UNIT I-web design

UNIT I

Basics of Web and Mobile Application Development, Native App, Hybrid App, Cross-Platform App, What is Progressive Web App, Responsive Web Design

1.1. What is a web application?

A web application (web app) is an application program that is stored on a remote server and delivered over the internet through a browser interface.

Developers design web applications for a wide variety of uses and users, from an organization to an individual for numerous reasons. Commonly used web applications can include webmail, online calculators or e-commerce shops. While users can only access some web apps by a specific browser, most are available no matter the browser.

1.2 How web applications work

Web applications work by utilizing a client-server model, where the client is typically a web browser or a mobile app, and the server is a computer system that hosts the application and handles data processing and storage.

Here's a general overview of how web applications function:

1. Client Sends a Request: When a user interacts with a web application by entering a URL or clicking on a link, the client (web browser) sends an HTTP request to the server. This request contains information about the action the user wants to perform, such as loading a specific page or submitting a form.

2. Server Processes the Request: The server receives the HTTP request and processes it based on the application's logic. This can involve querying databases, retrieving files, performing calculations, or any other necessary operations to generate the appropriate response.

3. Application Logic and Data Handling: The web application's backend, also known as the server-side, handles the application's logic and data. It interprets the request, accesses databases or other external services if needed, and processes the data to generate the response.

4. Server Sends a Response: Once the server has processed the request, it generates an HTTP response containing the requested information or actions. This response is typically in the form of HTML, CSS, JavaScript, and other assets required to render the web page on the client-side.

5. Client Receives and Renders the Response: The client (web browser) receives the HTTP response from the server. It interprets the HTML, renders the web page structure, applies CSS styles for presentation, and executes any JavaScript code for dynamic interactivity.

6. User Interacts with the Web Page: The user interacts with the web page through the client (browser). The user may click on buttons, fill out forms, or perform other actions that trigger additional HTTP requests to the server.

7. Repeat the Process: Steps 1 to 6 continue as the user interacts with the web application. Each user action generates new HTTP requests to the server, and the server processes these requests to produce relevant responses.

8. State Management: Web applications may need to maintain the state of user sessions, especially when users log in or perform multi-step processes. Session cookies, local storage, or server-side session management techniques are used to store and maintain user-specific data.

Web applications can be built using various technologies and frameworks, such as JavaScript for client-side interactivity, backend programming languages like Python, Ruby, Java, or Node.js for server-side processing, and databases for data storage. Popular frameworks like React, Angular, Vue.js, Django, Ruby on Rails, etc., facilitate the development of web applications by providing reusable components and tools for managing server-client interactions.

Web applications have a client-server architecture. Their code is divided into two components—client-side scripts and server-side scripts.

- √ **Client-side architecture**

The client-side script deals with user interface functionality like buttons and drop-down boxes. When the end user clicks on the web app link, the web browser loads the client-side script and renders the graphic elements and text for user interaction. For example, the user can read content, watch videos, or fill out details on a contact form. Actions like clicking the submit button go to the server as a client request.

- √ **Server-side architecture**

The server-side script deals with data processing. The web application server processes the client requests and sends back a response. The requests are usually for more data or to edit or save new data. For example, if the user clicks on the *Read More* button, the web application server will send content back to the user. If the user clicks the *Submit* button, the application server will save the user data in the database. In some cases, the server completes the data request and sends the complete HTML page back to the client. This is called server side rendering.

1.3 Different Types Of Web Applications

As the digital world continues to expand, web applications have become an essential tool for businesses and individuals alike. Web applications are designed to meet specific needs and requirements of the users, and choosing the right type of web application is crucial to achieve your desired goals.

1.3.1. Static Web Apps

Static web apps consist of restricted content and have no flexibility. Here, these apps are considered the pages generated by the server with very little or no interactivity.

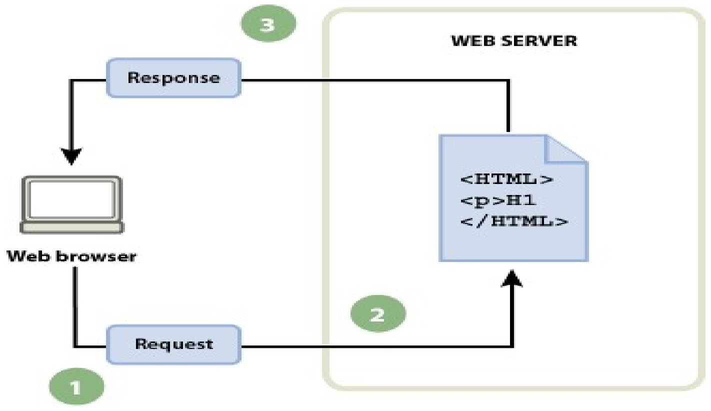


Fig 4.1 Static Web application Access

In short, static web apps appear to the clients the same as they are stored on the server. No content changes are made in the application server before the page is sent to a web browser.Languages used to create static web applications include HTML, CSS, JavaScript, etc. Even though one can integrate animated objects, GIFs, videos, and more in static web apps; however, it is hard to make updates.

Professional portfolios, digital resumes, landing pages for marketing, etc., are the best examples of static web applications.

Pros of Static Web Application

- √ Easy to host
- √ Quick to build
- √ Development cost is low
- √ Easy to index in the search engines
- √ Very fast to render on a slow internet connection.

1.3.2. Dynamic Web Apps

It is one of the best web application types as they fetch data in real-time based on the users' requests. They have enhanced technical complexity as compared to static web apps.

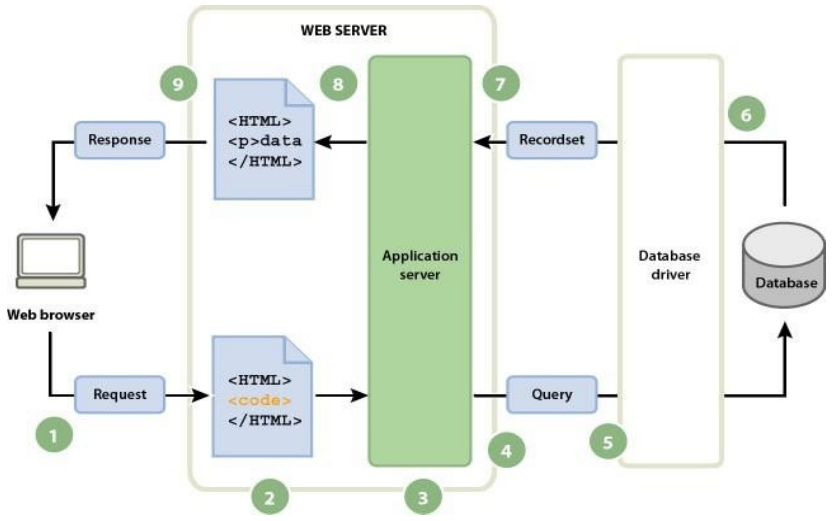


Fig 4.2 How a Dynamic Web App Works

Whenever a web server gets a request for a specific page, the page request reaches the software known as an application server. Databases are located on the server-side; hence the user gets updated content.

The application server understands the code, fetches the page from the database & sends the answer to the web server, which in turn sends it back to the web browser. In short, a dynamic web app requires a database to store data, and its content is continuously updated every time users access it. It is possible to achieve this using a content management system like WordPress, which has a built-in administration panel.

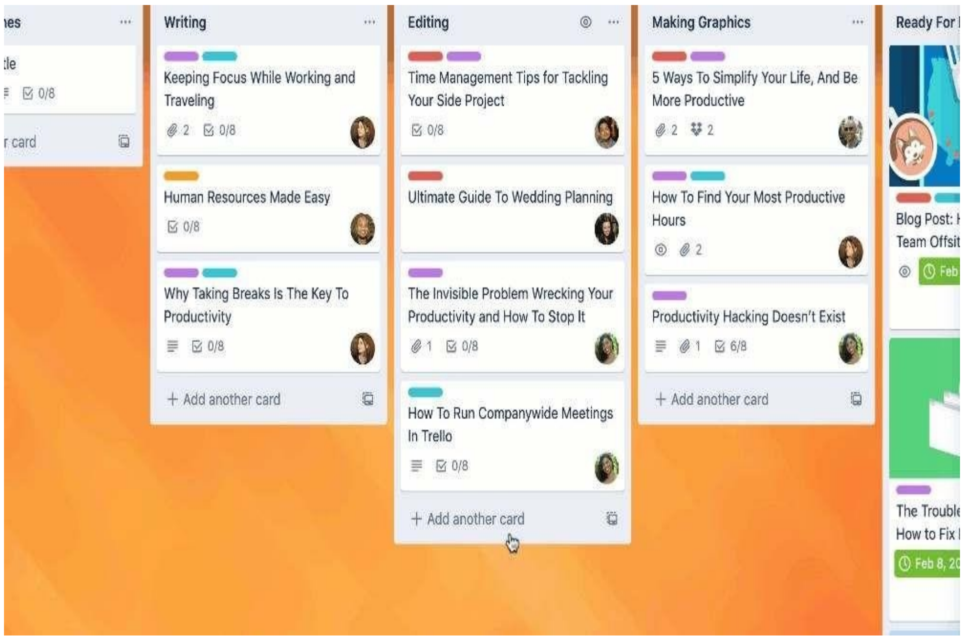
Pros of Dynamic Web Apps

- ✓ Better user experience
- ✓ Easy to update & maintain
- ✓ Highly interactive
- ✓ Quick Navigation
- ✓ Professional look

1.3.3. Single-Page Applications

Single-page apps enable users to interact with the web page without any hindrance. Here, requests and responses take place efficiently due to small amounts of data.

Briefly, SPAs are way quicker as compared to traditional web applications as they perform logic on the web browser instead of the server.


Writing
Keeping Focus While Working and Traveling
Human Resources Made Easy
Why Taking Breaks Is The Key To Productivity
Add another card
Editing
Time Management Tips for Tackling Your Side Project
Ultimate Guide To Wedding Planning
The Invisible Problem Wrecking Your Productivity and How To Stop It
How To Run Companywide Meetings In Trello
Add another card
Making Graphics
5 Ways To Simplify Your Life, And Be More Productive
How To Find Your Most Productive Hours
Productivity Hacking Doesn't Exist
Add another card

Moreover, you can update any single-page app as per the requirements in the future. Yet, due to universal URLs, they are not competent as per the latest SEO rules.

Single Page Application Advantages

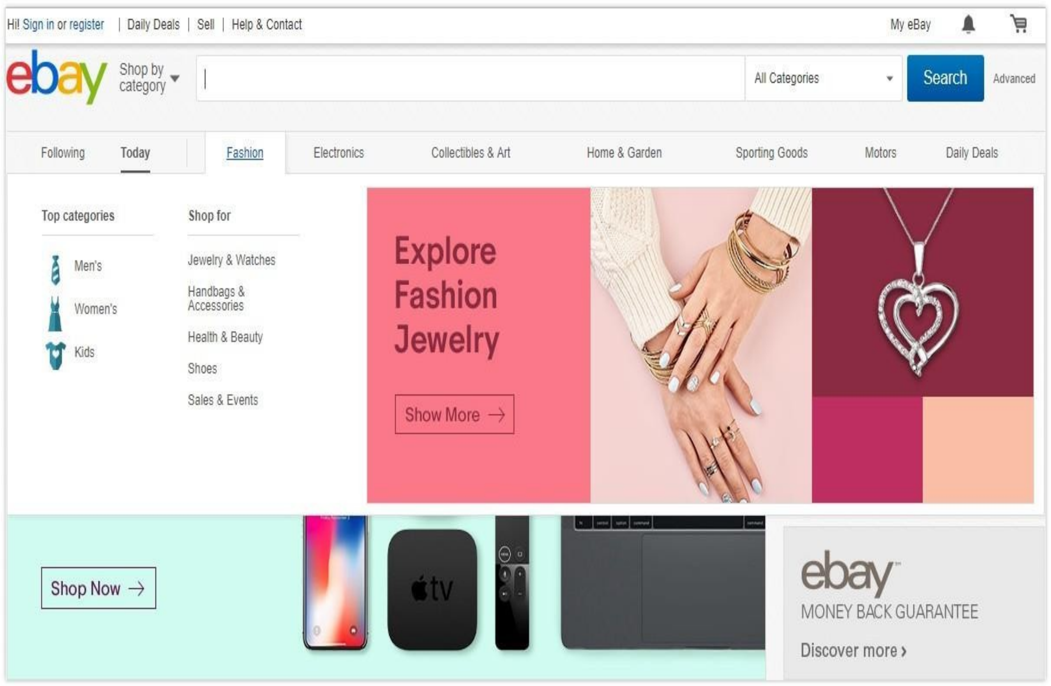
- √ Battery reusability
- √ Easy debugging
- √ Less complex implementation
- √ Better caching
- √ User-friendly

Some of Popular Single Page Apps are Gmail, Google Maps, AirBNB, Netflix, Pinterest, Paypal

1.3.4. Multi-Page Applications

Multi-page apps function similarly to traditional web applications. Here, the app reloads and displays a new page from the server in the browser anytime users perform a new action.In these types of web applications, logic is stored at the backend; hence requests from the clients go back to the server & are reverted.

Process of generating pages on the server, sending them to the client, and presenting them on the browser harms the user interface.It is possible to resolve this by utilizing AJAX technology, which makes sudden changes without a complete page reload. If MPAs are designed considering responsiveness, they can blend well with the mobile environment.



MPAs can be built using different languages such as HTML, CSS, JavaScript, AJAX, jQuery, etc. Web portals, Online Stores, Catalogs, Marketplaces, Enterprise Web Applications, etc., fall under the multi-page applications.

Pros of Multi-Page Applications

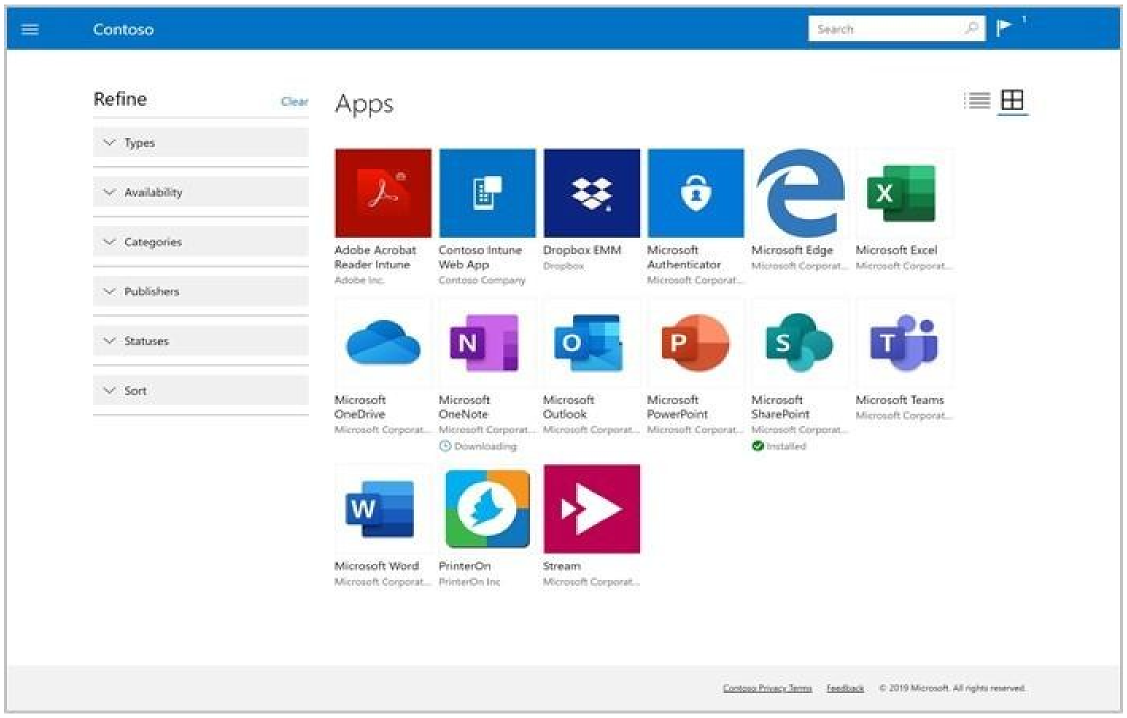
- √ More SEO-friendly
- √ Unlimited scalability
- √ Add unlimited pages in your existing app

Popular Multi-Page Applications

- √ Amazon
- √ Forbes

- √ CNN
- √ eBay

1.3.5. Portal Web Apps



It is one of its types of web apps in which various sections or categories are accessible on the home page. Here, this page consists of various details such as chats, emails, forums, user registration, etc.

Portals are best suited for businesses and enterprises that want to create customized interfaces according to their target audience's requirements.Here, all the registered users can only access the portal. Whenever a user signs in, the service provider can check the user's activity.According to the access allocated, specific features might be limited to particular users.

Government Portals, Student Portals, Real Estate Portals, Education University Portals, Patient Portals, etc., fall under the portal web apps.

Pros of Portal Web Apps

- √ Provides enhanced interaction

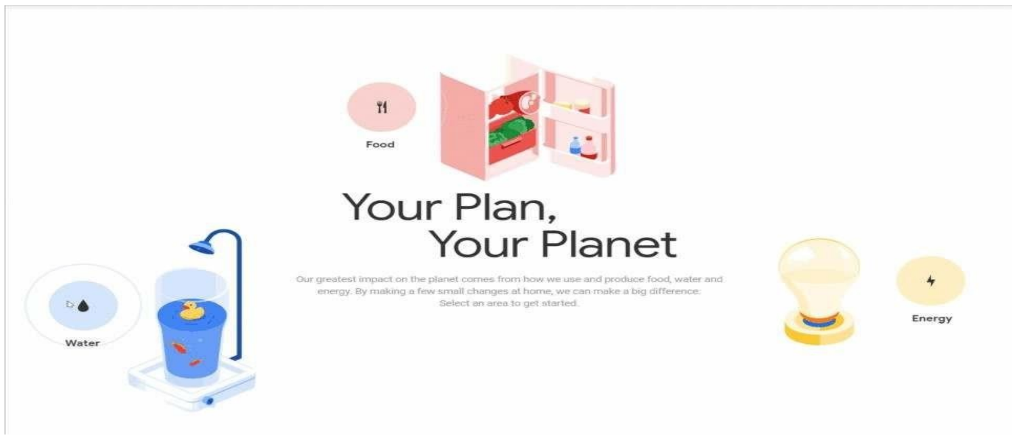
- √ Better integration
- √ Omnichannel presence
- √ Better customer retention

Examples of Portal Web Apps

- √ Domino's Pizza
- √ Stanford University
- √ Allianz

1.3.6. Animated Web Applications

Animated web applications are closely connected with FLASH technology. By creating these types of web apps, content can be represented by using various animated effects.Here, UI/UX designers have the freedom to become highly creative and integrate things that are not possible in different types of web-based applications.



Our greatest impact on the planet comes from how we use and produce food, water and energy. By making a few small changes at home, we can make a big difference. Select an area to get started.

The best practice is only to include animations that gain the user's attention & deliver valuable information. Also, this allows you to improve the user experience.The major drawback of creating animated web applications is that they are improper for web positioning and SEO as search engines can't fetch data from them.

HTML5, CSS3, JS, SVG, etc. are useful to create animated web applications.

1.3.7. Rich Internet Apps

Rich Internet Applications are mainly web applications having several functionalities of desktop applications. However, they are different from desktop applications. RIAs are introduced to resolve browser restrictions, and they depend on client-side plugins such as Flash, Shockwave, and Silverlight.

As these applications are built using these tools, they run efficiently and are very engaging. Moreover, they provide an eye-catching user experience and high interactivity as compared to traditional browser applications.

The two main problems with the RIA's are vulnerabilities and inconveniences they form. For instance, suppose the plugin is outdated, then several parts of the app or the entire app will not work accurately.

Benefits of Rich Internet Apps

- √ Improved data visualization
- √ Integration across various systems
- √ Control item synchronization
- √ Quick mobile access to information

Examples of Rich Internet Apps

- √ Adobe Flash
- √ Google Gears
- √ Microsoft Silverlight

1.3.8. JavaScript Powered Web Apps

After the availability of leading front-end frameworks, such as Angular, React.js, Vue.js, Node.js, the web-app logic has begun to move towards the client-side, providing higher adaptability as compared to the AJAX. Currently, client-wide logic can perform server-side tasks such as handling user requests and giving the responses, etc. Web Applications built using the above JS frameworks offer enhanced performance, various levels of user interaction and are usually SEO optimized. Client-Portals, Business-Centric Web Applications, etc., fall under the JavaScript-powered web apps.

