Progressive Web Apps (PWAs)

Lorenz Aichner, Sarah Hörtnagel, Hagen Leitner, and Fabian Szakács

706.041 Information Architecture and Web Usability 3VU WS 2025/2026
Graz University of Technology

14 Dec 2025

Abstract

Progressive Web Apps (PWAs) have rapidly grown into a powerful alternative to native mobile and desktop applications. They represent a significant evolution in modern web development, combining the accessibility of traditional web sites with the speed, reliability, and app-like experience of native applications.

This survey provides a overview of the PWA technology and discusses the main advantages of PWAs for users and developers, such as cross-platform compatibility, Add to Home Screen functionality, no manual updates, reduced development costs, and independence from app stores. The core components of PWAs are explained, including service workers for offline access and push notifications, the web app manifest for installability, and the HTTPS protocol for a secure application. The current state of PWAs in 2025 is reviewed, including browser support on desktop and mobile devices. The installation process on mobile and desktop devices is explained step-by-step. As a practical example, a conference web site was implemented as a PWA, demonstrating the behaviour and functionality of PWAs.

© Copyright 2025 by the author(s), except as otherwise noted.

This work is placed under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Progressive Web Apps (PWAs) are web-based applications which provide a native-like experience while still being accessible through the web [Andrews 2025]. With this, PWAs solve the previous existing problem faced by developers and companies as to whether to create native applications, which perform well but must be developed separately for each platform, or to use traditional web sites, which work on all devices but have limited features. PWAs combine the strengths of both approaches. By using Web Application Programming Interfaces (Web APIs) and modern web technologies, it is possible to deliver a native app-like experience on various end devices, with a single codebase and low resource consumption. PWAs are also useful for reducing costs and dependence on internal tools, such as logistics dashboards or field service applications [LePage and Richard 2020].

The goal of this survey paper is to highlight key advantages, explain the core technologies which power PWAs, show practical installation flows, and review the current status of PWAs in 2025. In addition, current browser and platform support is assessed. Finally, a PWA was implemented for a small fictitious conference, for a practical demonstration of the technology.

1.1 Core Principles of PWAs

According to LePage and Richard [2020], for PWAs to feel like a native application, they have to follow three core concepts: they need to be capable, reliable, and installable.

In terms of being capable, web applications have gained significant power and benefited from a growing array of functionality over the last several years. Some examples include video chat apps, geolocation, Web Real-Time Communication (WebRTC), Web Graphics Library (WebGL), and push notifications. With the development of WebAssembly, web applications can now also use additional and more efficient ecosystems, such as C, C++, or Rust. Web apps also have access to functions provided by the operating system, such as the file system or copy and paste. With this growing feature set, PWAs are ever more capable of competing with native applications.

PWAs also need to be perform reliably with low or no internet connectivity. With a PWA installed on a device, it is possible to cache data locally and retrieve content from the local cache when needed. In addition, the application must run smoothly and quickly enough to still meet user expectations, as for a real native app.

When a PWA is installed, users expect it to provide an experience comparable to that of a native application. For example, the application should display an icon on the home screen of the device and it should be visible and accessible like any other installed app. After launching a PWA, the user expects that a dedicated window is opened, which supports all shortcuts provided by the used operating system.

1.2 Key Advantages

Progressive web apps combine the characteristics of traditional web sites and native applications, providing advantages which are particularly attractive in multi-device, bandwidth-constrained, and cost-sensitive contexts. This section highlights some of the key advantages of PWAs, based on the work of Keshri [2025] and BS Team [2025b].

1.2.1 Add to Home Screen (A2HS)

Add to Home Screen (A2HS) functionality provides the possibility to "install" a PWA on the desktop or home screen of a device. After installation, the PWA can be launched via an icon like a native app and runs in a dedicated browser instance, with minimal or no UI elements depending on the defined display mode. This feature lets users access the PWA instantly after installation without downloads or any further manual updates.

1.2.2 Offline Access

A defining feature of PWAs is their ability to work reliably under poor network conditions. This capability is primarily enabled by service workers, which intercept network requests and serve pre-cached resources when the network is slow or unavailable. As a result, users can still perform key tasks, including for example browsing previously visited content, filling in forms, or managing a shopping cart, regardless of their internet connectivity. Beyond improving perceived reliability, offline access also reduces the impact of latency and packet loss. From a user experience perspective, this behaviour is similar to that of native applications, significantly reducing frustration caused by long waiting for loading icons and error pages.

1.2.3 Cross-Platform Compatibility

PWAs are built using standard web technologies (HTML, CSS, JavaScript, and associated Web APIs) and can therefore target a wide range of consumer hardware with a single codebase [Svaiko 2020]. The same PWA can run in modern browsers on desktop operating systems, on mobile platforms, and even constrained devices such as ChromeOS-based systems or smart TVs with embedded browsers. While minor platform-specific adaptations may still be required, for example to account for input modalities or available APIs, the overall development and maintenance effort is significantly lower than maintaining separate native applications. This cross-platform property is a key factor behind the adoption of PWAs in organisations which prioritise consistent user experience across different devices.

1.2.4 Independence from App Stores

Unlike traditional native applications, PWAs do not require distribution through centralised app marketplaces. Users can discover, launch and install PWAs directly from the browser. This independence from app marketplaces offers several advantages. Firstly, developers do not have to undergo long review processes, enabling faster iteration and continuous deployment. Secondly, updates are delivered automatically via the web, eliminating the need for manual updates by the users. Thirdly, organisations can avoid certain fees, policy restrictions, or geographic limitations associated with specific app marketplaces. Finally, PWAs are indexed by search engines, supporting deep linking and standard web analytics. Together, these factors lower the barrier to distribution and make PWAs an attractive option for both small developers and large enterprises.

1.2.5 Performance Improvements

Due to efficient caching and the application shell architecture, PWAs are highly efficient. The application shell, which acts as a UI skeleton, immediately loads a basic structure of the user interface from the cache, so it is not necessary to fetch it from an external server. A service worker can cache content and data, reducing the need to fetch data over the internet [BS Team 2025b].

1.2.6 Push Notifications

With the support of a service worker, PWAs can use the functionality of push notifications. Since notifications can be triggered even when the app is not open, this further helps to increase the number of users returning to the PWA, enforce re-engagement of users with promotions, news updates, or new feature announcements and boost conversion rates by for example reminding users of limited-time deals [BS Team 2025b].

