the textbox, you will see that the content in the div changes automatically.

3.7 LET US SUM UP

- Angular (Angular 2, 4, 5, 6...) is a new framework completely rewritten from the ground up, replacing the now well-known AngularJS framework (Angular 1.x)
- Angular was designed for the use of design patterns
- The Angular is now a reactive system by design
- The AngularJS where one of its selling points was two-way data binding which ended up causing a lot of major headaches for complex applications, Angular now enforces unidirectional data flow.
- The main goal of using a centralized state management is to make state changes predictable by imposing certain restrictions on how and when updates can happen, using unidirectional data flow.
- Models are used to represent your real data. The data in your model can be as simple as just having primitive declarations. If you are maintaining a

employee application, your data model could just have a empid and empname.

- Views are used to represent the presentation layer which is provided to the end users.
- The Controller represents the layer that has the business logic. User events trigger the functions which are stored inside your controller. The user events are part of the controller.
- Architecture of an Angular Application is based on the idea of Components. An Angular application starts with a Top level component called Root Component
- The `@Component()` decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.
- Pipes were earlier called filters in Angular1 and called pipes in Angular 2 and 4+. Sometimes application starts out with what seems like a simple task: get data, transform them, and display them to users.
- Pipes may be a user pipes, built-in pipes and custom pipes.
- Event binding refers to the data binding. Data binding is very important feature of Angular. Data binding is used for front-end framework, Interpolation and property binding in Angular.
- The event binding system provides us the way to attach a method defined in a component with an event. Event binding is built on top of the events defined in the DOM objects.

3.8 CHECK YOUR PROGRESS

1. The Angular is now a _____ system by design.

A. reactive	C. structure
B. template	D. dynamic
2. MVC stands for _____.

3. _____ are used to represent the presentation layer which is provided to the end users.
- A. Model
 - B. View
 - C. Controller
 - D. Server
4. The _____ represents the layer that has the business logic
- A. Model
 - B. View
 - C. Controller
 - D. Layer
5. DOM stands for _____.
6. Components are basically _____.
- A. Union
 - B. Objects
 - C. Classes
 - D. Structure
7. Pipes are also known as _____.
- A. filters
 - B. looping
 - C. conditions
 - D. comments
8. _____ takes any integers, strings, arrays, and date as input separated with | to be converted in the format as required and display the same in the browser.
- A. looping
 - B. comments
 - C. pipes
 - D. var
9. A pipe can accept any number of optional parameters to output is known as _____
- A. parameter pipe
 - B. custom pipe
 - C. built-in pipes
 - D. pipe
10. Create a new ts file which you want to use as pipe is a part of _____.
- A. parameter pipe
 - B. custom pipe
 - C. built-in pipes
 - D. pipe

3.9 CHECK YOUR PROGRESS:POSSIBLE ANSWER

- | | |
|--------------------------|-------------------|
| 1. reactive | 6. Classes |
| 2. Model View Controller | 7. Filters |
| 3. View | 8. pipes |
| 4. Controller | 9. parameter pipe |
| 5. Document Object Model | 10. custom pipe |

3.10 ASSIGNMENTS

Write the answer for the following questions.

1. Explain the MVC in detail.
2. What is pipes in angular.
3. Explain how to use pipes and also explain the built in pipes with example.
4. What is custom pipes write down the sample code for it.
5. What is event binding? Explain the two way event binding in detail.

Unit 4: Angular Directives

4

Unit Structure

- 4.1 Learning Objectives
- 4.2 Introduction to Directives
- 4.3 Using the Directives
- 4.4 Structure Directives
- 4.5 Attribute Directives
- 4.6 Let Us Sum Up
- 4.7 Check Your Progress
- 4.8 Check Your Progress: Possible Answer
- 4.9 Assignments

4.1 LEARNING OBJECTIVES

After studying this chapter, students should be able to understand:

- What is Directives and how useful in angular app.
- How to use the directives.
- The use of Ngif, Ngfor and Ngswitch with example.
- How to create a different CSS files to define the style in HTML.

4.2 INTRODUCTION

In a programming structure class is very important concept. Directives in Angular is a js class which is declared as `@directive`. A directive is a custom HTML element that is used to extend the power of HTML. Angular provides a number of built-in directives, which are attributes we add to our HTML elements that give us dynamic behavior.

Angular Directive is basically a class with a `@Directive` decorator. **Decorators** are functions that modify JavaScript classes. Decorators are used for attaching metadata to classes, it knows the configuration of those classes and how they should work. a component is also a directive-with-a-template. A `@Component decorator` is actually a `@Directive decorator` extended with template-oriented features. Angular renders a directive, it changes the DOM according to the instructions given by the directive. Directive appears within an element tag similar to attributes.

There are three kinds of directives in Angular:

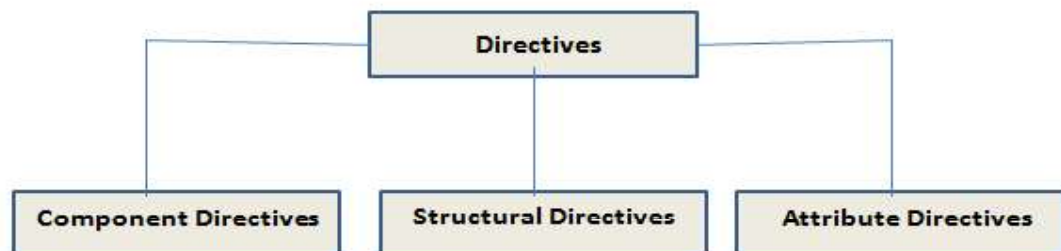


Figure-16 Angular Directives

1. Components directives—directives with a template, how the component should be processed, instantiated and used at runtime..
2. Structural directives—change the DOM layout by adding and removing DOM elements. It is denoted by using * sign.
3. Attribute directives—change the appearance or behavior of an element, component, or another directive.

4.3 USING COMPONENT DIRECTIVES

The Component processed is processed by component directives. It is mainly used to specify the HTML templates. It is the most commonly used directive in an Angular project. It is decorated with the `@component` decorator. This directive is a class. The component directive is used to specify the template/HTML for the Dom Layout. Its built-in is `@component`.

- `app.component.css`: contains all the CSS styles for the component
- `app.component.html`: contains all the HTML code used by the component to display itself
- `app.component.ts`: contains all the code used by the component to control its behavior

A root component is the first Angular component that gets bootstrapped when the application runs. Two things are special about this component:

First, if you open the application module file `src/app/app.module.ts`

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Added to the bootstrap array of the module definition.

Second, if you open the src/index.html file

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>HelloWorld</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
  <my-app> // component directives
    Hello This shows the component directives
  </my-app>
  <div id="print-section">
</div>
  <button (click)="print()">print</button>
  <p>Hello Worldssssssssssssss</p>
</body>
</html>
```

it's called inside the document <body> tag.

Now, let's open the component file src/app/app.component.ts,

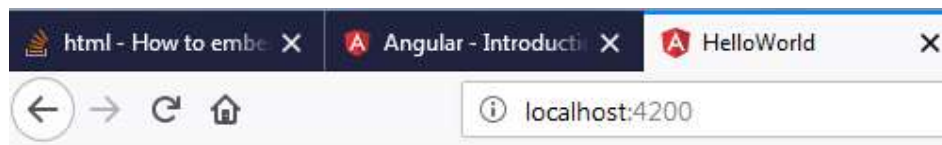
```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h2>{{title}}</h2>
<p *ngIf="showElement">Show Element</p>
<div [ngSwitch]="inpvalue">
<p style='color:blue' *ngSwitchCase="1">You have selected Aadhar Card</p>
<p style='color:blue' *ngSwitchCase="2">You have selected Passport</p>
<p style='color:blue' *ngSwitchCase="3">You have selected Voter ID</p>
<p style='color:red' *ngSwitchDefault>Sorry! Invalid selection....</p>
```

```
</div>`  
})  
  
export class AppComponent {  
  inpvalue: number = 4;  
}
```

In decorator part that is `@component` is the component decorator. We had customized own selector `my-app` to map in HTML files. In the template, I put the `name` property which will fetch its value from the `name` string from `AppComponent` class.

Output



4.4 STRUCTUREL DIRECTIVES

The structural directive is used to add or remove the HTML Element in the Dom Layout, typically by adding, removing, or manipulating elements... Its built-in types are `*NgIf`, `*NgFor`, `*NgSwitch`. Structural directives are easy to recognize by using an asterisk (*).

Types of built-in structural directive

- `NgIf`
- `NgFor`
- `NgSwitch`

NgIf

It is used to create or remove a part of the DOM tree depending on a condition.

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<div style='color:blue' *ngIf="true">You can See Passport....</div>`
})

export class AppComponent {
  inpvalue: number = 2;
}

```

Here If ngIf= true the text will be visible on the web page.

NgFor

It is used to customize data display. It is mainly used for displaying a list of items using repetitive loops.

App.component.ts file

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
})

export class AppComponent {
  cust: any[] = [
    {
      code: '1001', name: 'Nirav', gender: 'Male',
      total: 1255, dateOfBirth: '25/6/1990'
    },
    {
      code: '1002', name: 'Pooja', gender: 'Female',
      total: 1355, dateOfBirth: '9/6/1992'
    }
  ]
}

```

```
    },  
    {  
      code: '1003', name: 'Nishant', gender: 'Male',  
      total: 1455, dateOfBirth: '12/8/1995'  
    },  
    {  
      code: '1004', name: 'Neha', gender: 'Female',  
      total: 1555, dateOfBirth: '14/10/1989'  
    },  
  ],  
}
```

App.component.html

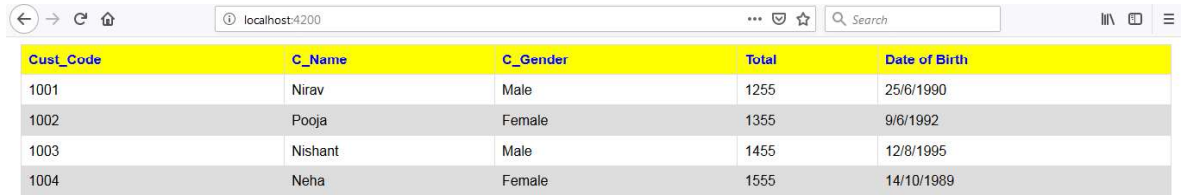
```
<!DOCTYPE html>  
<html>  
<head>  
  <title></title>  
  <meta charset="utf-8" />  
  <style>  
    table {  
      font-family: arial, sans-serif;  
      border-collapse: collapse;  
      width: 100%;  
    }  
  
    td, th {  
      border: 1px solid #dddddd;  
      text-align: left;  
      padding: 8px;  
    }  
  </style>  
</head>  
</html>
```

```

tr:nth-child(even) {
    background-color: #dddddd;
}
</style>
</head>
<body>
<table align="center" border="1" cellpadding="4" cellspacing="4">
  <thead>
    <tr>
      <th style="background-color: Yellow;color: blue">Cust_Code</th>
      <th style="background-color: Yellow;color: blue">C_Name</th>
      <th style="background-color: Yellow;color: blue">C_Gender</th>
      <th style="background-color: Yellow;color: blue">Total</th>
      <th style="background-color: Yellow;color: blue">Date of Birth</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor='let c of cust'>
      <td>{{c.code}}</td>
      <td>{{c.name}}</td>
      <td>{{c.gender}}</td>
      <td>{{c.total}}</td>
      <td>{{c.dateOfBirth}}</td>
    </tr>
  </tbody>
</table>
</body>
</html>

```


Output



Cust_Code	C_Name	C_Gender	Total	Date of Birth
1001	Nirav	Male	1255	25/6/1990
1002	Pooja	Female	1355	9/6/1992
1003	Nishant	Male	1455	12/8/1995
1004	Neha	Female	1555	14/10/1989

Ngswitch

A structural directive that adds or removes templates when the next match expression matches the switch expression.

The `[ngSwitch]` directive on a container specifies an expression to match against. The expressions to match are provided by `ngSwitchCase` directives on views within the container.

- Every view that matches is rendered.
- If there are no matches, a view with the `ngSwitchDefault` directive is rendered.
- Elements within the `[NgSwitch]` statement but outside of any `NgSwitchCase` or `ngSwitchDefault` directive are preserved at the location.

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h2>{{title}}</h2>

  <div [ngSwitch]="inpvalue">
  <p *ngSwitchCase="1">Monday</p>
  <p *ngSwitchCase="2">Tuesday</p>
  <p *ngSwitchCase="3">Wednesday</p>
  <p *ngSwitchDefault>Sorry Invalid selection!!</p>
  </div>`
})
export class AppComponent {
```

```
inpvalue: number = 1;
}
```

Output



4.5 ATTRIBUTE DIRECTIVES

The attribute directive changes the appearance or behavior of a DOM element. These directives look like regular HTML attributes in templates. The *ngModel* directive which is used for two-way is an example of an attribute directive. Some of the other attribute directives are listed below:

- **NgStyle:** Based on the component state, dynamic styles can be set by using *NgStyle*. Many inline styles can be set simultaneously by binding to *NgStyle*.
- **NgClass:** It controls the appearance of elements by adding and removing CSS classes dynamically.

NgStyle

Ngstyle used to change the look and feel of elements. The *NgStyle* directive lets you set a given DOM elements style properties. One way to set styles is by using the *NgStyle* directive and assigning it an object literal.

```
<div [ngStyle]='{'background-color':'red'}'></div>
```

This will change the div background color to red.

ngStyle is a very useful when its value is dynamic. The *values* in the object literal that we assign to *ngStyle* can be javascript expressions which are evaluated and the result of that expression is used as the value of the css property.

like this:

```
<div [ngStyle]='{"background-color":person.country === 'USA' ? 'green' : 'red'}"></div>
```

The above code uses the ternary operator to set the background color to green if the persons country is the USA else red. But the expression doesn't have to be *inline*, we can call a function on the component instead.

To demonstrate this lets define full example.

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
</head>
<body>

  <div [style.background-color]="yellow">
    Uses fixed yellow background
  </div>
</body>
</html>
```

Output



Another way to set fixed values is by using the NgStyle attribute and using key value pairs for each property you want to set, like this:

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
```

```

</head>
<body>

  <div [ngStyle]="{color: 'white', 'background-color': 'blue'}">

    Uses fixed yellow background

  </div>
</body>
</html>

```

Here we are setting both the color and the background-color properties. But the real power of the NgStyle directive comes with using dynamic values.

In our example, we are defining two input boxes with an apply settings button:

```

<!DOCTYPE html>
<html>
<head>
  <title></title>
</head>
<body>

  <div class="ui input">
    <input type="text" name="color" value="{{color}}" #colorinput>
  </div>

  <div class="ui input">
    <input type="text" name="fontSize" value="{{fontSize}}" #fontinput>
  </div>

  <button class="ui primary button" (click)="apply(colorinput.value,
fontinput.value)">
    Apply settings
  </button>
</body>

```

```
</html>
```

This will apply the color and fontsize after apply button click on element.

NgClass

The NgClass directive, represented by a ngClass attribute in your HTML template, allows you to dynamically set and change the CSS classes for a given DOM element.

The first way to use this directive is by passing in an object literal. The object is expected to have the keys as the class names and the values should be a truthy/falsy value to indicate whether the class should be applied or not.

Let's assume we have a CSS class called bordered that adds a dashed black border to an element:

Src/style.css

```
.bordered {  
border: 1px dashed black;  
background-color: #eee; }
```

Let's add two div elements: one always having the bordered class (and therefore always having the border) and another one never having it:

component.html

```
<div [ngClass]="{bordered: false}">This is never bordered</div>  
<div [ngClass]="{bordered: true}">This is always bordered</div>
```

4.6 LET US SUM UP

- Directives in Angular is a js class which is declared as @directive. A directive is a custom HTML element that is used to extend the power of HTML. Angular provides a number of built-in directives, which are attributes we add to our HTML elements that give us dynamic behavior.
- Decorators are functions that modify JavaScript classes. Decorators are used for attaching metadata to classes, it knows the configuration of those classes and how they should work. a component is also a directive-with-a-template.
- Three types of directives-Component, Structural, Attribute

- Components directives—directives with a template, how the component should be processed, instantiated and used at runtime..
- Structural directives—change the DOM layout by adding and removing DOM elements. It is denoted by using * sign.
- Attribute directives—change the appearance or behavior of an element, component, or another directive.
- The component directive is used to specify the template/HTML for the Dom Layout.
- The structural directive is used to add or remove the HTML Element in the Dom Layout, typically by adding, removing, or manipulating elements... Its built-in types are *NgIf,*NgFor,*NgSwitch. Structural directives are easy to recognize by using an asterisk (*).
- The attribute directive changes the appearance or behavior of a DOM element. These directives look like regular HTML attributes in templates(Ngstyle,Ngclass)

4.7 CHECK YOUR PROGRESS

1. _____ in Angular is a js class which is declared as @directive.

A. Style	C. Structure
B. Template	D. Directives

2. Directives are declared by using _____ symbol.

A. \$	C. !
B. @	D. &

3. _____ directives with a template, how the component should be processed, instantiated and used at runtime.

A. Component	C. Attribute
B. Structure	D. Class

4. _____ change the appearance or behavior of an element, component, or another directive.

A. Component	C. Attribute
B. Structure	D. Class

4.9 ASSIGNMENTS

Write the answer for the following questions.

1. What is directives?
2. Explain the component directive in detail with example.
3. Explain the Structurel directive in detail with example.
4. Explain the attribute directive in detail with example.

Unit 5: Working with Forms

5

Unit Structure

- 5.1 Learning Objectives
- 5.2 Introduction to Forms
- 5.3 Key Difference
- 5.4 Reactive Form Use
- 5.5 Template Driven Form Use
- 5.6 Form Validation
- 5.7 Let Us Sum Up
- 5.8 Check Your Progress
- 5.9 Check Your Progress: Possible Answer
- 5.9 Assignments

5.1 LEARNING OBJECTIVES

After studying this chapter, students should be able to understand:

- The importance of Form in application.
- Difference between the forms types.
- Use of Reactive form and Template driven form with code
- How to collect the correct information from form using validation.

5.1 INTRODUCTION

Forms are critical to any modern front-end application, and they're a feature that we use every day, even if don't realize it. Forms are required for securely logging in a user to the app, searching for all the available hotels in a particular city, booking a cab, building a to-do list, and doing tons of other things that we are used to. Some forms have just a couple of input fields, whereas other forms could have an array of fields that stretch to a couple of pages or tabs.

Interface,Forms are the most important aspect of any web application. To handle the various events of user need to design a meaningful form. Events get in like from clicking on links or moving the mouse it is through forms where we get the crucial data inputs from the end users.

Angular, being a full-fledged front-end framework, has its own set of libraries for building complex forms. The latest version of Angular has two powerful form-building strategies. They are:

- Reactive forms - more scalable, reusable, and testable
- Template-driven forms - adding a simple form to an app, such as an email list signup form

Both the technologies belong to the `@angular/forms` library and are based on the same form control classes. They differ remarkably in their philosophy, programming style, and technique. Choosing one over the other depends on your personal taste and also on the complexity of the form that you are trying to create. In

my opinion, you should try both the approaches first and then choose one that fits your style and the project at hand.

5.3 KEY DIFFERENCE INTRODUCTION

The table below summarizes the key differences between reactive and template-driven forms.

