

Forum:[Hack Forums](#)

Thread: [HF Popularity Tracker Script](#)

Charlie Sheen - April 24, 2026, 8:57 pm

Built this and forgot to share it. It's a popularity tracker for HF. It scans the popularity comments and ratings left on your profile, and stores them. You get a notification when a rating or comment has been changed. It keeps track of monthly gains and losses. Also has the option to filter and search comments rapidly.

To install you must have the Tampermonkey extension. From Tampermonkey click Create New Script, copy and paste the code in the user script window and save.

? loading encrypted image...

```
Code // ==UserScript==
// @name      HF Popularity Tracker
// @namespace  https://hackforums.net/
// @version   1.0.5
// @description Track HackForums popularity/reputation with native-style dropdown, stats, filters,
search, member cards, history, smart notifications, and profile/return-rep actions.
// @match     https://hackforums.net/*
// @grant     GM_notification
// @grant     GM_registerMenuCommand
// @run-at    document-idle
// ==/UserScript==

(function () {
    'use strict';

    /*****
    * CONFIG
    *****/
    const DEBUG = true;

    const AUTO_SCAN_MINUTES = 60;
    const STARTUP_SCAN_MIN_AGE_SECONDS = 180;
    const MAX_CONCURRENT_FETCHES = 2;
    const REQUEST_TIMEOUT_MS = 20000;
    const ERROR_BACKOFF_MINUTES = 10;

    const MAX_LOG_ITEMS = 500;
    const MAX_MEMBER_HISTORY = 200;
    const MAX_DETAIL_TOASTS = 5;
    const MAX_ACK_TOKENS = 2000;

    const ICON_PRIMARY = 'fa fa-ranking-star';
    const ICON_FALLBACK = 'fa fa-star';
    const LAST_UID_KEY = 'hf_pop_tracker_last_uid';
```

```

const DEFAULT_SETTINGS = {
  enableToasts: true,
  notifyNew: true,
  notifyUpdated: true,
  notifyRemoved: false,
  notifyPositive: true,
  notifyNeutral: true,
  notifyNegative: true,
  notifyCommentChangeOnly: false
};

/*****
 * RUNTIME STATE
 *****/
let CURRENT_UID = "";
let REPUTATION_BASE = "";
let STORAGE_KEYS = null;

let toolbarButtonLi = null;
let toolbarButtonAnchor = null;
let toolbarButtonIcon = null;
let toolbarBadge = null;

let dropdownContainer = null;
let dropdownBody = null;
let memberModal = null;

let scanInProgress = false;
let autoLoopStarted = false;

let uiState = {
  filterType: 'all',
  search: "",
  selectedMemberUid: "",
};

/*****
 * HELPERS
 *****/
function log(...args) {
  if (DEBUG) console.log('[HF Pop Tracker]', ...args);
}

function normalizeText(str) {
  return String(str || "")
    .replace(/\u00a0/g, ' ')
    .replace(/\s+/g, ' ')
    .trim();
}

function normalizeTokenText(str) {
  return normalizeText(str).toLowerCase();
}

```

```
function makeToken(parts) {
  return JSON.stringify(parts.map(part => normalizeTokenText(part)));
}
```

```
function escapeHtml(str) {
  return String(str || "")
    .replace(/&/g, '&amp;')
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;')
    .replace(/"/g, '&quot;')
    .replace(/'/g, '&apos;');
}
```

```
function safeParse(value, fallback) {
  if (value == null || value === "") return fallback;
  try {
    const parsed = JSON.parse(value);
    return parsed == null ? fallback : parsed;
  } catch {
    return fallback;
  }
}
```

```
function now() {
  return Date.now();
}
```

```
function timeAgo(ts) {
  if (!ts) return 'never';

  const diff = Math.floor((Date.now() - ts) / 1000);
  const abs = Math.abs(diff);
  const suffix = diff >= 0 ? 'ago' : 'from now';

  if (abs < 60) return `${abs}s ${suffix}`;
  if (abs < 3600) return `${Math.floor(abs / 60)}m ${suffix}`;
  if (abs < 86400) return `${Math.floor(abs / 3600)}h ${suffix}`;
  return `${Math.floor(abs / 86400)}d ${suffix}`;
}
```

```
function monthAgoTimestamp() {
  const d = new Date();
  d.setMonth(d.getMonth() - 1);
  return d.getTime();
}
```

```
function notify(title, text, timeout = 10000) {
  const settings = getSettings();
  if (!settings.enableToasts) return;
  try {
    GM_notification({ title, text, timeout });
  } catch (e) {
  }
}
```

```

        console.warn('[HF Pop Tracker] GM_notification failed:', e);
    }
}

function extractUidFromHref(href) {
    if (!href) return "";
    const m = String(href).match(/(?:&)uid=(\d+)/i);
    return m ? m[1] : "";
}

function extractPageFromHref(href) {
    if (!href) return null;
    const m = String(href).match(/(?:&)page=(\d+)/i);
    return m ? parseInt(m[1], 10) : null;
}

function extractRidFromHref(href) {
    if (!href) return "";
    const m = String(href).match(/(?:&)rid=(\d+)/i);
    return m ? m[1] : "";
}

function parseScoreValue(scoreStr) {
    const m = String(scoreStr || "").match(/(?:\+|-)?\d+/);
    return m ? parseInt(m[0], 10) : 0;
}

function typeColor(type) {
    if (type === 'positive') return '#4caf50';
    if (type === 'negative') return '#ff5f56';
    if (type === 'neutral') return '#d6a63a';
    return '#9e9e9e';
}

function typeIcon(type) {
    if (type === 'positive') return 'fa fa-plus-circle';
    if (type === 'negative') return 'fa fa-minus-circle';
    if (type === 'neutral') return 'fa fa-dot-circle';
    return 'fa fa-circle';
}

function createStorageKeys(uid) {
    return {
        snapshot: `hf_pop_tracker_snapshot_${uid}`,
        meta: `hf_pop_tracker_meta_${uid}`,
        log: `hf_pop_tracker_log_${uid}`,
        auto: `hf_pop_tracker_auto_${uid}`,
        settings: `hf_pop_tracker_settings_${uid}`,
        memberHistory: `hf_pop_tracker_member_history_${uid}`,
        acknowledgedAt: `hf_pop_tracker_acknowledged_at_${uid}`,
        acknowledgedTokens: `hf_pop_tracker_ack_tokens_${uid}`
    };
}

```

```

function findKnownUidFromStorage() {
  const lastUid = localStorage.getItem(LAST_UID_KEY);
  if (lastUid && /\d+$/.test(lastUid)) return lastUid;

  let newestUid = "";
  let newestScan = -1;

  for (let i = 0; i < localStorage.length; i++) {
    const key = localStorage.key(i) || "";
    const m = key.match(/^hf_pop_tracker_meta_(\d+)\$/);
    if (!m) continue;

    const meta = safeParse(localStorage.getItem(key), {});
    const lastScan = Number(meta?.lastScan || 0);
    if (lastScan > newestScan) {
      newestScan = lastScan;
      newestUid = m[1];
    }
  }

  if (newestUid) return newestUid;

  for (let i = 0; i < localStorage.length; i++) {
    const key = localStorage.key(i) || "";
    const m =
key.match(/^hf_pop_tracker_(?:snapshot|log|member_history|ack_tokens|acknowledged_at)_(\d+)\$/);
  );
    if (m) return m[1];
  }

  return "";
}

function getStore(key, fallback) {
  return safeParse(localStorage.getItem(key), fallback);
}

function setStore(key, value) {
  localStorage.setItem(key, JSON.stringify(value));
}

function getMeta() {
  const meta = getStore(STORAGE_KEYS.meta, {
    initialized: false,
    lastScan: 0,
    lastPageCount: 0,
    lastEntryCount: 0,
    uid: CURRENT_UID,
    backoffUntil: 0,
    lastError: "",
    lastSummary: null
  });
}

```

```

return {
  initialized: !!meta.initialized,
  lastScan: Number(meta.lastScan || 0),
  lastPageCount: Number(meta.lastPageCount || 0),
  lastEntryCount: Number(meta.lastEntryCount || 0),
  uid: String(meta.uid || CURRENT_UID),
  backoffUntil: Number(meta.backoffUntil || 0),
  lastError: String(meta.lastError || ''),
  lastSummary: meta.lastSummary || null
};
}

function setMeta(meta) {
  setStore(STORAGE_KEYS.meta, {
    initialized: !!meta.initialized,
    lastScan: Number(meta.lastScan || 0),
    lastPageCount: Number(meta.lastPageCount || 0),
    lastEntryCount: Number(meta.lastEntryCount || 0),
    uid: String(meta.uid || CURRENT_UID),
    backoffUntil: Number(meta.backoffUntil || 0),
    lastError: String(meta.lastError || ''),
    lastSummary: meta.lastSummary || null
  });
}

function getSnapshot() {
  const data = getStore(STORAGE_KEYS.snapshot, {});
  return data && typeof data === 'object' && !Array.isArray(data) ? data : {};
}

function setSnapshot(snapshot) {
  setStore(
    STORAGE_KEYS.snapshot,
    snapshot && typeof snapshot === 'object' && !Array.isArray(snapshot) ? snapshot : {}
  );
}

function getLogItems() {
  const items = getStore(STORAGE_KEYS.log, []);
  return Array.isArray(items) ? items : [];
}

function setLogItems(items) {
  setStore(STORAGE_KEYS.log, Array.isArray(items) ? items : []);
}

function getAcknowledgedAt() {
  return Number(localStorage.getItem(STORAGE_KEYS.acknowledgedAt) || 0);
}

function setAcknowledgedAt(ts) {
  localStorage.setItem(STORAGE_KEYS.acknowledgedAt, String(Number(ts || 0)));
}

