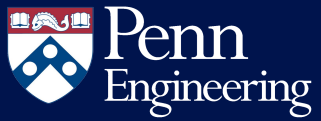
PWAs, Mobile Dev, and the State of JavaScript

CIS 1962 (Fall 2025)
December 1st, 2025



Lesson Plan

5	Progressive Web Apps
21	Mobile Development
33	JavaScript: Further Study Topics
43	The Current State of JavaScript

PollEverywhere!

We will use PollEv to take attendance and do polls during lecture. Scan this QR Code or use the link to join for attendance! (Please add your name for identification)

PollEv.com/voravichs673

Logistics

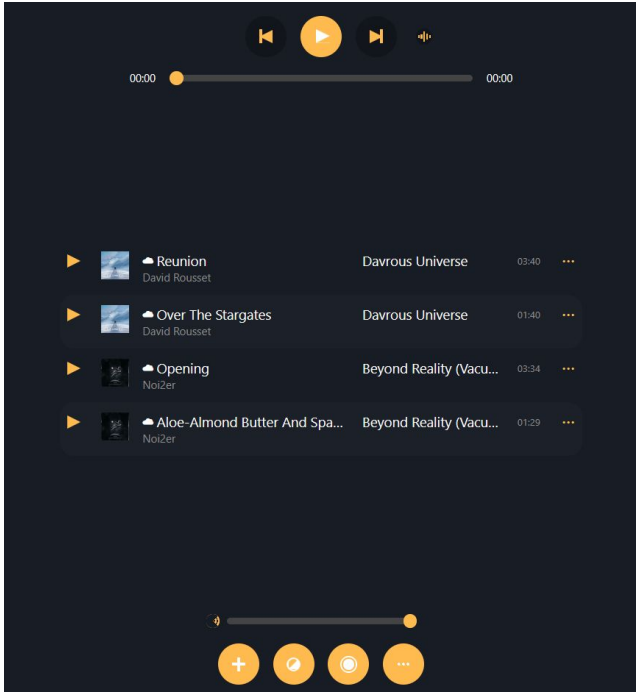
- Project Milestone 2 due tonight (Optional)
- Homework 5 due tonight
- Final Project Presentations next week!
  - Let us know via email/Ed if you cannot make it
  - Submit presentation slides to Canvas, including:
    - An introduction to your project: what functionality does your project have for users?
    - Tech stack: What frontend and backend technologies does your project use?
    - You should prepare to demo your app during your presentation as well.
    - You will have **5-6** minutes to present during class.

