Subject: إضافة نظام توجيه / توزيع زيارات `/sales` حسب مصدر الزيارة (Next.js)

مرحبًا،
عايزين نضيف نظام توجيه (redirect) لصفحة `/sales` بحيث الزيارات تتوزع على صفحات المبيعات `/sales1` .. `/sales8` حسب القواعد التالية، ويتم التنفيذ داخل تطبيق Next.js Server-side.

المنطق المطلوب (بوضوح)

1. لو الزيارة جايه من Google (referer يحتوي `google.com` أو `googleads`):
    - نطبق نسب ثابتة (Weighted distribution) على الصفحات التالية:
        - `/sales1` → 15%
        - `/sales2` → 15%
        - `/sales3` → 15%
        - `/sales4` → 13.75%
        - `/sales5` → 13.75%
        - `/sales6` → 13.75%
        - `/sales8` → 13.75%
2. لو الزيارة جايه من أي مصدر آخر (Direct, Facebook, TikTok, إلخ):
    - نطبق توزيع مُعدل:
        - الثلاث صفحات الخاصة بـ إيهاب (`/sales1`,`/sales2`,`/sales3`) إجماليهم 20% (يعني ~6.666% لكل صفحة).
        - الأربع صفحات الباقية (`/sales4`,`/sales5`,`/sales6`,`/sales8`) إجماليهم 80% — يعني 20% لكل صفحة.
3. نحافظ على أي query string / UTM parameters (مهم لتتبع الحملات عند إعادة التوجيه).

4. التنفيذ يجب أن يكون **server-side** قبل عرض أي صفحة عشان **Redirect** يكون فوري وسريع.

التقنية المطلوبة والملفات

- **Next.js v15.5.4 (app router أو pages router)**
- نضيف ملف `middleware.ts` في جذر المشروع (root) لتنفيذ الـ **redirect** قبل الوصول لأي صفحة.
- نستخدم `NextResponse.redirect()` من `next/server`.

الكود المقترح (ضعه في `/middleware.ts`)

هذا الكود جاهز للنسخ، يحافظ على الـ **query string**، ويتحقق من الـ **referer**، وينفّذ التوزيع كما طلبت.

```
// /middleware.ts
import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

export function middleware(req: NextRequest) {
  // ننسخ الـ URL (نحتاج الـ search params الحالي)
  const originalUrl = req.nextUrl
  const query = originalUrl.search || '' // => ?utm_source=... إلخ

  // نجلب الهيدر
  const referer = (req.headers.get('referer') || '').toLowerCase()

  // نجيب رقم عشوائي من 0 إلى 100
  const rnd = Math.random() * 100

  // حالة زوار Google (weighted)
  if (referer.includes('google.com') ||
referer.includes('googleads.g.doubleclick.net')) {
    let target = '/sales1' // افتراضي
    if (rnd < 15) target = '/sales1'
```

```
    else if (rnd < 30) target = '/sales2'
    else if (rnd < 45) target = '/sales3'
    else if (rnd < 58.75) target = '/sales4'
    else if (rnd < 72.5) target = '/sales5'
    else if (rnd < 86.25) target = '/sales6'
    else target = '/sales8'

    return NextResponse.redirect(`${target}${query}`)
  }

  // 4 حالة أي مصدر تاني – توزيع محدد: إيهاب (3 صفحات) = 20% إجمالاً، الباقي 80% يتوزع
(صفحات (20% لكل واحدة)
  // نستخدم نفس المقياس من 0..100
  let target = '/sales1'
  if (rnd < 6.6666667) target = '/sales1'   // ~6.666% (20جزء من الـ%)
  else if (rnd < 13.3333334) target = '/sales2' // next ~6.666%
  else if (rnd < 19.9999999) target = '/sales3' // next ~6.666%  =>
20~ المجموع%
  else if (rnd < 39.9999999) target = '/sales4' // 20%
  else if (rnd < 59.9999999) target = '/sales5' // 20%
  else if (rnd < 79.9999999) target = '/sales6' // 20%
  else target = '/sales8'                        // 20%

  return NextResponse.redirect(`${target}${query}`)
}

// نفعّل الميدل وير على مسار /sales فقط (أو /sales/* لو عايزين)
export const config = {
  matcher: ['/sales'],
}
```

ملاحظات عن الكود:

- **بيحتفظ بالـ query string (`originalUrl.search`) وبيضيفها على رابط الـ redirect.**
- **بنقارن `referer` بطريقة بسيطة (قد تضيف شروط إضافية لو فيه referrer proxies أو click trackers).**
- **`matcher: ['/sales']` يعني إن الروتين يُشغل لما حد يدخل `/sales`. لو عايز يشمل `/sales/` و`/sales?utm=...` يمررها. لو عايز تغطي أي لاحقة، استخدم `matcher:` فـ Next هذا يكفي لأن**

`['/sales/:path*']`.