Pros of JavaScript Powered Web Apps

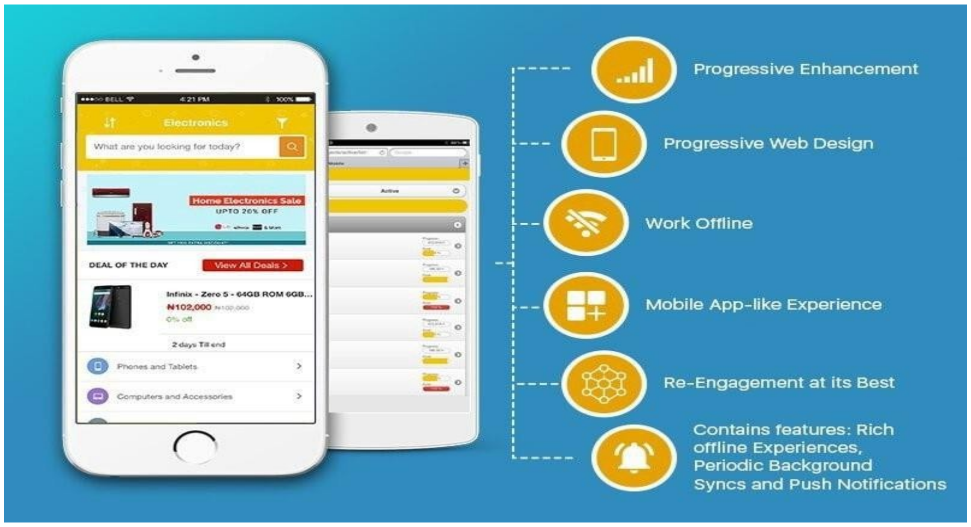
- √ Highly-Interactive
- √ Fast & Responsive
- √ Offline support
- √ Quick integration

Examples of JavaScript Powered Web Apps

- √ Yahoo
- √ Uber
- √ Linkedin
- √ Netflix
- √ Mozilla

1.3.9. Progressive Web Apps

PWAs are websites that look similar to mobile applications. Users can access the complete information and all the features of the web app using mobile browsers. Various experts say PWA is a modified version of SPA. Even though it's not valid based on the theory; however, the point is authentic in real life.


Progressive Enhancement
Progressive Web Design
Work Offline
Mobile App-like Experience
Re-Engagement at its Best
Contains features: Rich offline Experiences, Periodic Background Syncs and Push Notifications

The primary purpose of PWAs is not to apply new rules in architecture but enhance the speed and mobile adaptability of web apps. Here, improvements are made in caching, data transfer, and home screen installation.

Moreover, PWAs enable you to enhance the mobile web experience and provide your services to users despite slow/bad internet connections. Starbucks, Forbes, OLX, MakeMyTrip, etc., are some of the best examples of PWA.

Benefits of Progressive Web Apps

- √ Independent of app stores
- √ Offline support
- √ Enhanced performance
- √ No Installation & updates Required
- √ Access platform specific features

Examples of Progressive Web Apps

- √ Pinterest
- √ Twitter Lite
- √ Spotify
- √ MakeMyTrip

Difference between PWA and Native Application

Feature	Progressive Web Application	Native Application
Function offline	Yes	Yes
Installation requirement	There is no need to install it in mobile.	It is necessary to install it in the phone.
Push-notification.	It supports the push-notification feature.	It also supports the push-notification feature.
Platform	It supports the cross-platform.	It supports the specific platform. Example iOS, Android, and Windows

Data consumption	Low data consumption	High data consumption
Internet requirement	No internet requirement	Internet requirement
Cost	Low cost	High cost
Update the app	It does not require to update the application.	It requires to update the application.
Implementation	It is easy to implement.	It is complex to implement.
Indexed by google	Yes	No
Shareable	It is easy to share from anyone.	It shares the entire application, so it complex.

1.3.10. eCommerce Web Apps

This type of web application can be considered as an online store, same as eCommerce Apps or eCommerce Sites.Developing such web apps becomes complex as one has to handle transactions and integrate different payment methods such as PayPal, Debit/Credit Card, etc.

Several primary features of an eCommerce web app include adding new products, removing old products, handling payments, user-friendly interface, etc. To look after all these tasks, admin requires an effective management panel.To develop a feature-rich, reliable, and scalable eCommerce web app, you should hire dedicated eCommerce developers. These developers have developed numerous eCommerce web apps; hence, they are familiar with all the ins and outs of the eCommerce industry.

Pros of eCommerce Web Apps

- √ Enhanced brand recognition
- √ Boost conversation
- √ Increased engagement

- √ Cost-Effective
- √ Smaller size than mobile apps

1.4 Mobile Application Development

Mobile application development is the process of creating software applications that run on a mobile device, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles (code, binaries, assets, etc.) , implementing backend services such as data access with an API, and testing the application on target devices.

1.4.1 Mobile Applications and Device Platforms

There are two dominant platforms in the modern smartphone market. One is the iOS platform from Apple Inc. The iOS platform is the operating system that powers Apple's popular line of iPhone smartphones. The second is Android from Google. The Android operating system is used not only by Google devices but also by many other OEMs to built their own smartphones and other smart devices.

Although there are some similarities between these two platforms when building applications, developing for iOS vs. developing for Android involves using different software development kits (SDKs) and different development toolchain. While Apple uses iOS exclusively for its own devices, Google makes Android available to other companies provided they meet specific requirements such as including certain Google applications on the devices they ship. Developers can build apps for hundreds of millions of devices by targeting both of these platforms.

1.4.2 Alternatives for Building Mobile Apps

There are four major development approaches when building mobile applications

- Native Mobile Applications
- Cross-Platform Native Mobile Applications
- Hybrid Mobile Applications

- Progressive Web Applications

Each of these approaches for developing mobile applications has its own set of advantages and disadvantages. When choosing the right development approach for their projects, developers consider the desired user experience, the computing resources and native features required by the app, the development budget, time targets, and resources available to maintain the app.

Native Applications

Native mobile applications are written in the programming language and frameworks provided by the platform owner and running directly on the operating system of the device such as iOS and Android.

Cross-Platform Applications

Cross-platform native mobile applications can be written in variety of different programming languages and frameworks, but they are compiled into a native application running directly on the operating system of the device.

Hybrid-Web Applications

Hybrid mobile applications are built with standard web technologies - such as JavaScript, CSS, and HTML5 - and they are bundled as app installation packages. Contrary to the native apps, hybrid apps work on a 'web container' which provides a browser runtime and a bridge for native device APIs via Apache Cordova.

Progressive Web Applications

PWAs offer an alternative approach to traditional mobile app development by skipping app store delivery and app installations. PWAs are web applications that utilize a set of browser capabilities - such as working offline, running a background process, and adding a link to the device home screen - to provide an 'app like' user experience.

1.4.3 Native Vs Cross-Platform Vs Hybrid Vs Progressive Web Applications

Native Applications	Cross-Platform Applications	Hybrid-Web Applications	Progressive Web Applications
Best runtime performance	Single code base for multiple platforms	Shared code base between web and mobile apps	Same app is available both for web and mobile
Direct access to device APIs	Easy to build and maintain your app	Using web development skillset for building mobile apps	No installation required, accessible through a URL
Higher costs when building and maintaining your app	Dependents on bridges and libraries for native device features	Lower performance compared to native apps	Limited support for native device features
Multiple code-bases for each platform	Performance limitations due to bridging	Limited support for native device features	App capabilities depend on the browser in use

1.5 Comparing Native vs. Hybrid Applications

At the highest level, there are four main ways that native apps differ from hybrid apps as illustrated in the following table.

Native	Hybrid
Platform Specific	Cross Platform
Compiled Language	Scripting / Compiled

Access to Device Hardware	Plugins / Native Modules
Platform Frameworks	Web Frameworks

1.6 Why Choose the Hybrid/Cross-platform Approach?

One problem with native mobile application development is that it requires a highly specialized skill set. Although there are large and vibrant developer communities for C and Java -- the language families that are mostly used for native development --, there are fewer developers who are knowledgeable in platform-specific versions of those languages and their respective IDEs. In fact, skilled native app developers are in such demand, that many companies are hard-pressed to hire and retain them on staff, and instead they frequently have to resort to outside 3rd party design and development houses to build their apps for them.

1.7 How Hybrid and Cross-platform Frameworks Work?

Hybrid apps allow developers to use web technologies of HTML5/CSS/JavaScript and then encapsulate those web applications in a container that allows the web application to act like a native application on the device. Since hybrid mobile apps are just web apps running on an embedded browser environment, most of the code from a web app can be used to build a mobile app. As rendering and runtime performance of mobile browsers are ever-increasing, hybrid development is a viable alternative for web developers who want to build mobile apps quickly.

Similarly, PWAs are written using traditional web application programming technologies usually including some variant of JavaScript, HTML5, and CSS, and are accessed initially through a browser on the device or computer.

Most cross-platform frameworks such as - React Native and Native Script - provides native components to work with the cross-platform code, while some others such as Flutter and Xamarin compiles cross-platform code to the native code for better performance.

1.8 The Mobile Application Development Lifecycle

There are two interlinked core components of a mobile application:

1) the mobile application "Front-End" that resides on the mobile device, and

2) the services "Back-End" that supports the mobile front-end.

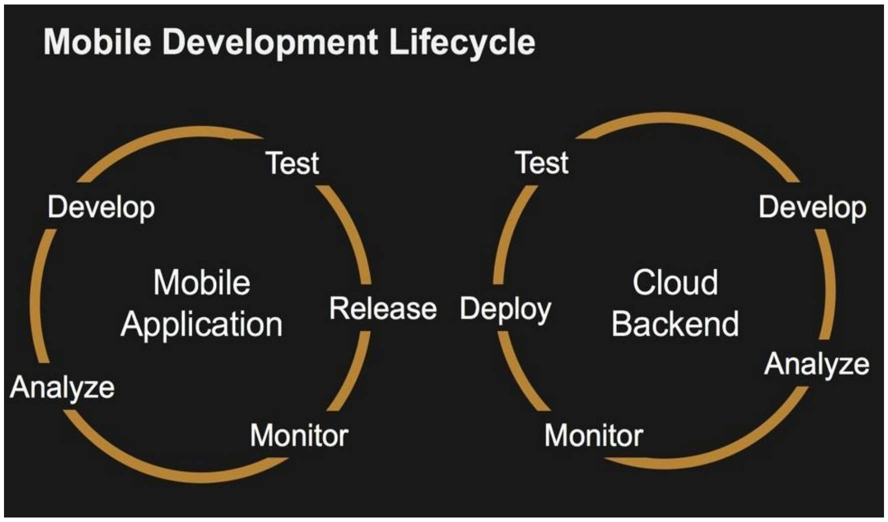


Fig 1.3 Lifecycle of Mobile Application Development

1.8.1 Front-end vs. Back-end

In the early days of the modern smartphone applications era, mobile applications went through a similar evolution as first websites. At first, the applications and sites where wholly contained within themselves and acted as little more than static advertisements for the brand, company, product, or service.

However, as connectivity and network capabilities improved, the applications became increasingly connected to sources of data and information that lived outside of the app itself, and the apps became increasingly dynamic as they were able to update their UI and content with data received over the network from queries to data sources.

s a result, the mobile front-end applications increasingly rely on and integrated with back-end services which provide data to be consumed through the mobile front-end. Such data can include, for example, product information for e-commerce apps or flight info for travel and reservation apps. For a mobile game, the data may include new levels or challenges and scores or avatars from other players.

1.8.2 How Front-end 'Talks' to the Back-end?

The mobile front-end obtains the data from the back-end via a variety of service calls such as APIs. In some cases, these APIs may be owned and operated by the same entity developing the mobile application. In other cases, the API may be controlled by a third party and access is granted to the mobile application via a commercial arrangement.

For example, a developer may obtain social media or advertising content by making calls to media or advertising company services. In this case, a developer may have to sign a contract in order to obtain credentials and a key that grants access to the API and governs how that developer can use it, how much it will cost, or how frequently it may be called, or how much data can be requested over what time period.

1.8.3 The Mobile Application Front-End

The mobile front-end is the visual and interactive part of the application the user experiences. It usually resides on the device, or there is at least an icon representing the app that is visible on the home screen or is pinned in the application catalog of the device. The application can be downloaded from the platform app store, side-loaded directly onto the device, or can be reached through the device's browser, as in the case for PWAs.

How Mobile Apps Integrate with the Backend?

Regardless of the size of the team, a critical element of the development effort is building the app logic that is responsible for making network calls to the back-end services, retrieve data and update the data in the back-end systems with new information generated from the app.

These back-end services are typically accessed through a variety of application programming interfaces, most commonly known as APIs. There are different types of APIs, such as REST and GraphQL, and there are also a wide variety of means and styles of accessing them. While some back-end service APIs are available directly to the application through calls in the platform itself, many of the specialized services have to be integrated into the app via a software development kit, commonly known as an SDK. Once the SDK has been added to the app via the development environment, then the application can make use of the APIs defined in the SDK.

1.8.4 The Mobile Application Back-End

Regardless of what front-end platform or development methodology is being used, delivering high-quality mobile applications that delight and retain users requires reliable back-end services.

Given the critical importance of back-end services for the success of the mobile application, the developers have several important architectural decisions that they must consider. These decisions include which services should they build themselves and which third party services should they leverage, and then should they run and maintain their own services or should they take advantage of 3rd party services.

The answer is increasingly clear; to improve developer productivity and efficiency, mobile app programmers should only build their own services if they are highly specific to the domain of the application and embody unique intellectual property. Also, even for the services they build themselves, they should almost always leverage cloud-based services to build and maintain their backend infrastructure.

1.8.5 List of Mobile Application Services

There are hundreds of cloud and 3rd party services that mobile application developers can leverage to speed up the development and delivery of their applications. However, it's unlikely that a developer is going to be able to become an expert in each of these individual services.

Instead, the mobile developers should look for a development environment that makes it easier for them to integrate, use, and consume the most commonly required capabilities into their application quickly and easily, while still preserving the freedom to take advantage of the many individual services available.

- Essential
    - User Sign-up/Sign-in and Management
    - Social login (Facebook sign-in, Twitter sign-in, etc.)
    - Analytics and User Engagement
    - Push Notifications

    - Real Device Testing

- Data Services
    - Cloud Storage
    - Real-time and Offline Data
    - Application Logic/Cloud Functions
- Machine Learning
    - Conversational Bots
    - Image and Video Recognition
    - Speech Recognition

1.9 What is a Progressive Web App (PWA)?

A progressive web app is a website that looks and feels like a native app. Progressive web apps are built in the web and run in the browser. There's no need to download the app from the Google Play Store or iOS App Store.PWAs are meant to eliminate a range of issues from slow networks to data limitation or complete lack of connectivity. Progressive Web Apps leverage the latest web technologies to provide a reliable, fast, and engaging user experience.

Twitter is the success story progressive web applications. Go ahead and log into your Twitter account via your smartphone’s browser. Ta-da! You’re now using a Progressive Web App that’s capable of performing real-time notifications, offline notifications, and other app-like functions.

Another progressive web app example can be found in **Gmail.** Again, log into your Gmail account via your smartphone’s browser, and you’ll experience an app-like experience that allows you to individually select emails, label them, move them between folders, and so forth. You’ll also see new emails drop into your inbox in real-time.

1.9.1 What Makes a Progressive Web App?

The term Progressive Web App was coined in 2015 by Alex Russell. Together with Frances Berriman, Russell "enumerated the attributes of [a] new class of applications" based on the gradual and powerful evolution of modern browsers. Here are those attributes as Russell and Berriman envisioned them:

- √ **Responsive:** To fit any form factor
- √ **Connectivity independent**: Progressively-enhanced with Service Workers (we'll explain these in more detail below) to let them work offline
- √ **App-like-interactions:** Adopt a Shell + Content application model to create appy navigations & interactions
- √ **Fresh:** Transparently always up-to-date thanks to the Service Worker update process
- √ **Safe:** Served via TLS (a Service Worker requirement) to prevent snooping
- √ **Discoverable**: Are identifiable as "applications" thanks to W3C Manifests and Service Worker registration scope allowing search engines to find them
- √ **Re-engageable**: Can access the re-engagement UIs of the OS; e.g. Push Notifications
- √ **Installable:** To the home screen through browser-provided prompts, allowing users to "keep" apps they find most useful without the hassle of an app store
- √ **Linkable**: Meaning they're zero-friction, zero-install, and easy to share.

1.9.2 Building a progressive web app

There are 3 fundamental components:

- **Web manifest**

  The web manifest is a JSON file that defines the look and feel of the PWA when it's installed. It's contains anything about looks like home screen icons, colors, names etc . In general, the web manifest includes metadata like the app name, version, description, theme colors, and screen orientation. A web manifest is essential for creating a native-like app experience.

- **Service worker**

  A service worker is JavaScript code that runs in the background of the PWA. Their primary use is to pre-cache resources, so the web app loads fast and can even have offline functionality. After the website has been visited once, the service worker saves - or caches - critical assets like HTML files and images. Service workers can also be used for other tasks such as push notifications and background data syncs.

- **Transport Layer Security**

PWAs are required to communicate over HTTPS by having an SSL certificate installed on its web server. The SSL certificate creates a secure, encrypted connection between the frontend app and backend server. These requirements leverage the TLS protocol to ensure secure data transfers when the web app communicates with the backend eCommerce and CMS systems. This is crucial for keeping user information safe and is critical for eCommerce stores that handle customer credit card information.

1.9.3 Benefits of Progressive Web Apps

While moving to a PWA will likely require some development work, there are enormous advantages in putting in the effort.

1. It's faster-

- With pre-caching, PWAs load fast even with poor connectivity on mobile devices. Faster loading also translates to better indexing by search engines, and therefore, PWAs have considerable SEO advantages.

2. It's better for SEO

- From an SEO standpoint, search engines view PWAs as websites and fully index them. Native apps, on the other hand, are not indexed and will not impact your SERP.

3. It reduces data needs

- Caching reduces the data transfer needs of your app. Fewer API requests reduce your need for server capacity and bandwidth to support your application. Your customers will also use less of their mobile data plan while browsing your web app. Along with this, PWAs usually take up less storage space, so users are more likely to download them than native apps.

4. There’s no need for app stores

    - One of the greatest competitive advantages of PWAs over native apps is discoverability. PWAs aren't downloaded via app stores, so there's less friction for potential users to install the app. You also don’t need to build, submit for review, and market separate apps for iOS and Android. Users that are already on your website have the option to install the app, giving them a better user experience and leading to a higher chance of adoption.

5. It reduces development costs

    - Native apps require entirely different tech stacks to develop, while PWAs use standard web technologies like HTML, CSS, and JavaScript that are well-know by frontend developers. Upgrading your existing website with PWA features is much cheaper than developing native apps from scratch, and any updates you need to make are instant without requiring review by Apple or Google.

6. It leads to better user experiences

    - PWAs are both responsive and progressive. Responsive means they're designed to work on whichever devices users have from phones to tablets or desktops. PWAs also built with the web development idea of progressive enhancements. This means they focus first on core content and features, and if the user's browser and internet connection meet the necessary more advanced features become available. These two tenets ensure a better digital experience for all users.

7. It leads to better engagement

    - There's no doubt that PWAs lead to better customer engagement. The caching and offline capabilities of PWAs mean users can browse the eCommerce store at any time, and won't abandon the web app from long loading times. Features like push notifications and location tracking let marketers provide relevant content to mobile users whenever and wherever. These factors make PWAs great for delivering enhanced digital experiences to shoppers.

