Engineer

Progressive Web App

Website, die sich wie eine App installieren und nutzen lässt

Was es ist

Eine Progressive Web App, auch als progressive web app geschrieben und meist zu PWA abgekürzt, ist eine Website, die ein Browser installieren und wie eine App ausführen kann: über ein eigenes Symbol, in einem eigenen Fenster und häufig mit Teilen, die auch ohne Netzwerk weiterarbeiten. Dabei bleibt sie die ganze Zeit eine gewöhnliche Website. Nichts kommt aus einem App-Store; die Seiten bestehen aus demselben HTML, CSS und JavaScript, das ein Browser in einem Tab laden würde.

Kein Standard definiert den Begriff. Die Standards definieren die beiden Bestandteile, aus denen sie aufgebaut wird. Der erste ist das Web-App-Manifest, eine JSON-Datei nach der W3C-Spezifikation Web Application Manifest, die der App einen Namen, Symbole, eine Start-URL und einen Anzeigemodus gibt. Diese Spezifikation setzt die Hürde niedrig an: Jede Website sei eine installierbare Webanwendung. Ein Browser kann eine Installationsmöglichkeit anbieten; die installierte App öffnet sich dann mit den Einstellungen des Manifests. Im Anzeigemodus `standalone` soll sie einer nativen Anwendung ähneln. Das kann ein eigenes Fenster und ein Symbol im Anwendungsstarter bedeuten; der Browser lässt seine gewöhnliche Oberfläche wie die URL-Leiste weg.

Der zweite Bestandteil ist der Service Worker aus der W3C-Spezifikation Service Workers. Es ist ein Skript, das der Browser getrennt von der Seite ausführt. Es sitzt zwischen Seite und Netzwerk, sieht die Anfragen der Seite und kann sie aus selbst verwalteten Caches beantworten. Dadurch kann eine Web-App ihre Seiten auch offline weiter ausliefern. Diese Caches gehören dem Skript: Die Spezifikation trennt sie vollständig vom gewöhnlichen HTTP-Cache des Browsers. Sie ändern sich nur, wenn das Skript sie ändert.

Web Application Manifest verwendet das Wort „progressive“ nie; die Service-Workers-Spezifikation verwendet es einmal, in einer Anmerkung dazu, wie Chromium eine PWA erkennt. Die PWA-Anleitung von Next.js empfiehlt Progressive Enhancement, damit die App auch in einem Browser gut funktioniert, der manche PWA-Funktionen nicht unterstützt. In diesem Sinne ist jeder Bestandteil optional. Ein Browser, der das Manifest ignoriert oder keine Service Worker unterstützt, zeigt weiterhin die Website.

Eine Progressive Web App ist auch keine eigene Art von Website. Eine Single-Page-Anwendung, eine statische Website und eine serverseitig gerenderte Website können jeweils ein Manifest und einen Service Worker ergänzen. Der Begriff beschreibt, was ein Browser nach dem Laden mit einer Website tun kann, nicht, wie ihr HTML entsteht.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Progressive Web App?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Oberflächen, die schnell laden, auf dem Smartphone funktionieren und sich in allen Browsern Ihrer Kunden gleich verhalten.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Full-Stack-GIS-Kartenentwicklung geleitet — Full-Stack-Kartenentwicklung von der Datenbank bis zum Browser

Full-Stack-GIS-Kartenentwicklung geleitet Kontakt aufnehmen →

Warum es wichtig ist

Die Anleitung von Next.js begründet den Einsatz mit drei Punkten. Aktualisierungen werden sofort veröffentlicht, ohne auf die Freigabe eines App-Stores zu warten. Eine Codebasis bedient jede Plattform. Und die App erhält Funktionen, die Menschen von nativen Apps erwarten, etwa Installation auf dem Startbildschirm und Push-Benachrichtigungen. Alle drei folgen daraus, dass die App eine Website bleibt: Eine neue Version kann ohne separate Veröffentlichung in einem App-Store bereitgestellt werden; wann das einzelne Gerät sie übernimmt, hängt vom Caching und von Service-Worker-Updates ab.