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/

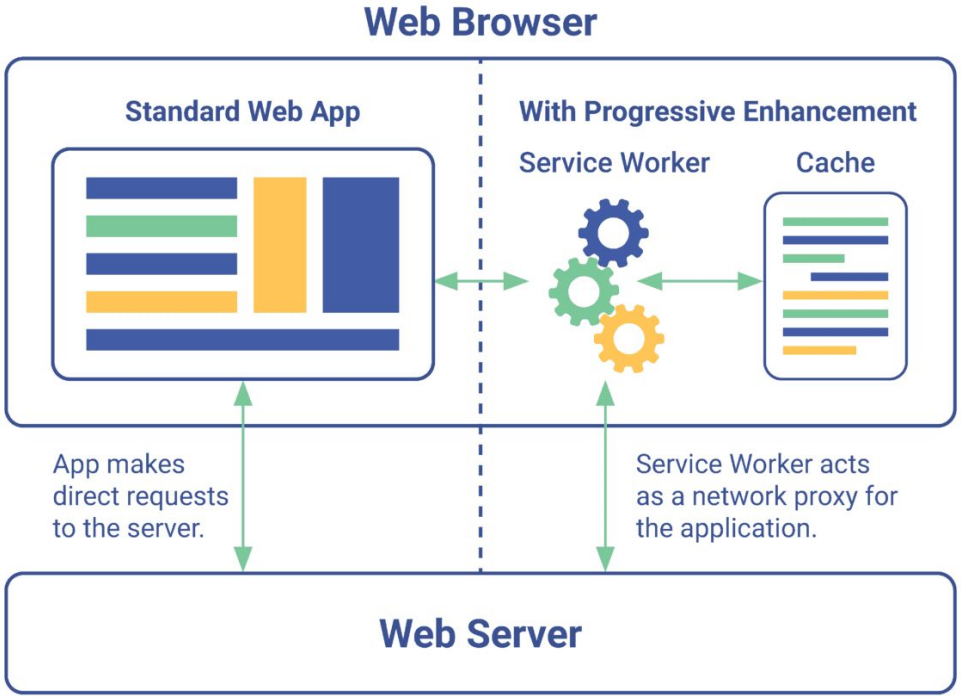
Enter... the PWA

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

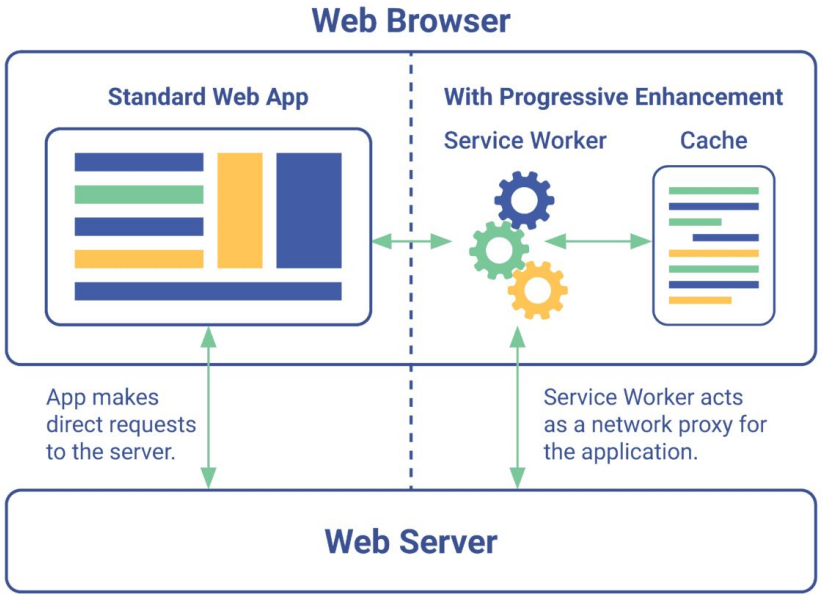
PWAs have the following features:

- 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);
};
```

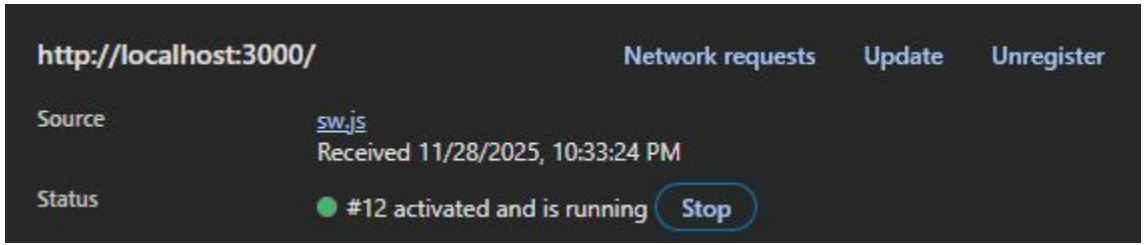
```
// main.js
const worker = new Worker('worker.js');

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 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!

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!

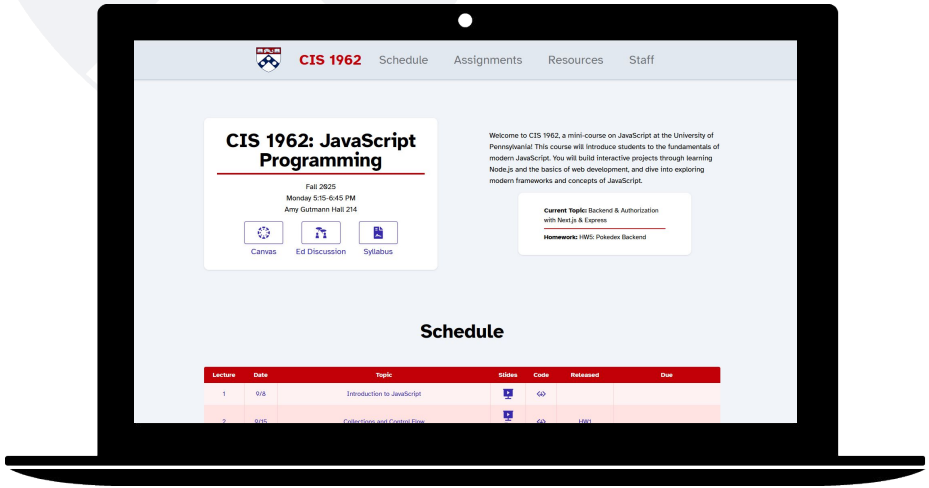
Approaches to Mobile Development

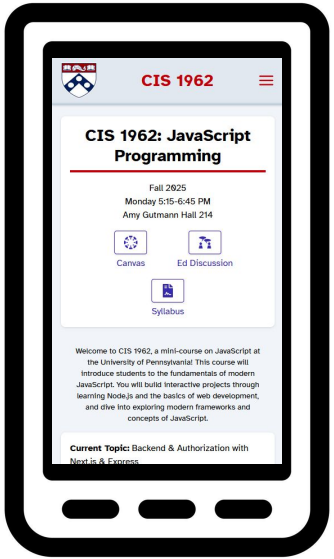
There are multiple approaches to developing apps:

- 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:

```css
@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

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://berkshirehathaway.com/

Demo: Class Website on Mobile

Let's demonstrate how you'd test mobile responsiveness on a website, by looking at our class website and seeing how it changes for mobile screen sizes!

Many browsers, in their DevTools, will have a "Responsive Design Mode" that will allow you to test different mobile screen sizes. This can also be used to test tap interactions well.

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)

JavaScript: Further Study Topics

What more is there to learn about JavaScript?

Congrats, you made it!

You made it to the end of this mini-course!

Over the course of this semester, you've learned the basics of working with JavaScript to build web applications and have started building sophisticated full-stack web applications of your own!

What's next?

What's left of JavaScript?

We've covered JS up from basic syntax and infrastructure, to working with HTML/CSS/JS, to frameworks and full-stack applications.

If you're interested in pursuing JS further, you can look at:

- **Further Frameworks** (Angular, Vue, Preact)
- **WebAssembly** (JS + other languages like Rust, C++, Go)
- **Edge Computing** (Cloudflare workers, Vercel edge functions)
- **Other Backend Frameworks** (Fastify, NestJS, GraphQL)
- **Authorization/Authentication** (OAuth, Auth0, Firebase Auth)

More Frameworks

While React and Next.js are quite popular, there are many frameworks out there that companies use, either alongside React or in place of it.

- **Angular**: Google, Microsoft Office web apps
  - Strong TS support, dependency injection, built-in testing, routing, forms, and states
- **Vue**: Alibaba, Xiaomi, Baidu, Nintendo, Grammarly
  - Easy learning curve, great docs, elegant reactivity
- **Preact**: Uber, Lyft, Pinterest
  - React-like API, smaller bundle for performance

WebAssembly (Wasm)

Web Assembly (Wasm) is a low-level format that lets code from languages like C, C++, Rust, and Go run alongside JS.

The fast speed afforded by these languages lets you do a lot more compute-heavy code on the web, and make use of existing code and libraries from the languages you bring over.

The code is compiled into a `.wasm` file that runs using JS.

Examples of WebAssembly

- Conway's Game of Life
- lichess.org
- Figma
- AutoCAD Web
- Commander Keen Web Port
- Diablo Web Port

Edge Computing

Edge Computing is a distributed computing model where data served at a physically closer "edge" server rather than a centralized server. This results on lower latency, and relies on multiple distributed server locations.

This has allowed certain platforms like Cloudflare and Vercel to deliver static content (in Content Delivery Networks or CDNs) and actual code (Edge functions/runtimes) on edge servers.

Edge Computing Examples

Cloudflare Workers

- Full-stack deployment platform using edge computing
- Low latency due to having 300+ data centers for physically close access

Vercel Serverless Edge Functions

- Vercel handles server management and delivers content from the closest servers
- Can configure content tailored to different regions when specified in settings (default is USA East, can pay to use different regions)

Other Backend Technologies

oRPC: Helps link clients and servers with end-to-end type safety and OpenAPI standards

GraphQL/Apollo Server: Data querying language and the library to create a GraphQL server. Helps manage endpoints for getting and mutating data.

Fastify: Fast, efficient alternative to Express with Native TS support and great docs

NestJS: Comprehensive backend architecture that uses Express/Fastify under the hood, with support for WebSockets, MicroServices, GraphQL, and more

Auth and the Other Auth

OAuth: standard protocol for authentication, allows third party apps (like Google, Github and Facebook) to be integrated as authorization and authentication mechanisms

Auth0: A cloud-based authentication and authorization platform that uses protocols like OAuth into apps. Can handle complex auth flows like SSO (Single sign-on, like Penn SSO) and social logins.

JavaScript: What's Next?

What's in store for the future of JavaScript?

JavaScript: The Current State

Current Version: ECMAScript 2025 (ES16)

- Iterator Objects
- Set Methods: intersection, difference, super/subset, disjoint
- Promise.try for handling sync and async errors in a promise chain

Future Version: ECMAScript 2026 (ES17)

- Proposal: `do` expressions (more robust expressions)
- Proposal: Pattern Matching (Better switch statement)
- `using` keyword: explicit resource management

JavaScript Trends

- Dominance of TypeScript in development workflows
- The rise of WebAssembly alongside JS
- Standardization of generative AI in development
- Baked-in security features to reduce XSS and CSRF
- Low-cost serverless computing
- Framework agnostic developers

The End, and Thank You

Thank you all for a wonderful mini-course experience!

See you at the final project presentations :)