8. It can increase conversions

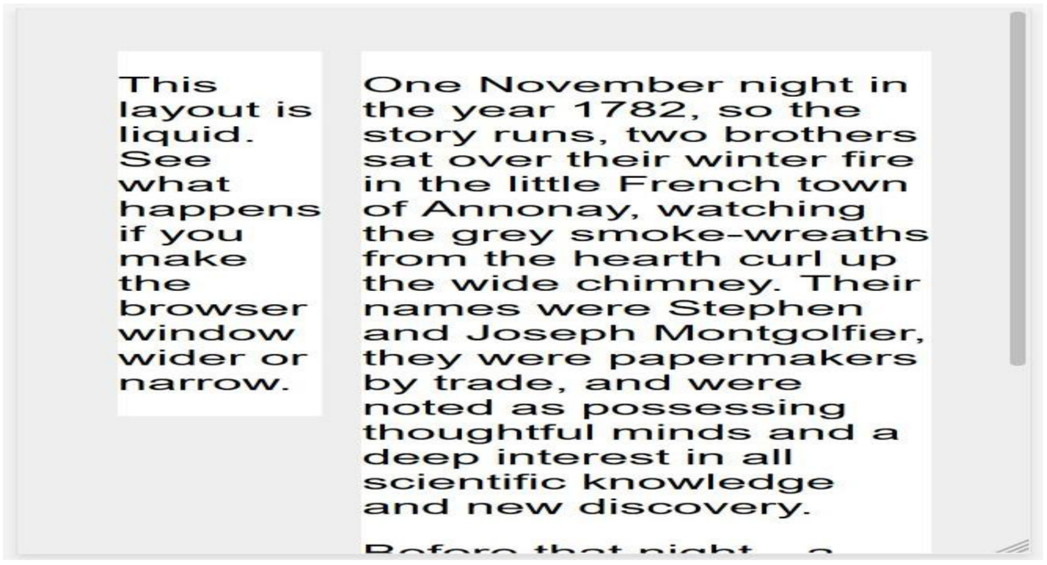
    - For eCommerce stores, moving to a PWA can drastically improve conversions. AliExpress, for example, increased its conversion rate for new users by 104% with its new PWA. Along with better engagement, PWAs can offer simplified user experience and checkout process that efficient moves user down the sales funnel.

1.10 Responsive web design

Responsive web design (RWD) is a web design approach to make web pages render well on all screen sizes and resolutions while ensuring good usability. It is the way to design for a multi-device web.

HTML is fundamentally responsive, or fluid. If you create a web page containing only HTML, with no CSS, and resize the window, the browser will automatically reflow the text to fit the viewport.

While the default responsive behavior may sound like no solution is needed, long lines of text displayed full screen on a wide monitor can be difficult to read. If wide screen line length is reduced with CSS, such as by creating columns or adding significant padding, the site may look squashed for the user who narrows their browser window or opens the site on a mobile device.



This layout is liquid. See what happens if you make the browser window wider or narrow.

One November night in the year 1782, so the story runs, two brothers sat over their winter fire in the little French town of Annonay, watching the grey smoke-wreaths from the hearth curl up the wide chimney. Their names were Stephen and Joseph Montgolfier, they were papermakers by trade, and were noted as possessing thoughtful minds and a deep interest in all scientific knowledge and new discovery.

Before that night, a

Fig 1.4 Liquid Layout Design

Creating a non-resizable web page by setting a fixed width doesn't work either; that leads to scroll bars on narrow devices and too much empty space on wide screens.

Responsive web design, or RWD, is a design approach that addresses the range of devices and device sizes, enabling automatic adaption to the screen, whether the content is viewed on a tablet, phone, television, or watch.

Responsive web design isn't a separate technology — it is an approach. It is a term used to describe a set of best practices used to create a layout that can respond to any device being used to view the content.

The term responsive design, coined by Ethan Marcotte in 2010, described using fluid grids, fluid images, and media queries to create responsive content, as discussed in Zoe Mickley Gillenwater's book Flexible Web Design.

At the time, the recommendation was to use CSS float for layout and media queries to query the browser width, creating layouts for different breakpoints. Fluid images are set to not exceed the width of their container; they have their max-width property set to 100%. Fluid images scale down when their containing column narrows but do not grow larger than their intrinsic size when the column grows. This enables an image to scale down to fit its content, rather than overflow it, but not grow larger and become pixelated if the container becomes wider than the image.

1.10.1 Media Queries

Media queries allow us to run a series of tests (e.g. whether the user's screen is greater than a certain width, or a certain resolution) and apply CSS selectively to style the page appropriately for the user's needs.

For example, the following media query tests to see if the current web page is being displayed as screen media (therefore not a printed document) and the viewport is at least 80rem wide. The CSS for the .container selector will only be applied if these two things are true.

```
@media screen and (min−width: 80rem) {
 .container {
   margin: 1em
   2em;
 }
}
```

We can add multiple media queries within a stylesheet, tweaking your whole layout or parts of it to best suit the various screen sizes. The points at which a media query is introduced, and the layout changed, are known as breakpoints.

A common approach when using Media Queries is to create a simple single-column layout for narrow-screen devices (e.g. mobile phones), then check for wider screens and implement a multiple-column layout when you know that you have enough screen width to handle it. Designing for mobile first is known as mobile first design.

If using breakpoints, best practices encourage defining media query breakpoints with relative units rather than absolute sizes of an individual device.

There are different approaches to the styles defined within a media query block; ranging from using media queries to <link> style sheets based on browser size ranges to only including custom properties variables to store values associated with each breakpoint.

Media queries can help with RWD, but are not a requirement. Flexible grids, relative units, and minimum and maximum unit values can be used without queries.

1.10.2 Responsive layout technologies

Responsive sites are built on flexible grids, meaning we don't need to target every possible device size with pixel perfect layouts.

By using a flexible grid, you can change a feature or add in a breakpoint and change the design at the point where the content starts to look bad. For example, to ensure line lengths don't become unreadably long as the screen size increases you can use columns; if a box becomes squashed with two words on each line as it narrows you can set a breakpoint.

Several layout methods, including Multiple-column layout, Flexbox, and Grid are responsive by default. They all assume that you are trying to create a flexible grid and give you easier ways to do so.

1. Multicol

With multicol, you specify a column-count to indicate the maximum number of columns you want your content to be split into. The browser then works out the size of these, a size that will change according to the screen size.

```
.container {
  column-count: 3;
}
```

If you instead specify a column-width, you are specifying a minimum width. The browser will create as many columns of that width as will comfortably fit into the container, then share out the remaining space between all the columns. Therefore the number of columns will change according to how much space there is.

```
.container {
  column-width: 10em;
}
```

You can use the columns shorthand to provide a maximum number of columns and a minimum column width. This can ensure line lengths don't become unreadably long as the screen size increases or too narrow as the screen size decreases.

2. Flexbox

In Flexbox, flex items shrink or grow, distributing space between the items according to the space in their container. By changing the values for flex-grow and flex-shrink you can indicate how you want the items to behave when they encounter more or less space around them.

In the example below the flex items will each take an equal amount of space in the flex container, using the shorthand of flex: 1 as described in the layout topic Flexbox: Flexible sizing of flex items.

.

```
.container {
  display: flex;
}

.item {
  flex: 1;
}
```

3. CSS grid

In CSS Grid Layout the fr unit allows the distribution of available space across grid tracks. The next example creates a grid container with three tracks sized at 1fr. This will create three column tracks, each taking one part of the available space in the container. You can find out more about this approach to create a grid in the Learn Layout Grids topic, under Flexible grids with the fr unit.

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

4. Responsive images

To ensure media is never larger than its responsive container, the following approach can be used:

img,

Picture,

```
video {
  max-width: 100%;
}
```

This scales media to ensure they never overflow their containers. Using a single large image and scaling it down to fit small devices wastes bandwidth by downloading images larger than what is needed.

Responsive Images, using the <picture> element and the <img> srcset and sizes attributes enables serving images targeted to the user's viewport and the device's resolution. For example, you can include a square image for mobile, but show the same scene as a landscape image on desktop.

The <picture> element enables providing multiple sizes along with "hints" (metadata that describes the screen size and resolution the image is best suited for), and the browser will choose the most appropriate image for each device, ensuring that a user will download an image size appropriate for the device they are using. Using <picture> along with max-width removes the need for sizing images with media queries. It enables targeting images with different aspect ratios to different viewport sizes.

5. Responsive typography

Responsive typography describes changing font sizes within media queries or using viewport units to reflect lesser or greater amounts of screen real estate.

Using media queries for responsive typography

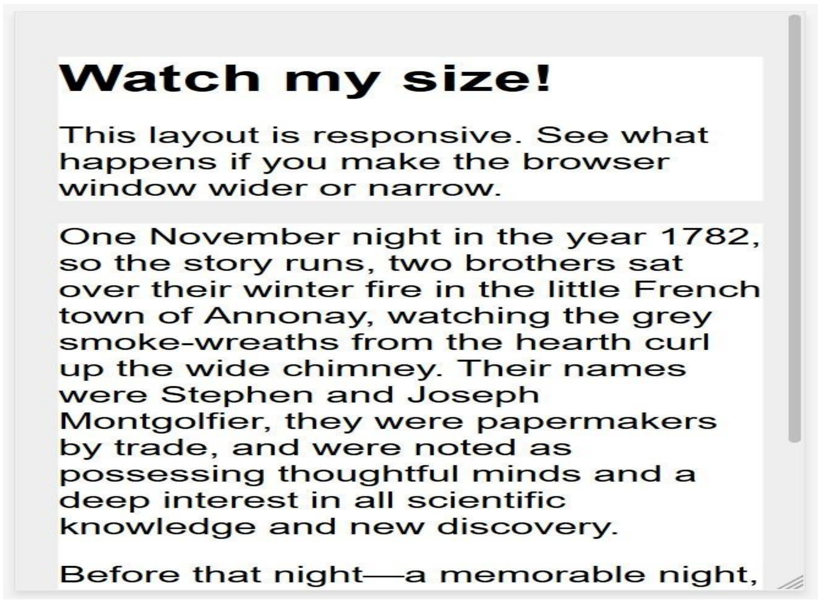
In this example, we want to set our level 1 heading to be 4rem, meaning it will be four times our base font size. That's a really large heading! We only want this jumbo heading on larger screen sizes, therefore we first create a smaller heading then use media queries to overwrite it with the larger size if we know that the user has a screen size of at least 1200px.

```
html {
  font-size:
  1em;
```

```
h1 {
  font-size: 2rem;
}

@media (min-width: 1200px) {
  h1 {
    font-size: 4rem;
  }
}
```

We have edited our responsive grid example above to also include responsive type using the method outlined. You can see how the heading switches sizes as the layout goes to the two column version.

On mobile the heading is smaller:



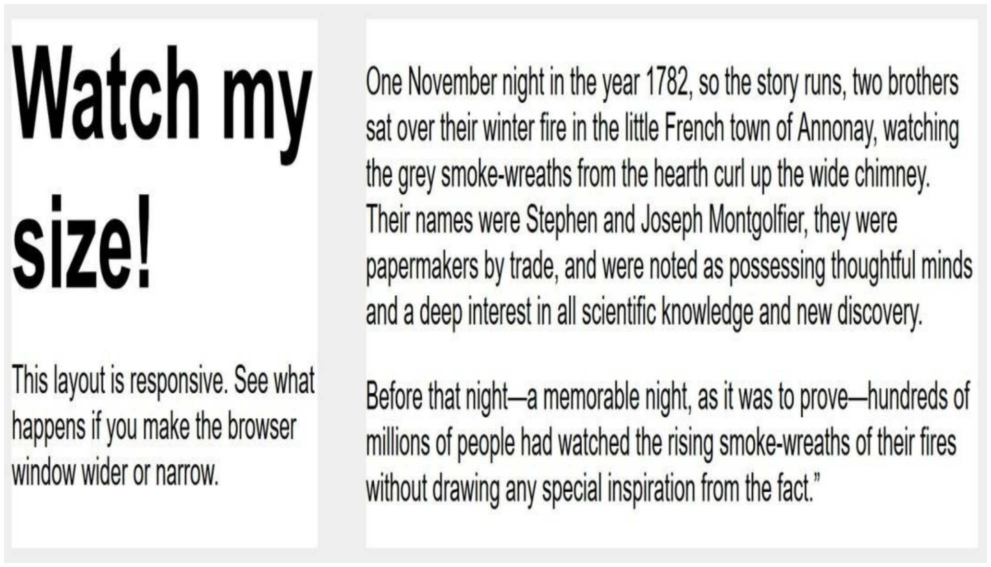
Watch my size!

This layout is responsive. See what happens if you make the browser window wider or narrow.

One November night in the year 1782, so the story runs, two brothers sat over their winter fire in the little French town of Annonay, watching the grey smoke-wreaths from the hearth curl up the wide chimney. Their names were Stephen and Joseph Montgolfier, they were papermakers by trade, and were noted as possessing thoughtful minds and a deep interest in all scientific knowledge and new discovery.

Before that night—a memorable night,

On desktop, however, we see the larger heading size:


Watch my size!

This layout is responsive. See what happens if you make the browser window wider or narrow.

One November night in the year 1782, so the story runs, two brothers sat over their winter fire in the little French town of Annonay, watching the grey smoke-wreaths from the hearth curl up the wide chimney. Their names were Stephen and Joseph Montgolfier, they were papermakers by trade, and were noted as possessing thoughtful minds and a deep interest in all scientific knowledge and new discovery.

Before that night—a memorable night, as it was to prove—hundreds of millions of people had watched the rising smoke-wreaths of their fires without drawing any special inspiration from the fact."

As this approach to typography shows, you do not need to restrict media queries to only changing the layout of the page. They can be used to tweak any element to make it more usable or attractive at alternate screen sizes.

Using viewport units for responsive typography

Viewport units vw can also be used to enable responsive typography, without the need for setting breakpoints with media queries. 1vw is equal to one percent of the viewport width, meaning that if you set your font size using vw, it will always relate to the size of the viewport.

```
h1 {
  font-size: 6vw;
}
```

The problem with doing the above is that the user loses the ability to zoom any text set using the vw unit, as that text is always related to the size of the viewport. Therefore you should never set text using viewport units alone.

There is a solution, and it involves using calc(). If you add the vw unit to a value set using a fixed size such as ems or rems then the text will still be zoomable. Essentially, the vw unit adds on top of that zoomed value:

```
h1 {

  font-size: calc(1.5rem + 3vw);   }
```

This means that we only need to specify the font size for the heading once, rather than set it up for mobile and redefine it in the media queries. The font then gradually increases as you increase the size of the viewport.

The viewport meta tag

If you look at the HTML source of a responsive page, you will usually see the following <meta> tag in the <head> of the document.

```
<meta name="viewport" content="width=device-width,initial-scale=1" />
```

This viewport meta tag tells mobile browsers that they should set the width of the viewport to the device width, and scale the document to 100% of its intended size, which shows the document at the mobile-optimized size that you intended.

Why is this needed? Because mobile browsers tend to lie about their viewport width.

This meta tag exists because when smartphones first arrived, most sites were not mobile optimized. The mobile browser would, therefore, set the viewport width to 980 pixels, render the page at that width, and show the result as a zoomed-out version of the desktop layout. Users could zoom in and pan around the website to view the bits they were interested in, but it looked bad.

By setting width=device-width you are overriding a mobile device's default, like Apple's default width=980px, with the actual width of the device. Without it, your responsive design with breakpoints and media queries may not work as intended on mobile browsers. If you've got a narrow screen layout that kicks in at 480px viewport width or less, but the device is saying it is 980px wide, that user will not see your narrow screen layout.

Responsive Web Design -

Grid-View

Many web pages are based on a grid-view, which means that the page is divided into columns:

Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page.

A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window.

Building a Responsive Grid-View

Let's start building a responsive grid-view.

First ensure that all HTML elements have the box-sizing property set to border-box. This makes sure that the padding and border are included in the total width and height of the elements.

Add the following code in your CSS:

```
* {
  box-sizing: border-box;
}
```

The following example shows a simple responsive web page, with two columns:

```
.menu {
  width: 25%;
  float: left;
}
.main {
  width: 75%;
  float: left;
}
```

The example above is fine if the web page only contains two columns.

However, we want to use a responsive grid-view with 12 columns, to have more control over the web page.

First we must calculate the percentage for one column: 100% / 12 columns = 8.33%.

Then we make one class for each of the 12 columns, `class="col-"` and a number defining how many columns the section should span:

CSS:

```
.col-1 {width: 8.33%;}

.col-2 {width: 16.66%;}

.col-3 {width: 25%;}

.col-4 {width: 33.33%;}

.col-5 {width: 41.66%;}

.col-6 {width: 50%;}

.col-7 {width: 58.33%;}

.col-8 {width: 66.66%;}

.col-9 {width: 75%;}

.col-10 {width: 83.33%;}

.col-11 {width: 91.66%;}

.col-12 {width: 100%;}
```

All these columns should be floating to the left, and have a padding of 15px:

CSS:

```
[class*="col-"] {
  float: left;
  padding: 15px;

  border: 1px solid red;
```

Each row should be wrapped in a `<div>`. The number of columns inside a row should always add up to 12:

HTML:

```
<div class="row">
  <div class="col-3">...</div> <!-- 25% -->
  <div class="col-9">...</div> <!-- 75% -->
```

The columns inside a row are all floating to the left, and are therefore taken out of the flow of the page, and other elements will be placed as if the columns do not exist. To prevent this, we will add a style that clears the flow:

CSS:

```
.row::after {
  content: "";
  clear: both;
  display: table;
```

We also want to add some styles and colors to make it look better:

Example

```
html {
font-family: "Lucida Sans", sans-serif;
}

.header {
background-color: #9933cc; color: #ffffff;
padding: 15px;
}

.menu ul {
list-style-type: none; margin: 0;
padding: 0;
}

.menu li { padding: 8px;
margin-bottom: 7px; background-color :#33b5e5; color: #ffffff;
box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.menu li:hover {
background-color: #0099cc;
}
```

PART-A

1. **Define web development? And its types?**
2. Web development refers to the creating, building, and maintaining of websites.
3. • It includes aspects such as web design, web publishing, web programming, and database
4. management.
5. • It is the creation of an application that works over the internet i.e. website
6. Web development refers to the creating, building, and maintaining of websites.
7. • It includes aspects such as web design, web publishing, web programming, and database
8. management.
9. • It is the creation of an application that works over the internet i.e. website

Web development refers to the creating, building, and maintaining of websites.

• **It includes aspects such as web design, web publishing, web programming, and database**

management.

• **It is the creation of an application that works over the internet i.e. website**

• Web development refers to the creating, building, and maintaining of websites.

• It includes aspects such as web design, web publishing, web programming, and database management.

• It is the creation of an application that works over the internet i.e. websites.

Two Types

Web Development can be classified into two ways:

• Frontend Development

• Backend Development

- **Frontend Development**

The part of a website where the user interacts directly is termed as front end. It is also

Referred to as the 'client side' of the application.

Backend Development

Backend is the server side of a website. It is part of the website that users cannot see and interact

with. It is the portion of software that does not come in direct contact with the users. It is used to

store and arrange data.

- **Backend Development**

Backend is the server side of a website. It is part of the website that users cannot see and interact

With. It is the portion of software that does not come in direct contact with the users. It is used to

Store and arrange data.

2. What is Mobile application development?

Mobile application development is the process of making software for smartphones, tablets and digital assistants, most commonly for the Android and iOS operating systems. The software can be preinstalled on the device, downloaded from a mobile app store or accessed through a mobile web browser. The programming and markup languages used for this kind of software development include Java, Swift, C# and HTML5.

3. What are the different ways to develop the mobile apps?

There are 3 different ways to develop Mobile apps:

1. 1st Party Native App development
2. Progressive web Application
3. Cross-Platform Application

There are 3 different ways to develop Mobile apps:

1. 1st Party Native App development

2. Progressive web Application

3. Cross-Platform Application

4. Difference between PWA and Native Application?

5. It saves money and time compared to creating applications separately for android, iOS, and other platforms.
6. Post can be read even if there is no internet.
7. Internet data is less used in it.

8. PWA is cheaper than the other applications.

Disadvantages of PWA

1. It supports a limited mobile browser. It does not run on the safari, edge, and IE browser.

2. iPhone users cannot establish connections securely in it.

3. It makes maximum use of the battery of the device.

4. It needs to be hosted on the server because it is a web app.

5. It cannot be downloaded from popular app stores such as Google Play and Apple App Store.

6. PWA does not provide the same level of support for all devices. For example, push

notifications in PWA work on Android, but not on iOS.

7. It supports limited hardware functionality.

Difference between PWA and Native Application

Feature

Progressive Web Application

Native Application

Function offline

Yes

Yes

Installation

requirement

There is no need to install it in mobile.

It is necessary to install it in the phone.

Push-notification.

It supports the push-notification

feature.

It also supports the push-notification

feature.

Platform

It supports the cross-platform.

