drop-push-notifications-spec

Drop Push Notification Delivery Service — Implementation Spec

Project: Drop (Fintech Payment App) **Task:** MC #1196 **Date:** 2026-02-17 **Author:** John (AI Director)

---

1. Executive Summary

Drop currently has notification stubs (database table, UI, demo-mode service) but no actual push delivery infrastructure. This spec defines a production-ready push notification system covering:

- Multi-platform delivery: Web Push (PWA primary), Firebase Cloud Messaging (Android future), APNs (iOS future)
- Notification taxonomy with security and compliance categories
- Device token management and lifecycle
- User preference management per Norwegian marketing laws
- Database schema for device tokens, notification queue, delivery logs
- API endpoints for device registration and preference management
- Integration with existing transaction and auth systems

MVP Recommendation: Web Push (PWA) — Drop currently has web app only, mobile apps planned for future. Start with Web Push API + service workers, add FCM/APNs when React Native mobile apps ship.

---

2. Architecture Overview

2.1 Unified Push Service Layer

File: `src/lib/push.ts` (NEW — replaces stub at `src/lib/services/notifications.ts`)

Provider-agnostic push notification abstraction. All push sending goes through this module.

Interface:

```
// src/lib/push.ts

export interface DeviceToken {
  id: string;
  userId: string;
  platform: 'web' | 'android' | 'ios';
  token: string;           // For FCM/APNs, this is the device token
  endpoint?: string;       // For Web Push, this is the subscription endpoint
  keys?: {                 // For Web Push only
    p256dh: string;
    auth: string;
  };
  createdAt: string;
  lastUsedAt: string;
}

export interface NotificationPayload {
  userId: string;
  type: NotificationType;
  title: string;
  body: string;
  data?: Record<string, unknown>;
  priority?: 'high' | 'normal' | 'low';
  category?: string;       // For iOS notification categories
  badge?: number;          // Badge count (iOS/Android)
  sound?: string;          // Sound file name
}

export interface DeliveryResult {
  success: boolean;
  messageId?: string;
  platform: 'web' | 'android' | 'ios';
  error?: string;
}

// Core send function
```

```
export async function sendPushNotification(
  payload: NotificationPayload
): Promise<DeliveryResult[]>;

// Device token management
export async function registerDeviceToken(
  userId: string,
  platform: 'web' | 'android' | 'ios',
  token: string,
  keys?: { p256dh: string; auth: string; endpoint: string }
): Promise<{ success: boolean; error?: string }>;

export async function unregisterDeviceToken(
  userId: string,
  deviceId: string
): Promise<{ success: boolean }>;

export async function refreshDeviceToken(
  oldToken: string,
  newToken: string
): Promise<{ success: boolean }>;

// Preference check
export async function canSendNotification(
  userId: string,
  type: NotificationType
): Promise<boolean>;
```

2.2 Notification Type Taxonomy

Security Principle: Transactional notifications = mandatory (no opt-out). Promotional = explicit consent required (Norwegian markedsføringsloven).

```
export type NotificationType =
  // TRANSACTIONAL (mandatory, no opt-out)
  | 'transfer_sent'           // Money sent from user's account
  | 'transfer_received'       // Money received into user's account
  | 'transfer_failed'         // Transfer failed (bank rejected)
  | 'login_alert'             // New device/location login
```

```
  | 'otp_code'                // OTP code for 2FA (future)
  | 'password_changed'        // Password changed (security alert)
  | 'bankid_linked'           // BankID linked to account
  | 'bankid_unlinked'         // BankID unlinked (security alert)
  | 'account_locked'          // Account locked due to suspicious activity
  | 'kyc_approved'            // KYC verification approved
  | 'kyc_rejected'            // KYC verification rejected

  // ACCOUNT (opt-in, enabled by default)
  | 'transaction_summary'     // Daily/weekly transaction summary
  | 'low_balance'             // Bank account balance below threshold
  | 'rate_update'             // Exchange rate update for pending transfer

  // PROMOTIONAL (opt-in, disabled by default, GDPR consent required)
  | 'referral'                // Referral program
  | 'new_feature'             // New feature announcement
  | 'special_offer';          // Special offers/promotions

export const NOTIFICATION_CATEGORIES = {
  transactional: [
    'transfer_sent',
    'transfer_received',
    'transfer_failed',
    'login_alert',
    'otp_code',
    'password_changed',
    'bankid_linked',
    'bankid_unlinked',
    'account_locked',
    'kyc_approved',
    'kyc_rejected',
  ],
  account: [
    'transaction_summary',
    'low_balance',
    'rate_update',
  ],
  promotional: [
    'referral',
    'new_feature',
```

```
    'special_offer',
  ],
} as const;

export const NOTIFICATION_PRIORITY = {
  transfer_sent: 'high',
  transfer_received: 'high',
  transfer_failed: 'high',
  login_alert: 'high',
  otp_code: 'high',
  password_changed: 'high',
  bankid_unlinked: 'high',
  account_locked: 'high',
  kyc_approved: 'normal',
  kyc_rejected: 'normal',
  bankid_linked: 'normal',
  transaction_summary: 'normal',
  low_balance: 'normal',
  rate_update: 'low',
  referral: 'low',
  new_feature: 'low',
  special_offer: 'low',
} as const;
```

Norwegian Marketing Law Compliance:

- **Markedsføringsloven §15:** Promotional notifications require EXPLICIT prior consent (opt-in)
- **GDPR Art. 6(1)(a):** Consent must be freely given, specific, informed, unambiguous
- **Implementation:** Promotional notifications OFF by default, require explicit user action to enable
- **Audit trail:** Log consent timestamp and context in `notification_preferences` table

---

2.3 Platform-Specific Implementations

A. Web Push API (PRIMARY — MVP)

Tech Stack:

- Service worker (`public/sw.js`) handles push events
- Web Push API (browser native, no SDK required)

- `web-push` npm library (server-side, VAPID signing)

VAPID (Voluntary Application Server Identification):

- Generate keys: `npx web-push generate-vapid-keys`
- Store in env: `VAPID_PUBLIC_KEY`, `VAPID_PRIVATE_KEY`, `VAPID_SUBJECT` (mailto:support@getdrop.no)

Client-Side Flow:

1. User visits Drop PWA
2. Service worker registers (`/sw.js`)
3. User grants notification permission (browser prompt)
4. Client subscribes to push: `registration.pushManager.subscribe({ userVisibleOnly: true, applicationServerKey: VAPID_PUBLIC_KEY })`
5. Client sends subscription object to server: `POST /api/notifications/register-device`
6. Server stores subscription in `device_tokens` table

Server-Side Flow:

1. Transaction completes → create notification in `notifications` table
2. Fetch user's Web Push subscriptions from `device_tokens` WHERE platform='web'
3. For each subscription:
   - Use `web-push` library to send notification
   - `webpush.sendNotification(subscription, JSON.stringify(payload))`
4. Log delivery result in `notification_log`

Service Worker (`public/sw.js`):

```
// Listen for push events
self.addEventListener('push', (event) => {
  const data = event.data ? event.data.json() : {};
  const { title, body, icon, badge, data: customData } = data;

  const options = {
    body: body,
    icon: icon || '/icon-192.png',
    badge: badge || '/badge-72.png',
    data: customData,
    vibrate: [200, 100, 200],
    tag: data.type || 'default',
    requireInteraction: data.priority === 'high',
  };
```

```
  event.waitUntil(
    self.registration.showNotification(title, options)
  );
});

// Handle notification click
self.addEventListener('notificationclick', (event) => {
  event.notification.close();
  const urlToOpen = event.notification.data?.url || '/dashboard';

  event.waitUntil(
    clients.matchAll({ type: 'window', includeUncontrolled: true }).then((clientList) => {
      // If Drop is already open, focus it
      for (const client of clientList) {
        if (client.url.includes(urlToOpen) && 'focus' in client) {
          return client.focus();
        }
      }
      // Otherwise, open new window
      if (clients.openWindow) {
        return clients.openWindow(urlToOpen);
      }
    })
  );
});
```

Dependencies:

```
{
  "dependencies": {
    "web-push": "^3.6.7"
  }
}
```

Env Vars:

```
# Web Push (VAPID keys)
VAPID_PUBLIC_KEY=BN...  # Public key (also exposed to client via /api/vapid-public-key)
VAPID_PRIVATE_KEY=...   # Private key (server-side only)
VAPID_SUBJECT=mailto:support@getdrop.no
```

B. Firebase Cloud Messaging (FUTURE — Android)

When: When React Native Android app ships.

Tech Stack:

- Firebase Cloud Messaging (FCM) HTTP v1 API
- `firebase-admin` SDK (server-side)
- `@react-native-firebase/messaging` (client-side)

Setup:

1. Create Firebase project at console.firebase.google.com
2. Add Android app to Firebase project (package name: `no.getdrop.app`)
3. Download `google-services.json`, place in React Native Android project
4. Download service account key JSON for server
5. Set env var: `FIREBASE_SERVICE_ACCOUNT_KEY` (base64-encoded JSON)

Client-Side Flow (React Native):

```
// React Native app startup
import messaging from '@react-native-firebase/messaging';

async function registerForPushNotifications() {
  const authStatus = await messaging().requestPermission();
  if (authStatus === messaging.AuthorizationStatus.AUTHORIZED) {
    const token = await messaging().getToken();
    // Send to server: POST /api/notifications/register-device
    await fetch('/api/notifications/register-device', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ platform: 'android', token }),
    });
  }
}

// Handle foreground messages
messaging().onMessage(async (remoteMessage) => {
  // Show in-app notification UI
});

// Handle background messages (background handler in index.js)
messaging().setBackgroundMessageHandler(async (remoteMessage) => {
```

```
  console.log('Background message:', remoteMessage);
});

// Handle token refresh
messaging().onTokenRefresh((token) => {
  // Update server with new token
});
```

Server-Side Flow:

```
import admin from 'firebase-admin';

// Initialize Firebase Admin (once on startup)
const serviceAccountKey = JSON.parse(
  Buffer.from(process.env.FIREBASE_SERVICE_ACCOUNT_KEY!, 'base64').toString('utf-8')
);

admin.initializeApp({
  credential: admin.credential.cert(serviceAccountKey),
});

// Send notification
async function sendFCMNotification(token: string, payload: NotificationPayload) {
  const message = {
    token: token,
    notification: {
      title: payload.title,
      body: payload.body,
    },
    data: payload.data || {},
    android: {
      priority: payload.priority === 'high' ? 'high' : 'normal',
      notification: {
        sound: payload.sound || 'default',
        badge: payload.badge,
      },
    },
  };

  const response = await admin.messaging().send(message);
```

```
  return response; // message ID
}
```

Dependencies:

```
{
  "dependencies": {
    "firebase-admin": "^12.0.0"
  }
}
```

Env Vars:

```
FIREBASE_SERVICE_ACCOUNT_KEY=base64-encoded-json
```

Cost: FREE — FCM has no quota limits or pricing.

C. Apple Push Notification service (FUTURE — iOS)

When: When React Native iOS app ships.

Tech Stack:

- APNs HTTP/2 API
- `apn` npm library (server-side)
- `@react-native-firebase/messaging` (works for both FCM and APNs)

Setup:

1. Create iOS app in Apple Developer account
2. Create Push Notification certificate or Auth Key (.p8 file)
3. Download .p8 file, note Key ID and Team ID
4. Set env vars: `APNS_KEY_ID`, `APNS_TEAM_ID`, `APNS_KEY_PATH` (or `APNS_KEY_CONTENT` as base64)

Server-Side Flow:

```
import apn from 'apn';

// Initialize APNs provider
const apnProvider = new apn.Provider({
  token: {
    key: process.env.APNS_KEY_CONTENT!, // .p8 file content
    keyId: process.env.APNS_KEY_ID!,
```

```
    teamId: process.env.APNS_TEAM_ID!,
  },
  production: process.env.NODE_ENV === 'production',
});

// Send notification
async function sendAPNsNotification(deviceToken: string, payload: NotificationPayload) {
  const notification = new apn.Notification();
  notification.alert = {
    title: payload.title,
    body: payload.body,
  };
  notification.badge = payload.badge;
  notification.sound = payload.sound || 'default';
  notification.category = payload.category;
  notification.priority = payload.priority === 'high' ? 10 : 5;
  notification.payload = payload.data || {};
  notification.topic = 'no.getdrop.app'; // iOS bundle ID

  const result = await apnProvider.send(notification, deviceToken);
  return result; // Array of { device, status }
}
```

Dependencies:

```
{
  "dependencies": {
    "apn": "^2.2.0"
  }
}
```

Env Vars:

```
APNS_KEY_ID=ABC123XYZ
APNS_TEAM_ID=DEF456UVW
APNS_KEY_CONTENT=base64-encoded-p8-file
```

Cost: FREE — APNs has no quota or pricing.

