The Definitive Guide to Cross-Platform Progressive Web App Push Notification Architecture

1. Introduction: The Convergence of Web and Native Messaging

The evolution of the mobile web has been defined by a persistent asymptotic approach toward native application capabilities. For nearly a decade, the "App Gap"—the functional disparity between native binaries installed via storefronts and web applications delivered via browsers—served as the primary differentiator in mobile strategy. Among these missing capabilities, **Push Notifications** represented the most significant retention mechanic available to developers. While the Android ecosystem, powered by the flexible Chromium engine, has supported the Web Push API for years, the iOS ecosystem remained a walled garden, restricting this critical engagement channel to native apps distributed through the App Store.

This landscape shifted fundamentally with the release of **iOS 16.4** in 2023. By implementing the W3C Push API and the Notifications API within WebKit, Apple effectively closed the loop, allowing Progressive Web Apps (PWAs) to achieve parity with native applications on the two dominant mobile operating systems.[1] This development allows a single codebase to engage users across billions of devices without the friction of app store approval processes or the overhead of maintaining separate Swift and Kotlin codebases.

However, the implementation of a truly cross-platform push notification system is far from trivial. It involves orchestrating a complex triangular architecture between the User Agent (browser), a third-party Push Service (Apple Push Notification service or Firebase Cloud Messaging), and an Application Server. It requires navigating divergent security models, strict battery optimization heuristics (such as Android's Doze mode and iOS's background execution limits), and precise cryptographic protocols (VAPID).

This report provides an exhaustive technical analysis and implementation guide for deploying production-grade push notifications for PWAs. It covers the full stack—from the manifest.json configuration required to trigger OS-level integration, to the Service Worker logic handling background events, to the Node.js infrastructure required to cryptographically sign and dispatch messages.

---

2. Architectural Foundations and The Web Push Protocol

To implement push notifications correctly, one must first understand the underlying architecture. Unlike traditional HTTP requests, which are client-initiated (pull), push notifications require the server to initiate communication. Since mobile devices cannot maintain open ports for every website due to battery constraints, the architecture relies on a persistent connection maintained by the Operating System to a centralized **Push Service**.

2.1 The Triangular Topology

The Web Push ecosystem operates on a decoupled triangular topology involving three distinct actors. This architecture is designed to ensure that the Application Server never communicates directly with the User Agent, preserving user privacy and device integrity.

1. **The User Agent (Client):** This is the browser (Safari, Chrome, Edge) running the PWA. It is responsible for requesting permission from the user, generating a push subscription (which includes the endpoint URL and encryption keys), and registering a Service Worker to handle incoming events.
2. **The Push Service (Intermediary):** This is a server maintained by the browser vendor. For Chrome on Android, this is usually Firebase Cloud Messaging (FCM). For Safari on iOS, it is the Apple Push Notification service (APNs). For Firefox, it is Mozilla's Push Service. This component is responsible for authenticating the Application Server, queuing messages when the device is offline, and delivering the message to the device via a persistent system-level connection.
3. **The Application Server (Backend):** This is the developer's server (Node.js, Python, etc.). It stores the user's subscription details and triggers notifications by sending encrypted HTTP/2 requests to the Push Service.[3]

2.2 The Protocol Stack

The communication between these entities is governed by a suite of IETF standards, primarily:

- **RFC 8030 (Generic Event Delivery Using HTTP Push):** Defines how the Application Server sends messages to the Push Service using HTTP POST requests.[4]
- **RFC 8291 (Message Encryption):** Specifies that all payloads must be encrypted using AES-128-GCM to ensure that the Push Service (Google/Apple) cannot read the content of the notifications it relays.
- **RFC 8292 (VAPID):** The Voluntary Application Server Identification protocol, which allows the Application Server to authenticate itself to the Push Service without needing to exchange proprietary API keys beforehand.[5]

2.3 Operating System Constraints and Divergences

While the *protocols* are standardized, the *implementation policies* differ significantly between Android and iOS. Understanding these differences is critical for a "write once, run anywhere" strategy.

