Monitoring, Accessibility, PWAs, and Mobile Applications

CIS 1962 (Winter 2026)
March 26th, 2026

Penn Engineering

Lesson Plan

5	Website Monitoring
13	Accessibility/ a11y
27	PWAs
43	Mobile Development

Logistics

- Homework 7 now due next week (4/5)
- Final Project details released!
  - https://upenn-cis-1962.vercel.app/hw/finalproject
  - You may work solo or as a pair.
  - Look out for Canvas assignments to submit this!
  - Project proposal due on April 5th (Please submit this on time so we can provide feedback)
  - Keep track of Milestone due date!

Review Activity

https://edstem.org/us/courses/91614/lessons/162384/slides/954291

Let's review some content from the previous lecture before we start!

Website Monitoring

What's the importance of monitoring and in what ways can we monitor user activity in apps?

Monitoring

Web app development doesn't end at deployment. It is important to monitor user actions and app performance, and also detect suspicious activity that could lead to attacks.

- Front-end Logging
- Monitoring Services
- Performance metrics for optimization
- User activity data for UI/UX research

Logging in the Frontend

You should already be familiar with being able to `console.log()` on the frontend of your app. You may also send your logs to a third-party service or a backend.

For monitoring purposes, logging helps with:

- Logging app errors
- Logging user actions (button click, navigation)
- Performance metrics (page load times)

Monitoring Services

Monitoring services connected to your app provide a clean dashboard for errors and logs. These may also help automate your monitoring, see trends, and output alerts to services like Slack or email.

Performance Metrics

A deployed app may differ wildly when used by real users from a development environment.

Below are some important metrics you can test:

- First Contentful Paint (FCP)
- Largest Contentful Paint (LCP)
- Cumulative Layout Shift (CLS)
- Interaction to Next Paint (INP)

You can use an API like PerformanceObserver or the `web-vitals` package to measure these!

User Analytics

Monitoring the behavior of users, such as where they click and how long they stay on your website, helps developers see patterns and prioritize chances to improve user experiences.

Things you may monitor include:

- Navigation, Clicks, and Scroll Depth
- Session length and bounce rates (how many people immediately leave instead of clicking more links)
- Interactions like video playing, image carousel use, etc.

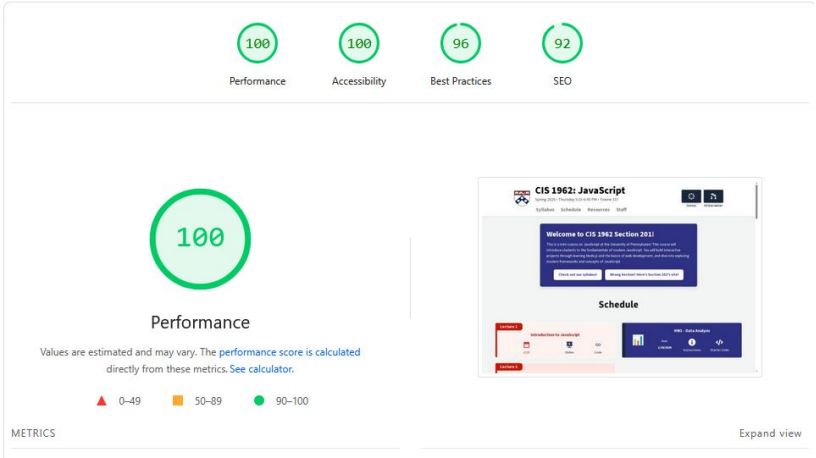
Class Demo

https://edstem.org/us/courses/91614/lessons/162384/slides/954309

Google Lighthouse Audit

Google Lighthouse is a service that provides quick and easy analysis of performance metrics and various other metrics, like accessibility and SEO.

