🎯 PROMPT: تطوير سوق الذهب على منصة Xchange

المهمة الرئيسية

مهمتك هي بناء نظام متكامل لتجارة Xchange Egypt. أنت مطور Full-Stack خبير تعمل على تطوير **سوق الذهب الإلكتروني** على منصة Xchange الذهب (جديد ومستعمل) مع مميزات تنافسية فريدة تجعل أفضل منصة لتجارة الذهب في مصر.

📚 الموارد المتاحة

لديك **حزمة تطوير كاملة** في الملف:

- xchange-gold-marketplace-dev-package.md

هذه الحزمة تحتوي على:

1. ✅ Database Schema (Prisma) كامل
2. ✅ API Endpoints Documentation مع أمثلة
3. ✅ User Stories التفصيلية
4. ✅ Integration Guides (Metals-API, XRF, Certificates)
5. ✅ Business Logic & Algorithms جاهزة
6. ✅ Testing Scenarios
7. ✅ Security & Compliance guidelines

اقرأ هذا الملف أولاً قبل البدء في أي تطوير.

🏗️ البنية التقنية للمشروع

Stack التقني:

- **Frontend:** Next.js 14+ (App Router)
- **Backend:** Express.js
- **Database:** PostgreSQL + Prisma ORM
- **Language:** TypeScript

- **Authentication:** JWT
- **File Storage:** Google Cloud Storage أو AWS S3
- **Real-time:** Socket.io (للأسعار الفورية)

المجلدات الأساسية:

```
xchange/
├── apps/
│   ├── web/              # Next.js frontend
│   └── api/              # Express backend
├── packages/
│   ├── database/         # Prisma schema
│   ├── ui/               # Shared UI components
│   └── types/            # Shared TypeScript types
├── prisma/
│   ├── schema.prisma     # Database schema
│   ├── migrations/       # Database migrations
│   └── seed.ts           # Seed data
└── docs/                 # Documentation
```

🎯 خطة التطوير المرحلية

المرحلة 1: Database & Core Models (أولوية قصوى)

المهمة: قم بتطوير Database Schema الكامل لسوق الذهب باستخدام Prisma.

الخطوات التفصيلية:

1. **أضف Models إلى الذهب `prisma/schema.prisma`:**
   - من حزمة التطوير "Database Schema". راجع القسم "1"
   - انسخ جميع الـ Models التالية:
     - `GoldProduct`
     - `GoldCategory`
     - `GoldPriceHistory`
     - `WorkmanshipFeeTemplate`
     - `BarterListing`
     - `BarterNegotiation`

- `GoldSavingsPlan`
- `SavingsPayment`
- `XRFTestResult`
- `AuthenticationCertificate`
- `GoldInsurance`
- `InsuranceClaim`

- **مهم:** تأكد من إضافة الحقول الجديدة إلى `User` model:

```prisma
model User {
  // ... existing fields

  // Gold-specific
  goldProductsSelling     GoldProduct[]       @relation("GoldSeller")
  barterListingsOffering  BarterListing[]     @relation("BarterOffering")
  barterListingsReceived  BarterListing[]     @relation("BarterCounterparty")
  barterNegotiations      BarterNegotiation[]
  savingsPlans            GoldSavingsPlan[]
  goldInsurances          GoldInsurance[]
  goldTraderVerified      Boolean             @default(false)
  goldTraderLevel         String?
  totalGoldTradeValue     Float               @default(0)
  preferredKarat          GoldKarat?
  notifyPriceAlerts       Boolean             @default(false)
  priceAlertThreshold     Float?
}
```

2. **أنشئ Migration:**

```bash
npx prisma migrate dev --name add_gold_marketplace
```

3. **أنشئ Seed Data للاختبار:**

- في `prisma/seed.ts`، أضف:
  - 3-5 فئات ذهب (سبائك، مشغولات، جنيهات ذهب)
  - 10+ منتجات ذهب نموذجية بأوزان وعيارات مختلفة

- 2-3 قوالب مصنعية (WorkmanshipFeeTemplate)
- سجلات أسعار تاريخية (آخر 30 يوم)