1.2.7 Search Engine Optimization (SEO)

As already mentioned, the use of PWAs increases prominence in search engines, since they are indexable like other web pages. Due to the smooth experience and innovative architecture, featuring efficient caching and fast performance, the user experience reduces bounce rates and motivates users to stay longer on the page, which contributes to a higher ranking of the PWA in search engines. Due to the efficient performance of PWAs, they tend to achieve higher SEO rankings. PWAs provide a user interface (UI) and user experience (UX) similar to native apps, supporting the mobile-first design approach promoted by search engines [Łabiński 2024].

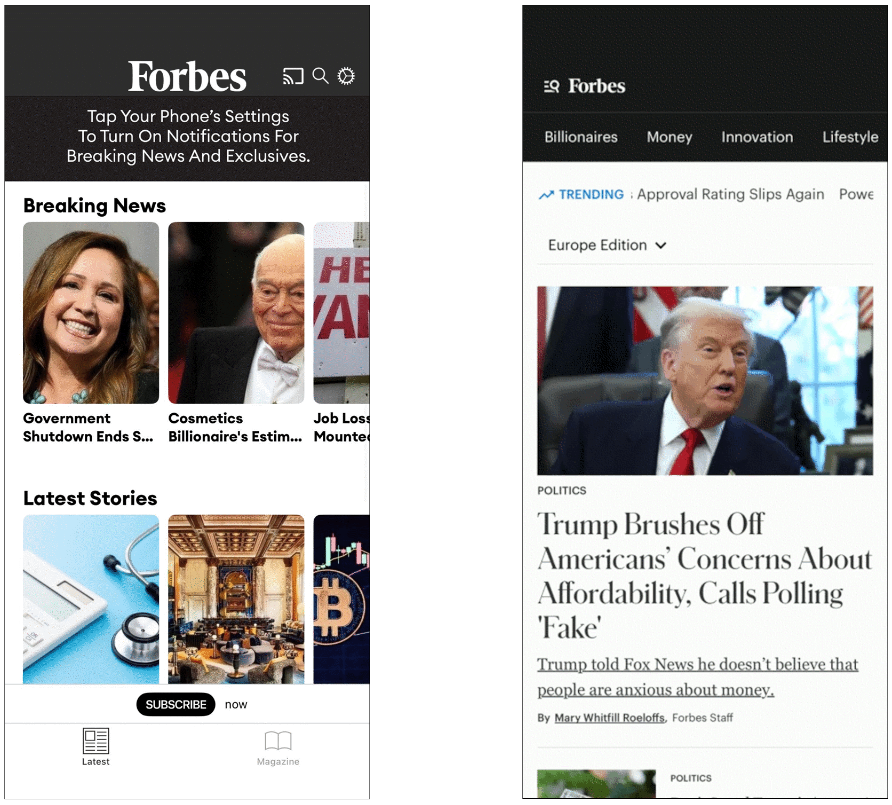
1.2.8 Reduced Development Costs

Specific native technologies and tools, such as those provided by Google or Apple, are not required for developing PWAs. Instead, developers can create a PWA using popular web development frameworks, such as React or Next.js. These web tools enable a single codebase to be deployed across multiple platforms. This approach reduces development costs, as updates do not depend on the App Store or Google's Play Store.

1.3 Real-World Examples

Many companies have publicly reported the benefits they have experienced through the adoption of PWAs. PWAs combine high performance, cross-platform availability, and cost efficiency, which makes them particularly attractive for companies. As described by BS Team [2025a], some notable examples of successful PWA implementations include: Starbucks, Forbes, X (formerly Twitter), Uber, Pinterest, Microsoft Office, Notion Amazon, and Spotify.

These examples demonstrate the broad adoption of progressive web apps across different industries and application areas, which additionally highlights the flexibility and effectiveness in improving user engagement and providing a near-native experience. Figure 1.1 shows the Forbes native app side-by-side with the Forbes PWA. The user interface of the PWA closely replicates that of the native application, demonstrating how PWAs can indeed deliver a similar look and feel while running entirely in the browser.



Tap Your Phone's Settings To Turn On Notifications For Breaking News And Exclusives.

Government Shutdown Ends S...

Cosmetics Billionaire's Estim...

Job Loss Mounte

Approval Rating Slips Again

Trump Brushes Off Americans' Concerns About Affordability, Calls Polling 'Fake'

Trump told Fox News he doesn't believe that people are anxious about money.

By Mary Whitfill Roeloffs, Forbes Staff

(a) Forbes native app. **(b)** Forbes PWA.

Figure 1.1: Visual comparison between the Forbes native app and its PWA. The user interface is nearly identical, demonstrating how PWAs can replicate native look and feel while running in a browser. [Screenshots taken by Sarah Hörtnagel.]

Chapter 2

Underlying Technologies

Three main technologies provide the basis for implementing PWAs: the web app manifest, service workers, and HTTPS.

2.1 Web App Manifest

The web app manifest is a short JSON file that makes the progressive web app discoverable and describes the details of the application [Grigsby 2018, page 62; Svaiko 2020]. The following entries in the manifest are required by Chromium-based browsers [MDN 2025a]:

- `name`: `name` is used in various contexts. Usually, it provides the label of the installed icon of the app, in the task manager, or in a list of installed web applications. If the `name` is too long, the `short_name` value can be used as an alternative [MDN 2025d].

- `icons`: `icons` is an array of objects defining the icons for the app. Developers are advised to define icons in multiple sizes to support different operating systems and screen sizes. At a minimum, a `192px` and a `512px` icon are required [MDN 2025c]. Each icon object must include a `src` member, which contains the icon file path. Optionally, developers can define the `type` to improve performance, causing the browser to ignore unsupported image formats and can use `sizes` to define the dimensions in which the icon can be used, which is especially useful for vector formats [MDN 2025a].

- `start_url`: `start_url` indicates which page should be displayed when the app is launched. According to Grigsby [2018, page 65], developers can also add parameters to the `start_url` in order to track how many users visit the web site from the home screen shortcut.

- `display`: `display` supports four modes [Grigsby 2018, pages 30–32; MDN 2025b]: `browser`, `minimal-ui`, `standalone`, and `fullscreen`. In `browser` mode, the app opens in a standard browser tab or window. The `minimal-ui` mode provides a minimal UI for navigation, while `standalone` mode presents the app like a native application within its own window. In `fullscreen` mode, the app fills the whole screen without any browser UI elements.

These four entries define the core properties of the PWA and are necessary for proper installation and operation. Additionally, the entry `prefer_related_applications` must be set to `false` or not defined. The visual appearance of the application can be further customised by setting the entries `background_color`, `theme_color` and `orientation` [Grigsby 2018, pages 30–32]. Listing 6.1 shows an example of a web app manifest.