It supports the specific platform. Example

iOS, Android, and Windows

Data consumption

Low data consumption

High data consumptio

Features	**Progressive Web Application**	**Native Application**
Installation requirement Installation requirement	There is no need to install it mobile	It is necessary to install it in the phone
Push-notification	It supports the push-notification Feature	It also supports the push-notification feature
Platform	It supports the cross-platform.	It supports the specific platform. Example

		iOS, Android, and Windows
Data consumption	Low data consumption	High data consumption
Shareable	It is easy to share from anyone.	It shares the entire application, so it Complex

5. What is Cross-Platform Development?

Cross-platform mobile development is an approach to developing software applications that are

compatible with multiple mobile operating systems (OSes) or platforms.

Multiple frameworks could be used for cross-platform app development.

- Titanium
- React Native
- Xamarin
- Flutter
- Native Script
- Ionic
- Js
- PhoneGap(Cordova)

6. Define hybrid apps? Mention the examples of hybrid apps?

A hybrid application is a software app that combines elements of both native and web

Applications.

Hybrid apps are popular because they allow developers to write code for a mobile app once and still accommodate multiple platforms. Because hybrid apps add an extra layer between the source code and the target platform, they may perform slightly slower than native or web versions of the same app.

Five hybrid mobile apps that is extremely popular among users across the globe:

1. Example #1: Instagram
2. Example #2: Uber
3. Example #3: Gmail
4. Example #4: Evernote
5. Example #5: Twitter

7. **What are the characteristics of progressive web application?**

Progressive: The term progressive means, a PWA application must work on any device and

improve the performance of the user's mobile browser and design.

2. Discoverable: A PWA is a website with some extra features. It can be searched via mobile

searching applications like Google Chrome. App Store or Play Store is not required for this.

3. Responsive: The UI of a progressive web app should fit the form factor and screen size of

the device.

4. App-like: A PWA application should look like a native application. Although the methods

for creating, sharing, launching, and updating of the PWA are completely different from the original application.

5. Connectivity-independent: It works even when connectivity is very low

1. Progressive: The term progressive means, a PWA application must work on any device and improve the performance of the user's mobile browser and design.

2. Discoverable: A PWA is a website with some extra features. It can be searched via mobile searching applications like Google Chrome. App Store or Play Store is not required for this.

3. Responsive: The UI of a progressive web app should fit the form factor and screen size of the device.

4. App-like: A PWA application should look like a native application. Although the methods for creating, sharing, launching, and updating of the PWA are completely different from the original application.

5. Connectivity-independent: It works even when connectivity is very low

8. What Is Responsive Web Design?

Responsive Web design is the approach that suggests that design and development should respond to the user’s behaviour and environment based on screen size, platform and orientation. The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries.

Responsive Web Design

- Responsive web design makes your web page look good on all devices.
- Responsive web design uses only HTML and CSS.
- Responsive web design is not a program or a JavaScript.

9. Difference between Progressive web Application and Responsive Web Application?

	PWA	**RWA**
Loading	PWAs are as fast as native apps. Thanks	Loading and processing data

Speed	to an app shell, the code will be cached on your device after the first load.	on a mobile device will take much longer.
Adaptability	Independent of the devices and platforms.	You need to write different code for various platforms.
Security	PWAs are reliably protected through the use of the HTTPS protocol.	While an RWA might be as secure as a PWA, responsive websites are not required to use secure protocols.
Home screen icon	It has a clickable icon just like a native app.	The ability to add an app icon to the home screen is absent here.
Push Notifications	Once the PWA is opened, the user will be prompted to enable unique push notifications.	Does not provide users with this option

10. what are the Challenges of Native App Development?

- Higher Development Costs
- More Development Time
- Need for Skilled Developers
- Require Constant Updates
- Lengthy Downloading Process

PART-B

1. Brief note on Mobile Application Development?
2. Explain detail about Basics of Web Application?

3. Brief note on Hybrid APP? Mention its Pros and Cons of Hybrid App?

4. Explain detail about Technical Components of Progressive Web Application?

5. Summarize the difference of Web apps, Hybrid apps and Native apps?

UNIT II

NATIVE APP DEVELOPMENT USING JAVA

Native Web App, Benefits of Native App, Scenarios to create Native App, Tools for creating Native App, Cons of Native App, Popular Native App Development Frameworks, Java &Kotlin for Android, Swift & Objective-C for iOS, Basics of React Native, Native Components, JSX, State, Props

Native Web App

A native app refers to a software program specifically written and created to work on platforms and devices where it can be preinstalled, downloaded, configured, and updated to the latest version via an app marketplace

A native application that is live on a device can be accessed through a home screen icon. It alerts the user of notifications, and it is capable of functioning offline as well.

Benefits of Native Apps

Here are some of the most prominent native app benefits:

- Security and Reliability
- Optimal Performance
- Access to All Built-In Device Hardware Features
- Improved User Experience
- Increased Scalability
- Fewer Bugs
- Instant Updating
- Works in Offline Mode

Security and Reliability

A native application is likely to have lesser platform-specific vulnerabilities than hybrid apps that depend highly on a browser security system.

It is robustly protected against misuse due to the multiple layers of the operating system’s security. A native app undergoes security inspection on every one of these layers and through each system or version upgrade.

Optimal Performance

Native apps encounter fewer bugs. Operating faster and more efficiently than with alternative apps, a native application is generally favorably responsive and dependable.

This ensures user satisfaction. Also, because they encounter fewer bugs, users find themselves free from worrying over the app's shutting down amid usage.

They even have commendable offline performance. After installation and during online operation, a native app maximizes its in-browser caching attribute, enabling availability in offline mode derived from its cached resources.

Access to All Built-In Device Hardware Features

Since a native application has high compatibility with specified platforms, its users can configure direct integration with the gadget's hardware like microphone and camera.

Advanced Customization Options

Native apps have full access to operative systems or device features, which can be highly customized.
Developers don't have limitations; they can access all parts of hardware that help create a unique user interface (UI) and user experience (UX).
For example, they can utilize a microphone function for sound effects or a camera for filters.

Improved User Experience

Every platform has its own UI/UX guidelines developers must follow. With native apps, these standards are rigorous, guaranteeing a consistent look and feel with the operating system.
The consistency of native mobile apps provides a much more intuitive and interactive user experience. Native apps are much faster and more efficient than web apps, which further enhances the user experience.

Increased Scalability

Native apps offer increased scalability, they can easily handle an increase in the number of users or amount of data being processed without sacrificing performance.

This is because native apps efficiently use the hardware or given platform, with minimum risk of crashing in case of extreme traffic increase.

Native apps can also be updated and improved over time to meet the changing needs of a growing user base.

Fewer Bugs

Native apps are designed for a specific operating system, such as iOS and Android, and they use programming languages and software development tools optimized for that platform.

Compared to web apps, which are designed to run on multiple platforms and may have compatibility issues, native apps result in fewer bugs and errors, providing a more stable and reliable user experience and leading to higher user satisfaction and retention.

Instant Updating

Android and iOS frequently release updates and developers must instantly implement them into the apps to preserve a satisfying user experience. This struggle doesn't affect native apps because they have instant updates.

Works in Offline Mode

Unlike cross-platform apps, native apps will work even when the Internet is unavailable. For example, you are driving, and the Internet is not stable en route.

When built using the native app development approach, this app will run even though there is no stable connection.

Drawbacks of a Native App Cost

The overall cost involved in the development and maintenance of a native app is considerably higher. This is due to the fact that there should be separate versions of the same application. Even substantial amount is needed to maintain the app. But still, native apps are cost effective in the long run.

Development

Developing a native app is a difficult process since separate developers are needed for each platform. For an example, different developers must be hired to develop Android and

IOS version of the same application. Moreover, it is not an easy task to develop native apps. It is incorporated with tough challenges.

Time Consumption

Since native apps are developed for multiple platforms, it requires more time. Native apps may require significant amount of time for making compared to their counterparts. Developers of native app have to take time to write codes for specific O/S.

Updates

Developers often come up with new updates in native apps for various reasons. Most often for fixing bugs and glitches. Hence, necessary updates needs to be implemented in app store so that so that users will be able to download them. Now the problem comes if the user isn’t aware of such updates or skips them to save storage space.

Download Requirement

Prior to using a native app, it is a must to download it from either App store or Plays store. There are several process involved in downloading a native app. They must find the app, go though the terms & conditions and then go with the download process. Sometimes the download process can be lengthy that the users wouldn't have patience.

Frequency of Updates and Too Much Storage Requirements

While the regular introduction of newer versions and upgrades is necessary to fix bugs and rectify errors and malfunctions, it can be tedious and disruptive for the end users. On top of that, these updates also take up a sizeable amount of device storage space.

Monetization of a Native Application

Native app developers can charge for every download. The payment process will go through the app store or shop which will take a percentage of the fees.

Native Apps – Pros & Cons

The benefits of Native Apps are:

- Easier & faster
- Rich functionalities
- Mutually offline & online features
- Confident, safe & secure

The disadvantages of Native Apps are:

- Expensive to maintain
- No compatibility of cross-platforms
- Affluent maintenance
- Limited apprises

Major examples of popular mobile native apps are:

- Messaging – WhatsApp
- Navigation program – Waze
- Social Media application – Twitter
- Game – Pokémon GO

Stages of App Development Process:

1. Validate the Idea to Decide the Core Concept
2. Build a User Persona to Understand the Target Audience
3. Conduct Competitor Analysis to Understand Your Competitor
4. Plan for UI/UX to Design Engaging Mobile App
5. Choose the Platform Between Native & Cross-platform
6. Identify Monetization Options to Build Profitable Mobile App
7. Make your App Secure

Validate the Idea to Decide the Core Concept

Having a concept for native mobile apps is not enough; you need to validate it for the app market. Check if people are ready to embrace the mobile app you plan to develop. Also, identify the following points to validate your idea clearly.

- Your application will solve a real problem
- Do people consider it to be a problem
- Are they looking for a solution?
- Will they accept the mobile app as their solution?

Build a User Persona to Understand the Target Audience

It is important to understand the users for both Android and iOS mobile app development platforms. The users will react differently to iOS and Android apps.

The initial step is to study the user, build separate personas and user insights play a pivotal role when it comes to answering questions like how to create an app.

- How do the users move through the app?
- What are some of the things they take for granted in an app?
- How do they use the app?
- What makes them engage with native and hybrid apps?
- How do users access mobile apps?

Conduct Competitor Analysis to Understand Your Competitor

Understanding the users or the market is not enough when developing native apps; you should also know your competition.Your competitors are not limited to native mobile apps; they extend to cross-platform and web apps. The businesses with cross-platform apps are your biggest competitors as they have an extensive reach.

- How are they marketing their apps?
- What are the optimization strategies?
- Why do people use apps for multiple platforms?
- What are the gaps that exist in the apps?
- What kind of feedback is the user giving?
-

Plan for UI/UX to Design Engaging Mobile App

Planning for user experience is one of the most crucial steps of native app development. Apart from this, You should identify your users, how they operate mobile apps, and what makes them tick.

It is equally important to know how they hold their phones, what apps they mostly use, and what makes them love these apps.

- What are the pain points users experience?
- What makes you frustrated when using the mobile app?
- What are some things you believe should be included in an app?

Choose the Platform Between Native & Cross-platform

You have already chosen between native and cross-platform app development and the approach you want to take. It is time to choose the particular operating system for app development.

The cost of app development will go up because you won't use the same code for the operating systems. The coding and development approach will change for the particular operating system, so choosing the operating system before you proceed with development is better.

You can opt for a native development approach to generate good revenue. However, if reach matters the most, you should develop a native Android app.

Identify Monetization Options to Build Profitable Mobile App

As native app developers, you can plan to have a subscription strategy or in-app advertisements to generate revenue. You can also use the freemium method to build a more profitable mobile app.

Whether you are building hybrid apps or developing separate apps for Android and iOS devices, having a monetization strategy comes in handy. It will help you realize how you aim to translate your mobile app into a profitable business.

As native app developers, you can plan to have a subscription strategy or in-app advertisements to generate revenue. You can also use the freemium method to build a more profitable mobile app.

Make your App Secure

The Apple app store has a clear definition of a secure app. While the underlying technology for both Android and iOS devices is secure, you should work in sync with the guidelines laid by the app stores.

It will help build highly secure apps for the different platforms. Native apps generally are considered to be safe and data-protective. However, you cannot take your chances.

- Check for all the data precautions you need to take with the particular platform
- Identify the compliances you need to include in the specific operating system

- Plan for security and authentication for your native mobile applications

Mobile App Development Framework

Mobile App Development Framework is a library that offers the required fundamental structure to create mobile applications for a specific environment. In short, it acts as a layout to support mobile app development. There are various advantages of Mobile App Development frameworks such as cost-effectiveness, efficiency, and many more. Moreover, mobile application frameworks can be classified majorly into 3 categories: Native Apps, Web Apps & Hybrid Apps.

Popular native app development frameworks are:

React Native:

React Native is one of the most recommended Mobile App Frameworks in the development industry. The framework, created by Facebook, is an open-source framework that offers you to develop mobile applications for Android & iOS platforms. The React Native framework is based on React and JavaScript that aims to develop native applications over hybrid applications that run on a web view. Moreover, it is a cross-platform development framework that uses a single code base for both Android & iOS applications. Some of the major benefits of React Native are mentioned below:

- Code Re-usability & Cost-Effective
- Compatible with third-party plugins
- Re-usable components for optimal performance
- Provides hot deployment features
- Ease of Maintenance

There are various renowned mobile applications built with React Native such as Instagram, Soundcloud, Uber Eats, and many more.

Xamarin:

Xamarin is also one of the most popular open-source frameworks used to develop mobile applications. The framework, acquired by Microsoft, is based on .Net and allows you to build native applications for Android, iOS, and Windows platforms. Xamarin comes with almost every required tool and library needed to build native applications and offers you to create rich experiences using native UI elements. Moreover, Xamarin also supports the feature of sharing the common codebase to make the development process more efficient and cost-effective. There are various benefits of Xamarin, some of these are mentioned below:

- Huge Community of around 1.4 million developers
- Native API Access & UI Support
- Easier API Integration
- Target All Platforms
- Cost-Effective & Faster Development Process

Some of the most popular and renowned mobile applications that are built on Xamarin are – OLO, Alaska Airlines, Storyo, and many more.

Flutter

Flutter is an open-source mobile app SDK that was launched by Google. It's Google's UI toolkit which is used to develop beautiful, natively compiled applications for mobile, web, and desktop through a single code base.

Features:

- Fast development
- Built-in material design
- Increased time-to-Market speed
- Rich motion APIs
- Own rendering engine
- Strong widget support
- Similar to Native app performance

Examples:

- Alibaba
- Google Ads
- Reflectly
- Cryptography

Swift	Objective C
• Clean code and fewer bugs and Fast speed applications • High performance and rapid application development • Auto tracking of memory usage	• It is compatible with C and C++ programming languages • Been in use for several years; has a strong community • Flexible coding environment

Java

Java is the most popular, widely used, object-oriented programming language designed by James Gosling. Using Java, we can create a variety of applications such as desktop applications, enterprise applications, cloud-based, and web-based applications. Usually, it is used to develop the back-end. Java is the primary choice for the developers when it comes to Android app development. Note that Android itself is written in Java.

Pros

- Memory is managed by the JVM automatically.
- It is more secure.
- Platform independent.
- Highly secure
- Provides vast community support.

Cons

- Syntaxes in Java are complex and large.
- Verbose and complex code.
- Its performance is poor.

Kotlin

Kotlin is also an object-oriented programming language just like Java developed by JetBrains. It is a general-purpose, open-source, sterically-typed, cross-platform pragmatic programming language with type inference. It is particularly designed to interoperate with Java and also to improve the existing Java models by offering solutions to API design deficiencies.

- Kotlin is specially designed for JVM and Android.
- Kotlin's standard library totally depends on the Java class library.
- It focused on safety, clarity, and interoperability.
- It is lightweight, clean, concise, and less verbose especially for writing callbacks, data classes, and getter/setters.

Why Kotlin?

Kotlin is a modern, concise, interoperable, and safe programming language. It provides a productive way to write a server-side application. It is compatible with the Java ecosystem because we can use our favorite framework and libraries with it. It also saves time and effort.

- It supporting new JVM features, like string concatenation via invokedynamic.
- Improved performance and exception handling for KMM projects.
- Experimental extensions for JDK path **Path("dir") / "file.txt".**

Pros

- It is compatible with existing Java code.
- It is interoperable with Java.
- It increases team productivity.
- It is easily maintainable.
- It is less buggy and more reliable.
- Provides rich API for application development.

Cons

- The compilation speed of Kotlin is slow.
- It has a small developer community.
- Memory consumption is high.

Difference Between Java and Kotlin

Features	Java	Kotlin
Primitive Type	Primitive types in Java are not objects.	Primitive types are objects.
Product	It is a product of **Oracle Corporation**.	It is a product of **JetBrains**.
Used For	It is used to develop stand-alone applications and enterprise applications.	It is used to develop server-side applications and android application development.
Compilation Time	Java's compilation time is pretty fast.	Its compilation time is slow in comparison to Java.
File Extensions	Java uses the extensions: **.java** (for source file), **.class** (for class file), **.jar** (for archived file).	Kotlin uses the extensions: **.kt** (for Kotlin source file), **.kts** (for Kotlin script file), **.ktm** (for Kotlin module)
Checked Exceptions	In Java, we take care of the checked exception by using the try-catch block.	There is no need to catch or declare any exception.
Concise	The code is not concise in comparison to Kotlin.	It reduces the boilerplate code.
Extension	We need to create a new class and inherit the	We can extend a class with new

Function	parent class if we want to extend the functionality of an existing class. So, the extension function is not supported by Java.	functionality by using the extension function.
Widening Conversion	Java supports the implicit conversion so we can convert a smaller type to a bigger one.	Kotlin does not support the implicit conversion. So, we cannot convert the smaller type to a bigger one.
Code Comparison	The line of code is just doubled than Kotlin.	It reduces the line of code to half.
Community Support	Java provides a very large community.	Its community is not so huge as Java.
Casting	In Java, we need to identify and perform the casting.	Kotlin supports the smart cast, which means that it identifies the immutable type and performs implicit casting automatically.
Type interface	It is mandatory to specify the data type, explicitly.	It is not mandatory to specify the type of variable, explicitly.
Null Values	We can assign null values to variables but cannot assign null values to an object.	We cannot assign null values to any variable and objects.
Ternary Operator	It is available in Java.	It does not support ternary operator.
Coroutines Support	Multithreading feature of Java makes it more complex because managing the multiple threads is a difficult task. Java blocks the thread if we initiate a long-running intensive operation like network I/O or CPU operations.	Like Java, we can create multiple threads (long-running intensive operations) in Kotlin also but coroutine can suspend a thread execution at a certain point without blocking the other threads.
Functional Programming	Java is not functional programming.	It is a combination of functional and procedural programming language.
Data Classes	If we need a class that can hold data only, for this we need to define getter, and setter methods, constructors, and other functions.	If we want to do the same in Kotlin, we declare the class with the keyword **Data**. Rest the work such as creating constructor, getter, and setter methods for the fields are done by the compiler.

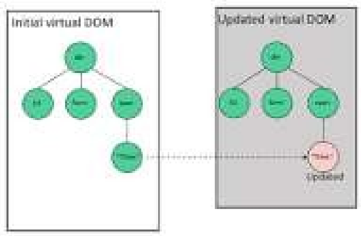
DOM

DOM stands for '**Document Object Model**'. it is a structured representation of the HTML elements that are present in a webpage or web app. DOM represents the entire UI of your application. The DOM is represented as a tree data structure. It contains a node for each UI element present in the web document.

It is a programming interface that allows us to create, change, or remove elements from the document. We can also add events to these elements to make our page more dynamic.