[Audit for our Class Website]



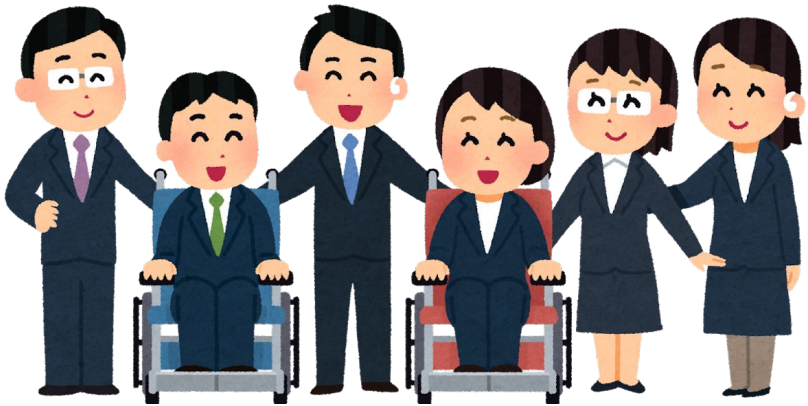
Accessibility

How do we design our apps to be accessible for users with disabilities and different circumstances?

Why Accessibility Matters

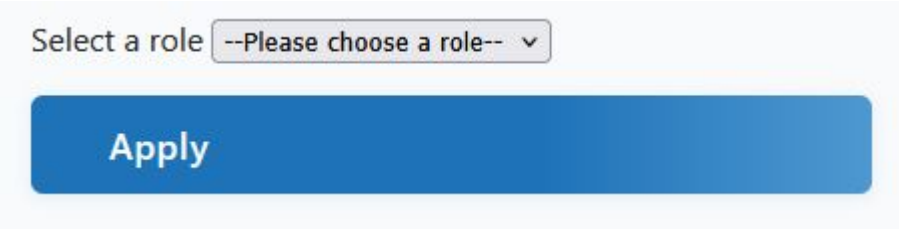
1 in 4 adults have disabilities.

Designing websites to be accessible helps reach wider audiences and comply with legal standards!



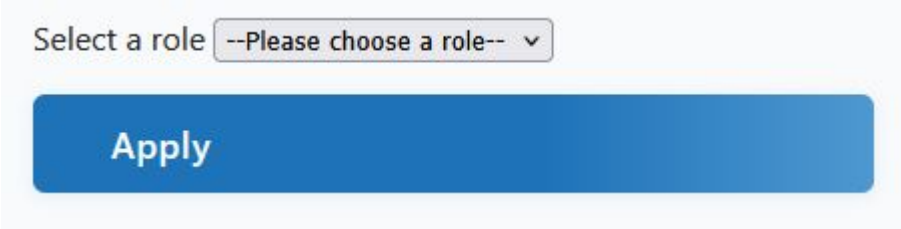
Why Accessibility Matters

```
...
<div onclick="submitForm()" class="fancy-button">
  Apply Now
</div>
```



Why Accessibility Matters

```
<button
  type="submit"
  class="fancy-apply-btn"
  aria-label="Apply for selected job"
>
  Apply
</button>
```



Bad Accessibility is Noticeable

https://www.art.yale.edu/

A11y and WCAG

A11y is an abbreviation for **accessibility**, and represents the initiative to design digital spaces that everyone can use. This means considering not just disabilities, but also slower internet connections and mobile devices.

WCAG (Web Content Accessibility Guidelines) is a set of standards are recommendations for developing accessible web content.

WCAG2

Content should be Perceivable

Information and user interface components must be presentable to users in ways they can perceive.

Examples:

- Text alternatives for images (alt text)
- Captions for videos
- Good color contrast (avoid bad color contrast)
- Use semantic HTML tags to structure a page and provide purposes and relationships between elements

Content should be Operable

Interface components and navigation must be usable by all, including with a keyboard.

Examples:

- All operations, interactions, and navigation can be performed with keyboard only
- Define focus pseudo-classes when using a keyboard
- Avoid content flashing quickly (avoiding seizures)
- Provide time for users to read and use content

Content should be Understandable

Information and operation of the UI must be clear and predictable.

Examples:

- Provide clear and simple language, and provide definitions for abbreviations, vernacular, and jargon
- Navigation is consistent across the entire interface
- Changes of context (like updates and navigation) should be initiated by the user, rather than sudden and automatic
- Help users avoid and navigate through errors (like clearing form errors, providing error feedback explicitly)

Content should be Robust

Information and operation of the UI must be clear and predictable.

Examples:

- Use Semantic HTML!
- Maximize compatibility between different browsers and devices such as mobile and tablet
- Use ARIA to provide roles and assistance where HTML is not enough

ARIA

Accessible Rich Internet Applications (ARIA) are a set of roles and attributes you can add to HTML to add meaning to pages for assistive technology like screen readers.