Feature	Android (Chromium)	iOS (WebKit)
Prerequisite	None (works in browser tab)	**Must be installed to Home Screen** (Standalone) [1]
Permission Prompt	Can be triggered anytime (with heuristics)	**Requires explicit User Gesture** (tap/click) [8]
Silent Push	Supported (can run logic without showing UI)	**Strictly Forbidden** (must show visible notification) [10]
Icon Display	Customizable via payload (icon, badge)	Uses **Home Screen App Icon** only [12]
Badging	Supported via payload or Badging API	Uses App Icon Badge (requires permission)
Retention	High (Notification History)	Ephemeral (disappears if not interacted with)

The most critical insight here is the **iOS Home Screen Requirement**. Unlike on Android, where a user can visit a site and subscribe immediately, iOS users *must* first add the web application to their Home Screen. This promotes the PWA to a first-class citizen within the OS, granting it its own storage sandbox and the ability to register a Service Worker that persists beyond the browser session. This creates a significant User Experience (UX) friction point that must be addressed in the implementation strategy.[1]

---

3. The PWA Environment: Manifest and Installation Configuration

Before any JavaScript code can execute to request permissions, the web application must be properly configured to be recognized as an "installable" app by the host operating system. This is controlled by the Web App Manifest and the HTML head configuration.

3.1 The Web App Manifest (manifest.json)

The manifest is a JSON file that provides the browser with metadata about the web application. While Android is lenient with manifest errors, iOS is pedantic. A single missing field or incorrect MIME type can prevent the "Add to Home Screen" logic from recognizing the app as a PWA, thereby disabling push capabilities.

The following configuration represents the 2025 "Gold Standard" for cross-platform compatibility. It addresses the standalone requirement for iOS and the maskable icon requirement for modern Android.

Code Block 1: The Universal manifest.json

JSON

```
{
  "name": "EnterpriseConnect",
  "short_name": "EntConnect",
  "description": "Secure enterprise communication channel.",
  "start_url": "/",
  "scope": "/",
  "display": "standalone",
  "orientation": "portrait",
  "background_color": "#ffffff",
  "theme_color": "#0f172a",
  "icons": [
    {
      "src": "/assets/icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "any"
    },
    {
      "src": "/assets/icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any"
    },
    {
      "src": "/assets/icons/maskable-icon-192x192.png",
```

```
    "sizes": "192x192",
    "type": "image/png",
    "purpose": "maskable"
  },
  {
    "src": "/assets/icons/maskable-icon-512x512.png",
    "sizes": "512x512",
    "type": "image/png",
    "purpose": "maskable"
  }
 ]
}
```

3.1.1 Critical Configuration Analysis

- **display: "standalone"**: This is the single most important line for iOS support. As noted in research snippet [11], Web Push on iOS requires the app to look and feel like a native app. If this is set to browser or minimal-ui, iOS will treat the Home Screen shortcut as a bookmark, launching it in Safari rather than its own process. In Safari context, the Service Worker may be terminated more aggressively, and the Push API may be restricted. standalone ensures the removal of the URL bar and navigation controls, signaling to the OS that this is an application.[14]
- **icons and the Maskable Requirement**: Android 8.0 (Oreo) introduced Adaptive Icons, which normalize app icons into shapes (circles, squircles, rounded squares) defined by the device manufacturer. If a standard square icon is provided without the purpose: "maskable" property, Android attempts to fit the square inside the device's shape, often resulting in the icon being shrunk and placed on a white background—the "plate" effect. By providing a maskable icon (where the critical visual information is within a "safe zone" center circle), developers ensure the icon looks native on Pixel, Samsung, and other Android launchers.[16]
- **start_url**: This must be within the scope of the Service Worker. If the start_url redirects to a different domain or a path outside the scope, the Service Worker may not control the page upon launch, breaking the message handling logic.