---

3. Database Schema

3.1 New Tables

`device_tokens`

Stores push notification device tokens/subscriptions.

SQLite:

```
CREATE TABLE IF NOT EXISTS device_tokens (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  platform TEXT NOT NULL CHECK(platform IN ('web','android','ios')),
  token TEXT,                    -- FCM/APNs device token (NULL for Web Push)
  endpoint TEXT,                 -- Web Push endpoint URL (NULL for FCM/APNs)
  p256dh_key TEXT,               -- Web Push p256dh key (NULL for FCM/APNs)
  auth_key TEXT,                 -- Web Push auth key (NULL for FCM/APNs)
  user_agent TEXT,               -- Browser/device info
  created_at TEXT DEFAULT (datetime('now')),
  last_used_at TEXT DEFAULT (datetime('now')),
  active INTEGER DEFAULT 1       -- 0 = deactivated (stale/unregistered)
);

CREATE UNIQUE INDEX IF NOT EXISTS idx_device_tokens_token ON device_tokens(token) WHERE token
IS NOT NULL;
CREATE UNIQUE INDEX IF NOT EXISTS idx_device_tokens_endpoint ON device_tokens(endpoint) WHERE
endpoint IS NOT NULL;
CREATE INDEX IF NOT EXISTS idx_device_tokens_user ON device_tokens(user_id);
CREATE INDEX IF NOT EXISTS idx_device_tokens_platform ON device_tokens(platform);
CREATE INDEX IF NOT EXISTS idx_device_tokens_active ON device_tokens(active);
```

PostgreSQL:

```
CREATE TABLE IF NOT EXISTS device_tokens (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  platform TEXT NOT NULL CHECK(platform IN ('web','android','ios')),
  token TEXT,
  endpoint TEXT,
```

```
  p256dh_key TEXT,
  auth_key TEXT,
  user_agent TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  last_used_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  active INTEGER DEFAULT 1
);

CREATE UNIQUE INDEX IF NOT EXISTS idx_device_tokens_token ON device_tokens(token) WHERE token IS NOT NULL;
CREATE UNIQUE INDEX IF NOT EXISTS idx_device_tokens_endpoint ON device_tokens(endpoint) WHERE endpoint IS NOT NULL;
CREATE INDEX IF NOT EXISTS idx_device_tokens_user ON device_tokens(user_id);
CREATE INDEX IF NOT EXISTS idx_device_tokens_platform ON device_tokens(platform);
CREATE INDEX IF NOT EXISTS idx_device_tokens_active ON device_tokens(active);
```

Deduplication: Same device registering multiple times → UPSERT on `token` or `endpoint` (updates `last_used_at`, reactivates if inactive).

Stale Token Cleanup: Cron job (daily) marks tokens as inactive if `last_used_at` > 90 days OR if delivery fails with "token invalid" error.

`notification_queue`

Queue for deferred/retry delivery (future — MVP sends immediately).

SQLite:

```
CREATE TABLE IF NOT EXISTS notification_queue (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  type TEXT NOT NULL,
  title TEXT NOT NULL,
  body TEXT NOT NULL,
  data TEXT,                    -- JSON string
  priority TEXT DEFAULT 'normal' CHECK(priority IN ('high','normal','low')),
  status TEXT DEFAULT 'pending' CHECK(status IN ('pending','sent','failed','cancelled')),
  attempts INTEGER DEFAULT 0,
  max_attempts INTEGER DEFAULT 3,
  next_retry_at TEXT,           -- ISO timestamp
  created_at TEXT DEFAULT (datetime('now')),
```

```
  sent_at TEXT,
  error TEXT
);

CREATE INDEX IF NOT EXISTS idx_queue_user ON notification_queue(user_id);
CREATE INDEX IF NOT EXISTS idx_queue_status ON notification_queue(status);
CREATE INDEX IF NOT EXISTS idx_queue_next_retry ON notification_queue(next_retry_at) WHERE
status = 'pending';
```

PostgreSQL:

```
CREATE TABLE IF NOT EXISTS notification_queue (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  type TEXT NOT NULL,
  title TEXT NOT NULL,
  body TEXT NOT NULL,
  data TEXT,
  priority TEXT DEFAULT 'normal' CHECK(priority IN ('high','normal','low')),
  status TEXT DEFAULT 'pending' CHECK(status IN ('pending','sent','failed','cancelled')),
  attempts INTEGER DEFAULT 0,
  max_attempts INTEGER DEFAULT 3,
  next_retry_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  sent_at TIMESTAMP,
  error TEXT
);

CREATE INDEX IF NOT EXISTS idx_queue_user ON notification_queue(user_id);
CREATE INDEX IF NOT EXISTS idx_queue_status ON notification_queue(status);
CREATE INDEX IF NOT EXISTS idx_queue_next_retry ON notification_queue(next_retry_at) WHERE
status = 'pending';
```

MVP Note: Queue table created but not used initially. MVP sends notifications synchronously. Post-MVP: add background worker that processes queue with retry logic (exponential backoff).

`notification_log`

Audit log of all push notification deliveries.

