Engineer

プログレッシブウェブアプリ

インストールしてアプリのように使えるウェブサイト

どのようなものか

プログレッシブウェブアプリは、英語でProgressive Web Appとも書き、通常はPWAと略します。ブラウザーからインストールし、専用アイコンから起動し、専用ウインドウでアプリのように動かせるウェブサイトです。通信がなくなっても、一部を使い続けられることが多くあります。ずっと普通のウェブサイトであることは変わりません。アプリストアから何かを取得するわけではなく、ページは、ブラウザーがタブ内で読み込むものと同じHTML、CSS、JavaScriptです。

この用語を定義する標準はありません。標準が定義しているのは、構成する二つの部分です。一つ目は、W3CのWeb Application Manifestが記述するJSONファイル、ウェブアプリマニフェストです。アプリの名前、アイコン、開始URL、表示モードを与えます。仕様の条件は広く、「どのウェブサイトも、インストール可能なウェブアプリケーションである」とします。ブラウザーはインストール手段を提供でき、インストールしたアプリは、その後マニフェストの設定を適用して開きます。`standalone` 表示モードでは、ネイティブアプリに似せることを目指します。専用ウインドウや、アプリケーションランチャーのアイコンなどを備える場合があり、URLバーのようなブラウザーの通常の操作部品は表示しません。

二つ目は、W3CのService Workers仕様にあるサービスワーカーです。ブラウザーがページとは別に動かすスクリプトで、ページとネットワークの間に入り、ページのリクエストを見て、自身が管理するキャッシュから応答できます。これにより、通信がなくても、ウェブアプリのページを返し続けられます。このキャッシュの管理者はスクリプトです。仕様では、ブラウザーの通常のHTTPキャッシュと完全に分け、スクリプトが変更したときだけ変わるものとしています。

Web Application Manifestには「progressive」という語は一度もなく、Service Workers仕様には、ChromiumがPWAを認識する方法についての注記に一度だけ出ます。Next.jsのPWAガイドは、PWA機能の一部に対応しないブラウザーでも

正常に使えるよう、プログレッシブエンハンスメントを勧めています。その意味で、どの部分も必須ではありません。マニフェストを無視するブラウザーでも、サービスワーカーを使えないブラウザーでも、ウェブサイト自体は表示します。

プログレッシブウェブアプリは、別種のウェブサイトでもありません。単一ページアプリケーション、静的サイト、サーバーで描画するサイトのどれも、マニフェストとサービスワーカーを追加できます。この言葉が表すのは、HTMLの作り方ではなく、読み込み後にブラウザーがそのサイトをどのように扱えるかです。

概念を判断に生かす

「プログレッシブウェブアプリとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 読み込みが速く、スマートフォンでも使え、お客様がどのブラウザーを使っていても同じように動く画面を提供します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 フルスタックのGIS地図開発を指揮 — DBからブラウザーまでのフルスタック地図開発

フルスタックのGIS地図開発を指揮 お問い合わせ →

なぜ重要なのか

Next.jsのガイドは、利点を三つ挙げます。アプリストアの審査を待たず、更新をすぐ配信できること。一つのコードベースがすべてのプラットフォームで使えること。そして、ホーム画面へのインストールやプッシュ通知など、ネイティブアプリに期待される機能を持てることです。いずれも、アプリがウェブサイトのままであることから来ています。新しい版は、アプリストアで別途リリースせずに公開できます。ただし、各端末が新しい版に切り替わる時期は、キャッシュとサービスワーカーの更新に左右されます。

サービスワーカーがある理由は、オフラインへの対応です。仕様の動機を述べる節は、従来のウェブアプリが、ネットワークに到達できることを前提としてきたため、接続の途切れへの対応では、ほかのアプリ基盤の方が優れていると説明します。サービスワーカーは、ウェブアプリにも中断を扱う方法を与えます。導入すれば、トンネル内の列車や、沿岸の電波が届かない船で使う際に、ブラウザーのエラーページで終わる必要がなくなります。

