1. Push Notification

- A push notification is a message that pops up on a mobile device. App publishers can send them at any time; users don't have to be in the app or using their devices to receive them.

- Types of Push Notification

1. Informative notifications

- Update Notification, Reminder ,Alert

2. Geolocation notifications

- An app can send a simple push notification to users when they enter or leave a location.

3. Catch-up notifications to re-engage users

- catch-up or re-engagement notifications is to motivate users

4. Promotional notifications

- to alert users about exclusive offers, giveaways, or sales

5. Recurrent push notifications

- sent to users only once at a specific date and time, to highlight editorial picks, for instance

6. Rating/survey push notifications

- Push notifications with ratings or surveys are a great way to gather user feedback and improve the customer experience.

7. Order push notifications

- notifications alert customers about what's going on with their orders, like order confirmations, order status updates, and tracking information

8. Trivia push notifications

- to send users some love with funny, entertaining messages. These notifications are creative and aren't intrusive.
- example : BuzzFeed News App

Create Push Notitication in Xamarin Forms

1. for this sign in to Azure accoutn , select Notification Hub Project

- create Resource Group(for all resources related to one project)
- create NameSpace , Hub Name
- select tier for no of user

2. In Notification Hub, Setting Section

- for Apple

1. in Entitlement.plist of our iOS Project ,enable push Notification
2. in Apple Developer console register the app with identifier(com.companyname.app) in App source code. + so now we can enable push notification in settings of our project in console, + for this, we need dev and prod SSL certificate (using CSR file in mac by keychain Access)
3. in azure notification hub, apple setting + auth mode ,select as Certificate + add .p12 file as certificate and can set password as of .p12 file

- for Android
    1. in Notification hub , we need Api Key (from FCM)[FireBase Cloud Messaging service]
    2. in firebase.google.com
    - Create new Project here,
    - in Android section, register our app, using Package Name in Source code
    - so after registrering, we get json file , keep it.
    - now, in VS2019, add json file to Android project
    - now install following Nuget Packages:
        1. Xamarin.GooglePlaySerivces.Base
        2. Xamarin.FireBase.Messaging
        3. Xamarin.Azure.NotificationHubs.Android
    - now in Source code, the json file buid change to GoogleServices.json
    - in Firebase Project, settings, we have a Server Key, that we need to paste as APi key in Azxure notification hub

3. Notification Hub API

- How to send Notification from Hub for FCM and APN
    - Test Send-> Payload -> Send
    1. Api: are in docs.microsoft
    - for send Notification we need Authorization, which can be created using
    - Generate a Shared Access Signature TOken docs article

- here we create our Notification message to send to app user, using 4 key elements, given in docs

3. Set up in Android Project

- in Android manifest add code and permission to Application tag, given in docs.microsoft, send push notification to Xamarin

- override default functionality to recieve messages, using a class FireBaseService

- having attribute [Service] , [IntentFilter(new [] {"com.google.firebase.MESSAGING_EVENT"})]

- class inherit from FirebaseMessagingService

- override OnNewToken()

- add method SendRegisterationToAzure() from OnNewToken()

- override OnMessageRecieved()

- add Intent to show notification to user by creating one method like SendLocalNotification() iN FireBaseService class

4. Set up in iOS Project

- Add Nuget package Xamarin.Azure.NotificationHubs.iOS
- in AppDelegate class,

1. in FinishedLaunching method, Register for Remote Notification using a method, where we first check if we have Permission ,if yes, then we Register using UIApplication object method, remember to invoke it on main Thread.
2. override the RegisteredForRemoteNotifications method, and register with Azure Notification Hub in this method. also UnregisterALL previous services.
3. override RecievedRemoteNotification method, and check for Notification came from Azure
4. this is all we have to do on iOS Project, and check on iOS device, as notification not work on simulator.