Virtual DOM

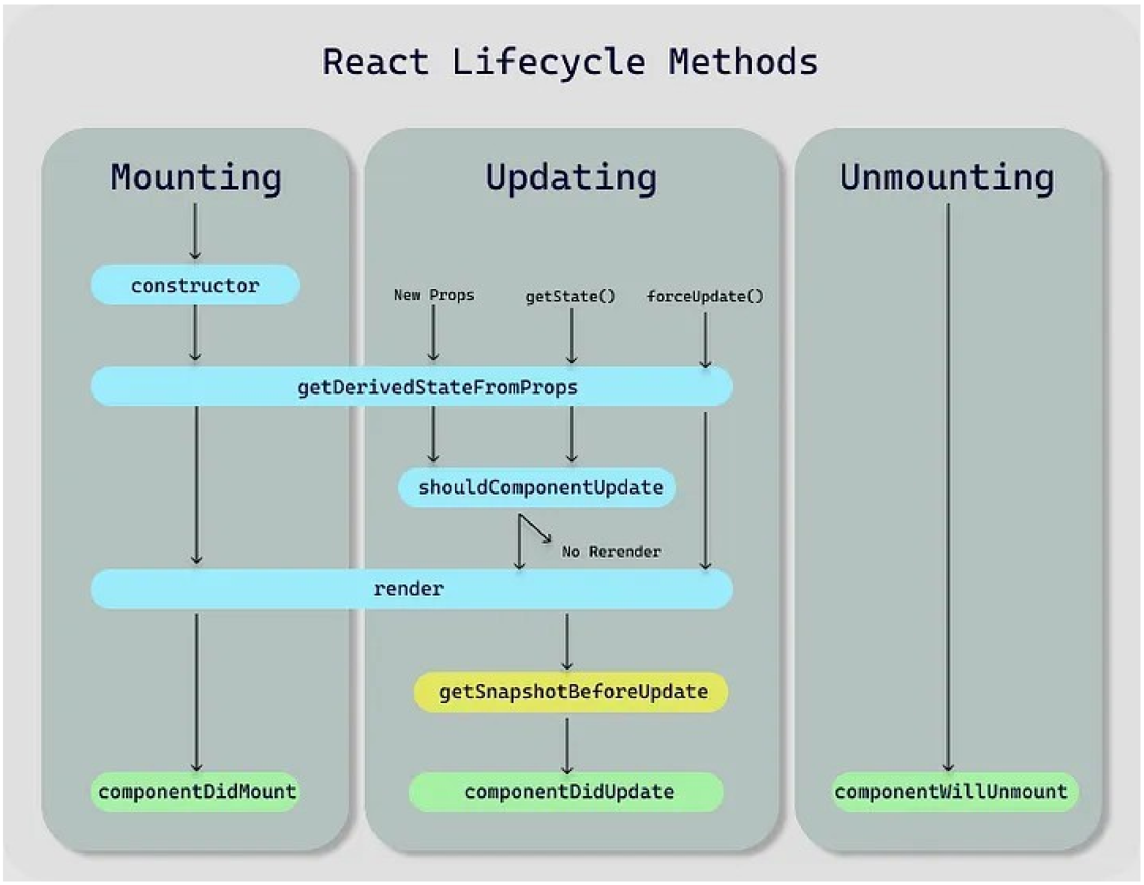
A virtual DOM object is the same as a real DOM object, except that it is a lightweight copy. This means that it cannot manipulate on-screen elements. Moreover, upon any change of a property, it only updates the corresponding nodes and not the entire tree. That makes it a quick and efficient alternative.



React Native LifeCycle Methods

All React class components have their own phases.

When an instance of a component is being created and inserted into the DOM, it gets properties, or props, and from now on they can be accessed using *this.props*.



A component's lifecycle can be divided into 4 parts:

- **Mounting**— an instance of a component is being created and inserted into the DOM.
- **Updating** — when the React component is born in the browser and grows by receiving new updates.
- **Unmounting**—the component is not needed and gets unmounted.
- **Error handling**—called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

Lifecycle Phases

Mounting
These methods are called in the following order when an instance of a component is being created and inserted into the DOM:

- constructor()
- static getDerivedStateFromProps()
- render()
- componentDidMount()

Updating

An update can be caused by changes to props or state. These methods are called in the following order when a component is re-rendered:

- static getDerivedStateFromProps()
- shouldComponentUpdate()
- render()
- getSnapshotBeforeUpdate()
- componentDidUpdate()

Unmounting

This method is called when a component is removed from the DOM:

- componentWillUnmount()

Error Handling :

These methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

- static getDerivedStateFromError()
- componentDidCatch()

constructor()

the constructor method is called before mounting to the DOM and rendering.

- constructor initialize state and bind event handler methods within the constructor method.
- This is the first part of the lifecycle and is only called when it is explicitly declared,
- so there is no need to declare it in every component you create.
- If you need to make any additional properties or subscriptions in this method, you should use componentDidMount().

- I will introduce it later on as we are going through each method the in order they are invoked.

static getDerivedStateFromProps() :

- Instead of calling setState, getDerivedStateFromProps simply returns an object containing the updated state.
- This function is rarely used and this function has no properties - this is intentional.

  getDerivedStateFromProps may be called multiple times for a single update, can use componentDidUpdate, which executes only once after the component updates.

- You can either return an object to update the state of the component or return null to make no updates
- this method allows a component to update its internal state in response to a change in props. The component state reached in this manner is referred to as a derived state.

Render()

- The render() method is responsible for generating the component's virtual DOM representation based on its current props and state.
- It is called every time the component needs to be re-rendered, either because its props or state have changed, or because a parent component has been re-rendered.
- after the static getDerivedStateFromProps method is called, the next lifecycle method in line is the render method.
- The render method must return a React Native component (JSX element) to render (or null, to render nothing).

Component DidMount()

- The componentDidMount() method is called once the component has been mounted into the DOM.
- It is typically used to set up any necessary event listeners or timers, perform any necessary API calls or data fetching, and perform other initialization tasks that require access to the browser's DOM API.

Updating

Each time something changes inside our component or parent component, when the state or props are changed, the component may need to be re-rendered. In simple terms, the component is updated.

static getDerivedStateFromProps()

Firstly, the static getDerivedStateFromProps method is invoked. That's the first method to be invoked for updating.
this method is invoked in both the mounting and updating phases.
shouldComponentUpdate()

- By default, or in most cases, you want a component to re-render when the state or props change.
- But you do have control over this behaviour.
- Here is the moment React decides whether we *should update a component* or not.
- Within this lifecycle method, you can return a boolean and control whether the component gets re-rendered or not, i.e upon a change in the state or props.
- This lifecycle method is mostly used for performance optimisation measures.

ComponentDidUpdate()

- componentDidUpdate() is invoked immediately after updating occurs.
- This method is not called for the initial render.
- This is also a good place to do network requests as long as you compare the current props to previous props
- (e.g. a network request may not be necessary if the props have not changed).

Unmounting

componentWillUnmount()

- componentWillUnmount() is invoked immediately before a component is unmounted and destroyed.
- Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount().

static getDerivedStateFromError()

- Whenever an error is thrown in a descendant component, this method is called first, and the error thrown is passed as an argument.
- Whatever value is returned from this method is used to update the state of the component.

UNIT -III

HYBRID APP DEVELOPMENT

Hybrid Web App, Benefits of Hybrid App, Criteria for creating Native App, Tools for creating Hybrid App, Cons of Hybrid App, Popular Hybrid App Development Frameworks, Ionic, Apache Cordova.

HYBRID WEB APP

What is hybrid application:

- A hybrid application is a software application that combines elements of both **native apps** and **web applications**.
- It is the combination of web apps and native apps which needs to be downloaded inside your devices like native apps but the program that are used to build the Hybrid application are written in HTML, CSS and JavaScript.
- The browser of our devices access is HTML, JavaScript and native APIs to the particular hardware.
- It can run both online and offline, if the hybrid software does not depend on data from the database then it can be use offline.
- It runs across various platform but need to deploy the app's wrapper.
- Example of hybrid application: Uber, Ola, Twitter etc.

What is Native app:

- Native apps are created by software programs (like java, kotlin, ruby etc) that runs on the particular devices and platforms.
- We need to download the native apps from the app stores (Google play, Apple's store) as it does not run in the browser Unlike our smartphones where each application after installation from the app stores have access to an icon on the screen of our device home screen.
- Native apps are developed specifically for one platform and can accessed all the features of our devices like camera, file manager, contacts, GPS etc.
- They are various platform on which we can build Native apps like desktop, smartphones, smartwatch etc.
- Native apps can work offline by using system notification.
- Example of Native apps: Facebook, WhatsApp etc.

What is web application:

- Web application is a software or program that runs and accessible using web browser unlike Native app which run on particular devices.
- We don’t need any particular SDK for developing web application. Frontend part is mostly created using HTML, CSS, JavaScript, bootstrap etc. and backend part could use MEAN stack, Hibernate etc.
- Unlike Native app, web application cannot be installed as it run inside the browser.
- Web application are connected through the server. The server requires bandwidth which helps web application to run on the browser all the time.
- Client displays the data of web application and take very little disk space on the client side. If the server connection get lost then the whole data may be lost.
- Example of web application: MakeMyTrip, Oyo, Flipkart, Amazon etc.

Features of hybrid applications

Hybrid applications features include the following:

- the ability to function whether the device is connected or not;
- integration with the mobile device's file system;
- integration with web-based services; and
- an embedded browser to improve access to dynamic online content.

How hybrid applications work

Hybrid apps work in the same manner as web apps, but are downloaded to the device like native apps. Similar to web apps, developers typically write hybrid apps in HTML5, CSS and JavaScript. Hybrid apps run code inside a container. The device's browser engine renders HTML, JavaScript and native APIs to access device-specific hardware.

Although a hybrid app will typically share similar navigation elements as a web app, whether the application can work offline depends on its functionalities. If an application does not need support from a database, developers can make it function offline.

Hybrid application pros and cons

Pros of hybrid apps include the following:

- will operate on different platforms;
- faster build time compared to native apps;
- cheaper to develop compared to building two versions of a native app for two different platforms;
- easier to launch patches and updates; and
- can work online and offline.

Some cons, however, include the following:

- Variations due to leaning development on one platform may occur -- for example, if a development team leans their work on one platform, another supported platform may lack in quality or suffer from bugs.
- The appearance of an application may vary from platform to platform.
- Developers need to test the application on a range of devices to ensure proper operation.
- User experience (UX) may fail if the user interface (UI) isn't similar to and well enough designed to what browsers the user is used to.

Hybrid vs. native vs. web

Developers build native applications specifically for the platform they are installed on. Native apps can take advantage of a mobile device's hardware, including the accelerometer, GPS and camera. Developers write native apps in the same language the platform's operating system is written in. For example, a native iOS app should be written in Objective-C and Swift.

Web applications are commonly written in JavaScript, HTML and CSS. Users don't need to download web applications; instead, they access them through a device's web browser. Web applications do not have the ability to leverage the hardware on a chosen platform.

Various frameworks used for Hybrid application:

1. **React Native framework:**
   - It is the most popular framework for hybrid application development.
   - It supports various IDE and tools for the development.
   - One of the best thing about React Native is that you can see the result of the code.
   - Because of the faster result, it is time efficient.
2. **Flutter:**
   - It is very easy to use and implement. For the developer who is new can easily get their hands on it.
   - Flutter bear various languages thus it helps the developer to use the language of their own on various platform.
3. **Ionic:**
   - It is best used for mobile app development.
   - It uses HTML, CSS and JavaScript.
   - It is used open source HTML5 development platform.
   - It uses single database for the development of hybrid application.
4. **jQuery Mobile:**
   - It is the JavaScript framework fully dependent on the plugins available in JavaScript like Content Slider, Image Slider, Pop-up Boxes, etc.
   - It is easier to implement as compared to other JavaScript libraries.
   - In this very less code is required to program.
5. **Appcelerator Titanium:**
   - The benefit to use Appcelerator Titanium is that it uses its own API i.e. it has independent API that easily access device hardware.
   - It can be reused across different platforms and apps.
   - It receives UI component.

CRITERIA FOR CREATING NATIVE APP

The application enables users to effortlessly create, edit, and delete notes, providing an uncomplicated yet impactful introduction to React Native's mobile app development capabilities.

Prerequisites:

- Introduction to React Native
- Introduction React Native Components
- React Native State
- React Native Props
- Expo CLI
- Node.js and npm (Node Package Manager)

Steps to install & configure React Native:

Steps to Create React Native Application:

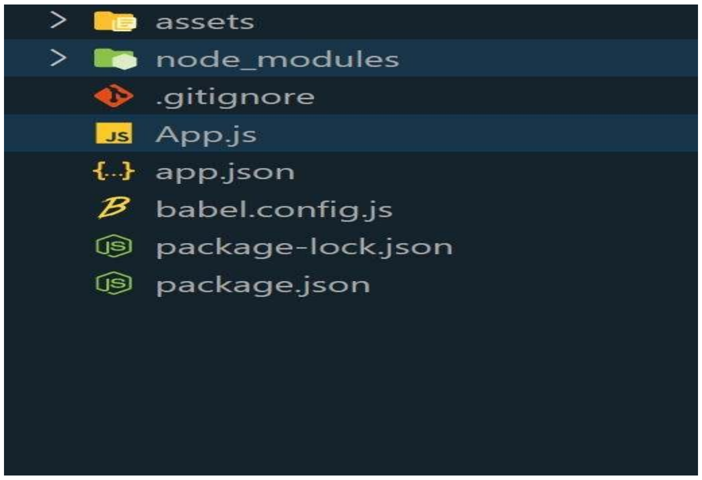
Step 1: Create a react native application by using this command:

```
npx create-expo-app basicNotes-app
```

Step 2: After creating your project folder, i.e. basicNotes-app, use the following command to navigate to it:

```
cd basicNotes-app
```

Project Structure:



```
> assets
> node_modules
  .gitignore
  App.js
  app.json
  babel.config.js
  package-lock.json
  package.json
```

Approach:

This code snippet in React Native allows you to build a simple notes app effortlessly. It effectively manages the state using useState, enabling you to handle note data, editing functionality, and modal visibility smoothly. The app's interface presents a scrollable list of notes, each featuring a title. By utilizing modals with title and content inputs, users can e-asily add, edit, and delete notes.

Example:

Step 3: Open **App.js** file, open it and paste the source code into it.

```
import React, { useState } from "react";
import {
    View,
    Text,
    TextInput,
    Button,
    ScrollView,
    TouchableOpacity,
    Modal,
    StyleSheet,
} from "react-native";

const App = () => {

    // State variables
    // Array to store notes
    const [notes, setNotes] = useState([]);

    // Selected note for editing
    const [selectedNote, setSelectedNote] = useState(null);

    // Note title
    const [title, setTitle] = useState("");

    // Note content
    const [content, setContent] = useState("");

    // Modal visibility state
    const [modalVisible, setModalVisible] = useState(false);

    // Function to handle saving a note
    const handleSaveNote = () => {
        if (selectedNote) {

            // If a note is selected, update it
            const updatedNotes = notes.map((note) =>
                note.id === selectedNote.id
                  ? { ...note, title, content }
                  : note
            );
            setNotes(updatedNotes);
            setSelectedNote(null);
        } else {

            // If no note is selected, add a new note
            const newNote = {
                id: Date.now(),
                title,
                content,
            };
            setNotes([...notes, newNote]);
        }
        setTitle("");
        setContent("");
        setModalVisible(false);
    };

    // Function to handle editing a note
    const handleEditNote = (note) => {
        setSelectedNote(note);
        setTitle(note.title);
        setContent(note.content);
        setModalVisible(true);
    };

    // Function to handle deleting a note
    const handleDeleteNote = (note) => {
        const updatedNotes = notes.filter(
            (item) => item.id !== note.id
        );
        setNotes(updatedNotes);
        setSelectedNote(null);
        setModalVisible(false);
    };

    return (
        <View style={styles.container}>
            {/* Title */}
            <Text style={styles.title}>My Notes</Text>

            {/* List of notes */}
            <ScrollView style={styles.noteList}>
                {notes.map((note) => (
                    <TouchableOpacity
                        key={note.id}
                        onPress={() => handleEditNote(note)}
            >
                <Text style={styles.noteTitle}>
                    {note.title}
                </Text>
            </TouchableOpacity>
        ))}
    </ScrollView>

    {/* Add Note button */}
    <TouchableOpacity
        style={styles.addButton}
        onPress={() => {
            setTitle("");
            setContent("");
            setModalVisible(true);
        }}
    >
        <Text style={styles.addButtonText}>
            Add Note
        </Text>
    </TouchableOpacity>

    {/* Modal for creating/editing notes */}
    <Modal
        visible={modalVisible}
        animationType="slide"
        transparent={false}
    >
        <View style={styles.modalContainer}>
            {/* Note title input */}
            <TextInput
                style={styles.input}
                placeholder="Enter note title"
                value={title}
                onChangeText={setTitle}
            />

            {/* Note content input */}
            <TextInput
                style={styles.contentInput}
                multiline
                placeholder="Enter note content"
                value={content}
                onChangeText={setContent}
            />

            {/* Buttons for saving, canceling, and deleting */}
            <View style={styles.buttonContainer}>
                <Button
                    title="Save"
```

```
                            onPress={handleSaveNote}
                            color="#007BFF"
                        />
                        <Button
                            title="Cancel"
                            onPress={() =>
                                setModalVisible(false)
                            }
                            color="#FF3B30"
                        />
                        {selectedNote && (
                            <Button
                                title="Delete"
                                onPress={() =>
                                    handleDeleteNote(
                                        selectedNote
                                    )
                                }
                                color="#FF9500"
                            />
                        )}
                    </View>
                </View>
            </Modal>
        </View>
    );
};

const styles = StyleSheet.create({
    container: {
        flex: 1,
        padding: 40,
        backgroundColor: "#e6e6e6",
    },
    title: {
        fontSize: 24,
        fontWeight: "bold",
        marginBottom: 10,
        color: "#333",
    },
    noteList: {
        flex: 1,
    },
    noteTitle: {
        fontSize: 15,
        marginBottom: 10,
        fontWeight: "bold",
        color: "black",
        backgroundColor: "white",
        height: 40,
        width: "100%",
        padding: 10,
        borderRadius: 8,
    },
    addButton: {
        alignItems: "center",
        justifyContent: "center",
        backgroundColor: "#007BFF",
        paddingVertical: 12,
        borderRadius: 5,
        marginTop: 10,
    },
    addButtonText: {
        color: "white",
        fontSize: 16,
        fontWeight: "bold",
    },
    modalContainer: {
        flex: 1,
        padding: 50,
        backgroundColor: "white",
    },
    input: {
        borderWidth: 1,
        borderColor: "#E0E0E0",
        padding: 10,
        marginBottom: 10,
        borderRadius: 5,
    },
    contentInput: {
        borderWidth: 1,
        borderColor: "#E0E0E0",
        padding: 10,
        marginBottom: 20,
        borderRadius: 5,
        height: 150,
        textAlignVertical: "top",
    },
    buttonContainer: {
        flexDirection: "row",
        justifyContent: "space-between",
    },
});

export default App;
```

Step 4: Go to the Terminal and type the following command to run the react native application.

```
npx expo start
```

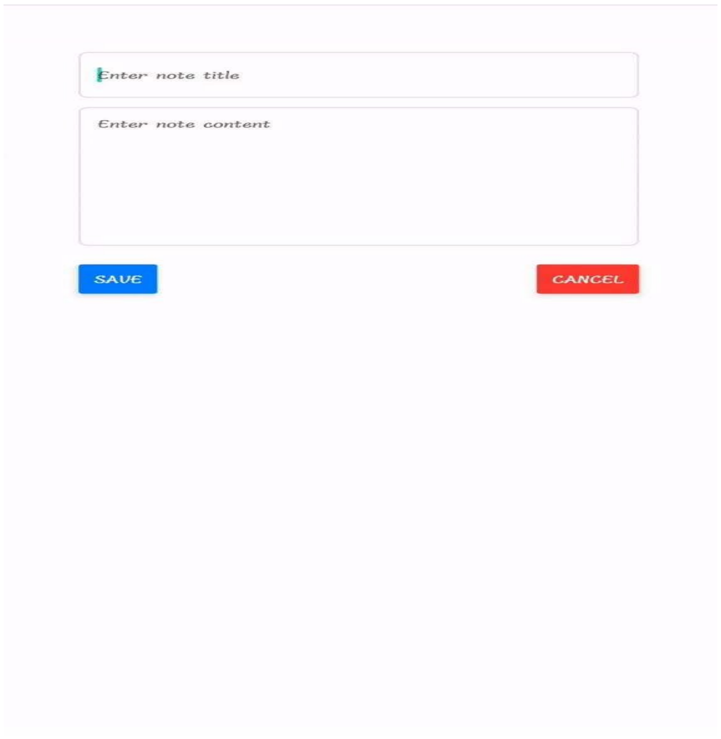
To run on Android:

```
npx react-native run-android
```

To run on Ios:

```
npx react-native run-ios
```

Output:



React Native is a framework developed by Facebook for creating native-style apps for iOS & Android under one common language, JavaScript. Initially, Facebook only developed React Native to support iOS. However, with its recent support of the Android operating system, the library can now render mobile UIs for both platforms.