Offline-Nutzung ist der Grund für den Service Worker. Der Abschnitt zur Motivation in der Spezifikation sagt, Webanwendungen hätten traditionell ein erreichbares Netzwerk vorausgesetzt; andere Anwendungsplattformen gingen deshalb besser mit unterbrochenen Verbindungen um. Service Worker geben auch Webanwendungen einen Weg, solche Unterbrechungen zu bewältigen. Mit

einem solchen Worker muss die Nutzung auch im Zug in einem Tunnel oder auf einem Schiff außer Reichweite der Küste nicht mehr auf der Fehlerseite des Browsers enden.

Die beiden Hälften sind unabhängig. Die Anleitung von Next.js weist darauf hin, dass Installationsaufforderungen keine Offline-Unterstützung brauchen; die Manifest-Spezifikation hat ihr eigenes Feld `serviceworker` entfernt. Eine App kann ohne Service Worker installierbar sein. Umgekehrt kann ein Service Worker eine gewöhnliche Website ganz ohne Manifest offline nutzbar machen.

Beiden Hälften gemeinsam ist HTTPS. Next.js nennt eine über HTTPS ausgelieferte Website als eine von zwei Voraussetzungen für die Installation einer App auf dem mobilen Startbildschirm; die andere ist ein gültiges Manifest. Laut Service-Workers-Spezifikation müssen Service Worker in sicheren Kontexten laufen. In der Praxis bedeutet das HTTPS, mit einer Ausnahme für localhost zur Entwicklung.

Wie es in der Praxis aussieht

Drei kleine, eigens für diesen Artikel geschriebene Dateien ergeben eine funktionierende PWA. Das Manifest benennt die App, wählt den Anzeigemodus und führt ihre Symbole auf:

```
{
  "name": "Field Notes",
  "short_name": "Notes",
  "start_url": "/",
  "display": "standalone",
  "icons": [
    { "src": "/icon-192.png", "sizes": "192x192", "type": "image/png" },
    { "src": "/icon-512.png", "sizes": "512x512", "type": "image/png" }
  ]
}
```

Jede Seite verlinkt das Manifest und registriert den Service Worker, allerdings nur in einem Browser, der ihn unterstützt:

```
<link rel="manifest" href="/manifest.webmanifest" />
<script>
  if ("serviceWorker" in navigator) {
```

```
    navigator.serviceWorker.register("/sw.js");
  }
</script>
```

Der Service Worker hält eine kleine Menge von Dateien für die Offline-Nutzung vor:

```
// sw.js
const CACHE = "shell-v2";
const SHELL = ["/offline.html", "/app.css", "/app.js"];

async function precache() {
  const cache = await caches.open(CACHE);
  await cache.addAll(SHELL);
}

async function dropOldCaches() {
  const names = await caches.keys();
  const old = names.filter((name) => name !== CACHE);
  await Promise.all(old.map((name) => caches.delete(name)));
}

async function respond(request) {
  if (request.mode === "navigate") {
    try {
      return await fetch(request);
    } catch {
      return caches.match("/offline.html");
    }
  }
  const hit = await caches.match(request);
  return hit ?? fetch(request);
}

self.addEventListener("install", (event) => {
  event.waitUntil(precache());
});

self.addEventListener("activate", (event) => {
  event.waitUntil(dropOldCaches());
});

self.addEventListener("fetch", (event) => {
  if (event.request.method !== "GET") return;
  event.respondWith(respond(event.request));
});
```