4. **اختبر الـ Schema:**

```bash
npm run db:seed
npm run db:studio  # افتح Prisma Studio للتأكد
```

معايير القبول:

- ✅ جميع الـ Models تم إضافتها بنجاح
- ✅ Migrations تعمل بدون أخطاء
- ✅ Seed data يتم إنشاؤه بنجاح
- ✅ العلاقات بين الجداول (Foreign Keys) صحيحة

المرحلة 2: Gold Pricing Service (حرج)

المهمة: تطوير خدمة الأسعار الفورية باستخدام Metals-API مع نظام Cache ذكي.

الخطوات:

1. **أنشئ ملف** `apps/api/src/services/goldPriceService.ts`:
   - انسخ الكود من القسم "4.1 Metals-API Integration"
   - أضف:
     - دالة `getCurrentGoldPrice()` - تستدعي Metals-API
     - دالة `getGoldPrice()` - مع Cache لمدة 60 ثانية
     - دالة `saveToHistory()` - حفظ الأسعار في Database

2. **Environment Variables:** أضف إلى `.env`:

```env
METALS_API_KEY=your_api_key_here
METALS_API_BASE_URL=https://metals-api.com/api
```

3. **أنشئ API Endpoint للأسعار:**
   - `GET /api/gold/prices/current` - الأسعار الحالية
   - `GET /api/gold/prices/history` - تاريخ الأسعار
   - `POST /api/gold/prices/calculate` - حساب سعر منتج

4. **Cron Job لتحديث الأسعار:**

```typescript
// apps/api/src/cron/updateGoldPrices.ts
import cron from 'node-cron';
import { getGoldPrice } from '../services/goldPriceService';

// كل دقيقة
cron.schedule('* * * * *', async () => {
  await getGoldPrice();
});
```

5. **WebSocket للبث المباشر:**
   - استخدم Socket.io لبث تحديثات الأسعار للمستخدمين المتصلين

```typescript
io.on('connection', (socket) => {
  socket.on('subscribe-gold-prices', () => {
    // Send price updates every 60s
  });
});
```

معايير القبول:

- ✅ تسترجع الأسعار من Metals-API API بنجاح
- ✅ يعمل Cache (لا يستدعي API أكثر من مرة كل 60 ثانية)
- ✅ الأسعار تُحفظ في `GoldPriceHistory` table
- ✅ Endpoints تعيد Response صحيحة
- ✅ WebSocket يبث التحديثات

المرحلة 3: Product Management APIs

المهمة: تطوير APIs لإدارة منتجات الذهب (CRUD operations).

الخطوات:

1. **أنشئ Controller:** `apps/api/src/controllers/goldProductController.ts`
2. **طور الـ Endpoints التالية:** **POST /api/gold/products** - إضافة منتج جديد

```typescript
async function createGoldProduct(req, res) {
  // 1. Validate input
  // 2. Calculate initial price using goldPriceService
  // 3. Set verificationStatus to PENDING
  // 4. Create product in database
  // 5. Return product with auto-calculated prices
}
```

GET /api/gold/products - قائمة المنتجات مع فلترة

```typescript
async function getGoldProducts(req, res) {
  // Support filters:
  // - karat (K24, K21, K18, K14)
  // - type (BULLION, COIN, JEWELRY)
  // - condition (NEW, EXCELLENT, GOOD, FAIR, SCRAP)
  // - minWeight, maxWeight
  // - minPrice, maxPrice
  // - verificationStatus
  //
  // Support sorting:
  // - price_asc, price_desc
  // - weight_asc, weight_desc
  // - created_asc, created_desc
  //
  // Support pagination:
  // - page, limit
}
```

GET /api/gold/products/:id - تفاصيل منتج واحد

```typescript
async function getGoldProduct(req, res) {
  // Include:
  // - Product details
  // - Seller info (name, rating, level)
  // - XRF test results (if verified)
  // - Certificate info
  // - Category
  // - Reviews
}
```