```

```

}

function getAcknowledgedTokens() {
  const tokens = getStore(STORAGE_KEYS.acknowledgedTokens, []);
  return Array.isArray(tokens) ? tokens.filter(Boolean).map(String) : [];
}

function getAcknowledgedTokenSet() {
  return new Set(getAcknowledgedTokens());
}

function setAcknowledgedTokens(tokens) {
  const unique = [...new Set((Array.isArray(tokens) ? tokens : []).filter(Boolean).map(String))];
  setStore(STORAGE_KEYS.acknowledgedTokens, unique.slice(-MAX_ACK_TOKENS));
}

function rememberAcknowledgedTokens(tokens) {
  setAcknowledgedTokens([...getAcknowledgedTokens(), ...tokens]);
}

function buildEntryStateToken(entry) {
  const e = sanitizeEntry(entry);
  if (!e) return "";
  return makeToken(['entry-state', e.rid, e.senderUid, e.type, e.score, e.comment]);
}

function buildEntrySeenToken(entry) {
  const e = sanitizeEntry(entry);
  if (!e || !e.rid) return "";
  return makeToken(['entry-seen', e.rid, e.senderUid, e.type, e.score]);
}

function buildAddedChangeToken(item) {
  const e = sanitizeEntry(item);
  if (!e) return "";
  return makeToken(['change-new', e.rid, e.senderUid, e.type, e.score, e.comment]);
}

function buildRemovedChangeToken(item) {
  const e = sanitizeEntry(item);
  if (!e) return "";
  return makeToken(['change-removed', e.rid, e.senderUid, e.type, e.score, e.comment]);
}

function buildUpdatedChangeToken(update) {
  const before = sanitizeEntry(update?.before);
  const after = sanitizeEntry(update?.after);
  if (!before || !after) return "";
  return makeToken([
    'change-updated',
    after.rid,
    after.senderUid,
    before.type,
  ]);
}

```

```

        before.score,
        before.comment,
        after.type,
        after.score,
        after.comment
    ]);
}

function buildLogFallbackToken(item) {
    if (!item || item.type !== 'change') return "";
    return makeToken([
        'log-change',
        item.changeKind || "",
        item.senderUid || "",
        item.repType || "",
        item.repScore || "",
        item.message || ""
    ]);
}

function tokensForAdded(item) {
    return [buildAddedChangeToken(item), buildEntryStateToken(item),
buildEntrySeenToken(item)].filter(Boolean);
}

function tokensForUpdated(update) {
    return [buildUpdatedChangeToken(update),
buildEntryStateToken(update?.after)].filter(Boolean);
}

function tokensForRemoved(item) {
    return [buildRemovedChangeToken(item)].filter(Boolean);
}

function tokensForLogItem(item) {
    const tokens = [];
    if (item?.changeSignature) tokens.push(String(item.changeSignature));
    if (Array.isArray(item?.ackTokens)) tokens.push(...item.ackTokens.map(String));
    const fallback = buildLogFallbackToken(item);
    if (fallback) tokens.push(fallback);
    return tokens.filter(Boolean);
}

function isLogItemAcknowledged(item, acknowledgedTokens = getAcknowledgedTokenSet()) {
    return tokensForLogItem(item).some(token => acknowledgedTokens.has(token));
}

function rememberCurrentAcknowledgementState() {
    const tokens = [];

    for (const item of getLogItems()) {
        if (item?.type === 'change') tokens.push(...tokensForLogItem(item));
    }
}

```

```

    for (const entry of Object.values(getSnapshot())) {
        tokens.push(buildEntryStateToken(entry), buildEntrySeenToken(entry));
    }

    rememberAcknowledgedTokens(tokens);
}

function acknowledgeAll() {
    rememberCurrentAcknowledgementState();
    setAcknowledgedAt(now());
    updateToolbarState(0);
    renderDropdownContent();
}

function getUnacknowledgedChangeCount() {
    if (!STORAGE_KEYS) return 0;

    const acknowledgedAt = getAcknowledgedAt();
    const acknowledgedTokens = getAcknowledgedTokenSet();
    return getLogItems().filter(item =>
        item &&
        item.ts > acknowledgedAt &&
        item.type === 'change' &&
        !isLogItemAcknowledged(item, acknowledgedTokens)
    ).length;
}

function getMemberHistory() {
    const items = getStore(STORAGE_KEYS.memberHistory, {});
    return items && typeof items === 'object' && !Array.isArray(items) ? items : {};
}

function setMemberHistory(history) {
    setStore(STORAGE_KEYS.memberHistory, history && typeof history === 'object' ? history :
{});
}

function getSettings() {
    const saved = getStore(STORAGE_KEYS.settings, {});
    return { ...DEFAULT_SETTINGS, ...(saved && typeof saved === 'object' ? saved : {}) };
}

function setSettings(settings) {
    setStore(STORAGE_KEYS.settings, { ...DEFAULT_SETTINGS, ...(settings || {}) });
}

function pushLog(type, message, extra = {}) {
    const items = getLogItems();
    items.unshift({
        ts: Date.now(),
        type: String(type || 'info'),
        message: String(message || ''),

```

```

    ...extra
  });
  while (items.length > MAX_LOG_ITEMS) items.pop();
  setLogItems(items);
  renderDropdownContent();
}

function isAutoEnabled() {
  return localStorage.getItem(STORAGE_KEYS.auto) !== '0';
}

function setAutoEnabled(enabled) {
  localStorage.setItem(STORAGE_KEYS.auto, enabled ? '1' : '0');
}

/*****
* UID DETECTION
*****/
function detectUidFromDom() {
  const selectors = [
    'a[href*="search.php?action=finduser&uid="]',
    'a[href*="search.php?action=finduserthreads&uid="]',
    'a[href*="marketcp.php?action=profile&uid="]',
    'a[href*="contracts.php?uid="]'
  ];

  for (const selector of selectors) {
    const link = document.querySelector(selector);
    if (link) {
      const uid = extractUidFromHref(link.href);
      if (uid) return uid;
    }
  }

  const knownUid = findKnownUidFromStorage();
  if (knownUid) return knownUid;

  const repLinks = [...document.querySelectorAll('a[href*="reputation.php?uid="]')];
  for (const a of repLinks) {
    const uid = extractUidFromHref(a.href);
    if (uid && /\d+$/i.test(uid)) return uid;
  }

  return "";
}

function ensureUid() {
  const uid = detectUidFromDom();
  if (!uid) throw new Error('Could not auto-detect your HackForums UID from the page.');
```

CURRENT_UID = uid;

REPUTATION_BASE = `\${location.origin}/reputation.php?uid=\${CURRENT\_UID}`;

STORAGE_KEYS = createStorageKeys(CURRENT_UID);

```

    localStorage.setItem(LAST_UID_KEY, CURRENT_UID);
    return uid;
}

```

```

/*****

```

```

* NETWORK

```

```

*****/

```

```

async function fetchUrl(url) {
    const controller = new AbortController();
    const timer = setTimeout(() => controller.abort(), REQUEST_TIMEOUT_MS);

    try {
        const res = await fetch(url, {
            method: 'GET',
            credentials: 'include',
            cache: 'no-store',
            signal: controller.signal
        });

        if (!res.ok) {
            throw new Error(`HTTP ${res.status} for ${url}`);
        }

        return await res.text();
    } finally {
        clearTimeout(timer);
    }
}

```

```

async function fetchPool(urls, concurrency) {
    const results = new Array(urls.length);
    let index = 0;

    async function worker() {
        while (true) {
            const current = index++;
            if (current >= urls.length) return;
            results[current] = await fetchUrl(urls[current]);
        }
    }

    const workers = [];
    const count = Math.min(concurrency, urls.length);
    for (let i = 0; i < count; i++) workers.push(worker());
    await Promise.all(workers);
    return results;
}

```

```

/*****

```

```

* PARSING

```

```

*****/

```

```

function parseDoc(html) {
    const parser = new DOMParser();

