

# Section 1

# Introduction to Web App development

---

## Working with HTML

### Structure

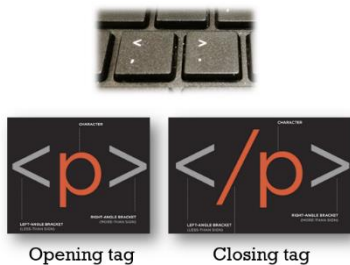
Many web apps act like electronic versions of documents such as Newspapers, which has headline, some text, and possibly some images. If the article is a long piece, there may be subheadings that split the story into separate sections or quotes from those involved. Structure helps readers understand the stories in the newspaper.

**Structure** is a way to build a web app to organize and make it easier for the reader to follow what he/she is reading. HTML code builds the structure or layout of a web app.



### The HTML code

The HTML code is made up of characters that live inside angled brackets, which are called HTML **elements**. Elements are usually made up of two **tags**: an opening tag and a closing tag.



Each HTML element tells the browser something about the information that sits between its opening and closing tags.

### Basic HTML elements

The basics HTML elements are:

1. The opening **<html>** tag element indicates that anything between it and a closing **</html>** tag is HTML code

2. A **<head>** element contains information about the page such as title
3. The contents of the **<title>** element are either shown in the top of the browser, above where you usually type in the URL of the page you want to visit, or on the tab for that page (if your browser uses tabs to allow you to view multiple pages at the same time).
4. The **<body>** tag indicates that anything between it and the closing tag **</body>** should be inside the main browser window.

**Example 1)** Explore HTML elements to create a first smartphone app.

For this example we are going to create our first app view using HTML and check our code through google Chrome browser and Android Studio Emulator.

#### Steps:

1. We start by writing the HTML code using an IDE text editor. We can use Visual Studio Code, VS code.



2. In VS code, open a new file and save it with your last name and the file extension **.html**. Make sure that you save the file in your local drive or computer Desktop.
3. The first line that we write in an html file is **<html>** tag to declare the html file. One of the great thing about VS code is that when you type a tag name, it will automatically build a basic code structure of the code. Write **html** and click on **html: 5**. VS code will create the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
</body>
</html>
```

The **<!DOCTYPE>** declaration must be the very first thing in your HTML document, before the **<html>** tag and it specifies the rules for the markup language, so that the browsers render the content correctly.

The attribute **lang** indicates the language and the value **en** means English. This line specifies that the html code is based in English.

The attribute **dir** indicates the direction of the text and the value **ltr** indicates that the text will be display from left-to-right.

Metadata is data (information) about data. The **<meta>** tag provides metadata about the HTML document. Metadata will not be displayed on the page, but will be machine parsable.

Meta elements are typically used to specify page description, keywords, author of the document, last modified and other metadata

**UTF-8** is the Unicode character sets from 1 to 4 bytes long. UTF-8 can represent any character in the Unicode standard. UTF-8 is backwards compatible with ASCII and the encoding for e-mail and web pages.

For app view development, it is an important to add the following line:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

The **viewport** is the user's visible area of a web page. It varies with the device, and will be smaller on a mobile phone than on a computer screen. viewport element gives the browser instructions on how to control the page's dimensions and scaling.



Without the viewport meta tag



With the viewport meta tag

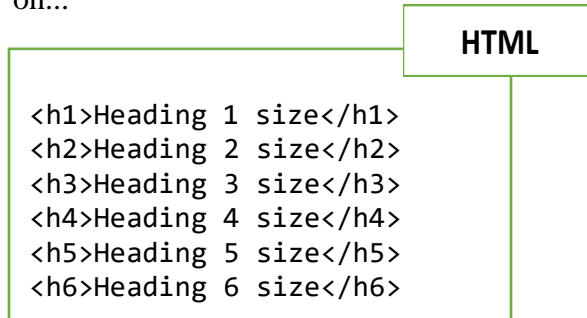
The **width=device-width** part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The **initial-scale=1.0** part sets the initial zoom level when the page is first loaded by the browser.

Once we have the **<head>** element sets, we can use some tags in the **<body>**. Most of the tags that we use in **<body>** are visible in the web app. We are try to add some heading the body.

HTML has six levels of headings:

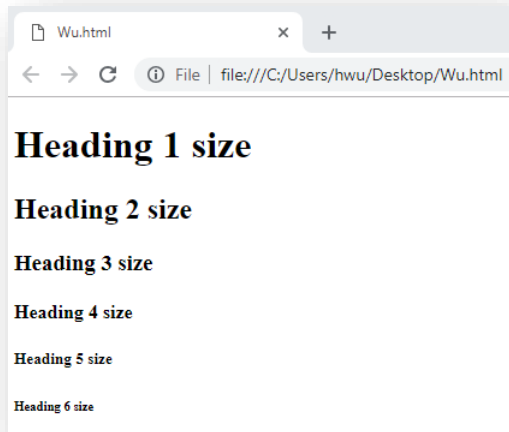
- `<h1>` is used for main headings
- `<h2>` is used for subheadings.
- If there are further sections under the subheadings then the `<h3>` element is used, and so on...



To see the output of the html code, we can go to the location of our html file, and double click on html icon



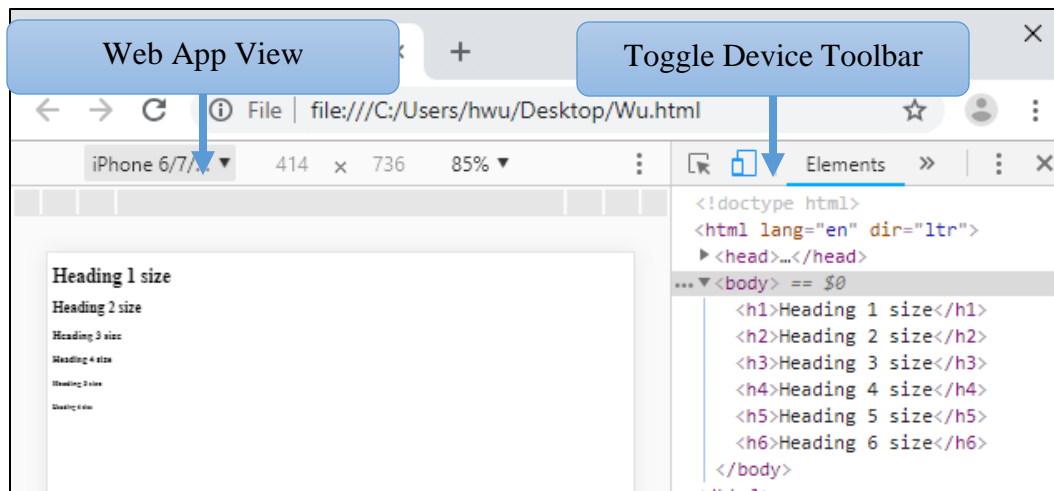
The html file will open in internet browser



Since we are creating web app, we can use the browser development tools. To get to the development tools in Google Chrome, we just hit the function key F12 from our keyboard



In the development window, we can select the “Toggle Device Toolbar”, and select the size of the app view.



Now that we have learn the basic set up of an app view, we can try more html elements.

## More elements

### Paragraph

The **<p> tag** specifies a paragraph of text. It is a block-level element and it automatically have margin before and after the tag.