3.2 The HTML Head Configuration (iOS Legacy Support)

While the W3C manifest is the standard, iOS retains legacy behaviors that require specific meta tags in the HTML document itself. Relying solely on the manifest often leads to the "screenshot icon" bug, where iOS uses a screenshot of the page as the icon instead of the defined asset.[13]

Code Block 2: Required HTML Head Elements

HTML

```
<link rel="manifest" href="/manifest.json">

<link rel="apple-touch-icon" href="/assets/icons/apple-touch-icon-180x180.png">

<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
<meta name="theme-color" content="#0f172a">
```

The apple-touch-icon should be a 180x180 pixel PNG. Unlike Android icons, **this icon should not be transparent**. iOS fills transparent areas with black, which often ruins the aesthetic. It is best practice to provide an opaque icon with a background color baked in for iOS.[18]

---

4. Cryptographic Identity and Security: The VAPID Protocol

Security is paramount in the Web Push architecture. Because the Push Service (e.g., Google's FCM) is an open relay, there must be a mechanism to ensure that only the legitimate Application Server can send notifications to a specific user. In the past, this required generating proprietary API keys (like the GCM Sender ID). This locked developers into specific vendors.

To solve this, the IETF introduced **VAPID (Voluntary Application Server Identification)**.

4.1 How VAPID Works

VAPID uses standard public-key cryptography (Elliptic Curve Digital Signature Algorithm, or ECDSA, using the P-256 curve).

1. **Key Generation:** The developer generates a VAPID key pair (Public and Private).
2. **Attestation (Frontend):** The Public Key is converted to a Uint8Array and passed to the browser during the subscribe() call.
3. **Registration (Push Service):** The browser sends this Public Key to the Push Service. The Push Service associates the new subscription endpoint with this Public Key.
4. **Authorization (Backend):** When the backend sends a notification, it creates a JSON Web Token (JWT) signed with the Private Key. It includes this JWT in the Authorization header of the HTTP request to the Push Service.
5. **Verification:** The Push Service verifies the signature using the stored Public Key. If they

match, the message is delivered.

This mechanism ensures that no one else can send messages to your users, even if they guess the subscription ID, because they lack the Private Key.[3]

4.2 Generating VAPID Keys

The standard tool for generating these keys is the web-push library. This library handles the complex ASN.1 formatting and Base64 URL-safe encoding required by the spec.

Execution Step: Run the following in your terminal to generate keys.

Bash

```
npm install -g web-push
web-push generate-vapid-keys
```

Output:
Public Key:
BNc.....
Private Key:
AbC.....
Storage Strategy: These keys are effectively the "root credentials" for your notification system. The Private Key must be stored in secure environment variables (e.g., .env file, AWS Secrets Manager). **Never commit the Private Key to version control.** If the Private Key is lost, you cannot send messages to existing subscribers. If it is compromised, an attacker can spam your users..[21]

---

5. Client-Side Engineering: The Service Worker

The Service Worker is the engine room of the PWA. It is a JavaScript file that runs in a background thread, separate from the main web page. This separation allows it to wake up and execute code even when the browser tab is closed (on Android) or when the app is suspended (on iOS).

5.1 Service Worker Registration

The registration process must be robust, handling potential errors and ensuring the worker is updated when the code changes.

Code Block 3: Registration Logic (client.js)

JavaScript

```
async function registerServiceWorker() {
 if ('serviceWorker' in navigator) {
  try {
   const registration = await navigator.serviceWorker.register('/sw.js', {
    scope: '/'
   });
   console.log('Service Worker registered with scope:', registration.scope);
   return registration;
  } catch (error) {
   console.error('Service Worker registration failed:', error);
  }
 } else {
  console.warn('Push notifications are not supported in this browser.');
 }
}
```

5.2 The push Event Handler

When the Push Service delivers a message to the device, the browser wakes up the Service Worker and dispatches a push event. The Service Worker is responsible for parsing the data and displaying the notification.

Constraint: The browser imposes a "Must Show Notification" rule. If the push event handler executes but fails to show a notification (e.g., due to an error or logic branch), the browser may display a default, generic notification: "This site has been updated in the background." This is a negative UX and can lead to the browser revoking push permissions. To avoid this, showNotification must always be called.[10]

Code Block 4: The push Event Handler (sw.js)