```

```

    return parser.parseFromString(String(html || ''), 'text/html');
}

function detectTotalPages(html) {
    const doc = parseDoc(html);
    let maxPage = 1;

    const links = [
        ...doc.querySelectorAll('a[href*="reputation.php?uid="][href*="page="]'),
        ...doc.querySelectorAll('.pagination a[href*="page="]'),
        ...doc.querySelectorAll('.multipage a[href*="page="]'),
        ...doc.querySelectorAll('a[href*="page="]')
    ];

    for (const a of links) {
        const page = extractPageFromHref(a.href);
        if (page && page > maxPage) maxPage = page;

        const txt = normalizeText(a.textContent);
        if (/^\d+$/.test(txt)) {
            const n = parseInt(txt, 10);
            if (n > maxPage) maxPage = n;
        }
    }

    return maxPage;
}

function parseRepTypeFromClass(el) {
    const strong = el?.querySelector?('(strong[class*="reputation_"]');
    const cls = `${String(el?.className || '')} ${String(strong?.className || '')}`;
    if (cls.includes('reputation_positive')) return 'positive';
    if (cls.includes('reputation_negative')) return 'negative';
    if (cls.includes('reputation_neutral')) return 'neutral';
    return 'unknown';
}

function extractRid(el) {
    if (!el) return "";

    const id = String(el.id || "");
    let m = id.match(/^rid[_-]?(\d+)$/i);
    if (m) return m[1];

    const deleteLink = el.querySelector('a[href*="reputation.php?action=delete"][href*="rid="]');
    if (deleteLink) {
        const rid = extractRidFromHref(deleteLink.href);
        if (rid) return rid;
    }

    const anyRepLink = el.querySelector('a[href*="rid="]');
    if (anyRepLink) {
        const rid = extractRidFromHref(anyRepLink.href);
    }
}

```

```

    if (rid) return rid;
  }

  return "";
}

function sanitizeEntry(entry) {
  if (!entry || typeof entry !== 'object') return null;

  return {
    rid: String(entry.rid || ""),
    senderUid: String(entry.senderUid || ""),
    senderName: normalizeText(entry.senderName || ""),
    senderHref: String(entry.senderHref || ""),
    type: String(entry.type || 'unknown'),
    score: String(entry.score || ""),
    comment: normalizeText(entry.comment || ""),
    dateText: normalizeText(entry.dateText || ""),
    rawText: normalizeText(entry.rawText || ""),
    page: Number(entry.page || 1)
  };
}

function buildStableKey(entry) {
  const e = sanitizeEntry(entry);
  if (!e) return "";
  return e.rid ? `rid:${e.rid}` : "";
}

function extractCommentFromRow(row, strongNode) {
  if (!row || !strongNode) return "";

  const clone = row.cloneNode(true);
  clone.querySelectorAll('.smalltext, script, style').forEach(el => el.remove());

  const strongClone = clone.querySelector('strong.reputation_positive,
strong.reputation_negative, strong.reputation_neutral, strong[class*="reputation_"]');
  if (!strongClone) return "";

  const fullText = normalizeText(clone.innerText || clone.textContent || "");
  const strongText = normalizeText(strongClone.textContent || strongNode.textContent || "");
  const idx = fullText.indexOf(strongText);
  if (idx === -1) return "";

  let tail = fullText.slice(idx + strongText.length);
  tail = tail.replace(/^\s*/, "");
  return normalizeText(tail);
}

function parseReputationPage(html, pageNumber) {
  const doc = parseDoc(html);
  const results = {};
  const rows = [...doc.querySelectorAll('td[id^="rid"]')];

```

```

for (const row of rows) {
  const rid = extractRid(row);
  if (!rid) continue;

  const senderLink = row.querySelector('a[href*="member.php?action=profile&uid="],
a[href*="member.php?uid="]');
  const senderUid = senderLink ? extractUidFromHref(senderLink.href) : "";
  const senderName = senderLink ? normalizeText(senderLink.textContent) : "";
  const senderHref = senderLink ? senderLink.href : "";

  const type = parseRepTypeFromClass(row);
  const rawText = normalizeText(row.textContent);

  let dateText = "";
  const smallText = row.querySelector('.smalltext');
  if (smallText) {
    const m = normalizeText(smallText.textContent).match(/Last updated\s+(.+)$/i);
    if (m) dateText = normalizeText(m[1]);
  }

  const strong = row.querySelector('strong.reputation_positive, strong.reputation_negative,
strong.reputation_neutral, strong[class*="reputation_"]');

  let score = "";
  if (strong) {
    const strongText = normalizeText(strong.textContent);
    const scoreMatch = strongText.match(/\(([+-]\d+)\)/);
    if (scoreMatch) score = scoreMatch[1];
  }

  const comment = extractCommentFromRow(row, strong);

  const entry = sanitizeEntry({
    rid,
    senderUid,
    senderName,
    senderHref,
    type,
    score,
    comment,
    dateText,
    rawText,
    page: pageNumber
  });

  const key = buildStableKey(entry);
  if (!key) continue;
  results[key] = entry;
}

return results;
}

```

```

async function scanAllPages(progressCb) {
  const firstHtml = await fetchUrl(REPUTATION_BASE);

  if (/you must be registered|please login|member login/i.test(firstHtml)) {
    throw new Error('Could not read your reputation page. Make sure you are logged in.');
```

```

  }

  const totalPages = detectTotalPages(firstHtml);
  progressCb?.(`Detected ${totalPages} page(s).`);
```

```

  const merged = parseReputationPage(firstHtml, 1);
  progressCb?.(`Parsed page 1/${totalPages}.`);
```

```

  if (totalPages === 1) return { totalPages, entries: merged };

  const urls = [];
  const pageNums = [];
```

```

  for (let page = 2; page <= totalPages; page++) {
    urls.push(`${REPUTATION_BASE}&page=${page}`);
    pageNums.push(page);
  }
```

```

  const htmlPages = await fetchPool(urls, MAX_CONCURRENT_FETCHES);

  for (let i = 0; i < htmlPages.length; i++) {
    const pageNum = pageNums[i];
    const parsed = parseReputationPage(htmlPages[i], pageNum);
    Object.assign(merged, parsed);
    progressCb?.(`Parsed page ${pageNum}/${totalPages}.`);
  }
```

```

  return { totalPages, entries: merged };
}
```

```

/*****
```

```

 * DIFF / HISTORY / STATS
```

```

 *****/
```

```

function diffSnapshots(oldSnap, newSnap) {
  const oldData = oldSnap && typeof oldSnap === 'object' ? oldSnap : {};
  const newData = newSnap && typeof newSnap === 'object' ? newSnap : {};
```

```

  const added = [];
  const removed = [];
  const updated = [];
```

```

  const oldKeys = new Set(Object.keys(oldData));
  const newKeys = new Set(Object.keys(newData));
```

```

  for (const key of newKeys) {
    if (!oldKeys.has(key)) {
      const item = sanitizeEntry(newData[key]);
```

```

        if (item) added.push(item);
    }
}

for (const key of oldKeys) {
    if (!newKeys.has(key)) {
        const item = sanitizeEntry(oldData[key]);
        if (item) removed.push(item);
    }
}

for (const key of newKeys) {
    if (!oldKeys.has(key)) continue;

    const before = sanitizeEntry(oldData[key]);
    const after = sanitizeEntry(newData[key]);
    if (!before || !after) continue;

    const changes = [];
    if ((before.type || "") !== (after.type || "")) changes.push(`type: ${before.type || 'unknown'} ->
${after.type || 'unknown'}`);
    if ((before.score || "") !== (after.score || "")) changes.push(`score: ${before.score || 'none'} ->
${after.score || 'none'}`);
    if ((before.comment || "") !== (after.comment || "")) changes.push('comment changed');

    if (changes.length) {
        updated.push({ before, after, changes });
    }
}

return { added, removed, updated };
}

function isAcknowledgedChange(tokens, acknowledgedTokens) {
    return tokens.some(token => acknowledgedTokens.has(token));
}

function filterAcknowledgedDiff(diff) {
    const acknowledgedTokens = getAcknowledgedTokenSet();

    return {
        added: diff.added.filter(item => !isAcknowledgedChange(tokensForAdded(item),
acknowledgedTokens)),
        removed: diff.removed.filter(item => !isAcknowledgedChange(tokensForRemoved(item),
acknowledgedTokens)),
        updated: diff.updated.filter(item => !isAcknowledgedChange(tokensForUpdated(item),
acknowledgedTokens))
    };
}

function computeDiffDelta(diff) {
    let delta = 0;

```

```

    for (const item of diff.added) delta += parseScoreValue(item.score);
    for (const item of diff.removed) delta -= parseScoreValue(item.score);
    for (const item of diff.updated) {
        delta += parseScoreValue(item.after.score) - parseScoreValue(item.before.score);
    }

    return delta;
}

function appendMemberHistory(diff) {
    const history = getMemberHistory();
    const stamp = now();

    function push(uid, event) {
        if (!uid) return;
        if (!history[uid]) history[uid] = [];
        history[uid].unshift({ ts: stamp, ...event });
        while (history[uid].length > MAX_MEMBER_HISTORY) history[uid].pop();
    }

    for (const item of diff.added) {
        push(item.senderUid, {
            kind: 'new',
            rid: item.rid,
            type: item.type,
            score: item.score,
            comment: item.comment,
            senderName: item.senderName,
            senderHref: item.senderHref
        });
    }

    for (const item of diff.removed) {
        push(item.senderUid, {
            kind: 'removed',
            rid: item.rid,
            type: item.type,
            score: item.score,
            comment: item.comment,
            senderName: item.senderName,
            senderHref: item.senderHref
        });
    }

    for (const item of diff.updated) {
        push(item.after.senderUid, {
            kind: 'updated',
            rid: item.after.rid,
            typeBefore: item.before.type,
            typeAfter: item.after.type,
            scoreBefore: item.before.score,
            scoreAfter: item.after.score,
            commentBefore: item.before.comment,