```
<p>Queensborough Community College - QCC</p>
```

### Bold

By enclosing words in the tags **<b>** and **</b>** we can make characters appear bold.

```
<p>Inside a product description you might see some <b>key features</b> in bold.</p>
```

### Italic

By enclosing words in the tags **<i>** and **</i>** we can make characters appear italic.

```
<p>My cousin is reading <i>The Adventures of Tom Sawyer</i> for two classes: <i><b>English</b></i> and <i><b>History</b></i>.</p>
```

## Line Breaks

The browser will automatically show each new paragraph or heading on a new line. But if you wanted to add a line break inside the middle of a paragraph you can use the line break tag `<br/>`

```
<p>The Earth<br/>gets one hundred tons heavier every day <br/>due to falling space dust. </p>
```

## Horizontal Rules

To create a break between themes — such as a change of topic in a book or a new scene in a play — you can add a horizontal rule between sections using the `<hr/>`

```
<p>Venus is the only planet that rotates clockwise.</p>
<hr /> <p>Jupiter is bigger than all the other planets combined.</p>
```

## Strikethrough

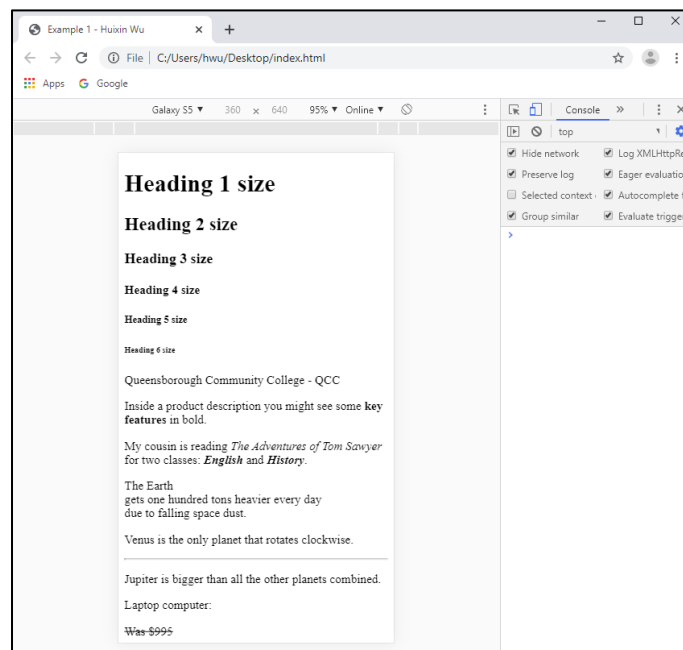
Strikethrough means to cross something out by drawing a line through it.

The `<s>` element indicates something that is no longer accurate or relevant (but that should not be deleted).

Visually the content of an `<s>` element will usually be displayed with a line through the center.