Differs	Reactive	Template-driven
Setup	More explicit, created in component class	Less explicit, created by directives
Data model	Structured	UnStructured
Predictability	Synchronous	Asynchronous
Form validation	Functions	Directives
Mutability	Immutable	Mutable
Scalability	Low-level API access	Abstraction on top of APIs
Use	More flexible, but needs a lot of practice	Easy to use
Data Binding	No data binding is done	Two way data binding
Testing	Easier unit testing	Unit testing is another challenge

Table-5 key differences between reactive and template-driven forms

Both reactive and template-driven forms share underlying building blocks.

- FormControl tracks the value and validation status of an individual form control.
- FormGroup tracks the same values and status for a collection of form controls.
- FormArray tracks the same values and status for an array of form controls.
- ControlValueAccessor creates a bridge between Angular FormControl instances and native DOM elements.

5.4 REACTIVE FORMS

Reactive forms are more powerful. They are more scalable, reusable, and testable. If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.

A *reactive* form is just an HTML form that's been wired up with RxJS to **manage its state as a realtime stream**. This means you can listen to changes to its value as an [Observable](#) and react accordingly with validation errors, feedback, database operations, etc. Each change to the form state returns a new state, which maintains the integrity of the model between changes. Reactive forms are built around observable streams, where form inputs and values are provided as streams of input values, which can be accessed synchronously. Reactive forms also provide a straightforward path to testing because you are assured that your data is consistent and predictable when requested. Any consumers of the streams have access to manipulate that data safely.

5.4.1 CREATING A FORM

To create a reactive form we have to import certain modules and need to generate form controls. So for that follow the below steps.

Step 1 : Registering the reactive module.

To use the reactive forms in angular need to import `ReactiveFormsModule` from the angular `/forms` package and add it to `NgModules` import array.(`app.module.ts`)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
```

```

declarations: [
  AppComponent,
],
imports: [
  BrowserModule,
  AppRoutingModule,
  ReactiveFormsModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }

```

Step 2: Generating and importing new form control

To generate a component for the control run the following command.

ng generate component NameEditor

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
C:\Users\Wirav\react\src\app>ng generate component NameEditor
CREATE src/app/name-editor/name-editor.component.html (30 bytes)
CREATE src/app/name-editor/name-editor.component.spec.ts (657 bytes)
CREATE src/app/name-editor/name-editor.component.ts (288 bytes)
CREATE src/app/name-editor/name-editor.component.css (0 bytes)
UPDATE src/app/app.module.ts (572 bytes)
C:\Users\Wirav\react\src\app>

```

The main class called FormControl class which used to define building blocks. For the single form control import the FormControl class into your component class and create a new instance to save as class property.

Step 3: Registering the controls in the Form

Once you created the control in the component class you must associated with form control element in the template. Update the template with the form control using the

formControl binding provided by [FormControlDirective](#) included in [ReactiveFormsModule](#).(nameeditor.component.html)

```
<label>
  Name:
  <input type="text" [formControl]="name">
</label>
```

5.4.2 MANAGING CONTROL VALUES

Powerful Reactive forms give you access to the form control state and value at a point in time. Any one can manipulate the current state and value through the component class or the component template. The following examples display the value of the form control instance and change it.

Displaying a form control value

To display the value in follow these ways:

- Through the valueChanges observable where you can listen for changes in the form's value in the template using [AsyncPipe](#) or in the component class using the subscribe() method.
- With the value property. which gives you a snapshot of the current value.

(name-editor.component.html)

```
<label>
  Name:
  <input type="text" [formControl]="name">
</label>
<p>
  Value: {{ name.value }}
</p>
```

Reactive forms provide access to information about a given control through properties and methods provided with each instance. These properties and methods

of the underlying [AbstractControl](#) class are used to control form state and determine when to display messages when handling validation.

Replacing a form control value

Methods are used to change the control values. Reactive forms have methods to change a control's value programmatically, which gives you the flexibility to update the value without user interaction. A form control instance provides a `setValue()` method that updates the value of the form control and validates the structure of the value provided against the control's structure. The following example adds a method to the component class to update the value of the control to *Angular* using the `setValue()` method.

(name-editor.component.ts)

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-name-editor',
  templateUrl: './name-editor.component.html',
  styleUrls: ['./name-editor.component.css']
})
export class NameEditorComponent implements OnInit {

  constructor() {}

  ngOnInit() {
  }
  updateName() {
    this.name.setValue('Angular');
  }
}
```

5.4.3 GROUPING FORMS CONTROL

Angular will provide the facility to group multiple controls. Form control can give the access over single input field, a form group will provide the group of control access. Each control in a form group instance is tracked by name when creating the form group.

So for that need to create one more component in our project.

ng generate component ProfileEditor

and import below line of code

```
import { FormGroup, FormControl } from '@angular/forms';
```

Step 1 : Creating an instance of Form Group

First need to Create a property in the component class named profileForm and set the property to a new form group instance. To initialize the form group, provide the constructor with an object of named keys mapped to their control.

For the profile form, add two form control instances with the names firstName and lastName.

(profileeditor.component.ts)

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'app-profile-editor',
  templateUrl: './profile-editor.component.html',
  styleUrls: ['./profile-editor.component.css']
})
export class ProfileEditorComponent {
  profileForm = new FormGroup({
    firstName: new FormControl(""),
```



```
lastName: new FormControl(""),
});
}
```

Step 2: Associating the FormGroup model and view

A form group tracks the status and changes for each of its controls, so if one of the controls changes, the parent control also emits a new status or value change. The model for the group is maintained from its members. After you define the model, you must update the template to reflect the model in the view.

(profile-editor.component.html)

```
<form [formGroup]="profileForm">

  <label>
    First Name:
    <input type="text" formControlName="firstName">
  </label>

  <label>
    Last Name:
    <input type="text" formControlName="lastName">
  </label>

</form>
```

Saving form data

The ProfileEditor component accepts input from the user, but in a real scenario you want to capture the form value and make available for further processing outside the component. The [FormGroup](#) directive listens for the submit event emitted by the form element and emits an ngSubmit event that you can bind to a callback function.

Add an ngSubmit event listener to the form tag with the onSubmit() callback method.

(profile-editor.component.html)

```
<form [formGroup]="profileForm" (ngSubmit)="onSubmit()">

  <label>
    First Name:
    <input type="text" formControlName="firstName">
  </label>
  <label>
    Last Name:
    <input type="text" formControlName="lastName">
  </label>
</form>
```

The `onSubmit()` method in the `ProfileEditor` component captures the current value of `profileForm`. Use [EventEmitter](#) to keep the form encapsulated and to provide the form value outside the component. The following example uses `console.warn` to log a message to the browser console.

```
export class ProfileEditorComponent {
  profileForm = new FormGroup({
    firstName: new FormControl(""),
    lastName: new FormControl(""),
  });

  onSubmit() {
    // TODO: Use EventEmitter with form value
    console.warn(this.profileForm.value);
  }
}
```

Displaying the component

To display the `ProfileEditor` component that contains the form, add it to a component template.

(app.component.html)

```
<app-profile-editor></app-profile-editor>
```

5.5 TEMPLATE DRIVEN FORMS

Forms are mainly used to attract the user. Forms enhance the look of the interface of app. Forms are the mainstay of business applications. You use forms to log in, submit a help request, place an order, book a flight, schedule a meeting, and perform countless other data-entry tasks.

Developing forms requires design skills (which are out of scope for this page), as well as framework support for *two-way data binding*, *change tracking*, *validation*, and *error handling*. Template driven forms are forms where we write logic, validations, controls etc, in the template part of the code (html code). The template is responsible for setting up the form, the validation, control, group etc. Template driven forms are suitable for simple scenarios, uses two way data binding using the [(NgModel)] syntax, easier to use though unit testing might be a challenge.

Template driven form is one complete form.

Code for creating a simple Template driven form. In this code we show how template-driven forms in Angular can be created. This code uses the FormsModule and NgModel directive to register form controls on an out NgForm.

(app.module.ts)

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { FormComponent } from './form.component';

@NgModule({
  imports: [BrowserModule, FormsModule],
  declarations: [AppComponent, FormComponent],
```

```
bootstrap: [AppComponent]
})
export class AppModule {}
```

(app.component.ts)

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  templateUrl: 'app/app.component.html'
})
export class AppComponent {}
```

Create a new component named form using below syntax inside app folder.

Ng generate component form

(app/form.component.ts)

```
import { Component } from '@angular/core';

@Component({
  selector: 'form-component',
  template: `
    <form #form="ngForm" (ngSubmit)="submit(form.value)">
      <div>
        <label>Firstname:</label>
        <input type="text" name="firstname" ngModel>
      </div>
      <div>
        <label>Lastname:</label>
        <input type="text" name="lastname" ngModel>
      </div>
      <div>
```

```

    <label>Street:</label>
    <input type="text" name="street" ngModel>
  </div>
  <div>
    <label>Zip:</label>
    <input type="text" name="zip" ngModel>
  </div>
  <div>
    <label>City:</label>
    <input type="text" name="city" ngModel>
  </div>

  <button type="submit">Submit</button>
</form>

<pre>
{{form.value | json}}
</pre>

<h4>Submitted</h4>
<pre>
{{value | json }}
</pre>
.
})
export class FormControl {

  value: any;

  submit(form) {
    this.value = form;
  }
}

```

(app.component.html)

```
<form-component></form-component>
```

Create one more ts file inside the app folder and set the below code for design using the bootstrap.

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
import { AppModule } from './app.module';  
  
platformBrowserDynamic().bootstrapModule(AppModule);
```

Save your entire project and output should be like.

Firstname:
Lastname:
Street:
Zip:
City:

```
{  
  "firstname": "",  
  "lastname": "",  
  "street": "",  
  "zip": "",  
  "city": ""  
}
```

Submitted

After submitting the data it look like

Firstname:
Lastname:
Street:
Zip:
City:

```
{  
  "firstname": "Core",  
  "lastname": "Angular",  
  "street": "Google",  
  "zip": "123456",  
  "city": "INC"  
}
```

Submitted

```
{  
  "firstname": "Core",  
  "lastname": "Angular",  
  "street": "Google",  
  "zip": "123456",  
  "city": "INC"  
}
```

5.6 FORM VALIDATION

User Interface play the vital role in any apps. To give the dynamic look at the app must have to forms in app. Forms are almost always present in any website or application. Forms can be used to perform countless data-entry tasks such as authentication, order submission or a profile creation.

Improve overall data quality by validating user input for accuracy and completeness. To design a form is easy but it must be well maintained. That means a form must be a validate form. Because forms are used to collection of information so information must be a in correct format so forms are need to be a validated.

5.6.1 TEMPLATE DRIVEN VALIDATION

To add validation to a template-driven form, you add the same validation attributes as you would with [native HTML form validation](#). Angular uses directives to match these attributes with validator functions in the framework.

Every time the value of a form control changes, Angular runs validation and generates either a list of validation errors, which results in an INVALID status, or null, which results in a VALID status.

You can then inspect the control's state by exporting `ngModel` to a local template variable. The following example exports `NgModel` into a variable called `name`:

Code of example.

```
<input id="name" name="name" class="form-control"
  required minlength="4" appForbiddenName="bob"
  [(ngModel)]="hero.name" #name="ngModel" >

<div *ngIf="name.invalid && (name.dirty || name.touched)"
  class="alert alert-danger">

  <div *ngIf="name.errors.required">
    Name is required.
  </div>

  <div *ngIf="name.errors.minlength">
    Name must be at least 4 characters long.
  </div>

  <div *ngIf="name.errors.forbiddenName">
    Name cannot be dob.
  </div>
</div>
```


Explanation of code

- `<input>` element carries the HTML validation attributes: `required` and `minlength`.
- `#name="ngModel"` exports `NgModel` into a local variable called `name`. `NgModel` mirrors many of the properties of its underlying `FormControl` instance.
- The `*ngIf` on the `<div>` element reveals a set of nested message divs but only if the name is invalid and the control is either dirty or touched.
- Each nested `<div>` can present a custom message for one of the possible validation errors. There are messages for `required`, `minlength`, and `forbiddenName`.

5.6.2 REACTIVE FORM VALIDATION

The most important part of application is form validation. To validate the Reactive form component class become the main class. Instead of adding validators through attributes in the template, you add validator functions directly to the form control model in the component class. Angular then calls these functions whenever the value of the control changes.

We can perform the validation on Reactive form in two ways, either using validators functions or using built-in validators.

Validator functions

There are two types of validator functions: sync validators and async validators.

- **Sync validators:** Is a functions that take a control instance and immediately return either a set of validation errors or null. You can pass these in as the second argument when you instantiate a `FormControl`.
- **Async validators:** Is a functions that take a control instance and return a Promise or Observable that later emits a set of validation errors or null. You can pass these in as the third argument when you instantiate a `FormControl`.

Built-in validators

Similar to the built-in validators that are available as attributes in template-driven forms, such as `required` and `minlength`, are all available to use as functions from the `Validators` class. For a full list of built-in validators

5.7 LET US SUM UP

- Forms are required for securely logging in a user to the app, searching for all the available hotels in a particular city, booking a cab, building a to-do list, and doing tons of other things that we are used to
- Interface, Forms are the most important aspect of any web application.
- Events get in like from clicking on links or moving the mouse it is through forms where we get the crucial data inputs from the end users.
- Angular, being a full-fledged front-end framework, has its own set of libraries for building complex forms
- Two types of form: Reactive and Template-driven
- Reactive forms - more scalable, reusable, and testable
- Template-driven forms - adding a simple form to an app, such as an email list signup form
- Both the technologies belong to the `@angular/forms` library and are based on the same form control classes
- Reactive forms are more explicit, created in component class as compare to template driven form.
- Unit Testing is complex in template driven forms.
- Reactive forms are more powerful. They are more scalable, reusable, and testable. If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.
- To use the reactive forms in angular need to import `ReactiveFormsModule` from the `angular/forms` package and add it to `NgModules` import array.(`app.module.ts`)
- Powerful Reactive forms give you access to the form control state and value at a point in time.

- Forms are mainly used to attract the user. Forms enhance the look of the interface of app. Forms are the mainstay of business applications
- Code for creating a simple Template driven form
- To give the dynamic look at the app must have to forms in app. Forms are almost always present in any website or application. Forms can be used to perform countless data-entry tasks such as authentication, order submission or a profile creation.
- To add validation to a template-driven form, you add the same validation attributes as you would with native HTML form validation.
- To validate the Reactive form component class become the main class. Instead of adding validators through attributes in the template, you add validator functions directly to the form control model in the component class. Angular then calls these functions whenever the value of the control changes.

5.8 CHECK YOUR PROGRESS

1. _____ are critical to any modern front-end application

A. Forms	C. CSS
B. Pages	D. Design

2. Angular supports the _____ types of forms.

A. 1	C. 4
B. 3	D. 2

4. _____ forms are more scalable, reusable, and testable

A. Template	C. Normal
B. Reactive	D. A & B

5. _____ forms adding a simple form to an app, such as an email list signup.

A. Template	C. Normal
B. Reactive	D. A & B

6. _____ tracks the value and validation status of an individual form control.

A. FormArray	C. FormValue
B. FormGroup	D. FormControl

7. _____ tracks the same values and status for an array of form controls.
- A. FormArray
 - B. FormGroup
 - C. FormValue
 - D. FormControl
8. To use the reactive forms in angular need to import _____
- A. ReactiveFormsModule
 - B. ReactiveForms
 - C. ReactiveModule
 - D. ReactiveCore
9. to create the new componet _____ syntax is used.
- A. ng generate
 - B. ng generate component
 - C. ng generate component
comp_name
 - D. generate component
10. There are two types of validator functions: sync validators and _____ validators.
- A. void
 - B. null
 - C. empty
 - D. asyn

5.9 CHECK YOUR PROGRESS:POSSIBLE ANSWER

- 1. Forms
- 2. 2
- 4. Reactive
- 5. Template
- 6. FormControl
- 7. FormArray
- 8. ReactiveFormsModule
- 9. ng generate component
comp_name
- 10. asyn

5.10 ASSIGNMENTS

Write the answer for the following questions.

1. What is forms?
2. List the types of form in angular with definition.
3. Differentiate reactive form and template driven form

4. How to create reactive form explain in detail.
5. How to create template driven form explain in detail.
6. What is form validation.
7. Explain how to validate reactive form with validation functions.

Block-3

Working With IONIC

Unit 1: Setting up the Environment for IONIC

1

Unit Structure

- 1.12 Learning Objective
- 1.13 Ionic Framework
- 1.14 How to build Mobile Apps
- 1.15 Node.js
- 1.16 Installing CLI
- 1.17 IDE's
- 1.18 iOS setup
- 1.19 Android setup
- 1.20 Let Us Sum Up
- 1.21 Check Your Progress
- 1.22 Check Your Progress: Possible Answers
- 1.23 Activities

1.1 LEARNING OBJECTIVE

After studying this chapter, students should be able to understand.

- How Ionic Framework works.
- All about mobile app thought process.
- Node.js installation
- CLI installation
- iOS and Android setup

1.2 IONIC FRAMEWORK

Ionic framework is a powerful tool to build hybrid mobile apps. It's open source (<https://github.com/ionic-team/ionic>) and has over 35,000 stars on GitHub, the popular social coding platform. Ionic framework is not the only player in hybrid mobile apps development, but it's the one that draws a lot of attention and is recommended as the first choice by many developers. Ionic is popular for the following reasons:

- Based on Web Components standards and is framework agnostic. Web Components are W3C specifications of components for the web platform. Ionic components are built as custom elements using its own open source tool, Stencil. Being framework agnostic makes Ionic components work with any framework. Developers are free to choose the framework to use, including Angular, React, and Vue.
- Provides beautifully designed out-of-box UI components that work across different platforms. Common components include lists, cards, modals, menus, and pop-ups. These components are designed to have a similar look and feel as native apps. With these built-in components, developers can quickly create prototypes with good enough user interfaces and continue to improve them.
- Leverages Apache Cordova as the runtime to communicate with native platforms. Ionic apps can use all the Cordova plugins to interact with the

native platform. Ionic Native further simplifies the use of Cordova plugins in Ionic apps.

- Performs great on mobile devices. The Ionic team devotes great effort to make it perform well on different platforms.

The current release version of Ionic framework is 4.0. Ionic 4 is the first version of Ionic to be framework agnostic. Ionic Core is the set of components based on Web Components. Ionic Angular is the framework binding of Ionic Core with Angular. This book focuses on Ionic Angular with Angular 6.

Apart from the open source Ionic framework, Ionic also provides a complete solution Ionic Pro for mobile app development, which includes the following products:

- **Ionic Creator** – Ionic Creator is a desktop app to create Ionic apps using drag-and-drop. It helps nontechnical users to quickly create simple apps and prototypes.
- **Ionic View** – Ionic View allows viewing Ionic apps shared by others directly on the phones. It's a great tool for app testing and demonstration.
- **Ionic Deploy** – Ionic Deploy performs hot updates to apps after they are published to app stores.
- **Ionic Package** – Ionic Package builds Ionic apps and generates bundles ready for publishing to app stores. With Ionic Package, we don't to manage local build systems and can use the cloud service instead.
- **Ionic Monitor** – Ionic Monitor can monitor apps and report runtime errors.

1.3 HOW TO BUILD MOBILE APPS

Even with the frameworks and services mentioned above, it's still not an easy task to build mobile apps. There are multiple stages in the whole development life cycle from ideas to published apps. A typical process may include the following major steps:

- **Ideas brainstorming.** This is when we identify what kind of mobile apps to build. It usually starts from vague ideas and expands to more concrete solutions.
- **Wire-framing and prototyping.** This is when we draw on the whiteboard to identify main usage scenarios. Prototypes may be created to demonstrate core usage scenarios for better communications with stakeholders.
- **User experiences design.** This is when all pages and navigation flows are finalized, and we are now clear what exactly needs to be built.
- **Implementation.** This is when the development team implements the pages to fulfill requirements.
- **Testing.** Unit testing should be part of implementation of pages and components. End-to-end testing is also required to verify all usage scenarios. All these tests should be executed automatically.
- **Continuous integration.** Continuous integration is essential for code quality. If every code commit can be tested automatically, then the development team will be more confident about the product's quality.
- **Publishing.** This is when the app is published to app stores.
- **Operations.** After the app is published, we still need to continuously monitor its running status. We need to capture errors and crash logs occurred on users' devices.

1.4 NODE.JS

Node.js is the runtime platform for Ionic CLI. To use Ionic CLI, we first need to install Node.js (<https://nodejs.org/>) on the local machine. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It provides a way to run JavaScript on the desktop machines and servers. Ionic CLI itself is written in JavaScript and executed using Node.js. There are two types of release versions of Node.js – the stable LTS versions and current versions with the latest features. It's recommended

to use Node.js version 6 or greater, especially the latest LTS version (8.12.0 at the time of writing).

Installing Node.js also installs the package management tool npm. npm is used to manage Node.js packages used in projects. Thousands of open source packages can be found in the npmjs registry (<https://www.npmjs.com/>). If you have background with other programming languages, you may find npm is similar to Apache Maven (<https://maven.apache.org/>) for Java libraries or Bundler (<http://bundler.io/>) for Ruby gems.

1.5 INSTALLING CLI

Ionic CLI

After Node.js is installed, we can use npm to install the Ionic command-line tool and Apache Cordova.