```

```

        commentAfter: item.after.comment,
        senderName: item.after.senderName,
        senderHref: item.after.senderHref
    });
}

setMemberHistory(history);
}

function computeStats(snapshot) {
    const entries = Object.values(snapshot || {}).map(sanitizeEntry).filter(Boolean);

    let positives = 0;
    let negatives = 0;
    let neutrals = 0;
    let totalRepScore = 0;

    for (const entry of entries) {
        totalRepScore += parseScoreValue(entry.score);
        if (entry.type === 'positive') positives++;
        else if (entry.type === 'negative') negatives++;
        else if (entry.type === 'neutral') neutrals++;
    }

    const logs = getLogItems();
    const cutoff = monthAgoTimestamp();
    let monthlyDelta = 0;

    for (const item of logs) {
        if (!item || !item.ts || item.ts < cutoff) continue;
        if (typeof item.delta === 'number') monthlyDelta += item.delta;
    }

    return {
        totalRepScore,
        positives,
        negatives,
        neutrals,
        monthlyDelta,
        totalEntries: entries.length
    };
}

function buildSummaryFromDiff(diff) {
    const addCount = diff.added.length;
    const removeCount = diff.removed.length;
    const updateCount = diff.updated.length;
    const total = addCount + removeCount + updateCount;

    return {
        total,
        addCount,
        removeCount,

```

```

        updateCount,
        delta: computeDiffDelta(diff),
        text: `${total} change(s): ${addCount} new, ${updateCount} updated, ${removeCount}
removed.`
    };
}

/*****
* NOTIFICATION FILTERS
*****/
function shouldNotifyItem(kind, type, isCommentOnlyChange) {
    const s = getSettings();

    if (kind === 'new' && !s.notifyNew) return false;
    if (kind === 'updated' && !s.notifyUpdated) return false;
    if (kind === 'removed' && !s.notifyRemoved) return false;

    if (s.notifyCommentChangeOnly && kind === 'updated' && !isCommentOnlyChange) return
false;

    if (type === 'positive' && !s.notifyPositive) return false;
    if (type === 'neutral' && !s.notifyNeutral) return false;
    if (type === 'negative' && !s.notifyNegative) return false;

    return true;
}

/*****
* ACTIONS
*****/
function openProfile(uid) {
    if (!uid) return;
    window.open(`${location.origin}/member.php?action=profile&uid=${uid}`, '_blank');
}

function openRepPage(uid) {
    if (!uid) return;
    window.open(`${location.origin}/reputation.php?uid=${uid}`, '_blank');
}

/*****
* UI
*****/
function injectStyles() {
    if (document.getElementById('hf-pop-tracker-native-styles')) return;

    const style = document.createElement('style');
    style.id = 'hf-pop-tracker-native-styles';
    style.textContent = `
        #hf-popularity-toolbar-btn > a {
            position: relative;
            display: inline-flex !important;
            align-items: center;

```

```

    gap: 4px;
}

#hf-popularity-toolbar-btn.active > a {
    color: #7bc2ff !important;
}

#hf-popularity-toolbar-btn .hf-popularity-num {
    position: relative;
    display: none;
    min-width: 16px;
    height: 16px;
    line-height: 16px;
    border-radius: 999px;
    background: #1f8bff;
    color: #fff;
    font-size: 10px;
    font-weight: 700;
    text-align: center;
    padding: 0 4px;
    margin-left: 2px;
    top: -1px;
}

#hf-popularity-container {
    display: none;
    position: absolute;
    width: 520px;
    max-width: calc(100vw - 24px);
    z-index: 99999;
    background: #111;
    border: 1px solid rgba(255,255,255,.06);
    box-shadow: 0 12px 36px rgba(0,0,0,.4);
    overflow: hidden;
}

#hf-popularity-container.show {
    display: block;
}

#hf-popularity-container .notify-header {
    padding: 10px 12px;
    background-color: #454545;
    color: #fff;
    display: flex;
    align-items: center;
    justify-content: space-between;
}

#hf-popularity-container .notify-header-heading {
    font-size: 16px;
    color: #fff;
    font-weight: 600;
}

```

```

}

#hf-popularity-container .hf-pop-head-right {
  display: flex;
  align-items: center;
  gap: 14px;
}

#hf-popularity-container .hf-pop-head-right a {
  color: #fff;
  text-decoration: none;
}

#hf-popularity-notifications-container {
  background: #111;
  max-height: 75vh;
  overflow-y: auto;
}

#hf-popularity-notifications-container .notify-section-header {
  padding: 8px 12px;
  background: #1b1b1b;
  border-top: 1px solid rgba(255,255,255,.04);
  border-bottom: 1px solid rgba(255,255,255,.04);
}

#hf-popularity-notifications-container .notify-section-header .notify-header-heading {
  color: #ddd;
  font-size: 13px;
  font-weight: 600;
}

#hf-popularity-notifications-container .notify-notice-container {
  display: flex;
  gap: 10px;
  padding: 10px 12px;
  border-bottom: 1px solid rgba(255,255,255,.05);
  background: #111;
}

#hf-popularity-notifications-container .notify-notice-container:hover {
  background: #171717;
}

#hf-popularity-notifications-container .notify-notice-avatar-container {
  width: 36px;
  min-width: 36px;
  display: flex;
  align-items: flex-start;
  justify-content: center;
}

#hf-popularity-notifications-container .notify-notice-avatar {

```

```
width: 36px;
height: 36px;
border-radius: 2px;
overflow: hidden;
background: #181818;
display: flex;
align-items: center;
justify-content: center;
}

#hf-popularity-notifications-container .notify-notice-avatar i {
  font-size: 16px;
}

#hf-popularity-notifications-container .notify-notice-details-container {
  width: 100%;
  min-width: 0;
}

#hf-popularity-notifications-container .hf-pop-tinytext {
  color: #888;
  font-size: 11px;
  margin-top: -1px;
}

#hf-popularity-notifications-container .hf-pop-mainline {
  margin-top: 3px;
  color: #ababab;
  line-height: 1.45;
  font-size: 13px;
  word-break: break-word;
}

#hf-popularity-notifications-container .hf-pop-mainline strong {
  color: #ddd;
}

#hf-popularity-notifications-container .hf-pop-controls,
#hf-popularity-notifications-container .hf-pop-filter-row {
  display: flex;
  flex-wrap: wrap;
  gap: 8px;
  margin-top: 8px;
}

#hf-popularity-notifications-container .hf-pop-btn,
#hf-popularity-notifications-container .hf-pop-filter-btn {
  display: inline-flex;
  align-items: center;
  gap: 6px;
  background: #1b1b1b;
  color: #ddd;
  border: 1px solid rgba(255,255,255,.08);
```

```
border-radius: 4px;
padding: 6px 8px;
font-size: 12px;
text-decoration: none;
cursor: pointer;
}

#hf-popularity-notifications-container .hf-pop-btn:hover,
#hf-popularity-notifications-container .hf-pop-filter-btn:hover,
#hf-popularity-notifications-container .hf-pop-filter-btn.active {
background: #252525;
color: #fff;
}

#hf-popularity-notifications-container .hf-pop-overview-grid {
display: grid;
grid-template-columns: repeat(5, 1fr);
gap: 8px;
width: 100%;
}

#hf-popularity-notifications-container .hf-pop-stat-box {
background: #181818;
border: 1px solid rgba(255,255,255,.05);
padding: 8px;
border-radius: 4px;
}

#hf-popularity-notifications-container .hf-pop-stat-label {
font-size: 10px;
color: #888;
text-transform: uppercase;
letter-spacing: .4px;
margin-bottom: 4px;
}

#hf-popularity-notifications-container .hf-pop-stat-value {
font-size: 15px;
color: #f0f0f0;
font-weight: 700;
}

#hf-pop-search {
width: 100%;
box-sizing: border-box;
background: #1b1b1b;
color: #eee;
border: 1px solid rgba(255,255,255,.08);
padding: 8px 10px;
font-size: 12px;
outline: none;
}
```

```

#hf-pop-member-modal {
  display: none;
  position: fixed;
  z-index: 100000;
  inset: 0;
  background: rgba(0,0,0,.6);
}

#hf-pop-member-modal.show {
  display: block;
}

#hf-pop-member-modal .hf-pop-member-card {
  width: 560px;
  max-width: calc(100vw - 24px);
  max-height: calc(100vh - 40px);
  overflow-y: auto;
  background: #111;
  border: 1px solid rgba(255,255,255,.08);
  box-shadow: 0 18px 48px rgba(0,0,0,.5);
  margin: 20px auto;
}

#hf-pop-member-modal .hf-pop-member-head {
  padding: 12px;
  background: #454545;
  color: #fff;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

#hf-pop-member-modal .hf-pop-member-body {
  padding: 12px;
}

#hf-pop-member-modal .hf-pop-member-history-item {
  border-bottom: 1px solid rgba(255,255,255,.05);
  padding: 10px 0;
}

.hf-type-pill {
  display: inline-flex;
  align-items: center;
  gap: 6px;
  padding: 2px 8px;
  border-radius: 999px;
  font-size: 11px;
  font-weight: 700;
  color: #fff;
}

.hf-type-positive { background: #2e7d32; }