JavaScript

```
self.addEventListener('push', function(event) {
```

```
  let data = {};

  // 1. Parse Payload
  if (event.data) {
    try {
      data = event.data.json();
    } catch (e) {
      // Fallback for plain text
      data = { title: 'Notification', body: event.data.text() };
    }
  }

  // 2. Configure Notification Options
  const options = {
    body: data.body |

| 'New content available',
    // Android-specific: Large icon on the right
    icon: '/assets/icons/icon-192x192.png',
    // Android-specific: Small monochrome icon for status bar
    badge: '/assets/icons/badge-96x96.png',
    // Data passed to the click handler
    data: {
      url: data.url |

| '/',
      timestamp: Date.now()
    },
    // Vibration pattern (Android only)
    vibrate: ,
    // Actions (Buttons)
    actions: data.actions |

|
  };

  // 3. Show Notification & Keep SW Alive
  // event.waitUntil is CRITICAL. It extends the lifetime of the push event
  // until the promise settles. Without it, the browser may terminate the
  // worker before the notification is displayed.
  event.waitUntil(
    self.registration.showNotification(data.title |
```

```
| 'App Name', options)
  );
});
```

5.2.1 Platform Nuances in Visuals

- **Android:** Supports icon (large image), badge (status bar icon), image (large picture body), and actions (buttons). The badge should be a PNG with transparency; Android tints it with the system color.
- **iOS:** Ignores the icon and badge properties in the visual banner. It strictly uses the PWA's Home Screen icon. It does, however, support actions if configured correctly.[12]

5.3 The notificationclick Event Handler

The default behavior when a user clicks a notification is... nothing. The notification closes, and that's it. To open the app, the developer must explicitly handle the notificationclick event.

This handler requires sophisticated logic to avoid "tab clutter." Instead of blindly opening a new window every time, the code should check if the app is already open. If it is, it should focus that existing window. If not, it should open a new one. This is achieved using the clients.matchAll() API.[24]

Code Block 5: Intelligent Window Management (sw.js)

JavaScript

```
self.addEventListener('notificationclick', function(event) {
  // 1. Close the notification immediately to clear the tray
  event.notification.close();

  // 2. Extract the deep link URL
  const targetUrl = new URL(event.notification.data.url, self.location.origin).href;

  // 3. Window Focus Logic
  const promiseChain = clients.matchAll({
    type: 'window',
    includeUncontrolled: true // Look for tabs not currently controlled (e.g. fresh loads)
  }).then((windowClients) => {
    // Strategy: Find a window strictly matching the URL, or at least the same origin
    let matchingClient = null;
```

```
    for (let i = 0; i < windowClients.length; i++) {
      const client = windowClients[i];
      // Check if the client is visible and matches origin
      if (client.url === targetUrl && 'focus' in client) {
        return client.focus();
      }
    }

    // If no matching window found, open a new one
    if (clients.openWindow) {
      return clients.openWindow(targetUrl);
    }
  });

  event.waitUntil(promiseChain);
});
```

Insight on iOS Multitasking: On iOS, clients.matchAll behaves differently depending on whether the app is in the background or fully suspended. If the app is suspended, the array may be empty. The clients.openWindow command on iOS, when the app is installed as a PWA, correctly launches the Standalone PWA instance rather than a Safari tab, preserving the native-like experience.[7]

---

6. User Experience and Permission Strategy

The most significant implementation hurdle for cross-platform PWA push is the User Experience flow, specifically bridging the gap between Android's permissive model and iOS's restrictive model.

6.1 The "Double Permission" Pattern

On iOS 16.4+, calling Notification.requestPermission() will **fail silently** or be automatically denied if:

1. The app is not installed to the Home Screen.
2. The call is not triggered by a user gesture (e.g., a button click).

Therefore, a naive "ask on page load" strategy will result in 0% conversion on iOS. The application must implement a "Double Permission" or "Educational" UI pattern.

1. **Contextual Prompt (Soft Ask):** The UI displays a custom in-app banner: "Enable notifications to stay updated?"
2. **OS Check:** When the user clicks "Yes", the code checks if it is running on iOS and if it is in Standalone mode.

- **If iOS + Browser:** Show a modal explaining how to "Add to Home Screen" (Share -> Add to Home Screen).
- **If iOS + Standalone (or Android):** Call the native Notification.requestPermission().