```
<p>Laptop computer:</p> <p><s>Was $995</s></p> <p>Now only $375</p>
```

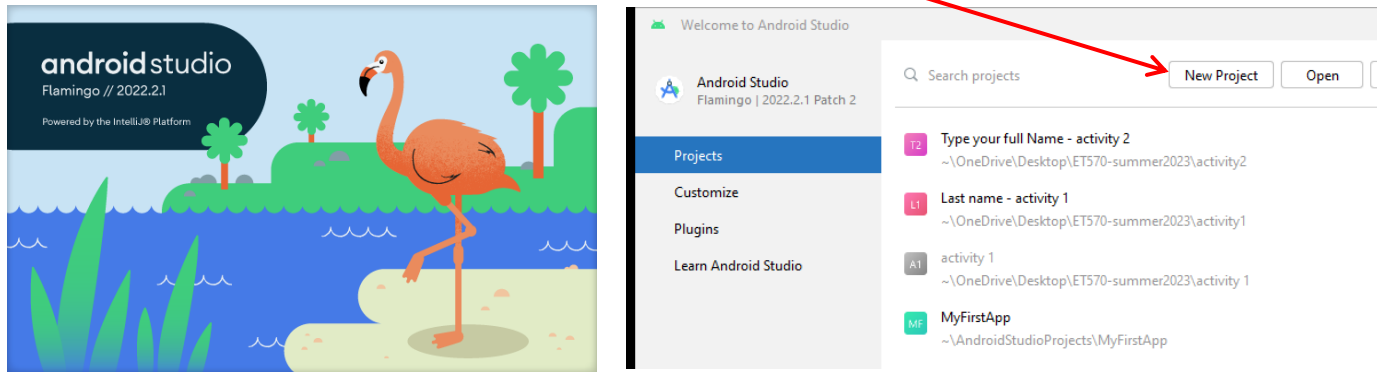
If we refresh the internet browser, the output will look as:



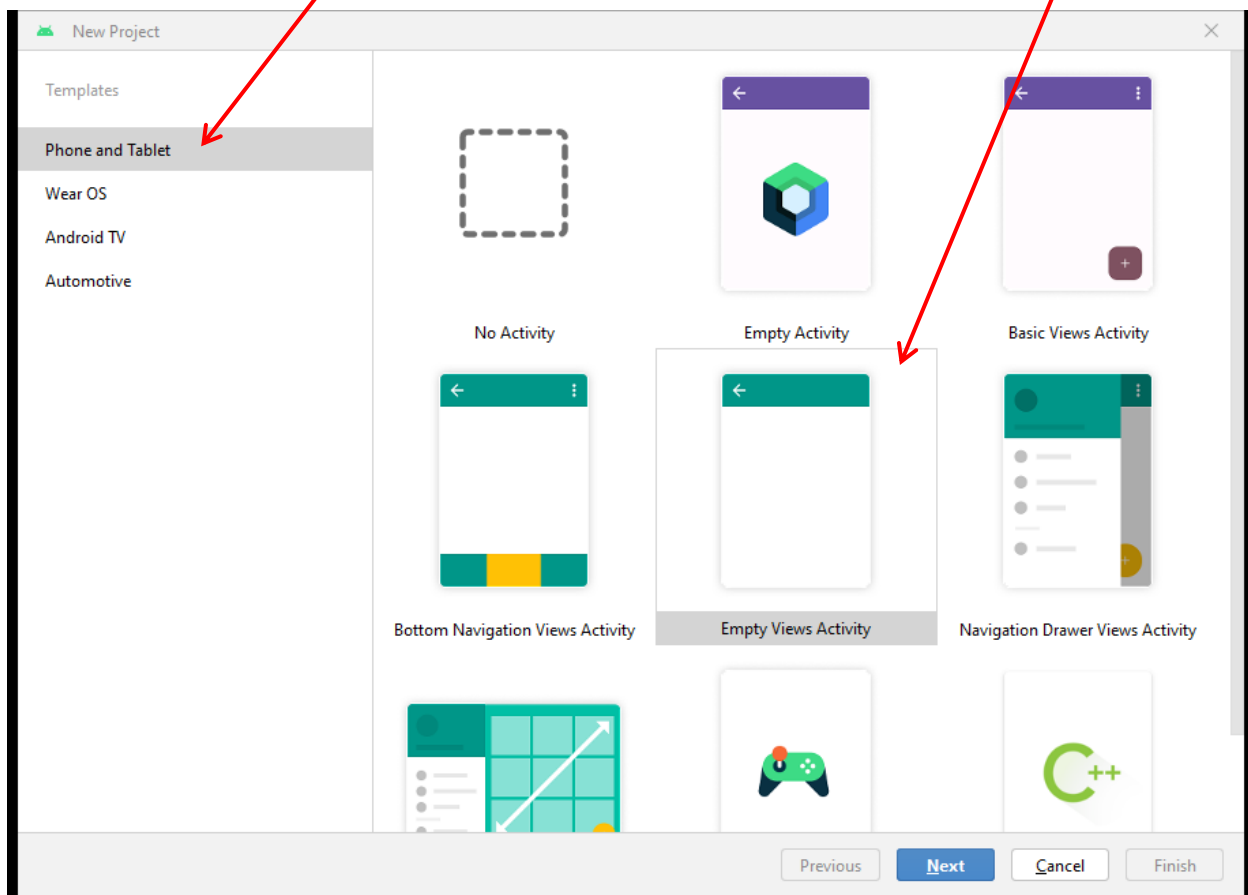
Now is time to convert a web app into a smartphone app using cordova and Android Studio.

## How to convert a web app to android app

**Step 1)** Open Android Studio and select Start a **New Project** Android Studio project



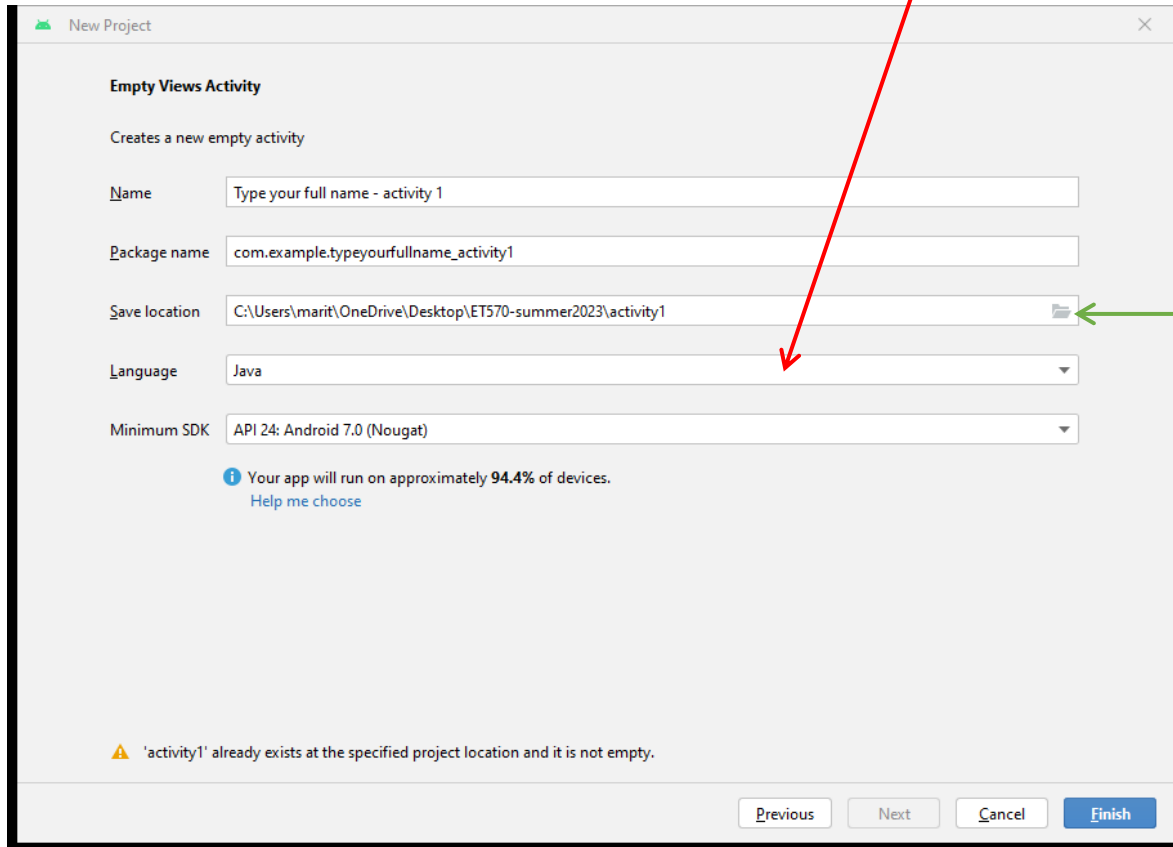
**Step 2)** In the New Project window, since we are converting a web app into an Android app, make sure to select **Phone and Tablet** in Templates. From the right list, select **Empty Views Activity**. Click **Next** when your selection is done.



**Step 3)** in the **Empty View Activity** window, if you want, you can change the Application name and the Company domain. That two information is to Android Studio to create a java package under the Company domain name and the main java class to the Application name.

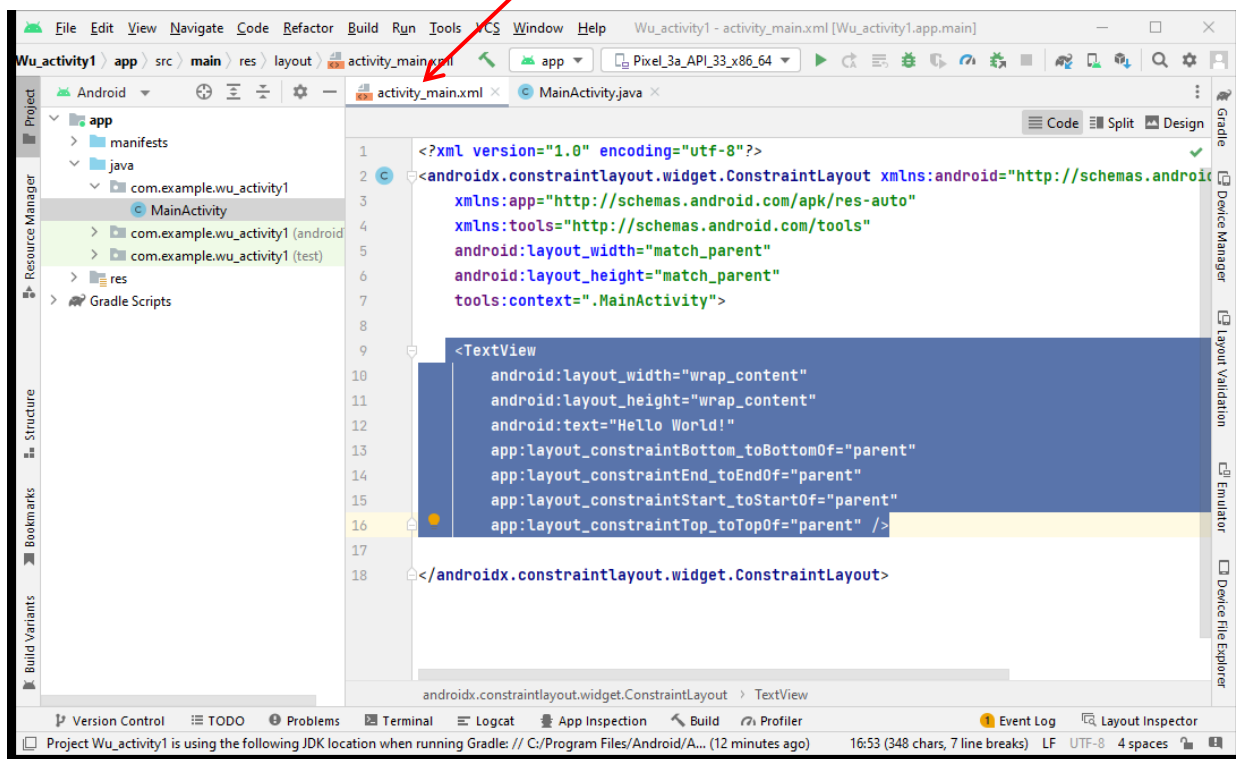