```

```

        .hf-type-negative { background: #b71c1c; }
        .hf-type-neutral { background: #8d6e00; }
        .hf-type-unknown { background: #666; }
    `;
    document.head.appendChild(style);
}

function getAvatarHtml(iconClass, color) {
    return `
        <div class="notify-notice-avatar-container">
            <div class="notify-notice-avatar">
                <i class="${escapeHtml(iconClass)}" aria-hidden="true"
style="color:${escapeHtml(color)};"></i>
            </div>
        </div>
    `;
}

function closeDropdown() {
    if (!dropdownContainer) return;
    dropdownContainer.classList.remove('show');
    toolbarButtonLi?.classList.remove('active');
}

function openDropdown() {
    createDropdown();
    dropdownContainer.classList.add('show');
    positionDropdown();
    toolbarButtonLi?.classList.add('active');
    renderDropdownContent();
}

function toggleDropdown() {
    if (!dropdownContainer || !dropdownContainer.classList.contains('show')) openDropdown();
    else closeDropdown();
}

function positionDropdown() {
    if (!toolbarButtonAnchor || !dropdownContainer) return;

    const rect = toolbarButtonAnchor.getBoundingClientRect();
    const top = rect.bottom + window.scrollY + 8;
    let left = rect.left + window.scrollX - 20;

    const maxLeft = window.scrollX + window.innerWidth - dropdownContainer.offsetWidth - 12;
    if (left > maxLeft) left = Math.max(window.scrollX + 12, maxLeft);
    if (left < window.scrollX + 12) left = window.scrollX + 12;

    dropdownContainer.style.top = `${top}px`;
    dropdownContainer.style.left = `${left}px`;
}

function createMemberModal() {

```

```

if (memberModal) return;

memberModal = document.createElement('div');
memberModal.id = 'hf-pop-member-modal';
memberModal.innerHTML = `
  <div class="hf-pop-member-card">
    <div class="hf-pop-member-head">
      <span id="hf-pop-member-title">Member Summary</span>
      <a href="javascript:void(0)" id="hf-pop-member-close" style="color:#fff;"><i class="fa
fa-times-circle fa-lg"></i></a>
    </div>
    <div class="hf-pop-member-body" id="hf-pop-member-body"></div>
  </div>
`;
document.body.appendChild(memberModal);

memberModal.addEventListener('click', (e) => {
  if (e.target === memberModal) memberModal.classList.remove('show');
});

memberModal.querySelector('#hf-pop-member-close').addEventListener('click', (e) => {
  e.preventDefault();
  memberModal.classList.remove('show');
});
}

function openMemberSummary(uid) {
  if (!uid) return;
  createMemberModal();

  const snapshot = getSnapshot();
  const history = getMemberHistory();
  const currentEntries = Object.values(snapshot).map(sanitizeEntry).filter(x => x && x.senderUid
=== uid);
  const memberHist = history[uid] || [];
  const name = currentEntries[0]?.senderName || memberHist[0]?.senderName || `UID ${uid}`;

  const totalCurrent = currentEntries.reduce((sum, x) => sum + parseScoreValue(x.score), 0);

  const currentHtml = currentEntries.length
    ? currentEntries.map(entry => `
      <div class="hf-pop-member-history-item">
        <div class="hf-pop-tinytext">Current entry - RID ${escapeHtml(entry.rid)}</div>
        <div class="hf-pop-mainline">
          <span class="hf-type-pill
hf-type-${escapeHtml(entry.type)}">${escapeHtml(entry.type)} ${escapeHtml(entry.score)}</span>
          <div style="margin-top:6px;">${escapeHtml(entry.comment || '(no comment)')}</div>
        </div>
      </div>
    `).join("")
    : `<div class="hf-pop-member-history-item"><div class="hf-pop-mainline">No current entries
for this member.</div></div>`;

```

```

const historyHtml = memberHist.length
? memberHist.map(item => `
  <div class="hf-pop-member-history-item">
    <div class="hf-pop-tinytext">${escapeHtml(new Date(item.ts).toLocaleString())} -
    ${escapeHtml(item.kind)}</div>
    <div class="hf-pop-mainline">
      ${
        item.kind === 'updated'
        ? `
          <div><strong>Score:</strong> ${escapeHtml(item.scoreBefore || '0')} ->
          ${escapeHtml(item.scoreAfter || '0')}</div>
          <div><strong>Comment:</strong> "${escapeHtml(item.commentBefore || '')}"
          -> "${escapeHtml(item.commentAfter || '')}"</div>
        `
        : `
          <div><strong>Type:</strong> ${escapeHtml(item.type || '')}
          ${escapeHtml(item.score || '')}</div>
          <div><strong>Comment:</strong> ${escapeHtml(item.comment || '(no
comment)')}</div>
        `
      }
    </div>
  </div>
`
).join("")
: `<div class="hf-pop-member-history-item"><div class="hf-pop-mainline">No recorded
history yet.</div></div>`;

```

```

memberModal.querySelector('#hf-pop-member-title').textContent = `Member Summary -
${name}`;

```

```

memberModal.querySelector('#hf-pop-member-body').innerHTML = `
  <div class="hf-pop-controls" style="margin-bottom:12px;">
    <a href="javascript:void(0)" class="hf-pop-btn" id="hf-pop-member-open-profile"><i
class="fa fa-user"></i><span>Open Profile</span></a>
    <a href="javascript:void(0)" class="hf-pop-btn" id="hf-pop-member-return-rep"><i
class="fa fa-star"></i><span>Return Rep</span></a>
  </div>

```

```

  <div class="hf-pop-overview-grid" style="margin-bottom:12px;">
    <div class="hf-pop-stat-box">
      <div class="hf-pop-stat-label">Member UID</div>
      <div class="hf-pop-stat-value">${escapeHtml(uid)}</div>
    </div>
    <div class="hf-pop-stat-box">
      <div class="hf-pop-stat-label">Current Entries</div>
      <div class="hf-pop-stat-value">${currentEntries.length}</div>
    </div>
    <div class="hf-pop-stat-box">
      <div class="hf-pop-stat-label">Current Score Sum</div>
      <div class="hf-pop-stat-value">${totalCurrent >= 0 ? '+' : ''}${totalCurrent}</div>
    </div>
    <div class="hf-pop-stat-box">
      <div class="hf-pop-stat-label">History Events</div>
      <div class="hf-pop-stat-value">${memberHist.length}</div>
    </div>
  </div>

```

```

    </div>
    <div class="hf-pop-stat-box">
        <div class="hf-pop-stat-label">Name</div>
        <div class="hf-pop-stat-value" style="font-size:12px;">${escapeHtml(name)}</div>
    </div>
</div>

```

```

    <div class="notify-section-header"><span class="notify-header-heading">Current
Entries</span></div>
    ${currentHtml}

```

```

    <div class="notify-section-header" style="margin-top:12px;"><span
class="notify-header-heading">History</span></div>
    ${historyHtml}
`;

```

```

memberModal.querySelector('#hf-pop-member-open-profile').addEventListener('click', (e) => {
    e.preventDefault();
    openProfile(uid);
});

```

```

memberModal.querySelector('#hf-pop-member-return-rep').addEventListener('click', (e) => {
    e.preventDefault();
    openRepPage(uid);
});

```

```

memberModal.classList.add('show');
}

```

```

function createDropdown() {
    if (dropdownContainer) return;

```

```

    dropdownContainer = document.createElement('div');
    dropdownContainer.id = 'hf-popularity-container';
    dropdownContainer.innerHTML = `
        <div class="notify-header">
            <span class="notify-header-heading">Popularity Tracker</span>
            <div class="hf-pop-head-right">
                <a href="javascript:void(0);" id="hf-pop-header-scan" title="Scan now"><i class="fa
fa-sync-alt fa-lg"></i></a>
                <a href="javascript:void(0);" id="hf-pop-header-toggle-auto" title="Toggle hourly auto
scan"><i class="fa fa-bolt fa-lg"></i></a>
                <a href="javascript:void(0);" id="hf-pop-header-reset" title="Reset baseline"><i
class="fa fa-undo fa-lg"></i></a>
                <a href="javascript:void(0);" id="hf-pop-header-close" title="Close"><i class="fa
fa-times-circle fa-lg"></i></a>
            </div>
        </div>
        <div id="hf-popularity-notifications-container"></div>
    `;

```

```

document.body.appendChild(dropdownContainer);
dropdownBody = dropdownContainer.querySelector('#hf-popularity-notifications-container');

