```
// ============================================
// FILE: events/channelHandler.js - FULL FIXED VERSION
// ============================================

const { EmbedBuilder } = require('discord.js');
const fs = require('fs').promises;
const fsSync = require('fs');
const path = require('path');
const { safeRename, getUsernameFromTopic } = require('../functions/rename');

// ===============================
// ⚙️ CONSTANTS
// ===============================
const CAT = {
  SLEEP: '1427958263281881088',
  ACTIVE: '1411034825699233943',
  CYBER: '1446077580615880735',
  DREAM: '1445997821336748155',
  GLITCH: '1445997659948060712',
  EMPTY: '1463173837389828097'
};

const ROLE_ID = '1411991634194989096';
const REPORT_CHANNEL = '1438039815919632394';
const DATA_FILE = path.join(__dirname, '../data/streaks.json');

const STREAK_CATS = [CAT.ACTIVE, CAT.CYBER, CAT.DREAM, CAT.GLITCH];
const ALL_CATS = Object.values(CAT);

const BIOME_MAP = {
  DREAMSPACE: { badge: '🌸', category: CAT.DREAM },
  CYBERSPACE: { badge: '🌐', category: CAT.CYBER },
  GLITCH: { badge: '🧩', category: CAT.GLITCH }
};

module.exports = (client) => {
  const data = new Map();
  let saveTimer = null;
  let saving = false;
  const processing = new Set();

  // ===============================
  // 💾 DATA PERSISTENCE
  // ===============================
  function load() {
    try {
      if (fsSync.existsSync(DATA_FILE)) {
        const json = JSON.parse(fsSync.readFileSync(DATA_FILE, 'utf8'));
        Object.entries(json).forEach(([id, info]) => data.set(id, info));
        console.log('✅ Loaded', data.size, 'channels');
      }
    } catch (e) {
      console.error('❌ Load error:', e.message);
    }
  }

  async function save() {
    if (saving) return;
    try {
      saving = true;
      const dir = path.dirname(DATA_FILE);
      if (!fsSync.existsSync(dir)) await fs.mkdir(dir, { recursive: true });
      await fs.writeFile(DATA_FILE, JSON.stringify(Object.fromEntries(data), null
      console.log('💾 Saved');
    } catch (e) {
      console.error('❌ Save error:', e.message);
    } finally {
      saving = false;
    }
  }

  function scheduleSave() {
    if (saveTimer) clearTimeout(saveTimer);
    saveTimer = setTimeout(save, 2000);
  }

  load();

  // ================================
  // 🛠️ HELPERS
  // ================================
  function getData(id) {
    if (!data.has(id)) {
      data.set(id, {
        streak: 0,
        badges: [],
        times: [],
        days: 0,
        date: null,
        firstBiome: null,
        moving: false
      });
    }
    return data.get(id);
  }

  function getDate() {
    return new Date().toISOString().split('T')[0];
  }

  function getNext13H() {
    const now = new Date();
    const next = new Date(Date.UTC(now.getUTCFullYear(), now.getUTCMonth(), now.g
    if (now >= next) next.setUTCDate(next.getUTCDate() + 1);
    return next;
  }

  function formatTime(ms) {
    const h = Math.floor(ms / 3600000);
    const m = Math.floor((ms % 3600000) / 60000);
    return h + 'h ' + m + 'm';
  }

  function getUserId(topic) {
    if (!topic) return null;
    const parts = topic.trim().split(/\s+/);
    return parts.length >= 2 && /^\d{17,20}$/.test(parts[1]) ? parts[1] : null;
  }

  function parseStreak(name) {
    const m = name.match(/〔(\d+)🔥〕/);
    return m ? parseInt(m[1], 10) : 0;
  }

  function parseBadges(name) {
    const badges = [];
    const m = name.match(/^(.+?)★】/);
    if (!m) return badges;
    const prefix = m[1];
    const withCounter = prefix.match(/x\d+(🌸|🌐|🧩)/g);
    if (withCounter) withCounter.forEach(b => badges.push(b));
    ['🌸', '🌐', '🧩'].forEach(e => {
      if (prefix.includes(e) && !badges.some(b => b.includes(e))) badges.push(e);