While many semantic HTML elements are already accessible (<button>, <nav>), it's important to use ARIA to provide extra context for custom elements or plain elements.

```
<button aria-pressed="true">
```

```
<div aria-live="polite">
```

Testing a11y

Can you navigate through your whole app with only the Tab, Space, and Enter keys?

Test page content with screen reader apps like NVDA

Use automated tools like:

- Lighthouse
- Axe DevTools

Class Activity

https://edstem.org/us/courses/91614/lessons/162384/slides/955185

Resources

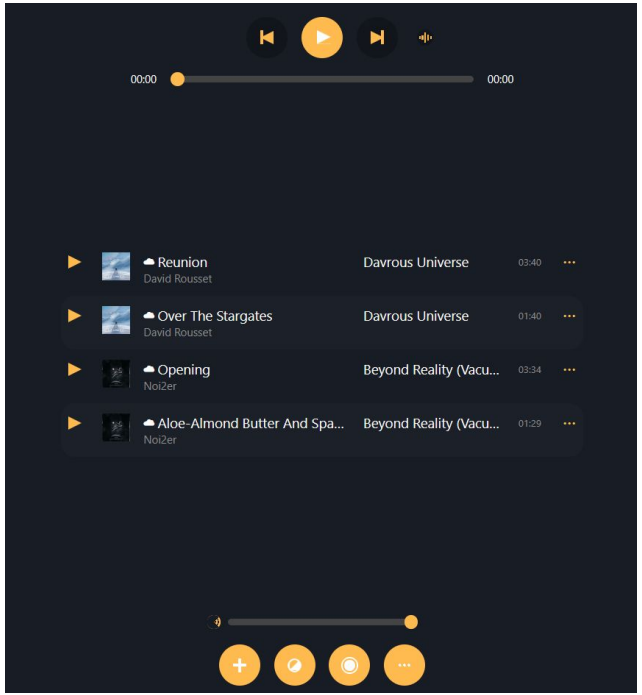
- a11y Project
- WCAG2
- MDN ARIA Docs

Progressive Web Applications

How do you make web apps that behave like normal applications, that can be accessed offline?

Imagine this...

You've made a nice web application that plays music, but you feel like it could work well as an offline mobile application.



https://microsoftedge.github.io/Demos/pwamp/

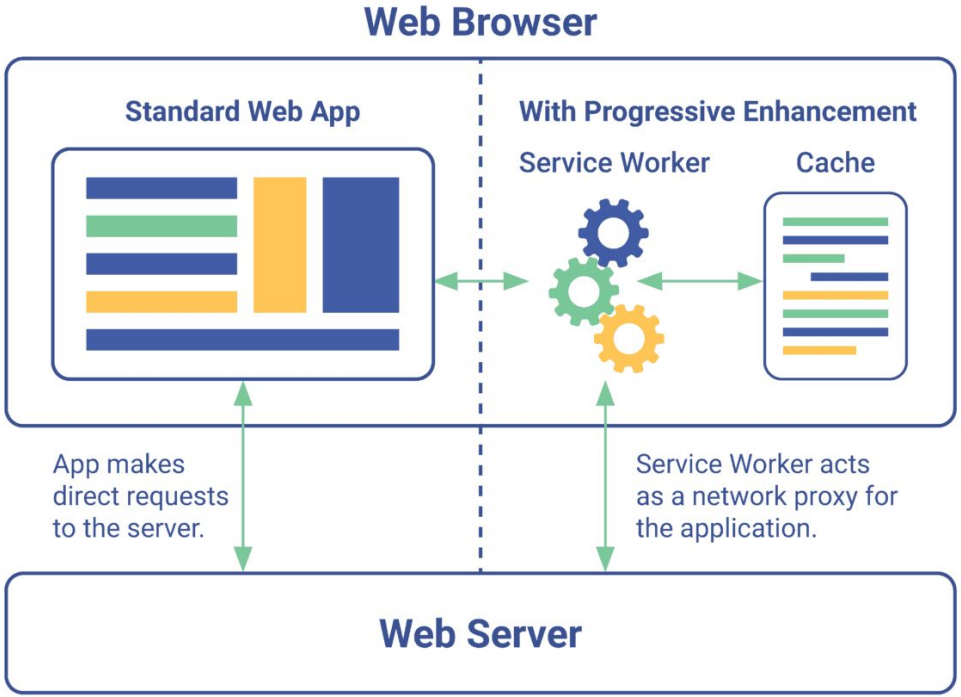
Enter... the PWA

Progressive Web Applications (PWAs) arose in the late 2010s as a response to mobile applications that were faster and more responsive.