2. Section Xaml and Advanced Xamarin Forms

1. Platform specific

- Android
- include namespace, AndroidSpecific, and change attrbutes as required for different pages, like tabbedpage here

```
          <TabbedPage
              xmlns:android="clr-namespace:Xamarin.Forms.PlatformConfiguration.AndroidSpecific;assembly=Xamarin.Forms.Core"
          android:TabbedPage.ToolbarPlacement="Bottom"
          >
          </TabbedPage>
```

- iOS +include namespace, iOSSpecific, and change attrbutes as required for different features, like here for SafeArea

```
<ContentPage
 xmlns:ios="clr-namespace:Xamarin.Forms.PlatformConfiguration.iOSSpecific;assembly=Xamarin.Forms.Core"
             ios:Page.UseSafeArea="True"
             >
             </ContentPage>
```

2. Custom TitleView

- i XAML Page we can set custom title using NavigationPage.TitleVIew tag

```
<NavigationPage.TitleView >
        <StackLayout Orientation="Horizontal"
                     VerticalOptions="Center"
                     Spacing="10" BackgroundColor="Black">
            <Image  Source="expense.png"  />
            <Label
                   VerticalOptions="Center"
                   Text="Finance"
                FontSize="Medium" TextColor="White"/>
        </StackLayout>
    </NavigationPage.TitleView>
```

3. OnPlatform at Attribute level

- to make changes to attribute based on platform app is running on.

```
<Label TextColor="{OnPlatform Android=White, iOS=Black, Default=Black}" />
```

- we can also use static resources as value in OnPlatform

```
<Label TextColor="{OnPlatform Android={StaticResource androidAccent}, iOS=
{StaticResource iOSAccent}, Default={StaticResource iOSAccent}}"/>
```

4. Using CSS style sheet

- first add stylesheet as application resource

```

 <Application.Resources>

    <!-- Application resource dictionary -->
        <StyleSheet Source="../StyleSheets/styles.css" />
    </Application.Resources>
```

- then we can use stylesheet in XAML using element name like stacklayout, or styleid(#id) or styleClass (.class)

```
listview{
    margin:20px;
}

^label{
    color:black;
}

.titleLabel{

    font-weight:bold;
    margin:20px,20px,20px,0;
    color:rebeccapurple;
}
```

5. Visual State Manager

- use this to make changes to your controls based on user action, or screen size
- for button in XAML we declare visualstate tag, and how it works is handled in code back file
- here we have three most common states of Normal,Disabled, and Focused.

```
 <Button  x:Name="stateButton" Pressed="stateButton_Pressed"
Released="stateButton_Released"
           IsEnabled="False">
            <VisualStateManager.VisualStateGroups>
                <VisualStateGroup Name="CommonStates">
                    <VisualState Name="Normal">
                        <VisualState.Setters>
                            <Setter Property="BackgroundColor" Value="Blue" />
                        </VisualState.Setters>
                    </VisualState>
                    <VisualState Name="Disabled">
                        <VisualState.Setters>
                            <Setter Property="BackgroundColor" Value="Transparent"
/>
                        </VisualState.Setters>
                    </VisualState>
                    <VisualState Name="Focused">
                        <VisualState.Setters>
                            <Setter Property="BackgroundColor" Value="Green" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </Button>
```

```
   private void stateButton_Pressed(object sender, System.EventArgs e)
        {
            VisualStateManager.GoToState(stateButton, "Focused");
        }

        private void stateButton_Released(object sender, System.EventArgs e)
        {
            VisualStateManager.GoToState(stateButton, "Normal");
        }

```

- similarly, for size changed state, we can set property

```
   <Label x:Name="catLabel" Text="Categories" StyleClass="titleLabel"
HorizontalTextAlignment="Center" FontSize="30">
            <VisualStateManager.VisualStateGroups>

                <VisualStateGroup Name="OrientationStates">
                    <VisualState Name="Landscape">
                        <VisualState.Setters>
                            <Setter Property="IsVisible" Value="false" />

                        </VisualState.Setters>
                    </VisualState>

                    <VisualState Name="Portrait" >
                        <VisualState.Setters>
                            <Setter Property="IsVisible" Value="true" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </Label>
```

```
// here SizeChanged is a event of contentPage, same with Width and Height
 private void CategoriesPage_SizeChanged(object sender, System.EventArgs e)
        {
            string visualState = Width > Height ? "Landscape" : "Portrait";
            VisualStateManager.GoToState( catLabel, visualState);

        }