二つの部分は独立しています。Next.jsのガイドは、インストールの案内を出すために、オフライン対応は必要ないと述べます。マニフェスト仕様からも、`serviceworker` メンバーは削除されました。サービスワーカーなしでもインストール可能にでき、マニフェストがまったくない普通のサイトでも、サービスワーカーでオフライン対応にできます。

共通の条件はHTTPSです。Next.jsは、スマートフォンのホーム画面へインストールするための二条件として、HTTPSで配信するサイトと、有効なマニフェストを挙げます。Service Workers仕様は、サービスワーカーが安全なコンテキストで動くことを求めます。実際にはHTTPSを使い、開発用のlocalhostは例外として認められます。

実際の活用例

この記事用に書いた三つの小さなファイルで、動くPWAを作れます。マニフェストは、アプリの名前、表示モード、アイコンを指定します。

```
{
  "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" }
  ]
}
```

各ページは、マニフェストを参照し、対応するブラウザーでだけサービスワーカーを登録します。

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

サービスワーカーは、オフライン用に少数のファイルを保存します。

```
// 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));
});
```

マニフェストの拡張子は、登録されている `.webmanifest` で、メディア型 `application/manifest+json` として配信します。サービスワーカーの既定の範囲は、スクリプトを置いたディレクトリーなので、ルートに置きます。 `/js/sw.js` なら、管理できるのは `/js/` 配下だけです。ただし、サーバーが `Service-Worker-Allowed` ヘッダーを送れば、その範囲を広げられます。インストール時に `shell-v2` というキャッシュを開き、オフライン用ページ、スタイルシート、スクリプトを保存します。どれかのファイルがエラーの状態コードを返すと、 `addAll()` が失敗します。そのPromiseを `waitUntil()` へ渡しているため、中途半端なキャッシュを残すのではなく、インストール全体が失敗します。有効化時には、名前が異なるキャッシュをすべて消します。ページへの移動は、まずネットワークへ問い合わせ、応答がなければ、キャッシュのオフライン用ページへ戻ります。それ以外は、キャッシュにあればそこから返し、なければ取得します。 `GET` 以外のリクエストには介入しません。

実在するプラットフォームにこの仕組みを一式実装した例も、この会社が紹介しています。ポートフォリオの記録は、必要だった理由を説明します。利用者の多くは、スマートフォンを使う船員や現場担当者で、電波がないか弱いことが多いため、ネイティブアプリのようにインストールし、通信が途切れても使える必要がありました。両方を実装しています。複数サイズのアイコン、起動画面、テーマ色を備え、ホーム画面へインストールできます。サービスワーカーはnext-pwaプラグインで設定し、Workboxの実行時キャッシュで、フォント、画像、音声、動画、CDN素材をキャッシュ優先で返し、HTTPのキャッシュ制御ヘッダーも使います。リクエストごとには、これらの素材が変わらないためです。サービスワーカーにはHTTPSが必要なので、インストールとオフライン動作は、実機でHTTPSを通して試しました。記録された成果は、インストール後にオフラインでも使い続け、再訪時にはキャッシュから読み込むアプリです。

インストールの部分は、使う側から見ても違いがあります。このサイトのアプリとしてインストールする案内は、各ブラウザーでのコマンドの場所を説明します。iPhoneのSafariなら共有メニューと「ホーム画面に追加」、パソコンのChrome、Edge、Brave、Opera、Vivaldiならアドレスバー右側の小さなアイコン、macOS Sonoma以降のSafariなら「Dockに追加」です。iOS自身は、インストールを提案しないことも記します。Firefoxは、Windowsでは143以降、Microsoft StoreからFirefoxをインストールした場合は150以降で、ウェブアプリ専用ウインドウを使えますが、macOSとLinuxでは使えません。また、このサイトをインストールして得られるのは、アイコンとウインドウであり、オフライン用コピーではない、と明示します。オフライン対応には、サービスワーカーが必要だからです。