Code Block 6: Cross-Platform Permission Logic

JavaScript

```
// Helper to detect iOS
const isIOS = () => {
  return /iPad|iPhone|iPod/.test(navigator.userAgent) &&!window.MSStream;
};

// Helper to detect Standalone Mode
const isStandalone = () => {
  return ('standalone' in window.navigator) && (window.navigator.standalone);
};

async function handleSubscriptionRequest() {
  // 1. iOS Browser Guard
  if (isIOS() &&!isStandalone()) {
    // Trigger custom UI modal here
    showInstallInstructions();
    return;
  }

  // 2. Request Native Permission
  // MUST be inside a user-gesture handler (click event)
  const permission = await Notification.requestPermission();

  if (permission === 'granted') {
    await subscribeUserToPush();
  } else {
    console.log('User denied permissions');
  }
}

// Bind to UI Button
document.getElementById('notify-btn').addEventListener('click', handleSubscriptionRequest);
```

6.2 Subscription Generation

Once permission is granted, the client must subscribe to the push service. This involves passing the VAPID Public Key (converted to a Uint8Array) to the Push Manager.

Code Block 7: Subscription Logic

JavaScript

```
function urlBase64ToUint8Array(base64String) {
  const padding = '='.repeat((4 - base64String.length % 4) % 4);
  const base64 = (base64String + padding).replace(/-/g, '+').replace(/_/g, '/');
  const rawData = window.atob(base64);
  const outputArray = new Uint8Array(rawData.length);
  for (let i = 0; i < rawData.length; ++i) {
    outputArray[i] = rawData.charCodeAt(i);
  }
  return outputArray;
}

async function subscribeUserToPush() {
  const registration = await navigator.serviceWorker.ready;

  const subscribeOptions = {
    userVisibleOnly: true, // Mandatory for standard push
    applicationServerKey: urlBase64ToUint8Array('YOUR_PUBLIC_VAPID_KEY')
  };

  try {
    const subscription = await registration.pushManager.subscribe(subscribeOptions);
    // Send the subscription object to the server
    await sendSubscriptionToServer(subscription);
  } catch (err) {
    console.error('Failed to subscribe the user: ', err);
  }
}
```

The userVisibleOnly: true parameter is a contractual obligation. It tells the browser, "I promise that every push message I receive will result in a visible notification." Browsers enforce this to prevent developers from using push notifications as a stealthy background tracking or

data-syncing mechanism without user awareness.[27]

---

7. Server-Side Engineering: Node.js Implementation

The backend is responsible for storing user subscriptions and dispatching encrypted messages to the Push Service endpoints. While one could manually implement the encryption (HKDF, AES-GCM), the web-push library for Node.js is the industry standard abstraction.

7.1 Prerequisite Configuration

Install the necessary libraries:

Bash

```
npm install web-push express body-parser
```

7.2 The Dispatch Engine

The following Node.js implementation demonstrates a production-ready endpoint. It highlights two critical requirements often missed in basic tutorials:

1. **The Urgency Header:** Required for reliable delivery on iOS.
2. **Error Handling (410 Gone):** Required to maintain database hygiene.

Code Block 8: Node.js Backend (server.js)

JavaScript

```
const webpush = require('web-push');
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.json());

// CONFIGURATION
// These should be loaded from environment variables in production
```

```
const publicVapidKey = process.env.VAPID_PUBLIC_KEY;
const privateVapidKey = process.env.VAPID_PRIVATE_KEY;
const adminEmail = 'mailto:admin@yourcompany.com'; // Required by spec

webpush.setVapidDetails(
  adminEmail,
  publicVapidKey,
  privateVapidKey
);

// MOCK DATABASE
// In production, use MongoDB, PostgreSQL, or Redis
let subscriptions =;

// 1. Save Subscription Endpoint
app.post('/api/subscribe', (req, res) => {
  const subscription = req.body;
  // Validate subscription object structure
  if (!subscription.endpoint ||!subscription.keys) {
    return res.status(400).json({ error: 'Invalid subscription object' });
  }

  subscriptions.push(subscription);
  res.status(201).json({});
});

// 2. Trigger Push Endpoint
app.post('/api/trigger-push', async (req, res) => {
  const { title, body, url } = req.body;

  const notificationPayload = JSON.stringify({
    title: title,
    body: body,
    url: url,
    // Add logic for actions here
  });

  const promises = subscriptions.map(sub => {
    return webpush.sendNotification(
      sub,
      notificationPayload,
      {
        // ADVANCED CONFIGURATION
```

```
      headers: {
        // 'Urgency' is critical for iOS power management interaction
        'Urgency': 'normal',
      }
    }
  ).catch(err => {
    // 3. Handling Expired Subscriptions
    if (err.statusCode === 410 |

| err.statusCode === 404) {
      console.log('Subscription expired (User likely uninstalled PWA). Deleting...');
      // Remove 'sub' from database
      return deleteSubscriptionFromDB(sub.endpoint);
    }
    console.error('Push error:', err);
  });
 });

 await Promise.all(promises);
 res.status(200).json({ message: 'Notifications processed' });
});

app.listen(5000, () => console.log('Server started on port 5000'));
```