In **Save location**, open a directory (folder) to save your application activity, in this case, we can create a directory named *activity1* and save our app activity in *activity1*.

If you are working with Java, you can click on **Language** and select **Java**. Click on **Finish** to launch to Android Studio

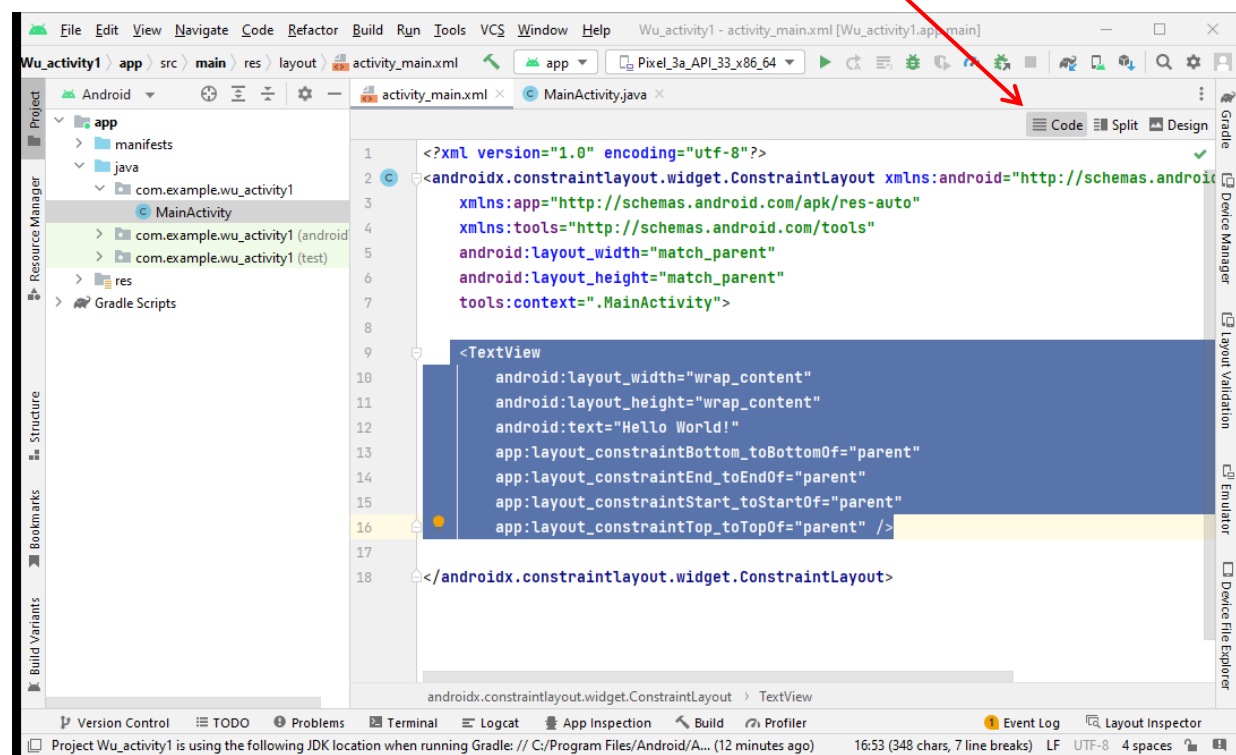




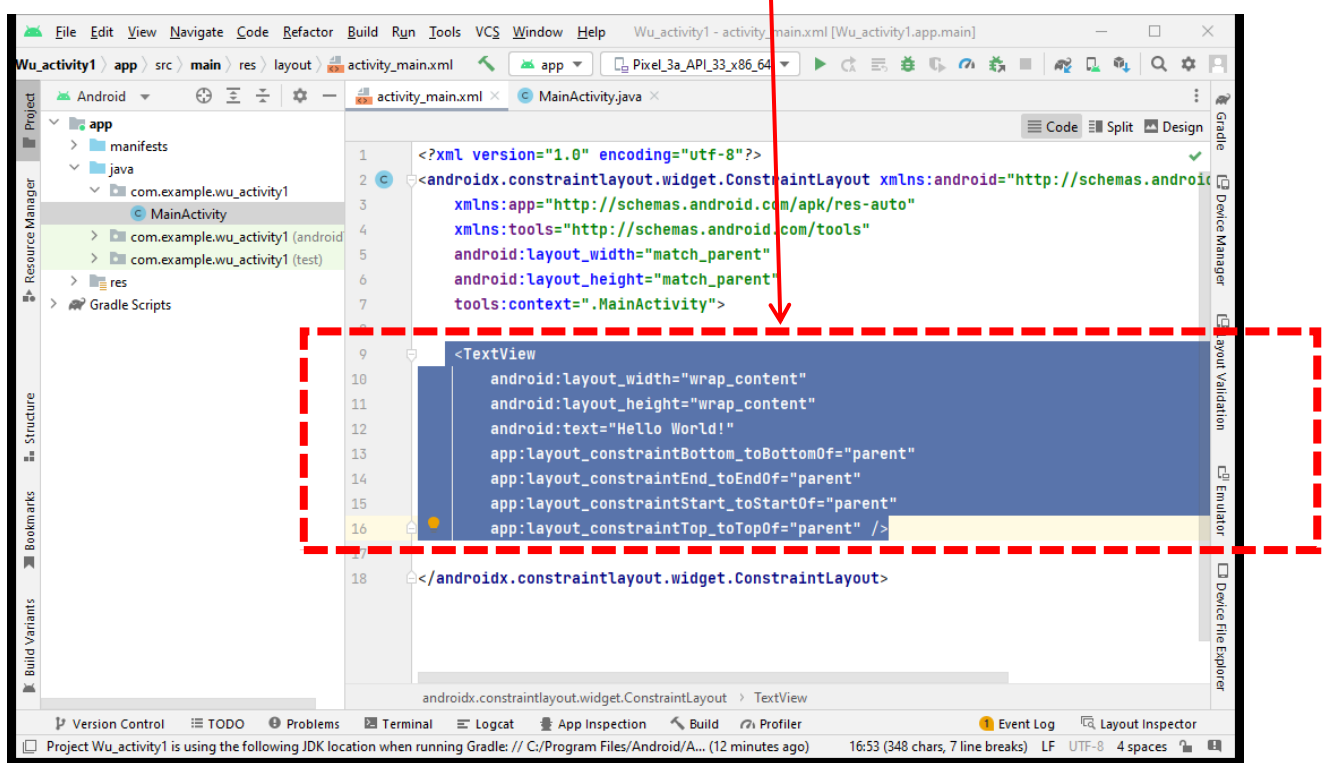
**Step 4)** Modifying *activity\_main.xml* file. Once in Android Studio, click on *activity\_main.xml* file, which is located on the top of the workspace.



Once in the file, go to the right-top of the workspace and click the **Code** tab.



In the XML code, select and delete the complete **<TextView>** tag



Add the **<WebView>** tag in the XML file. The **<WebView>** tag is used to display the web app in the android device window.