    });
    return badges;
  }

  // ================================
  // ⏱️ TÍNH THỜI GIAN MACRO THỰC TẾ
  // ================================

  /**
   * Tính tổng thời gian macro từ timestamps
   * Loại bỏ khoảng nghỉ (break) - chỉ tính thời gian macro liên tục
   *
   * @param {Array} times - Mảng timestamps
   * @param {Number} maxGap - Khoảng cách tối đa giữa 2 message (ms)
   * @returns {Number} Tổng thời gian macro (ms)
   */
  function calcActive(times, maxGap = 10 * 60 * 1000) {
    console.log('🔍 calcActive called with:', times?.length, 'timestamps');

    if (!times || times.length === 0) {
      console.log('  ❌ No times provided');
      return 0;
    }

    if (times.length === 1) {
      console.log('  ⚠️ Only 1 timestamp, returning 0');
      return 0;
    }

    // Validate timestamps are numbers
    const validTimes = times.filter(t => typeof t === 'number' && !isNaN(t));
    if (validTimes.length !== times.length) {
      console.log('  ⚠️ Some invalid timestamps filtered:', times.length - validT
    }

    if (validTimes.length < 2) {
      console.log('  ❌ Not enough valid timestamps');
      return 0;
    }

    const sorted = [...validTimes].sort((a, b) => a - b);
    let totalActive = 0;
    let gaps = 0;

    console.log('  📊 First timestamp:', new Date(sorted[0]).toISOString());
    console.log('  📊 Last timestamp:', new Date(sorted[sorted.length - 1]).toISC

    for (let i = 1; i < sorted.length; i++) {
      const gap = sorted[i] - sorted[i - 1];
      if (gap <= maxGap) {
        totalActive += gap;
      } else {
        gaps++;
        console.log(`  ⏸️ Gap ${gaps}: ${formatTime(gap)} (ignored)`);
      }
    }

    console.log('  ✅ Total active:', formatTime(totalActive));
    return totalActive;
  }

  function getCatName(id) {
    return {
      [CAT.SLEEP]: 'Dormant',
      [CAT.ACTIVE]: 'Active',
      [CAT.CYBER]: 'Cyberspace',
      [CAT.DREAM]: 'Dreamspace',
      [CAT.GLITCH]: 'Glitch',
      [CAT.EMPTY]: 'Empty'
    }[id] || 'Unknown';
  }

  // ================================
  // 🔍 DETECT BIOMES - 2 PHƯƠNG THỨC
  // ================================

  function detectBiomeFromMessage(msg) {
    const check = (text) => {
      if (!text) return null;
      const t = text.toUpperCase();
      if (t.includes('DREAMSPACE')) return 'DREAMSPACE';
      if (t.includes('CYBERSPACE')) return 'CYBERSPACE';
      if (t.includes('GLITCH')) return 'GLITCH';
      return null;
    };

    if (msg.embeds && msg.embeds.length > 0) {
      for (const e of msg.embeds) {
        const result = check(e.title) || check(e.description);
        if (result) return result;
      }
    }

    return check(msg.content);
  }

  async function detectBiomesFromEveryone(ch) {
    try {
      console.log('🔍 Searching for @everyone message in', ch.name);
      const messages = await ch.messages.fetch({ limit: 50 });
      const everyoneMsg = messages.find(m => m.content.includes('@everyone'));

      if (!everyoneMsg) {
        console.log('⚠️ No @everyone message found');
        return [];
      }

      console.log('✅ Found @everyone at', new Date(everyoneMsg.createdTimestamp)

      const around = messages.filter(m =>
        Math.abs(m.createdTimestamp - everyoneMsg.createdTimestamp) < 30 * 60 * 1
      );

      console.log('📊 Found', around.size, 'messages within 30 minutes');

      const found = new Set();

      for (const msg of around.values()) {
        for (const embed of msg.embeds || []) {
          const text = `${embed.title || ''} ${embed.description || ''}`.toUpperC
          for (const key of Object.keys(BIOME_MAP)) {
            if (text.includes(key)) {
              found.add(key);
              console.log('🌈 Found biome in embed:', key);