SQLite:

```
CREATE TABLE IF NOT EXISTS notification_log (
  id TEXT PRIMARY KEY,
  notification_id TEXT REFERENCES notifications(id),  -- NULL for push-only (no in-app)
  user_id TEXT NOT NULL REFERENCES users(id),
  device_token_id TEXT REFERENCES device_tokens(id),
  platform TEXT NOT NULL CHECK(platform IN ('web','android','ios')),
  type TEXT NOT NULL,
  title TEXT NOT NULL,
  body TEXT NOT NULL,
  status TEXT NOT NULL CHECK(status IN ('sent','failed','skipped')),
  message_id TEXT,               -- Provider message ID (FCM/APNs/Web Push)
  error TEXT,
  sent_at TEXT DEFAULT (datetime('now'))
);

CREATE INDEX IF NOT EXISTS idx_notif_log_user ON notification_log(user_id);
CREATE INDEX IF NOT EXISTS idx_notif_log_status ON notification_log(status);
CREATE INDEX IF NOT EXISTS idx_notif_log_sent_at ON notification_log(sent_at);
CREATE INDEX IF NOT EXISTS idx_notif_log_platform ON notification_log(platform);
```

PostgreSQL:

```
CREATE TABLE IF NOT EXISTS notification_log (
  id TEXT PRIMARY KEY,
  notification_id TEXT REFERENCES notifications(id),
  user_id TEXT NOT NULL REFERENCES users(id),
  device_token_id TEXT REFERENCES device_tokens(id),
  platform TEXT NOT NULL CHECK(platform IN ('web','android','ios')),
  type TEXT NOT NULL,
  title TEXT NOT NULL,
  body TEXT NOT NULL,
  status TEXT NOT NULL CHECK(status IN ('sent','failed','skipped')),
  message_id TEXT,
  error TEXT,
  sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX IF NOT EXISTS idx_notif_log_user ON notification_log(user_id);
CREATE INDEX IF NOT EXISTS idx_notif_log_status ON notification_log(status);
CREATE INDEX IF NOT EXISTS idx_notif_log_sent_at ON notification_log(sent_at);
```

```
CREATE INDEX IF NOT EXISTS idx_notif_log_platform ON notification_log(platform);
```

Retention: Keep indefinitely for audit trail (fintech compliance). Monitoring queries:

```
-- Delivery rate last 24h
SELECT
  platform,
  COUNT(*) as total,
  SUM(CASE WHEN status='sent' THEN 1 ELSE 0 END) as sent,
  SUM(CASE WHEN status='failed' THEN 1 ELSE 0 END) as failed
FROM notification_log
WHERE sent_at > datetime('now', '-1 day')
GROUP BY platform;

-- Failure reasons
SELECT error, COUNT(*) as count
FROM notification_log
WHERE status='failed' AND sent_at > datetime('now', '-7 days')
GROUP BY error
ORDER BY count DESC
LIMIT 10;
```

`notification_preferences`

User preferences for notification types (opt-in/opt-out, quiet hours).

SQLite:

```
CREATE TABLE IF NOT EXISTS notification_preferences (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  category TEXT NOT NULL CHECK(category IN ('transactional','account','promotional')),
  enabled INTEGER DEFAULT 1,
  quiet_hours_start TEXT,        -- HH:MM format (e.g., "22:00")
  quiet_hours_end TEXT,          -- HH:MM format (e.g., "08:00")
  created_at TEXT DEFAULT (datetime('now')),
  updated_at TEXT DEFAULT (datetime('now'))
);

CREATE UNIQUE INDEX IF NOT EXISTS idx_notif_pref_user_cat ON notification_preferences(user_id,
```

```
category);
```

PostgreSQL:

```
CREATE TABLE IF NOT EXISTS notification_preferences (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL REFERENCES users(id),
  category TEXT NOT NULL CHECK(category IN ('transactional','account','promotional')),
  enabled INTEGER DEFAULT 1,
  quiet_hours_start TEXT,
  quiet_hours_end TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE UNIQUE INDEX IF NOT EXISTS idx_notif_pref_user_cat ON notification_preferences(user_id,
category);
```

Default behavior:

- Transactional: ALWAYS enabled (row doesn't exist = enabled, quiet hours ignored)
- Account: Enabled by default (created on first app launch)
- Promotional: Disabled by default (created on first app launch)

Quiet hours logic:

- If notification type = transactional → send immediately (ignore quiet hours)
- Otherwise, check user's local time (use timezone from user profile or default to Europe/Oslo)
- If current time is between `quiet_hours_start` and `quiet_hours_end` → queue for later (send at `quiet_hours_end`)

3.2 Schema Changes to Existing Tables

`notifications` (existing table — ADD columns)

Additions:

```
-- SQLite migration
ALTER TABLE notifications ADD COLUMN notification_type TEXT;
ALTER TABLE notifications ADD COLUMN priority TEXT DEFAULT 'normal' CHECK(priority IN
```

```
('high','normal','low'));
ALTER TABLE notifications ADD COLUMN data TEXT;  -- JSON string with extra context
```

PostgreSQL migration:

```
ALTER TABLE notifications ADD COLUMN notification_type TEXT;
ALTER TABLE notifications ADD COLUMN priority TEXT DEFAULT 'normal';
ALTER TABLE notifications ADD COLUMN data TEXT;

ALTER TABLE notifications ADD CONSTRAINT notifications_priority_check
  CHECK (priority IN ('high','normal','low'));
```

Purpose: Existing `notifications` table stores in-app notifications. New columns allow linking in-app notifications to push notifications (same notification shows in both Notifications screen AND push).

---

4. API Endpoints

4.1 POST `/api/notifications/register-device`

Purpose: Register device token for push notifications.

Auth: Required (bearer token).

Request:

```
{
  "platform": "web",
  "token": "fcm-token-or-apns-token",  // For FCM/APNs
  "subscription": {                     // For Web Push only
    "endpoint": "https://fcm.googleapis.com/...",
    "keys": {
      "p256dh": "base64-encoded-p256dh",
      "auth": "base64-encoded-auth"
    }
  },
  "userAgent": "Mozilla/5.0 ..."
```

```
}
```

Response (200):

```
{
  "data": {
    "deviceId": "dtk_abc123",
    "registered": true
  }
}
```

Errors:

- 400: Missing platform or token/subscription
- 401: Unauthorized
- 409: Device already registered (returns existing deviceId)

Logic:

1. Validate platform ('web', 'android', 'ios')
2. For Web Push: validate subscription object (endpoint, keys.p256dh, keys.auth)
3. For FCM/APNs: validate token format
4. Check if token/endpoint already exists:
   - If exists → update `last_used_at`, set `active=1`, return existing ID
   - If not exists → insert new row
5. Return deviceId

File: `src/app/api/notifications/register-device/route.ts` (NEW)

4.2 DELETE `/api/notifications/devices/:deviceId`

Purpose: Unregister device token (user logs out or revokes permission).

Auth: Required.

Response (200):

```
{
  "data": { "success": true }
}
```

Logic:

1. Verify device belongs to authenticated user
2. Set `active=0` (soft delete — keep for audit trail)

File: `src/app/api/notifications/devices/[deviceId]/route.ts` (NEW)

4.3 GET `/api/notifications/preferences`

Purpose: Get user's notification preferences.

