CCS332 App development - 2021 Reg

App Development

1. Basics of Web and Mobile application development

1.1.Basics of Web Development

- **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.

- **Frontend Roadmap:**

  Frontend may be HTML, CSS, JavaScript, BootStrap.

- **HTML**: HTML stands for Hyper Text Markup Language. It is used to design the front end portion of web pages using markup language. It acts as a skeleton for a website since it is used to make the structure of a website.

- **CSS:** Cascading Style Sheets fondly referred to as CSS is a simply designed language intended to simplify the process of making web pages presentable. It is used to style our website.

- **JavaScript:** JavaScript is a scripting language used to provide a dynamic behavior to our website.

- **Bootstrap:** Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular CSS framework for developing responsive, mobile-first websites. Nowadays, the websites are perfect for all browsers (IE, Firefox, and Chrome) and for all sizes of screens (Desktop, Tablets, Phablets, and Phones).

- Bootstrap 4

- Bootstrap 5

- The Roadmap of Front end is shown in Fig 1.

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 may be

- **PHP:** PHP is a server-side scripting language designed specifically for web development.
- **Java:** Java is one of the most popular and widely used programming languages. It is highly scalable.

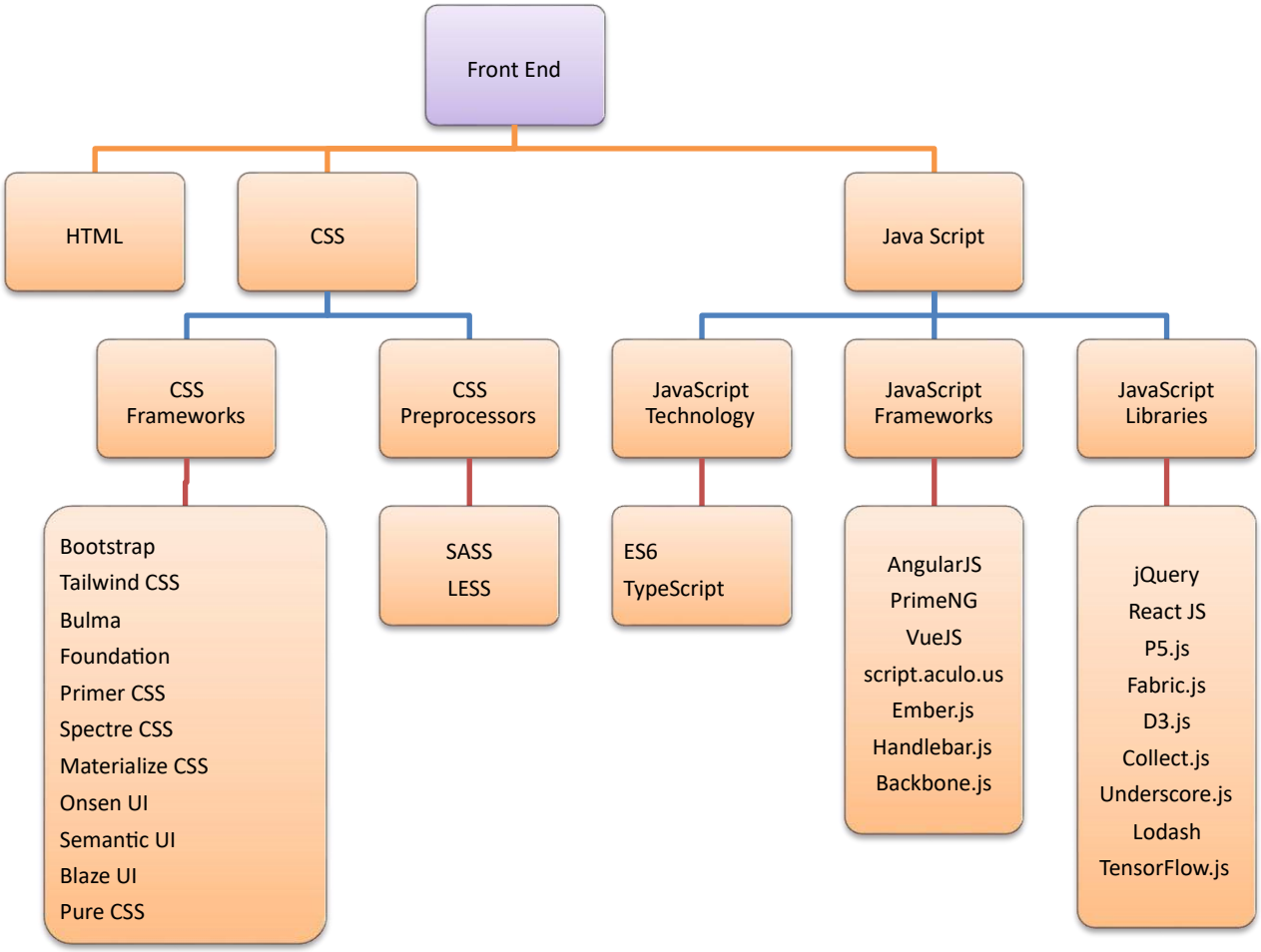
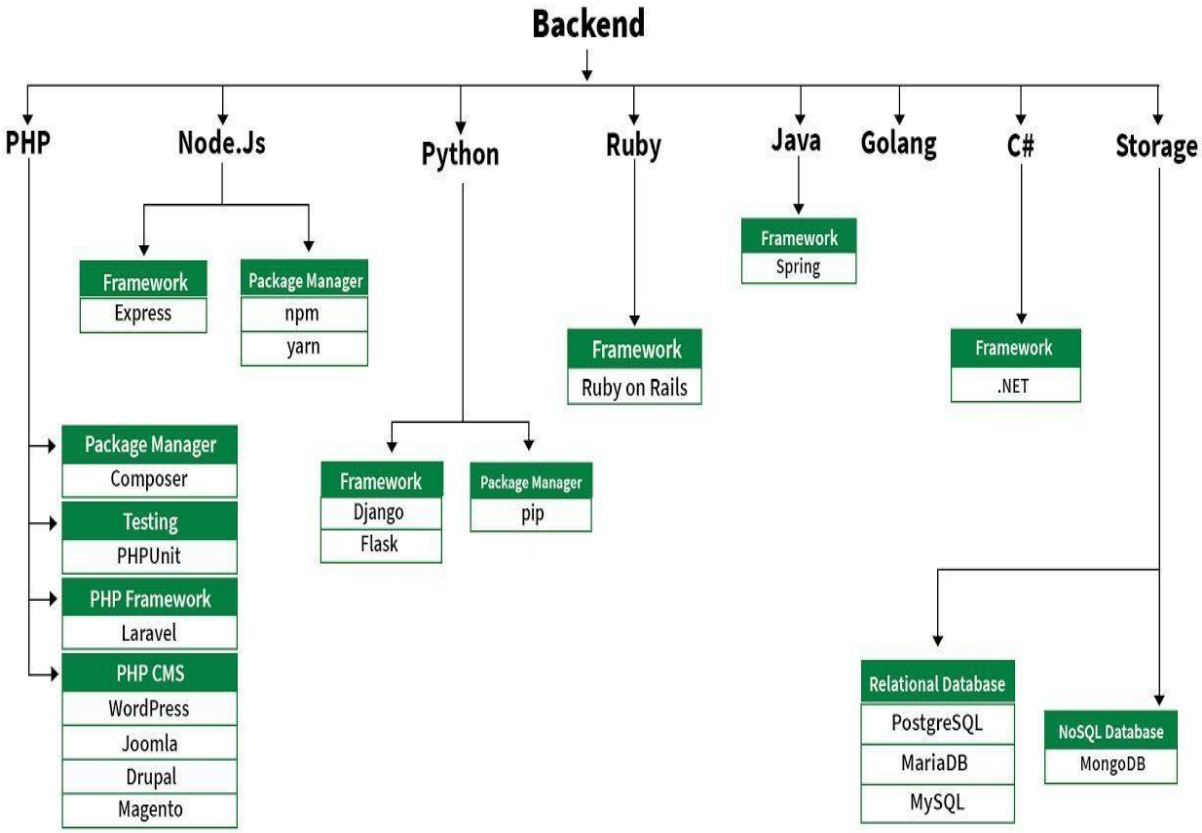


