Dokumentenverwaltung

DMS-System KI-Tool

Progressive Web Apps

Webseite als App auf iPhone und Android — Was es ist, wie es funktioniert, was es kann und nicht kann.

Erstellt: 9. Mai 2026 | **Anlass:** DMS-Mobile-Projekt | **Zielgruppe:** Du selbst, fürs Archiv

1. Worum geht es?

Wir haben heute aus deiner DMS-Webseite eine "echte" iPhone-App gemacht — mit eigenem Icon auf dem Homescreen, Vollbildmodus ohne Browser-Leiste und Push-Benachrichtigungen. Diese Anleitung erklärt das Konzept dahinter, den Fachbegriff und zeigt dir, wie du das Verfahren auf jede andere Webseite übertragen kannst, die du selbst kontrollierst.

Der Fachbegriff: Was wir gebaut haben, heißt im Web-Jargon **Progressive Web App (PWA)**. "Progressive" deshalb, weil die Seite weiterhin als normale Webseite funktioniert — aber wenn sie auf einem modernen Gerät läuft, kann sie zusätzlich App-ähnliche Features anbieten. Auf iOS heißt der eigentliche Installations-Vorgang **"Add to Home Screen"** (kurz A2HS), zu Deutsch: "Zum Home-Bildschirm".

> Eine PWA ist also **eine Webseite mit App-Eigenschaften**, keine native App im klassischen Sinn. Sie wird nicht aus dem App Store geladen, sondern direkt von einer URL aus auf dem Gerät installiert.

2. Was eine installierte PWA kann

Sobald die Webseite als PWA auf dem Homescreen liegt, fühlt sie sich wie eine native App an:

- **Eigenes Icon** auf dem Homescreen, ohne Browser-Logo dahinter
- **Vollbild-Modus** beim Tap aufs Icon — keine Adressleiste, keine Lesezeichen, keine Tabs
- **Eigener App-Switcher-Eintrag** beim Multitasking, mit eigenem Icon in der Karten-Vorschau
- **Push-Benachrichtigungen** — auch wenn die App geschlossen ist, kann der Server Nachrichten ans Gerät schicken (auf iOS nur in der PWA, im Safari geht es nicht)
- **Eigener Storage** — LocalStorage, Cookies, Service-Worker-Cache sind getrennt von der Safari-Version derselben Seite
- **Funktioniert offline**, falls die Seite das implementiert (über den Service Worker)

Wichtiger Trick auf iOS: Dieselbe URL ist für das Betriebssystem **zwei verschiedene Welten**:

- In Safari aufgerufen → eine normale Website (kein Push, eingeschränkter Storage, andere Cookies)
- Aus dem Homescreen-Icon getippt → eine eigenständige PWA (mit Push, eigenem Storage, eigenen Cookies)

Das wirkt zunächst irritierend, ist aber Apples bewusste Entscheidung: Push-Benachrichtigungen gibt es nur als "Belohnung" dafür, dass der Nutzer die PWA bewusst installiert hat. So vermeidet Apple, dass jede Website ungefragt Push-Anfragen schicken kann.

3. Was die Webseite mitbringen muss

Damit eine URL als PWA installierbar wird, braucht die Webseite drei Bausteine auf dem Server — zusätzlich zum normalen HTML/CSS/JS:

3.1 Web App Manifest

Eine Datei namens **manifest.webmanifest** (im JSON-Format), die die App beschreibt: Name, Icon-Größen, Start-URL, Theme-Color, Display-Modus. Beispiel:

```
{
"name": "DMS Mobile",
"short_name": "DMS Mobile",
"start_url": "/posteingang/DMS-Client-Start.html",
"scope": "/posteingang/",
"display": "standalone",
"theme_color": "#22c55e",
"background_color": "#0f172a",
"icons": [
{ "src": "icons/icon-192.png", "sizes": "192x192", "type": "image/png" },
{ "src": "icons/icon-512.png", "sizes": "512x512", "type": "image/png" }
]
}
```

Wichtige Felder erklärt:

