Capabilities

- Always check for feature support
- Powerful web
- Empowering your PWA
  - Green capabilities
    - Basics
    - Hardware and sensors
    - Operating system integration
  - Light-green capabilities
    - The basics
    - Hardware and sensors
    - Operating system integration
- Resources

Progressive Web Apps can do more than just rendering content on the screen or connecting to web services. PWAs can deal with files from the file system, they can interact with the system's clipboard, they can access hardware that is connected to the device, and much more. Every Web API is available for your PWA, and there are some extra APIs available when your app is installed.

You can use What Web Can Do Today to know what's possible on each platform. For individual APIs or capabilities, you can use Can I Use or the browser compatibility tables on MDN.

Always check for feature support

The first letter in PWA stands for progressive, and it comes from the idea of progressive enhancement and feature detection. You should not expect your app to work the same way on every device. The diversity of contexts and abilities on billions of devices in different countries makes PWAs an excellent platform, thanks to their progressiveness.

This means that you need to develop your app in layers that may not be available on every device and to check availability before usage.

You need to check with JavaScript if an API exists before using it or ask an API if a service is available on that particular device.

Powerful web

The web is super powerful today. For example:

- You can build a hyperlocal video chat app with WebRTC, geolocation, and push messages.
- You can make an app installable.
- You can add video effects with WebAssembly.
- You can even bring it into virtual reality with WebGL and WebXR.

While there are many capabilities on the web, there are still gaps, and that's where the web capabilities project comes in. The project describes a group of APIs to increase what the web is capable of. We discussed this project and how you can participate in it in the Experimental chapter.

Empowering your PWA

Let's split the PWA capabilities APIs into four groups:

- Green: APIs available on every browser on every platform, when technically possible. Most of them have been shipped for many years, they are considered mature, and you can use them with confidence. An example API from this group is the geolocation API.
- Light green: APIs are available only on some browsers. Considering the lack of support on some platforms, they matured within the supported subgroup of browsers so that you can use the capability confidently on them. An example API from this group is WebUSB.
- Yellow: experimental APIs. These APIs are not yet mature; they are only available on some browsers, and within tests or trials. We talked some about these capabilities in the Experimental chapter.
- Red: the group for APIs that you can't use in a PWA, and where plans to add them are still long term. An example API from this group is geofencing.

Some capabilities require user permission: in most cases, the permission dialog appears on first usage. Today, you can request a single permission at a time, but in the future, it may be possible to request many permissions in one dialog on some platforms, using the Permissions API.

Green capabilities

Below is a list of the most important capabilities you can use in your PWA.

Basics

- *Caching files locally* using the Cache API, as seen in the Caching chapter.
- *Executing tasks in threads* using web workers, as we saw in the Complexity management chapter.
- *Managing network requests* with different strategies in a service worker, as seen in the Service workers chapter.
- *2D Canvas* for rendering 2D graphics on the screen using the Canvas API.
- *2D and 3D high-performance Canvas*, or WebGL, for rendering 3D graphics.
- *WebAssembly*, or WASM, for executing low-level compiled code for performance.
- *Real-Time communication*, using the WebRTC API.
- *Web Performance* APIs to measure and help provide a better experience. See the Performance API guide for more information.
- *Store data locally* with IndexedDB and storage management to query quota and request persistent storage, as seen in the Offline data chapter.
- *Low-level audio* thanks to the Web Audio API.
- *Foreground detection* using the Page Visibility API.
- *Network communication* using the Fetch API and the WebSocket API.

Hardware and sensors

- *Geolocation* obtains the user's location through different providers, such as satellite or Wi-Fi through the Geolocation API.
- *Camera and microphone* receive media streams from cameras and microphones using the Media devices interface.
- *Sensors* gather real-time information from the accelerometer, gyroscope, magnetometer, and others using the Sensors API or older interfaces, such as DeviceMotionEvent and DeviceOrientationEvent. On Safari, you need to use a non-standard permission dialog request to use them.
- *Touch and pointer* access information about all the touches and pointer-based clicks you make with your finger, a mouse, a trackpad, or a pen, thanks to Pointer events and Touch events.

Safari also supports Gesture events, a non-standard API for detecting rotation and pinching gestures, which should be used with care.

- *Gamepads* to read information coming from standard gamepads and joysticks connected to the device using the Gamepad API.
- *Biometric authentication* (such as face or fingerprint recognition) using Web Authentication or WebAuthn.

Operating system integration

- *Speech synthesis and voice recognition* use the platform's installed voices to speak to the user and to recognize what the user is saying, thanks to the Web Speech API.
- *Share content* from your PWA to other apps and places on the device, thanks to the Web Share API, as we'll see in the OS integration chapter.
- *Access the clipboard* to save and retrieve content from the clipboard in different formats, thanks to the Clipboard API, as I showed in the OS integration chapter.
- *Manage user's credentials and passwords* using the Credential Management API.
- *Enable picture-in-picture* video playing within the OS using the picture-in-picture API.
- *Render content in fullscreen* thanks to the Fullscreen API, as I showed in the Windows chapter.