Fig 1. Front End Road Map

- **Python:** Python is a programming language that lets you work quickly and integrate systems more efficiently.

- **Node.js:** Node.js is an open source and cross-platform runtime environment for executing JavaScript code outside a browser.

- Backend Roadmap is shown in the following figure



1.2. Basics of MobileApp Development

Mobile apps are majorly developed for 3 Operating System.

1. Android
2. IOS
3. Windows

There are 3 different ways to develop Mobile apps:

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

1st Party Native App development:

These types of apps normally run in the native devices, that is, it runs only in the OS that it is specifically designed for it. These apps cannot be used on different devices using a different OS. The apps that are developed for android are normally coded using Java or Kotlin languages. The IDE normally used for android app development is Android Studio which provides all features and the apps

that are developed for IOS are generally coded in Swift language or Objective-C. The IDE suggested for IOS App Development is XCode.

Example,

Here's an example of a 1st party native app:

A retail company wants to improve the in-store shopping experience for its customers. They develop a 1st party native app that allows customers to:

- Browse the store's inventory and product information
- Create a shopping list
- Scan barcodes to view product information and reviews
- Locate items in the store using an interactive map
- Pay for items directly through the app, without having to wait in line at the register
- The app is only available to the company's customers and can only be used in their physical stores. The app is designed to integrate with the company's existing systems, such as inventory management and point-of-sale systems.

This app is developed by the retail company for their own use, to improve the in-store customer experience, increase sales and gain insights from the customer's behavior.

In this example, the retail company is the 1st party, and the app is a native app, because it is developed for the specific platform (iOS or Android) and can take full advantage of the device's capabilities and features.

2. Progressive web Application:

Progressive web apps are essentially a website which runs locally on your device. The technologies used are Microsoft Blazor, React, Angular JS, Native Script, Iconic. These technologies normally used for web development propose. The apps' UI is developed the same way as they are developed while developing the website. This category has many ups and downs let's start with the advantages of Progressive web apps.

Example,

Here's an example of a Progressive Web App:

A news website wants to provide its users with a better mobile experience. They develop a Progressive Web App that:

- Allows users to access the website offline by storing content on the user's device
- Sends push notifications to users to alert them of breaking news
- Can be installed on the user's home screen like a native app

- Provides a fast and smooth browsing experience
- Has a responsive design that adapts to different screen sizes
- Users can access the PWA by visiting the website on their mobile browser. They are prompted to install the PWA on their home screen, which allows them to access the website offline and receive push notifications.

In this example, the news website is the 1st party and the app is a Progressive web app, because it can be accessed through a web browser and can be installed on the user's device like a native app.
It also allows users to access the content offline and have a fast and smooth experience.

3. Cross-Platform Application:

These are frameworks that allow developing total native applications which have access to all the native features of IOS and Android but with the same code base. These apps run on both Android and IOS. So normally the development speeds of these apps are very fast and the maintenance cost is low. The performance speed is comparatively low to 1st party native apps but faster than PWA.