            }
          }
        }
      }

      console.log('✅ Total biomes detected:', [...found]);
      return [...found];

    } catch (e) {
      console.error('⚠️ Detect biomes error:', e.message);
      return [];
    }
  }

  // ================================
  // 👤 ROLE MANAGEMENT
  // ================================
  async function updateRole(ch, add) {
    try {
      const userId = getUserId(ch.topic);
      if (!userId) return;
      const member = await ch.guild.members.fetch(userId).catch(() => null);
      if (!member) return;
      const has = member.roles.cache.has(ROLE_ID);
      if (add && !has) {
        await member.roles.add(ROLE_ID);
        console.log('✅ Role added:', member.user.tag);
      } else if (!add && has) {
        await member.roles.remove(ROLE_ID);
        console.log('❌ Role removed:', member.user.tag);
      }
    } catch (e) {
      console.error('⚠️ Role error:', e.message);
    }
  }

  // ===============================
  // 🌈 BIOME HANDLING
  // ===============================
  async function handleBiome(ch, biomeKey) {
    try {
      const d = getData(ch.id);
      const biome = BIOME_MAP[biomeKey];
      if (!biome) return;

      const idx = d.badges.findIndex(b => b.includes(biome.badge));

      if (idx !== -1) {
        const m = d.badges[idx].match(/x(\d+)/);
        const count = m ? parseInt(m[1], 10) : 1;
        d.badges[idx] = 'x' + (count + 1) + biome.badge;
        console.log('🔄 Badge++:', d.badges[idx]);
      } else {
        if (!d.firstBiome) {
          d.firstBiome = biomeKey;
          d.badges = [biome.badge];
          d.moving = true;
          await ch.setParent(biome.category, { lockPermissions: false });
          await new Promise(r => setTimeout(r, 500));
          console.log('🌟 First biome:', biomeKey);
        } else {
          d.badges.push(biome.badge);
          console.log('➕ Badge added:', biome.badge);
        }
      }

      await safeRename(ch, d.streak, d.badges);
      await updateRole(ch, true);
      scheduleSave();
    } catch (e) {
      console.error('❌ Biome error:', e.message);
    }
  }

  // ===============================
  // 📨 WEBHOOK DETECTION - REAL-TIME!
  // ===============================
  client.on('messageCreate', async (msg) => {
    try {
      if (msg.webhookId) {
        console.log('═══════════════════════════════════');
        console.log('🎯 WEBHOOK DETECTED!');
        console.log('  Channel:', msg.channel.name);
        console.log('  Category:', msg.channel.parentId);
        console.log('  Webhook ID:', msg.webhookId);
        console.log('  Author:', msg.author.tag);
        console.log('  Author ID:', msg.author.id);
        console.log('  Has Embeds:', (msg.embeds?.length || 0));
        console.log('  Content:', msg.content.substring(0, 100));
        console.log('═══════════════════════════════════');
      }

      if (!msg.webhookId) return;

      const ch = msg.channel;

      if (!ch || !ALL_CATS.includes(ch.parentId)) {
        console.log('❌ Not in tracked category');
        return;
      }

      const userId = getUserId(ch.topic);
      if (!userId) {
        console.log('❌ No user ID in topic:', ch.topic);
        return;
      }

      console.log('✅ Webhook belongs to channel with valid topic');
      console.log('✅ PROCESSING WEBHOOK');

      const now = Date.now();
      const d = getData(ch.id);
      const today = getDate();

      if (d.date !== today) {
```

```
  d.times = [];
  d.date = today;
  console.log(' 📅 New day detected, reset times');
}

const biomeKey = detectBiomeFromMessage(msg);
if (biomeKey) {
  console.log('🌈 Biome detected from message:', biomeKey);
}