```
<WebView
    android:id="@+id/webView"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</WebView>
```

The complete **activity\_main.xml** code will look as follows

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <WebView
        android:id="@+id/webView"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </WebView>

</androidx.constraintlayout.widget.ConstraintLayout>
```

**Step 5) Modifying MainActivity.java file.** Once the **assets** folder is set, now we need to write a JAVA code to display the <WebView> in an Android phone. To do so, click on **MainActivity.java** file and delete all the code from line 7, which is the line that starts with **public class**. Once the lines are deleted, copy and paste the following JAVA lines of code from line 7:

```
import android.webkit.WebChromeClient;
import android.webkit.WebView;

public class MainActivity extends AppCompatActivity {
    private WebView activity;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        activity = (WebView) findViewById(R.id.webView);
        activity.getSettings().setJavaScriptEnabled(true);
        activity.loadUrl("file:///android_asset/index.html");

        // Interacting with browser
        activity.setWebChromeClient(new WebChromeClient()); } }
```

Remember that **loadUrl** constructor is used to load the HTML file, for this, we will need to add the name of the loading HTML: **project1.loadUrl("file:///android\_asset/index.html");**

The complete *MainActivity.java* file looks as

```
package com.example.activity2_wu;


import androidx.appcompat.app.AppCompatActivity;

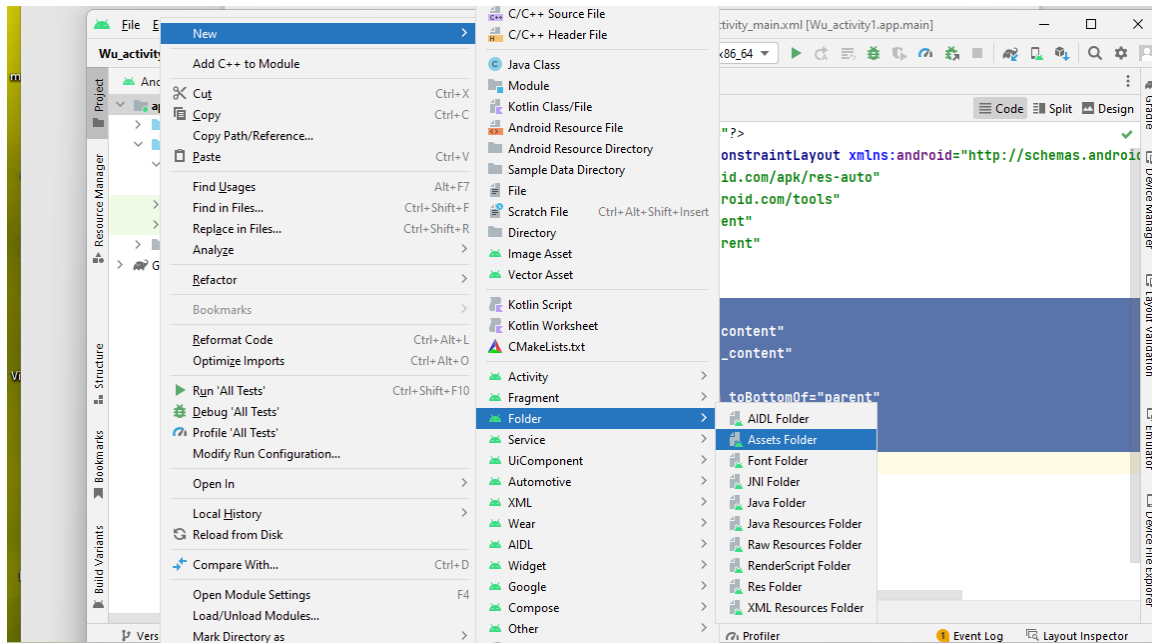
import android.os.Bundle;

import android.webkit.WebChromeClient;
import android.webkit.WebView;

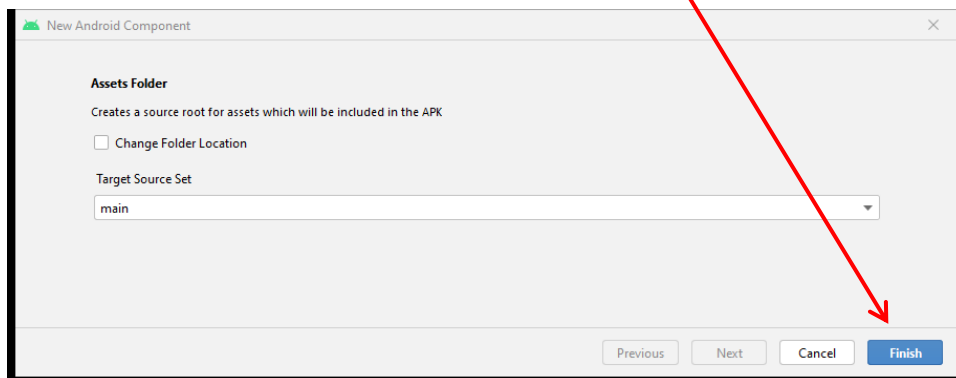
public class MainActivity extends AppCompatActivity {
    private WebView activity;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        activity = (WebView) findViewById(R.id.webView);
        activity.getSettings().setJavaScriptEnabled(true);
        activity.loadUrl("file:///android_asset/index.html");

        // Interacting with browser
        activity.setWebChromeClient(new WebChromeClient()); }
```

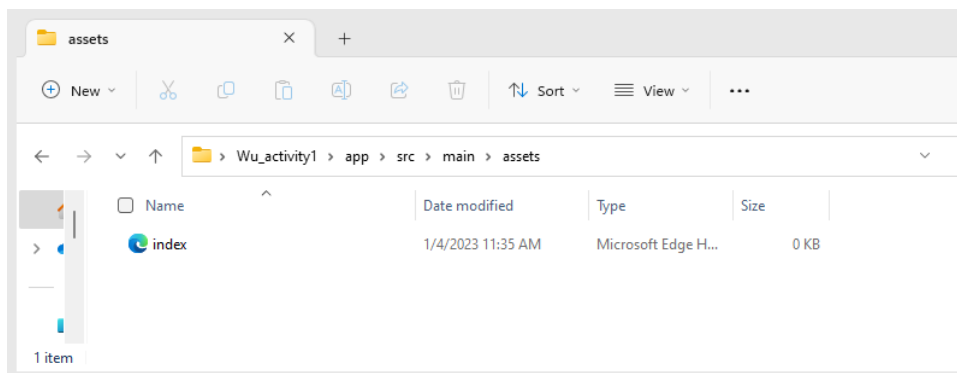
**Step 6) Creating assets folder.** Now, we are going to create an **assets** folder. The **assets** folder is used to store all HTML, CSS, JS files and support images of the web app. To create the **assets** folder, right click on the app icon  , select New → Folder → Assets Folder