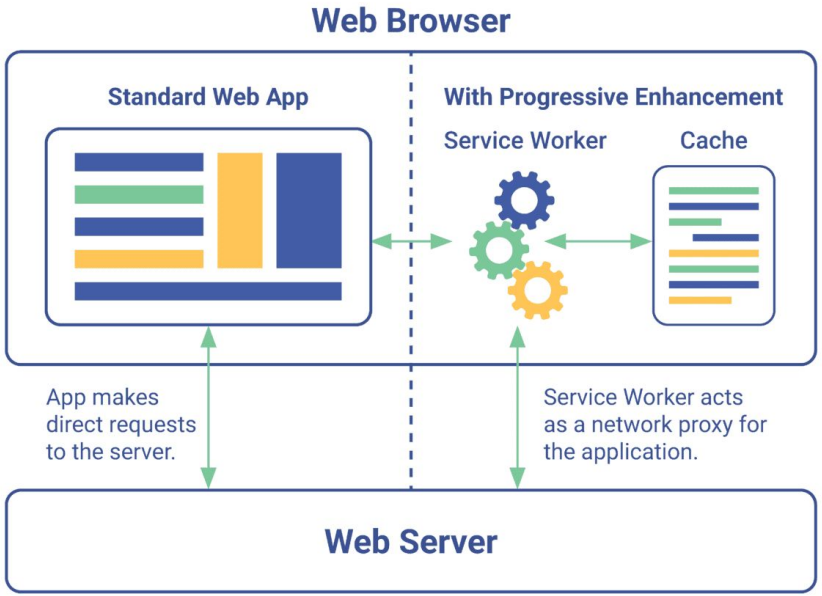
PWAs have the following features:

- Responsiveness (works on many screen sizes, like mobile)
- Installable
- Offline Use with Caching
- OS Integration & App Store Integration

PWA Architecture



How Does it Work?



Service Worker: A special JS file that runs in the background separate from the app. Registered on first visit, caching key files locally.

Web App Manifest: A JSON file linked to the HTML that contains data about the app, and allows it to be installed

Web Workers: Background Tasks

Web workers allow us to run tasks in the background so that the main thread isn't blocked.

They have limited access to structures like the DOM, but work well for heavy computation tasks.

Web Worker Example

```
// worker.js
self.onmessage = function(e) {
  const limit = e.data;
  const primes = [];
  for (let n = 2; n <= limit; n++) {
    let isPrime = true;
    for (let i = 2; i <= Math.sqrt(n); i++) {
      if (n % i === 0) { isPrime = false; break; }
    }
    if (isPrime) primes.push(n);
  }
  // Send result back to main thread
  self.postMessage(primes);
};
```

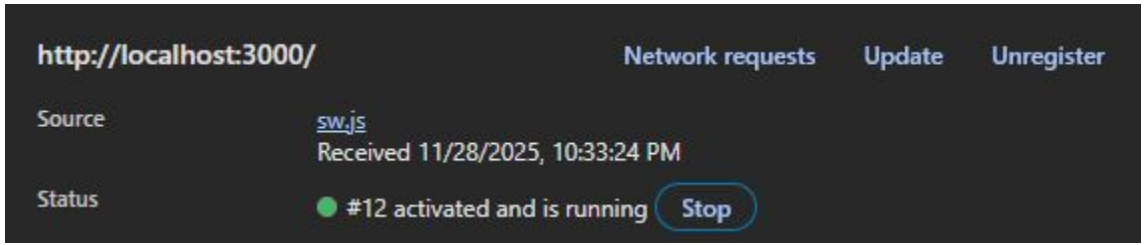
```
// main.js
const worker = new Worker('worker.js');

worker.onmessage = function(e) {
  console.log("Primes:", e.data);
  document.getElementById('output').textContent =
"Primes: " + e.data.length;
};

// find all primes up to 100,000
worker.postMessage(100000);

document.getElementById('output').textContent =
"Calculating primes...";
```

Service Workers

Service workers are a special type of Web worker, and are the heart of PWAs.

They are the middleman between the app and network requests, and can handling caching and background tasks.