```

```

dropdownContainer.querySelector('#hf-pop-header-scan').addEventListener('click', (e) => {
  e.preventDefault();
  runScan(true, true);
});

=> {
  dropdownContainer.querySelector('#hf-pop-header-toggle-auto').addEventListener('click', (e) => {
    e.preventDefault();
    setAutoEnabled(!isAutoEnabled());
    pushLog('info', `Auto scan turned ${isAutoEnabled() ? 'on' : 'off'}.`);
    updateToolbarState();
    renderDropdownContent();
  });

  dropdownContainer.querySelector('#hf-pop-header-reset').addEventListener('click', (e) => {
    e.preventDefault();
    resetBaseline();
  });

  dropdownContainer.querySelector('#hf-pop-header-close').addEventListener('click', (e) => {
    e.preventDefault();
    closeDropdown();
  });

  document.addEventListener('click', (e) => {
    if (!dropdownContainer || !toolbarButtonLi) return;
    if (!dropdownContainer.classList.contains('show')) return;

    const insideDropdown = dropdownContainer.contains(e.target);
    const insideButton = toolbarButtonLi.contains(e.target);
    const insideModal = memberModal?.contains(e.target);

    if (!insideDropdown && !insideButton && !insideModal) closeDropdown();
  });

  window.addEventListener('resize', () => {
    if (dropdownContainer?.classList.contains('show')) positionDropdown();
  });

  window.addEventListener('scroll', () => {
    if (dropdownContainer?.classList.contains('show')) positionDropdown();
  }, true);
}

function renderOverviewBlock() {
  const meta = getMeta();
  const stats = computeStats(getSnapshot());

  const backoffLine = meta.backoffUntil && now() < meta.backoffUntil
    ? `  
<strong>Paused:</strong> retrying in ${escapeHtml(timeAgo(meta.backoffUntil))}`
    : "";

```

```

const monthly = stats.monthlyDelta >= 0 ? `+${stats.monthlyDelta}` : `${stats.monthlyDelta}`;

return `
  <div class="notify-section-header">
    <span class="notify-header-heading">Scoreboard</span>
  </div>

  <div class="notify-notice-container">
    ${getAvatarHtml('fa fa-chart-line', '#4ea1ff')}
    <div class="notify-notice-details-container">
      <div class="hf-pop-overview-grid">
        <div class="hf-pop-stat-box">
          <div class="hf-pop-stat-label">Total Rep Score</div>
          <div class="hf-pop-stat-value">${stats.totalRepScore >= 0 ? '+' :
"}${stats.totalRepScore}</div>
        </div>
        <div class="hf-pop-stat-box">
          <div class="hf-pop-stat-label">Positives</div>
          <div class="hf-pop-stat-value" style="color:#4caf50;">${stats.positives}</div>
        </div>
        <div class="hf-pop-stat-box">
          <div class="hf-pop-stat-label">Negatives</div>
          <div class="hf-pop-stat-value" style="color:#ff5f56;">${stats.negatives}</div>
        </div>
        <div class="hf-pop-stat-box">
          <div class="hf-pop-stat-label">Neutrals</div>
          <div class="hf-pop-stat-value" style="color:#d6a63a;">${stats.neutrals}</div>
        </div>
        <div class="hf-pop-stat-box">
          <div class="hf-pop-stat-label">Monthly +/-</div>
          <div class="hf-pop-stat-value" style="color:${stats.monthlyDelta >= 0 ? '#4caf50' :
'#ff5f56'};">${monthly}</div>
        </div>
      </div>

      <div class="hf-pop-mainline" style="margin-top:10px;">
        <strong>UID:</strong> ${escapeHtml(CURRENT_UID || '?')}<br>
        <strong>Pages:</strong> ${meta.lastPageCount || 0}<br>
        <strong>Last Scan:</strong> ${escapeHtml(timeAgo(meta.lastScan || 0))}<br>
        <strong>Acknowledged:</strong>
        ${escapeHtml(timeAgo(getAcknowledgedAt()))}<br>
        <strong>Baseline:</strong> ${meta.initialized ? 'Ready' : 'Not initialized'}<br>
        <strong>Hourly auto scan:</strong> ${isAutoEnabled() ? 'On' : 'Off'}<br>
        <strong>Status:</strong> ${scanInProgress ? 'Scanning...' : 'Idle'}${backoffLine}
      </div>

      <div class="hf-pop-controls">
        <a href="javascript:void(0)" class="hf-pop-btn" id="hf-pop-btn-scan"><i class="fa
fa-sync-alt"></i><span>Scan Now</span></a>
        <a href="javascript:void(0)" class="hf-pop-btn" id="hf-pop-btn-auto"><i class="fa
fa-bolt"></i><span>Hourly: ${isAutoEnabled() ? 'On' : 'Off'}</span></a>
        <a href="javascript:void(0)" class="hf-pop-btn" id="hf-pop-btn-ack"><i class="fa
fa-check-circle"></i><span>Acknowledge All</span></a>

```

```

        <a href="javascript:void(0)" class="hf-pop-btn" id="hf-pop-btn-reset"><i class="fa
fa-undo"></i><span>Reset Baseline</span></a>
    </div>
</div>
</div>
};
}

function renderFiltersBlock() {
    const settings = getSettings();
    const filters = [
        ['all', 'All'],
        ['new', 'New'],
        ['updated', 'Updated'],
        ['removed', 'Removed'],
        ['positive', 'Positive'],
        ['neutral', 'Neutral'],
        ['negative', 'Negative']
    ];

    return `
        <div class="notify-section-header">
            <span class="notify-header-heading">Filters & Notifications</span>
        </div>

        <div class="notify-notice-container">
            ${getAvatarHtml('fa fa-filter', '#9c7cff')}
            <div class="notify-notice-details-container">
                <input id="hf-pop-search" type="text" placeholder="Search by name or comment..."
value="${escapeHtml(uiState.search)}">

                <div class="hf-pop-filter-row">
                    ${filters.map(([key, label]) => `
                        <a href="javascript:void(0)" class="hf-pop-filter-btn ${uiState.filterType === key ?
'active' : ''}" data-filter="${escapeHtml(key)}">${escapeHtml(label)}</a>
                    `).join("")}
                </div>

                <div class="hf-pop-controls" style="margin-top:10px;">
                    <a href="javascript:void(0)" class="hf-pop-btn hf-pop-setting-toggle"
data-key="enableToasts"><i class="fa fa-bell"></i><span>Toasts: ${settings.enableToasts ? 'On' :
'Off'}</span></a>
                    <a href="javascript:void(0)" class="hf-pop-btn hf-pop-setting-toggle"
data-key="notifyNew"><span>New: ${settings.notifyNew ? 'On' : 'Off'}</span></a>
                    <a href="javascript:void(0)" class="hf-pop-btn hf-pop-setting-toggle"
data-key="notifyUpdated"><span>Updated: ${settings.notifyUpdated ? 'On' : 'Off'}</span></a>
                    <a href="javascript:void(0)" class="hf-pop-btn hf-pop-setting-toggle"
data-key="notifyRemoved"><span>Removed: ${settings.notifyRemoved ? 'On' : 'Off'}</span></a>
                    <a href="javascript:void(0)" class="hf-pop-btn hf-pop-setting-toggle"
data-key="notifyPositive"><span>+ : ${settings.notifyPositive ? 'On' : 'Off'}</span></a>
                    <a href="javascript:void(0)" class="hf-pop-btn hf-pop-setting-toggle"
data-key="notifyNeutral"><span>0 : ${settings.notifyNeutral ? 'On' : 'Off'}</span></a>
                    <a href="javascript:void(0)" class="hf-pop-btn hf-pop-setting-toggle"

```

```

data-key="notifyNegative"><span>- : ${settings.notifyNegative ? 'On' : 'Off'}</span></a>
    <a href="javascript:void(0)" class="hf-pop-btn hf-pop-setting-toggle"
data-key="notifyCommentChangeOnly"><span>Comment-only updates:
${settings.notifyCommentChangeOnly ? 'On' : 'Off'}</span></a>
    </div>
</div>
</div>
};
}

function iconForLogType(type) {
  switch (type) {
    case 'change': return { icon: 'fa fa-star', color: '#ffcc4d' };
    case 'changes': return { icon: 'fa fa-bell', color: '#4ea1ff' };
    case 'baseline': return { icon: 'fa fa-database', color: '#8bc34a' };
    case 'error': return { icon: 'fa fa-exclamation-triangle', color: '#ff5f56' };
    case 'scan': return { icon: 'fa fa-sync-alt', color: '#9c7cff' };
    default: return { icon: 'fa fa-info-circle', color: '#9e9e9e' };
  }
}

function filterLogs(logs) {
  const term = normalizeText(uiState.search).toLowerCase();
  const acknowledgedAt = getAcknowledgedAt();
  const acknowledgedTokens = getAcknowledgedTokenSet();

  return logs.filter(item => {
    const msg = normalizeText(item.message).toLowerCase();
    const name = normalizeText(item.senderName || "").toLowerCase();
    const type = normalizeText(item.repType || "").toLowerCase();

    if (!term && item.type === 'change' && (item.ts <= acknowledgedAt ||
isLogItemAcknowledged(item, acknowledgedTokens))) return false;
    if (!term && item.type !== 'change' && item.ts <= acknowledgedAt) return false;
    if (term && !msg.includes(term) && !name.includes(term)) return false;

    const f = uiState.filterType;
    if (f === 'all') return true;
    if (f === 'new') return item.changeKind === 'new';
    if (f === 'updated') return item.changeKind === 'updated';
    if (f === 'removed') return item.changeKind === 'removed';
    if (f === 'positive') return type === 'positive';
    if (f === 'neutral') return type === 'neutral';
    if (f === 'negative') return type === 'negative';

    return true;
  });
}

function renderLogControls(item) {
  const uid = item.senderUid || "";
  return `
    <div class="hf-pop-controls">