On the New Android Component window, click **Finish**

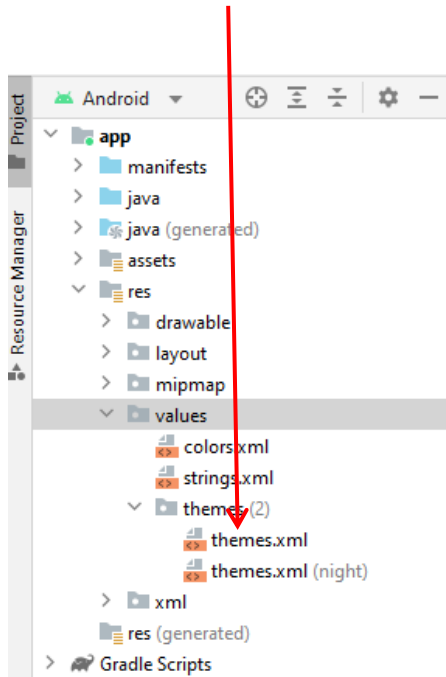


**Optional Step 7) Transferring web technology files to assets folder.** Once the assets folder is created, you can post your activity 1's html file to the assets folder. To find the assets folder, go to the **activity** folder → click on **app** folder → click on **src** folder → click on **main** folder → click on **assets** folder




## Step 8) Optional step


This step is used to modify the action bar, which is bar that appears on the top of the app. To do so, open the *themes.xml* file



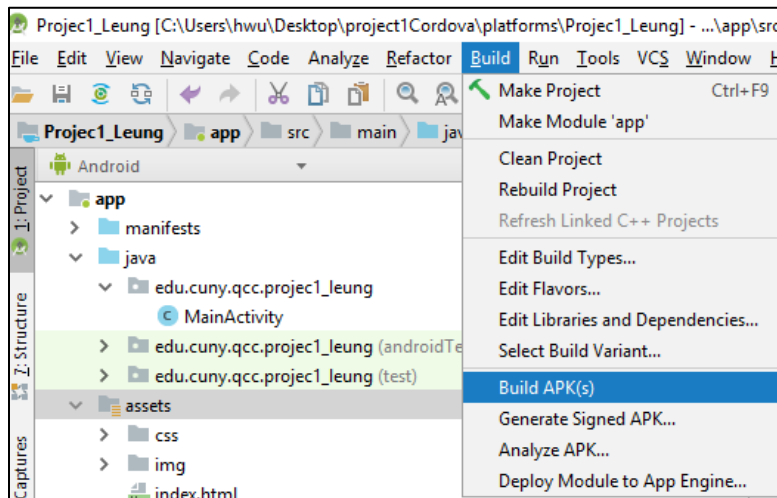
From the *styles.xml* file, we can modify the **parent** attribute from the `<style>` tag. From the parent, change **DarkActionBar** to **NoActionBar**



```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.Activity2Wu"
parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

**Step 9) Running the emulator.** To run the emulator and test that the web app is working properly,  click on the *Run* icon .

**Step 10) Creating an APK file.** Also, if you want to pack and build your web app into an APK file<sup>1</sup>, go to the menu bar, click on *Build* and select *Build APK(s)*



For more information on how to convert a web app to an android app, you can follow this video on YouTube →  
Converting a HTML+CSS+JS Code into an Android App using Android Studio!

<https://www.youtube.com/watch?v=CjiSaGWvsEU>

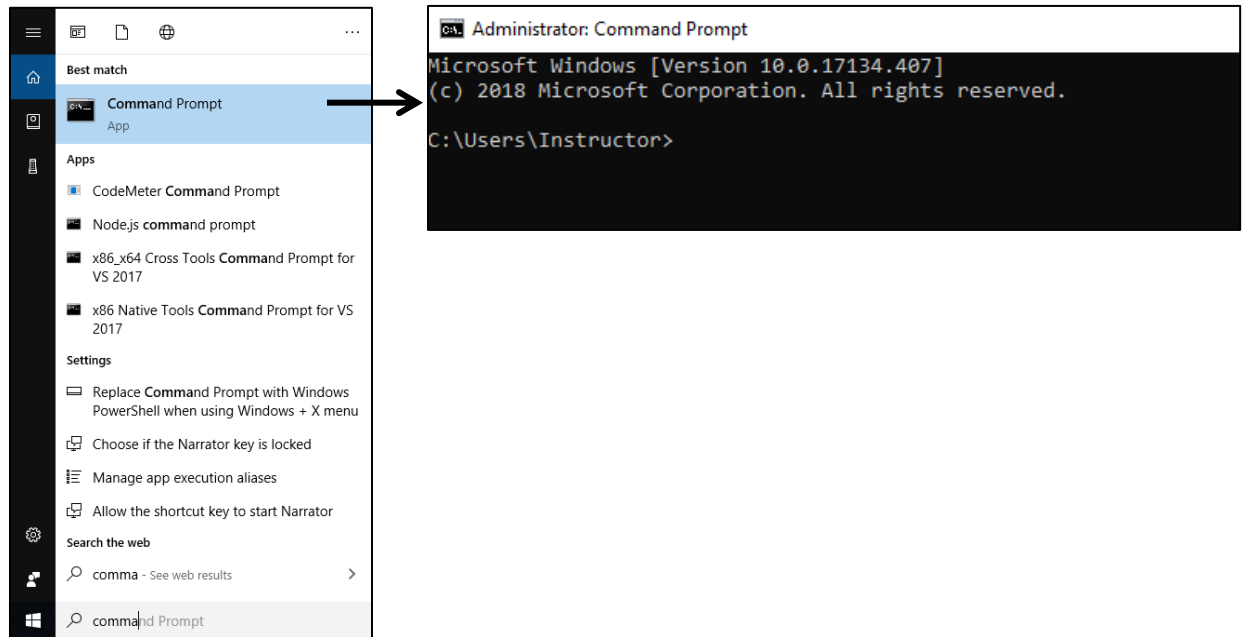
<sup>1</sup> **Android Package (APK)** is the package file format used by the **Android** operating system for distribution and installation of mobile apps and middleware.

## How to convert a web app into a native app using Cordova

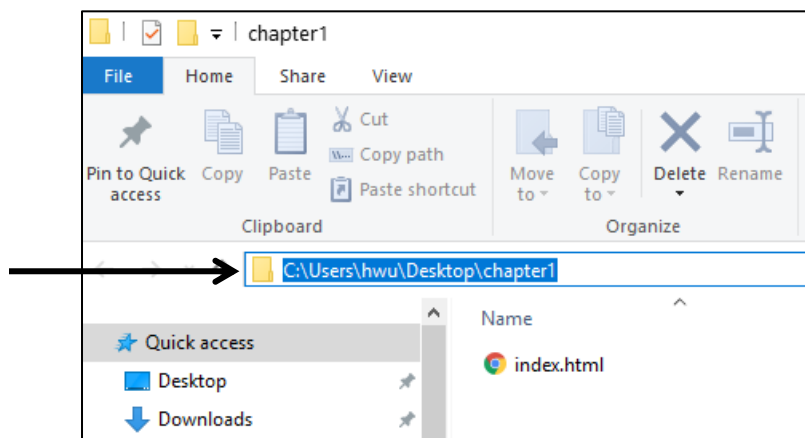
To convert a web app into a Cordova app, we have to make sure that the loading HTML file must be named *index.html*

Once you have Cordova installed and the web app ready to convert, we:

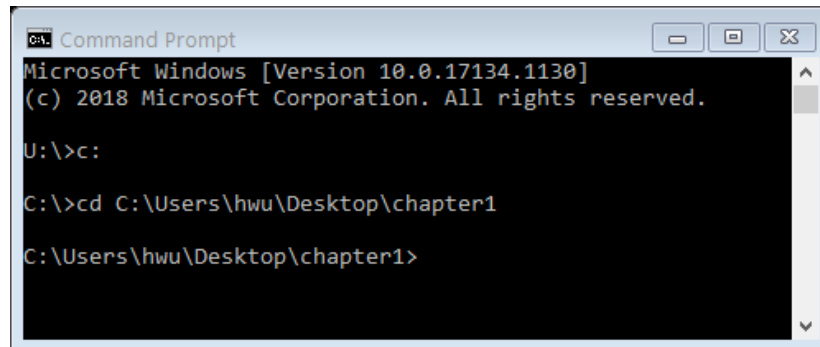
**Step 1)** Open the Command prompt



**Step 2)** At the command prompt, we can go to our project folder by using the *cd* (*current directory*) command. If we don't know our project folder location, we can open the project folder, click on the direction, and copy the location.



After it, we can go to the command prompt, type *cd* space, paste the location of the project folder, and click enter.