Example,

Here's an example of a cross-platform application:

A project management company wants to create a project management tool that can be used by teams on different platforms. They develop a cross-platform application that:

- Can be used on Windows, Mac, iOS, and Android devices
- Allows users to create and assign tasks, set deadlines, and track progress
- Integrates with popular tools such as Google Calendar and Trello
- Has a user-friendly interface that works seamlessly across all platforms
- The application can be downloaded from the company's website or from different app stores such as App Store, Google Play Store, Microsoft Store, and Mac App Store, depending on the platform.

This example illustrates how the company developed a project management tool that can be used by teams on different platforms, Windows, Mac, iOS and Android, which is a cross-platform application. It allows teams to collaborate and manage their projects seamlessly, regardless of the platform they use.

2. Native App

The term *native app* refers to platforms such as Mac and PC. The Photos, Mail or Contacts applications that are preinstalled and configured on every Apple computer is the example for Native App.

2.1. Native App Development

Native app development is the process of creating mobile applications specific to a single platform.

Examples are Google's Android and Apple's Ios.

The programming languages and tools required for developing native apps are specific to each platform. Android app developers would use Java or Kotlin, whereas iOS app developers would use Objective C or Swift. A native Android application cannot run on the iOS system, nor can a native iOS app run on the Android system.

For users to access the native applications, they have to download them from the platformspecific stores, i.e., App Store for iOS and Play Store for Android.

Benefits of Native App Development

1. Flawless Performance

Since native apps are developed for a specific platform, they are completely optimized for that particular platform. They use the platform's core programming language and APIs. Further, they get complete hardware and OS support. These factors combine to make sure that they provide the best possible performance.

Another reason for their fast and responsive nature is that all the visual and content elements of a native app are downloaded and stored on the device. So as a user navigates through a native app, everything loads quickly, thereby considerably reducing the load speed.

2. Superior UI and UX

Every platform has a set of guidelines designed to enhance the user experience. When these platform-specific guidelines are followed, the applications behave as if tailor-fit for that platform. Their look and feel is completely consistent with the operating system, making them feel like an integral part of the platform.

Thus, for a user using a native app, the learning curve is very low. They can understand the app layout naturally and interact with it in an intuitive manner. Their familiarity with the actions and gestures enable them to navigate through the app quickly. The consistent UI and superior UX is a great benefit offered by native apps.

3. Access to Device Features

Native apps can directly access all the tools and features of the particular device and operating system. They can access hardware components like GPS and camera without the need for any intermediary plugins. This direct access to device features encourages faster execution of the apps and also opens the door for more creative solutions to be implemented.

4. Better Store Support

Since native apps adhere to the guidelines of the specific platforms, it is easier to publish them in the respective app stores. Better compliance with store guidelines enables them to get ranked high on the app store. These apps receive better store support as they deliver consistent performance.

5. More Security

Compared to cross-platform app development, native app development is more secure. As they are not dependent on different browsers or development platforms, they are less vulnerable to security threats. Also, the use of platform-specific programming language plays a role in increasing data protection. In hybrid apps, the use of universal languages increases the risk elements. Native apps offer better data encryption and higher levels of protection.

6. Easy Testing

A significant advantage of native app development is the availability of inbuilt testing tools. This makes app testing easier and more efficient. It enables better troubleshooting, accurate error detection and makes it simple for developers to test apps and find bugs.

7. Fewer Bugs

In hybrid app development, there is a dependency with mobile app development frameworks like Ionic or Xamarin. This bridge adds an extra layer, which increases the chances of bugs occurring during development. In native apps, there are fewer dependencies and hence fewer chances for bugs to occur.

8. Instant Updates

Native apps can access the latest Android and iOS release updates quickly. This allows developers to build native apps with the latest features, enabling users to access new features with an OS update.

But for hybrid apps, the developers have to wait for the dependent tools to implement the new features. Therefore the users of hybrid apps cannot access the latest updates instantly. This hampers the user experience to a large extent.

Challenges of Native App Development

Despite the many wonderful benefits of native apps, they also have some drawbacks or challenges. Let us see what they are and how they affect native app development.

1. Higher Development Costs

Since you will have to build separate apps for different platforms, the development will get costly. For each platform, you will need to hire different development teams. Further,

app maintenance also has to be done separately for each platform. This is a major challenge of native app development.

2. More Development Time

Unlike hybrid apps, native apps cannot use a single codebase for multiple platforms. Separate codes need to be written for different platforms. That doubles up the time of app development.

3. Need for Skilled Developers

The development of native apps is somewhat complex, and a skilled team of developers are needed to create a successful native app. The programming languages used are more advanced, and it isn't easy to find developers proficient in those languages.

4. Require Constant Updates

Whenever there is a new update or a bug fix, the users have to update the app to the latest version from the respective app store. Otherwise, they may experience glitches while operating the app.

5. Lengthy Downloading Process

Compared to web app development, a major disadvantage of native app development is that native apps have to be downloaded. The app download is a multi-step process including going to the app store, finding an app, complying with its terms, downloading, and installing. This requires users to spend considerable time and effort and may result in user attrition.

Tools and Technologies Used in Native App Development

Native app development involves writing code in the native programming language and compiling the native code to run on a processor. To make this process of creating a native app easier, developers use two tools — a software development kit (SDK) and an integrated development environment (IDE).

SDK is a set of tools that a developer would need while writing programs for a particular platform. These tools include compilers, debuggers, profilers, and other tools, as well as libraries, frameworks, header files, etc.

An IDE is a platform for writing and debugging applications. It brings together all the components needed for programming in one place. An IDE creates a convenient user interface for developers to make programming easier.

Some SDKs have dedicated IDE included with them, while some don't. If an SDK does not include an IDE, developers can download and install any compatible IDE or use a text editor to write programs.