```

```

        ${uid ? `<a href="javascript:void(0)" class="hf-pop-btn hf-pop-open-profile"
data-uid="${escapeHtml(uid)}"><i class="fa fa-user"></i><span>Profile</span></a>` : ""}
        ${uid ? `<a href="javascript:void(0)" class="hf-pop-btn hf-pop-return-rep"
data-uid="${escapeHtml(uid)}"><i class="fa fa-star"></i><span>Return Rep</span></a>` : ""}
        ${uid ? `<a href="javascript:void(0)" class="hf-pop-btn hf-pop-member-summary"
data-uid="${escapeHtml(uid)}"><i class="fa fa-id-card"></i><span>Summary</span></a>` : ""}
    </div>
    `;
}

function renderTypePill(type, score) {
    const cls = `hf-type-${escapeHtml(type || 'unknown')}`;
    return `<span class="hf-type-pill ${cls}"><i
class="${escapeHtml(typeIcon(type))}"></i>${escapeHtml(type || 'unknown')} ${escapeHtml(score ||
"")}</span>`;
}

function renderActivityBlock() {
    const logs = filterLogs(getLogItems());

    if (!logs.length) {
        return `
        <div class="notify-section-header">
            <span class="notify-header-heading">Recent Activity</span>
        </div>
        <div class="notify-notice-container">
            ${getAvatarHtml('fa fa-inbox', '#9e9e9e')}
            <div class="notify-notice-details-container">
                <div class="hf-pop-mainline">No matching entries.</div>
            </div>
        </div>
        `;
    }

    return `
    <div class="notify-section-header">
        <span class="notify-header-heading">Recent Activity</span>
    </div>
    ${logs.map(item => {
        const visual = item.repType ? { icon: typeIcon(item.repType), color:
typeColor(item.repType) } : iconForLogType(item.type);
        return `
        <div class="notify-notice-container">
            ${getAvatarHtml(visual.icon, visual.color)}
            <div class="notify-notice-details-container">
                <div class="hf-pop-tinytext">${escapeHtml(new
Date(item.ts).toLocaleString())}</div>
                <div class="hf-pop-mainline">
                    ${item.repType ? renderTypePill(item.repType, item.repScore) : ""}
                    <div style="margin-top:${item.repType ? '6px' :
'0'};">${escapeHtml(item.message)}</div>
                </div>
                ${renderLogControls(item)}
            </div>
        </div>
        `;
    })}
    `;
}

```

```

        </div>
      </div>
    `;
  }).join("")
  `;
}

```

```

function bindControls() {
  dropdownBody.querySelector('#hf-pop-btn-scan')?.addEventListener('click', (e) => {
    e.preventDefault();
    runScan(true, true);
  });

  dropdownBody.querySelector('#hf-pop-btn-auto')?.addEventListener('click', (e) => {
    e.preventDefault();
    setAutoEnabled(!isAutoEnabled());
    pushLog('info', `Auto scan turned ${isAutoEnabled() ? 'on' : 'off'}.`);
    renderDropdownContent();
  });

  dropdownBody.querySelector('#hf-pop-btn-ack')?.addEventListener('click', (e) => {
    e.preventDefault();
    acknowledgeAll();
  });

  dropdownBody.querySelector('#hf-pop-btn-reset')?.addEventListener('click', (e) => {
    e.preventDefault();
    resetBaseline();
  });

  dropdownBody.querySelector('#hf-pop-search')?.addEventListener('input', (e) => {
    uiState.search = e.target.value || "";
    renderDropdownContent();
  });

  dropdownBody.querySelectorAll('.hf-pop-filter-btn').forEach(btn => {
    btn.addEventListener('click', (e) => {
      e.preventDefault();
      uiState.filterType = btn.getAttribute('data-filter') || 'all';
      renderDropdownContent();
    });
  });

  dropdownBody.querySelectorAll('.hf-pop-setting-toggle').forEach(btn => {
    btn.addEventListener('click', (e) => {
      e.preventDefault();
      const key = btn.getAttribute('data-key');
      const settings = getSettings();
      settings[key] = !settings[key];
      setSettings(settings);
      renderDropdownContent();
    });
  });
}

```

```

dropdownBody.querySelectorAll('.hf-pop-open-profile').forEach(btn => {
  btn.addEventListener('click', (e) => {
    e.preventDefault();
    openProfile(btn.getAttribute('data-uid') || "");
  });
});

dropdownBody.querySelectorAll('.hf-pop-return-rep').forEach(btn => {
  btn.addEventListener('click', (e) => {
    e.preventDefault();
    openRepPage(btn.getAttribute('data-uid') || "");
  });
});

dropdownBody.querySelectorAll('.hf-pop-member-summary').forEach(btn => {
  btn.addEventListener('click', (e) => {
    e.preventDefault();
    openMemberSummary(btn.getAttribute('data-uid') || "");
  });
});
}

```

```

function renderDropdownContent() {
  if (!dropdownBody || !STORAGE_KEYS) return;

  dropdownBody.innerHTML = `
    ${renderOverviewBlock()}
    ${renderFiltersBlock()}
    ${renderActivityBlock()}
  `;

  bindControls();
}

```

```

function buildToolbarUI() {
  injectStyles();
  createDropdown();
  createMemberModal();

  const host =
    document.querySelector('.lower.panel-nav-lower .menu.panel_links') ||
    document.querySelector('ul.menu.panel_links') ||
    document.querySelector('ul.menu.user_links');

  if (!host) throw new Error('Could not find the HF toolbar host.');
```

```

  let li = document.getElementById('hf-popularity-toolbar-btn');
  if (!li) {
    li = document.createElement('li');
    li.id = 'hf-popularity-toolbar-btn';
    li.className = 'button-dropdown';
    li.innerHTML = `

```

```

        <a href="javascript:void(0)" class="dropdown-toggle" data-tooltip="Popularity Tracker">
            <i class="${ICON_PRIMARY}" id="hf-pop-toolbar-icon" aria-hidden="true"></i>
            <span>?</span>
            <span class="hf-popularity-num">0</span>
        </a>
    `;
    host.appendChild(li);
}

toolbarButtonLi = li;
toolbarButtonAnchor = li.querySelector('a');
toolbarButtonIcon = li.querySelector('#hf-pop-toolbar-icon');
toolbarBadge = li.querySelector('.hf-popularity-num');

toolbarButtonAnchor.addEventListener('click', (e) => {
    e.preventDefault();
    toggleDropdown();
});

setTimeout(() => {
    if (toolbarButtonIcon && toolbarButtonIcon.offsetWidth === 0) {
        toolbarButtonIcon.className = ICON_FALLBACK;
    }
}, 1000);

updateToolbarState();
}

function updateToolbarState(changeCount = getUnacknowledgedChangeCount()) {
    if (toolbarBadge) {
        toolbarBadge.textContent = String(changeCount);
        toolbarBadge.style.display = changeCount > 0 ? 'inline-flex' : 'none';
    }

    if (toolbarButtonLi) {
        const active = scanInProgress || dropdownContainer?.classList.contains('show');
        toolbarButtonLi.classList.toggle('active', !!active);
    }
}

/*****
* CHANGE REPORTING
*****/

function formatAddedItem(item) {
    return `${item.senderName || 'Unknown'} left ${item.type}${item.score ? ` (${item.score})` : ''} :
    "${item.comment ? ` - "${item.comment}"` : ""}`;
}

function formatRemovedItem(item) {
    return `${item.senderName || 'Unknown'}'s popularity entry disappeared`;
}

function formatUpdatedItem(update) {