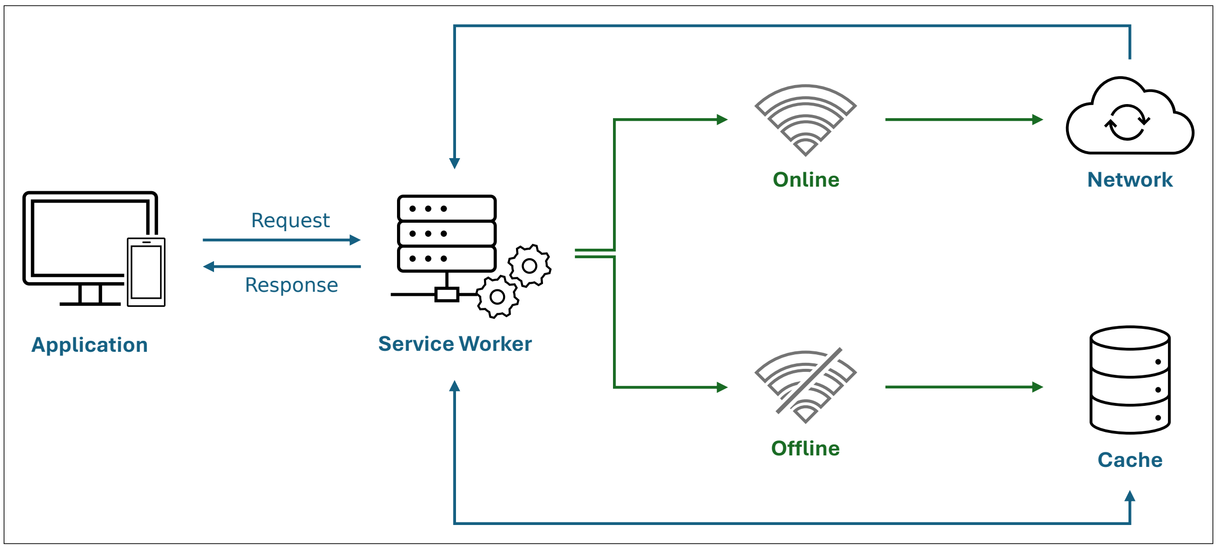


Figure 2.1: PWA behaviour in online and offline modes. The service worker intercepts requests to the server, and then determines whether to fetch data from the network or from the cache. [Illustration drawn by Sarah Hörtnagel.]

2.2 Service Workers

A service worker is a powerful technology for background tasks such as caching and handling push notifications [Grigsby 2018, pages 7–8; Svaiko 2020]. Service workers are JavaScript scripts, which can be used to intercept network requests in order to serve cached pages when the application is offline, acting as a network proxy [BS Team 2025b]. Figure 2.1 illustrates the service worker as the interface between application, network, and cache.

A common use of service workers is to cache all assets needed to render the application and therefore be able to display the app shell as soon as the application is launched [Grigsby 2018, page 19]. Additionally, the content of the web site can also be cached and made available for offline use. By using event handlers, the shell can already be pre-cached during installation without needing the attention of the user, which results in an app-like experience from the outset.

Another feature which makes a progressive web app feel like a native app is push notifications. While they are sometimes overused, when implemented appropriately, push notifications can be useful for subscriptions or actions which require the attention of the user [Grigsby 2018, pages 98–99; Svaiko 2020]. A service worker can subscribe to push notifications and when the server sends such a notification, the corresponding service worker is triggered by the `onpush` event handler [MDN 2025e].

2.3 Hypertext Transfer Protocol Secure (HTTPS)

For a progressive web app to work properly, it has to be served over HTTPS [Grigsby 2018, page 7]. This requirement is based on the fact that service workers can only run in a secure context. Using HTTPS is extremely important to protect against man-in-the-middle attacks and provide secure service worker functionality [BS Team 2025b]. The only exception is during local development, where sites served using `localhost` or `127.0.0.1` are treated as a secure environment [MDN 2025a].

Chapter 3

Current Status of PWAs

PWAs have evolved from an experimental concept in the late 2010s into a mature and widely deployed application model today (November 2025). Major technology companies are making use of PWAs as a strategic option alongside native and hybrid applications, particularly for organisations which benefit from cross-platform design, lower development costs, and improved performance on the web [Craig 2025].

3.1 Browser-Level Support

As of 2025, PWAs are supported across all major desktop and mobile browsers. The core enabling technologies (service workers, the web app manifest, and modern web APIs), are available in Chromium-based browsers and Safari on both macOS and iOS/iPadOS. Firefox currently does not support web app manifests, which means PWAs cannot be installed from a Firefox browser window. However, it supports all the other features of progressive web apps [MDN 2025a].

Historically, Safari was seen as lagging in PWA support [Craig 2025]. Service worker functionality, a prerequisite for offline capabilities and reliable caching, appeared in Chrome 40 (January 2015) and Firefox 44 (January 2016), but was only introduced in Safari 11.1 in March 2018, roughly three years after Chrome and just over two years after Firefox [MDN 2025f]. Even after this milestone, important PWA features, such as web push notifications remained unsupported on Apple platforms. Full web push support for home screen web apps on iOS and iPadOS did not arrive until iOS/iPadOS 16.4 in 2023, roughly seven years after it was available in Chromium-based browsers. However, by November 2025 these gaps have largely been closed. PWAs can now be installed and run with push notifications on all major platforms and Safari is no longer a major blocker for cross-platform PWA deployments, even though some advanced APIs continue to appear later or with more restrictions compared to other browsers [MDN 2025e].

3.2 Deepening Device and Operating System Integration

The PWA platform increasingly provides access to advanced device and operating system features. In addition to traditional Web APIs, modern browsers expose a growing set of capabilities [Craig 2025], such as:

- Biometric authentication (implemented through WebAuthn and platform-specific extensions).
- Bluetooth communication (via the Web Bluetooth API in supported browsers).
- Direct file system access (through the File System Access API or the Origin Private File System).
- Background synchronisation and periodic background task execution.

Concrete support for these features is largely dependent on the combination of browser and operating system. Therefore, application designers must still treat PWAs as a progressive enhancement strategy, which means that core functionality must remain fully operational on all platforms, whereas advanced features can be conditionally enabled where the relevant APIs are available.

In terms of market penetration, PWAs have clearly moved from niche innovation to mainstream adoption by 2025. Industry reports referenced by Craig [2025] estimate that the global PWA market is projected to exceed $15 billion by 2025, driven by increasing adoption across sectors such as retail, media, finance, healthcare and education.