```
$ npm i -g cordova ionic
```

Note : You may need to have system administrator privileges to install these two packages. For Linux and macOS, you can use sudo. For Windows, you can start a command-line window as the administrator. However, it's recommended to avoid using sudo when possible, as it may cause permission errors when installing native packages. Treat this as the last resort. The permission errors usually can be resolved by updating the file permissions of the node.js installation directory.

After finishing installation of Ionic CLI and Cordova, we can use the command ionic to start developing Ionic apps.

You are free to use Windows, Linux, or macOS to develop Ionic 4 apps. Node.js is supported across different operating systems. One major limitation of Windows or Linux is that you cannot test iOS apps using the emulator or real devices. Some open source Node.js packages may not have the same test coverage on Windows as Linux or macOS. So they are more likely to have compatibility issues when running on Windows. But this should only affect the CLI or other tools, not Ionic 4 itself.

After Ionic CLI is installed, we can run `ionic info` to print out current runtime environment information and check for any warnings in the output; see Listing 1-1. The output also provides detailed information about how to fix those warnings.

Ionic:

ionic (Ionic CLI) : 4.12.0 (/usr/local/lib/node_modules/ionic)

Ionic Framework : @ionic/angular 4.3.1

@angular-devkit/build-angular : 0.13.8

@angular-devkit/schematics : 7.3.8

@angular/cli : 7.3.8

@ionic/angular-toolkit : 1.5.1

System:

NodeJS : v8.9.4 (/usr/local/bin/node)

npm : 5.6.0

OS : macOS Mojave

1.6 IDE'S

You are free to use your favorite IDEs and editors when developing Ionic apps. IDEs and editors should have good support for editing HTML, TypeScript, and Sass files. For open source alternatives, Visual Studio Code (<https://code.visualstudio.com/>) and Atom (<https://atom.io/>) are both popular choices tools. For commercial IDEs, WebStorm (<https://www.jetbrains.com/webstorm/>) is recommended for its excellent support of various programming languages.

1.6.1 LEARNING MORE ABOUT VISUAL STUDIO CODE

Setting up Visual Studio Code

Getting up and running with Visual Studio Code is quick and easy. It is a small download so you can install in a matter of minutes and give VS Code a try.

Cross platform

VS Code is a free code editor which runs on the macOS, Linux and Windows operating systems.

Follow the platform specific guides below:

- macOS
- Linux
- Windows

VS Code is lightweight and should run on most available hardware and platform versions. You can review the System Requirements to check if your computer configuration is supported.

Extensions

VS Code extensions let third parties add support for additional:

- Languages - C++, C#, Go, Java, Python
- Tools - ESLint, JSHint , PowerShell
- Debuggers - Chrome, PHP XDebug.
- Keymaps - Vim, Sublime Text, IntelliJ, Emacs, Atom, Visual Studio, Eclipse

Extensions integrate into VS Code's UI, commands, and task running systems so you'll find it easy to work with different technologies through VS Code's shared interface.

Some useful extension:

1. Ionic Extension Pack
2. Ionic 4 Snippets
3. Prettier - Code formatter
4. TSLint
5. Peacock
6. Auto Rename Tag

7. Angular Language Service
8. Angular v7 Snippets
9. angular2-switcher

Next steps

Once you have installed and set up VS Code, these topics will help you learn more about VS Code:

- Additional Components - Learn how to install Git, Node.js, TypeScript and tools like Yeoman.
- User Interface - A quick orientation to VS Code.
- Basic Editing - Learn about the powerful VS Code editor.
- Code Navigation - Move quickly through your source code.
- Debugging - Debug your source code directly in the VS Code editor.
- Proxy Server Support - Configure your proxy settings.

1.7 IOS SETUP

Developing iOS apps with Ionic requires macOS and Xcode (<https://developer.apple.com/xcode/>) You need to install Xcode and Xcode command-line tools on macOS. After installing Xcode, you can open a terminal window and type command shown below.

```
$ xcode-select -p
```

If you see output like that below, then command-line tools have already been installed.

```
/Applications/Xcode.app/Contents/Developer
```

Otherwise, you need to use the following command to install it.

```
$ xcode-select --install
```

After the installation is finished, you can use **xcode-select -p** to verify.

To run Ionic apps on the iOS simulator using Ionic CLI, package `ios-sim` is required. Another package `ios-deploy` is also required for deploying to install and debug apps. You can install both packages using the following command.

```
$ npm i -g ios-sim ios-deploy
```

1.8 ANDROID SETUP

To develop Ionic apps for Android, Android SDK must be installed. Before installing Android SDK, you should have JDK installed first. Read this guide (<https://docs.oracle.com/javase/8/docs/technotes/guides/install/>) about how to install JDK 8 on different platforms. It's recommended to install Android Studio (<https://developer.android.com/studio/index.html>), which provides a nice IDE and bundled Android SDK tools.

Note: Android API level 22 is required to run ionic apps. Make sure that the required Sdk platform is installed.

Stand-alone SDK tools is just a ZIP file; unpack this file into a directory and it's ready to use. The downloaded SDK only contains basic SDK tools without any Android platform or third-party libraries. You need to install the platform tools and at least one version of the Android platform. Run `android` in tools directory to start Android SDK Manager to install platform tools and other required libraries.

After installing Android SDK, you need to add SDK's tools and platform-tools directories into your PATH environment variable, so that SDK's commands can be found by Ionic. Suppose that the SDK tools is unpacked into `/Development/android-sdk`, then add `/Development/android-sdk/tools` and `/Development/android-sdk/platform-tools` to PATH environment variable. For Android Studio, the Android SDK is installed into directory `/Users/<username>/Library/Android/sdk`.

To modify PATH environment variable on Linux and macOS, you can edit `~/.bash_profile` file to update PATH as shown below. The PATH environment below

uses the Android SDK from Android Studio. You can replace it with another directory if stand-alone SDK tools is used.

```
export PATH=${PATH}:/Users/<username>/Library/Android/sdk/platform-tools \
:/Users/<username>/Library/Android/sdk/tools
```

To modify the PATH environment variable on Windows, you can follow the steps below.

- 1 Click **Start** menu, then right-click **Computer** and select **Properties**.
- 2 Click **Advanced System Settings** to open a dialog.
- 3 Click **Environment Variables** in the dialog and find **PATH** variable in the list, then click **Edit**.
- 4 Append the path of tools and platform-tools directories to the end of **PATH** variable.

It's highly recommended to use Android Studio instead of stand-alone SDK tools. Stand-alone SDK tools is more likely to have configuration issues.

1.9 LET US SUM UP

- Use and application of Ionic framework
- All brief thought for build mobile apps
- Installing all the pre-requisite
- Visual studio code and its extension
- Setting up iOS and Android environment.

1.10 CHECK YOUR PROGRESS

1. Ionic is used to build _____ mobile apps
 - A. Native
 - B. Hybrid
 - C. Web app
 - D. Standard

2. Ionic used _____ to interact with native platform.
- A. Angular
 - B. Swift
 - C. Java
 - D. Apache Cordova
3. Latest version of Ionic is _____.
- A. 2.0
 - B. 2.5
 - C. 3.0
 - D. 4.0
4. _____ is the runtime platform for Ionic CLI.
- A. React.js
 - B. Node.js
 - C. Vue.js
 - D. Angular.js
5. _____ command is used to get the complete information of Ionic
- A. ionic info
 - B. ionic getdata
 - C. ionic detail
 - D. ionic i
6. Full form of IDE.
- A. Integrated Demand Environment
 - B. Internal Development Engine
 - C. Integrated Development Environment
 - D. Intel Development Environment

1.11 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

Answer

- | | | |
|------|------|------|
| 1. B | 2. D | 3. D |
| 4. B | 5. A | 6. C |

1.12 ACTIVITIES

- Install NodeJS
 - Try to get information about NPM
- Install Ionic and Cordova
- Install IDE(Visual Studio Code)
- Install all the required extension for IDE, So that development can be more productive.
- Setup Android
- Setup iOS (MacOS system are mandatory)

Unit 2: Developing First Mobile Application

2

Unit Structure

- 2.1 Learning Objective
- 2.2 Introduction
- 2.3 Create your first hybrid app
- 2.4 Scaffolding Ionic
- 2.5 Working with Simulator
- 2.6 Running the Mobile Application on Android Phone
- 2.7 Running the Mobile Application on Apple Phone
- 2.8 Android Debugging
- 2.9 iOS Debugging
- 2.10 WebView
- 2.11 Let Us Sum Up
- 2.12 Check your Progress
- 2.13 Check your Progress: Possible Answers
- 2.14 Assignments

2.1 LEARNING OBJECTIVE

After studying this chapter, students should be able to understand.

- First app in Ionic
- Folder structure of Ionic
- Working with simulator
- Debugging

2.2 INTRODUCTION

After the local development environment is set up successfully, it's time to create new Ionic apps. The easiest way to create Ionic apps is using Ionic CLI. Before we can create Ionic 4 apps using Ionic CLI, we need to enable the feature project-angular first by running the following command.

```
$ ionic config set -g features.project-angular true
```

Apps are created using the command `ionic start`. Below is the syntax of using `ionic start`.

```
$ ionic start <name> <template> [options]
```

The first argument of `ionic start` is the name of the new app, while the second argument is the template name. We can also pass extra options after these two arguments. If not enough arguments are provided, Ionic CLI can help you to finish the setup interactively with prompts. Ionic CLI supports creation of projects of three types. Project types are specified using the option `--type`, for example, `--type=angular`.

- `angular` – Ionic Angular projects for Ionic 4.
- `ionic-angular` – Ionic 2/3 projects.
- `ionic1` – Ionic 1 projects.

For project templates, Ionic provides different types of application templates. All available templates can be listed with the following command. A template may have versions for different project types.

\$ ionic start --list

We can choose a proper template to create the skeleton code of the app. It's also possible to pass URLs of other Git repositories as the templates to use. Ionic also maintains a marketplace (<https://market.ionicframework.com/starters/>) for the community to share project starters. You can find many paid or free project starters in the marketplace. Below are the available options of ionic start.

- --type – Allowed values are angular, ionic-angular and ionic1.
- --cordova – Enable Cordova integration.
- --capacitor - Enable Capacitor integration.
- --pro-id – Link this app with Ionic Dashboard.
- --no-deps – Do not install npm dependencies. Useful when you only want to explore the content of a project starter.
- --no-git – Do not initialize a Git repo.
- --no-link – Skip the prompt about connecting the app with Ionic Dashboard.
- --project-id – Specify the slug for the app. The slug is used for the directory name and npm package name.
- --package-id – Specify the bundle ID/application ID for the app. This is the unique ID of the app when publishing to the Apple store or Google Play. It's highly recommended to set this value when Cordova integration is enabled. The value of this option should be in the reverse domain format, for example, com. mycompany.myapp. If not specified, the default value io.ionic.starter is used.

2.3 CREATE YOUR FIRST HYBRID APP

Blank App

This template blank only generates basic code for the app. This template should be used when you want to start from a clean code base

```
$ ionic start blankApp blank
```

```
$ cd blankApp
```

```
$ ionic serve
```

This will open the browser with the preview of your app which will **reload automatically** once you change anything inside your project.

Starter	Description
Tabs	A starting project with a simple tabbed interface
Blank	A blank starter project
Sidemenu	A starting project with a side menu with navigation in the content area
Super	A starting project complete with pre-built pages, providers and best practices for Ionic development.
Conference	A project that demonstrates a realworld application
Tutorial	A tutorial based project that goes along with the Ionic documentation
Aws	AWS Mobile Hub Starter

Table-6 Project Starter

Starters are constructed within the Ionic Starters repository by overlaying a starter app onto a set of base files, constructing a compressed archive of the files, and uploading it around the world. The Ionic CLI then downloads and extracts the starter template archive and personalizes files for each new app.

Local Development

After a new app is created using ionic start, we can navigate to the app directory and run ionic serve to start the local development server. The browser should

automatically open a new window or tab that points to the address `http://localhost:8100/`. You should see the UI of this Ionic app. Ionic sets up livereload by default, so when any HTML, TypeScript or Sass code is changed, it automatically refreshes the page to load the page with updated code. There is no need for a manual refresh.

The default port for the Ionic local development server is 8100. The port can be configured using the option `--port` or `-p`. For example, we can use `ionic serve -p 9090` to start the server on port 9090.

2.4 SCAFFOLDING IONIC

Let's walk through the anatomy of an Ionic app. Inside of the folder that was created, we have a typical Cordova project structure where we can install native plugins, and create platform-specific project files.



Figure-17: Scaffolding Ionic

➤ Config Files

File name	Description
package.json	<p>package.json is a JSON file that describes this Node.js project. This file contains various metadata of this project, including name, description, version, license, and other information. This file is also used by npm or yarn to manage a project's dependencies.</p> <p>package.json also contains metadata used by Cordova. All Cordova-related configurations are specified in the property cordova. The value of this property is a JavaScript object with other configuration values. For example, plugins specifies the installed plugins for this app, while platforms specifies supported platforms.</p> <p>The content of package.json file can be managed by tools like npm, yarn, or Ionic CLI, or edited manually using text editors.</p>
config.xml	<p>config.xml is the configuration file for Apache Cordova. More information about this file can be found at the Apache Cordova website (https://cordova.apache.org/docs/en/dev/config_ref/index.html - The config.xml File). The content of this file is usually managed by Cordova CLI.</p>
tsconfig.json	<p>tsconfig.json is the JSON file to configure how TypeScript compiler compiles the TypeScript code into JavaScript. For example, compilerOptions specifies the compiler options. In these compiler options, emitDecoratorMetadata and experimentalDecorators must be enabled to use Angular decorators. More details about tsconfig.json can be found in the official website (http://www.typescriptlang.org/docs/</p>

	<p>handbook/tsconfig-json.html).</p> <pre> { "compileOnSave": false, "compilerOptions": { "baseUrl": "./", "outDir": "./dist/out-tsc", "sourceMap": true, "declaration": false, "module": "es2015", "moduleResolution": "node", "emitDecoratorMetadata": true, "experimentalDecorators": true, "importHelpers": true, "target": "es5", "typeRoots": ["node_modules/@types"], "lib": ["es2018", "dom"] } } </pre>
ionic.config.json	<p>ionic.config.json is the config file for Ionic itself. From this file, we can see the app is an Ionic 4 project with Cordova integration. Because the project is created from an Ionic starter template, it</p>

	<p>also has the file <code>ionic.starter.json</code> to describe the template used to create it. The file <code>ionic.starter.json</code> can be removed after the code skeleton is generated.</p> <pre>{ "name": "My App", "type": "angular", "id": "abc123", "integrations": { "cordova": {} } }</pre>
<code>tslint.json</code>	<code>tslint.json</code> is the config file for TSLint. Ionic 4 provides a default configuration. More rules (https://palantir.github.io/tslint/rules/) can be added to match the development team's style guide.
<code>angular.json</code>	<code>angular.json</code> is the configuration file of Angular CLI.

➤ Cordova Files

Besides the `config.xml` in the root directory, the directories `hooks`, `platforms`, `plugins`, and `www` are all managed by Cordova.

File name	Description
<code>platforms</code>	For each supported platform, there is a subdirectory in the directory <code>platforms</code> to contain built files for this platform. This app has subdirectories <code>ios</code> and <code>android</code> for iOS and Android platforms, respectively. These files are generated by Cordova and should not be edited manually.
<code>plugins</code>	This directory contains various Cordova plugins used in this app.
<code>www</code>	The directory <code>www</code> contains static files of this app. This is also the directory of the whole app's static files, including built JavaScript and CSS files and other assets.

➤ App Files

All the app's main source code is in the directory `src`. The directory `resources` contains the image files for the app icons and the splash screen.

File name	Description
<code>index.html</code>	As we know, an Ionic app is just a web app running inside the browser-like <code>WebView</code> component. The <code>index.html</code> is the entry point of the whole Ionic app. The markup in <code>index.html</code> is very simple with only one element <code>app-root</code> .
<code>Assets</code>	The directory <code>assets</code> contains various assets used in the app, including <code>favicon</code> and <code>fonts</code> .
<code>Theme</code>	The directory <code>theme</code> contains <code>Sass</code> files to customize the look and feel of the app. The file <code>variables.scss</code> contains colors for different themes. If you don't need to customize the look and feel of Ionic 4, just leave this file unchanged.

➤ Environment Files

The directory `environments` contains configuration files for different environments. `Development` and `production` environments are defined by default.

File name	Description
<code>environment.ts</code>	The difference between these two environments is the value of the property <code>production</code> . We can add extra environment-specific configurations into these two files. To use the production environment, we can add the option <code>--env</code> , for example, <code>ng build --env=prod</code> .
<code>environment.prod.ts</code>	

➤ Skeleton Code

The template `blank` already includes some basic code. It has only one page.

File Path	Description
<code>App</code>	The directory <code>app</code> contains modules and components of

	the app.
Components	The directory components contains components declared in the app.
pages	The directory pages contains code for different pages. Each page has its own subdirectory.
app.module.ts	<p>The file app.module.ts declares the root module of the app. This file contains only a single empty class AppModule. Most of the code is using @NgModule decorator to annotate the class AppModule. The imported module IonicModule.forRoot() in the array of property imports is the difference between normal Angular modules and Ionic modules. The method IonicModule.forRoot() makes sure that the service providers, components, and directives from Ionic Angular are provided when the module is loaded. These components and directives can be used anywhere in the module. AppComponent is the root component of the app, so it's in the array of the property bootstrap to bootstrap the Ionic app. The property declarations contains only the AppComponent, while entryComponents is an empty array. StatusBar and SplashScreen in the property providers are used by Cordova. RouteReuseStrategy is related to Angular Router</p> <pre> import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/platform-browser'; import { RouteReuseStrategy } from '@angular/router'; import { IonicModule, IonicRouteStrategy } from '@ionic/angular'; </pre>

	<pre> import { SplashScreen } from '@ionic-native/splash-screen/ngx'; import { StatusBar } from '@ionic-native/status-bar/ngx'; import { AppRoutingModuleModule } from './app-routing.module'; import { AppComponent } from './app.component'; @NgModule({ declarations: [AppComponent], entryComponents: [], imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModuleModule], providers: [StatusBar, SplashScreen, { provide: RouteReuseStrategy, useClass: IonicRouteStrategy }], bootstrap: [AppComponent] }) export class AppModule {} </pre>
app-routing.module.ts	<p>The file app-routing.module.ts defines the routes used by Angular Router in the app. Here the path home points to the HomePageModule. We'll see more details of routing.</p> <pre> import { NgModule } from '@angular/core'; import { Routes, RouterModule } from '@angular/router'; const routes: Routes = [</pre>

	<pre> { path: "", redirectTo: 'home', pathMatch: 'full' }, { path: 'home', loadChildren: './home/home.module#HomePageModule' },]; @NgModule({ imports: [RouterModule.forRoot(routes)], exports: [RouterModule] }) export class AppRoutingModule { } </pre>
app.component.ts	<p>The file app.component.ts declares the main component of the app. In the decorator @Component, the property templateUrl is set to the file app.component.html. The constructor function declares a public parameter platform of type Platform. Angular injector creates the instance of Platform and provides it when this component is instantiated. The method ready() of Platform returns a promise that is resolved when the Cordova platform is ready to use. Here it set the styles of the status bar and hide the splash screen.</p> <pre> import { Component } from '@angular/core'; import { Platform } from '@ionic/angular'; import { SplashScreen } from '@ionic-native/splash-screen/ngx'; import { StatusBar } from '@ionic-native/status-bar/ngx'; @Component({ selector: 'app-root', templateUrl: 'app.component.html' </pre>

	<pre> })) export class AppComponent { constructor(private platform: Platform, private splashScreen: SplashScreen, private statusBar: StatusBar){ this.initializeApp(); } initializeApp() { this.platform.ready().then(() => { this.statusBar.styleDefault(); this.splashScreen.hide(); }); } } </pre>
app.component.html	<p>The template file app.component.html uses ion-app as the root element of the app. The component ion-router-outlet is the Ionic Angular integration of Angular Router.</p> <pre> <ion-app> <ion-router-outlet></ion-router-outlet> </ion-app> </pre>
main.ts	<p>The main.ts file contains the logic to bootstrap the Ionic app. In the production environment, the method enableProdMode is invoked to enable Angular's production mode for better performance.</p> <pre> import { enableProdMode } from '@angular/core'; </pre>

	<pre>import { platformBrowserDynamic } from '@angular/platform-browser-dynamic'; import { AppModule } from './app/app.module'; import { environment } from './environments/environment'; if (environment.production) { enableProdMode(); } platformBrowserDynamic().bootstrapModule(AppModule) .catch(err => console.log(err));</pre>
global.scss	<p>The file global.scss contains additional global CSS styles. This file can also be used as the entry point to import other Sass files. If you don't need to customize the styles of the app, just leave this file empty.</p>
Home Page Files	<p>The home page has its own subdirectory home under the directory app. Each component has five files with similar names.</p> <p>home.page.ts – TypeScript file for the component class.</p> <p>home.module.ts – TypeScript file for the module.</p> <p>home.page.spec.ts – TypeScript file for the Jasmine test spec.</p> <p>home.page.html – HTML file as the view template.</p> <p>home.page.scss – Scss file for styles.</p>
package.json	<p>This file contains all dependencies (NPM packages) of our application. You can add new packages or update the version of packages already included. By executing the command <code>npm install</code> in the project directory the</p>

	dependencies listed in package.json are downloaded and added to the project automatically.
--	--

2.5 WORKING WITH SIMULATOR/EMULATORS

After finishing the basic testing using browsers, it's time to test on device emulators. First, we need to configure the platforms' support for the app. Ionic apps created by Ionic CLI have no platforms added by default.

2.6 RUNNING THE MOBILE APPLICATION ON ANDROID PHONE

To add the Android platform, we can use the following command.