خطوات التنفيذ (بالتفصيل على الـ VPS)

فتح المشروع على السيرفر

```
cd /path/to/project
git pull origin main
```

1.
2. إنشاء الملف
   في جذر المشروع أنشئ `middleware.ts` والصق الكود أعلاه.

3. لو تستخدم TypeScript تأكد من إعدادات tsconfig. إذا مشروع JS فقط، أنشئ `middleware.js` مع تحويل صغير للـ syntax.

بناء وتشغيل

```
npm install
npm run build
npm run start   # أو الطريقة اللي بتشغّل بيها app على VPS (pm2, systemd, docker ...)
```

4.
5. إذا شغّال عبر proxy (nginx)
   بالطريقة العادية. (عادة ما يحدث query string وأنه يعبر الـ `referer`، ما بيشيلش header تأكد إن nginx شيء تلقائي).

6. اختبار محلي سريع (curl)

محاكاة زيارة من Google:

```
curl -I -H "Referer: https://www.google.com/search?q=test" "https://cv-qsr.sa/sales"
```

- راجع هيدر `Location:` لو ضفت واحد query اللي بيرجع ويُظهر الصفحه الموجّهة إليها مع الـ.

محاكاة زيارة من مصدر تاني (مثلاً Facebook):

```
curl -I -H "Referer: https://www.facebook.com"
"https://cv-qsr.sa/sales"
```

7. كرّر الاختبارات عدة مرات لتلاحظ التوزيع (أو شغّل سكربت صغير يعد التوجيهات).
8. اختبار بالمتصفح
   افتح نافذة متخفي (incognito) وجرب الوصول للرابط `https://cv-qsr.sa/sales` مباشرة، وبعدين جرب وضع `?utm_source=test` وتابع Redirect وظهور الـ UTM في الـ destination URL.

تتبع و Logging (اقتراح اختياري)

- لو حابب تسجل كل تحويل في قاعدة بيانات PostgreSQL عبر Prisma:
  - لا تنفّذ عمليات DB مباشرة من `middleware` لأن `middleware` قد يعمل في Edge runtime في بعض الإعدادات. أفضل طريقة:
    1. أنشئ API route `/api/log-redirect` (Server route) يكتب سجل في جدول `traffic_distribution` عبر Prisma.

أو من صفحة الوجهة. مثال مبسّط لاستدعاء (fire-and-forget) `fetch()` عبر `middleware` استدعِ هذا المسار من الـ middleware:

```
// بعد تحديد target ولكن قبل return redirect
// استدعاء مباشر بدون redirect - لا تقطع تنفيذ timeout أو مع await بالانتظار الطويل
fetch('https://cv-qsr.sa/api/log-redirect', {
  method: 'POST',
  headers: { 'content-type': 'application/json' },
  body: JSON.stringify({ target, referer, ts: Date.now() })
}).catch(() => {})
```

2.
3. API route يستقبل ويدخل السجل في PostgreSQL عبر Prisma.

- هذا بيسمح لك تجميع تقارير دقيقة عن التوزيع والمصدر.

كيف نتحقق إن النِّسَب مظبوطة عمليًا؟

- اعمل اختبار تحميل (مثلاً سكربت يرسل `Referer: google.com`) 10,000 طلب مع وشوف نسبة الـ redirect.
- سجّل redirect counts في جدول DB ولمدة أسبوع ثم احسب النسب.
- راقب الـ UTM في Google Analytics لـ utm لتتأكد إن الزيارات بتتجه للصفحات المرادة مع المحافظة على الـ params.

ملاحظات نهائية للمطور

- إذا كان التطبيق يعمل على أكثر من instance (load-balanced) فـ استخدام `Math.random()` لكل request يكفي للتوزيع العادل على المدى الطويل.
- لو محتاج توزيع محدد دقيق جدًا وللاحتفاظ بحالة (Redis متسلسلة، نحتاج تخزين عدّاد مركزي Round-Robin أو DB). لكن المطلوب هنا يعتمد على نسب عشوائية/موزونة، فـ `Math.random()` مناسب وسهل.
- تأكد إن أي تغييرات لا تؤثر على NextAuth أو صفحات المبيعات نفسها