3.3 Impact on User Engagement, Conversion and Data Usage

Case studies from large-scale deployments provide an insight into the impact of PWAs on user engagement and business performance. Craig [2025] summarises several representative examples:

- Retail and e-commerce: Companies such as Alibaba and Flipkart report significant performance improvements after introducing PWAs. For instance, Flipkart's PWA is associated with a reported 70% increase in conversion rates and a tripling of time spent on the site when compared to the previous mobile web implementation.

- Food and beverage: Starbucks introduced a PWA which supports menu browsing, order customisation and cart functionality to add items to the shopping cart even while being not connected to the internet. According to reported data, this PWA deployment resulted in a doubling of daily active users, highlighting the role of improved performance and offline access in repeated user engagement.

- Media and entertainment: Spotify and Pinterest use PWAs to deliver app-like experiences directly through the browser. For Spotify, the PWA is reported to load faster than the native app and additionally uses less mobile data. This making it particularly attractive in bandwidth-constrained environments.

Beyond these specific examples, PWA case studies consistently report several positive outcomes. These include reduced page load times and improved perceived performance, as well as increased time-on-site and greater session depth. Organisations also observe higher conversion rates, for example in add-to-cart actions, checkout completion or subscription uptake, along with notably lower data consumption and reduced storage requirements compared to native applications. These improvements are commonly attributed to a combination of technical and design factors, including aggressive caching via service workers, reduced network round-trips, offline fallbacks for essential user flows and installable, app-like user interfaces which reduce friction in comparison to traditional mobile web sites. Since end users have little tolerance for loading delays, these performance and reliability improvements translate directly into higher perceived quality and increased user satisfaction [Craig 2025].

Chapter 4

Installing a PWA

To illustrate the simple installation process on both desktop and mobile environments, the PWA of the coffee company Starbucks serves as an excellent example. This chapter describes Starbucks PWA installation on a desktop PC with Chrome under Windows 11 and on an iPhone with Safari under iOS 26.1. The goal is to show the user experience when interacting with a PWA for the first time.

4.1 Installation on Desktop PC (Windows 11, Chrome)

The four steps to install the Starbucks PWA on a desktop PC with Chrome under Windows 11 are illustrated in Figure 4.1. They are:

a) *Click Install Button*: On arrival at the Starbucks web site, the web browser automatically detects that a progressive web app is available for this web site and displays an Install button on the right side of the address bar (see Figure 4.1a).

b) *Confirm Installation*: Clicking the Install button opens an installation dialogue window, where the user is asked to confirm that the PWA should be installed (see Figure 4.1b).

c) *Desktop Icon Added*: Once confirmed, the browser installs the PWA and adds its icon to the PC's desktop (see Figure 4.1c).

d) *Launch from Desktop*: The PWA can now be launched like a native application by double-clicking its desktop icon. The PWA opens in a stand-alone browser window without any browser UI elements, providing the user with an app-like experience (see Figure 4.1d).

The installation process is straightforward. Once installed, launching a PWA is very similar to launching a native desktop application.

4.2 Installation on Smartphone (iOS 26.1, Safari)

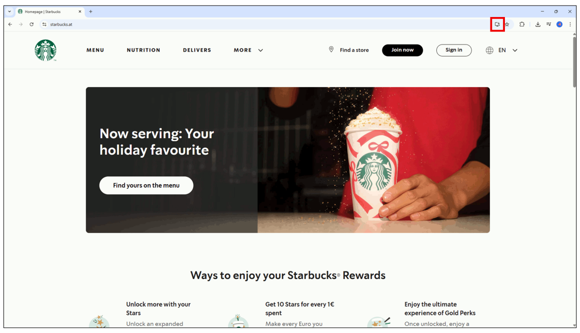
The process to installing a PWA on a smartphone is similar. The six steps to install the Starbucks PWA on an iPhone with Safari under iOS 26.1 are illustrated in Figure 4.2. They are:

a) *Tap ...*: After opening Starbucks web site, tapping on the three dots on the bottom right of the screen opens the browser menu (see Figure 4.2a).

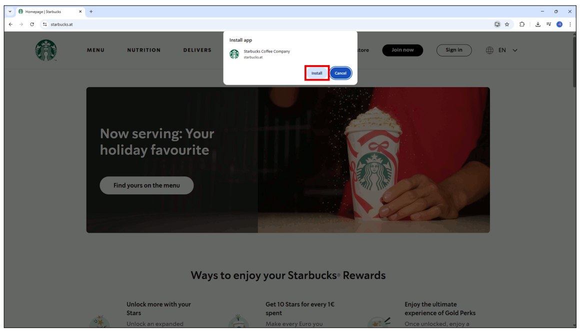
b) *Tap Share*: Tapping the Share option open the next menu (see Figure 4.2b).

c) *Tap Add to Home Screen*: Tapping Add to Home Screen opens the confirmation dialogue (see Figure 4.2c).

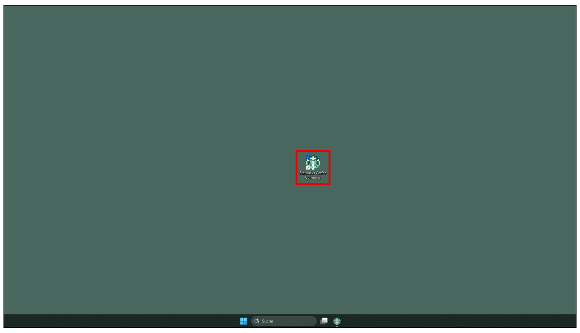
d) *Tap Add*: A confirmation screen appears, where the user is able to adjust the app name and icon. Installation is confirmed by tapping the Add button (see Figure 4.2d).


MENU
NUTRITION
DELIVERS
MORE
Find a store
Join now
Sign in
EN
Now serving: Your holiday favourite
Find yours on the menu
Ways to enjoy your Starbucks® Rewards
Unlock more with your Stars
Get 10 Stars for every 1€ spent
Enjoy the ultimate experience of Gold Perks

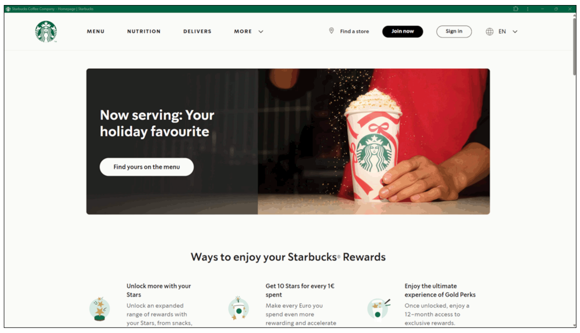
(a) Click Install button.