Example) Chrome > DevTools > Application



Service Workers: Registration

To have a service worker in your app, you need to register it!

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function () {
    navigator.serviceWorker
      .register('/sw.js')
      .then(function (registration) {
        console.log("ServiceWorker registration successful: ", registration.scope);
      })
      .catch(function (error) {
        console.log("ServiceWorker registration failed: ", error);
      });
  });
}
```

`sw.js` in the above code contains the service worker, including installation, activation, fetching, and caches.

Service Workers: Installation

The Service Worker API provides events to handle various parts of a service worker lifecycle.

self: refers to the service worker global scope

"install": triggers the first time a service worker is registered.

```
// This installation caches all the critical files of the app so it can work offline!
self.addEventListener("install", (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => cache.addAll(URLS_TO_CACHE))
  );
  self.skipWaiting();
  console.log("Service Worker installed & pre-cached");
});
```

Service Workers: Activation

"activate": occurs after installation, when the service worker takes control (managing network requests)

```
// This activation removes old caches for previous service worker versions and then
lets the service worker take control.
self.addEventListener("activate", (event) => {
  event.waitUntil(
    caches.keys().then(names =>
      Promise.all(names.filter(n => n !== CACHE_NAME).map(n => caches.delete(n)))
    )
  );
  self.clients.claim();
  console.log("Service Worker activated");
});
```

Service Workers: Fetch

"fetch": This event fires whenever a network request is made, including getting the HTML/CSS/JS of the page.

```
// This fetch intercepts any network requests and retrieves the cached versions if
available
self.addEventListener("fetch", (event) => {
  event.respondWith(
    caches.match(event.request).then((response) => {
      return response || fetch(event.request);
    })
  );
});
```

Service Workers: Caching

If working in a secure setting (HTTPS or localhost), you can access the CacheStorage and store items there. Service workers will use this to store static site files to retrieve later.

http://localhost:3000

Bucket name *default*

#	Name	Response-T...	Content-Type	Content-Le...	Time Cached	Vary Header
0	/app.js	basic	application/...	396	11/28/2025,...	Accept-Enc...
1	/icon-192.png	basic	image/png	1,461	11/28/2025,...	
2	/icon-512.png	basic	image/png	1,178	11/28/2025,...	
3	/index.html	basic	text/html	677	11/28/2025,...	Accept-Enc...
4	/manifest.json	basic	application/...	428	11/28/2025,...	Accept-Enc...
5	/sw.js	basic	application/...	862	11/28/2025,...	Accept-Enc...

Installing a PWA

In order for a webapp to be installable as a standalone app, you must have:

- A manifest.json linked in the head

  <link rel="manifest" href="manifest.json" />
- A registered service worker
- HTTPS or local host (for security)

```
{
  "name": "My Demo PWA",
  "short_name": "DemoPWA",
  "start_url": ".",
  "display": "standalone",
  "background_color": "#f1f5f9",
  "theme_color": "#3b82f6",
  "description": "A simple PWA demo.",
  "icons": [
    {
      "src": "icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

Demo: A Basic Installable PWA

Let's demonstrate a basic PWA using all the previous code.

We'll use Google Chrome to test various offline features as it has robust testing features in its DevTools!

https://edstem.org/us/courses/91614/lessons/162384/slides/955189

PWA Examples

- https://microsoftedge.github.io/Demos/pwamp/
- https://microsoftedge.github.io/Demos/1DIV/dist/
- https://2048.love2dev.com/
- https://m.uber.com/go/home
- https://open.spotify.com/
- https://canvas.upenn.edu/

*While you can't see the service worker on some of these links, it may be registered at a narrower scope

Mobile Development

How do you develop web apps to be used with mobile devices?

JavaScript... on your phone?

Most everyone has a smartphone, and those phones can access browsers. This means we will see JS's influence on mobile devices as well!

It's important to consider the mobile market when developing your applications. It will likely make up a large part of your users!