if (ch.parentId === CAT.SLEEP || ch.parentId === CAT.EMPTY) {
  console.log('⏰ WAKING UP from', getCatName(ch.parentId));

  d.streak = parseStreak(ch.name) || 0;
  d.times = [now];
  d.days = 0;

  if (biomeKey) {
    console.log('→ To BIOME (from message):', biomeKey);
    await handleBiome(ch, biomeKey);
  } else {
    console.log('→ Checking for @everyone messages...');
    const biomeKeys = await detectBiomesFromEveryone(ch);

    if (biomeKeys.length > 0) {
      console.log('→ To BIOME (from @everyone):', biomeKeys[0]);

      await handleBiome(ch, biomeKeys[0]);

      for (let i = 1; i < biomeKeys.length; i++) {
        const biome = BIOME_MAP[biomeKeys[i]];
        if (!biome) continue;

        const idx = d.badges.findIndex(b => b.includes(biome.badge));
        if (idx !== -1) {
          const m = d.badges[idx].match(/x(\d+)/);
          const count = m ? parseInt(m[1], 10) : 1;
          d.badges[idx] = 'x' + (count + 1) + biome.badge;
          console.log('🔄 Badge++ (additional):', d.badges[idx]);
        } else {
          d.badges.push(biome.badge);
          console.log('➕ Badge added (additional):', biome.badge);
        }
      }

      await safeRename(ch, d.streak, d.badges);
```

```
        } else {
          console.log('→ To ACTIVE (no biome found)');
          d.moving = true;
          await ch.setParent(CAT.ACTIVE, { lockPermissions: false });
          await new Promise(r => setTimeout(r, 500));
          await safeRename(ch, d.streak, d.badges);
        }
      }

      await updateRole(ch, true);
      scheduleSave();
      return;
    }

    if (biomeKey) {
      await handleBiome(ch, biomeKey);
    }

    d.times.push(now);
    console.log('  ⏱️ Timestamp saved. Total times:', d.times.length);
    scheduleSave();

    console.log('✅ Webhook processed');

  } catch (e) {
    console.error('❌ messageCreate error:', e.message);
    console.error(e.stack);
  }
});

// ================================
// ⏰ DAILY CHECK 13:00 VN - FIXED!
// ================================
async function dailyCheck() {
  try {
    console.log('═══════════════════════════════════');
    console.log('🔔 DAILY CHECK - 13:00 VN');
    console.log('⏱️ Calculating active time from saved timestamps...');
    console.log('═══════════════════════════════════');

    const guild = client.guilds.cache.first();
    if (!guild) return;

    const report = await guild.channels.fetch(REPORT_CHANNEL).catch(() => null)

    const channels = guild.channels.cache.filter(c =>
      c.type === 0 && STREAK_CATS.includes(c.parentId)
```

```
);

console.log(`📊 Scanning ${channels.size} channels...`);

const results = { above18h: [], above12h: [], above6h: [] };

for (const [, ch] of channels) {
  try {
    const d = getData(ch.id);

    console.log(`\n🔍 Processing: ${ch.name}`);
    console.log(`  📊 Saved timestamps: ${d.times?.length || 0}`);

    if (!d.times || !Array.isArray(d.times) || d.times.length === 0) {
      console.log(`  ⚠️ No timestamps saved today → 0 hours`);

      d.days++;
      console.log(`  ⚠️ WARNING: Day ${d.days}/3 (0h < 6h)`);

      if (d.days >= 3) {
        const old = d.streak;
        d.streak = 0;
        d.badges = [];
        d.firstBiome = null;
        d.moving = true;
        console.log(`  😴 MOVING TO DORMANT (lost ${old}🔥)`);
        await ch.setParent(CAT.SLEEP, { lockPermissions: false });
        await new Promise(r => setTimeout(r, 500));
        await updateRole(ch, false);
        await safeRename(ch, 0, []);
        d.days = 0;
      }

      d.times = [];
      d.date = getDate();
      continue;
    }

    const active = calcActive(d.times, 10 * 60 * 1000);
    const hours = active / 3600000;

    console.log(`  ⏱️ Active time: ${formatTime(active)} (${hours.toFixed(2