7.3 Deep Dive: The Urgency Header and iOS Power Management

The Urgency header is defined in RFC 8030, but its importance skyrocketed with iOS 16.4. Apple's APNs is aggressive about battery preservation. If a server sends messages without an urgency preference, or incorrectly flags everything as high, APNs may throttle delivery.

Urgency Value	APNs Priority	Use Case	Battery Impact
high	10	Critical alerts, chat messages, incoming calls. Wakes device immediately.	High. May be throttled if excessive.
normal	5	Social updates, content alerts, marketing.	Low. Delivered when device is active or energy

			efficient.
low	-	Non-urgent data updates.	Very Low.
very-low	-	Background sync (often not supported for Web Push).	Negligible.

Best Practice: Default to normal for 90% of notifications. Only use high for time-sensitive, transactional interactions (e.g., "Your taxi has arrived"). Misusing high urgency for marketing spam is a violation of APNs policy and can lead to silent delivery failures.[28]

---

8. Production Readiness: Scaling and Reliability

The Node.js example above uses a simple array loop (Promise.all), which is sufficient for testing but disastrous for production. If you have 100,000 subscribers, Promise.all will attempt to open 100,000 HTTP connections simultaneously, crashing the Node process or triggering rate limiters at the Push Service.

8.1 The Queuing Architecture

For production, the dispatch logic must be decoupled from the HTTP request handler using a job queue (e.g., BullMQ with Redis, or RabbitMQ).

Mechanism:

1. **API Handler:** Receives the "Send Newsletter" request. It queries the database for all 100,000 active subscriptions.
2. **Producer:** It pushes 100,000 jobs onto a Redis queue. Each job contains the payload and the specific user's subscription.
3. **Worker:** A separate fleet of worker processes pulls jobs from the queue. They process them in controlled batches (e.g., concurrency of 50).
4. **Rate Limiting:** The workers respect the rate limits of the Push Services. While Web Push endpoints generally handle high volume, regulating outbound traffic prevents backend congestion.
5. **Retry Logic:** If a request fails with a 5xx error (Push Service internal error) or network timeout, the queue automatically retries the job with exponential backoff.[31]

8.2 Handling The "Ghost" Problem (410 Gone)

User churn in PWAs is invisible. Unlike an uninstallation event in a native app store which might trigger a webhook, a user simply deleting a PWA or clearing browser cookies leaves no trace. The Application Server retains the subscription in its database.

When the server attempts to push to this "ghost" subscription, the Push Service responds with HTTP status 410 Gone.

Critical Implementation Detail: The backend *must* listen for 410 errors and immediately delete the subscription from the database. If this cleanup is ignored, the database fills with dead rows, slowing down queries, and the server wastes resources encrypting and sending messages that will never be delivered. Furthermore, Push Services monitor the error rate; a sender with a 90% error rate may be flagged as spam.[32]

---

9. Advanced Platform Nuances and Debugging

9.1 iOS WebKit Specifics

- **No "Silent Push":** In native iOS, "silent push" (content-available: 1) allows an app to wake up, fetch data, and update the UI without alerting the user. Web Push on iOS *does not* support this. Every push event must result in a user-visible notification.
- **Icon Limitations:** As previously noted, the notification icon cannot be changed dynamically. It will always be the Home Screen icon.
- **Badge Count:** The Web Push API interacts with the Badging API (navigator.setAppBadge). On iOS, setting the badge count via the Service Worker works, but it updates the red dot on the Home Screen icon, not the banner itself.