Das Manifest verwendet `.webmanifest`, die dafür registrierte Dateiendung, und wird mit dem Medientyp `application/manifest+json` ausgeliefert. Der Service Worker liegt im Wurzelverzeichnis, weil sein Geltungsbereich standardmäßig dem Verzeichnis des Skripts entspricht. Ein Worker unter `/js/sw.js` könnte deshalb nur Seiten unter `/js/` steuern, sofern der Server keinen Header `Service-Worker-Allowed` sendet. Bei der Installation öffnet er einen Cache namens `shell-v2` und speichert Offline-Seite, Stylesheet und Skript. `addAll()` schlägt fehl, wenn eine dieser Dateien einen Fehlerstatus zurückliefert. Weil `waitUntil()` dieses Promise erhalten hat, schlägt damit die gesamte Installation fehl, statt einen halb gefüllten Cache zu hinterlassen. Bei der Aktivierung löscht er jeden anders benannten Cache. Seitennavigationen gehen zuerst ans Netzwerk und fallen auf die gespeicherte Offline-Seite zurück, wenn es nicht antwortet. Alles andere wird, falls vorhanden, aus dem Cache bedient und andernfalls abgerufen. Anfragen mit einer anderen Methode als `GET` bleiben unberührt.

Das Unternehmen beschreibt eine vollständige Umsetzung auf einer realen Plattform. Der Portfolioeintrag erklärt den Bedarf: Viele Nutzer der Plattform sind Seeleute und Außendienstmitarbeiter mit Mobiltelefonen, häufig offline oder mit schwacher Verbindung. Die App musste sich deshalb wie eine native App installieren lassen und bei Signalverlust weiterarbeiten. Der Eintrag behandelt beide Hälften. Die App lässt sich auf dem Startbildschirm installieren und bringt Symbole in mehreren Größen, einen Splashscreen und Designfarben mit. Der Service Worker wird über das Plugin next-pwa eingerichtet. Das Laufzeit-Caching von Workbox liefert Schriftarten, Bilder, Audio, Video und CDN-Dateien nach dem Cache-first-Prinzip aus, weil diese sich nicht von Anfrage zu Anfrage ändern; hinzu kommen HTTP-Cache-Control-Header. Weil Service Worker HTTPS benötigen, wurden Installation und Offline-Verhalten über HTTPS auf echten Geräten getestet. Als Ergebnis nennt der Bericht eine App, die Menschen installieren und offline weiterverwenden; bei erneuten Besuchen lädt sie aus dem Cache.

Auch die Installation sieht aus Lesersicht unterschiedlich aus. Die Anleitung zur Installation dieser Website als App zeigt, wo die einzelnen Browser den Befehl anbieten: im Teilen-Menü unter „Zum Home-Bildschirm“ in Safari auf dem iPhone, über ein kleines Symbol rechts in der Adressleiste von Chrome, Edge, Brave, Opera und Vivaldi auf einem Computer sowie über „Zum Dock hinzufügen“ in Safari ab macOS Sonoma. Sie hält auch fest, dass iOS nie von sich aus eine Installation vorschlägt. Firefox bietet unter Windows ab Version 143 eigene Web-App-Fenster, bei Installationen aus dem Microsoft Store ab Version

150, nicht jedoch unter macOS oder Linux. Die Seite sagt ebenso klar, was die Installation dieser Website liefert: ein Symbol und ein Fenster, aber keine Offline-Kopie, denn Offline-Unterstützung braucht einen Service Worker.

Wo es schiefgeht

Nutzer behalten die alte Version. Laut Service-Workers-Spezifikation werden Cache-Objekte nur auf ausdrückliche Anweisung des Autors aktualisiert und verfallen nur, wenn er sie löscht. Ein Cache-first-Worker liefert deshalb das gestrige Stylesheet, bis sein Skript etwas anderes vorgibt. Nehmen Sie eine Version in den Cache-Namen auf, ändern Sie sie bei jeder Änderung gespeicherter Dateien und löschen Sie die alten Caches bei der Aktivierung, wie im Beispiel. Eine Skriptänderung startet die Aktualisierung, weil der Browser das neu abgerufene `sw.js` Byte für Byte mit der installierten Fassung vergleicht. Selbst dann wartet der neue Worker, bis kein Tab und kein Fenster den alten verwendet, sofern er nicht `skipWaiting()` aufruft. Wer einen Tab offen lässt, verwendet daher die vorherige Version so lange, wie dieser offen bleibt. Die Anleitung von Next.js liefert `sw.js` mit `Cache-Control: no-cache, no-store, must-revalidate` aus, damit Nutzer immer die neueste Fassung des Workers selbst erhalten.