    console.log(`  📊 Current streak: ${d.streak}🔥`);
    console.log(`  ⚠️ Warning days: ${d.days}/3`);
    console.log(`  📋 Timestamps breakdown:`);

    if (d.times.length > 0) {
```

```
    const sorted = [...d.times].sort((a, b) => a - b);
    const first = new Date(sorted[0]).toLocaleTimeString('vi-VN');
    const last = new Date(sorted[sorted.length - 1]).toLocaleTimeString('
    console.log(`     First: ${first}`);
    console.log(`     Last: ${last}`);
    console.log(`     Count: ${d.times.length} webhooks`);

    let totalGaps = 0;
    for (let i = 1; i < sorted.length; i++) {
      const gap = sorted[i] - sorted[i - 1];
      if (gap > 10 * 60 * 1000) {
        totalGaps++;
        console.log(`     Gap ${totalGaps}: ${formatTime(gap)} (break det
      }
    }
  }

  if (hours >= 18) results.above18h.push({ ch, active });
  if (hours >= 12) results.above12h.push({ ch, active });
  if (hours >= 6) results.above6h.push({ ch, active });

  if (hours >= 6) {
    d.streak++;
    d.days = 0;
    console.log(`  ✅ STREAK SAVED: ${d.streak - 1}🔥 → ${d.streak}🔥`);
    await safeRename(ch, d.streak, d.badges);
  } else {
    d.days++;
    console.log(`  ⚠️ WARNING: Day ${d.days}/3 (${hours.toFixed(2)}h < 6h

    if (d.days >= 3) {
      const old = d.streak;
      d.streak = 0;
      d.badges = [];
      d.firstBiome = null;
      d.moving = true;
      console.log(`  😴 MOVING TO DORMANT (lost ${old}🔥)`);
      await ch.setParent(CAT.SLEEP, { lockPermissions: false });
      await new Promise(r => setTimeout(r, 500));
      await updateRole(ch, false);
      await safeRename(ch, 0, []);
      d.days = 0;
    }
  }

  d.times = [];
  d.date = getDate();
```

```
    } catch (err) {
      console.error(`  ❌ Error processing ${ch.name}:`, err.message);
      console.error(err.stack);
    }
  }

  scheduleSave();

  console.log('\n════════════════════════════════════');
  console.log('📊 DAILY CHECK SUMMARY:');
  console.log(`  🏆 18+ hours: ${results.above18h.length} channels`);
  console.log(`  ⭐ 12+ hours: ${results.above12h.length} channels`);
  console.log(`  ✨ 6+ hours: ${results.above6h.length} channels`);
  console.log('════════════════════════════════════');

  if (report) {
    const date = new Date().toLocaleDateString('vi-VN', { timeZone: 'Asia/Ho_
    const embeds = [];

    [
      { key: 'above18h', title: '🏆 18+ Hours', color: 0xFFD700 },
      { key: 'above12h', title: '⭐ 12+ Hours', color: 0xC0C0C0 },
      { key: 'above6h', title: '✨ 6+ Hours', color: 0xCD7F32 }
    ].forEach(cfg => {
      if (results[cfg.key].length > 0) {
        const desc = results[cfg.key]
          .map(r => `**${r.ch.name}** - ${getCatName(r.ch.parentId)} - ${form
          .join('\n');
        embeds.push(
          new EmbedBuilder()
            .setTitle(cfg.title)
            .setColor(cfg.color)
            .setDescription(desc)
            .setTimestamp()
        );
      }
    });

    if (embeds.length > 0) {
      await report.send({ content: `📊 **Daily Report** - ${date}`, embeds })
      console.log('✅ Report sent to channel');
    } else {
      await report.send(`📊 **Daily Report** - ${date}\n⚠️ No channels reache
      console.log('⚠️ No channels reached 6+ hours');
    }
  }
```

```
    console.log('✅ Daily check completed\n');

  } catch (e) {
    console.error('❌ Daily check error:', e.message);
    console.error(e.stack);
  } finally {
    scheduleDailyCheck();
  }
}

function scheduleDailyCheck() {
  const next = getNext13H();
  console.log('⏰ Next check:', next.toISOString());
  setTimeout(dailyCheck, next - new Date());
}

// ================================
// 🎬 OTHER EVENTS
// ================================