You can integrate your PWA with many platform-specific apps using URI schemes and universal URLs, as I showed in the OS Integration chapter.

Light-green capabilities

Here is a list of the most important capabilities you can use in your PWA, with the caveat that they may not be available on every browser.

The basics

- *WebGL 2.0*, the new version of the WebGL spec to match OpenGL 3.0.
- *Codecs*, low-level access to the individual frames of a video stream and chunks of audio; useful for web applications that require full control over the way media is processed through the Web Codecs API.

Hardware and sensors

- *Advanced camera controls* to access pan, tilt, and zoom controls, in addition to the Media APIs.
- *Bluetooth LE* to communicate with Bluetooth low-energy devices near the user using the Web Bluetooth API. Check out communicating with Bluetooth devices over JavaScript for

more information.

- *Near-field communication* to exchange data over NFC via light-weight NFC data exchange format (NDEF) messages, such as an NFC tag or card using the WebNFC API. You can also read Interact with NFC devices on Chrome for Android for more details.
- *Serial peripheral* for low-level access to devices connected to the device using a serial port, USB, or serial over Bluetooth using the Web Serial WPI. In the following link you can learn how to read from and write to a serial port.
- *USB device* access to communicate with devices connected via USB WebUSB API. Check out access to USB devices on the web for more information.
- *Human interface devices* let your PWA interact with any kind of device prepared for human interaction that is uncommon, using the WebHID API. Check out this guide on connecting to uncommon HID devices.

Accessing hardware devices on the web offers a comprehensive guide to understanding how different hardware APIs interact with each other.

- *Ambient Light* reads the current light level or illuminance of the ambient light around the device, in addition to the Sensors API.
- *Vibration* gives the user haptic feedback when something happens, if the device supports it, through the Vibration API.
- *Recording media* captures the data generated by a MediaStream or HTMLMediaElement object (such as a `<video>` tag) for analysis, processing, or saving to disk, thanks to the MediaRecorder API.
- *Applying a wake Lock to the screen* prevents the device from dimming, or locking the screen, when your PWA needs to keep running, using the Screen Wake Lock API.
- *Virtual reality* enables you to use a headset and other devices in your PWA, thanks to the WebXR Device API.
- *Augmented reality* can be achieved in your PWA in many ways, such as using the WebXR Device API or the Safari Quick Look app for AR content.
- *Detect inactive users* with the Idle Detection API.
- *Orientation lock* locks the orientation to portrait or landscape while the PWA is on the screen, thanks to the Screen Orientation API, or the `orientation` property of the Web App Manifest for installed apps.
- *Present content* on projectors and secondary displays, thanks to the Presentation API.
- *Lock a pointer* to receive delta movement information from pointers (mice, trackpads, and pointers) instead of position values—useful for some games—thanks to the Pointer Lock API.

Operating system integration

- *Read and write files* on the device, thanks to the File System Access API, as you saw in the OS Integration chapter.
- *Get content from other apps* thanks to Web Share Target, as I showed in the OS integration chapter.
- *Get contact data* using the Contact Picker API, as shown in the OS integration chapter.
- *Sync in the background* while the PWA is not used, thanks to the Background Synchronization API.
- *Task scheduling* while the PWA is not in use, thanks to the Web Periodic Background Synchronization API.
- *Send notifications* using Web Push and Web Notifications APIs.
- *Transfer files in the background* while the user is not using your PWA, thanks to the Background Fetch API.
- *Integrate your media playing* with the operating system using the Media Session API.
- *Manage payments in your PWA*, thanks to the Payment Request API. Apple also offers an Apple Pay JS library on top of the Payment Request API.
- *Query network status*, such as connection type (4G, WiFi) and a Save Data flag using the Network Information API.
- *Capture the user's screen* for screencast or screen sharing using the Screen Capture API.
- *Detect shapes* using on-device hardware-accelerated detectors, including barcodes (human faces and text OCR are still in development) using the Shape Detection API.
- *Query a device's memory* using the Device Memory interface.
- *One-time passwords over SMS* let you automatically receive a code via SMS sent from your server using the WebOTP API. Safari implements a solution subset based on the `<input>` element. Read more about it in WebKit's blog.
- *Manage the virtual keyboard* that appears on mobile devices screens using the Virtual Keyboard API.

If you publish a PWA to some app catalogs and stores, you may access additional APIs. For example, if you publish a PWA to Google Play using a Trusted Web Activity, you can use the Digital Goods API to charge your users for subscriptions and premium content.

If you have a legitimate use case that cannot be realized with the present set of APIs, you can submit a request to analyze the use case for a possible future web API, as we'll see in the Experimental chapter.

Resources

- MDN: Web APIs
- What Web Can Do Today
- What PWA Can Do Today
- Can I Use
- iOS/iPadOS Approximate location
- Meet Face ID and Touch ID for the Web

□□□ #3
□ □□□ □□□ 2026-10-31 22:55:05 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:28 UTC