Es funktioniert auf dem Laptop, aber nicht auf dem Testtelefon. `navigator.serviceWorker` existiert nur in sicheren Kontexten. Auf einer über einfaches HTTP ausgelieferten Seite ergibt die übliche Funktionsprüfung stillschweigend false. Der Worker wird nie registriert, und die Seite meldet nichts davon. Die Spezifikation erlaubt Browsern, localhost, 127.0.0.0/8 und ::1/128 für die Entwicklung zuzulassen. Deshalb kann ein auf dem eigenen Entwicklungsrechner funktionierender Worker fehlen, wenn ein Telefon dieselbe App per HTTP über die lokale Netzwerkadresse dieses Rechners öffnet. Next.js bietet für lokale HTTPS-Tests `next dev --experimental-https`. Testen Sie Installation und Offline-Verhalten auf echten Geräten über HTTPS, bevor Sie eines von beiden als erledigt betrachten.

Ein Fetch-Listener tut nichts. Die Service-Workers-Spezifikation hält fest, dass manche Websites einen `fetch`-Listener mit leerem Rumpf registrieren, nur damit Chromium sie als Progressive Web App erkennt. Die Spezifikation erkennt inzwischen Listener wie `() => {}` und empfiehlt Browsern die Warnung, dass solche Listener unnötig sind und die Leistung beeinträchtigen können. Schreiben Sie einen Handler, der Anfragen beantwortet, oder lassen Sie den Listener weg.

Sich auf eine Installationsaufforderung verlassen. Ob und wie eine Installation angeboten wird, entscheidet der Browser. Next.js sagt, moderne Browser zeigten automatisch eine Installationsaufforderung, sobald eine Website ein gültiges Manifest und HTTPS hat. Die Anleitung rät davon ab, einen eigenen Installationsknopf auf `beforeinstallprompt` aufzubauen: Dieses Ereignis funktioniert nicht über alle Browser und Plattformen hinweg, Safari unter iOS eingeschlossen. Die Manifest-Spezifikation hat das Ereignis inzwischen ganz entfernt. Auch Anzeigemodus und Start-URL sind nur Empfehlungen. Ein Browser ohne Unterstützung für `standalone` fällt erst auf `minimal-ui` und dann auf `browser` zurück, also auf einen gewöhnlichen Tab. Erklären Sie Nutzern die Installation, wie es die iOS-Anweisungen der Next.js-Anleitung tun, und entwerfen Sie die App so, dass sie auch in einem Tab geöffnet werden kann.

Weiterführende Lektüre

- Web Application Manifest — Manifestformat, Felder und Anzeigemodi sowie die Bedeutung von Installation und Deinstallation.
- Service Workers — Lebenszyklus, Geltungsbereich und Aktualisierungsregeln des Workers, die Anforderung sicherer Kontexte und die Cache API.
- Web Application Manifest: Installierbare Webanwendungen — wie ein Browser eine Webanwendung installiert, startet und wieder entfernen lässt.
- Service Workers: Lebensdauer von Caches verstehen — warum Cache-Objekte sich nie aktualisieren oder verfallen, sofern das Skript es nicht anordnet.
- Eine Progressive Web Application (PWA) mit Next.js erstellen — Manifestdatei, Push-Benachrichtigungen, Installationsanweisungen für iOS, HTTPS in der Entwicklung und Sicherheitsheader für den Service Worker.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist eine Progressive Web App?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/progressive-web-app/