```

6. Image Button

- used to set iamge as a button

```
 <ImageButton Source="moneys.png"   Clicked="ImageButton_Clicked"
                  >
            <VisualStateManager.VisualStateGroups>
                <VisualStateGroup Name="CommonStates">
                    <VisualState Name="Normal">
                        <VisualState.Setters>
                            <Setter Property="Scale" Value="1" />
                        </VisualState.Setters>
                    </VisualState>

                    <VisualState Name="Pressed">
                        <VisualState.Setters>
                            <Setter Property="Scale" Value="0.8" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </ImageButton>
```

7. Improving ListVIew Performance

- we can do this, by changing CachingStratergy, from RetainElement to Recycle Element, so list data is not kept in memory and data in refetched and assigned.
    - use RatianElement in case of 20-30 element in list
    - for more than that,use RecycleElement

```

<ListView   CachingStrategy="RetainElement" />

<ListView  CachingStrategy="RecycleElement" />
<ListView   CachingStrategy="RecycleElementAndDataTemplate" />
```

3. Bot Service

- for creating the bot service , log in to azure console

1. Create a QnA Maker service

- add all information, with tier to use, and resource group for service

2. Log in to https://www.qnamaker.ai/ with Azure credentials

- here we have our QnA service project, here we have to add the Knowledge base, which is document or url from which we get the Question and Answer
- here now we can use Test option, to check if QnA is working or not
- Select Create a Bot option ,in dashboard of QnaMaker site
- now after successful creation of bot, we are taken to azure dashboard

3. now in azure, we are redirected to create another service that is Web App bot, which is already configured for our QnA project,

- we need to fill rest and create the bot

4. now we can directly use the Endpoint url with the Secret key to start using the bot service in WebView Control of our App, secret key is in Web app proejct.

5. now if we want to create a custom bot UI, we need to create an Activity, for recieving and sending messages., using a helper class botServerHelper, which set a connection and listen for user input and provide bot responses to View.

4. Using Native Libraries in Xamarin.Forms Project

1. Using java Library in Xamarin.Forms Project

- in Android Studio, when java library is built we get .aar file
- add this arr file to jar folder in Android Binding Library Project.
- build this Library Project, so we get .dll file to reference in our XF project
- some libraries are old so, check min sdk version, and change it if error occurs
- now we can access all functionality in library, can demonstrate it using a dependency service, accessing functionality from android project to shared.

Linking for APK sizes

- here in Android -> Build options, for linking we have 3 options

1. Sdk Assemblies only + APk size is very small compared to None option,(12.8 Mb)

2. Sdk and User Assemblies + + APk size is very small compared to None option,(12.7 Mb) + in this case, if linker is not preserving some code and we manually want to tell it to do that, we can use [Android.Runtime.Preserve] attribute upon the block of code to preserve.

3. None
   + here Apk size is largest(60 Mb),we can use this option to check if linker error is there, if error still persist , then its not related to linker. + not recommended to release to user, only troubleshooting.

4. Using Objective C library in Xamarin.Forms Project

- in XCode , Objective C library is build , we get .a file, different for simulator ,iphone, and xcode, combine it to get final .a file.

- Create a iOS Binding Library Project,now add this .a file in NativeLibrary section of this project

- now here we get a ApiDefinition.cs , in this file we have to map all the functionality in Objective C to C#

- now build the project , we get our .dll file , which we can refer in our XF iOS project

- now to get the functioanlity from dll, we can use a dependency service.

linking for IPA size

- 1. Link Framework Sdk only
     - APk size is very small compared to None option,(12.8 Mb)
  2. Link All
     - - APk size is very small compared to None option,(12.7 Mb)
     - in this case, if linker is not preserving some code and we manually want to tell it to do that, we can use [Preserve] attribute upon the block of code to preserve.
  3. Dont Link

- here Apk size is largest(80 Mb),we can use this option to check if linker error is there, if error still persist , then its not related to linker.
- not recommended to release to user, only troubleshooting.