- **name**: voller App-Name, erscheint im App-Switcher
- **short_name**: Kurzform fürs Homescreen-Icon (max. ~12 Zeichen)
- **start_url**: welche Page öffnet sich beim Tap aufs Icon
- **scope**: welcher Bereich der Webseite zur App gehört (URLs außerhalb öffnet das System im normalen Browser)
- **display: standalone**: Vollbild-Modus — ohne diesen Wert verhält sich die installierte Seite wie ein Browser-Lesezeichen
- **theme_color**: Farbe der Statusbar, wenn die App läuft

3.2 HTML-Meta-Tags

In jeder HTML-Page, die als App-Einstieg dienen kann, gehören diese Tags in den **<head>**:

```
<link rel="manifest" href="manifest.webmanifest">
<meta name="theme-color" content="#22c55e">
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="default">
<meta name="apple-mobile-web-app-title" content="DMS Mobile">
<link rel="apple-touch-icon" href="icons/apple-touch-icon-180.png">
```

Apple-spezifisch: iOS ignoriert das Manifest in einigen Punkten und verlangt eigene Meta-Tags — daher die "apple-..."-Varianten zusätzlich. Das ist historisch gewachsen und nicht änderbar.

3.3 Service Worker

Ein **Service Worker** ist eine JavaScript-Datei (typisch: **sw.js**), die im Hintergrund läuft — auch wenn die Seite gar nicht offen ist. Er erfüllt mehrere Aufgaben:

- **Empfängt Push-Nachrichten** vom Server und zeigt sie als Notification an
- **Reagiert auf Notification-Taps** und öffnet die richtige Page der App
- **Optional: Caching** für Offline-Betrieb — kann Inhalte vorhalten, sodass die App auch ohne Internet reagiert

Der Service Worker wird beim ersten Aufruf der Seite einmalig vom Browser registriert und bleibt persistent — bis der Nutzer die PWA wieder löscht.

> **HTTPS ist Pflicht!** Service Worker funktionieren nur über verschlüsselte Verbindungen. Das ist eine Sicherheitsmaßnahme: Da der Service Worker im Hintergrund Daten empfängt und Push-Nachrichten zeigt, dürfen die Daten unterwegs nicht manipulierbar sein. Bei deinem DuckDNS+Let's-Encrypt-Setup ist das schon erledigt.

3.4 Icons

Mehrere Größen werden gebraucht, weil Android und iOS unterschiedliche Anforderungen haben:

Größe	Format	Verwendung
192×192	PNG	Android Standard-Icon
512×512	PNG	Android High-Res, Splash-Screen
512×512	PNG (maskable)	Android Adaptive Icons (mit Sicherheitszone)
180×180	PNG	iOS Apple-Touch-Icon (Homescreen)

Online-Generatoren wie **maskable.app** oder **realfavicongenerator.net** erstellen aus einer einzigen Vorlage alle benötigten Größen automatisch.

4. Wie der Nutzer die App installiert

4.1 Auf iPhone (Safari)

- Safari öffnen und die URL eingeben
- Unten in der Safari-Leiste das **Teilen-Icon** antippen (Quadrat mit Pfeil nach oben)
- Im Sheet runterscrollen → **"Zum Home-Bildschirm"**
- Vorschau prüfen: App-Name, Icon
- Oben rechts **"Hinzufügen"** tippen
- Das Icon erscheint auf dem Homescreen. von dort starten

iOS macht keinen automatischen Vorschlag zum Installieren — der Nutzer muss aktiv den Weg über das Teilen-Menü finden. Daher ist es sinnvoll, in der Webseite einen kleinen Hinweisbanner einzubauen, der iOS-Nutzern erklärt, wie sie installieren. Das hatten wir bei DMS Mobile auch eingebaut.

4.2 Auf Android (Chrome / Edge / Firefox)

- Webseite im Browser öffnen
- Browser zeigt automatisch einen Banner "App installieren?" oder "Zum Startbildschirm hinzufügen"
- Alternative: Im Menü (drei Punkte) → "App installieren" oder "Zum Startbildschirm hinzufügen"
- Bestätigen, fertig

Android ist deutlich aktiver beim Anbieten der Installation. Wenn die Webseite alle PWA-Kriterien erfüllt, schlägt Chrome es von selbst vor.