```
$ ionic cordova platform add android --save
```

Then we need to finish several tasks before building the app for Android.

- Install Gradle. Gradle is the build tool for Android apps. Follow the official instructions (<https://gradle.org/install/>) to install Gradle on your local machine.
- Accept Android SDK licenses. Use the sdkmanager tool in Android SDK to accept all SDK package licenses by running `sdkmanager --licenses`. The tool `sdkmanager` can be found in the directory of `<Android_Home>/sdk/ tools/bin`.
- Create an Android Virtual Device (AVD). Follow the official instructions (<https://developer.android.com/studio/run/managing-avds>) to create a new AVD.

Now the app can be built for the Android platform using the following command.

```
$ ionic cordova build android
```

We can start the emulator and test the app; Ionic app running on the Android 8.1 emulator. If the emulator is not started, the following command will try to start it.

```
$ ionic cordova emulate android
```

When running on the emulator, we can also use the option `--livereload` to enable livereload, so the app refreshes automatically when the code changes.

2.7 RUNNING THE MOBILE APPLICATION ON APPLE PHONE

We can use the following command to add iOS platform support.

```
$ ionic cordova platform add ios --save
```

Then the app can be built for iOS platform using the following command. If you just installed Xcode, you may need to open Xcode to install additional components first.

```
$ ionic cordova build ios
```

Now you can start the emulator and test your app.

```
$ ionic cordova emulate ios
```

Running the code above will launch the default iOS emulator. If you want to use a different emulator, you can use `--target` flag to specify the emulator name. To get a list of all the targets available in your local environment, use the following command.

```
$ cordova emulate ios --list
```

Then you can copy the target name from the output and use it in the command `ionic cordova emulate ios`, see the code below to use the iPhone 8 with the iOS 11.3 emulator.

```
$ ionic cordova emulate ios --target=" iPhone-8, 11.3"
```

2.8 ANDROID DEBUGGING

Use Chrome for Development

Using iOS or Android emulators to test and debug Ionic apps is not quite convenient because emulators usually consume a lot of system resources and take a long time to start or reload apps. A better alternative is to use Chrome browser for basic testing and debugging. To open Chrome DevTools, you can open the Chrome system menu and select More Tools ► Developer Tools. Once the developer tools

window is opened, you need to click the mobile phone icon on the top menu bar to enable device mode. Then you can select different devices as rendering targets: for example, Apple iPhone X or Nexus 6P.

Use Chrome DevTools for Android Debugging

For Android platform, when an Ionic app is running on the emulator or a real device, we can use Chrome DevTools (<https://developers.google.com/web/tools/chrome-devtools/>) to debug the running app. Navigate to `chrome://inspect/#devices` in Chrome and you should see a list of running apps. Clicking inspect launches the DevTools to inspect the running app. If you cannot see the app in the list, make sure that the device is listed in the result of the command `adb devices`.

2.9 IOS DEBUGGING

Use Safari Web Inspector for iOS Debugging

For an iOS platform, when an Ionic app is running on the emulator or a real device, we can use Safari Web Inspector (<https://developer.apple.com/safari/tools/>) to debug the running app. After opening Safari, in the Develop menu, you should see a menu item like Simulator – iPhone X - iOS 11.3 (15E217). This menu item has a subitem called localhost - index. html. Clicking this menu item opens the Web Inspector for debugging.

2.10 WEB VIEW

The Web View powers web apps in native devices. Ionic maintains a Web View plugin for apps integrated with Cordova. The plugin is provided by default when using the Ionic CLI. For apps integrated with Capacitor, the Web View is automatically provided.

Ionic apps are built using web technologies and are rendered using Web Views, which are a full screen and full-powered web browser.

Modern Web Views offer many built-in HTML5 APIs for hardware functionality such as cameras, sensors, GPS, speakers, and Bluetooth, but sometimes it may also be

necessary to access platform-specific hardware APIs. In Ionic apps, hardware APIs can be accessed through a bridge layer, typically by using native plugins which expose JavaScript APIs.

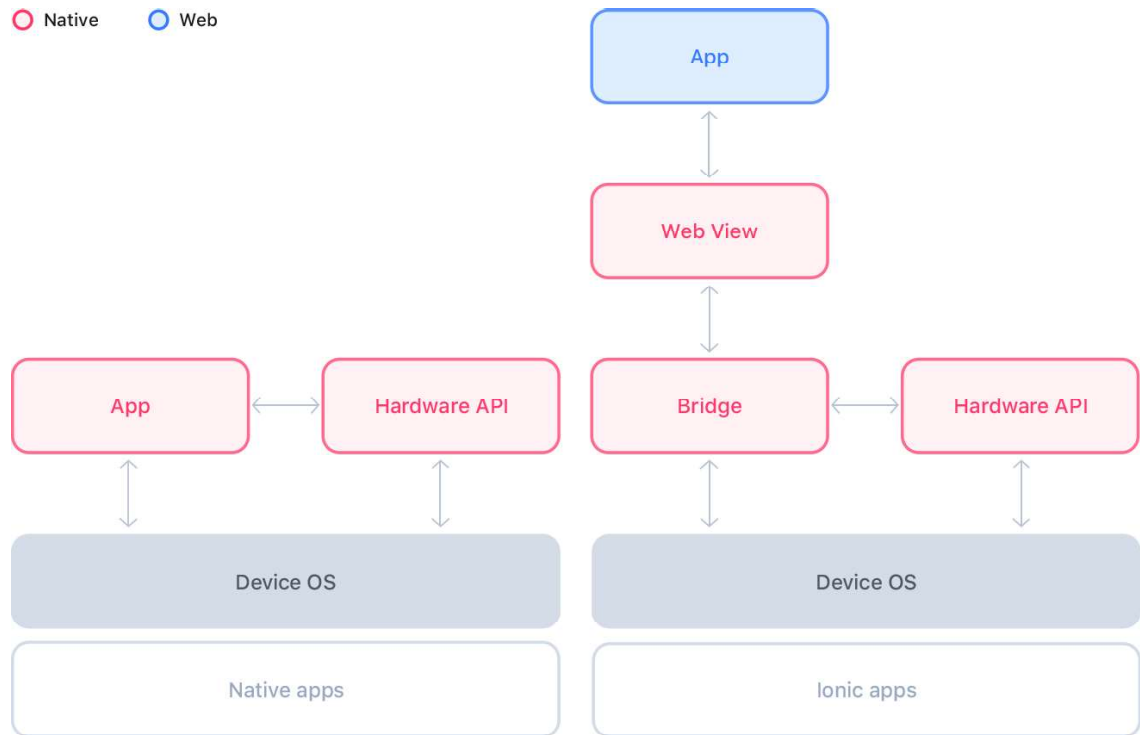


Figure-18 Structure of Ionic App

The Ionic Web View plugin is specialized for modern JavaScript apps. For both iOS and Android, app files are always hosted using the http:// protocol with an optimized HTTP server that runs on the local device.

2.11 LET US SUM UP

- Understanding the predefined starters of Ionic framework
- Learning structure of Ionic project.
- Emulators
- Create, Build and Run the individual platform
 - iOS
 - Android

- Debugging the platform
 - iOS
 - Android
- Understanding webview

2.12 CHECK YOUR PROGRESS

1. Command for ionic start <name> _____, to create app with tabs.

A. blank	B. tabs
C. super	D. No possible
2. Is it possible to configure the port on which ionic app will be serve.

A. True	B. False
---------	----------
3. _____ file contains metadata of ionic apps.

A. config.xml	B. package.json
C. angular.json	D. ionic.config.json
4. _____ file contains Name of the ionic apps.

A. config.xml	B. package.json
C. angular.json	D. ionic.config.json
5. Platform folder is managed by _____

A. Cordova	B. Angular
C. Ionic	D. Javascript
6. The _____ is the entry point of the whole Ionic app.

A. index.html	B. index.js
C. package.json	D. angular.json
7. The file _____ declares the root module of the app

A. index.html	B. app.module.ts
C. app.component.ts	D. app.html

2.14 ASSIGNMENTS

- Create & Run a blank project with name “Automobiles”
- Create & Run a tab project with name “Supermarket”
- Create & Run a sidemenu project with name “MasterDetails”
- Checkout all the config files.
- Go through the files and folder of project
- Try to debug the platform
 - iOS
 - Android

Unit 3: Typescript

3

Unit Structure

- 3.1 Learning Objectives
- 3.2 Introduction
- 3.3 Why Typescript ?
- 3.4 Basic Types
- 3.5 Functions
- 3.6 Interface and Classes
- 3.7 Let us sum up
- 3.8 Assignments

3.1 LEARNING OBJECTIVE

After studying this chapter, students should be able to understand.

- Typescript
- Variables
- Function, Interface and Classes
- Decorators
- Angular

3.2 INTRODUCTION

Building hybrid mobile apps with Ionic requires mostly front-end skills, including HTML, JavaScript, and CSS. You should have basic knowledge of these programming languages before reading this book.

Ionic Angular is the framework binding for the Angular framework, and it's also the framework. Other than standard JavaScript, Ionic Angular uses TypeScript (<https://www.typescriptlang.org/>) by default. This is because Angular uses TypeScript by default. You are still free to use JavaScript if you don't want to learn a new programming language. But TypeScript is strongly recommended for enterprise applications development. As the name suggests, TypeScript adds type information to JavaScript. Developers with knowledge of other static-typing programming languages, for example, Java or C#, may find TypeScript very easy to understand. The official TypeScript documentation (<https://www.typescriptlang.org/docs/index.html>) is a good starting point to learn TypeScript.

3.3 WHY TYPESCRIPT?

The reason why TypeScript is recommended for Ionic apps development is because TypeScript offers several benefits compared to standard JavaScript.

Compile-Time Type Checks

TypeScript code needs to be compiled into JavaScript code before it can run inside of the browsers because browsers don't understand TypeScript. This process is called transpiling. During the compiling, the compiler does type checks using type declarations written in the source code. These static type checks can eliminate potential errors in the early stage of development. JavaScript has no static-typing information in the source code. A variable declared with `var` can reference to any type of data. Even though this provides maximum flexibility when dealing with variables in JavaScript, it tends to cause more latent issues due to incompatible types in the runtime. For most of the variables and function arguments, their types are static and won't change in the runtime. For example, it's most likely an error when assigning a string to a variable that should only contain a number. This kind of error can be reported by the TypeScript compiler in the compile time.

In the below example, the variable `port` represents the port that a server listens on. Even though this variable should only contain a number, it's still valid to assign a string `9090` to `port` in JavaScript. This error may only be detected in the runtime.

```
var port = 8080;
port = '9090';
// -> valid assignment
```

However, the TypeScript code in below example declares the type of `port` is number. The following assignment causes a compiler error. So developers can find out this error immediately and fix it right away.

```
let port: number = 8080;
port = '9090';
// -> compiler error!
```

Rich Feature Sets

Apart from the essential compile-time type checks, TypeScript is also a powerful programming language with rich feature sets. Most of these features come from current or future versions of ECMAScript, including ES6, ES7, and ES8. Using these features can dramatically increase the productivity of front-end developers. You'll see the usages of these features in the code of the sample app.

Better IDE Support

With type information in the TypeScript source code, modern IDEs can provide smart code complete suggestions to increase developers' productivity. IDEs can also do refactoring for TypeScript code. Navigation between different files, classes, or functions is easy and intuitive. Front-end developers can enjoy the same coding experiences as Java and C# developers.

3.4 BASIC TYPES

The key point of writing TypeScript code is to declare types for variables, properties, and functions. TypeScript has a predefined set of basic types. Some of those types come from JavaScript, while other types are unique in TypeScript.

3.4.1 BOOLEAN

Boolean type represents a simple true or false value. A Boolean value is declared using type `boolean` in TypeScript.

```
let isActive: boolean = false;
```

```
isActive = true;
```

3.4.2 NUMBER

Numbers are all floating-point values in TypeScript. A number is declared using type `number` in TypeScript. TypeScript supports decimal, hexadecimal, binary, and octal literals for numbers. All these four numbers in the code below have the same decimal value 20.

```
let p1: number = 20;    // decimal
```

```
let p2: number = 0x14; // hexadecimal
```

```
let p3: number = 0b10100; // binary
```

```
let p4: number = 0o24; // octal
```

3.4.3 STRING

String type represents a textual value. A string is declared using type string in TypeScript. Strings are surrounded by double quotes (") or single quotes ('). It's up to the development team to choose whether to use double quotes or single quotes. The key point is to remain consistent across the whole code base. Single quotes are more popular because they are easier to type than double quotes that require the shift key.

```
let text: string = 'Hello World';
```

TypeScript also supports ES6 template literals, which allow embedded expressions in string literals. Template literals are surrounded by backticks (`). Expressions in the template literals are specified in the form of `${expression}`.

```
let a: number = 1;
```

```
let b: number = 2;
```

```
let result: string = `${a} + ${b} = ${a + b}`;
```

```
// -> string "1 + 2 = 3"
```

3.4.4 NULL AND UNDEFINED

null and undefined are special values in JavaScript. In TypeScript, null and undefined also have a type with name null and undefined, respectively. These two types only have a single value.

```
let v1: null = null;
```

```
let v2: undefined = undefined;
```

By default, it's allowed to assign null and undefined to variables declared with other types. For example, the code below assigns null to the variable v with type string.

```
let v: string = null;
```

However, null values generally cause errors in the runtime and should be avoided when possible. TypeScript compiler supports the option `--strictNullChecks`. When this option is enabled, TypeScript compiler does a strict check on null and undefined values. null and undefined can only be assigned to themselves and variables with type any. The code above will have a compile error when `strictNullChecks` is enabled.

3.4.5 ARRAY

Array type represents a sequence of values. The type of an array depends on the type of its elements. Appending `[]` to the element type creates the array type. In the code below, `number[]` is the type of arrays with numbers, while `string[]` is the type of arrays with strings. Array type can also be used for custom classes or interfaces. For example, `Point[]` represents an array of Point objects.

```
let numbers: number[] = [1, 2, 3];
```

```
let strings: string[] = ['a', 'b', 'c'];
```

3.4.6 TUPLE

The elements of an array are generally of the same type, that is, a homogeneous array. If an array contains a fixed number of elements of different types, that is, a heterogeneous array, it's called a tuple. The tuple type is declared as an array of element types. In the code below, the tuple `points` has three elements with types `number`, `number`, and `string`.

```
let points: [number, number, string] = [10, 10, 'P1'];
```

Tuples are useful when returning multiple values from a function because a function can only have at most one return value. Tuples of two elements, a.k.a. pairs, are commonly used. Be careful when using tuples with more than two elements, because elements of tuples can only be accessed using array indices, so it reduces the code readability. In this case, tuples should be replaced with objects with named properties. So it's better to change the type of `points` to an actual interface.

3.4.7 ENUM

Enum type represents a fixed set of values. Each value in the set has a meaningful name and a numeric value associated with the name. In the code below, the value of status is a number with value 1. By default, the numeric values of enum members start from 0 and increase in sequence. In the enum Status, Status.Started has value 0, Status.Stopped has value 1, and so on.