9.2 Android Chromium Specifics

- **Notification Grouping:** Android automatically groups notifications from the same origin. You can control this grouping using the tag property in the showNotification options. If a new notification arrives with the same tag as an existing one, it replaces the old one rather than stacking. This is useful for "Score Updates" where you only want the latest score visible.
- **Interaction Requirements:** Android requires the user to have interacted with the page at least once before permission can be requested. This is generally handled by the browser blocking the prompt, but on some versions, it throws an error.

9.3 Troubleshooting Guide

- **Issue:** Notification shows on Desktop but not Mobile.
    - **Cause:** Mobile devices often have "Data Saver" or "Battery Optimization" modes that restrict background processes. On Android, verify the PWA is not "Restricted" in App Battery usage. On iOS, verify "Background App Refresh" is enabled (though Web

Push bypasses some of this, system-wide Low Power Mode can delay delivery).

- **Issue:** "Permission Denied" immediately on iOS.
  - **Cause:** You are likely calling requestPermission outside of a click handler, or the app is not in standalone mode.
- **Issue:** Service Worker error "window is not defined".
  - **Cause:** You are trying to access DOM elements or window properties inside sw.js. The Service Worker runs in a worker context, not the window context. It has no access to the DOM.

---

10. Conclusion

Implementing cross-platform push notifications for Progressive Web Apps is a rigorous engineering challenge that demands strict adherence to emerging standards and platform-specific policies. The introduction of Web Push in iOS 16.4 marked a turning point, enabling a unified messaging strategy that encompasses the vast majority of mobile users.

By configuring the **Web App Manifest** with standalone and maskable properties, implementing **VAPID** for secure identification, designing a robust **Service Worker** that manages window focus intelligently, and deploying a **Node.js** backend that respects the **Urgency** protocol and handles **410 Gone** cleanup, developers can build a notification system that rivals native performance.

The "write once, deploy everywhere" dream of the web is now a reality for engagement strategies, provided the implementation respects the distinct architectural personalities of Android and iOS.

Configuration Core Summary (Cheatsheet)

Component	Critical Configuration Step
Manifest	Set display: "standalone". Include maskable icons for Android.
iOS HTML	Add apple-touch-icon (180px, no transparency) and apple-mobile-web-app-capable.
Frontend	Guard Notification.requestPermission with isIOS() + isStandalone() checks.

Service Worker	Use event.waitUntil. Implement clients.matchAll for click handling.
Backend	Sign with VAPID keys. Set Urgency: 'normal' header. Handle 410 errors.
UX	Use "Double Permission" pattern. Guide iOS users to "Add to Home Screen".

Works cited

1. iOS web push setup - OneSignal Documentation, accessed on December 16, 2025, https://documentation.onesignal.com/docs/en/web-push-for-ios
2. Sending web push notifications in web apps and browsers - Apple Developer, accessed on December 16, 2025, https://developer.apple.com/documentation/usernotifications/sending-web-push-notifications-in-web-apps-and-browsers
3. The Web Push Protocol | Articles - web.dev, accessed on December 16, 2025, https://web.dev/articles/push-notifications-web-push-protocol
4. RFC 8030: Generic Event Delivery Using HTTP Push, accessed on December 16, 2025, https://www.rfc-editor.org/rfc/rfc8030.html
5. Web Push: what is VAPID (applicationServerKey)? - Pushpad, accessed on December 16, 2025, https://pushpad.xyz/blog/web-push-what-is-vapid
6. RFC 8292 - Voluntary Application Server Identification (VAPID) for Web Push, accessed on December 16, 2025, https://datatracker.ietf.org/doc/html/rfc8292
7. How do I enable iOS Web Push notifications on my PWA website? - Batch Documentation, accessed on December 16, 2025, https://doc.batch.com/developer/technical-guides/how-to-guides/web/how-to-integrate-batchs-snippet-using-google-tag-manager/how-do-i-enable-ios-web-push-notifications-on-my-pwa-website
8. Asking permission to use notifications | Apple Developer Documentation, accessed on December 16, 2025, https://developer.apple.com/documentation/usernotifications/asking-permission-to-use-notifications
9. Notification: requestPermission() static method - Web APIs | MDN, accessed on December 16, 2025, https://developer.mozilla.org/en-US/docs/Web/API/Notification/requestPermission_static
10. Notification behavior | Articles | web.dev, accessed on December 16, 2025, https://web.dev/articles/push-notifications-notification-behaviour
11. Setting Up Web Push Notifications on iOS and iPadOS - PushEngage, accessed on December 16, 2025, https://www.pushengage.com/documentation/setting-up-web-push-notifications