Native App Development for Android

Native app development for Android uses tools like

- Android SDK combined with Android Studio
- Firebase
- Android Jetpack
- Mockplus
- Command-line tools for Windows, Mac, and Linux.

The programming language that it uses is Java or Kotlin. Java is a popular programming language used widely by developers all over the world. A notable aspect of Java is its "Write once, run anywhere" feature. It means that a compiled Java code can run on any Java supporting platform without recompilation. Java is an object-oriented programming language and is secure, robust, and developer-friendly.

Native App Development for iOS

The tools and resources that support the development of mobile apps for iOS are

- iOS SDK integrated with Cocoa Touch UI framework
- XCode (official IDE)
- Swift Playgrounds
- TestFlight (for beta testing)

Apart from these, there are also third-party tools like AppCode and CodeRunner.

The programming language Swift is known for its speed, safety, and leading advancements aimed to provide consistent and powerful performance to developers. Swift is well recognized for its excellent error detection and handling capabilities.

3. Hybrid application (Hybrid app)

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.

Working of Hybrid Apps:

Developers build hybrid apps using web technologies such as JavaScript, CSS and HTML. The code is then wrapped within a native application using open-source hybrid app development frameworks, like Ionic or React Native. This allows the app to run through each platform's embedded browser instead of the web browser, which means they can be installed on mobile devices and submitted to app stores for sale, just like regular native apps.

Five Best Examples of Hybrid Apps

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

Example #1: Instagram

Instagram is the most popular social media app for sharing images and videos. Going hybrid enables the app to support tons of media and offline data. The app permits users to access its features and media, photos, and short videos while also being in the offline mode.

Example #2: Uber

Another great example of hybrid app development in action is the Uber application. With easy navigation and an intuitive user interface, Uber is easily accessible and convenient, which is why this hybrid app is probably the most used ridesharing app in the world.

Example #3: Gmail

Previously built using HTML, Google has upgraded to HTML 5 to provide users with new, advanced features and functionalities, thus enhancing the user experience. This is the perfect example of combining native and web elements to create a high-performance mobile application.

Example #4: Evernote

With a beautiful design and its ability to work flawlessly on different devices, this productivity app stands out for its performance. Something that goes against the premise that hybrid apps are slow and of low performance.

Example #5: Twitter Twitter is another hybrid app proving that performance is not an issue for this type of app development. The social media application can handle huge amounts of traffic, and it's accessed by millions of users every day.

Pros & Cons of Hybrid App Development

Hybrid apps cost less, take less time to create, and are easier to manage than native ones. Other significant benefits of hybrid app development include:

- Easier scalability. Because hybrid apps use a single codebase, they can be deployed across devices. For example, when you build them for Android, you can quickly launch them on iOS.
- Single codebase to manage. Unlike native app building, where you have to create two apps, with hybrid software building, you develop only one app, so you only need to manage one database.
- Faster build time. Since there is one database to manage, it takes less time to make a hybrid than native apps.
- Low cost of development. Hybrid mobile apps cost less than native apps. Because developers write one set of code, the initial costs and the maintenance costs are low.
- Offline availability. Hybrid apps will work in an offline mode due to their native infrastructure. Users can still load the application and see the previously loaded data if they can't access real-time data.

However, hybrid app development has a few disadvantages as well:

- User experience. Considering that there is one codebase for all platforms, the user experience might not be positive. We are talking about different operating systems, so it is difficult to customize an app based on one platform.
- Lower performance. Hybrid mobile apps load in a web view that is difficult to reach a native performance, and that's one of their biggest drawbacks.
- Availability of features. Some new features of the hybrid software are not available for some platforms. What's more, some native features may not even exist.
- Errors. Hybrid software can come with hidden errors.

4. 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.

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

- Titanium
- React Native

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

Each one of these comes with its unique feature and offerings.

Choose the Appropriate Near-native or Hybrid App Framework:

The choice of the best solution for near-native or hybrid development depends on multiple factors, including:

- The complexity of the solution
- Developers' skills
- The target time to market
- Available budget

Without taking these factors into consideration, it's impossible to say which of these tools for nearnative and hybrid mobile application development is the best.

Examples for cross-platform mobile applications

Well-known examples of cross-platform mobile applications include:

- Instagram, Skype, Walmart, and Airbnb (React Native)
- Google Ads, My BMW App, eBay Motors, and the New York Times (Flutter)
- The World Bank, Fox Sports, Alaska Airlines, and BBC Good Food (Xamarin)

Pros of cross-platform mobile apps development

1. Faster development

With cross-platform solutions, developers mostly work with one codebase which handles iOS and Android and there is no need to build separate Android or iOS projects. Everything is in one place. Cross-platform applications are built as single projects but support different devices. It is possible to reuse a big part of an app's code between platforms, and that's why development is much quicker.

2. Lower costs

In comparison to native applications, a cross-platform one can be about 30% cheaper than building iOSand Android apps separately. We just have one project for both iOS and Android, and big chunks of code and other assets can be reused between the platforms. In this way, cross-platform development is faster so, as a result, the overall cost for each platform is less than developing native mobile apps.

3. Wider audience

Many app owners face a challenge when starting with mobile app development: is it better to create an Android or an iOS application first? No matter which platform has a bigger market, it is always a trade-off which can cost a lot of money. Starting with a cross-platform solution we get a much wider audience at the very beginning – we can target both markets at the same time. We can also build for one platform first to release the product faster, and then quickly iterate on adding support for second platform.

3. Consistency between platforms

iOS and Android have some differences in terms of building navigation and design. In cross-platform development, most popular UI differences are handled by default. In native development, even in a team with great communication, there can be some differences in implementing functions. In cross-platform development, this situation doesn't occur often, as the platforms share the same codebase. It also allows you to build a more coherent brand identity in apps on both platforms with less effort.