Auth: Required.

Response (200):

```
{
  "data": {
    "transactional": {
      "enabled": true,
      "canDisable": false
    },
    "account": {
      "enabled": true,
      "canDisable": true
    },
    "promotional": {
      "enabled": false,
      "canDisable": true
    },
    "quietHours": {
      "enabled": false,
      "start": null,
      "end": null
    }
  }
}
```

Logic:

1. Fetch rows from `notification_preferences` WHERE user_id = ?
2. If no rows exist → create defaults (transactional=1, account=1, promotional=0)

3. Return preferences

File: `src/app/api/notifications/preferences/route.ts` (NEW, GET handler)

4.4 PUT `/api/notifications/preferences`

Purpose: Update user's notification preferences.

Auth: Required.

Request:

```
{
  "account": { "enabled": true },
  "promotional": { "enabled": false },
  "quietHours": {
    "enabled": true,
    "start": "22:00",
    "end": "08:00"
  }
}
```

Response (200):

```
{
  "data": { "updated": true }
}
```

Errors:

- 400: Invalid category or quiet hours format
- 403: Attempt to disable transactional notifications

Logic:

1. Validate categories (cannot disable transactional)
2. Validate quiet hours format (HH:MM, 00:00-23:59)
3. UPSERT preferences into `notification_preferences` table
4. Update `updated_at = now()`

File: `src/app/api/notifications/preferences/route.ts` (NEW, PUT handler)

4.5 GET `/api/notifications/history`

Purpose: Get user's push notification delivery history (debugging/audit).

Auth: Required.

Query params:

- `limit` (default: 50, max: 100)
- `offset` (default: 0)

Response (200):

```
{
  "data": [
    {
      "id": "nlog_abc123",
      "type": "transfer_received",
      "title": "Du mottok penger",
      "platform": "web",
      "status": "sent",
      "sentAt": "2026-02-17T14:30:00Z"
    }
  ],
  "pagination": {
    "limit": 50,
    "offset": 0,
    "total": 123
  }
}
```

Logic:

1. Query `notification_log` WHERE user_id = ? ORDER BY sent_at DESC LIMIT ? OFFSET ?
2. Count total rows for pagination
3. Return list

File: `src/app/api/notifications/history/route.ts` (NEW)

4.6 GET `/api/vapid-public-key`

Purpose: Expose VAPID public key to client for Web Push subscription.

Auth: Not required (public endpoint).

Response (200):

```
{
  "publicKey": "BN4GvZtEZiZuqaasbD-..."
}
```

File: `src/app/api/vapid-public-key/route.ts` (NEW)

---

5. Integration Points

5.1 Transaction Complete (Remittance + QR Payment)

Files to modify:

- `src/app/api/transactions/remittance/route.ts`
- `src/app/api/transactions/qr-payment/route.ts`

After transaction status = 'completed':

```
import { sendPushNotification } from '@/lib/push';
import { randomId } from '@/lib/utils';

// Create in-app notification (existing table)
const notificationId = randomId('ntf');
await run(
  `INSERT INTO notifications (id, user_id, type, title, body, notification_type, priority,
data)
   VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,
  [
    notificationId,
    userId,
    'transaction_complete',
    'Transaksjon fullført',
    `${amount} ${currency} sendt til ${recipientName}`,
```

```
    'transfer_sent',
    'high',
    JSON.stringify({ transactionId: txId, amount, currency }),
  ]
);

// Send push notification
await sendPushNotification({
  userId: userId,
  type: 'transfer_sent',
  title: 'Transaksjon fullført',
  body: `${amount} ${currency} sendt til ${recipientName}`,
  priority: 'high',
  data: {
    transactionId: txId,
    amount,
    currency,
    url: `/dashboard/transactions/${txId}`,
  },
});

// If recipient is a Drop user, notify them
if (recipientUserId) {
  const recipientNotifId = randomId('ntf');
  await run(
    `INSERT INTO notifications (id, user_id, type, title, body, notification_type, priority,
data)
     VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,
    [
      recipientNotifId,
      recipientUserId,
      'transaction_complete',
      'Du mottok penger',
      `${amount} ${currency} fra ${senderName}`,
      'transfer_received',
      'high',
      JSON.stringify({ transactionId: txId, amount, currency }),
    ]
  );
```

```
  await sendPushNotification({
    userId: recipientUserId,
    type: 'transfer_received',
    title: 'Du mottok penger',
    body: `${amount} ${currency} fra ${senderName}`,
    priority: 'high',
    data: {
      transactionId: txId,
      amount,
      currency,
      url: `/dashboard/transactions/${txId}`,
    },
  });
}
```

5.2 Login Alert (New Device)

File to modify: `src/app/api/auth/login/route.ts`

After successful login:

```
import crypto from 'crypto';
import { sendPushNotification } from '@/lib/push';

const ip = getClientIp(request);
const userAgent = request.headers.get('user-agent') || 'Unknown';

// Generate device fingerprint
const deviceFingerprint = crypto.createHash('sha256')
  .update(`${ip}:${userAgent}`)
  .digest('hex');

// Check if device is new
const existingDevice = await getOne<{ id: string }>(
  "SELECT id FROM sessions WHERE user_id = ? AND device_fingerprint = ?",
  [userId, deviceFingerprint]
);

if (!existingDevice) {
```

```
  // New device → send login alert
  const notificationId = randomId('ntf');
  await run(
    `INSERT INTO notifications (id, user_id, type, title, body, notification_type, priority,
data)
     VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,
    [
      notificationId,
      userId,
      'security',
      'Ny pålogging oppdaget',
      `Pålogging fra ${userAgent.slice(0, 50)}`,
      'login_alert',
      'high',
      JSON.stringify({ ip, userAgent }),
    ]
  );

  await sendPushNotification({
    userId: userId,
    type: 'login_alert',
    title: 'Ny pålogging oppdaget',
    body: `Pålogging fra ${userAgent.slice(0, 50)}`,
    priority: 'high',
    data: {
      ip,
      userAgent,
      url: '/profile/security',
    },
  });
}

// Add device_fingerprint to session insert (schema change needed)
await run(
  `INSERT INTO sessions (id, user_id, token_hash, expires_at, device_fingerprint)
   VALUES (?, ?, ?, ?, ?)`,
  [sessionId, userId, tokenHash, expiresAt, deviceFingerprint]
);
```