Install app
Starbucks Coffee Company
Install
Cancel
Now serving: Your holiday favourite
Find yours on the menu
Ways to enjoy your Starbucks® Rewards

(b) Confirm installation.



(c) Desktop icon added.


MENU
NUTRITION
DELIVERS
MORE
Find a store
Join now
Sign in
EN
Now serving: Your holiday favourite
Find yours on the menu
Ways to enjoy your Starbucks® Rewards
Unlock more with your Stars
Get 10 Stars for every 1€ spent
Enjoy the ultimate experience of Gold Perks

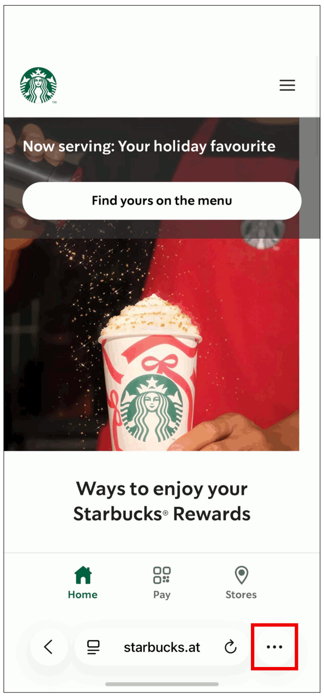
(d) Launch from desktop.

Figure 4.1: Installing the Starbucks PWA on a desktop PC (Windows 11, Chrome). [Screenshots taken by Sarah Hörtnagel.]

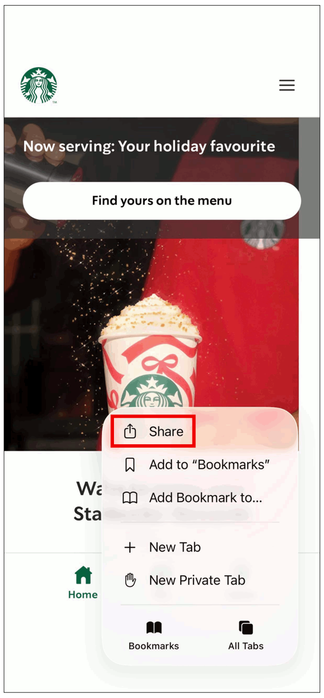
e) *Home Screen Icon Added*: The app icon of the PWA has been added to the home screen of the phone (see Figure 4.2e).

f) *Launch from Home Screen*: Tapping the PWA icon launches the app in full-screen mode without any browser controls, just like a native app (see Figure 4.2f).

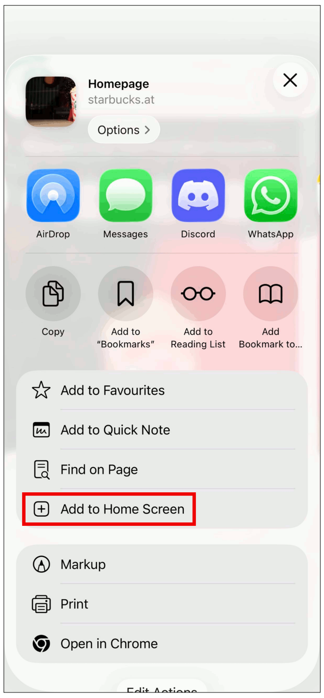
The installation process on a smartphone is similar to that on a desktop PC and is equally straightforward. Once installed, launching a PWA is very similar to launching a native smartphone app.


Now serving: Your holiday favourite
Find yours on the menu
Ways to enjoy your Starbucks® Rewards
Home
Pay
Stores
starbucks.at
...

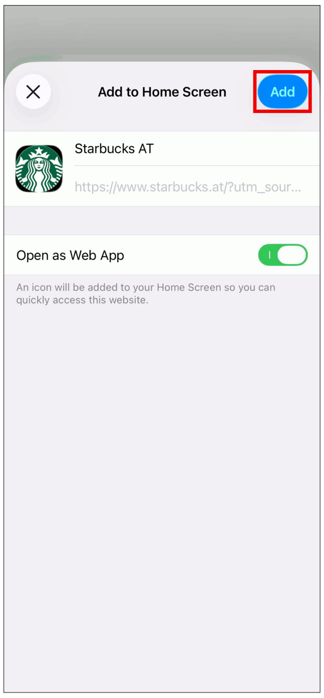
(a) Tap ....



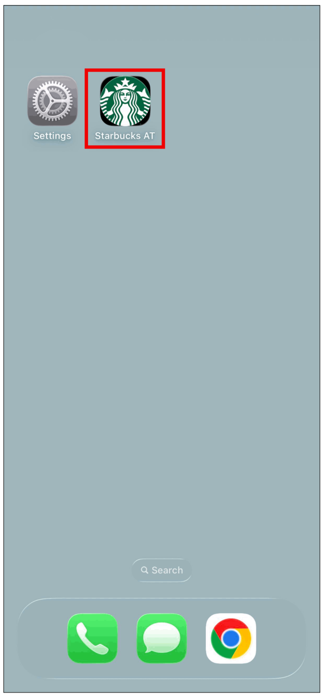
(b) Tap Share.



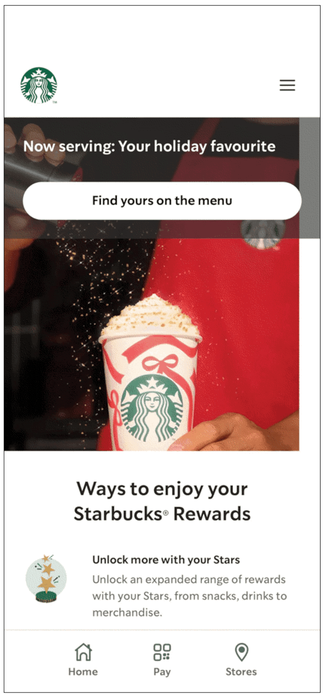
(c) Tap Add to Home Screen.


Add to Home Screen
Add
Starbucks AT
https://www.starbucks.at/?utm_sour...
Open as Web App
An icon will be added to your Home Screen so you can quickly access this website.

(d) Tap Add.



(e) Home screen icon added.


Now serving: Your holiday favourite
Find yours on the menu
Ways to enjoy your Starbucks® Rewards
Unlock more with your Stars
Unlock an expanded range of rewards with your Stars, from snacks, drinks to merchandise.
Home
Pay
Stores

(f) Launch from home screen.