4.3 Auf Desktop (Chrome / Edge)

Auch am Desktop installierbar. In der Adressleiste erscheint rechts ein kleines Plus-Symbol oder ein "Installieren"-Button. Tap darauf, und die App bekommt ein eigenes Fenster ohne Browser-Chrome.

5. Wo PWAs funktionieren

Plattform	Installation	Push-Benachrichtigungen
iOS Safari	ab iOS 11.3 (2018)	ab iOS 16.4 (2023)
Android Chrome	ab Chrome 70 (2018)	ab Chrome 42 (2015)
Android Firefox	ab Version 79	ab Version 44
Desktop Chrome	alle Versionen	alle Versionen
Desktop Edge	alle Versionen	alle Versionen

Desktop Safari	ab macOS Sonoma (2023)	ab macOS Sonoma
Desktop Firefox	eingeschränkt	ja

Kurzfassung: PWAs laufen heute auf allen relevanten Geräten und Betriebssystemen. Push-Benachrichtigungen auf iOS sind der Spätzünder — die kamen erst 2023 und nur unter der Bedingung, dass die Seite als PWA installiert ist.

6. Was eine PWA nicht kann

Eine PWA ist mächtig, aber keine vollwertige native App. Was nicht geht (jedenfalls nicht auf iOS):

- **Bluetooth-Zugriff** (auf Android schon, auf iOS nicht)
- **NFC** (für Bezahlfunktionen, Smart-Home-Geräte etc.)
- **Tiefer Filesystem-Zugriff** — nur eingeschränkter App-Sandkasten
- **Keine Widgets** auf dem Homescreen (nur das App-Icon selbst)
- **Begrenzter Storage auf iOS** — lange Zeit auf 50 MB limitiert, seit iOS 17 mit Nutzerzustimmung mehr
- **Kein App Store Listing** — die App ist nicht in den Stores auffindbar, sondern nur über die URL erreichbar

Für die meisten Anwendungsfälle — besonders interne Tools wie deins — reichen die PWA-Möglichkeiten locker. Die Einschränkungen sind nur dann relevant, wenn du tief in Hardware-Features eintauchen willst (Bluetooth-LE, Kamera mit Spezial-APIs, Hintergrund-GPS-Tracking etc.).

7. Deinstallation und Updates

7.1 PWA löschen

Auf iPhone: Lange aufs App-Icon tippen → "App entfernen" → "Vom Home-Bildschirm entfernen". Damit verschwindet das Icon, der Service Worker, der LocalStorage, einfach alles — wie eine normale App-Deinstallation. Nach der Deinstallation kann jederzeit neu installiert werden, indem du die URL erneut in Safari aufrufst.

7.2 Updates

Bei einer normalen App geht ein Update über den App Store. Bei einer PWA passiert das anders:

- **HTML/CSS/JS-Änderungen** werden vom Browser bei der nächsten Online-Verbindung automatisch geholt — sofort wirksam beim nächsten App-Start
- **Service-Worker-Änderungen** brauchen einen Versions-Bump (im Code eine Variable hochzählen). Der Browser erkennt das, lädt die neue Version, aktiviert sie beim nächsten Start
- **Manifest-Änderungen** (Name, Icon, Start-URL) erfordern auf iOS oft eine komplette Neuinstallation der PWA — iOS cached das Manifest aggressiv

Vorteil: Du kannst jederzeit Updates ausrollen, ohne auf einen App-Store-Review-Prozess warten zu müssen. Nachteil: Auf iOS sind Manifest-Änderungen fummelig, weil die Nutzer ihre PWA neu installieren müssen.