PUT /api/gold/products/:id - تعديل منتج

```typescript
async function updateGoldProduct(req, res) {
  // Only allow seller or admin
  // Recalculate price if weight/karat changed
}
```

DELETE /api/gold/products/:id - حذف منتج

```typescript
async function deleteGoldProduct(req, res) {
  // Soft delete (set isAvailable = false)
  // Only allow if no active orders
}
```

3. **أضف Validation Middleware:**
   - استخدم `zod` للـ validation

```typescript

```

```
import { z } from 'zod';

const createGoldProductSchema = z.object({
  type: z.enum(['BULLION', 'COIN', 'JEWELRY', 'CUSTOM']),
  karat: z.enum(['K24', 'K21', 'K18', 'K14']),
  condition: z.enum(['NEW', 'EXCELLENT', 'GOOD', 'FAIR', 'SCRAP']),
  grossWeight: z.number().positive(),
  netWeight: z.number().positive(),
  workmanshipFee: z.number().nonnegative(),
  // ... more fields
});
```

4. **أضف Authorization:**
   - فقط المستخدمين المسجلين يمكنهم إضافة منتجات
   - يمكنهم التعديل/الحذف Admin فقط صاحب المنتج أو

معايير القبول:

- ✅ جميع الـ CRUD operations تعمل
- ✅ Validation يعمل بشكل صحيح
- ✅ Filtering & Sorting & Pagination تعمل
- ✅ Authorization محكمة
- ✅ السعر يُحسب تلقائياً عند الإنشاء

المرحلة 4: Pricing Calculator Service

المهمة: تطوير خدمة حساب الأسعار الديناميكي بدقة عالية.

الخطوات:

1. **أنشئ `apps/api/src/services/pricingService.ts`:**
   - انسخ دالة `calculateGoldPrice()` (القسم 5.1) من حزمة التطوير
   - انسخ دالة `calculateBuybackPrice()`
2. **أضف API Endpoint: POST /api/gold/prices/calculate**

```
typescript
```

```
async function calculatePrice(req, res) {
  const {
    karat,
    grossWeight,
    netWeight,
    workmanshipFeePerGram,
    hasGems,
    gemsValue,
    condition,
    includeVAT
  } = req.body;

  const result = await calculateGoldPrice({...});

  res.json({ success: true, data: result });
}
```

3. **أضف Unit Tests:**

```
typescript

// tests/services/pricingService.test.ts
describe('Pricing Service', () => {
  test('calculates correct price for 10g 21K jewelry', async () => {
    // Test from dev package section 6.1
  });

  test('applies condition discount correctly', async () => {
    // ...
  });
});
```

معايير القبول:

- ✅ الحسابات دقيقة 100%
- ✅ جميع المكونات محسوبة (ذهب، مصنعية، ضرائب، دمغة)
- ✅ الخصومات تُطبق حسب الحالة
- ✅ Unit Tests تمر بنجاح

المرحلة 5: XRF Verification System

المهمة: نظام التحقق من أصالة الذهب باستخدام XRF.

الخطوات:

1. **أنشئ `apps/api/src/services/xrfService.ts`:**
   - للآن، استخدم Mock Data (سيتم التكامل لاحقاً مع الجهاز الفعلي)

```typescript
async function performXRFTest(productId: string) {
  // Mock XRF test results
  return {
    goldPercentage: 87.5,  // For 21K
    silverPercentage: 10.2,
    copperPercentage: 2.3,
    testDuration: 120  // seconds
  };
}
```

2. **أنشئ Verification Endpoint: POST /api/gold/verify/xrf**

```typescript
async function verifyXRF(req, res) {
  // 1. Check user is technician or admin
  // 2. Get product
  // 3. Perform XRF test (or use mock)
  // 4. Calculate actual karat from gold percentage
  // 5. Compare with expected karat (tolerance ±0.5)
  // 6. Save XRFTestResult
  // 7. Update product verificationStatus
  // 8. Generate certificate if verified
  // 9. Return result
}
```