```

```

const after = update.after;
const before = update.before;
let detail = `${after.senderName || 'Unknown'} updated popularity: ${update.changes.join(', ')}`;

if ((before.comment || '') !== (after.comment || '')) {
  detail += ` | old: "${before.comment || ''}" | new: "${after.comment || ''}"`;
}

return detail;
}

function writeDiffToLog(diff, summary) {
  pushLog('changes', summary.text, { delta: summary.delta });

  for (const item of diff.added) {
    const ackTokens = tokensForAdded(item);
    pushLog('change', `New popularity: ${formatAddedItem(item)}`, {
      senderUid: item.senderUid,
      senderName: item.senderName,
      repType: item.type,
      repScore: item.score,
      changeKind: 'new',
      changeSignature: ackTokens[0] || "",
      ackTokens,
      delta: parseScoreValue(item.score)
    });
  }

  for (const item of diff.updated) {
    const before = parseScoreValue(item.before.score);
    const after = parseScoreValue(item.after.score);
    const ackTokens = tokensForUpdated(item);
    pushLog('change', `Popularity updated: ${formatUpdatedItem(item)}`, {
      senderUid: item.after.senderUid,
      senderName: item.after.senderName,
      repType: item.after.type,
      repScore: item.after.score,
      changeKind: 'updated',
      changeSignature: ackTokens[0] || "",
      ackTokens,
      delta: after - before
    });
  }

  for (const item of diff.removed) {
    const ackTokens = tokensForRemoved(item);
    pushLog('change', `Popularity removed: ${formatRemovedItem(item)}`, {
      senderUid: item.senderUid,
      senderName: item.senderName,
      repType: item.type,
      repScore: item.score,
      changeKind: 'removed',
      changeSignature: ackTokens[0] || "",

```

```

        ackTokens,
        delta: -parseScoreValue(item.score)
    });
}
}

function sendGroupedToasts(diff, summary) {
    const settings = getSettings();
    if (!settings.enableToasts || summary.total === 0) return;

    const anyNotify =
        diff.added.some(x => shouldNotifyItem('new', x.type, false)) ||
        diff.updated.some(x => shouldNotifyItem('updated', x.after.type, x.before.comment !==
x.after.comment && x.before.score === x.after.score && x.before.type === x.after.type)) ||
        diff.removed.some(x => shouldNotifyItem('removed', x.type, false));

    if (!anyNotify) return;

    notify('HF Popularity Tracker', summary.text, 12000);

    const details = [];

    for (const item of diff.added) {
        if (shouldNotifyItem('new', item.type, false)) {
            details.push({ title: 'New popularity', text: formatAddedItem(item) });
        }
    }

    for (const item of diff.updated) {
        const commentOnly = item.before.comment !== item.after.comment &&
            item.before.score === item.after.score &&
            item.before.type === item.after.type;

        if (shouldNotifyItem('updated', item.after.type, commentOnly)) {
            details.push({ title: 'Popularity updated', text: formatUpdatedItem(item) });
        }
    }

    for (const item of diff.removed) {
        if (shouldNotifyItem('removed', item.type, false)) {
            details.push({ title: 'Popularity removed', text: formatRemovedItem(item) });
        }
    }

    for (const detail of details.slice(0, MAX_DETAIL_TOASTS)) {
        notify(detail.title, detail.text, 10000);
    }
}

/*****
* MAIN
*****/
async function runScan(manual = false, ignoreCooldown = false) {

```

```

    if (scanInProgress) {
      if (manual) pushLog('info', 'A scan is already in progress.');
```

return;

```
    }

    ensureUid();
    const metaBefore = getMeta();

    if (!ignoreCooldown) {
      if (metaBefore.backoffUntil && now() < metaBefore.backoffUntil) {
        if (manual) {
          pushLog('info', `Scan paused due to server backoff. Try again in
${timeAgo(metaBefore.backoffUntil)}.`);
          openDropdown();
        }
        return;
      }

      if (metaBefore.lastScan && (now() - metaBefore.lastScan) < (AUTO_SCAN_MINUTES * 60
* 1000)) {
        if (manual) {
          pushLog('info', `Recent scan already completed ${timeAgo(metaBefore.lastScan)}.
Skipping duplicate scan.`);
          openDropdown();
        }
        return;
      }
    }

    scanInProgress = true;
    updateToolbarState();
    renderDropdownContent();

    try {
      pushLog('scan', 'Starting popularity scan...');

      const result = await scanAllPages((msg) => log(msg));
      const newSnapshot = result.entries || {};
      const oldSnapshot = getSnapshot();
      const meta = getMeta();

      const oldCount = Object.keys(oldSnapshot).length;
      const newCount = Object.keys(newSnapshot).length;

      meta.lastScan = Date.now();
      meta.lastPageCount = result.totalPages;
      meta.lastEntryCount = newCount;
      meta.uid = CURRENT_UID;
      meta.backoffUntil = 0;
      meta.lastError = "";

      if (!meta.initialized || oldCount === 0) {
        setSnapshot(newSnapshot);
      }
    }
  }
}

```

```

    meta.initialized = true;
    setMeta(meta);

    pushLog('baseline', `Baseline saved silently with ${newCount} entries across
    ${result.totalPages} page(s).`);

    if (manual) {
        pushLog('info', 'Initial scan completed. No toasts were sent because this was the
baseline.');
```

baseline.');

```

        openDropdown();
    }

    updateToolbarState();
    return;
}

const diff = diffSnapshots(oldSnapshot, newSnapshot);
const summary = buildSummaryFromDiff(diff);
const reportableDiff = filterAcknowledgedDiff(diff);
const reportableSummary = buildSummaryFromDiff(reportableDiff);

meta.lastSummary = summary;
setSnapshot(newSnapshot);
setMeta(meta);

if (summary.total === 0) {
    pushLog('scan', `No changes found. ${newCount} entries scanned across
    ${result.totalPages} page(s).`);
    if (manual) openDropdown();
    updateToolbarState();
    return;
}

if (reportableSummary.total === 0) {
    pushLog('scan', `No new unacknowledged changes. ${summary.total} acknowledged
repeat(s) ignored.');
```

repeat(s) ignored.');

```

    if (manual) openDropdown();
    updateToolbarState();
    return;
}

appendMemberHistory(reportableDiff);
writeDiffToLog(reportableDiff, reportableSummary);
sendGroupedToasts(reportableDiff, reportableSummary);

updateToolbarState();
openDropdown();
} catch (err) {
    console.error('[HF Pop Tracker] Scan error:', err);

    const meta = getMeta();
    meta.lastError = String(err);

```

```

        if (/HTTP 503|HTTP 429|HTTP 502|HTTP 500/i.test(String(err))) {
            meta.backoffUntil = now() + (ERROR_BACKOFF_MINUTES * 60 * 1000);
            setMeta(meta);
            pushLog('error', `${String(err)} - pausing scans for ${ERROR_BACKOFF_MINUTES}
minutes.`);
        } else {
            setMeta(meta);
            pushLog('error', String(err));
        }

        if (manual) openDropdown();
    } finally {
        scanInProgress = false;
        updateToolbarState();
        renderDropdownContent();
    }
}

```

```

function resetBaseline() {
    ensureUid();

```

```

    localStorage.removeItem(STORAGE_KEYS.snapshot);
    localStorage.removeItem(STORAGE_KEYS.meta);
    localStorage.removeItem(STORAGE_KEYS.memberHistory);
    localStorage.removeItem(STORAGE_KEYS.acknowledgedAt);
    localStorage.removeItem(STORAGE_KEYS.acknowledgedTokens);

```

```

    setMeta({
        initialized: false,
        lastScan: 0,
        lastPageCount: 0,
        lastEntryCount: 0,
        uid: CURRENT_UID,
        backoffUntil: 0,
        lastError: "",
        lastSummary: null
    });

```

```

    pushLog('baseline', 'Baseline reset. Next scan will silently rebuild it.');
```

```

    updateToolbarState();
    renderDropdownContent();
    openDropdown();
}

```

```

function maybeStartupScan() {
    const meta = getMeta();

```

```

    if (meta.backoffUntil && now() < meta.backoffUntil) {
        pushLog('info', 'Startup auto-scan skipped due to backoff. Retrying later.');
```

```

        return;
    }

```

```

    if (meta.lastScan && (now() - meta.lastScan) < (STARTUP_SCAN_MIN_AGE_SECONDS *

```

```

1000)) {
    log('Skipping startup scan because a recent scan already happened.');
```

return;

```

}

    runScan(false, false);
}

function startAutoLoop() {
    if (autoLoopStarted) return;
    autoLoopStarted = true;

    setInterval(() => {
        try {
            ensureUid();
        } catch {
            return;
        }

        if (!isAutoEnabled()) return;
        runScan(false, false);
    }, AUTO_SCAN_MINUTES * 60 * 1000);
}

function installMenuCommands() {
    GM_registerMenuCommand('HF Pop Tracker: Scan now', () => runScan(true, true));
    GM_registerMenuCommand('HF Pop Tracker: Open dropdown', () => openDropdown());
    GM_registerMenuCommand('HF Pop Tracker: Acknowledge all', () => {
        ensureUid();
        acknowledgeAll();
    });
    GM_registerMenuCommand('HF Pop Tracker: Reset baseline', () => resetBaseline());
    GM_registerMenuCommand('HF Pop Tracker: Toggle auto', () => {
        ensureUid();
        setAutoEnabled(!isAutoEnabled());
        renderDropdownContent();
        pushLog('info', `Auto scan turned ${isAutoEnabled() ? 'on' : 'off'}.`);
    });
}

function init() {
    ensureUid();
    buildToolbarUI();
    installMenuCommands();
    startAutoLoop();

    setTimeout(() => {
        maybeStartupScan();
    }, 2500);
}

init();
})();

```