Prerequisites:

- Basic knowledge of ReactJs.
- Html, CSS, and javascript with ES6 syntax.
- NodeJs should be installed in your system.
- Jdk and android studio for testing your app on the emulator

In this article, we will show an Activity Indicator in react native using **react-native-paper** library . we will display how a text is getting updated after 6 seconds and the activity indicator disappears.

Creating React Native App:

Step 1: Create a react-native project :

npx react-native init DemoProject

Step 2: Now install react-native-paper

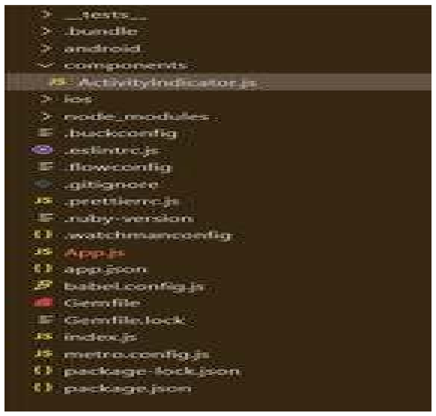
npm install react-native-paper

Step 3: Start the server

npx react-native run-android

Now go to your project and create a components folder. Inside the components folder, create a file ActivityIndicator.js

Project Structure: The project should look like this:



```
> __tests__
> .bundle
> android
v components
    ActivityIndicator.js
> ios
> node_modules
  .buckconfig
  .eslintrc.js
  .flowconfig
  .gitignore
  .prettierrc.js
  .ruby-version
  .watchmanconfig
  App.js
  app.json
  babel.config.js
  Gemfile
  Gemfile.lock
  index.js
  metro.config.js
  package-lock.json
  package.json
```

To display an ActivityIndicator using react-native-paper, we have to pass **animating** props as true. To hide it, just set the animating prop to false. To change the color of the indicator, we can pass **color** props and change the size of the indicator, this library provides **size** props. It can be any one of the following:

Size = 'small' | 'large' | number

Example: We will use useState, useEffect hooks of react-native to update the state of the components.

- ActivityIndicator.js

```
import React, { useState, useEffect } from "react";

import { Text, View, StyleSheet } from 'react-native';

import { ActivityIndicator, } from "react-native-paper";




const ActivityIndicatorExample = () => {

    const [text, setText] = useState('');

    const [animate, setAnimate] = useState(true);



    useEffect(() => {

        setTimeout(() => {

            setText("Value updated successfully");

            setAnimate(false);

        }, 6000);

    })

    return (
        <View style={styles.activityI}>

            <ActivityIndicator animating={animate}

                color="red" size="large" />

            <Text style={styles.text}>{text}</Text>

        </View>

    )

}



export default ActivityIndicatorExample;

const styles = StyleSheet.create({

    activityI: {

        alignContent: "center",

        margin: 50

    },

    text: {

        fontSize: 30,
        fontWeight: "bold"

    }

})
```

- App.js

```
import React from 'react';

import { Text, View, StyleSheet, Alert } from 'react-native';

import ActivityIndicatorExample from './components/ActivityIndicator';



const App: () => Node = () => {

    return (



        <View>

            <ActivityIndicatorExample />

        </View>
    );

};



export default App;
```

Save it and restart the server by the following command:

```
npx react-native run-android
```

Popular Hybrid App Development Frameworks

Hybrid mobile app frameworks are chart-topping tools for building hybrid apps. They are defined as making **fast work of programming apps**. These frameworks include APIs, libraries of code, and other features to make coding mobile apps easier and faster.

1. React Native
2. Ionic Framework
3. NativeScript
4. Quasar
5. Kendo UI
6. Framework7
7. Aurelia
8. Onsen UI
9. Ext JS
10. Axway Appcelerator
11. Svelte Native
12. Xamarin

1. React Native

React Native

With React Native, you can build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components.

Using this framework, you don't build a "mobile web app", an "HTML5 app", or a "hybrid app". **You build a real mobile app that's indistinguishable from an app built using Objective-C or Java**. React Native uses the same fundamental UI building blocks as regular iOS and Android apps. You just put those building blocks together using JavaScript and React.

If you're already creating apps with React Native, ensure you protect their code by following our guide concerning React Native protection with Jscrambler integration.

2. Ionic Framework

ionic

The Ionic Framework is a complete open-source SDK for hybrid mobile app development. It provides tools and services for hybrid mobile app development using Web technologies like CSS, HTML5, and Sass.

Apps can be built with these Web technologies and then distributed through native app stores to be installed on devices by leveraging Cordova.

This framework is **100% free and open-source**. It is licensed by MIT and powered by a massive worldwide community. They have over 120 native device features like Bluetooth, HealthKit, Finger Print Auth, and more with Cordova and PhoneGap plugins and TypeScript extensions.

You can use their CLI to create, build, test, and deploy your Ionic apps on any platform. The framework has the Ionicons icon pack with hundreds of the most common app icons. And you can develop your apps with Live Reload because compiling and redeploying your app at every step of development is for chumps. There are more useful features like deep linking, AoT Compiling, and a custom animation API.

Ionic is framework-agnostic and **has official support for React, Preact, Angular, and Vue**, as well as for Web Components.

3. Native Script

Originally created by Progress, NativeScript apps are built using JavaScript or by using any language that converts to JavaScript, such as TypeScript.

This hybrid mobile app framework **has deep integration with modern Angular versions**, including full-stack features like integration with the Angular CLI, router support, and code generation. It includes integration with Vue via a community-developed plugin, which enables using the Vue CLI, Vuex, and other nice Vue.js features.

What does a hybrid mobile app built with NativeScript look like?
Well, mobile applications built with NativeScript are actually fully native apps and use the same APIs as if they were developed in Xcode or Android Studio. This means you get a platform-native UI without WebViews and native performance.

Additionally, software developers can repurpose third-party libraries from Cocoapods, Android Arsenal, Maven, and npm.js in their mobile applications without the need for wrappers.

4. Quasar

The Quasar Framework is powered by Vue.js and enables developers to write code once and deploy it simultaneously as a website (SPA, PWA, SSR), mobile app (iOS, Android), and desktop application (using Electron) using a single code base.

Out of the box, Quasar brings state-of-the-art UI, following Google Material guidelines. It also offers, while keeping a small performance overhead:

- HTML/CSS/JS minification;
- Cache busting;
- Tree shaking;
- Source mapping;
- Code-splitting and lazy loading;
- ES6 transpiling;
- Linting code; and
- Accessibility features.

Actually, the creators of this hybrid mobile app framework claim that it is "the most performance-focused framework".

With the Quasar CLI, developers also benefit from additional features such as hot-reloading. One of the framework's most praised benefits is its very thorough documentation and active community.

It's worth mentioning that Quasar is **100% free, open-source, and licensed under MIT**.

5. Kendo UI

Kendo UI provides a very extensive collection of JavaScript UI components with libraries for jQuery, Angular, React, and Vue for creating hybrid mobile apps.

Powered by Progress Telerik (the same parent organization as NativeScript). It comes packed with dozens of ready-to-use widgets for jQuery, Angular, React, and Vue.

Kendo UI is **focused on allowing development teams to quickly build high-performance hybrid mobile apps** with excellent performance.

This hybrid app framework is open-source, but it is aimed at enterprise customers. Thus, there are no free versions available. One of the key selling points of Kendo UI is that you get world-class support. It also gives the guarantee that every component has been tested by their strict QA process.

Among the most noteworthy Kendo UI clients, we find HP, NASA, and over 140,000 other companies across the world. It is definitely a hybrid app development framework to consider if you're looking for an enterprise-grade solution with dedicated support.

6. Framework7

Framework7 is a free and open-source mobile HTML framework to develop hybrid mobile apps, web apps, and progressive web apps (PWAs) with a native look and feel. Plus, it can be paired with extra tools like Electron and NW.js, allowing you to build native desktop apps.

This hybrid app framework can be an indispensable prototyping tool to show working app prototypes as soon as possible, in case you need to. It is focused on iOS and Google Materials design to bring the best experience and simplicity.

Much like similar hybrid app frameworks, Framework7 offers a rich ecosystem of components for popular JS frameworks like Vue, React, and Svelte (read the blog post about Svelte vs. React and which one to choose when building the same web app).

Some useful features provided by Framework7 are:

- Native scrolling;
- Library-agnostic;
- Pages transition animation;
- Multiple views support;
- Hardware-accelerated animations via CSS3;
- Route pages by using a combination of XHR, caching, browser history, and preloading.

7. Aurelia

Aurelia characterizes itself as a "collection of Modern JavaScript modules, which, when used together, function as a powerful platform for building browser, desktop, and mobile applications". As such, any of Aurelia's modules can be used on their own in any JavaScript or Node.js project.

With a focus on clean yet powerful code, Aurelia is **especially aimed at those who prefer to use vanilla JavaScript or TypeScript**. In fact, they go so far as claiming that they are "the only framework that lets you build components with plain, vanilla JavaScript/TypeScript".

Still, because it strictly follows web standards, it can readily be integrated with any framework or library out there.

Some of the most powerful Aurelia modules include metadata, dependency injection, binding, templating, and routing. Its ecosystem includes plugins for state management, internationalization, and validation, as well as tools like a CLI, Chrome debugger, and VS Code plugin.

Because Aurelia is based on a high-performance reactive system, it batches DOM updates faster than virtual-DOM-based frameworks.

This hybrid mobile app framework is 100% free to use, open-source, and licensed under the MIT license.

8. Onsen UI

Onsen UI has quickly grown in adoption since its release in 2013. It is an open-source framework under the Apache v2 license.

Onsen UI is a framework-agnostic UI component, you can choose and switch among the frameworks: AngularJS, Angular, React, and Vue.js, or go pure JavaScript to build your hybrid apps.

The framework architecture **consists of three layers**:

1. CSS components;
2. Web components; and
3. Framework bindings.

It also features a large collection of ready-to-use, responsive, out-of-the-box components.

This framework is very easy to use, flexible, has semantic markup components, and is free to use for commercial projects.

9. Ext JS

Ext JS is an enterprise-grade product for building cross-platform, end-to-end mobile web apps with HTML5 and JavaScript. It is **especially suited to building data-intensive**, cross-platform web and **mobile applications**.

This hybrid mobile app framework isn't afraid to state that it is "The Best JavaScript Framework In The World". And in fact, among the 10,000 companies using the framework, we find Apple, Adobe, Cisco, Nvidia, and many other global enterprises.

For individual developers and freelancers, Ionic, Onsen UI, or Framework7 will be a better choice - but, for enterprise applications, Ext JS leads the way.

ExtJS scores highly against its competitors by providing a native look and feel across all of the platforms it supports. It helps create high-performance hybrid mobile apps with a near-native experience and packs ready-to-use widgets with a native look and feel for all leading platforms, including iOS, Android, Windows Phone, and Blackberry.

The framework also features a drag-and-drop HTML5 visual application builder with tons of ready-to-use templates. It has built-in support for Angular and React.

10. Axway Appcelerator

Appcelerator (also known as Appcelerator Titanium) is a JavaScript-based hybrid app framework. It is cross-platform, with full support for Android and iOS. In the end, the compiled code is a combination of native and JavaScript that improves performance for hybrid mobile app development. It has integrations available for Angular and Vue.

Appcelerator is a good solution for creating hybrid mobile apps. To get started with Appcelerator, download Titanium Studio. The Titanium SDK is equipped with a number of mobile platform APIs and Cloud services to use as an app backend. It comes with platform-independent APIs, which make it easier to access phone hardware.

Appcelerator uses Hyperloop to join the cross-platform power of Titanium with direct access to any native API using JavaScript. The framework has both free and paid plans.

11. Svelte Native

Svelte Native is the youngest framework on this list for hybrid app development. With the Svelte framework quickly growing in popularity among web developers thanks to its simplicity, the mobile framework quickly followed in its footsteps.

Svelte Native is powered by Svelte and NativeScript.

While mobile app frameworks like React Native perform the bulk of their work on the actual mobile device, Svelte Native takes a new approach by shifting that work into a compile step at build time.

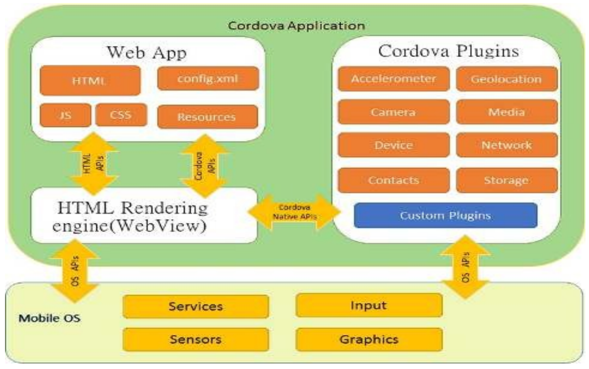
IONIC FRAMEWORK

It is to be used to create powerful web apps and deployed into native environment being Android or iOS.

There is no need to learn any android, swift or objective C(earlier used for creating iOS apps) to create apps i.e if you don't know any one these technologies but still wants to create stunning apps and want your name to be published on the play store or iStore, then it is the one of the great way. The prerequisites for this is that if you are familiar with HTML, CSS and JavaScript then you are to good to go. If you know Angular that is of course a plus point.

Now the basic question that some of the minds may get struck with is that how can an app being designed in web technology can be run on native environment like Android or iOS. This question is answered by Apache Cordova(also called as PhoneGap). It is mobile application development framework or simply it is used to deploy web apps in such way that they fit into the native environment which we want out of it as the end product.

The sample image below clearly indicates the working of cordova



Another popular opensource framework for building hybrid mobile apps, Ionic comes with a library of HTML, CSS, and Javascript gestures, tools, and components, all mobile-optimized. Ionic was built with SASS and is actually optimized for AngularJS so that you can easily build a highly interactive app for dynamic views. Ionic then allows you to push your code through Cordova once ready.

Steps to create your first ionic application

1. **Download and Install Iconic :**

   The first step is to download and install the ionic on your system. Be clear here that since is ionic is Npm module so therefore it can only be installed through Npm i.e node package manager. In other words, you must have already Nodejs installed on your system. Download Nodejs from here.. I would recommend to install the LTS version since it is stable. After installing Nodejs your can install ionic since Npm will be installed automatically. Here I will show the installation process. I have used ubuntu but if you are on Windows then don't worry I will guide the windows fans as well :). For only linux users, write the below the command on your terminal to update the repositories below installing ionic.Windows users do not need to do anything.

   ```
   sudo apt-get update
   ```

   Then install the ionic by using the commands which are different in linux or windows. For linux users

   ```
   sudo apt-get -g install ionic
   ```

   Since it is global installation so you need to write the -g and sudo For Windows users follow the command below.

   You may need to run the command prompt as admin

   ```
   npm install ionic
   ```

2. **After installation run the following command :**

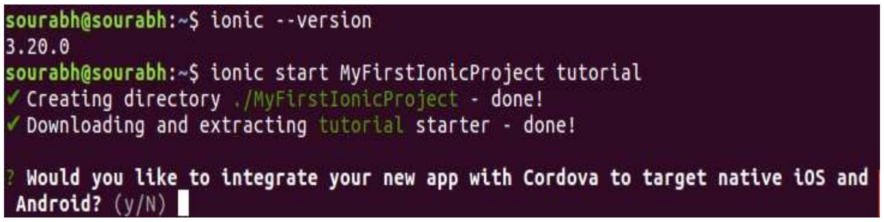
   ```
   ionic --version
   ```

   This is basically to test whether ionic has been successfully installed on your system
   Next step is to make your ionic app by using the following command

   ```
   ionic start name_of_project template_name
   ```

The command is the basic syntax to follow. It is same for both windows as well as linux users.

***The image below clearly depicts the process** :*

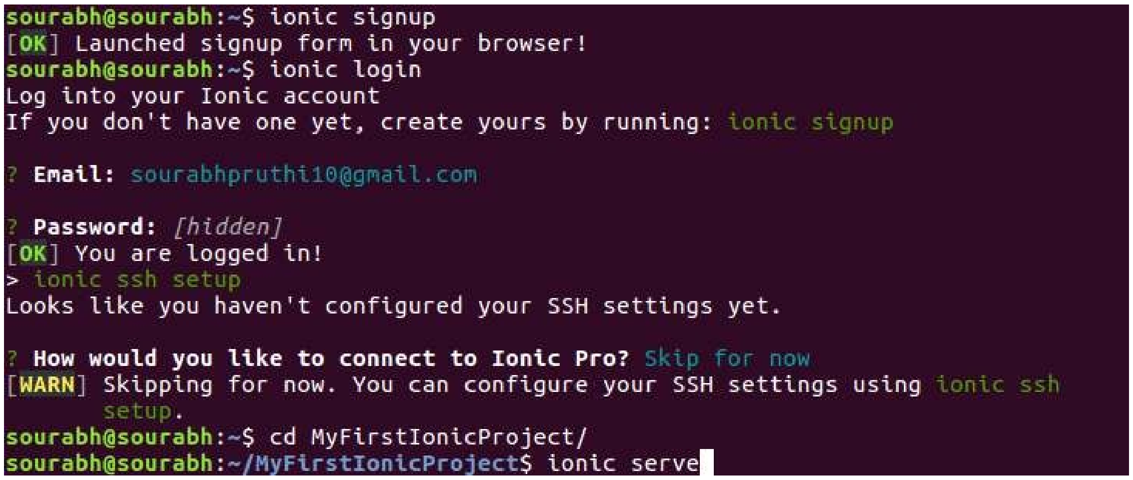


```
sourabh@sourabh:~$ ionic --version
3.20.0
sourabh@sourabh:~$ ionic start MyFirstIonicProject tutorial
✓ Creating directory ./MyFirstIonicProject - done!
✓ Downloading and extracting tutorial starter - done!

? Would you like to integrate your new app with Cordova to target native iOS and
 Android? (y/N)
```

Let me explain the above command. Start here, basically tells to create a new ionic app, next is the name of the app and then followed by the starter template. There are various other template being provided by ionic depending upon the requirement like tab, sidemenu, blank etc. It is the basic layout of the app that is going to be built further. After running the command it will also ask whether you want to integrate the app with cordova you can simply type yes

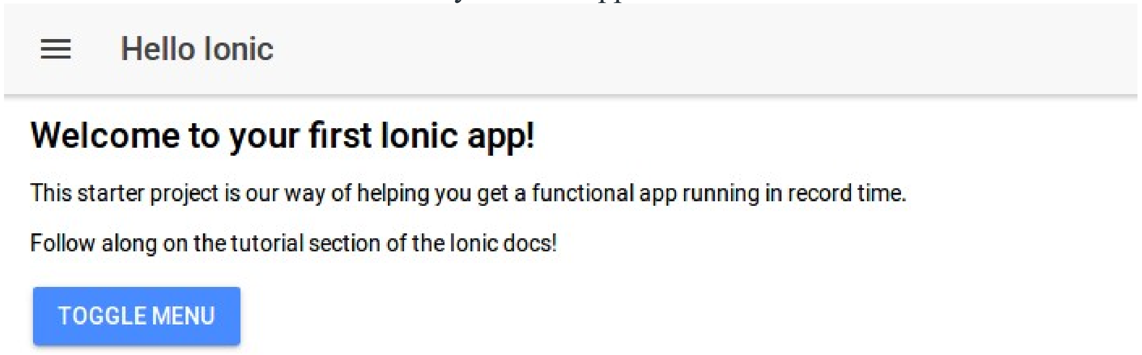
3. **Project Directory:** The last step is to move through the project directory by typing the following command

cd project_name



```
sourabh@sourabh:~$ ionic signup
[OK] Launched signup form in your browser!
sourabh@sourabh:~$ ionic login
Log into your Ionic account
If you don't have one yet, create yours by running: ionic signup

? Email: sourabhpruthi10@gmail.com

? Password: [hidden]
[OK] You are logged in!
> ionic ssh setup
Looks like you haven't configured your SSH settings yet.

? How would you like to connect to Ionic Pro? Skip for now
[WARN] Skipping for now. You can configure your SSH settings using ionic ssh
       setup.
sourabh@sourabh:~$ cd MyFirstIonicProject/
sourabh@sourabh:~/MyFirstIonicProject$ ionic serve
```

Voila!! You have created your first app.