問題が起きるところ

利用者が古い版を使い続けます。 Service Workers仕様では、作者が明示的に求めない限り、キャッシュオブジェクトは更新されず、作者が消さない限り期限も切れません。そのため、キャッシュ優先のワーカーは、スクリプトが指示を変えるまで、昨日のスタイルシートを返し続けます。例のように、キャッシュ名にバージョンを入れ、保存するファイルが変われば名前を変え、有効化時に古いキャッシュを消します。ブラウザーは、新しく取得した `sw.js` を、インストール済みのものとバイト単位で比べるため、更新のきっかけはスクリプトの変更です。それでも、新しいワーカーは、 `skipWaiting()` を呼ばなければ、古いものを使うタブやウインドウがなくなるまで待ちます。タブを開き続ける利用者は、その間ずっと古い版を使います。Next.jsのガイドは、ワーカー自身の最新版を必ず取得できるよう、 `sw.js` を `Cache-Control: no-cache, no-store, must-revalidate` で配信します。

ノートパソコンでは動き、テスト用スマートフォンでは動きません。 `navigator.serviceWorker` が存在するのは、安全なコンテキストだけです。暗号化しないHTTPで開いたページでは、通常の機能確認が何も告げずに偽となり、ワーカーは登録されず、その事実がページに表示されることもありません。仕様では、ブラウザーがlocalhost、127.0.0.0/8、::1/128を開発用として許可してよいとしています。そのため、開発者自身のマシンで動くワーカーも、スマートフォンがローカルネットワーク上のそのマシンのアドレスへHTTPで接続して、同じアプリを開くと、利用できないことがあります。Next.jsには、ローカルでHTTPSを試す `next dev --experimental-https` があります。インストールもオフライン動作も、完了とする前に、実機とHTTPSで試します。

fetchリスナーが何もしません。 Service Workers仕様は、ChromiumにPWAとして認めさせるためだけに、空の `fetch` リスナーを登録するサイトがあると記します。仕様は、今では `() => {}` のようなリスナーを検出し、不要なだけでなく性能を損なう可能性もある、とブラウザーが警告することを勧めています。リクエストに応答するハンドラーを書くか、リスナー自体を付けないようにします。

インストール案内が出ることを前提にします。 案内するかどうか、どう案内するかは、ブラウザーが決めます。Next.jsは、有効なマニフェストとHTTPSがあれば、現在のブラウザーは自動で案内すると説明する一方、 `beforeinstallprompt` を使った独自ボタンは勧めません。このイベントは、すべてのブラウザーやプラットフォームで使えるわけではなく、iOSのSafariにも対応しないためです。その後、マニフェスト仕様は、イベント自体を削除しました。表示モードと開始URLも、あくまで指定側の希望です。 `standalone` に非対応なら、 `minimal-ui` 、その次は普通のタブである `browser` へ戻ります。Next.jsのiOS向け案内のように、インストール方法を伝えつつ、タブで開いて使う場合にも対応して設計します。

参考資料

- Web Application Manifest — マニフェストの形式、メンバー、表示モード、インストールとアンインストールの意味です。
- Service Workers — ワーカーのライフサイクル、範囲、更新規則、安全なコンテキストの条件、Cache APIを扱います。
- Web Application Manifest：インストール可能なウェブアプリ — ブラウザーがアプリをインストールし、起動し、利用者が削除できるようにする仕組みです。
- Service Workers：キャッシュの寿命 — スクリプトが指示しない限り、キャッシュオブジェクトが更新も失効もしない理由です。
- Next.jsでプログレッシブウェブアプリ（PWA）を作る方法 — マニフェストファイル、プッシュ通知、iOSのインストール案内、開発中のHTTPS、サービスワーカーのセキュリティーヘッダーです。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「プログレッシブウェブアプリとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/progressive-web-app/