Figure 4.2: Installing the Starbucks PWA on an iPhone (iOS 26.1, Safari). [Screenshots taken by Sarah Hörtnagel.]

Chapter 5

Platform and Browser Support

Since early 2025, progressive web apps have received full support from all major web browsers, with Apple finally providing comprehensive functionality [Craig 2025]. This development represents a significant milestone in the standardisation and widespread adoption of PWAs.

5.1 Desktop Platforms

All major browsers, namely Firefox, Google Chrome, Safari and Microsoft Edge, now fully support the core technologies needed for developing functional PWAs on their respective operating systems. Table 5.1 shows an overview of specific support for the key features of PWAs for the major browsers on the three main desktop platforms:

- *Windows*: Although Firefox provides functionality for service workers and push notifications, PWAs can not be installed using Firefox on Windows, since the browser does not fully support the web app manifest [MDN 2025a]. Instead, users are given the possibility to add a browser tab to the taskbar, which opens in a minimalist browser window with only limited app-like functionality. In contrast, Edge (pre-installed on Windows devices) and Chrome both support installing PWAs without any problems, enabling full integration and native-like behaviour.

- *macOS*: On macOS Sonoma (Safari 17) and later, users are able to install PWAs directly in Safari, the default browser pre-installed on Apple devices, just as they can using Chrome and Edge [MDN 2025a]. Again, Firefox does not support PWA installation on the desktop.

- *Linux*: On Linux, Chrome offers full functionality for PWAs without any issues. Firefox provides functionality for service workers and push notifications, but still does not support the installation of a PWA on the desktop.

5.2 Mobile Platforms

Mobile support for PWAs has improved significantly in 2025, with full support across all major browsers. Table 5.2 shows the support for PWAs on the two major mobile platforms:

- *iOS*: As Apple eased its restrictions against PWAs from iOS 16.4 onward, all major browsers on iOS are now able to install PWAs from the `Share` menu [MDN 2025a]. Nevertheless, it is important to mention that Safari still only partially implements several PWA-related features. The standard installation workflow used my many PWAs, including the `beforeinstallprompt` event, is not supported on iOS. In addition, Safari does not support all elements of the web app manifest: customisation entries such as `display`, `icons`, or `start_url` may be ignored or handled differently compared to Chromium-based browsers [MDN 2025g].

- *Android*: On Android, all major browsers provide full support for PWAs.

	Windows			macOS				Linux	
	Chrome	Edge	Firefox	Chrome	Edge	Firefox	Safari	Chrome	Firefox
Add to Home Screen (A2HS)	✓	✓	✗	✓	✓	✗	✓	✓	✗
Service Workers	✓	✓	✓	✓	✓	✓	✓	✓	✓
Push Notifications	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.1: PWA support across major desktop browsers on Windows, macOS and Linux. Data from MDN [2025e], MDN [2025f] and MDN [2025g].

	iOS			Android			
	Chrome	Safari	Firefox	Chrome	Firefox	Samsung Internet	UC Browser
Add to Home Screen (A2HS)	✓	✓	✓	✓	✓	✓	✓
Service Workers	✓	✓	✓	✓	✓	✓	✓
Push Notifications	✓	✓	✓	✓	✓	✓	✓

Table 5.2: PWA support across major mobile browsers on iOS and Android. Data from MDN [2025e], MDN [2025f] and MDN [2025g].

Chapter 6

Example Conference PWA

To demonstrate how a web site can be implemented as a PWA, a simple PWA for a small conference web site was implemented, furnished with both offline functionality and support for push notifications. The Conference PWA is implemented using Hugo, an open-source static site generator which converts content files into fast, pre-rendered HTML web sites [Hugo 2025]. The implementation was based on two previous projects, both of which used Hugo. The project IAWEB-Project [Golob et al. 2022] served as a template for the PWA. The content for the Conference PWA came from the project DCC Contentful Hugo [Rauter 2025]. The source code for the Conference PWA is available on GitHub [Aichner et al. 2025b]. A live demo of the PWA is also deployed [Aichner et al. 2025a]. Figure 6.1 shows the Conference PWA in Chrome on Windows, with the button to install the PWA to the desktop.

6.1 Web App Manifest

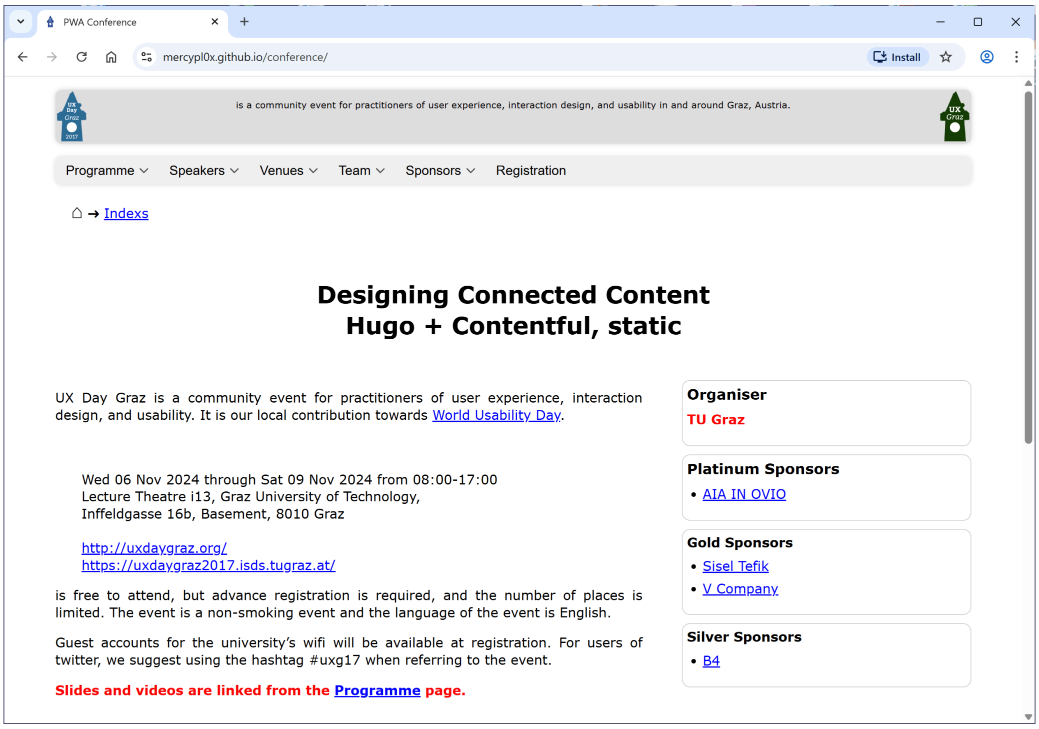
After setting up the static conference web site, the next step was to add a web app manifest for the project. The manifest file contains all necessary metadata required for browsers to recognize whether the web site is installable and thus trigger the display of the installation prompt. The implemented manifest can be seen in Listing 6.1. The required entries `name`, `icons`, `start_url` and `display` are defined appropriately. Additionally, the entries `background_color` and `theme_color` are defined for visual customisation across different platforms and devices.