```
enum Status { Running, Stopped, NotWorking };  
  
let status: Status = Status.Stopped;
```

It's also possible to assign specific numeric values to enum values. In the code below, enum values Read, Write, and Execute have their assigned values. The value of permission is 3.

```
enum Permission { Read = 1, Write = 2, Execute = 4 };  
  
let permission = Permission.Read | Permission.Write;
```

To convert an enum value back to its textual format, we can do the lookup by treating the enum type as an array.

```
let status: string = Status[1];  
  
// -> 'Stopped'
```

3.4.8 ANY

Any type is the escape bridge from the TypeScript world to the JavaScript world. When a value is declared with type any, no type checking is done for this value. While type information is valuable, there are some cases when type information is not available, so we need the any type to bypass the compile-time check. Below are two common cases of using the type any.

- Migrate a JavaScript code base to TypeScript. During the migration, we can annotate unfinished values as any to make the TypeScript code compile.

- Integrate with third-party JavaScript libraries. If TypeScript code uses a third-party JavaScript library, we can declare values from this library as any to bypass type checks for this library. However, it's better to add type definitions for this kind of libraries, either by loading type definitions from community-driven repositories or creating your own type definitions files.

In the code below, the variable `val` is declared as `any` type. We can assign a string, a number, and a Boolean value to it.

```
let val: any = 'Hello World';  
  
val = 100; // valid  
  
val = true; // valid
```

3.4.9 VOID

Void means no type. It's commonly used as the return type of functions that don't return a value. The return type of the `sayHello` function below is `void`.

```
function sayHello(): void {  
    console.log('Hello');  
}
```

`void` can also be used as a variable type. In this case, the only allowed values for this variable are `undefined` and `null`.

3.4.10 UNION

Union type represents a value that can be one of several types. The allowed types are separated with a vertical bar (`|`). In the code below, the type of the variable `stringOrNumber` can be either `string` or `number`.

```
let stringOrNumber: string | number = 'Hello World';  
  
stringOrNumber = 44;  
  
stringOrNumber = 'Test';
```

Union types can also be used to create enum-like string literals. In the code below, the type `TrafficSignalColor` only allows three values.

```
type TrafficSignalColor = 'Red' | 'Green' | 'Yellow';  
let color: TrafficSignalColor = 'Red';
```

3.5 FUNCTIONS

Functions are important building blocks of JavaScript applications. TypeScript adds type information to functions. The type of a function is defined by the types of its arguments and return values.

As shown in below example, we only need to declare function types either on the variable declaration side or on the function declaration side. TypeScript compiler can infer the types from context information.

```
let size: (str: string) => number = function(str) {  
    return str.length;  
};  
  
let multiply = function(v1: number, v2: number): number {  
    return v1 * v2;  
}
```

Function types are useful when declaring high-order functions, that is, functions that take other functions as arguments or return other functions as results. When specifying types of functions used as arguments or return values, only type information is required, for example, **(string) => number** or **(number, number) => number**. We don't need to provide the formal parameters. In below example, `forEach` is a high-order function that takes functions of type **(any) => void** as the second argument.

```
function forEach(array: any[], iterator: (any) => void) {  
    for (let item in array) {
```



```
    iterator(item);  
  }  
}  
  
forEach([1, 2, 3], item => console.log(item));  
  
// -> Output 1, 2, 3
```

Arguments

JavaScript uses a very flexible strategy to handle function arguments.

A function can declare any number of formal parameters. When the function is invoked, the caller can pass any number of actual arguments. Formal parameters are assigned based on their position in the arguments list. Extra arguments are ignored during the assignment. When not enough arguments are passed, missing formal parameters are assigned to undefined. In the function body, all the arguments can be accessed using the array-like arguments object. For example, using arguments[0] to access the first actual argument. This flexibility of arguments handling is a powerful feature and enables many elegant solutions with arguments manipulation in JavaScript. However, this flexibility causes an unnecessary burden on developers to understand. TypeScript adopts a stricter restriction on arguments. The number of arguments passed to a function must match the number of formal parameters declared by this function. Passing more or fewer arguments when invoking a function is a compile-time error.

If a parameter is optional, we can add ? to the end of the parameter name, then the compiler doesn't complain when this parameter is not provided when invoking this function. Optional parameters must come after all the required parameters in the function's formal parameters list. Otherwise, there is no way to correctly assign arguments to those parameters. For example, given a function func(v1?: any, v2: any, v3?: any), when it's invoked using func(1, 2), we could not determine whether value 1 should be assigned to v1 or v2.

We can also set a default value to a parameter. If the caller doesn't provide a value or the value is undefined, the parameter will use the default value. The parameter timeout of function delay has a default value 1000. The first invocation of delay

function uses the default value of timeout, while the second invocation uses the provided value 3000.

```
function delay(func: () => void, timeout = 1000) {  
    setTimeout(func, timeout);  
}  
  
delay(() => console.log('Hello'));  
  
// -> delay 1000ms  
  
delay(() => console.log('Hello'), 3000);  
  
// -> delay 3000ms
```

3.6 INTERFACE AND CLASSES

TypeScript adds common concepts from object-oriented programming languages. This makes it very easy for developers familiar with other object-oriented programming languages to move to TypeScript.

Interfaces

Interfaces in TypeScript have two types of usage scenarios. Interfaces can be used to describe the shape of values or act as classes contracts.

Describe the Shape of Values

In typical JavaScript code, we use plain JavaScript objects as the payload of communication. But the format of these JavaScript objects is opaque. The caller and receiver need to implicitly agree on the data format, which usually involves collaboration between different team members. This type of opacity usually causes maintenance problems.

For example, a receiver function may accept an object that contains the properties name, email, and age. After a later refactoring, the development team found that the date of birth should be passed instead of the age. The caller code was changed to pass an object that contains the properties name, email, and dateOfBirth. Then the

receiver code failed to work anymore. These kinds of errors can only be found in the runtime if developers failed to spot all those places that rely on this hidden data format contract during refactoring. Because of this potential code breaking, developers tend to only add new properties while still keeping those old properties, even though those properties were not used anymore. This introduces “bad smells” to the code base and makes future maintenance much harder.

Interfaces in TypeScript provide a way to describe the shape of an object. As shown in below example, if we update interface User to remove the property age and add a new property dateOfBirth, TypeScript compiler will throw errors on all the places where the age property is used in the whole code base. This is a huge benefit for code refactoring and maintenance.

```
interface User {  
    name: string;  
    email: string;  
    age: number;  
}  
  
function processUser(user: User) {  
    console.log(user.name);  
}  
  
processUser({  
    name: 'Alex',  
    email: 'alex@example.org',  
    age: 34,  
});
```

Classes

Class is the fundamental concept in object-oriented programming languages. ES6 added the classes concept to JavaScript. TypeScript also supports classes.

Below example shows important aspects of classes in TypeScript. A class can be abstract. An abstract class cannot be instantiated directly, and it contains abstract methods that must be implemented in derived classes. Classes also support inheritance. The members of a class are public by default. public, protected, and private modifiers are supported with similar meanings as in other object-oriented programming languages.

Classes can have constructor functions to create new instances. In the constructor function of a subclass, the constructor of its parent class must be invoked using `super()`. The constructor of `Rectangle` takes two parameters `width` and `height`, but the constructor of the subclass `Square` takes only one parameter, so `super(width, width)` is used to pass the same value `width` for both parameters `width` and `height` in the `Rectangle` constructor function.

```
abstract class Shape {  
    abstract area(): number;  
}  
  
class Rectangle extends Shape {  
    private width: number;  
    private height: number;  
    constructor(width: number, height: number) {  
        super();  
        this.width = width;  
        this.height = height;  
    }  
    area() {  
        return this.width * this.height;  
    }  
}  
  
class Square extends Rectangle {
```

```
    constructor(width: number) {
      super(width, width);
    }
  }

  class Circle extends Shape {
    private radius: number;
    constructor(radius: number) {
      super();
      this.radius = radius;
    }
    area() {
      return Math.PI * this.radius * this.radius;
    }
  }

let rectangle = new Rectangle(5, 4);
let square = new Square(10);
let circle = new Circle(10);

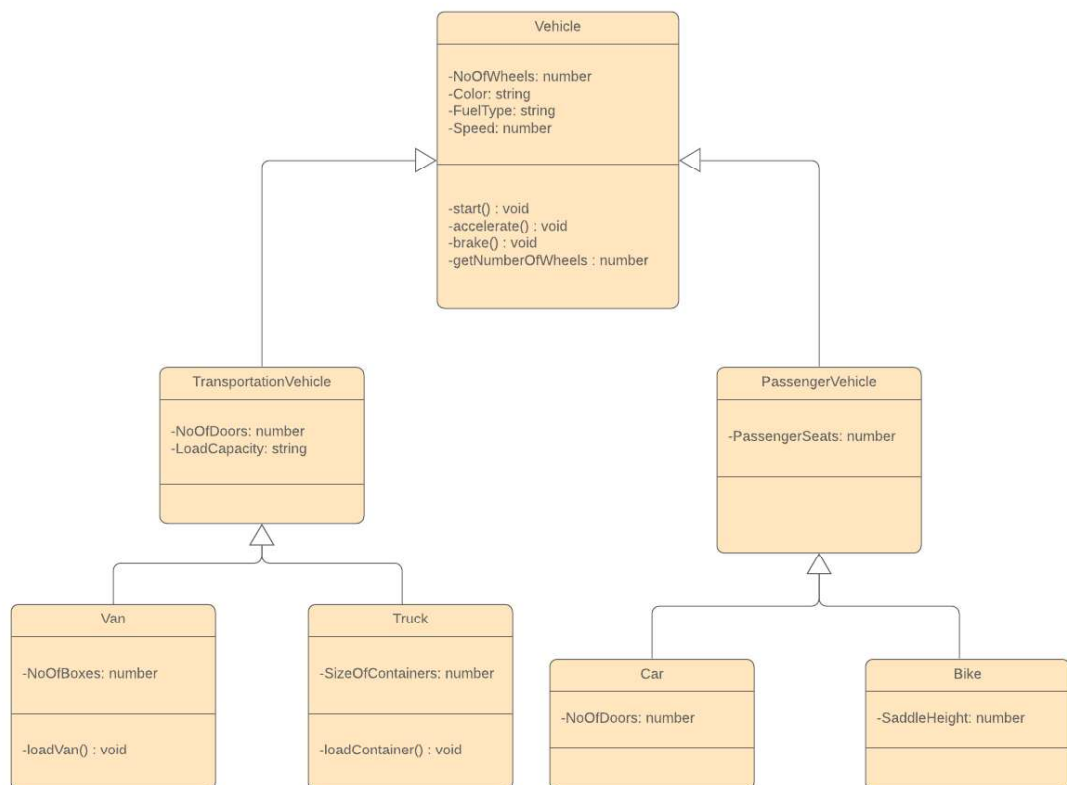
console.log(rectangle.area());
// -> 20
console.log(square.area());
// -> 100
console.log(circle.area());
// -> 314.1592653589793
```

3.7 LET US SUM UP

- Understanding Typescript
- Data Types in Typescript
- Understanding Function
- Understanding interface and Classes
- Create the following class with proper data type.

3.8 ASSIGNMENTS

Try to implement the below diagram. It contain class, variable, and function.



Block-4

Advance of IONIC

Unit 1: Ionic UI Controls

1

Unit Structure

- 4.1 Learning Objective
- 4.2 Introduction
- 4.3 Input
- 4.4 Labels
- 4.5 Checkbox
- 4.6 Radio Button
- 4.7 Selects
- 4.8 Toggles
- 4.9 Ranges
- 4.10 Header and Footer
- 4.11 Toolbar
- 4.12 Card Layout
- 4.13 List
- 4.14 Grid Layout
- 4.15 Let Us Sum Up
- 4.16 Activities

1.1 LEARNING OBJECTIVE

After studying this chapter, students should be able to understand.

- Basic UI Controls of Ionic

1.2 INTRODUCTION

To gather users' information, we need to use different input controls, including standard HTML form elements like inputs, checkboxes, radio buttons and selects; and components designed for mobile platforms, like toggles or ranges. Ionic provides out-of-box components with beautiful styles for different requirements.

1.3 INPUT

The component ion-input is for different types of inputs. This component supports the following properties.

- type – The type of the input. Possible values are text, password, email, number, search, tel, or url. The default type is text.
- value – The value of the input.
- placeholder – The placeholder of the input.
- disabled – Whether the input is disabled or not.
- clearInput – Whether to show the icon that can be used to clear the text.
- clearOnEdit – Whether to clear the input when the user starts editing the text. If the type is password, the default value is true; otherwise the default value is false.
- accept – If the type is file, this property specifies a comma-separated list of content types of files accepted by the server.
- autocapitalize – Whether the text should be automatically capitalized. The default value is none.
- autocomplete – Whether the value should be automatically completed by the browser. The default value is off.

- autocorrect – Whether auto-correction should be enabled. The default value is off.
- autofocus – Whether the control should have input focus when the page loads.
- debounce – The amount of time in milliseconds to wait to trigger the event ionChange after each keystroke. The default value is 0.
- inputmode – The hint for the browser for the keyboard to display.
- max – The maximum value.
- maxlength – The maximum number of characters that the user can enter.
- min – The minimum value.
- minlength – The minimum number of characters that the user can enter.
- step – The increment at which a value can be set. This property is used with min and max.
- multiple – Whether the user can enter multiple values. It only applies when the type is email or file.
- name – Name of the control.
- pattern – A regular expression to check the value.
- readonly – Whether the value can be changed by the user.
- required – Whether the value is required. spellcheck – Whether to check the spelling and grammar.
- size – The initial size of the control.

ion-input also supports following **events**.

- ionBlur – Fired when the input loses focus.
- ionFocus – Fired when the input has focus.
- ionChange – Fired when the value has changed.
- ionInput – Fired when a keyboard input occurred.

Below is a basic sample of using ion-input.

```
<ion-input type="text" [(ngModel)]= "name" name="name" required></ion-input>
```

1.4 LABEL

Labels can be used to describe different types of inputs. ion-label is the component for labels. It supports different ways to position the labels relative to the inputs using the property position.

- fixed - Labels are always displayed next to the inputs.
- floating - Labels will float above the inputs if inputs are not empty or have focus.
- stacked - Labels will always appear on the top of inputs.

We can add the property position to the ion-label to specify the position.

Below is a basic sample of using ion-label.

```
<ion-label floating>Username</ion-label>
```

1.5 CHECKBOX

The component ion-checkbox creates checkboxes with Ionic styles. It has the following properties.

- color - The color of the checkbox. Only predefined color names like primary and secondary can be used.
- checked - Whether the checkbox is checked. The default value is false.
- disabled - Whether the checkbox is disabled. The default value is false.

ion-checkbox also supports following **events**.

- ionBlur – Fired when the input loses focus.
- ionFocus – Fired when the input has focus.

- ionChange – Fired when the value has changed.

Below is a basic sample of using ion-checkbox.

```
<ion-checkbox [(ngModel)]="enabled"></ion-checkbox>
```

1.6 RADIO BUTTON

Radio buttons can be checked or unchecked. Radio buttons are usually grouped together to allow the user to make selections. A radio button is created using the component ion-radio. ion-radio supports properties color, checked, and disabled with the same meaning as the ion- checkbox. ion-radio also has a property value to set the value of the radio button. ion-radio supports the event ionSelect that fired when it's selected.

A radio buttons group is created by the component ion-radio-group, then all the descendant ion-radio components are put into the same group. Only one radio button in the group can be checked at the same time. It has the following properties.

- checked - Whether the radio-button is selected. The default value is false.
- color - The color of the radio-button. Only predefined color names like primary and secondary can be used.
- disabled - Whether the radio-button is disabled. The default value is false.

ion-radio also supports following **events**.

- ionBlur – Fired when the input loses focus.
- ionFocus – Fired when the input has focus.
- ionChange – Fired when the value has changed.

In the example below, we create a group with three radio buttons.

```
<ion-radio-group>
```

```
<ion-list>
```

```
<ion-list-header>
```

```

Traffic colors
</ion-list-header>
<ion-item>
  <ion-label>Red</ion-label>
  <ion-radio slot="start" value="red"></ion-radio>
</ion-item>
<ion-item>
  <ion-label>Green</ion-label>
  <ion-radio slot="start" value="green"></ion-radio>
</ion-item>
  <ion-item>
    <ion-label>Blue</ion-label>
    <ion-radio slot="start" value="blue"></ion-radio>
  </ion-item>
</ion-list>
</ion-radio-group>

```

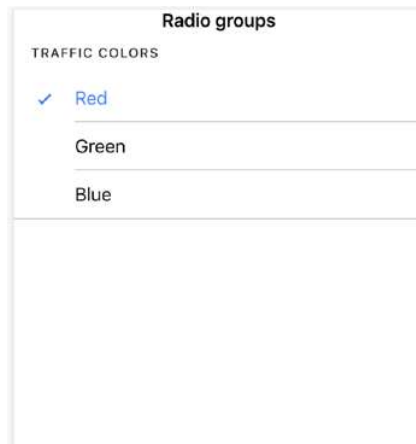


Figure-19 Radio Button

1.7 SELECTS

The component `ion-select` is similar to the standard HTML `<select>` element, but its UI is more mobile friendly. The options of `ion-select` are specified using `ion-select-option`. If the `ion-select` only allows a single selection, each `ion-select-option` is rendered as a radio button in the group. If the `ion-select` allows multiple selections, then each `ion-select-option` is rendered as a checkbox. Options can be presented using alerts or action sheets. Below are configuration options for `ion-select`.

- `multiple` – Whether the `ion-select` supports multiple selections.
- `disabled` – Whether the `ion-select` is disabled.
- `interface` – The interface to display the `ion-select`. Possible values are `alert`, `popover`, and `action-sheet`. The default value is `alert`.
- `interfaceOptions` – Additional options passed to the interface.
- `okText` – The text to display for the OK button.
- `cancelText` – The text to display for the cancel button.
- `placeholder` – The text to display when no selection.
- `selectedText` – The text to display when selected.

`ion-select` also supports the following **events**.

- `ionChange` – Fired when the selection has changed.
- `ionCancel` – Fired when the selection was canceled.
- `ionBlur` – Fired when the select loses focus.
- `ionFocus` – Fired when the select has focus.

The `ion-select` in the below example renders a **single selection select**.