3. **Certificate Generation:**
   - أنشئ `apps/api/src/services/certificateService.ts`
   - استخدم `qrcode` package لإنشاء QR Code
   - استخدم `pdfkit` لإنشاء PDF Certificate

```typescript
async function generateCertificate(productId: string) {
  // 1. Generate unique certificate number
  // 2. Create QR code with verification URL
  // 3. Generate PDF certificate
  // 4. Upload to cloud storage
  // 5. Save to AuthenticationCertificate table
  // 6. Return certificate number
}
```

4. **Certificate Verification Endpoint: GET /api/gold/verify/certificate/:certificateNumber**

معايير القبول:

- ✅ XRF verification endpoint يعمل
- ✅ Certificate يتم إنشاؤه تلقائياً
- ✅ QR Code يحتوي على بيانات صحيحة
- ✅ PDF certificate احترافي
- ✅ Verification endpoint يعيد بيانات دقيقة

⭐ المرحلة 6: Barter System (نظام المقايضة)

المهمة: تطوير نظام المقايضة الفريد - الميزة التنافسية الأساسية!

الخطوات:

1. أنشئ **`apps/api/src/services/barterService.ts`**:
   - انسخ `findBarterMatches()` من حزمة التطوير
   - انسخ `calculateBarterPriceDifference()`
2. **Barter Endpoints: POST /api/gold/barter/create** - إنشاء عرض مقايضة

```typescript
```

```
async function createBarter(req, res) {
  const {
    barterType,          // "GOLD_TO_MOBILE"
    goldItemsOffered,    // ["gold-prod-1", "gold-prod-2"]
    seekingItemType,     // "MOBILE"
    seekingItemId,       // Optional specific item
    seekingValue,        // 35000 EGP
    seekingDescription,  // "iPhone 15 Pro 256GB"
    expiresInDays        // 7
  } = req.body;

  // 1. Validate user owns all gold items
  // 2. Calculate total gold value
  // 3. Create BarterListing
  // 4. Set expiry date
  // 5. Return listing
}
```

GET /api/gold/barter/matches - العثور على مطابقات

typescript

```
async function findMatches(req, res) {
  // Use barterService.findBarterMatches()
  // Return sorted by matchScore (highest first)
}
```

POST /api/gold/barter/:id/accept - قبول عرض

typescript

```
async function acceptBarter(req, res) {
  // 1. Validate counterparty owns the item
  // 2. Update barter status to ACCEPTED
  // 3. Initiate Escrow for both items
  // 4. Schedule inspection
  // 5. Return next steps
}
```

POST /api/gold/barter/:id/negotiate - التفاوض

typescript

```
async function negotiate(req, res) {
  // 1. Create BarterNegotiation record
  // 2. Update barter status to NEGOTIATING
  // 3. Notify other party
  // 4. Return negotiation details
}
```

3. **Matching Algorithm:**
    - راجع القسم 5.2 في حزمة التطوير
    - تأكد من حساب `matchScore` بشكل صحيح:
        - 60% من التطابق السعري
        - 30% من سمعة المستخدم
        - 10% بونص للمستخدمين الموثقين

معايير القبول:

- ✅ يمكن إنشاء barter listings
- ✅ Matching algorithm يعيد نتائج مرتبة
- ✅ Accept/Negotiate endpoints تعمل
- ✅ Escrow يتم تفعيله تلقائياً
- ✅ Integration يعمل (Mobile/Car) مع أسواق أخرى

المرحلة 7: Gold Savings Program

المهمة: برنامج الادخار الشهري - شراء الذهب بالتقسيط.