6.2 Enabling Offline Availability

A service worker was implemented to cache all the conference content and provide access to it when offline. Listing 6.2 shows the code from the event listener, which is triggered when the PWA is installed and caches the initial content of the web site. Furthermore, when a new version of the service worker is available and activated, the previous cache is deleted and the updated service worker caches the latest version of the web site. This mechanism makes sure that the progressive web app always provides the latest content when a stable network connection is available.

The list of files to be cached is generated by a bash script, shown in Listing 6.3. The script is either executed after building the project locally or after the build step of an automated workflow. It systematically looks through the build folder, which is commonly set to `public/` when using Hugo, excludes any specified files and finally writes the paths of all remaining files into the array `CACHED` in file `sw.js`.

Once the PWA is deployed and hosted, the registered service worker intercepts all network requests initiated by the application and attempts to retrieve content from the network. If the attempt fails, for example due to limited network connectivity, the cached version is used as a fallback. Listing 6.4 shows the required event listener.



is a community event for practitioners of user experience, interaction design, and usability in and around Graz, Austria.

Programme Speakers Venues Team Sponsors Registration

⌂ → Indexs

Designing Connected Content
Hugo + Contentful, static

UX Day Graz is a community event for practitioners of user experience, interaction design, and usability. It is our local contribution towards World Usability Day.

Wed 06 Nov 2024 through Sat 09 Nov 2024 from 08:00-17:00
Lecture Theatre i13, Graz University of Technology,
Inffeldgasse 16b, Basement, 8010 Graz

http://uxdaygraz.org/
https://uxdaygraz2017.isds.tugraz.at/

is free to attend, but advance registration is required, and the number of places is limited. The event is a non-smoking event and the language of the event is English.

Guest accounts for the university's wifi will be available at registration. For users of twitter, we suggest using the hashtag #uxg17 when referring to the event.

Slides and videos are linked from the Programme page.

Organiser
TU Graz

Platinum Sponsors
- AIA IN OVIO

Gold Sponsors
- Sisel Tefik
- V Company

Silver Sponsors
- B4

Figure 6.1: The Conference PWA open in Chrome on Windows. Note the Install button in the top right corner.

```
{
  "name": "PWA Conference",
  "short_name": "PWA Conf",
  "icons": [
    {
      "src": "icon.svg",
      "sizes": "any",
      "type": "image/svg+xml",
      "purpose": "any"
    },
    {
      "src": "icon.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "maskable"
    }
  ],
  "start_url": ".",
  "id": "conference",
  "display": "standalone",
  "orientation": "portrait",
  "background_color": "#FFFFFF",
  "theme_color": "#FFFFFF"
}
```

Listing 6.1: The web app manifest defines the name, icons, display settings, theme configuration and additional metadata for the Conference PWA.

```javascript
const CACHE_VERSION = 'v1';
const CACHE = `cached-${CACHE_VERSION}`;

function precache()
{
  return caches.open(CACHE).then((cache) => cache.addAll(CACHED));
}

self.addEventListener("install", (event) => {
  const scope = self.registration.scope;
  CACHED = CACHED.map((f) => new URL(f, scope).toString());

  event.waitUntil(precache());
  self.skipWaiting();
});

self.addEventListener("activate", (event) => {
  event.waitUntil(
    (async () => {
      const keys = await caches.keys();

      await Promise.all(
        keys
        .filter((key) => key !== CACHE)
        .map((key) => caches.delete(key))
      );
      await self.clients.claim();
    })()
  );
});
```

Listing 6.2: Installation and activation of the event handler for the PWA's caching service worker. Resources are added to the cache when the PWA is installed. If the cached content has changed, the service worker updates the cache.

6.3 Implementing Push Notifications

Another feature which has not yet been implemented is the use of push notifications. This would be a great addition to the survey web site and a practical use case would be subscriptions to specific conference panels. If any changes occur to these panels, subscribers could be notified respectively by receiving a push notification on their mobile or desktop devices through the PWA. This way, also the user engagement could be improved, as users would receive timely updates on all important information about the conference.

Since the project is based on a static web site, some sort of back end service would be required to run the push notifications. In order to minimize the development effort, Grigsby [2018, pages 101–103] recommends using push service providers. These services manage the connection to all Push API browser endpoints and often also provide a UI to write and schedule messages. After setting up the project with a provider (for example OneSignal [2025]), all provided service worker files supplied by the provider must be added to the project. Finally, to initialize the provider on the web site, the Software Development Kit (SDK) snippet of the provider, which is a set of pre-built code, libraries and tools to simplify the integration of push notifications into the web site, must be added to the <head> section of the HTML. The <head> contains metadata which are loaded before the main content of the page, which makes it possible that the SDK is properly initialized.

```bash
#!/bin/bash
set -euo pipefail

excluded=(
  '*.mp4'
  'reports/*'
)

printf "Excluding %s from cache\n" "${excluded[@]}"

find_excludes=()
for pat in "${excluded[@]}"; do
  find_excludes+=( ! -path "public/$pat" )
done

files=$(
  find public -type f "${find_excludes[@]}" \
    | sed 's%^public/%%' \
    | sort \
    | sed 's%^%\"%; s%$%\", %'
)

echo "Found $(echo "$files" | wc -l) files to cache"

files="$(echo "$files" | tr -d '\n')"

sed -i "s%let CACHED = \[\]\;%let CACHED = \[$files\]\;%;" public/sw.js
```

Listing 6.3: Bash script executed after the build process to generate the list of files to cache for the PWA.

```
self.addEventListener('fetch', (event) => {
  event.respondWith(
    fromNetwork(event.request, MS_WAIT).catch(() => fromCache(event.request))
  );
});

function fromNetwork(request, timeout)
{
  return new Promise((fulfill, reject) => {
    const timeoutId = setTimeout(reject, timeout);

    fetch(request).then(
      (response) => {
        clearTimeout(timeoutId);
        fulfill(response);
      },
      (err) => {
        clearTimeout(timeoutId);
        reject(err);
      }
    );
  });
}

function fromCache(request)
{
  return caches.open(CACHE).then((cache) =>
    cache.match(request).then((matching) =>
      matching || Promise.reject('no-match')
    )
  );
}
```

Listing 6.4: The fetching event handler of the PWA's caching service worker, managing network requests with a network-first strategy and fallback to the cache after a timeout.