```
<ion-select placeholder="Select a color">  
  <ion-select-option value="red">Red</ion-select-option>  
  <ion-select-option value="green" selected>
```

```
Green
</ion-select-option>
<ion-select-option value="blue">Blue</ion-select-option>
</ion-select>
```

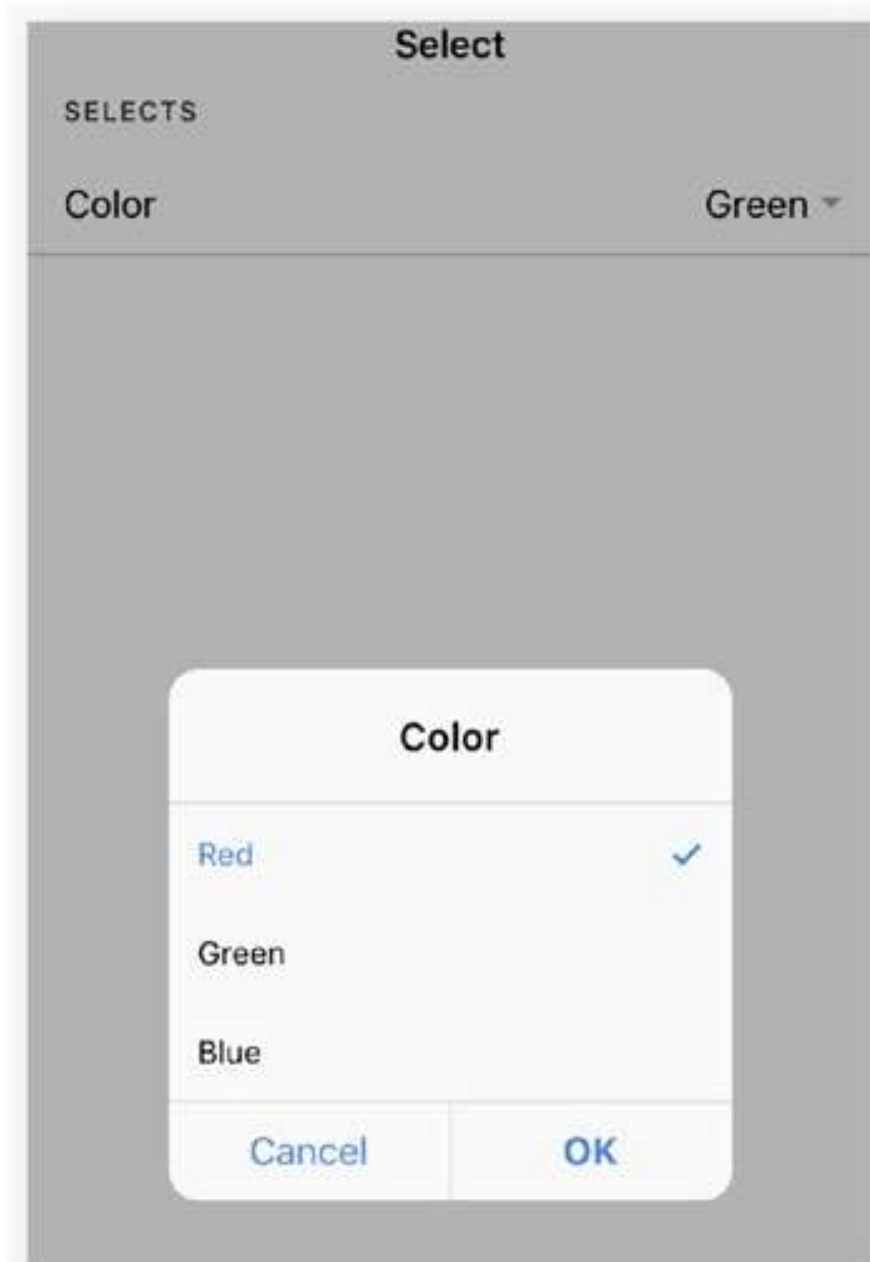


Figure-20 Select with single selection

Multiple selections select

```
<ion-select multiple="true" placeholder="Select browsers">  
  <ion-select-option>IE</ion-select-option>  
  <ion-select-option selected>Chrome</ion-select-option>  
  <ion-select-option selected>Firefox</ion-select-option>  
</ion-select>
```

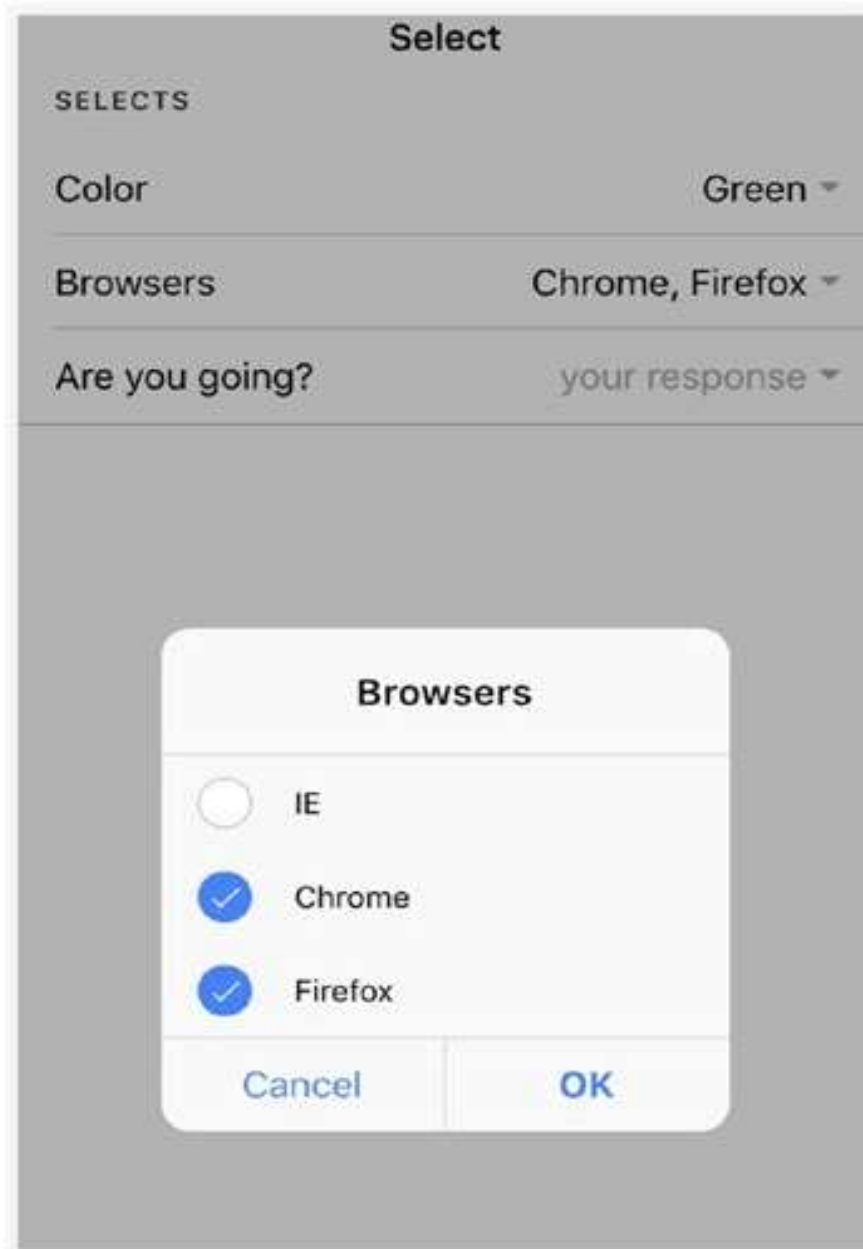


Figure-21 Select with multi-selection

Use action sheet to display

```
<ion-select interface="action-sheet" placeholder="your  
response">  
  <ion-select-option>Yes</ion-select-option>  
  <ion-select-option>No</ion-select-option>  
  <ion-select-option>Maybe</ion-select-option>  
</ion-select>
```

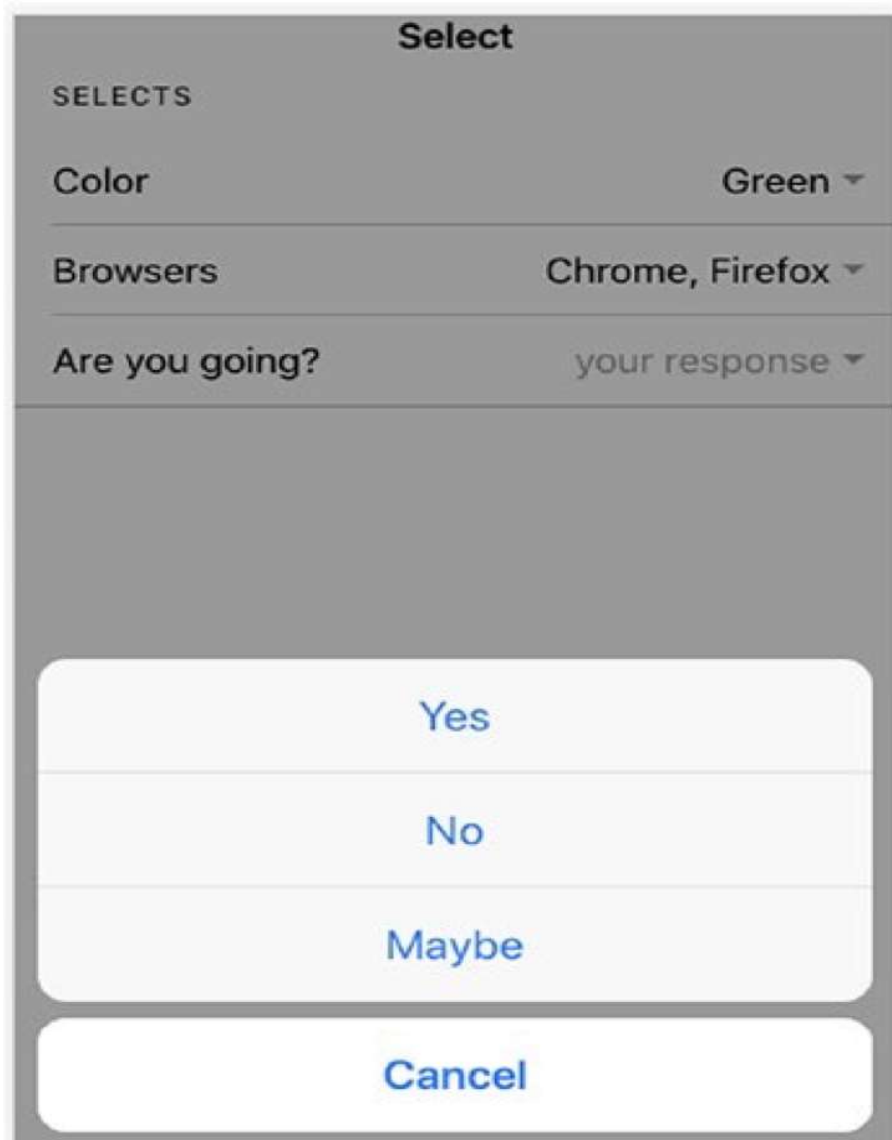


Figure-22 Select using action sheet

1.8 TOGGLES

Like checkboxes, toggles represent Boolean values but are more user friendly on the mobile platforms. `ion-toggle` supports the same properties and events as `ion-checkbox`. See the code below for a sample of `ion-toggle`.

```
<ion-toggle [(ngModel)]="enabled"></ion-toggle>
```

1.9 RANGES

Range sliders allow users to select from a range of values by moving the knobs. By default, a range slider has one knob to select only one value. It also supports using dual knobs to select a lower and upper value. Dual knobs range sliders are perfect controls for choosing ranges, that is, a price range for filtering.

The component `ion-range` has the following properties. Standard properties, including `color` and `disabled`, are omitted.

- `min` and `max` - Set the minimum and maximum integer value of the range. The default values are 0 and 100, respectively.
- `step` - The value granularity of the range that specifies the increasing or decreasing values when the knob is moved. The default value is 1.
- `snaps` - Whether the knob snaps to the nearest tick mark that evenly spaced based on the value of `step`. The default value is `false`.
- `pin` - Whether to show a pin with current value when the knob is pressed. The default value is `false`.
- `debounce` - How many milliseconds to wait before triggering the `ionChange` event after a change in the range value. The default value is 0.
- `dualKnobs` - Whether to show two knobs. The default value is `false`.

To add labels to either side of the slider, we can use the property slot of the child components of the `ion-range`. Labels can be texts, icons, or any other components.

Labels of ion-range

```
<ion-range min="1" max="5">  
  <ion-icon name="sad" slot="start"></ion-icon>  
  <ion-icon name="happy" slot="end"></ion-icon>  
</ion-range>
```

Step and snaps

```
<ion-range step="10" snaps="true" pin="true">  
  <ion-label slot="start">Min</ion-label>  
  <ion-label slot="end">Max</ion-label>  
</ion-range>
```

Double knobs

```
<ion-range dual-knobs="true" min="0" max="10000">  
  <ion-label slot="start">Low</ion-label>  
  <ion-label slot="end">High</ion-label>  
</ion-range>
```

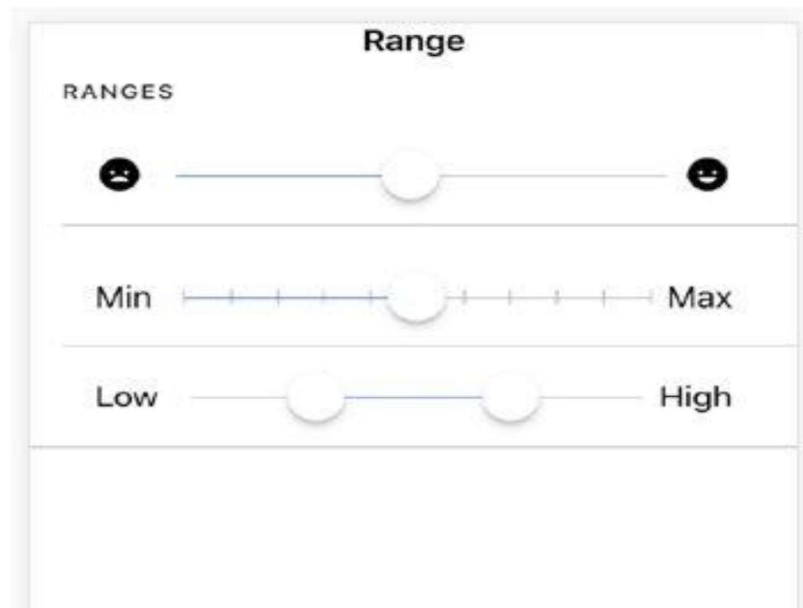


Figure-23 Range control

1.10 HEADER AND FOOTER

Header is a parent component that holds the toolbar component. It's important to note that ion-header needs to be the one of the three root elements of a page. Headers are fixed regions at the top of a screen that can contain a title label, and left/right buttons for navigation or to carry out various actions.

Footer is a root component of a page that sits at the bottom of the page. Footer can be a wrapper for ion-toolbar to make sure the content area is sized correctly.

```
<ion-header>
  <ion-navbar>
    <ion-title>Header</ion-title>
  </ion-navbar>
  <ion-toolbar>
    <ion-title>Subheader</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content></ion-content>

<ion-footer>
  <ion-toolbar>
    <ion-title>Footer</ion-title>
  </ion-toolbar>
</ion-footer>
```

1.11 TOOLBARS

A toolbar is a generic container for text and buttons. It can be used as a header, sub-header, footer, or sub-footer. Toolbars are created using the component ion-toolbar.

Buttons in a toolbar should be placed inside of the component `ion-buttons`. We can use the property `slot` to configure the position of the `ion-buttons` inside of the toolbar.

- `secondary` - On iOS, positioned to the left of the content; on Android and Windows phones, positioned to the right.
- `primary` - On iOS, positioned to the right of the content; on Android and Windows phones, positioned to the far right.
- `start` - Positioned to the left of the content in LTR, and to the right in RTL.
- `end` - Positioned to the right of the content in LTR, and to the left in RTL.

```
<ion-app>
```

```
  <ion-header>
```

```
    <ion-toolbar>
```

```
      <ion-buttons slot="start">
```

```
        <ion-button>
```

```
          <ion-icon name="menu" slot="icon-only">
```

```
        </ion-icon>
```

```
      </ion-button>
```

```
    </ion-buttons>
```

```
    <ion-title>My App</ion-title>
```

```
    <ion-buttons slot="end">
```

```
      <ion-button>
```

```
        <ion-icon name="settings" slot="icon-only">
```

```
      </ion-icon>
```

```
    </ion-button>
```

```
  </ion-buttons>
```

```
  </ion-toolbar>
```

```
</ion-header>
```

```
<ion-content padding>
```

```
        App content
    </ion-content>
</ion-app>
```

1.12 CARD LAYOUT

Cards are a great way to display important pieces of content, and are quickly emerging as a core design pattern for apps. They are a great way to contain and organize information, while also setting up predictable expectations for the user. With so much content to display at once, and often so little screen real estate, cards have fast become the design pattern of choice for many companies, including the likes of Google, Twitter, and Spotify.

For mobile experiences, Cards make it easy to display the same information visually across many different screen sizes. They allow for more control, are flexible, and can even be animated. Cards are usually placed on top of one another, but they can also be used like a "page" and swiped between, left and right.

Cards are created using the component `ion-card`. A card can have a header and content that can be created using `ion-card-header` and `ion-card-content`, respectively. Below example shows a simple card with a header and content.

```
<ion-card>
  <ion-card-header>
    Header
  </ion-card-header>
  <ion-card-content>
    Card content
  </ion-card-content>
</ion-card>
```

In the `ion-card-content`, we can include different kinds of components. The component `ion-card-title` can be used to add title text to the content. The component

ion-card-subtitle adds a subtitle to the content. Below example shows a card with an image and a title.

```
<ion-card>
  <ion-card-content>
    
    <ion-card-title>Item 1</ion-card-title>
    <ion-card-subtitle>Another item</ion-card-subtitle>
    <p>
      This is item 1.
    </p>
  </ion-card-content>
</ion-card>
```

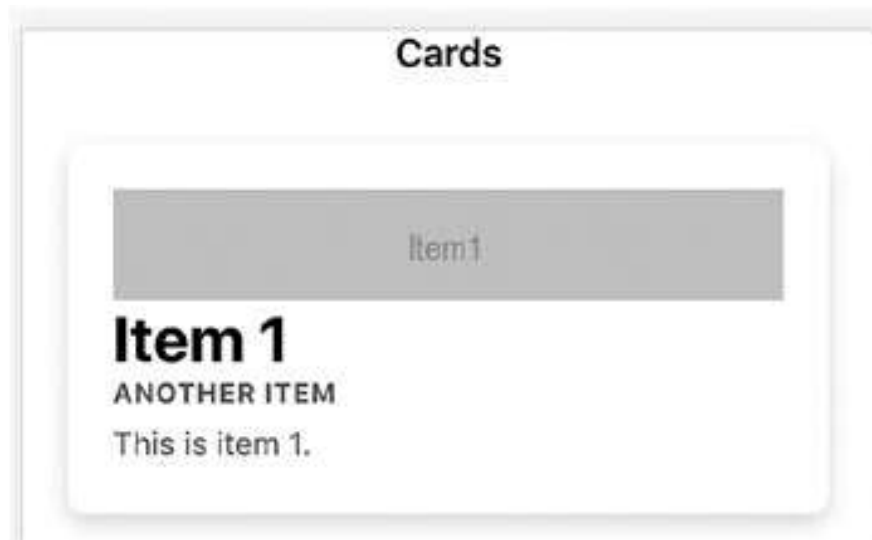


Figure-24 Card layout

1.13 LIST

Lists are one of the most common interface elements in mobile applications. They are an efficient way to display lots of information in a small space and the act of

scrolling through a list is basically second nature for most mobile users. Facebook uses a list for their news feed, as does Instagram and many others.

Given the importance of lists, the Ionic team have put a lot of effort into creating an optimised list component that has smooth scrolling, inertia, acceleration and deceleration, and everything else that gives list that nice “native” feel.

Both the list, which contains items, and the list items themselves can be any HTML element.

Using the List and Item components make it easy to support various interaction modes such as swipe to edit, drag to reorder, and removing items.