4. Reusable code This is one of the greatest things about cross-platform applications – it's possible to create one codebase for Android and iOS at the same time. In native applications it is necessary to write code separately, which is usually done by two different developers or teams. In custom mobile app development, the whole codebase is in one place. This saves a lot of time, because one developer can handle both platforms concurrently.

Cons of cross-platform mobile apps development

1. Lower performance

Performance is one of the most important characteristics of an app. It depends on many factors but, in general, if we compare two applications where one is native and the other is crossplatform and both have the same functionalities, the native one will be slightly faster. However, these differences in performance are usually small, especially when it comes to simple applications.

2. Harder code design

Cross-platform applications have to adapt their design and functionalities not only to specific devices but also to platforms, which have many differences. As a result, it creates extra work for developers who have to handle specific exceptions for a variety of devices and platform differences, especially with more complicated features. These issues don't occur that often in native apps, so developers can focus on solving users' problems.

3. Long wait time for new features

Every time Google or Apple introduces a new feature for Android or iOS, it takes some time to update applications to support this new feature. In native apps, new SDKs are provided with the updates much faster than for cross-platform frameworks.

5. Progressive Web Applications

The full form of PWA is Progressive Web Applications. PWA is HTML 5 webpage. It is a web application like a mobile application that is stored directly on the mobile from the website. In other words, it is a website that runs on the mobile browser of the user. PWA is a new technology. PWA allows the website to be stored in your device. It creates an icon in the form of a website app, and that icon feels like a mobile application upon opening.

After the PWA icon of the website is created in the device, all the posts that are open with the help of the internet, are automatically stored in your device which does not need the internet to read the second time. It is a methodology where we combine native app experience with the Browser feature.

Generally, when we have to build applications, we have to develop that application for different platforms, such as iOS, Android, and Windows. But in the case of PWA, we do not need to develop separate applications for different platforms. We only need to create HTML 5 based webpages that can run on any mobile browser.

Characteristics of PWA

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.

Advantages of PWA

1. PWA works like an app on mobile and looks very impressive.
2. It does not need an update.
3. It is easily saved in the device.
4. It's immediately loads on your mobile.
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 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.

Technical components of PWA

PWA has five components.

1. Web App manifest
2. Application shell
3. Service worker
4. Webpack
5. Transport Layer Security (TLS)

1. Web App manifest

The web app manifest is the first component of the PWA. It is a simple JSON file that controls a user's application. Usually, it is named "manifest.json". It is the most important component for the presence of PWA. When you first connect PWA to a network, a mobile browser reads the "manifest.json" file and stores it locally in cache memory.

When there is no network access in PWA, the mobile browser uses the application's cache memory to run the PWA program while offline.

The "manifest.json" file helps the PWA to give a look of a native app. With the help of the manifest.json file, the PWA developer can control how the application is presented to the user mobile screen. The PWA developer can also set a theme for the mobile's splash-screen and the application's address bar.

The "manifest.json" file allows the PWA developer to search for a centralized location for the metadata of the web application. The JSON file defines the links to icons and icon sizes, the full and abbreviated name of an app, types, background color, theme, locations, and other visual details that are required for an app-like experience.

2. Application Shell

The application shell is the third technical component of the PWA. It is specialized to split the static and dynamic content of the application. The minimal CSS, HTML, JavaScript, and any other dynamic and static resources offer the structure for your web page. It reduces the actual content that is unique to the webpage. This component ensures a very critical approach to the development of progressive web apps.

It permits the PWA to be executed without any connection. The basic design elements of this component enable it to perform such a task. It drives especially for applications that are based on JavaScript.

3. Service Worker

A service worker is a web worker. It is a JavaScript file that runs aside from the mobile browser. In other words, it is another technical component that promotes the functionality of PWA. The service worker retrieves the resources from the cache memory and delivers the messages.

It is independent of the application to which they are connected, and has many consequences:

- The service worker does not block the synchronized XHR, so it cannot be used in local storage (It is designed to be completely asynchronous).
- It can receive information from the server even when the application is not running. It shows notifications in the PWA application without opening in the mobile browser.
- It cannot directly access the DOM. Therefore, the PostMessage and Message Event Listener method is used to communicate with the webpage. The PostMessage method is used to send data, and the message event listener is used to receive data.

Things to understand about it:

- It is a programmable-network proxy that helps you monitor how your page handles network requests.
- It only works on the HTTPS because HTTPS is very secure, and it intercepts network requests and modifies responses.

Service worker lifecycle

The service worker lifecycle is the most complex part of the PWA. There are three stages in the lifetime of a service worker:

1. Registration
2. Installation
3. Activation

Registration

A service worker is basically a JavaScript file. You need to register it in your crucial JavaScript code to use a service worker. Registration tells your browser location of the service worker and starts installing it on the background. One thing that distinguishes a service worker file from a standard JavaScript file is that a service worker runs in the background away from the mobile's main browser. This process is the first phase of the service worker's lifecycle.

The code of service worker registration is placed in the main.js file.

```
1.  if ("Service-Worker- in navigator.js)
2.  {
3.  navigator. Service - worker. register ('/service-worker. json')
4.  then (function (registration)
5.  {
6.  console.log ('Registration successful finish, scope is:', registration. scope.);   7.
    }
8.   Catch (function (error)
9.   {
10.  console.log ('Registration failed, error:', error.js);
11.  }
12.  }
```

First, this code checks whether the browser supports the service worker. The service worker is then registered with **navigator.serviceWorker.register** when the service worker returns a promise. If the promise is fulfilled, the registration is successful; otherwise, the registration is failed.