Chapter 7

Concluding Remarks

There has been a significant evolution in Progressive Web Apps (PWAs) over the past few years. PWAs have matured into a fully established application model, a cross-platform alternative to native applications and have become a fixed part of modern web development.

PWAs and their technical foundations are now consistently supported across all major platforms. A significant, long-standing gap from the past was closed by Apple in November 2025, when complete PWA functionality was provided on iOS and macOS. Only Firefox remains with a sole gap in support, namely for the Add to Home Screen (A2HS) feature on desktop devices.

From a technical perspective, the implementation of PWAs is no longer a complex or abstract process. The underlying concepts, including the web app manifest, service workers, offline caching strategies, and push notifications are easy to understand and well-documented. As a result, the technology of PWAs has reached a high level of stability and standardisation, making the threshold for implementing PWAs lower than ever. Converting existing web sites into installable, offline-capable applications is no longer a challenge.

For developers, PWAs offer several advantages over native mobile apps. They do not depend on app marketplaces such as App Store or Google Play Store. The development of PWAs reduces costs by creating a single application which works across all platforms. Based reports from companies like Starbucks, Flipkart, and Spotify, the use of PWAs brings measurable improvements in user engagement, conversion rates, and data efficiency. This further suggests that PWAs are not only a technological convenience, but can serve as a strategic instrument for companies to build user-centred applications. In short, PWAs can now be seen as a central component of web and application development, rather than as just an optional improvement.

Bibliography

Aichner, Lorenz, Sarah Hörtnagel, Hagen Leitner and Fabian Szakács [2025a]. *Conference PWA Demo*. 14 Dec 2025. `https://mercypl0x.github.io/conference/` (cited on page 15).

Aichner, Lorenz, Sarah Hörtnagel, Hagen Leitner and Fabian Szakács [2025b]. *Conference PWA Repository*. 06 Dec 2025. `https://github.com/mercypl0x/conference` (cited on page 15).

Andrews, Keith [2025]. *Information Architecture and Web Usability: Course Notes*. 21 Oct 2025. `https://courses.isds.tugraz.at/iaweb/iaweb.pdf` (cited on page 1).

BS Team [2025a]. *40+ Best PWA Examples (PWA Apps) By Industries in 2025*. Simicart, 09 Oct 2025. `https://simicart.com/blog/progressive-web-apps-examples/` (cited on page 3).

BS Team [2025b]. *What Is PWA? How does PWA work?* Simicart, 19 Mar 2025. `https://simicart.com/blog/what-is-progressive-web-app/` (cited on pages 2–3, 6).

Craig, Jerrie [2025]. *Progressive Web Apps (PWAs) in 2025: Are They Still the Future?* NashTech, 09 Sep 2025. `https://our-thinking.nashtechglobal.com/insights/progressive-web-apps-in-2025` (cited on pages 7–8, 13).

Golob, Ožbej, Robin Karlsson and kingkihu [2022]. *IAWEB-Project*. 10 Dec 2022. `https://github.com/ozbej/IAWEB-project` (cited on page 15).

Grigsby, Jason [2018]. *Progressive Web Apps*. A Book Apart, 12 Nov 2018. 157 pages. ISBN 1937557731 (cited on pages 5–6, 17).

Hugo [2025]. *Hugo: Introduction*. 20 Nov 2025. `https://gohugo.io/about/introduction/` (cited on page 15).

Keshri, Kailash [2025]. *Why Progressive Web Apps (PWAs) Are Making a Comeback in 2025*. Logic Square, 07 Oct 2025. `https://logic-square.com/why-progressive-web-apps-are-making-a-comeback-in-2025/` (cited on page 2).

Łabiński, Mateusz [2024]. *How does PWA impact SEO?* Vaimo, 14 Feb 2024. `https://vaimo.com/blog/how-does-pwa-impact-seo/` (cited on page 3).

LePage, Pete and Sam Richard [2020]. *What are Progressive Web Apps?* Google, 06 Jan 2020. `https://web.dev/articles/what-are-pwas` (cited on page 1).

MDN [2025a]. *Making PWAs installable*. Mozilla Developer Network, 30 Jun 2025. `https://developer.mozilla.org/docs/Web/Progressive_web_apps/Guides/Making_PWAs_installable` (cited on pages 5–7, 13).

MDN [2025b]. *Manifest Member: display*. Mozilla Developer Network, 08 Aug 2025. `https://developer.mozilla.org/docs/Web/Progressive_web_apps/Manifest/Reference/display` (cited on page 5).

MDN [2025c]. *Manifest Member: icons*. Mozilla Developer Network, 23 Jun 2025. `https://developer.mozilla.org/docs/Web/Progressive_web_apps/Manifest/Reference/icons` (cited on page 5).

MDN [2025d]. *Manifest Member: name*. Mozilla Developer Network, 14 Sep 2025. `https://developer.mozilla.org/docs/Web/Progressive_web_apps/Manifest/Reference/name` (cited on page 5).

MDN [2025e]. *Push API*. Mozilla Developer Network, 28 May 2025. `https://developer.mozilla.org/docs/Web/API/Push_API` (cited on pages 6–7, 14).

MDN [2025f]. *ServiceWorker*. Mozilla Developer Network, 23 Jun 2025. `https://developer.mozilla.org/docs/Web/API/ServiceWorker` (cited on pages 7, 14).

MDN [2025g]. *Web application manifest*. Mozilla Developer Network, 08 Aug 2025. `https://developer.mozilla.org/docs/Web/Progressive_web_apps/Manifest` (cited on pages 13–14).

OneSignal [2025]. *Web SDK setup*. OneSignal, 2025. `https://documentation.onesignal.com/docs/en/web-sdk-setup` (cited on page 17).

Rauter, Yannik [2025]. *DCC Contentful Hugo*. 24 Jan 2025. `https://github.com/yannikrauter/dcc-contentful-hugo` (cited on page 15).

Svaiko, Gert [2020]. *How To Optimize Progressive Web Apps: Going Beyond The Basics*. Smashing Magazine, 23 Dec 2020. `https://smashingmagazine.com/2020/12/progressive-web-apps/` (cited on pages 2, 5–6).