client.on('channelCreate', async (ch) => {
  try {
    if (ch.type !== 0 || !ALL_CATS.includes(ch.parentId)) return;
    console.log('🆕 Channel created:', ch.name);

    for (let i = 0; i < 5; i++) {
      await new Promise(r => setTimeout(r, 500));
      await ch.fetch();
      if (ch.topic) break;
    }

    if (!ch.topic) return;

    const d = getData(ch.id);
    d.streak = 0;
    d.badges = [];

    if (ch.parentId === CAT.SLEEP || ch.parentId === CAT.EMPTY) {
      await updateRole(ch, false);
    } else if (STREAK_CATS.includes(ch.parentId)) {
      await updateRole(ch, true);
    }

    await safeRename(ch, 0, []);
    scheduleSave();

  } catch (e) {
    console.error('❌ channelCreate error:', e.message);
  }
});

client.on('channelUpdate', async (old, ch) => {
  try {
    if (!ch || ch.type !== 0) return;
    if (!ALL_CATS.includes(ch.parentId) && !ALL_CATS.includes(old.parentId)) re
    if (processing.has(ch.id)) return;

    processing.add(ch.id);

    try {
      const d = getData(ch.id);

      if (old.parentId !== ch.parentId) {
        console.log('📦 Category change:', ch.name);

        if (d.moving) {
          d.moving = false;
          scheduleSave();
          return;
        }

        await new Promise(r => setTimeout(r, 500));
        await ch.fetch();

        if (STREAK_CATS.includes(ch.parentId)) {
          await updateRole(ch, true);
          d.days = 0;
          await safeRename(ch, d.streak, d.badges);
        } else if (ch.parentId === CAT.SLEEP || ch.parentId === CAT.EMPTY) {
          await updateRole(ch, false);
          d.streak = 0;
          d.days = 0;
          d.times = [];
          d.badges = [];
          d.firstBiome = null;
          await safeRename(ch, 0, []);
        }

        scheduleSave();
      }

      if (old.name !== ch.name) {
        const streak = parseStreak(ch.name);
        const badges = parseBadges(ch.name);
        if (streak >= 0 && streak !== d.streak) d.streak = streak;
        if (badges.length > 0) d.badges = badges;
        scheduleSave();
      }

    } finally {
      processing.delete(ch.id);
    }

  } catch (e) {
    console.error('❌ channelUpdate error:', e.message);
    processing.delete(ch.id);
  }
});

client.on('channelDelete', (ch) => {
  if (data.has(ch.id)) {
    data.delete(ch.id);
    scheduleSave();
    console.log('🗑️ Deleted:', ch.name);
  }
});

// ================================
// 🚀 INITIALIZATION
// ================================
async function scanAll(guild) {
  try {
    console.log('🔍 Scanning...');

    const channels = guild.channels.cache.filter(c =>
      c.type === 0 && ALL_CATS.includes(c.parentId)
    );

    const today = getDate();

    for (const [, ch] of channels) {
      try {
        const d = getData(ch.id);
        const streak = parseStreak(ch.name);
        const badges = parseBadges(ch.name);

        if (streak >= 0 && streak !== d.streak) d.streak = streak;
        if (badges.length > 0) d.badges = badges;
        if (d.date !== today) d.times = [];

        if (STREAK_CATS.includes(ch.parentId)) await updateRole(ch, true);
        else if (ch.parentId === CAT.SLEEP) await updateRole(ch, false);

        await safeRename(ch, d.streak, d.badges);

      } catch (e) {
        console.error('⚠️ Sync error:', ch.name, e.message);
      }
    }

    scheduleSave();
    console.log('✅ Synced', channels.size, 'channels');

  } catch (e) {
    console.error('❌ Scan error:', e.message);
  }
}

client.once('ready', async () => {
  try {
    console.log('🤖 Bot ready!');
    const guild = client.guilds.cache.first();
    if (guild) {
      await scanAll(guild);
      scheduleDailyCheck();
    }
    console.log('🚀 All systems go!');
  } catch (e) {