Schema change:

```
-- Add to sessions table
ALTER TABLE sessions ADD COLUMN device_fingerprint TEXT;
CREATE INDEX IF NOT EXISTS idx_sessions_device ON sessions(device_fingerprint);
```

5.3 Account Events (KYC, BankID, Password Change)

Files to modify:

- `src/app/api/auth/change-password/route.ts` (if exists, else create)
- `src/app/api/kyc/verify/route.ts` (if exists)
- `src/app/api/bankid/link/route.ts` (if exists)

Pattern (same for all):

```
// After event (e.g., password changed)
await sendPushNotification({
  userId: userId,
  type: 'password_changed',
  title: 'Passord endret',
  body: 'Passordet ditt ble nettopp endret. Hvis dette ikke var deg, kontakt support
umiddelbart.',
  priority: 'high',
  data: {
    timestamp: new Date().toISOString(),
    url: '/profile/security',
  },
});
```

---

6. User Preference Management UI

6.1 Notification Settings Screen

File to create: `src/app/profile/notifications/page.tsx` (MODIFY existing if exists)

UI Components:

1. **Permission Status** — Shows if browser/OS notifications enabled
   - If not enabled → show "Enable Notifications" button → triggers browser permission prompt
2. **Category Toggles:**
   - Transactional: Always ON (disabled toggle, grayed out, "Required for security")
   - Account: Toggle (ON by default)
   - Promotional: Toggle (OFF by default, show consent text)
3. **Quiet Hours:**
   - Toggle "Enable Quiet Hours"
   - Time pickers: Start time (default 22:00), End time (default 08:00)
   - Note: "Transactional notifications (security alerts, payments) will still be sent"
4. **Device List:**
   - Shows registered devices (platform, last used)
   - "Remove" button per device

Client-Side Logic:

```
// Check if push notifications supported
if ('serviceWorker' in navigator && 'PushManager' in window) {
  // Check current permission
  const permission = Notification.permission;
  if (permission === 'default') {
    // Show "Enable Notifications" button
  } else if (permission === 'granted') {
    // Subscribe to push
    const registration = await navigator.serviceWorker.ready;
    const subscription = await registration.pushManager.subscribe({
      userVisibleOnly: true,
      applicationServerKey: urlBase64ToUint8Array(vapidPublicKey),
    });

    // Send to server
    await fetch('/api/notifications/register-device', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        platform: 'web',
        subscription: subscription.toJSON(),
        userAgent: navigator.userAgent,
      }),
    });
  } else {
```

```
    // Permission denied → show "Notifications blocked" message
  }
}
```

Promotional Consent UI:

```
<div>
  <Toggle
    checked={promotionalEnabled}
    onChange={async (enabled) => {
      if (enabled) {
        // Show consent dialog
        const confirmed = await showConsentDialog(
          "Markedsføringsvarsler",
          "Jeg samtykker til å motta tilbud, anbefalinger og produktoppdateringer fra Drop. "
+
          "Jeg kan trekke tilbake samtykket når som helst i innstillinger."
        );
        if (confirmed) {
          await updatePreferences({ promotional: { enabled: true } });
        }
      } else {
        await updatePreferences({ promotional: { enabled: false } });
      }
    }}
  />
  <label>Tilbud og kampanjer</label>
  <p className="text-xs text-gray-500">
    Motta tilbud, anbefalinger og produktoppdateringer (krever samtykke)
  </p>
</div>
```

---

7. Norwegian Marketing Law Compliance

7.1 Markedsføringsloven §15 (Marketing Consent)

Law: Promotional electronic messages (email, SMS, push) require EXPLICIT prior consent from recipient.

Drop Implementation:

- Promotional notifications OFF by default
- User must actively enable in settings
- Consent dialog shows clear purpose ("tilbud, anbefalinger, produktoppdateringer")
- Consent timestamp logged in `notification_preferences.updated_at`
- User can withdraw consent at any time (toggle OFF)

Audit trail:

```
-- Query: Show consent history for user
SELECT
  user_id,
  category,
  enabled,
  created_at,
  updated_at
FROM notification_preferences
WHERE user_id = ? AND category = 'promotional';
```

7.2 GDPR Compliance

Art. 6(1)(a) — Consent as legal basis:

- Consent must be freely given, specific, informed, unambiguous
- Drop: Toggle + consent dialog = unambiguous affirmative action
- Drop: Separate toggle for promotional (not bundled with account/transactional)

Art. 7 — Conditions for consent:

- Burden of proof: controller must demonstrate consent was given
- Drop: `notification_preferences` table logs timestamp and enabled status

Art. 17 — Right to erasure:

- User can delete account → all notification preferences deleted (CASCADE on user_id FK)

8. Security Considerations

8.1 No Sensitive Data in Push Payload

Risk: Push notifications travel through third-party servers (FCM, APNs, Web Push service). Payload can be logged/intercepted.

Mitigation:

- NEVER include: passwords, tokens, full card numbers, bank account numbers
- DO include: generic message ("Du mottok penger") + deep link to app
- Sensitive details fetched AFTER user opens app (authenticated API call)

Example (GOOD):

```
{
  "title": "Ny transaksjon",
  "body": "Du mottok kr 2 500,00",
  "data": {
    "transactionId": "tx_abc123",
    "url": "/dashboard/transactions/tx_abc123"
  }
}
```

Example (BAD):

```
{
  "title": "Ny transaksjon",
  "body": "Du mottok kr 2 500,00 fra John Doe (john@example.com) til konto 1234.56.78901",
  "data": { ... }
}
```

8.2 Token Encryption at Rest

Risk: Device tokens stored in plain text in DB → if DB compromised, attacker can send spam push notifications to all users.

Mitigation (Post-MVP):

- Encrypt `token`, `endpoint`, `p256dh_key`, `auth_key` columns at rest
- Use AES-256-GCM with key stored in env var (`DEVICE_TOKEN_ENCRYPTION_KEY`)
- Decrypt on read, encrypt on write

MVP: Store in plain text (same risk level as session tokens — if DB is compromised, session tokens are also exposed). Add encryption in Phase 2.

8.3 Rate Limiting

Risk: Attacker with stolen session token spams push notifications to user or all users.