```
<ion-list>
  <ion-item>
    <ion-label>Apple</ion-label>
  </ion-item>
  <ion-item>
    <ion-label>Apricots</ion-label>
  </ion-item>
  <ion-item>
    <ion-label>Avocado</ion-label>
  </ion-item>
  <ion-item>
    <ion-label>Banana</ion-label>
  </ion-item>
  <ion-item>
    <ion-label>Blueberries</ion-label>
  </ion-item>
</ion-list>
```

1.14 GRID LAYOUT

The grid is a powerful mobile-first flexbox system for building custom layouts. It is composed of three units — a grid, row(s) and column(s). Columns will expand to fill their row, and will resize to fit additional columns. It is based on a 12 column layout with different breakpoints based on the screen size.

In the item card, we need to display two or three buttons. These buttons should take up the same horizontal space of the line. This layout requirement can be easily achieved by using the grid layout. The grid layout is implemented using the CSS3 flexbox layout (<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>).

The grid layout uses three components: `ion-grid`, `ion-row`, and `ion-col`. `ion-grid` represents the grid itself, `ion-row` represents a row in the grid, `ion-col` represents a column in a row. Rows take up the full horizontal space in the grid and flow from top to bottom. Horizontal space of a row is distributed evenly across all columns in the row. Grid layout is based on a 12-column layout. We can also specify the width for each column using attributes from `col-1` to `col-12`. The number after `col-` is the number of columns it takes in the 12-column layout, for example, `col-3` means it takes 3/12 of the whole width. By default, columns in a row flow from the left to the right and are placed next to each. We can use the property `offset-*` to specify the offset from the left side. We can use the same pattern as in `col-*` to specify the offset, for example, `offset-3` and `offset-6`. Columns can also be reordered using attributes `push-*` and `pull-*`. The attributes `push-*` and `pull-*` adjust the left and right of the columns, respectively. The difference between `offset-*` and `push-*` and `pull-*` is that `offset-*` changes the margin of columns, while `push-*` and `pull-*` change the CSS properties left and right, respectively.

For the alignment of rows and columns, we can add attributes like `align-items-start`, `align-self-start`, and `justify-content-start` to `ion-row` and `ion-col`. These attribute names are derived from flexbox CSS properties and values. For example, `align-items-start` means using `start` as the value of the CSS property `align-items`.

```
<ion-grid>
```

```
  <ion-row>
```

```
    <ion-col>
```

```
        <ion-button expand="full">1</ion-button>
</ion-col>
        <ion-col>
        <ion-button expand="full">2</ion-button>
</ion-col>
        <ion-col>
        <ion-button expand="full">3</ion-button>
</ion-col>
</ion-row>
<ion-row>
        <ion-col>
        <ion-button expand="full">4</ion-button>
</ion-col>
        <ion-col>
        <ion-button expand="full">5</ion-button>
</ion-col>
        <ion-col>
        <ion-button expand="full">6</ion-button>
</ion-col>
</ion-row>
<ion-row>
        <ion-col>
        <ion-button expand="full">7</ion-button>
</ion-col>
        <ion-col>
        <ion-button expand="full">8</ion-button>
```

```
</ion-col>
  <ion-col>
    <ion-button expand="full">9</ion-button>
  </ion-col>
</ion-row>
<ion-row>
  <ion-col col=4>
    <ion-button expand="full">0</ion-button>
  </ion-col>
  <ion-col col=8>
    <ion-button expand="full" color="secondary">=</ion-button>
  </ion-col>
</ion-row>
</ion-grid>
```

1.15 LET US SUM UP

- All the basic UI control of Ionic 4.
- How control are used, in respect to scenario.

1.16 ACTIVITIES

- With help of ion-list and ion-card create a list of country.
- With help of ion-select create a multi select for restaurant menu.
- With the help of ion-range create a RGB color screen value from 0 to 255.

Unit 2: Advanced Components

2

Unit Structure

- 2.1 Learning Objective
- 2.2 Introduction
- 2.3 Action Sheet
- 2.4 Popover
- 2.5 Slides
- 2.6 Tabs
- 2.7 Menu
- 2.8 Loading
- 2.9 Check Your Progress
- 2.10 Check Your Progress: Possible Answers
- 2.11 Activities

2.1 LEARNING OBJECTIVE

After studying this chapter, students should be able to understand.

- Advance Ionic component

2.2 INTRODUCTION

When implementing those user stories for the app, we already use many Ionic built-in components. There are still some Ionic components that are useful but not included in the app. We are going to discuss several important components, including action sheet, popover, slides, tabs, modal, and menu. After reading this chapter, you should know how to use these components.

2.3 ACTION SHEET

An action sheet is a special kind of dialog that lets user choose from a group of options. It's like the component `ion-alert` we mentioned before, but only buttons are allowed in an action sheet. It can also be used as menus. An action sheet contains an array of buttons. There are three kinds of buttons in action sheets: destructive, normal, or cancel. This can be configured by setting the property `role` to `destructive` or `cancel`. Destructive buttons usually represent dangerous actions, for example, deleting an item or canceling a pending request. Destructive buttons have different styles to clearly convey the message to the user, and they usually appear first in the array buttons. Cancel buttons always appear last in the array buttons.

Just like alerts and loading indicators, there are two Ionic components for action sheets. The component `ion-action-sheet-controller` is responsible for creating, presenting, and dismissing action sheets. The component `ion-action-sheet` is the actual component displayed to the user. Action sheets are created using the method `create()` of `ion-action-sheet-controller`.

The method `create()` takes an options object with the following possible properties.

- **header** - The title of the action sheet.
- **subHeader** - The subtitle of the action sheet.
- **cssClass** - The extra CSS classes to add to the action sheet.
- **backdropDismiss** - Whether the action sheet should be dismissed when the backdrop is tapped.
- **buttons** - The array of buttons to display in the action sheet.

Each button in the of array of buttons is a JavaScript object with the following possible properties.

- **text** - The text of the button.
- **icon** - The icon of the button.
- **handler** - The handler function to invoke when the button is pressed.
- **cssClass** - The extra CSS classes to add to the button.
- **role** - The role of the button. Possible values are `destructive`, `selected`, and `cancel`.

The return value of `create()` is a `Promise<HTMLIonActionSheet Element>` instance. After the promise is resolved, we can use methods `present()` or `dismiss()` of `HTMLIonActionSheetElement` to present or dismiss the action sheet, respectively. When the action sheet is dismissed by user tapping the backdrop, the handler of the button with role `cancel` is invoked automatically. When working with Angular, we can use the service `ActionSheetController` from Ionic Angular.

Action sheets also emit different life-cycle related events.

- **ionActionSheetDidLoad** - Emitted after the action sheet has loaded.
- **ionActionSheetDidUnload** - Emitted after the action sheet has unloaded.
- **ionActionSheetDidPresent** - Emitted after the action sheet has presented.
- **ionActionSheetWillPresent** - Emitted before the action sheet is presented.
- **ionActionSheetWillDismiss** - Emitted before the action sheet is dismissed.

- **ionActionSheetDidDismiss** - Emitted after the action sheet is dismissed.

```
export class ActionSheetComponent {  
  actionSheet: HTMLIonActionSheetElement;  
  constructor(private actionSheetCtrl: ActionSheetController) {}  
  async chooseAction() {  
    this.actionSheet = await this.actionSheetCtrl.create({  
      header: 'Choose your event',  
      backdropDismiss: true,  
      buttons: [ {  
        text: 'Remove',  
        role: 'destructive',  
        icon: 'trash',  
        handler: this.removeFile.bind(this),  
      }, {  
        text: 'Move',  
        icon: 'move',  
        handler: this.moveFile.bind(this),  
      }, {  
        text: 'Cancel',  
        role: 'cancel',  
        icon: 'close',  
        handler: this.close.bind(this),  
      } ]  
    });  
    return this.actionSheet.present();  
  }  
}
```

```
    }  
    close() {  
      this.actionSheet.dismiss();  
    }  
    removeFile() {  
    }  
    moveFile() {  
    }  
  }  
}
```

2.4 POPOVER

A popover floats on top of the current page. Popovers are created by wrapping existing components. We use the method `create()` of the component `ion-popover-controller` to create popovers. The method `create()` has only one parameter, which is a JavaScript object containing the following properties.

- **component** - The component that's wrapped in the popover.
- **componentProps** - The data object to pass to the popover component.
- **showBackdrop** - Whether to show the backdrop.
- **backdropDismiss** - Whether the backdrop should be dismissed when clicking outside of the popover.
- **cssClass** - Extra CSS classes to add.
- **enterAnimation** - Animation to use when the popover is presented.
- **event** - The click event object to determine the position of showing the popover.

The return value of `create()` is a `Promise<HTMLIonPopoverElement>` instance. The popover can be dismissed by invoking `dismiss()` of the resolved `HTMLIonPopoverElement` instance. The method `dismiss()` can accept an optional object that passed to the callback function configured by `onDidDismiss()` of the

HTMLIonPopoverElement instance. This is how data is passed between the component wrapped by the popover and the component that creates the popover.

Now we use an example to demonstrate how to pass data when using popovers; see the example below. The component contains some text, and we want to use a popover to change the font size. In the PopOverComponent, we use the injected PopoverController instance to create a new HTMLIonPopoverElement. When invoking create(), the component to show is FontSizeChooserComponent, and we pass the current value of fontSize to the component in the componentProps. The event object of the click event is passed as the value of the property event, so the popover is positioned based on the position of the click event. If no event is passed, the popover will be positioned in the center of the current view. We use present() to show the popover. We then use onDidDismiss() to add a callback function to receive the updated value of fontSize from the popover.

```
import { Component } from '@angular/core';
import { PopoverController } from '@ionic/angular';
import { PopoverComponent } from '../component/popover/popover.component';

@Component({
  selector: 'popover-example',
  templateUrl: 'popover-example.html',
  styleUrls: ['./popover-example.css']
})

export class PopoverExample {
  constructor(public popoverController: PopoverController) {}

  async presentPopover(ev: any) {
    const popover = await this.popoverController.create({
      component: PopoverComponent,
      event: ev,

```

```
        translucent: true
    });
    return await popover.present();
}
}
```

Popover also emit different life-cycle related events.

- **ionPopoverDidDismiss** - Emitted after the popover has dismissed.
- **ionPopoverDidPresent** - Emitted after the popover has presented.
- **ionPopoverWillDismiss** - Emitted before the popover has dismissed.
- **ionPopoverWillPresent** - Emitted before the popover has presented.

2.5 SLIDES

The slides component is a container for multiple views. The user can swipe or drag between different views. Slides are commonly used for tutorials and galleries.

Slides are created using components `ion-slides` and `ion-slide`. `ion-slides` is the container component for `ion-slide` components inside of it. When creating the `ion-slides`, we can use the property options to configure it. Ionic slides uses Swiper as its implementation. The property options takes the same value as in the Swiper API (<http://idangero.us/swiper/api/>). The property `pager` controls whether to show the pagination bullets.

After the slides component is created, we can also programmatically control the slide transitions using the following methods.

- **slideTo(index, speed, runCallbacks)** - Transition to the slide with the specified index.
- **slideNext(speed, runCallbacks)** - Transition to the next slide.
- **slidePrev(speed, runCallbacks)** - Transition to the previous slide.
- **getActiveIndex()** - Get the index of the active slide.

- **getPreviousIndex()** - Get the index of the previous slide.
- **length()** - Get the total number of slides.
- **isBeginning()** - Get whether the current slide is the first slide.
- **isEnd()** - Get whether the current slide is the last slide.
- **startAutoplay()** - Start autoplay.
- **stopAutoplay()** - Stop autoplay.

In below example, we create an ion-slides component with the reference variable set to slides. It contains three ion-slide components.

```
<ion-slides #slides>
  <ion-slide>
    Slide 1
  </ion-slide>
  <ion-slide>
    Slide 2
  </ion-slide>
  <ion-slide>
    Slide 3
  </ion-slide>
</ion-slides>
<div>
  <ion-button (click)="prev()">Prev</ion-button>
  <ion-button (click)="next()">Next</ion-button>
</div>
```

It has two buttons to go to the previous or next slide. The component SlidesComponent in below example has a `@ViewChild` property slides that binds to an `ElementRef` object. The property `nativeElement` returns the ion-slides element. The property `loaded` is used to check whether the slides component is loaded. The

method `componentOnReady` returns a Promise that resolved when the component is ready. The method `isValid()` is required to check whether the Slides component is ready to use.

```
import { Component, ViewChild, ElementRef, OnInit } from
```

```
'@angular/core';
```

```
@Component({
```

```
    selector: 'app-slides',
```

```
    templateUrl: './slides.component.html',
```

```
    styleUrls: ['./slides.component.css']
```

```
})
```

```
export class SlidesComponent implements OnInit {
```

```
    @ViewChild('slides') slidesElem: ElementRef;
```

```
    loaded = false;
```

```
    slides: any;
```

```
    ngOnInit() {
```

```
        this.slides = this.slidesElem.nativeElement;
```

```
        this.slides.componentOnReady().then(() => {
```

```
            this.loaded = true;
```

```
        });
```

```
    }
```

```
    prev() {
```

```
        if (this.isValid()) {
```

```
            this.slides.slidePrev();
```

```
        }
```

```
    }
```

```
    next() {
```

```

        if (this.isValid()) {
            this.slides.slideNext();
        }
    }

    isValid(): boolean {
        return this.loaded && this.slides != null;
    }
}

```

Slider also emit different life-cycle related events.

- **ionSlideDidChange** - Emitted after the active slide has changed.
- **ionSlideDoubleTap** - Emitted when the user double taps on the slide's container.
- **ionSlideDrag** - Emitted when the slider is actively being moved.
- **ionSlideNextEnd** - Emitted when the next slide has ended.
- **ionSlideNextStart** - Emitted when the next slide has started.
- **ionSlidePrevEnd** - Emitted when the previous slide has ended.
- **ionSlidePrevStart** - Emitted when the previous slide has started.
- **ionSlideReachEnd** - Emitted when the slider is at the last slide.
- **ionSlideReachStart** - Emitted when the slider is at its initial position.
- **ionSlidesDidLoad** - Emitted after Swiper initialization
- **ionSlideTap** - Emitted when the user taps/clicks on the slide's container.
- **ionSlideTouchEnd** - Emitted when the user releases the touch.
- **ionSlideTouchStart** - Emitted when the user first touches the slider.
- **ionSlideTransitionEnd** - Emitted when the slide transition has ended.
- **ionSlideTransitionStart** - Emitted when the slide transition has started.
- **ionSlideWillChange** - Emitted before the active slide has changed.

2.6 TABS

Tabs are commonly used components for layout and navigation. Different tabs can take the same screen estate, and only one tab can be active at the same time.

Tabs components are created using the component `ion-tabs`, while individual tabs are created using `ion-tab`. `ion-tabs` supports the standard properties `color` and `mode` and the following special properties.

- `tabbarHidden` - When this property is true, hide the tab bar.
- `tabbarLayout` - The layout of the tab bar. Possible values are `icon-top`, `icon-start`, `icon-end`, `icon-bottom`, `icon-hide`, `title-hide`.
- `tabbarPlacement` - The position of the tab bar. Possible values are `top` and `bottom`.
- `tabbarHighlight` - Whether to show a highlight bar under the selected tab. The default value is false.

Once `ion-tabs` is created, we can get the `ion-tab` instance of this component. The component `ion-tabs` instance provides different methods to interact with the tabs.

- `select(tabOrIndex)` - Select a tab by its index or its `ion-tab` instance.
- `getTab(index)` - Get the `ion-tab` instance by the index.
- `getSelected()` - Get the selected `ion-tab` instance.

Each `ion-tab` also supports the following properties to configure it.

- `active` - Whether the tab is active.
- `href` - The URL of the tab.
- `label` - The title of the tab.
- `icon` - The icon of the tab.
- `badge` - The badge to display on the tab button.
- `badgeStyle` - The color of the badge.
- `disabled` - Whether the tab button is disabled.
- `show` - Whether the tab button is visible.

- tabsHideOnSubPages - Whether the tab is hidden on subpages.

ion-tab also emits the event ionSelect when it's selected.

In the template file below, we create an ion-tabs with the reference name set to tabs. Each ion-tab has its title and icon. The first tab has a button to go to the second tab.

```
<ion-tabs tabbar-placement="top" #tabs>
  <ion-tab label="Tab 1" icon="alarm">
    Tab One
    <ion-button (click)="gotoTab2()">Select Tab 2</ion-button>
  </ion-tab>
  <ion-tab label="Tab 1" icon="albums">
    Tab Two
  </ion-tab>
  <ion-tab label="Tab 1" icon="settings">
    Tab Three
  </ion-tab>
</ion-tabs>
```

In the TabsComponent below, we use the decorator @ViewChild to get the reference to the ion-tabs element and use its method select() to select the second tab.

```
import { Component, OnInit, ViewChild, ElementRef } from
'@angular/core';
@Component({
  selector: 'app-tabs',
  templateUrl: './tabs.component.html',
  styleUrls: ['./tabs.component.css']
```

```

})
export class TabsComponent implements OnInit {
    @ViewChild('tabs') tabsElem: ElementRef;
    tabs: any;
    constructor() { }
    ngOnInit() {
        this.tabs = this.tabsElem.nativeElement;
    }
    gotoTab2() {
        this.tabs.select(1);
    }
}

```

2.7 MENU

Using `ion-menu-toggle` is generally enough to control the visibility of the menu. However, when there are multiple menus at both sides, or the visibility of the menu depends on complex logic, it's better to use `ion-menu-controller`. It has the following methods.

- **open(menuId)** - Open the menu with specified id or side.
- **close(menuId)** - Close the menu with specified id or side. If no menuId is specified, then all open menus will be closed.
- **toggle(menuId)** - Toggle the menu with a specified id or side.
- **enable(shouldEnable, menuId)** - Enable or disable the menu with a specified id or side. For each side, when there are multiple menus, only one of them can be opened at the same time. Enabling one menu will also disable other menus on the same side.

- **swipeEnable(shouldEnable, menuId)** - Enable or disable the feature to swipe to open the menu.
- **isOpen(menuId)** - Check if a menu is opened.
- **isEnabled(menuId)** - Check if a menu is enabled.
- **get(menuId)** - Get the ion-menu instance with a specified id or side.
- **getOpen()** - Get the opened ion-menu instance.
- **getMenus()** - Get an array of all ion-menu instances.
- **isAnimating()** - Check if any menu is currently animating.

We create two menus at the start and end side. The ion-menu-button toggles the menu at the start side. If the property contentId is not specified for the ion-menu, it looks for the element with the attribute main in its parent element as the content.

```
<ion-app>
  <ion-menu side="start">
    <ion-header>
      <ion-toolbar>
        <ion-title>Start Menu</ion-title>
      </ion-toolbar>
    </ion-header>
  <ion-content>
    <ion-list>
      <ion-item>
        <ion-button (click)="openEnd()">Open end</ion-button>
      </ion-item>
    </ion-list>
  </ion-content>
</ion-menu>
```

```

<ion-menu side="end">
  <ion-header>
    <ion-toolbar>
      <ion-title>End Menu</ion-title>
    </ion-toolbar>
  </ion-header>
</ion-menu>
<div main>
  <ion-header>
    <ion-toolbar>
      <ion-buttons slot="start">
        <ion-menu-button menu="start">
          <ion-icon name="menu"></ion-icon>
        </ion-menu-button>
      </ion-buttons>
      <ion-title>App</ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content>
    Content
  </ion-content>
</div>
</ion-app>

```

The MenuComponent uses MenuController to open the menu at the end side.