```
Command Prompt
Microsoft Windows [Version 10.0.17134.1130]
(c) 2018 Microsoft Corporation. All rights reserved.

U:\>c:

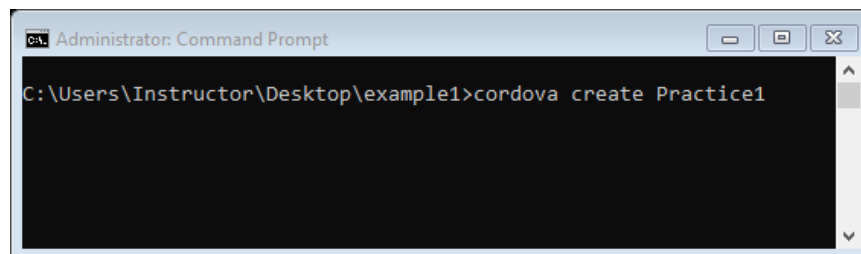
C:\>cd C:\Users\hwu\Desktop\chapter1

C:\Users\hwu\Desktop\chapter1>
```

**Step 3)** Once at the folder, we need to create a Cordova project folder by using the following command line:

Folder name

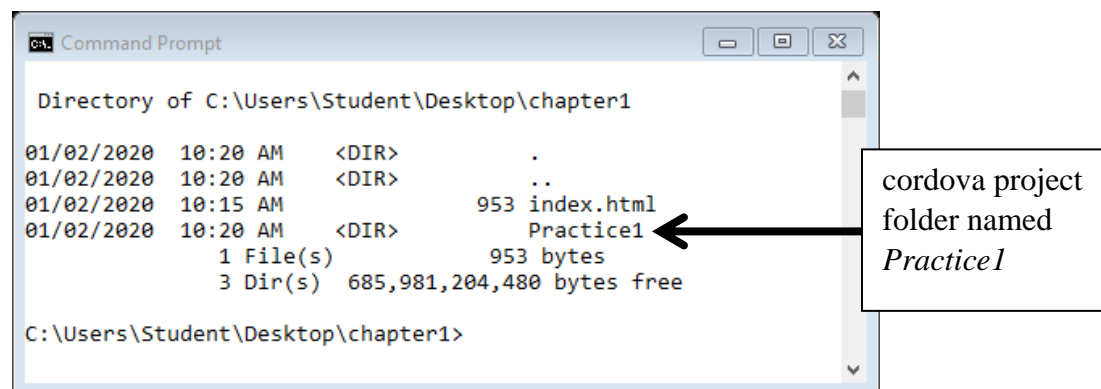
**cordova create Practice1**



```
Administrator: Command Prompt

C:\Users\Instructor\Desktop\example1>cordova create Practice1
```

We can check if the cordova project folder is created by typing *dir*



```
Command Prompt

Directory of C:\Users\Student\Desktop\chapter1

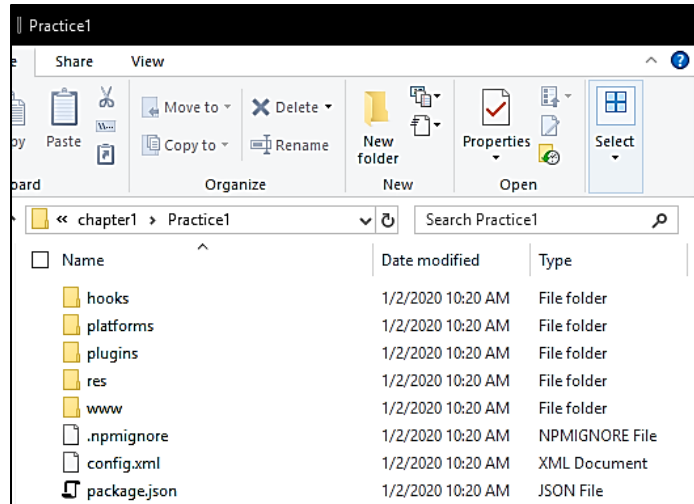
01/02/2020  10:20 AM    <DIR>          .
01/02/2020  10:20 AM    <DIR>          ..
01/02/2020  10:15 AM             953 index.html
01/02/2020  10:20 AM    <DIR>          Practice1
               1 File(s)              953 bytes
               3 Dir(s)  685,981,204,480 bytes free

C:\Users\Student\Desktop\chapter1>
```

cordova project folder named *Practice1*

If we open the project folder from the windows interface, we can also see that the cordova project folder is already created:





**Step 4)** We can copy all the project web app files and paste them into the *www* folder inside of the cordova project folder *Practice1*.

**Step 5)** Back to the command prompt, we go into the *Practice1* folder by using typing:

```

C:\Users\Student\Desktop\chapter1>cd Practice1
C:\Users\Student\Desktop\chapter1\Practice1>

```

***cd Practice1***

**Step 6)** Inside the *Practice1* folder, we are going to add an android or iOS platform to our Cordova project by typing:

```

C:\Users\Student\Desktop\chapter1\Practice1>cordova platform add android
Using cordova-fetch for cordova-android@8.0.0

```

***cordova platform add android***

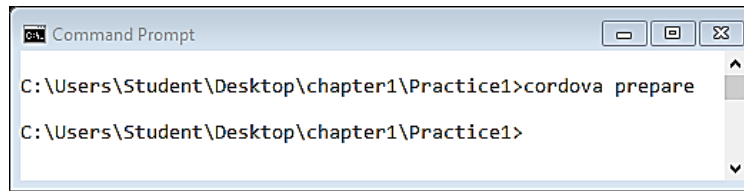
Once we clicked enter, cordova will run and create the platform.