5. Localization

1. Locatization og C# data for showing app UI data in different language to different user, we need to set up a Resource folder in which

- Resource.rsx file , containing a key value pair , where key is what data is and value is its name in default language .
- now similarly for all other languages, we need to create, a xml file using the format, Resource.ln.rsx, Where ln is acronym for the language,
- example: for spanish, it will be Resource.es.rsx.
- now in this xml file, we need to add key value pairs , with key being the same in first file, and value is name in this language(i.e spanish here)
- now we can include the namespace and assign the data in c# file, using format:
- Resource.data : here i used, AppResources.housingCategory

2. here we can set the default language , to be used by our app in app.xaml.cs, after InitializeComponent() method

- using the code

> AppResources.Culture = new System.Globalization.CultureInfo("en-US");

- here we can set acronym for language, which we want to set as default.

3. For Locatization of data in Xaml

- we need to foolow all steps in Location for C# data, and here
- create a custom markup extension to translate the data from .rsx file to our View, using a content property, can be string set by [ContentProperty("Text")] , here string property is Text. and the class extends from IMarkupExtension interface.
- and after that include the namespace of this custom extension and use this extension to set the data.

6. Platform Specific

For Android

- Platform-specifics allow you to consume functionality that's only available on a specific platform, without implementing custom renderers or effects.

1. The following platform-specific functionality is provided for Xamarin.Forms views, pages, and layouts on Android:

- Using the default padding and shadow values of Android buttons
- Setting the input method editor options for the soft keyboard for an Entry.
- Enabling a drop shadow on a ImageButton
- Enabling fast scrolling in a ListView.
- Controlling the transition that's used when opening a SwipeView.

- Controlling whether a WebView can display mixed content.
- Enabling zoom on a WebView.

- for pages

- Setting the height of the navigation bar on a NavigationPage.
- Disabling transition animations when navigating through pages in a TabbedPage.
- Enabling swiping between pages in a TabbedPage.
- Setting the toolbar placement and color on a TabbedPage

For iOS

- The following platform-specific functionality is provided for Xamarin.Forms views, pages, and layouts on iOS

1. Blur support for any VisualElement
2. Enabling a drop shadow on a VisualElement.
3. Enabling a VisualElement object to become the first responder to touch events.

- for views

4. Setting the Cell background color
5. Controlling when item selection occurs in a DatePicker
6. Ensuring that inputted text fits into an Entry by adjusting the font size.
7. etting the cursor color in a Entry.
8. Controlling whether ListView header cells float during scrolling.
9. Controlling whether row animations are disabled when the ListView items collection is being updated
10. Setting the separator style on a ListView
11. Controlling when item selection occurs in a Picker.
12. Controlling whether a SearchBar has a background
13. Enabling the Slider.Value property to be set by tapping on a position on the Slider bar, rather than by having to drag the Slider thumb
14. Controlling the transition that's used when opening a SwipeView.
15. Controlling when item selection occurs in a TimePicker.

- for pages

16. Controlling whether the detail page of a FlyoutPage has shadow applied to it, when revealing the flyout page.
17. Hiding the navigation bar separator on a NavigationPage.
18. Controlling whether the navigation bar is translucent
19. Controlling whether the status bar text color on a NavigationPage is adjusted to match the luminosity of the navigation bar
20. Controlling whether the page title is displayed as a large title in the page navigation bar
21. Setting the visibility of the home indicator on a Page
22. Setting the status bar visibility on a Page.
23. Ensuring that page content is positioned on an area of the screen that is safe for all iOS devices
24. Setting the presentation style of modal pages.
25. Setting the translucency mode of the tab bar on a TabbedPage. |

- for layout

26. Controlling whether a ScrollView handles a touch gesture or passes it to its content.

- for Application class

27. Disabling accessibility scaling for named font sizes.
28. Enabling control layout and rendering updates to be performed on the main thread.
29. Enabling a PanGestureRecognizer in a scrolling view to capture and share the pan gesture with the scrolling view.