Hello Ionic

Welcome to your first Ionic app!

This starter project is our way of helping you get a functional app running in record time.

Follow along on the tutorial section of the Ionic docs!

Pros and Cons of Ionic framework

Advantages

Develop once deploy anywhere

- As it is being built over it is very useful in creating powerful and robust applications.
- Quick development, low cost of maintenance

Disadvantages

- Bad Performance as compared to native app.
- High skills requirement for complex apps
- Built-in navigation can be complex

APACHE CORDOVA

Apache Cordova is an open-source platform for developing mobile apps through web applications like HTML, CSS, JavaScript. Cordova is very useful to web-developers as they can turn their web pages to a web app with native app functionalities easily using Cordova. This is an extremely helpful feature as normal web apps don't have this functionality.

Cordova is used to making cross-platform mobile applications and provides a wide range of plugins for better functionality of the app which is easy to embed.

Installation

We are installing the Cordova command-line tool. If not already installed follow the steps given below :

1. Download Node.js and install it from here.
2. Using npm utility(Node.js) to install Cordova module

Installation on Linux / macOS

Prefixing the *sudo* command to the *npm* command might be needed to install the utility

```
$ sudo npm install -g cordova
```

Installation on Windows

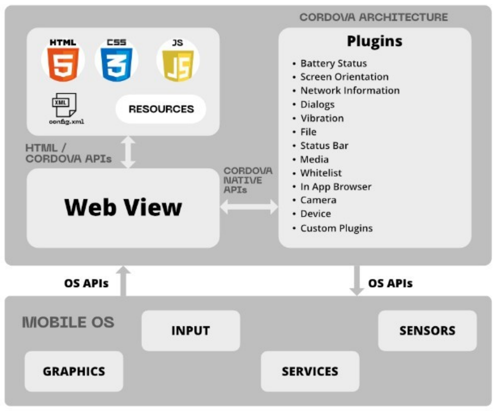
The -g flag tells the npm utility to install Cordova globally

```
C:\>npm install -g cordova
```

Run Cordova in the command line to check if properly installed if installed it should print help text.

Cordova's Application Architecture

Cordova has a high-level design the diagram shown below depicts its architecture



Cordova's application architecture

Web View: This is the user interface of the Cordova application. The applications used are integrated with the web view and the native components(for hybrid apps).
Web App: This is the basic web page layout made using HTML, CSS, JavaScript. This is the core of the Cordova application the web app runs in the web view. The file config.xml is responsible for the information on the app

Plugins

Plugins are one of the best features in Cordova. Integrating plugins adds apps functionality and attractiveness. Cordova maintains a set of plugins called Core Plugins which provides application capabilities like Camera, Battery, File transfer etc. In addition to the core plugins, there are several third-party plugins that provide additional bindings to features. Cordova does not provide any mv framework or widgets. Plugins are necessary for functionality like communication between Cordova and custom native components. Plugins can be searched using the npm command or searched at the link given below

Development Paths

Basically there are two development paths in Cordova each with its own advantages :

1. **Cross-Platform Workflow:** This workflow is centered around the command-line interface(CLI) and mostly used when a developer wants the application to run on different platforms. This workflow has very little need for platform-specific developments. Here the CLI copies assets of different platforms into sub-directories for each of the platforms and has a common interface to apply plugins.

2. **Platform-centered Workflow:** This workflow is centered around lower-level shell scripts for a specific platform and is used when a developer is focused on building an application on a single platform and wants to modify it at a lower level like adding native components to the web-based components. This workflow does not have any high-level tools. If a user wants to modify the application with SDK the Platform-centered workflow is used

Features :

1. Command Line Interface: Used for installing plugins and writing commands to build a Cordova application
2. Cordova Plugins: Many APIs can be used in Cordova to add functionality to a Cordova application
3. Cordova Core Components: A set of components used to build the application

Advantages of using Cordova

1. Easy to use and does not require a lot of time to make an application with Cordova.
2. There is no need to learn a specific development programming language to develop an application.
3. Cordova follows a plugin architecture, many plugins to work with which can be added and modified. We can enable and disable plugins as per our priorities.
4. Is a platform for developing an application that can be used in different platforms — Ubuntu, Windows, Blackberry, etc.

Limitations

1. Not all plugins are compatible with every platform.
2. Hybrid apps are slower than native apps.
3. Not optimum for making an application that requires a large set of data.

UNIT 4
CROSS-PLATFORM APP DEVELOPMENT USING REACT-NATIVE

What is cross-platform mobile development?

Cross-platform mobile development is an approach to developing software applications that are compatible with multiple mobile operating systems (OSes) or platforms. These apps are *platform-agnostic*, meaning they can be used regardless of the OS powering the mobile device.

With cross-platform mobile app development, developers can build applications that can run on different platforms with one single code system. It means the company can release the product faster and with better quality. Since it is compatible with various mobile operating systems, the application can reach a broader audience.

Its rapid development, turnaround time, and cost-effective quality make it very suitable for startups. Building a cross-platform app can help with some common mobile application development challenges.

Benefits of cross-platform mobile development

Cross-platform apps have shareable code that can be reused across multiple platforms. A single codebase speeds up development and cuts development costs, particularly for repetitive tasks like data serialization and API calls. Faster development usually translates to faster time to market.

Adopting the cross-platform approach enables project managers to use their development resources more effectively since they don't have to assign separate resources for developing apps for different platforms. Also, fewer lines of code means there are fewer chances of bugs and security errors, reducing the time and effort required for code testing and maintenance.

Another advantage is that in many cases, developers need to only know standard languages. Development tools and frameworks are available to do most of the heavy lifting. Further, cross-platform apps have a wider reach since they can satisfy the needs of audiences using different OSes and devices.

Drawbacks of cross-platform mobile development

Performance glitches are the most common issue with cross-platform apps. Many such apps have limited functionality since they cannot support many native-only functions of mobile devices, such as advanced graphics. Poor design is another common problem, resulting in poor UX.

That said, improvements to development technologies and frameworks are helping to overcome these issues and create cross-platform apps that have the following characteristics:

- Flexible.
- Adaptable.
- Stable.
- High performing.
- Highly functional.
- Able to deliver good UX.

Popular cross-platform development frameworks

Cross-platform app developers can choose from many mobile app development platforms, each with its own capabilities and benefits. Among the most popular frameworks are the following:

Xamarin. Launched in 2011, Xamarin is an Open Source framework for developing cross-platform and hybrid apps that can work seamlessly on any mobile platform, including Android and iOS. Xamarin uses C# programming and Microsoft's .NET framework. It provides its own integrated development environment (IDE), as well as numerous software development kits (SDKs). The platform was once independent but later acquired by Microsoft. Today, it is available under the MIT License as part of Visual Studio IDE and source code editor.

Flutter. Like Xamarin, Flutter is an open source cross-platform framework. Created by Google, Flutter uses the Dart programming language. It is suitable for building many kinds of cross-platform apps that look native on multiple mobile platforms, especially the following:

- Minimum viable products (MVP).
- Apps that may put a heavy load on the device CPU or GPU.
- Apps that need to share UI components while looking as close to native as possible.

Flutter also incorporates *platform channel* technology that enables developers to create platform-specific code. Additionally, its *hot reload* feature allows them to make code changes and view them instantly.

React Native. Introduced by Facebook in 2015, React Native is suitable for building both hybrid and cross- platform apps. It is based on the React JavaScript library and supports multiple languages, including Java, Swift and C. Some of the benefits of this framework are the following:

- It can render a native-like interface.
- Code is reusable.
- It provides numerous ready-to-apply features.
- It converts the source code into native elements to enhance UX.
- It's easy to use.

Ionic. Ionic provides a simple syntax and a library of HTML, CSS and JavaScript components to develop interactive cross-platform apps. The framework features hardware-accelerated transitions and touch-optimized gestures to improve app speed and performance. Its single shared codebase enables developers to create flexible UIs for all major OSes. Developers can also choose eye-pleasing themes, form controls, add inline overlays and much more.

Sencha. Sencha is a JavaScript grid to build data-intensive cross-platform applications for both web and mobile. It is based on modern web technologies, such as ES6, HTML5, JavaScript and CSS, and provides more than 140 UI components to quickly build apps for mobile devices.

Criteria for creating Cross-platform App:

Creating a cross-platform app involves considering various factors to ensure compatibility and functionality across different operating systems and devices. Here are some criteria to keep in mind:

1. **Platform Compatibility**: Ensure the app works seamlessly across major platforms like iOS, Android, and possibly others like Windows or web browsers.

2. **User Interface (UI) Design**: Design the UI to adapt to different screen sizes, resolutions, and input methods while maintaining consistency and usability across platforms.

3. **Development Framework**: Choose a cross-platform development framework or toolset that supports multiple platforms, such as React Native, Flutter, Xamarin, or Progressive Web Apps (PWAs).

4. **Performance Optimization**: Optimize the app's performance to deliver smooth user experience across different devices and platforms, considering factors like processing power, memory, and network conditions.

5.**Native Features Integration**: Utilize platform-specific features and APIs when necessary to provide users with a native-like experience on each platform.

6.**Code Reusability**: Maximize code reuse across platforms to reduce development time and effort, while still allowing for platform-specific customization when needed.

7.**Testing Strategy**: Implement a comprehensive testing strategy to ensure the app functions correctly and consistently across various platforms, devices, and OS versions.

8.**Security and Compliance**: Address security concerns and comply with platform-specific security guidelines and regulations to protect user data and ensure app acceptance on app stores.

9.**Updates and Maintenance**: Plan for ongoing updates and maintenance to address platform updates, bug fixes, and feature enhancements across all supported platforms.

10. **User Feedback and Iteration**: Gather feedback from users on different platforms to identify issues and areas for improvement, and iterate on the app to enhance its cross-platform compatibility and user experience over time.

FLUTTER:

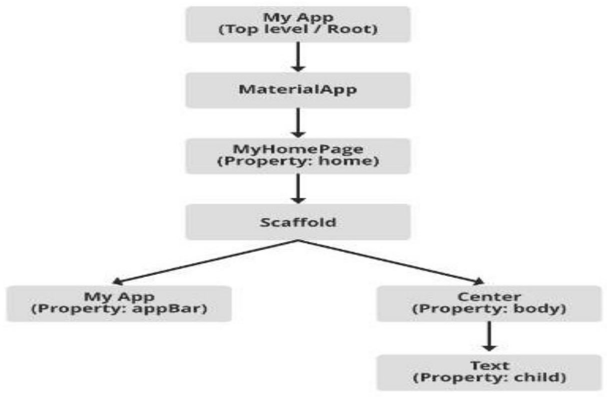
- Flutter is Google's Mobile SDK to build native iOS and Android, Desktop (Windows, Linux, macOS), and Web apps from a single codebase.
- When building applications with Flutter everything towards Widgets – the blocks with which the flutter apps are built.
- They are structural elements that ship with a bunch of material design-specific functionalities and new widgets can be composed out of existing ones too.
- The process of composing widgets together is called composition. The User Interface of the app is composed of many simple widgets, each of them handling one particular job. That is the reason why Flutter developers tend to think of their flutter app as a tree of widgets.

Flutter architecture application mainly consists of:

- **Widgets**
- **Gestures**
- **Concept of State**
- **Layers**

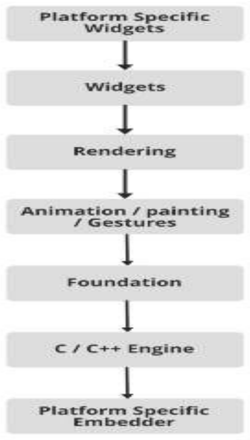
Widgets

1. Widgets are the primary component of any flutter application. It acts as a UI for the user to interact with the application.
2. Any flutter application is itself a widget that is made up of a combination of widgets. In a standard application, the root defines the structure of the application followed by a Material App widget which basically holds its internal components in place. This is where the properties of the UI and the application itself is set.


My App (Top level / Root)
MaterialApp
MyHomePage (Property: home)
Scaffold
My App (Property: appBar)
Center (Property: body)
Text (Property: child)

Layers

1. The Flutter framework is categorized based on its complexity and establishes a hierarchy based on the decreasing level of these complexities. These categories are often called Layers.
2. These layers are built on top of one another. The topmost layer is a widget specific to the operating system of the device (ie, Android or iOS). The second layer consists of the native flutter widgets, which comprise structural UI components, gesture detectors, state management components, etc. This third layer is where all the Ui and state rendering occurs. It is the layer that includes all the visible components of the flutter application. The following layer consists of animations used in transitions, image flow, and gestures. These further go on to the very high level of system design that is not the target of this article. The below diagram gives an overview of the same:


Platform Specific Widgets
Widgets
Rendering
Animation / painting / Gestures
Foundation
C / C++ Engine
Platform Specific Embedder

Gestures

All physical form of interaction with a flutter application is done through pre-defined gestures. Gesture-Detectors are used for the same. It is an invisible widget that is used to process physical interaction with the flutter application. The interaction includes gestures like tapping, dragging, and swiping, etc. These features can be used to creatively enhance the user experiences of the app by making it perform desired actions based on simple gestures.

Concept of State

If you have ever worked with React-js, you might be familiar with the concept of a state. The states are nothing but data objects. Flutter also operates on similar turf. For the management of state in a Flutter application, Stateful-Widget is used. Similar to the concept of state in React-js, the re-rendering of widgets specific to the state occurs whenever the state changes. This also avoids the re-rendering of the entire application, every time the state of a widget changes.

Setting Up Flutter

- **Step 1: Set up the Flutter SDK**
  2. Download the latest SDK
  3. Extract the zip file and place the contained '*flutter*' folder in the desired directory.
- **Step : 2 Set up Android Studio**: Android Studio automatically downloads the development tools required for Flutter to work with Android.
- **Step 3: Set up Visual Studio Code**: Visual Studio Code (or VS Code) is a light code editor that can be used in Flutter development. In this article, VS is used instead of Android Studio as it is lighter and has the minimal required features.

Widgets: Each element on a screen of the Flutter app is a widget. The view of the screen completely depends upon the choice and sequence of the widgets used to build the apps. And the structure of the code of an apps is a tree of widgets.
Category of Widgets:
There are mainly 14 categories in which the flutter widgets are divided. They are mainly segregated on the basis of the functionality they provide in a flutter application.

1. **Accessibility:** These are the set of widgets that make a flutter app more easily accessible.
2. **Animation and Motion:** These widgets add animation to other widgets.
3. **Assets, Images, and Icons:** These widgets take charge of assets such as display images and show icons.
4. **Async:** These provide async functionality in the flutter application.
5. **Basics:** These are the bundle of widgets that are absolutely necessary for the development of any flutter application.
6. **Cupertino:** These are the iOS designed widgets.
7. **Input:** This set of widgets provides input functionality in a flutter application.
8. **Interaction Models:** These widgets are here to manage touch events and route users to different views in the application.
9. **Layout:** This bundle of widgets helps in placing the other widgets on the screen as needed.
10. **Material Components:** This is a set of widgets that mainly follow material design by Google.
11. **Painting and effects:** This is the set of widgets that apply visual changes to their child widgets without changing their layout or shape.
12. **Scrolling:** This provides scrollability of to a set of other widgets that are not scrollable by default.
13. **Styling:** This deals with the theme, responsiveness, and sizing of the app.
14. **Text:** This displays text.

1. **Flutter Framework:**

- **Widgets**: The core building blocks of Flutter's UI, everything in Flutter is a widget, from a simple text to complex layouts[1].
- **Dart Platform**: Flutter apps are written in the Dart language and make use of many of its more advanced features[1].
- **Flutter Engine**: Provides a low-level implementation of Flutter's core API, including graphics (through Skia), text layout, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain[1].
- **Foundation Library**: The base class library and design-specific widgets that interact with the engine[1].

2. Material Components:

- **Actions**: Widgets like FloatingActionButton, IconButton, and others that represent user actions[2].
- **Communication**: Widgets like SnackBar, AlertDialog, and LinearProgressIndicator that handle user communication[2].
- **Containment**: Widgets like Card, BottomSheet, and ListTile that contain content and actions within a structured layout[2].
- **Navigation**: Widgets like AppBar, BottomNavigationBar, and TabBar that handle navigation within the app[2].
- **Selection**: Widgets like Checkbox, Chip, and others that allow the user to make selections[2].

3. Development Tools:

- **DevTools**: A suite of performance and debugging tools for Dart and Flutter.
- **Hot Reload**: Allows developers to see the effects of their changes almost instantly without losing the current application state.

4. Testing & Integration:

- **Unit Tests**: Validate the behavior of a single function, method, or class.
- **Widget Tests**: Validate the behavior of individual widgets.
- **Integration Tests**: Test a complete app or a large part of an app.

5. Compilation Modes:

- **Debug Mode**: Compilation is optimized for fast development and runtime debugging.
- **Release Mode**: Compilation is optimized for deploying the app, with optimizations for performance and size.

6. Flutter Architecture:

- **Layer Model**: Flutter uses a layered approach where each layer is built upon the previous one[3].
- **Reactive User Interfaces**: Flutter's reactive model allows developers to update the UI by changing the state of the app

Widget lifecycle in flutter:

Certainly! In Flutter, understanding the widget lifecycle is crucial for managing the state and behavior of widgets effectively. Here's an explanation of the widget lifecycle:

1. Creation:

- **StatelessWidget**: It's created when its parent widget rebuilds and provides the new configuration. Since it's immutable, it doesn't have a lifecycle beyond the build method.
- **StatefulWidget**: It's created when its parent widget rebuilds and provides the new configuration. The framework then calls `createState()` to create a `State` object.

2. Initialization:

- **initState()**: This is the first method called after the widget is inserted into the widget tree, which makes it the perfect place to initialize data that depends on the context.

3. State Changes:

- **setState()**: When you want to change the internal state of a widget, you call `setState()`, which triggers a call to the `build()` method, resulting in the UI updating to reflect the new state.

4. Building:

- **build()**: This method is called whenever the widget's state changes or when its parent widget rebuilds. It returns a `Widget` that describes the UI based on the current state.

5. Updating:

- **didUpdateWidget()**: Called when the widget's configuration changes. It's a place to respond to any changes in the widget's properties received from the parent.

6. Deactivation:

- **deactivate()**: This method is called when the widget is removed from the tree temporarily.

7. Disposal:

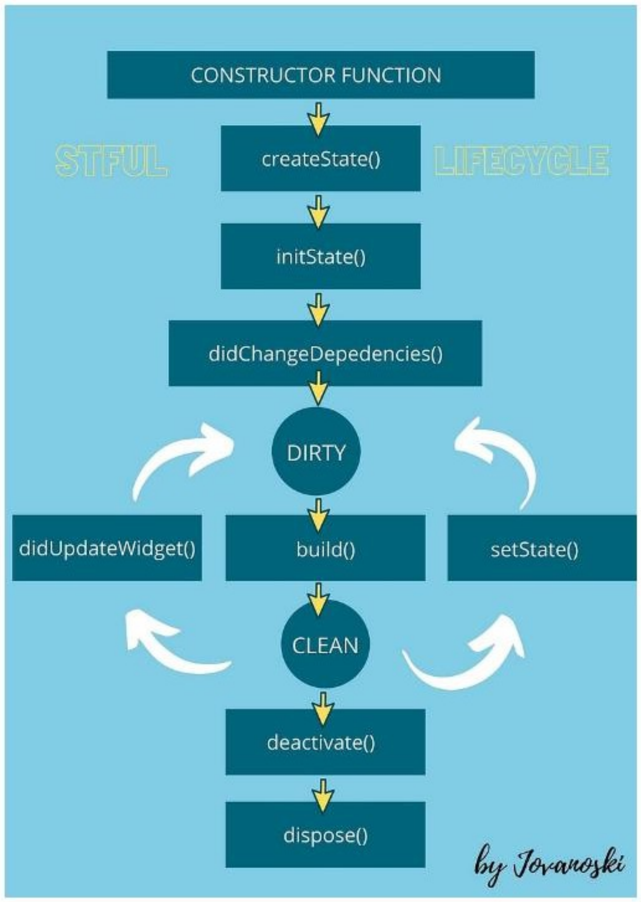
- **dispose()**: When the widget is permanently removed from the tree, `dispose()` is called, which is where you should release any resources held by the widget.

8. Dependency Changes:

- **didChangeDependencies()**: Called when an `InheritedWidget` the widget depends on changes.