```

Using cordova-fetch for cordova-android@8.0.0
Adding android project...
Creating Cordova project for the Android platform:
  Path: platforms\android
  Package: io.cordova.hellocordova
  Name: HelloCordova
  Activity: MainActivity
  Android target: android-28
Subproject Path: CordovaLib
Subproject Path: app
Android project created with cordova-android@8.1.0
Plugin 'cordova-plugin-whitelist' found in config.xml... Migrating it to package.json
Discovered saved plugin "cordova-plugin-whitelist". Adding it to the project
Installing "cordova-plugin-whitelist" for android
Adding cordova-plugin-whitelist to package.json
C:\Users\Student\Desktop\chapter1\Practice1>

```

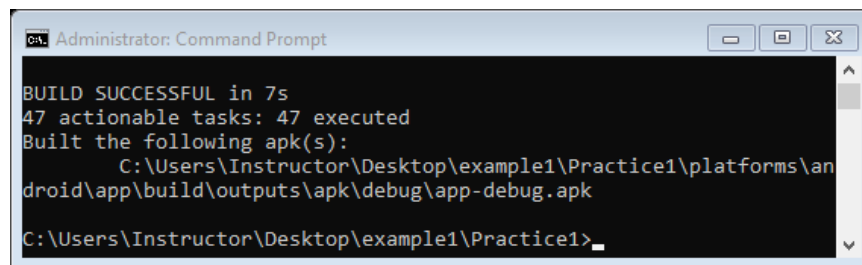
**Step 7)** Once the platform is created, we need to prepare and compile the cordova project Practice1. Type *cordova prepare*



```
Command Prompt
C:\Users\Student\Desktop\chapter1\Practice1>cordova prepare
C:\Users\Student\Desktop\chapter1\Practice1>
```

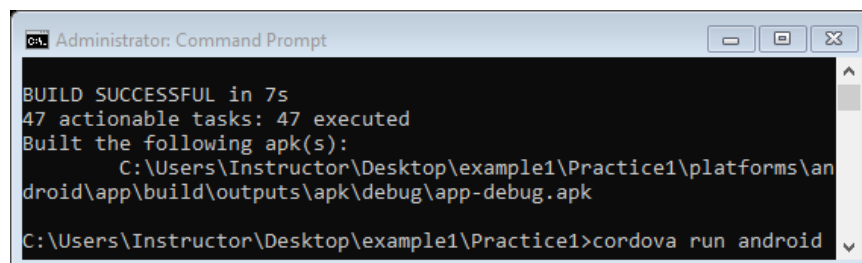
**Step 8)** Now we are going to compile the cordova project by typing: *cordova compile*

Once we clicked enter, cordova will run the compiling and at the end of the compiling, the command prompt will show BUILD SUCCESSFUL



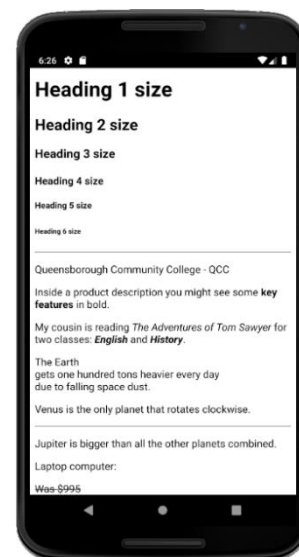
```
Administrator: Command Prompt
BUILD SUCCESSFUL in 7s
47 actionable tasks: 47 executed
Built the following apk(s):
  C:\Users\Instructor\Desktop\example1\Practice1\platforms\android\app\build\outputs\apk\debug\app-debug.apk
C:\Users\Instructor\Desktop\example1\Practice1>
```

**Step 9)** After compiling, we can run our project in an emulator by typing: *cordova run android*

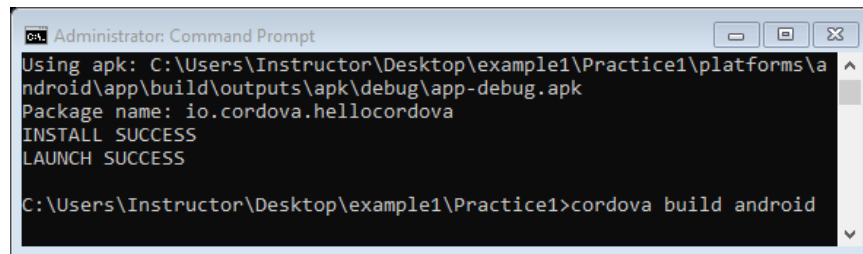


```
Administrator: Command Prompt
BUILD SUCCESSFUL in 7s
47 actionable tasks: 47 executed
Built the following apk(s):
  C:\Users\Instructor\Desktop\example1\Practice1\platforms\android\app\build\outputs\apk\debug\app-debug.apk
C:\Users\Instructor\Desktop\example1\Practice1>cordova run android
```

This command line will run the emulator from Android Studio



**Step 10)** We can also create an Android package .apk, by typing: **cordova build android**

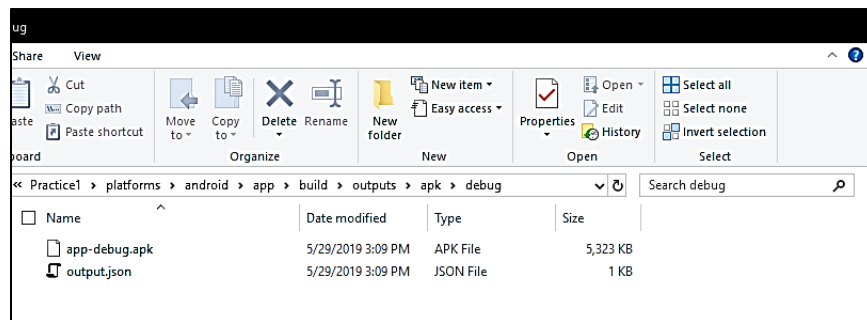


```
Administrator: Command Prompt
Using apk: C:\Users\Instructor\Desktop\example1\Practice1\platforms\android\app\build\outputs\apk\debug\app-debug.apk
Package name: io.cordova.hellocordova
INSTALL SUCCESS
LAUNCH SUCCESS

C:\Users\Instructor\Desktop\example1\Practice1>cordova build android
```

Once the .apk is build, we can find it at:

example1\Practice1\platforms\android\app\build\outputs\apk\debug



For more information about the cross-platform conversion using Cordova, you can check the following YouTube video ➔ Cordova: How to Create, Build, Add Platform and Run an html file into Android and iOS emulator.

<https://www.youtube.com/watch?v=cxJAiUeb5bQ>