Mitigation:

- Per-user rate limit: max 100 push notifications per hour
- Global rate limit: max 10,000 push notifications per hour (protect against abuse)
- Rate limit key: `push:{userId}:{hour}`

Implementation:

```
import { checkRateLimit } from '@/lib/rate-limit';

async function sendPushNotification(payload: NotificationPayload) {
  const hour = new Date().toISOString().slice(0, 13); // "2026-02-17T14"
  const rateLimitKey = `push:${payload.userId}:${hour}`;

  const allowed = await checkRateLimit(rateLimitKey, 100, 3600); // 100 per hour
  if (!allowed) {
    console.warn(`[Push] Rate limit exceeded for user ${payload.userId}`);
    return [{ success: false, error: 'Rate limit exceeded', platform: 'web' }];
  }

  // Send notification...
}
```

---

9. Monitoring & Alerting

9.1 Delivery Metrics

Dashboard queries (daily cron or on-demand):

```
-- Delivery rate by platform (last 24h)
SELECT
  platform,
  COUNT(*) as total,
  ROUND(AVG(CASE WHEN status='sent' THEN 1.0 ELSE 0.0 END) * 100, 2) as success_rate
FROM notification_log
WHERE sent_at > datetime('now', '-1 day')
GROUP BY platform;

-- Opt-out rate by category
SELECT
  category,
  COUNT(*) as total_users,
  SUM(CASE WHEN enabled=0 THEN 1 ELSE 0 END) as opted_out,
  ROUND(AVG(CASE WHEN enabled=0 THEN 1.0 ELSE 0.0 END) * 100, 2) as opt_out_rate
FROM notification_preferences
GROUP BY category;

-- Top failure reasons
SELECT
  error,
  COUNT(*) as count,
  platform
FROM notification_log
WHERE status='failed' AND sent_at > datetime('now', '-7 days')
GROUP BY error, platform
ORDER BY count DESC
LIMIT 10;
```

9.2 Alerts (Slack/Email)

Trigger conditions:

- Delivery rate drops below 90% (hourly check)
- More than 100 failures in 1 hour
- Web Push VAPID keys missing on startup
- FCM/APNs credentials invalid (auth error)

Implementation (future):

```
// In src/lib/alerts.ts (from drop-supporting-systems-plan.md)
await sendSlackAlert({
  severity: 'high',
  message: `Push notification delivery rate dropped to ${rate}% (platform: ${platform})`,
  link: 'http://localhost:3030/monitoring/push',
});
```

---

10. Cost Analysis

Platform	Setup Cost	Operating Cost	Free Tier	Paid Tier
Web Push	0 kr	0 kr	Unlimited (browser-managed)	N/A
FCM (Android)	0 kr	0 kr	Unlimited	N/A
APNs (iOS)	794 kr/year (Apple Developer)	0 kr	Unlimited	N/A

Total Annual Cost: 794 kr (Apple Developer membership only).

Infrastructure cost: Minimal — push notification sending is lightweight (HTTP requests), no message queue or worker needed for MVP.

Post-MVP (if >100k push/day): Consider message queue (BullMQ + Redis, ~200 kr/month on Railway/Fly.io).

---

11. Implementation Plan

Phase 1: Web Push (MVP) — 3 days

Day 1: Backend Infrastructure (6h)

- Create `device_tokens`, `notification_queue`, `notification_log`, `notification_preferences` tables
- Implement `src/lib/push.ts` with Web Push support
- Generate VAPID keys, add to env vars
- Create API endpoints:
  - POST `/api/notifications/register-device`
  - GET `/api/notifications/preferences`
  - PUT `/api/notifications/preferences`

    - GET `/api/vapid-public-key`

Day 2: Frontend Integration (6h)

- Create service worker `public/sw.js` (push event listener)
- Register service worker in `src/app/layout.tsx`
- Build notification settings UI (`src/app/profile/notifications/page.tsx`)
- Implement permission request flow
- Test Web Push subscription registration

Day 3: Integration + Testing (6h)

- Integrate push notifications into transaction routes (remittance, QR payment)
- Integrate login alert into auth/login route
- Add `device_fingerprint` column to `sessions` table
- Test end-to-end flows:
    - Register device → send test notification → receive on device
    - Complete transaction → receive push notification
    - Login from new device → receive security alert
    - Update preferences → verify opt-out works
- Deploy to staging

Total: 18 hours (3 days @ 6h/day)

Phase 2: FCM (Android) — 1 day (FUTURE)

When: React Native Android app ready for testing.

Tasks:

- Set up Firebase project, download service account key
- Implement FCM support in `src/lib/push.ts`
- Add `firebase-admin` dependency
- Test with React Native app

Phase 3: APNs (iOS) — 1 day (FUTURE)

When: React Native iOS app ready for testing.

Tasks:

- Generate APNs Auth Key (.p8 file) from Apple Developer account
- Implement APNs support in `src/lib/push.ts`

- Add `apn` dependency
- Test with React Native app

Phase 4: Advanced Features — 2 days (FUTURE)

Features:

- Background queue + retry logic (BullMQ + Redis)
- Quiet hours enforcement (defer notifications to later)
- Rich notifications (images, actions, reply)
- Device token encryption at rest
- Admin dashboard for monitoring delivery rates

---

12. Testing Strategy

12.1 Unit Tests

Test files to create:

- `tests/unit/push.test.ts` — Test `sendPushNotification()`, `registerDeviceToken()`, preference checks
- `tests/unit/notification-preferences.test.ts` — Test default preferences, opt-in/opt-out logic

Coverage:

- □ Transactional notifications always sent (ignore preferences)
- □ Account notifications respect opt-out
- □ Promotional notifications require opt-in
- □ Quiet hours respected (except transactional)
- □ Rate limiting enforced
- □ Device token deduplication (same token = update, not insert)

12.2 Integration Tests

Test files to create:

- `tests/integration/push-flow.test.ts` — End-to-end push notification flow

Scenarios:

1. **Register device → send notification → verify log entry**
   - POST `/api/notifications/register-device` with Web Push subscription
   - Trigger transaction → verify `notification_log` entry created with status='sent'
2. **Opt-out → verify notification skipped**
   - Update preferences: account notifications OFF
   - Trigger transaction → verify notification NOT sent (status='skipped' in log)
3. **Login alert → new device**
   - Login from new User-Agent → verify login alert notification sent
4. **Promotional consent → verify GDPR compliance**
   - Enable promotional notifications → verify `notification_preferences.updated_at` updated