```

import { Component } from '@angular/core';
import { MenuController } from '@ionic/angular';

@Component({
  selector: 'app-menu',
  templateUrl: './menu.component.html',
  styleUrls: ['./menu.component.css']
})
export class MenuComponent {
  constructor(private menuCtrl: MenuController) {}

  openEnd() {
    this.menuCtrl.open('end');
  }
}

```

2.8 LOADING

An overlay that can be used to indicate activity while blocking user interaction. The loading indicator appears on top of the app's content, and can be dismissed by the app to resume user interaction with the app. It includes an optional backdrop, which can be disabled by setting **showBackdrop:false** upon creation.

Creating

Loading indicators can be created using a Loading Controller. They can be customized by passing loading options in the loading controller's create method. The spinner name should be passed in the spinner property, and any optional HTML can be passed in the content property. If a value is not passed to spinner the loading indicator will use the spinner specified by the platform.

Dismissing

The loading indicator can be dismissed automatically after a specific amount of time by passing the number of milliseconds to display it in the duration of the loading options. To dismiss the loading indicator after creation, call the **dismiss()** method on the loading instance. The **onDidDismiss** function can be called to perform an action after the loading indicator is dismissed.

Loading controllers supports this properties.

- **animated** - the loading indicator will animate.
- **backdropDismiss** - If true, the loading indicator will be dismissed when the backdrop is clicked.
- **cssClass** - Additional classes to apply for custom CSS. If multiple classes are provided they should be separated by spaces.
- **duration** - Number of milliseconds to wait before dismissing the loading indicator.
- **enterAnimation** - Animation to use when the loading indicator is presented.
- **keyboardClose** - If true, the keyboard will be automatically dismissed when the overlay is presented.
- **leaveAnimation** - Animation to use when the loading indicator is dismissed.
- **message** - Optional text content to display in the loading indicator.
- **spinner** - The name of the spinner to display("bubbles" | "circles" | "crescent" | "dots" | "lines" | "lines-small" | null | undefined).

Events:

ionLoadingDidDismiss - Emitted after the loading has dismissed.

ionLoadingDidPresent - Emitted after the loading has presented.

ionLoadingWillDismiss - Emitted before the loading has dismissed.

ionLoadingWillPresent - Emitted before the loading has presented.

```

import { Component } from '@angular/core';
import { LoadingController } from '@ionic/angular';

@Component({
  selector: 'loading-example',
  templateUrl: 'loading-example.html',
  styleUrls: ['./loading-example.css']
})
export class LoadingExample {
  constructor(public loadingController: LoadingController) {}

  async presentLoading() {
    const loading = await this.loadingController.create({
      message: 'Hellooo',
      duration: 2000
    });

    await loading.present();

    const { role, data } = await loading.onDidDismiss();
    console.log('Loading dismissed!');
  }

  async presentLoadingWithOptions() {
    const loading = await this.loadingController.create({
      spinner: null,
      duration: 5000,
      message: 'Please wait...',
    });
  }
}

```

```
        translucent: true,  
        cssClass: 'custom-class custom-loading'  
    });  
    return await loading.present();  
}  
}
```

2.9 CHECK YOUR PROGRESS

1. Strings are surrounded by _____ or _____
 - A. Single slash or Double slash
 - B. Single quote or Double quote
2. _____ enabled will have a compile error when for null.
 - A. nullCheck
 - B. strictNullChecks
 - C. nullNotAllowed
 - D. noNullAllowed
3. let strings: = ['1', '2', '3'] will create _____ kind of array
 - A. number
 - B. char
 - C. string
 - D. letter
4. Enum type represents _____ set of values.
 - A. fixed
 - B. variable
 - C. alike
 - D. same
5. _____ type is the escape bridge from the TypeScript world to the JavaScript world
 - A. Free
 - B. Zone
 - C. NoRule
 - D. Any

Unit 3: Advanced Topics in IONIC

3

Unit Structure

- 3.1 Learning Objective
- 3.2 Platform
- 3.3 Themes
- 3.4 Storage
- 3.5 Publish
- 3.6 Check your Progress
- 3.7 Check your Progress: Possible Answers
- 3.8 Activities

3.1 LEARNING OBJECTIVE

After studying this chapter, students should be able to understand.

- Working with different platform
- How theming working in Ionic
- Working with storage
- Publishing Ionic app.

3.2 PLATFORM

We have class Platform. It can be used to interact with the underlying platform. We used the method `ready()` of Platform to wait for the Cordova platform to finish initialization. The class Platform also has other methods.

- **platforms()** – Depends on the running device, the return value can be an array of different platforms. Possible values of platforms are android, cordova, core, ios, ipad, iphone, mobile, mobileweb, phablet, tablet, windows, and electron. When running on the iPhone emulator, the return value of `platforms()` is `["ios","iphone"]`.
- **is(platformName)** – Checks if the running platform matches the given platform name. Because the method `platforms()` can return multiple values, the method `is()` can return true for multiple values.
- **versions()** – Gets version information for all the platforms. When running on the iPhone emulator, the return value of `versions()` is `[{"name":"iphone"}, {"name":"ios", "settings":{"mode":"ios","tabsHigh light":false,"statusbarPadding":false,"keyboard Height":250,"isDevice":true,"deviceHacks":true}}]`.
- **isRTL()** – Checks if the language direction is right to left.
- **width() and height()** – Gets the width and height of the platform's viewport, respectively.

- **isPortrait() and isLandscape()** – Checks if the app is in portrait or landscape mode, respectively.
- **ready()** – This method returns a promise that is resolved when the platform is ready and we can use the native functionalities. The resolved value is the name of the platform that was ready.
- **url()** – Gets web page's URL.
- **getQueryParam(key)** – Gets query parameter.

There are three important EventEmitters in the Platform that are related to app states. The EventEmitter `pause` emits events when the app is put into the background. The EventEmitter `resume` emits events when the app is pulled out from the background. These two EventEmitters are useful when dealing with app state changes. The EventEmitter `resize` emits events when the browser window has changed dimensions.

3.3 THEMING

Theme support is baked right into Ionic apps. Changing the theme is as easy as updating the `$colors` map in your `src/theme/variables.scss` file:

```
$colors: (  
  primary: #488aff,  
  secondary: #32db64,  
  danger: #f53d3d,  
  light: #f4f4f4,  
  dark: #222  
);
```

The fastest way to change the theme of your Ionic app is to set a new value for `primary`, since Ionic uses the primary color by default to style most components. Colors can be removed from the map if they aren't being used, but `primary` should not be removed.

Ionic provides different look and feels based on the current platform. The styles are grouped as different modes. Each platform has a default mode that can also be overridden. It's possible to use iOS styles on Android devices. here are two modes: md for Material Design styles, ios for iOS styles. The platform ios uses the mode ios by default, and other platforms use the mode md by default.

Ionic uses modes to customize the look of components. Each platform has a default mode, but this can be overridden. For example, an app being viewed on an Android platform will use the md (Material Design) mode. The <ion-app> will have class="md" added to it by default and all of the components will use Material Design styles.

Platform	Mode	Details
ios	ios	Viewing on an iphone, ipad, or ipod will use the iOS styles.
android	Md	Viewing on any android device will use the Material Design styles.
windows	Wp	Viewing on any windows device inside cordova or electron uses the Windows styles.
core	Md	Any platform that doesn't fit any of the above platforms will use the Material Design styles.

Table-7 Platform and Mode

Overriding the Mode Styles

Each Ionic component has up to three stylesheets used to style it. For example, the tabs component has a core stylesheet that consists of styles shared between all modes, a Material Design stylesheet which contains the styles for the md mode, an iOS stylesheet for the ios mode, and a Windows stylesheet for the wp mode. Not all components are styled differently for each mode, so some of them will only have the core stylesheet, or the core stylesheet and one of the mode stylesheets.

You can use the class that is applied to the ion-app to override styles. For example, if you wanted to override all buttons in Material Design (md) mode to have capitalized text:

Once the mode is selected for the app, the html element will have the attribute mode set to the mode name, for example, `<html mode="ios">`. The element `<ion-app>` will have the mode name as a CSS class name, for example, `<ion-app class="md">` for the mode md. This class name can be used to override styles for different modes. In the code below, we add extra styles only for the mode md.

```
.md {
    font-size: 16px;
}

.md .button {
    text-transform: capitalize;
}

.button-md {
    text-transform: capitalize;
}
```

Ionic components have the property mode to set the mode. This mode overrides the platform's default mode for this component only.

```
<ion-app>
  <ion-header>
    <ion-toolbar>
      <ion-title>Range</ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content padding>
    <ion-list>
      <ion-item>
```

```

<ion-label slot="start">md</ion-label>
<ion-range mode="md" value="50">
  <ion-icon mode="md" slot="start" name="angry">
  </ion-icon>
  <ion-icon mode="md" slot="end" name="smile">
  </ion-icon>
</ion-range>
</ion-item>
<ion-item>
<ion-label slot="start">ios</ion-label>
<ion-range mode="ios" value="50">
  <ion-icon mode="ios" slot="start" name="angry">
  </ion-icon>
  <ion-icon mode="ios" slot="end" name="smile">
  </ion-icon>
</ion-range>
</ion-item>
</ion-list>
</ion-content>
</ion-app>

```

Ionic has different Sass variables to configure the styles. These variables can be overridden in the file `src/theme/variables.scss`. For example, the variable `$button-md-font-size` configures the button font size of mode `md`. The default value is `14px`. We can add the code below to the file `variables.scss` to change the variable's value.

```
$button-md-font-size: 16px;
```

3.4 STORAGE

To store data in app, and so the user can access all the data across different devices. For some cases, it's not required that the data needs to be shared between different devices.

For this kind of data, we can store the data on the device. In this case, we can use the key/value pairs storage provided by Ionic. The package `@ionic/storage` is already installed as part of the starter template, so we can use it directly.

The storage stores key/value pairs. The value of each pair can be data of any type. If the value is a JavaScript object, it's serialized to a JSON string before saving. When the data is retrieved, the JSON string is deserialized back to a JavaScript object. The Ionic storage package wraps the `localForage` library (<https://github.com/localForage/localForage>). It provides a common API to access different storage engines, including SQLite, IndexedDB, WebSQL, and `localStorage`. The actual engine used in the runtime depends on the availability of the platform. The best storage engine to use is SQLite, because it's natively supported on iOS and Android platforms. We can install the plugin `cordova-sqlite-storage` to make SQLite available on different platforms.

```
$ cordova plugin add cordova-sqlite-storage --save
```

We use the class `Storage` from `@ionic/storage` to interact with the underlying storage engine. The module created by `IonicStorageModule.forRoot()` should be imported. The instance of class `Storage` can be injected into components. `Storage` has the following methods.

- **get(key)** – Gets the value by key.
- **set(key, value)** – Sets the value of the given key.
- **remove(key)** – Removes the given key and its value.
- **clear()** – Clears the whole store.
- **keys()** – Gets all the keys in the store.
- **length()** – Gets the number of keys.

- **forEach(callback)** – Invokes the callback function for each key/value pair in the store.

Most of the operations in Storage are asynchronous. The return values of methods `get()`, `set()`, `remove()`, and `clear()` are all Promise objects that resolved when the operations are completed. In below example, we use `set()` to set the value first, then use `get()` to read the value and assign it to the property value.

```
import { Component, OnInit } from '@angular/core';
```

```
import { Storage } from '@ionic/storage';
```

```
@Component({
```

```
  selector: 'app-page-home',
```

```
  templateUrl: 'home.page.html',
```

```
  styleUrls: ['home.page.scss'],
```

```
})
```

```
export class HomePage implements OnInit {
```

```
  value: any;
```

```
  constructor(private storage: Storage) { }
```

```
  ngOnInit() {
```

```
    const obj = {
```

```
      name: 'Alex',
```

```
      email: 'alex@example.org'
```

```
    };
```

```
    this.storage.set('value', obj)
```

```
      .then(_ => this.storage.get('value')
```

```
        .then(v => this.value = v));
```

```
  }
```

```
}
```

The method `IonicStorageModule.forRoot()` accepts an optional object to configure the storage engine. This object has the following properties.

- **name** - Name of the storage.
- **storeName** - Name of the store.
- **driverOrder** - The array of driver names to test and use. The default value is `['sqlite', 'indexeddb', 'websql', 'localstorage']`.

3.5 PUBLISH

After the app has been developed and tested, it's time to publish it to app stores. Here, we'll discuss tasks related to app publish. An app needs to have proper icons and splash screens. Ionic provides a way to generate these icons and splash screens.

3.5.1 ICONS AND SPLASH SCREENS

Before the app can be published, we need to replace the default icons and splash screens. Ionic can generate icons and splash screens from source images to create images of various sizes for each platform. We only need to provide an image for the icon and another image for the splash screen, then Ionic can generate all necessary images. Source images can be `.png` file, `.psd` file from PhotoShop or `.ai` file from Adobe Illustrator.

For icons, the source image should be file `icon.png`, `icon.psd` or `icon.ai` in the directory `resources` of the Ionic project. The icon image should have a size of at least `192 x 192 px` without the round corners. For splash screens, the source image should be file `splash.png`, `splash.psd` or `splash.ai` in the directory `resources`. The splash screen should have a size of at least `2732 x 2732 px` with the image centered in the middle.

We use the command `ionic resources` to generate those resource files for icons and splash screens.


```
// Icons only
```

```
$ ionic resources –icon
```

```
// Splash screens only
```

```
$ ionic resources –splash
```

```
// Both icons and splash screens
```

```
$ ionic resources
```

Generated icons and splash screens are saved to the subdirectory `ios` and `android` of the directory `resources`.

3.5.2 DEPLOY TO DEVICES

We can deploy the app to a device for testing. For iOS, open the generated project in the directory `platforms/ios` with Xcode and use Xcode to deploy to the device. For Android, open the generated project in the directory `platforms/android` with Android Studio to deploy to the device.

Ionic CLI commands `ionic run ios` and `ionic run android` can also be used to deploy apps to the device.

Ionic Deploy

After we publish the app's first version to app stores, we need to continuously release new versions to the users. Usually, these new versions need to go through the same review process as the first version, which may take a long time to finish. This can delay the delivery of new features and bug fixes. For Cordova apps, since the majority code is written in HTML, JavaScript, and CSS, it's possible to perform live updates without installing new versions. These static files can be replaced by the wrapper to update to the new versions. Ionic Pro provides the deploy service to perform live deployments.

You'll need an Ionic Pro account to use this feature. After being logged in to Ionic Pro, we need to create a new app in the dashboard and link the app to Ionic Pro. Because we already created the Ionic app, the following command is used to link it. You can find the `app_id` in the dashboard.

\$ ionic link --pro-id <app_id>

Ionic Pro uses a Git-based workflow to manage app updates. The command `ionic link` will prompt to set up the Git repository. Just follow the instructions displayed when running `ionic link` to finish the setup. Here we use Ionic Pro as the Git repository. A new Git remote called `ionic` is added to the repository, and we can push the current code to this remote. After the link, the file `ionic.config.json` is updated to include the property `pro_id`.

In the Ionic Pro dashboard for the app, go to the tab `Code` and select `Channels`. Two channels `Master` and `Production` have already been created. `Master` channel is for binaries for development, while `Production` channel is for binaries for app stores. Clicking the button `Set up deploy` next to a channel shows a dialog with instructions on how to set up the deploy. There are three options of how updates are installed.

- “background” mode checks for updates when the app is opened from a completely closed state. It will download the update in the background when the user is using the app. The update is applied when the app is closed and opened the next time.
- “auto” mode checks for updates when the app is opened from a completely closed state. It will wait on the splash screen until the update is downloaded and applied. This mode forces users to always use the latest version.
- “none” mode doesn’t download or apply updates automatically. The entire update process is managed by you using the plugin API. This is not recommended as it may break the app with broken updates. Using background and auto mode won’t have this issue as the updates in these two modes are done in the native layer.

We are going to use the background mode for the app. The dialog already shows the command to run to install the plugin `cordova-plugin-ionic`.

```
$ cordova plugin add cordova-plugin-ionic --save \  
--variable APP_ID="<app_id>" \  
--variable CHANNEL_NAME="Master" \  
--variable UPDATE_METHOD="background"
```

After a commit is pushed to the Git repository, a new build will run. You can check the builds in the tab Builds. For each build, it can be manually deployed to a channel. A channel can also be configured to auto- deploy builds in a Git branch.

3.6 CHECK YOUR PROGRESS

1. .scss supports variable
 - A. True
 - B. False
2. Is it possible to use iOS styles on Android devices.
 - A. Yes
 - B. No
3. _____ mode is use for Android.
 - A. md
 - B. ios
 - C. an
 - D. wp
4. cordova-sqlite-storage makes _____ database available on iOS and Android.
 - A. Oracle
 - B. MS-Access
 - C. MongoDB
 - D. SQLite
5. _____ is used to clear the storage
 - A. clear()
 - B. removeAll()
 - C. remove()
 - D. deleteAll()

3.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

- | | | |
|------|------|------|
| 1. A | 2. A | 3. A |
| 4. D | 5. A | |

3.8 ACTIVITIES

- Create a app and assign Splash Screen and Icon for Specific Platform
- Create a app and use Primary and Secondary color for changing the Button and App Background
- Try to create custom Theme variable and use it in app
- Try to go through Config.xml and then build the individual plarform

યુનિવર્સિટી ગીત

સ્વાધ્યાય: પરમં તપ:

સ્વાધ્યાય: પરમં તપ:

સ્વાધ્યાય: પરમં તપ:

શિક્ષણ, સંસ્કૃતિ, સદ્ભાવ, દિવ્યબોધનું ધામ
ડૉ. બાબાસાહેબ આંબેડકર ઓપન યુનિવર્સિટી નામ;
સૌને સૌની પાંખ મળે, ને સૌને સૌનું આભ,
દશે દિશામાં સ્મિત વહે હો દશે દિશે શુભ-લાભ.

અભણ રહી અજ્ઞાનના શાને, અંધકારને પીવો ?
કહે બુદ્ધ આંબેડકર કહે, તું થા તારો દીવો;
શારદીય અજવાળા પહોંચ્યાં ગુર્જર ગામે ગામ
ધ્રુવ તારકની જેમ ઝળહળે એકલવ્યની શાન.

સરસ્વતીના મયૂર તમારે ફળિયે આવી ગહેકે
અંધકારને હડસેલીને ઉજાસના ફૂલ મહેંકે;
બંધન નહીં કો સ્થાન સમયના જવું ન ઘરથી દૂર
ઘર આવી મા હરે શારદા દૈન્ય તિમિરના પૂર.

સંસ્કારોની સુગંધ મહેંકે, મન મંદિરને ધામે
સુખની ટપાલ પહોંચે સૌને પોતાને સરનામે;
સમાજ કેરે દરિયે હાંકી શિક્ષણ કેરું વહાણ,
આવો કરીયે આપણ સૌ
ભવ્ય રાષ્ટ્ર નિર્માણ...
દિવ્ય રાષ્ટ્ર નિર્માણ...
ભવ્ય રાષ્ટ્ર નિર્માણ