- way to set iOS-specific formatting

1. create a custom renderer for control
2. Configuring display options in Info.plist
3. Setting control styles via the UIAppearance API

change Tab Bar image size

- using a custom renderer.

```
[assembly: ExportRenderer(typeof(TabbedPage), typeof(CustomTabbedPageRenderer))]
namespace TabBarIconSize.Droid
{
    class CustomTabbedPageRenderer : TabbedPageRenderer
    {
        public CustomTabbedPageRenderer(Context context)
            : base(context)
        {

        }

        protected override void
OnElementChanged(ElementChangedEventArgs<TabbedPage> e)
        {
            base.OnElementChanged(e);

            // Get TabbedPage's bottomNavigationView
            var bottomNavigationView = (GetChildAt(0) as
Android.Widget.RelativeLayout).GetChildAt(1) as BottomNavigationView;

            // Set Icon size
            bottomNavigationView.ItemIconSize = 50;
        }
}
```

listview Caching stratergy

1. The RetainElement

- caching strategy specifies that the ListView will generate a cell for each item in the list, and is the default ListView behavior
- Each cell has a large number of bindings

2. The RecycleElement

- caching strategy specifies that the ListView will attempt to minimize its memory footprint and execution speed by recycling list cells.
- Each cell has a small to moderate number of bindings

3. RecycleElement with a DataTemplateSelector

- When a ListView uses a DataTemplateSelector to select a DataTemplate, the RecycleElement caching strategy does not cache DataTemplates. Instead, a DataTemplate is selected for each item of data in the list.

4. RecycleElementAndDataTemplate

- The RecycleElementAndDataTemplate caching strategy builds on the RecycleElement caching strategy by additionally ensuring that when a ListView uses a DataTemplateSelector to select a DataTemplate, DataTemplates are cached by the type of item in the list. Therefore, DataTemplates are selected once per item type, instead of once per item instance.

Add stylesheet in Application resources

```
<Application.Resources>

    <!-- Application resource dictionary -->
        <StyleSheet Source="../StyleSheets/styles.css" />
    </Application.Resources>
```

We can add Visual State Manager to Style

```
<ContentPage.Resources>
        <!--<StyleSheetExtension Source="/StyleSheets/styles.css" />-->
        <ResourceDictionary>
            <Style TargetType="Label" x:Key="titleLabel"
ApplyToDerivedTypes="False" >
                <Setter Property="VisualStateManager.VisualStateGroups">
                    <VisualStateGroupList x:Name="Common">

                        <VisualStateGroup Name="OrientationStates">
                            <VisualState Name="Landscape">
                                <VisualState.Setters>
                                    <Setter Property="IsVisible" Value="false" />

                                </VisualState.Setters>
                            </VisualState>

                            <VisualState Name="Portrait" >
                                <VisualState.Setters>
                                    <Setter Property="IsVisible" Value="true" />
                                </VisualState.Setters>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateGroupList>
                </Setter>
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>
```

OnPlatform Default type means

- Default Action
- (optional) The Action to execute if no Action was provided for the current OS.

DEX(Dalvik Executable) Compiler D8 and Dx

1. dex compilation is a key step in building an APK.
2. DX and D8 are DEX compiler.
3. This is the process of transforming .class bytecode into .dex bytecode for the Android Runtime.
4. it directly impacts your app's build time, .dex file size, and runtime performance.
5. Comparison D8 and Dx

- comparing with the current DX compiler, D8 compiles faster and outputs smaller .dex files, while having the same or better app runtime performance.

code shrinker: r8 vs proguard

- used to converts our java byte code into an optimized dex code.

1. Proguard, We use it for reducing the size, improving the performance of the application by shrinking unused resources.

2. Google released R8 as a replacement of Proguard to help developers shrink the code with the better-generated output (APK). They are considered much faster compared to Proguard.

3. R8 is a tool that converts our java byte code into an optimized dex code. It iterates through the whole application and then it optimizes like removing unused classes, methods, etc. It runs on the compile time. It helps us to reduce the size of the build and to make our app to be more secure. R8 uses Proguard rules to modify its default behavior.

- To reduce the APK size, we have three different techniques:

1. Shrinking or Tree Shaking:

- Shrinking is the process of removal of unreachable code from our Android project. R8 performs some static analysis to get rid of unreachable code and removes the uninstantiated object.

2. Optimization:

- This is used to optimize the code for size. It involves dead code removal, unused argument removal, selective inlining, class merging, etc.

3. Identifier Renaming:

- In this process, we obfuscate the class name and other variable names. For example, if the name of the class is "MainActivity", then it will be obfuscated to "a" or something else but smaller in size.