12.3 Manual Testing Checklist

- ☐ Web Push permission prompt appears on first visit
- ☐ Notification received after granting permission
- ☐ Notification click opens correct URL in app
- ☐ Notification settings UI shows correct state (enabled/disabled per category)
- ☐ Quiet hours toggle works (notifications deferred)
- ☐ Device removal works (device marked inactive)
- ☐ Promotional consent dialog shows before enabling
- ☐ Rate limit enforced (send 101 notifications → last one fails)

---

13. Success Metrics

Week 1 (Post-Deployment)

- ☐ 0 push notification failures (delivery rate 100%)
- ☐ <5% users block notifications after permission prompt
- ☐ 0 GDPR complaints about unsolicited promotional notifications

Month 1

- ☐ >70% users with push notifications enabled

- [ ] >50% notification open rate (click-through from push to app)
- [ ] <10% opt-out rate for account notifications
- [ ] 0 support tickets about missing notifications

Quarter 1

- [ ] Push notifications enabled for all transaction types
- [ ] Login alerts sent for 100% of new device logins
- [ ] Promotional notifications available (consent flow tested)
- [ ] FCM/APNs integration ready for mobile app launch

---

14. Rollout Strategy

Staging (1 week)

- Deploy Web Push to staging environment
- Test with internal users (Alem, team)
- Verify DNS/HTTPS setup (Web Push requires HTTPS)
- Check spam score (should be N/A for push, unlike email)

Production (Gradual)

- **Week 1:** Enable Web Push for NEW users only (flag in `users` table: `push_enabled`)
- **Week 2:** Monitor delivery rate, opt-out rate, open rate
- **Week 3:** Enable for ALL users (remove flag, make default)
- **Week 4:** Enable transactional notifications (transaction receipts, login alerts)
- **Month 2:** Enable account notifications (summaries, low balance)
- **Month 3:** Enable promotional notifications (with consent flow)

---

15. Acceptance Criteria

Push Service Layer:

- [ ] `sendPushNotification()` sends via Web Push in production
- [ ] `sendPushNotification()` logs to console in demo mode

- [ ] `registerDeviceToken()` stores device token in DB
- [ ] Device token deduplication works (UPSERT on endpoint/token)
- [ ] Stale token cleanup marks inactive tokens (>90 days)

Database:

- [ ] All tables created on `initDb()`
- [ ] Indexes exist for performance queries
- [ ] Default preferences created on first app launch

API Endpoints:

- [ ] POST `/api/notifications/register-device` registers Web Push subscription
- [ ] GET `/api/notifications/preferences` returns user preferences
- [ ] PUT `/api/notifications/preferences` updates preferences
- [ ] GET `/api/vapid-public-key` exposes public key

Integration:

- [ ] Transaction completion sends push notification to sender
- [ ] Transfer received sends push notification to recipient (if Drop user)
- [ ] Login from new device sends security alert
- [ ] Promotional notifications require consent dialog

Compliance:

- [ ] Transactional notifications always sent (no opt-out)
- [ ] Promotional notifications OFF by default
- [ ] Consent timestamp logged in DB
- [ ] No sensitive data in push payload

UI:

- [ ] Notification settings screen shows category toggles
- [ ] Quiet hours UI functional
- [ ] Device list shows registered devices
- [ ] Permission prompt appears on first visit

---

16. Dependencies

Add to `package.json`:

```
{
  "dependencies": {
    "web-push": "^3.6.7"
  }
}
```

Future (when mobile apps ship):

```
{
  "dependencies": {
    "firebase-admin": "^12.0.0",
    "apn": "^2.2.0"
  }
}
```

Install:

```
cd ~/ALAI/products/Drop/src/drop-app
npm install web-push
```

---

17. Env Vars

Add to `.env.example`:

```
# --- Push Notifications ---

# Web Push (VAPID keys)
# Generate: npx web-push generate-vapid-keys
VAPID_PUBLIC_KEY=BN...
VAPID_PRIVATE_KEY=...
VAPID_SUBJECT=mailto:support@getdrop.no

# Firebase Cloud Messaging (future - Android)
# FIREBASE_SERVICE_ACCOUNT_KEY=base64-encoded-json
```

```
# Apple Push Notification service (future - iOS)
# APNS_KEY_ID=ABC123XYZ
# APNS_TEAM_ID=DEF456UVW
# APNS_KEY_CONTENT=base64-encoded-p8-file
```

Generate VAPID keys:

```
npx web-push generate-vapid-keys
# Outputs:
# Public Key: BN4GvZtEZiZuqaasbD-...
# Private Key: ...
```

---

18. File List

Files to CREATE:

```
src/lib/push.ts                                    # Push notification service layer
src/app/api/notifications/register-device/route.ts  # Device registration endpoint
src/app/api/notifications/devices/[deviceId]/route.ts # Device unregister endpoint
src/app/api/notifications/preferences/route.ts      # Preferences GET/PUT endpoints
src/app/api/notifications/history/route.ts          # Notification history endpoint
src/app/api/vapid-public-key/route.ts               # VAPID public key endpoint
public/sw.js                                       # Service worker (push event listener)
tests/unit/push.test.ts                            # Unit tests for push service
tests/integration/push-flow.test.ts                # Integration tests
```

Files to MODIFY:

```
src/lib/db.ts                                      # Add device_tokens, notification_queue,
notification_log, notification_preferences tables
src/lib/db.ts                                      # Add notification_type, priority, data
columns to notifications table
src/lib/db.ts                                      # Add device_fingerprint column to
sessions table
src/app/api/transactions/remittance/route.ts       # Add push notification on completion
src/app/api/transactions/qr-payment/route.ts       # Add push notification on completion
src/app/api/auth/login/route.ts                    # Add login alert for new devices
src/app/profile/notifications/page.tsx             # Modify to add preference toggles
```

```
src/app/layout.tsx                          # Register service worker
.env.example                                # Add VAPID_* env vars
package.json                                # Add web-push dependency
src/lib/services/notifications.ts           # DELETE (replaced by src/lib/push.ts)
```

Total: 9 new files, 11 modified files.

END OF SPEC

Revision #3
Created 2026-02-18 08:44:46 UTC by John
Updated 2026-05-24 20:00:47 UTC by John