Here's a simplified flow of the lifecycle methods for a `StatefulWidget`:

```
createState() -> initState() -> didChangeDependencies() -> build()
-> setState() -> build() -> didUpdateWidget() -> build()
-> deactivate() -> dispose()
```



Seven Cycles of StatefulWidget

The lifecycle of a stateful widget in Flutter consists of seven cycles. Understanding these cycles is essential for managing the state and controlling the behavior of the widget. Let’s explore each cycle:

- **createState():** This method is required and creates a State object for the widget. It holds all the mutable state for that widget. The State object is associated with the BuildContext by setting the mounted property to true.
- **initState():** This method is automatically called after the widget is inserted into the tree. It is executed only once when the state object is created for the first time. Use this method for initializing variables and subscribing to data sources.
- **didChangeDependencies():** The framework calls this method immediately after initState(). It is also called when an object that the widget depends on changes. Use this method to handle changes in dependencies, but it is rarely needed as the build method is always called after this.
- **build():** This method is required and is called many times during the lifecycle. It is called after didChangeDependencies() and whenever the widget needs to be rebuilt. Update the UI of the widget in this method.
- **didUpdateWidget():** This method is called when the parent widget changes its configuration and requires the widget to rebuild. It receives the old widget as an argument, allowing you to compare it with the new widget. Use this method to handle changes in the widget's configuration.
- **setState():** The setState() method notifies the framework that the internal state of the widget has changed and needs to be updated. Whenever you modify the state, use this method to trigger a rebuild of the widget's UI.
- **deactivate():** This method is called when the widget is removed from the widget tree but can be reinserted before

the current frame changes are finished. Use this method for any cleanup or pausing ongoing operations.

- **dispose():** This method is called when the State object is permanently removed from the widget tree. Use this method for cleaning up resources, such as data listeners or closing connections.

Xamarin:

Xamarin is an open-source framework that allows developers to build cross-platform apps for **Android, iOS, and UWP (Universal Windows Platform)** using a single shared **C# codebase**. It was originally based on the Mono project and was acquired by Microsoft in 2016. Since then, it has been integrated into the .NET platform. Here are some key points about Xamarin:

- **Single Codebase**: Xamarin enables developers to write once and use the same codebase across different platforms.
- **Native Performance**: Xamarin provides close–to–native performance, UI, and controls.
- **Microsoft Backing:** Being part of Microsoft's ecosystem ensures stability, continuous support, and wide learning opportunities.

Pros of Xamarin:

1. **Code Sharing, or 'Write Once, Use Everywhere'**:
   - Xamarin allows building single–code solutions for iOS, Android, and other platforms.
   - Achieves **60% to 95%** code sharing while maintaining close–to–native performance[1].
2. **Support and Full Technical Backing by Microsoft**:
   - Microsoft's involvement ensures stability, performance, and continuous developer support.
   - Wide learning opportunities are available for Xamarin developers[1].
3. **Native User Experiences**:
   - Xamarin provides access to native APIs, allowing developers to create truly native user interfaces.
   - UI elements are rendered natively on each platform, resulting in a consistent and familiar experience.
4. **Full Hardware Support**:
   - Xamarin apps can access device features like camera, GPS, sensors, and more.
   - Developers can utilize platform–specific APIs seamlessly.
5. **Open Source with Strong Corporate Support**:
   - Xamarin is open source, which encourages community contributions and transparency.
   - Microsoft's backing ensures long–term support and enhancements.
6. **Simplified Maintenance**:
   - Managing a single codebase simplifies maintenance and updates.
   - Bug fixes and feature enhancements apply universally.
7. **Complete Development Ecosystem**:

- Xamarin integrates well with Visual Studio and Visual Studio for Mac.
- Developers can leverage existing .NET libraries and tools.

Cons of Xamarin:

1. **Pricing**:
    - While Xamarin itself is free and open source, commercial development often requires using **Microsoft Visual Studio**, which comes with licensing costs[1].

CHARACTERISTICS OF Xamarin:

When learning Xamarin, you'll encounter various topics, including:

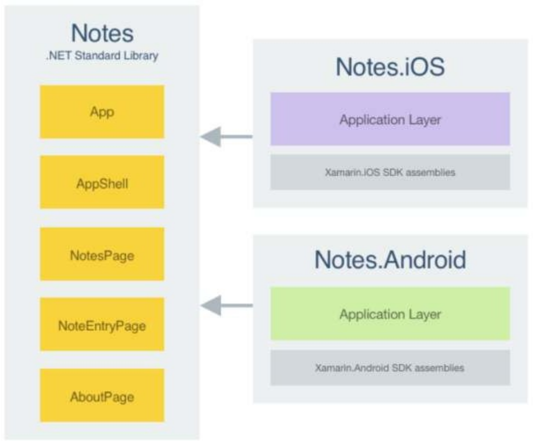
1. **Anatomy of a Xamarin.Forms Application**:
    - Understanding the structure of a Xamarin.Forms app.
    - Shared code, platform−specific projects (Android, iOS, UWP), and the .NET Standard library.
    - NuGet packages and dependencies.
2. **UI Development with Xamarin.Forms**:
    - Creating native controls using Xamarin.Forms.
    - Layouts, views, and responsive design.
3. **Accessing Native APIs**:
    - Interacting with platform−specific features (camera, geolocation, sensors).
    - Dependency injection and platform−specific implementations.
4. **Data Persistence and Storage**:
    - Working with databases (e.g., SQLite).
    - Serialization and data models.
5. **Navigation and Routing**:
    - Implementing navigation between pages.
    - Master−detail views, tabbed navigation, and navigation stacks.
6. **Testing and Debugging**:
    - Unit testing, UI testing, and debugging techniques.
    - Xamarin Test Cloud for automated testing.

ARCHITECTURE OF XAMARIN: ANATOMY OF XAMARIN.FORMS

Understanding this structure is essential for building cross-platform mobile apps using Xamarin.Forms. Here are the key components:

1. **Solution and Projects**:
    - A Xamarin.Forms app is organized into a **solution** containing one or more **projects**.
    - Each project represents a specific platform (e.g., Android, iOS, UWP) or shared code.
    - In the case of the **Notes application**, there are three projects:
        - **Notes**: A **.NET Standard library** holding shared code and UI.
        - **Notes.Android**: Contains Android-specific code and serves as the entry point for Android apps.

        - **Notes.iOS**: Holds iOS-specific code and is the entry point for iOS apps.
2. **Dependencies**:
    - The **Notes .NET Standard library project** includes dependencies managed via **NuGet packages** and the **SDK**:
        - **NuGet**: Packages like Xamarin.Forms, Xamarin.Essentials, Newtonsoft.Json, and sqlite-net-pcl.
        - **SDK**: The NETStandard.Library metapackage referencing the complete set of NuGet packages defining .NET Standard.
3. **Shared Code and UI**:
    - The **Notes project** contains shared code and UI components.
    - Developers write code here that can be reused across platforms.
    - Xamarin.Forms simplifies cross-platform UI development by providing native controls and layouts.
4. **Platform-Specific Projects**:
    - The **Notes.Android** and **Notes.iOS** projects contain platform-specific code.
    - These projects handle platform-specific features, UI adjustments, and app initialization.
5. **Application Lifecycle**:
    - Understanding the **lifecycle** of Xamarin.Forms apps is crucial:
        - **Construction**: Instantiation of widgets and initialization.
        - **Initialization**: `initState()` for setup tasks.
        - **Build**: Rendering UI via `build()`.
        - **State Updates**: Triggered by changes (e.g., user interactions).
        - **Rebuilding**: Efficient UI updates.
        - **Deactivation**: Cleanup when removed temporarily.
        - **Disposal**: Final cleanup when permanently removed.



Xamarin.Forms

Xamarin.Forms is an open-source UI framework. Xamarin.Forms allows developers to build Xamarin.iOS, Xamarin.Android, and Windows applications from a single shared codebase. Xamarin.Forms allows developers to create user interfaces in XAML with code-behind in C#. These user interfaces are rendered as performant native controls on each platform. Some examples of features provided by Xamarin.Forms include:

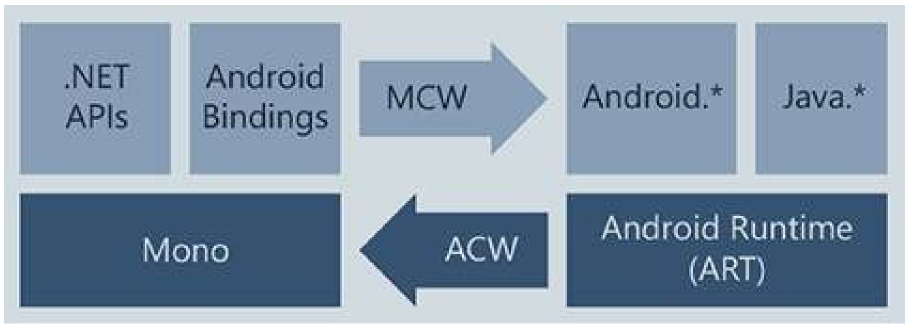
- XAML user-interface language
- Databinding
- Gestures
- Effects
- Styling

Xamarin.Essentials

Xamarin.Essentials is a library that provides cross-platform APIs for native device features. Like Xamarin itself, Xamarin.Essentials is an abstraction that simplifies the process of accessing native functionality. Some examples of functionality provided by Xamarin.Essentials include:

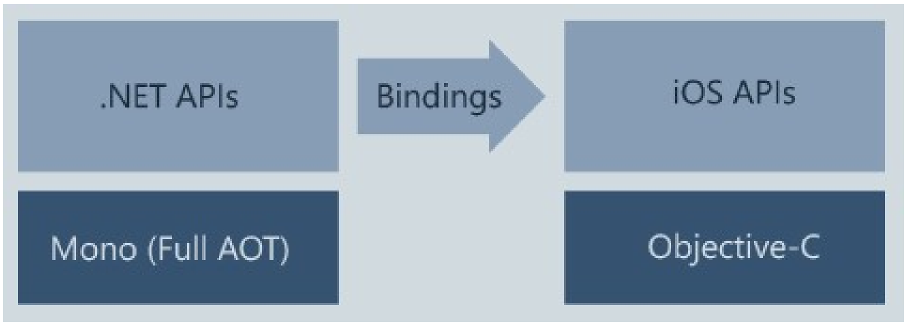
- Device info
- File system
- Accelerometer
- Phone dialer
- Text-to-speech
- Screen lock

Xamarin.Android



Xamarin.Android applications compile from C# into **Intermediate Language (IL)** which is then **Just-in-Time (JIT)** compiled to a native assembly when the application launches. Xamarin.Android applications run within the Mono execution environment, side by side with the Android Runtime (ART) virtual machine. Xamarin provides .NET bindings to the Android.* and Java.* namespaces. The Mono execution environment calls into these namespaces via **Managed Callable Wrappers (MCW)** and provides **Android Callable Wrappers (ACW)** to the ART, allowing both environments to invoke code in each other.

Xamarin.iOS



Xamarin.iOS applications are fully **Ahead-of-Time (AOT)** compiled from C# into native ARM assembly code. Xamarin uses **Selectors** to expose Objective-C to managed C# and Registrars to expose managed C# code to Objective-C. Selectors and Registrars collectively are called "bindings" and allow Objective-C and C# to communicate.

Application lifecycle for cross-platform development in Xamarin.

Understanding this lifecycle is crucial for managing state, optimizing your app, and handling side effects. Here's an overview:

1. **Software Development Lifecycle (SDLC)**:
   - The process of software development is called the **Software Development Lifecycle** (SDLC).
   - In the context of mobile application development, the SDLC includes several phases:
     - **Inception**: Defining the project scope, requirements, and initial planning.
     - **Design**: Creating wireframes, user interfaces, and architectural decisions.
     - **Development**: Writing code, implementing features, and testing.
     - **Stabilization**: Ensuring stability, fixing bugs, and optimizing performance.
     - **Deployment**: Preparing for release, packaging, and distributing the app.
     - **Maintenance**: Post-release updates, bug fixes, and ongoing support[1].
2. **Xamarin Application Lifecycle**:
   - The Xamarin application lifecycle follows similar principles across platforms (iOS, Android, and UWP). Here are the key stages:
   - **Construction**:
     - When you create a Xamarin.Forms page or view, the framework constructs the corresponding native UI elements.
     - Initialization tasks occur during this phase.

- **Initialization (initState)**:
  - After construction, the framework invokes the `initState()` method (similar to Xamarin.Forms' `OnAppearing()`).
  - In this phase, you can perform setup tasks, such as initializing data, subscribing to events, and configuring UI elements.
- **Build (Rendering)**:
  - The `build()` method (equivalent to Xamarin.Forms' `OnAppearing()` and `OnDisappearing()`) is called.
  - It returns a widget tree describing how the UI should look based on the current state.
  - The framework efficiently updates the UI by comparing the new tree with the previous one.
- **State Updates**:
  - When the app's state changes (due to user interactions, data updates, etc.), you signal this by calling `setState()`.
  - This triggers a rebuild of the UI, ensuring it reflects the updated state.
- **Rebuilding (Efficient UI Updates)**:
  - The framework efficiently updates only the necessary parts of the UI.
  - It uses a diffing algorithm to minimize changes and improve performance.
- **Deactivation (Temporary Removal)**:
  - If a page is removed temporarily (e.g., navigating away), the framework calls `deactivate()`.
  - Use this phase for cleanup tasks (e.g., pausing animations, canceling subscriptions).
- **Disposal (Permanent Removal)**:
  - When a page is permanently removed (e.g., app exit), the framework calls `dispose()`.
  - Perform final cleanup (e.g., releasing resources, unsubscribing from events).

3. **Testing and Deployment**:
   - After building your app, thoroughly test it on different platforms and devices.
   - Deploy the app to app stores (Google Play, App Store, etc.) for distribution.

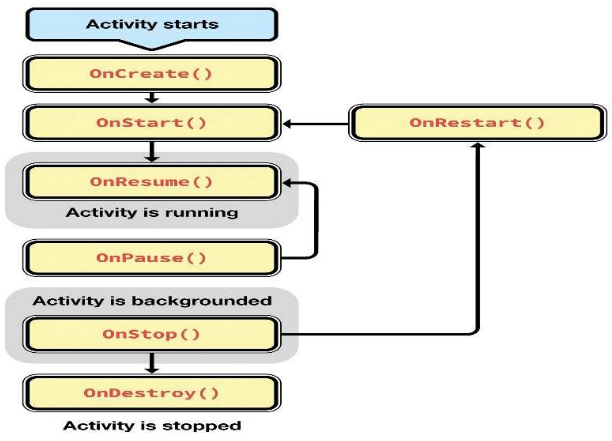
Xamarin.Forms App Lifecycle Overview

1. **Lifecycle Methods**:
   - The `Application` base class provides three essential lifecycle methods:
     - **OnStart**: Called when the application starts. Use it for initialization tasks.
     - **OnSleep**: Called each time the application goes to the background (e.g., when the user switches to another app or locks the device). Save state or pause ongoing tasks.
     - **OnResume**: Called when the application is resumed after being sent to the background. Restore state or resume tasks.
   - Note that there is no specific method for application termination. Under normal circumstances (not a crash), termination occurs from the `OnSleep` state without additional notifications to your code.
2. **Page Navigation Events**:
   - Xamarin.Forms provides two events related to page navigation:

        - **PageAppearing**: Raised when a page is about to appear on the screen. Use it to track pages as they become visible.
        - **PageDisappearing**: Raised when a page is about to disappear from the screen (e.g., navigating away). Useful for cleanup tasks related to the disappearing page.
    - These events are raised from the `Page` base class immediately after the `Page.Appearing` and `Page.Disappearing` events, respectively.
3. **Modal Navigation Events**:
    - Modal pages are pages that temporarily overlay the current page (e.g., dialogs, pop- ups).
    - Four events allow you to respond to modal pages being shown and dismissed:
        - **ModalPushing**: Raised when a page is modally pushed.
        - **ModalPushed**: Raised after a page has been pushed modally.
        - **ModalPopping**: Raised when a page is modally popped.
        - **ModalPopped**: Raised after a page has been popped modally.
    - The `ModalPopping` event arguments include a `Cancel` property. Setting `Cancel` to true cancels the modal pop.

Example Implementation:

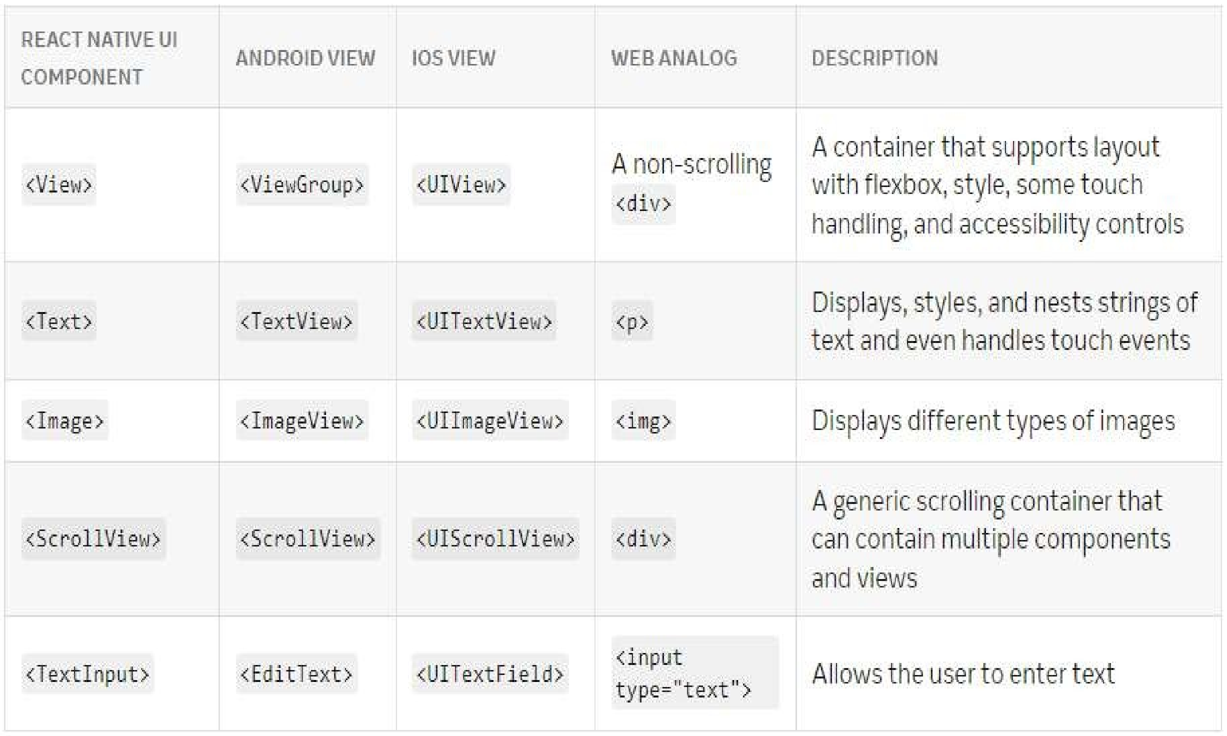
```
protected override void OnStart()
{
    // Initialization when the app starts
    Debug.WriteLine("OnStart");
}

protected override void OnSleep()
{
    // Save state or pause ongoing tasks when the app goes to the
background
    Debug.WriteLine("OnSleep");
}

protected override void OnResume()
{
    // Restore state or resume tasks when the app is resumed
    Debug.WriteLine("OnResume");
}
```



REACT NATIVE:

React Native is an open source framework for building Android and iOS applications using [React](https://react.dev/) and the app platform's native capabilities. With React Native, you use JavaScript to access your platform's APIs as well as to describe the appearance and behavior of your UI using React components: bundles of reusable, nestable code. You can learn more about React in the next section. But first, let’s cover how components work in React Native.

Core Components: React Native has many Core Components for everything from controls to activity indicators. You can find them all documented in the API section. You will mostly work with the following Core Components:



REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
`<View>`	`<ViewGroup>`	`<UIView>`	A non-scrolling `<div>`	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Displays, styles, and nests strings of text and even handles touch events
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Displays different types of images
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	A generic scrolling container that can contain multiple components and views
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Allows the user to enter text

React Native runs on React, a popular open source library for building user interfaces with JavaScript. To make the most of React Native, it helps to understand React itself. This section can get you started or can serve as a refresher course.

- components
- JSX
- Props
- state