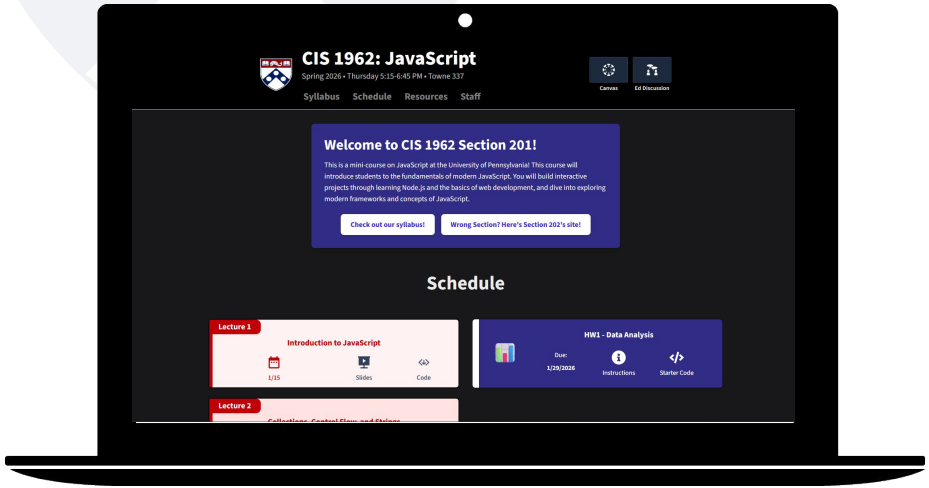
Approaches to Mobile Development

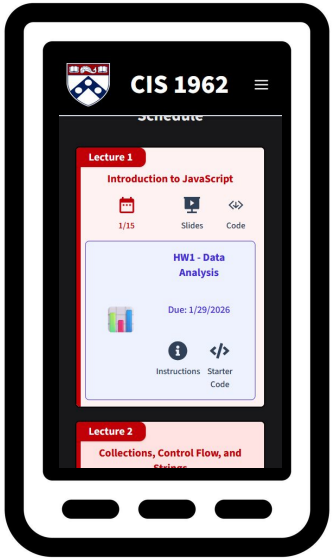
There are multiple approaches to developing apps:

- Responsive web applications
- PWAs
- Hybrid/WebView Apps (Cordova, Ionic)
- Mobile Native Apps (React Native, Flutter)

Mobile Responsiveness

Even if you don't want to build a mobile app at the outset of your project, it's important to make your app adapt to a mobile layout.





Mobile Responsiveness in CSS

You can use a media rule in CSS to define styles used at certain screen sizes:

```
@media (max-width: 600px) {
  .container {
    flex-direction: column;
  }
  .sidebar {
    width: 100%;
  }
}
```

Additionally many CSS libraries include support for mobile responsiveness, including Tailwind:

```
<h2 className="text-2xl sm:text-3xl md:text-4xl text-indigo-800 font-bold mb-2 flex gap-2">
```

Mobile-First Design

Because many users will browse on phones, and Google uses mobile-first indexing for searches, it's often best to take a **mobile-first design** approach to building the front-end of your applications.

In mobile-first design:

- Design small screens first
- Progressively add features to larger screens to adapt the layout

Class Activity

https://edstem.org/us/courses/91614/lessons/162384/slides/955191

Mobile UNResponsive Examples

Many "old internet" websites have bad mobile responsiveness.

However there are still some modern examples!

- https://www.goodreads.com/
- https://www2.pnwx.com/
- https://arngren.net/
- https://www.spacejam.com/1996/
- https://berkshirehathaway.com/
- https://www.weather.gov/

PWAs and Mobile Development

PWAs let you have a mobile app-like experience on a browser.

PWAs can be installed like mobile apps and accessed like you would a mobile app downloaded from an app store (without going through an app store).

Develop your app in JS, and distribute your app through a browser or a PWA download!

WebView Wrappers

WebView wrappers are a hybrid mobile option: They are apps written and developed in JS, but packaged and rendered in a native app "shell". Essentially, a mini-browser in an app.

Examples of WebView Frameworks include:

- Apache Cordova
- Ionic
- Capacitor

Native JavaScript in Mobile

The term "native JS" in mobile development refers to using JavaScript to develop apps on mobile devices, as opposed to using traditional native languages like Swift (iOS) or Kotlin/Java (Android).

Frameworks like React Native and NativeScript provide the bridge between JavaScript and actual native UI widgets

(i.e. Image > UIImageView (Swift)

Mobile Resources

- https://reactnative.dev/
- https://www.evoba.com/learn/mobile-responsiveness
- https://cordova.apache.org/
- https://www.pwabuilder.com/
- https://caniuse.com/ (Cross browser compatibility)