    console.error('❌ Ready error:', e.message);
  }
});

// ================================
// 🧪 COMMANDS
// ================================
client.on('messageCreate', async (msg) => {
  // Test daily check
  if (msg.content === '!testdaily') {
    await msg.reply('🔄 Running daily check manually...');
    await dailyCheck();
    return;
  }

  // Xem thông tin channel hiện tại
  if (msg.content === '!info') {
    try {
      const ch = msg.channel;
      if (!ALL_CATS.includes(ch.parentId)) {
```

```
  await msg.reply('⚠️ Channel này không được track!');
  return;
}

const d = getData(ch.id);
const today = getDate();

console.log('\n🔍 !info command called');
console.log('  Channel:', ch.name);
console.log('  Channel ID:', ch.id);
console.log('  Data:', JSON.stringify(d, null, 2));

// Tính thời gian từ timestamps đã lưu
const active = calcActive(d.times || [], 10 * 60 * 1000);
const hours = active / 3600000;

console.log('  Calculated hours:', hours);

// Tạo embed info
const embed = new EmbedBuilder()
  .setTitle(`📊 Channel Info: ${ch.name}`)
  .setColor(0x00AE86)
  .addFields(
    { name: '🔥 Current Streak', value: `${d.streak}`, inline: true },
    { name: '📅 Date', value: d.date || 'N/A', inline: true },
    { name: '⚠️ Warning Days', value: `${d.days}/3`, inline: true },
    { name: '📍 Category', value: getCatName(ch.parentId), inline: true }
    { name: '🎨 Badges', value: d.badges.length > 0 ? d.badges.join(' ')
    { name: '🌟 First Biome', value: d.firstBiome || 'None', inline: true
    { name: '📊 Today\'s Data', value: '───────────────', inline: false }
    { name: '⏱️ Webhooks Received', value: `${(d.times || []).length} mes
    { name: '⏰ Active Time (Today)', value: `${formatTime(active)} (${hc
    { name: '✅ Streak Status', value: hours >= 6 ? '✅ Will save' : `⚠️
  )
  .setTimestamp();

// Hiển thị timestamps chi tiết nếu có
if (d.times && d.times.length > 0) {
  const sorted = [...d.times].sort((a, b) => a - b);
  const first = new Date(sorted[0]).toLocaleTimeString('vi-VN', { timeZon
  const last = new Date(sorted[sorted.length - 1]).toLocaleTimeString('vi

  embed.addFields({
    name: '📋 Timeline',
    value: `First webhook: ${first}\nLast webhook: ${last}\nTotal span: $
    inline: false
  });
```

```
    }

    await msg.reply({ embeds: [embed] });

  } catch (e) {
    console.error('❌ !info error:', e);
    await msg.reply('❌ Error: ' + e.message);
  }
  return;
}

// Xem tất cả channels đang track
if (msg.content === '!stats') {
  try {
    const guild = msg.guild;
    const channels = guild.channels.cache.filter(c =>
      c.type === 0 && STREAK_CATS.includes(c.parentId)
    );

    let text = '```\n📊 ACTIVE CHANNELS STATS\n\n';

    for (const [, ch] of channels) {
      const d = getData(ch.id);
      const active = calcActive(d.times || [], 10 * 60 * 1000);
      const hours = active / 3600000;

      text += `${ch.name}\n`;
      text += `  Streak: ${d.streak}🔥 | Today: ${hours.toFixed(1)}h`;
      text += ` | Webhooks: ${(d.times || []).length}`;
      text += ` | Warns: ${d.days}/3\n\n`;
    }

    text += '```';

    await msg.reply(text);

  } catch (e) {
    await msg.reply('❌ Error: ' + e.message);
  }
  return;
}

// Reset channel data (admin only)
if (msg.content.startsWith('!reset')) {
  if (!msg.member.permissions.has('Administrator')) {
    await msg.reply('❌ Admin only!');
    return;
```

```
      }

      const ch = msg.channel;
      if (!ALL_CATS.includes(ch.parentId)) {
        await msg.reply('⚠️ Channel này không được track!');
        return;
      }

      const d = getData(ch.id);
      d.times = [];
      d.days = 0;
      scheduleSave();

      await msg.reply('✅ Reset timestamps và warning days!');
      return;
    }
  });
};
```