-for-ios-ipad/
12. PWA Android & iOS look of push notification - firebase - Stack Overflow, accessed on December 16, 2025, https://stackoverflow.com/questions/79745528/pwa-android-ios-look-of-push-notification
13. PWA Icon on iOS is not available after adding to home screen - Stack Overflow, accessed on December 16, 2025, https://stackoverflow.com/questions/75219514/pwa-icon-on-ios-is-not-available-after-adding-to-home-screen
14. How to Set Up Push Notifications for Your PWA (iOS and Android) - MobiLoud, accessed on December 16, 2025, https://www.mobiloud.com/blog/pwa-push-notifications
15. Different manifest.json version for Android and iOS ? : r/PWA - Reddit, accessed on December 16, 2025, https://www.reddit.com/r/PWA/comments/nrahb0/different_manifestjson_version_for_android_and_ios/
16. Web app manifest - PWA - web.dev, accessed on December 16, 2025, https://web.dev/learn/pwa/web-app-manifest
17. Why are PWA icons still a mess in 2025? (And my attempt to fix the 'white circle' problem), accessed on December 16, 2025, https://www.reddit.com/r/PWA/comments/1p92mor/why_are_pwa_icons_still_a_mess_in_2025_and_my/
18. How to Favicon in 2025: Three files that fit most needs - Evil Martians, accessed on December 16, 2025, https://evilmartians.com/chronicles/how-to-favicon-in-2021-six-files-that-fit-most-needs
19. PWA Icon Requirements and Safe Areas Explained | Logo Foundry - Professional Logo to Favicon & App Icon Generator, accessed on December 16, 2025, https://logofoundry.app/blog/pwa-icon-requirements-safe-areas
20. Web Push Interoperability Wins | Blog - Chrome for Developers, accessed on December 16, 2025, https://developer.chrome.com/blog/web-push-interop-wins
21. Vapid Key Generator | VapidKeys.com, accessed on December 16, 2025, https://vapidkeys.com/
22. generate-vapid-keys.ts - negrel/webpush - GitHub, accessed on December 16, 2025, https://github.com/negrel/webpush/blob/master/cmd/generate-vapid-keys.ts
23. How to generate web push vapid keys in node js? - Stack Overflow, accessed on December 16, 2025, https://stackoverflow.com/questions/63979872/how-to-generate-web-push-vapid-keys-in-node-js
24. WindowClient: focus() method - Web APIs - MDN Web Docs, accessed on December 16, 2025, https://developer.mozilla.org/en-US/docs/Web/API/WindowClient/focus
25. ServiceWorkerGlobalScope: notificationclick event - Web APIs | MDN, accessed on December 16, 2025,

https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerGlobalScope/notificationclick_event
26. Common Notification Patterns - Base Site - Web Push Book, accessed on December 16, 2025, https://web-push-book.gauntface.com/common-notification-patterns/
27. Using the Notifications API - MDN Web Docs, accessed on December 16, 2025, https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API/Using_the_Notifications_API
28. Set the priority of a message | Firebase Cloud Messaging - Google, accessed on December 16, 2025, https://firebase.google.com/docs/cloud-messaging/customize-messages/setting-message-priority
29. Master iOS 18 Priority Notifications and Summaries: All You Need to Know - EngageLab, accessed on December 16, 2025, https://www.engagelab.com/blog/ios-18-priority-notifications
30. Sending notification requests to APNs | Apple Developer Documentation, accessed on December 16, 2025, https://developer.apple.com/documentation/usernotifications/sending-notification-requests-to-apns
31. Implementing Reliable Push Notifications with Queueing and Retry in Node.js Backend | by Ramesh Vantaku | Medium, accessed on December 16, 2025, https://medium.com/@rameshchanduv.a/implementing-reliable-push-notifications-with-queueing-and-retry-in-node-js-backend-fc226569252
32. Full Stack Web Push API Guide - Bocoup, accessed on December 16, 2025, https://www.bocoup.com/blog/full-stack-web-push-api-guide