Scope of registration

This scope determines the web pages that are managed by a service worker. The location of the service worker defines the default scope. Whenever you register a service worker file at the main folder of the system, it is not important to specify its scope, because it controls all webpages.

```
1. navigator.service.worker.register ('/sw.js', { scope: '/'  });
```

Installation

When a new service worker is registered with the help of navigator.serviceWorker.register, the JavaScript code is downloaded, and the installation state is entered. If the installation succeeds, the service worker further proceeds to the next state.

```
1.  const assets to.cache = [
2.  '/js/app.js',
```

```
'/about.html',
'/index.html',
'/css/app.css',
]

        self. addEventListener ('install', function (event) {
        event. wait Until (
        caches. Open('staticAssetsCache')
        then (function (cache) {
        return cache Add All (assetsToCache.);
        })
        );
        });
```

Activation

Once the service worker is successfully installed, it converts to the activation phase. If there is an open page controlled by the previous service worker, the new service worker enters the waiting state. The new service worker is activated only when no pages are loaded in the old service worker. A service worker is not activated immediately after installation.

A service worker will only be active in these cases:

- If no service worker currently active.
- If the user refreshes the webpage.

Service-worker.js

```
self.addEventListener('activate', function(event) {
// Perform some task
});
```

The service worker can manage network requests rather than caching. It roves around the latest Internet API.

1. Fetch: The Fetch API is a basic resource of the GUI. It makes it easier to control webpage requests and responses than older XMLHttprequests, and this often needs extra syntax, and its example is controlling the redirects. When a resource is requested on the network than the fetal event is terminated.

Fetch request example:

```
1. fetch('ABCs/ABC.json')
2. then (function (response){
3. // response function
4. })
5. catch (function (error)
6. {
7. console.log ('problem section: \n', error);
8. });
```

2. Cache API: A cache interface has been provided for the service worker API, which allows you to create a repository of responses as requested. However, this interface was designed for service workers. It does not update the memory in the cache unless specifically requested.

Features of the service worker

- **Offline:** Enabling offline functionality is possibly the most demanding service worker facility.
- **Caching:** The control of cache content is the most common feature for service workers.
- **Content delivery networks:** The CDN and other external material may be difficult to handle. The PWA developers sometimes select out of publicly hosted software due to sameorigin rules and SSL, but you may still upload scripts from CDNs.
- **Push notifications:** The feature of push notifications is the best way to interact with users and visitors. This feature enhances the performance of the PWA application.
- **Background synchronization:** It is another very important feature of a service worker that synchronizes tasks in the background.

4. Webpack

The webpack is the fourth component of the PWA. It is used to design the PWA front-end. It allows the PWA-developers to gather all JavaScript resources and data in one location.

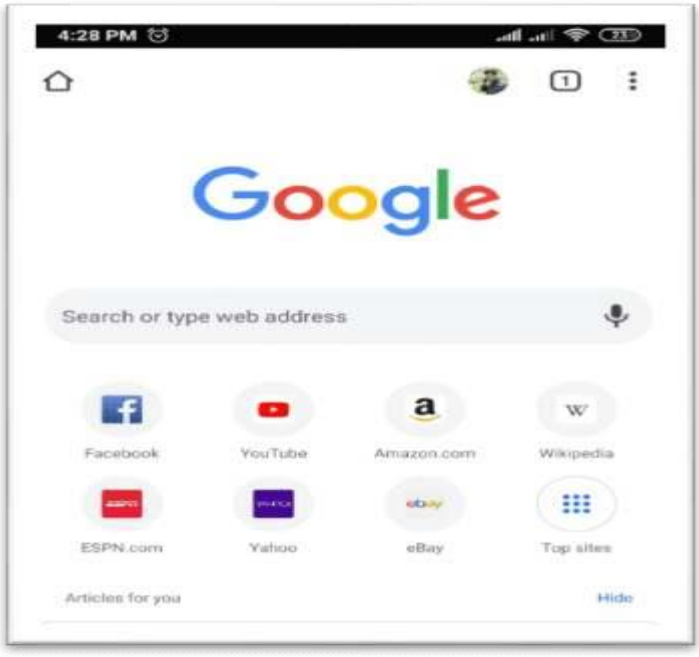
5. Transport Layer Security (TLS)

The transport layer security is the fifth component of the PWA. This component is a standard for all robust and secure data exchange between any two applications. The integrity of the data requires the website's service through the HTTPS and an SSL certificate installed on the server.

Install a PWA

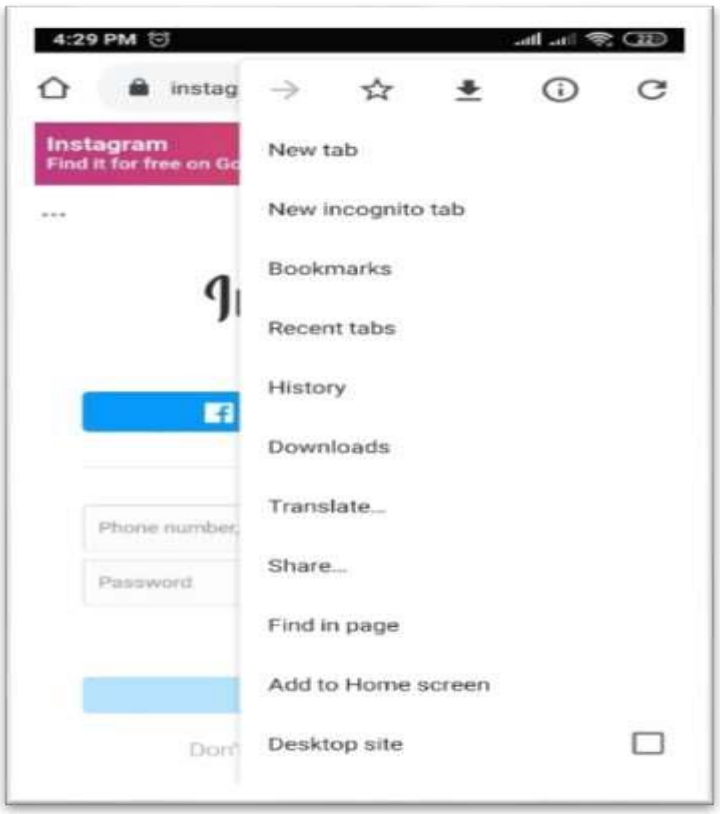
On Android phones, installing a PWA is pretty simple.