8. Was kostet das?

8.1 Aufwand für eine bestehende Webseite

Wenn du eine bereits existierende Webseite zur PWA machen willst, ist der Aufwand überschaubar — vorausgesetzt, sie läuft schon über HTTPS. Schritt für Schritt:

Schritt	Aufwand	Was passiert
1. Manifest schreiben	~10 min	JSON-Datei mit App-Daten erzeugen
2. Icons erzeugen	~15 min	Generator nutzen, vier PNG-Größen
3. Service Worker schreiben	~30 min	Skelett-sw.js, Push-Handler optional
4. Meta-Tags einbauen	~5 min pro Page	Sechs Tags in den HTML-head
5. Testen auf iPhone	~15 min	Safari → Zum Homescreen → öffnen
Gesamt (ohne Push)	**~1 Stunde**	**Webseite installierbar als App**
Plus: Push integrieren	**+2-3 Stunden**	**VAPID-Keys, Server-Endpoint, Frontend-UI**

Hosting-Kosten: Null. Du brauchst nur eine Domain mit HTTPS und einen Webserver, der statische Dateien ausliefern kann — alles, was du sowieso schon hast. Kein App-Store-Account, kein Apple-Developer-Programm (99 Euro/Jahr gespart), kein Google-Play-Setup.

8.2 Push-Benachrichtigungen extra

Wenn du Push-Benachrichtigungen willst (so wie bei DMS Mobile), kommt noch dazu:

- **VAPID-Keypair** einmalig generieren (Public + Private Key für die Server-Identität)
- **Backend-Endpoints** für Subscriptions speichern und Pushes senden — meist eine Library wie web-push (Node) oder web-push-php (PHP)
- **Frontend-UI** — ein Knopf "Push aktivieren", der die Permission abfragt und die Subscription ans Backend schickt
- **Trigger** — wann werden Pushes gesendet? Cron-Jobs, Realtime-Events, etc.

Das ist mehr Arbeit als die reine PWA-Installation, aber technisch geradlinig. Bei DMS Mobile haben wir es in einer Nachmittags-Session geschafft.

9. Wann lohnt sich eine PWA?

Eine PWA ist die richtige Wahl, wenn:

- **Du selbst die Webseite kontrollierst** — du brauchst Server-Zugriff für Manifest, Service Worker, Icons
- **Deine Hauptzielgruppe einen Browser nutzt** — kein App-Store-Marketing nötig
- **Schnelle Updates wichtig sind** — keine Review-Wartezeit
- **Plattform-Unabhängigkeit zählt** — eine Codebasis für iPhone, Android, Desktop
- **Hardware-nahe Features nicht entscheidend sind** — kein Bluetooth-LE, kein NFC, kein Background-GPS
- **Push-Benachrichtigungen reichen** — klassische App-Features wie Widgets oder System-Integrationen brauchst du nicht

Eine native App ist die richtige Wahl, wenn:

- **App-Store-Auffindbarkeit** für dein Geschäftsmodell entscheidend ist
- **Tiefe Hardware-Integration** nötig ist (Sensoren, Bluetooth, spezielle Kamera-APIs)
- **Performance-kritische Echtzeit-Anforderungen** bestehen (3D-Spiele, Video-Bearbeitung)
- **In-App-Käufe** abgewickelt werden sollen (Apple/Google verlangen das im Store-Kontext)

Für interne Tools, Dashboards, Verwaltungs-Apps oder Hobby-Projekte ist die PWA fast immer die bessere Wahl. Du sparst dir die App-Store-Politik, hast volle Kontrolle, kannst jederzeit updaten und brauchst keinen Mac für die iOS-Entwicklung.

10. Merkliste, falls du nochmal eine PWA baust

Für später, wenn du eine andere Webseite "app-isieren" willst:

- HTTPS sicherstellen (sonst funktioniert kein Service Worker)
- manifest.webmanifest mit name, start_url, scope, display, theme_color, icons
- Icons in 4 Größen erzeugen: 192, 512, 512-maskable, 180-apple-touch
- sw.js mit install/activate/fetch-Listenern (minimal-Skelett reicht)
- Optional für Push: push- und notificationclick-Handler im sw.js
- Sechs Meta-Tags in den HTML-head jeder einstiegsfähigen Page
- Optional: A2HS-Hinweisbanner für iOS-Nutzer
- Optional: iOS-Standalone-Modus erkennen und Banner ausblenden
- Auf iPhone testen: Installation, Vollbild, Reload-Verhalten
- Bei Updates: Service-Worker-Version hochzählen, Cache-Strategie prüfen