Step 1: The first thing to do is open google chrome on your mobile phone.

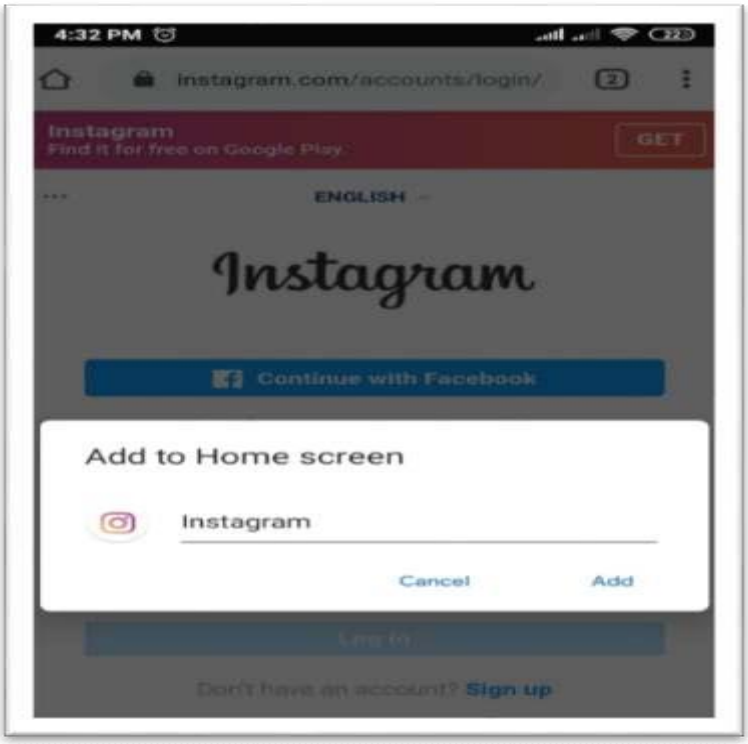


Step 2: Then, open the website, for example, Instagram.com.

Step 3: Now, click on the three dots of the right corner in the google chrome.



Step 4: Then, click on the Add to Home screen option and add it.



On iOS, you visit the website with the Safari browser and click the share icon. This opens a submenu with more icons and an option to save to the home screen.

Some popular examples of Progressive Web Apps

AliExpress

AliExpress is a popular e-commerce website that was introduced by the Alibaba Group. They turned their website a few years ago to a Progressive Web App for mobile phones.

After changing to a Progressive Web App, AliExpress saw incredible results:

- 104% increase in conversions for new users.
- 3x increase loading time in the PWA app.
- 50% higher re-engagement.

Flipkart

In India, Flipkart is the largest e-commerce website. Flipkart has recently updated its website for mobile users to a PWA that is called Flipkart Lite. Flipkart lite combines the best of both the web and native apps. It provides a fast and uninterrupted experience to smartphone phone users.

After changing to a Progressive Web App, Flipkart saw incredible results:

- 70% increase in conversions.
- 40% higher re-engagement.
- 3x increase loading time in the PWA app.
- 65% increase in pages session in the PWA app.

Twitter Lite

Twitter is a popular platform for social media. In **2017**, they changed their website to Progressive Web App for the mobile users that are called twitter lite. Twitter Lite loads posts instantly. It reduces data usage by optimizing pictures and keep on cached data.

Twitter Lite rebuilds customers with push notifications system. It also allows users to connect the Progressive Web App to their home screen of mobile phones.

After changing to a Progressive Web App, Twitter saw incredible results:

- 75% increase in Tweets sent.
- 20% decrease in bounce rate in the PWA.
- 65% increase in pages session in the PWA.

Instagram

Instagram is a very popular social media app. They changed their website to Progressive Web App for mobile users. It reduces data consumption through processing photos.

After changing to a Progressive Web App, Instagram saw incredible results:

- 77% increase in conversions.
- 50% higher re-engagement.
- 3x increase loading time in the PWA app.
- 4x lower data usage in the PWA app.

BookMyShow

BookMyShow is a popular website in India that is used to book tickets online. In 2017, they changed their website to Progressive Web App for mobile users.

Some uses had many problems with the BookMyShow mobile app. But PWA solved all problems. It gives smartphone users a smooth ticket-booking experience.

After changing to a Progressive Web App, Instagram saw incredible results:

- Over 80% increase of conversions in the PWA app.
- Smaller size than the iOS app and android app.
- 5x increase in conversion rates.

MakeMyTrip

MakeMyTrip is the first online travel website in India. The website owner decided to invest in a progressive web application to provide its users with a fast and appealing mobile web experience. After changing to a Progressive Web App, Instagram saw incredible results:

- 3x increase in conversion rates.
- Smaller size than the iOS app and android app.
- Over 160% increase of user sessions in the PWA app.

6. 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.

Designing for the best experience for all users

Web pages can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device.

Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:

Desktop **Tablet** **Phone**

It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

Responsive Web Design - The Viewport

Viewport

The viewport is the user's visible area of a web page.

The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size.

Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen.

Setting the Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the `<meta>` tag.

You should include the following `<meta>` viewport element in all your web pages:

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

This gives the browser instructions on how to control the page's dimensions and scaling.

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

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

Here is an example of a web page *without* the viewport meta tag, and the same web page *with* the viewport meta tag:

Without the viewport meta tag

With the viewport meta tag

Size Content to the Viewport

Users are used to scroll websites vertically on both desktop and mobile devices - but not horizontally!

So, if the user is forced to scroll horizontally, or zoom out, to see the whole web page it results in a poor user experience.

Responsive Web Design - Grid-View

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:

Example

```
.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% -->
</div>
```

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;
}
```

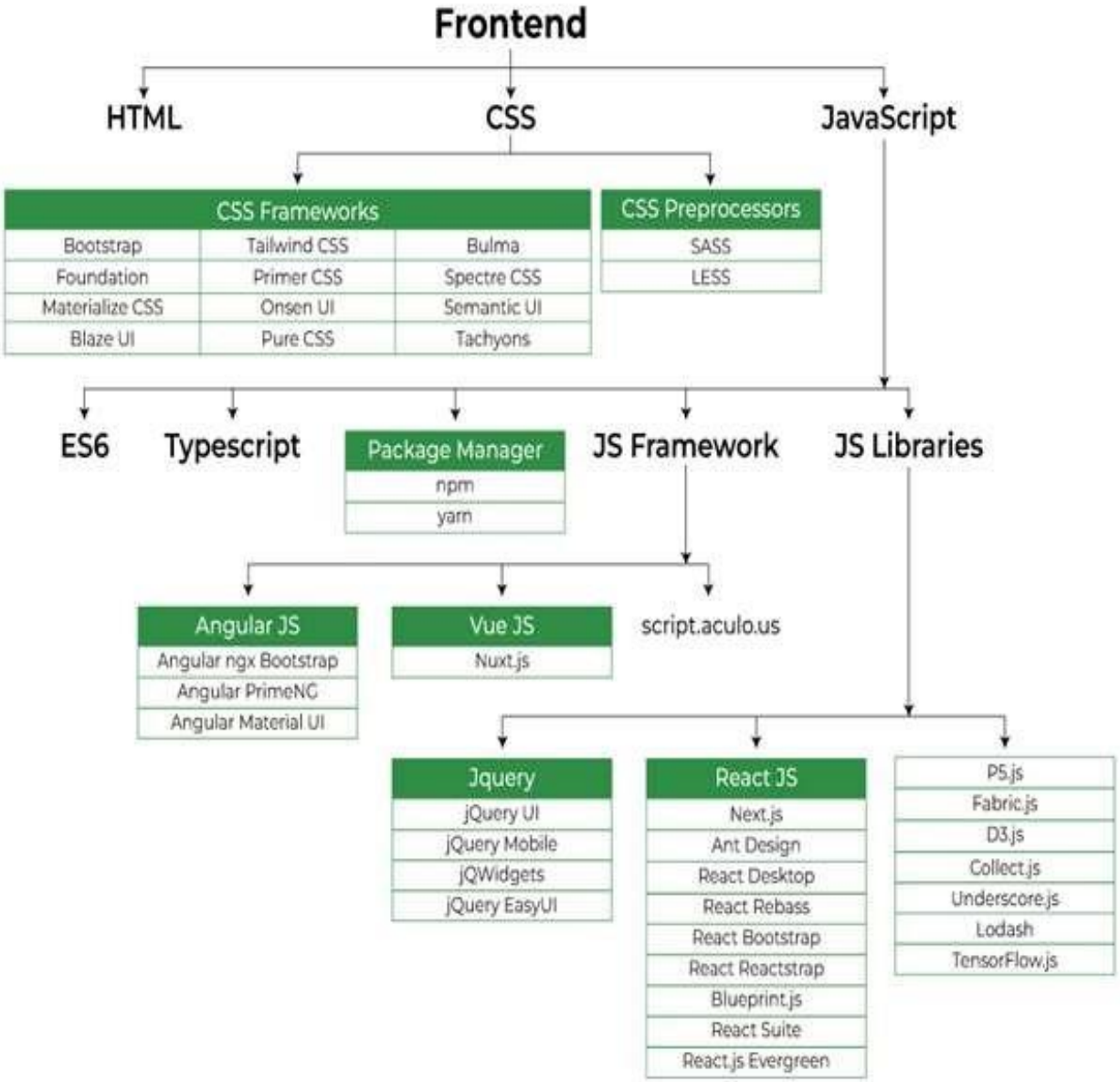
We also want to add some styles and colors to make it look better:

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;
}
```

References **https://www.codewithrandom.com/2022/11/20/restaurant-website-using-html-css/** **https://www.freecodecamp.org/news/responsive-web-design-modern-website-code-forbeginners/** **https://www.w3schools.com**


Frontend
HTML
CSS
JavaScript
CSS Frameworks
Bootstrap
Tailwind CSS
Bulma
Foundation
Primer CSS
Spectre CSS
Materialize CSS
Onsen UI
Semantic UI
Blaze UI
Pure CSS
Tachyons
CSS Preprocessors
SASS
LESS
ES6
Typescript
Package Manager
npm
yarn
JS Framework
JS Libraries
Angular JS
Angular ngx Bootstrap
Angular PrimeNG
Angular Material UI
Vue JS
Nuxt.js
script.aculo.us
Jquery
jQuery UI
jQuery Mobile
jQWidgets
jQuery EasyUI
React JS
Next.js
Ant Design
React Desktop
React Rebass
React Bootstrap
React Reactstrap
Blueprint.js
React Suite
React.js Evergreen
P5.js
Fabric.js
D3.js
Collect.js
Underscore.js
Lodash
TensorFlow.js