الخطوات:

1. **Savings Endpoints: POST /api/gold/savings/plans** - إنشاء خطة

```
typescript
```

```
async function createSavingsPlan(req, res) {
  const { targetGrams, targetKarat, monthlyAmount } = req.body;

  // 1. Get current gold price
  // 2. Calculate estimated completion months
  // 3. Create GoldSavingsPlan
  // 4. Return plan details
}
```

POST /api/gold/savings/plans/:id/pay - إضافة دفعة

```
typescript

async function addPayment(req, res) {
  const { amount, paymentMethod } = req.body;

  // 1. Process payment
  // 2. Get current gold price
  // 3. Calculate grams added (amount / price)
  // 4. Update plan progress
  // 5. Create SavingsPayment record
  // 6. Check if target reached
  // 7. Return updated plan
}
```

GET /api/gold/savings/plans/:id - تفاصيل الخطة

```
typescript

async function getSavingsPlan(req, res) {
  // Include:
  // - Progress (current grams, target, %)
  // - Payment history
  // - Current value at today's price
  // - Unrealized gain/loss
  // - Estimated completion date
}
```

2. **Auto-debit Integration:**
   - تكامل مع بوابات الدفع المصرية (Paymob/Fawry)
   - Cron job شهري لخصم المبلغ تلقائياً

معايير القبول:

- ✅ يمكن إنشاء خطط ادخار
- ✅ الدفعات تُضاف بنجاح
- ✅ الجرامات تُحسب بدقة
- ✅ Progress يُحدث في real-time
- ✅ Unrealized gain/loss محسوب

المرحلة 8: Buyback System

المهمة: نظام إعادة شراء الذهب من المستخدمين.

الخطوات:

1. **Buyback Endpoints: POST /api/gold/buyback/quote** - طلب عرض سعر

```
typescript

  async function getBuybackQuote(req, res) {
    const {
      karat,
      weight,
      condition,
      hasDamgha,
      hasOriginalReceipt,
      photos
    } = req.body;

    // 1. Use calculateBuybackPrice() service
    // 2. Generate quote ID
    // 3. Set 24h expiry
    // 4. Return price range
  }
```

POST /api/gold/buyback/:quoteId/accept - قبول العرض

```
typescript
```

```
async function acceptBuyback(req, res) {
  // 1. Validate quote still valid
  // 2. Schedule XRF verification appointment
  // 3. Return appointment details
}
```

2. **AI Photo Analysis (Optional - مستقبلي):**
   - استخدام Vision API لتقدير الحالة من الصور
   - للآن، اعتمد على input المستخدم

معايير القبول:

- ✅ Quote generation يعمل
- ✅ السعر محسوب بدقة
- ✅ 24h expiry يُفرض
- ✅ Appointment scheduling يعمل

المرحلة 9: Frontend Development

المهمة: بناء واجهة المستخدم باستخدام Next.js 14.

الصفحات المطلوبة:

1. **Gold Marketplace Home** (`/gold`)
   - عرض الفئات (سبائك، مشغولات، جنيهات)
   - أسعار الذهب الحية (real-time WebSocket)
   - Featured products
   - Search bar
2. **Product Listing** (`/gold/products`)
   - Grid/List view toggle
   - Filters sidebar (karat, type, price, weight)
   - Sorting options
   - Pagination

3. **Product Detail** (`/gold/products/[id]`)
    - Image gallery
    - Full specifications
    - Price breakdown table
    - XRF verification badge
    - Certificate download button
    - Seller info
    - Reviews section
    - "Buy Now" / "Add to Cart" buttons
    - "Create Barter" button
4. **Create Barter** (`/gold/barter/create`)
    - Select gold items to offer
    - Choose what you're seeking (Mobile/Car/Scrap/Gold)
    - Set target value
    - Preview matches
    - Submit listing
5. **Barter Matches** (`/gold/barter/matches`)
    - List of matching offers
    - Match score indicator
    - Price difference calculator
    - Accept/Negotiate buttons
6. **Savings Program** (`/gold/savings`)
    - Create plan form
    - Active plans dashboard
    - Payment history
    - Progress charts (Chart.js)
7. **Sell Gold** (`/gold/sell`)
    - Upload photos
    - Enter details (karat, weight, condition)
    - Get instant quote

- Schedule verification

UI Components المطلوبة:

```typescript
// components/gold/GoldPriceTicker.tsx
// Real-time price display using WebSocket

// components/gold/ProductCard.tsx
// Reusable product card with image, price, karat badge

// components/gold/KaratBadge.tsx
// Visual badge for karat (24K, 21K, etc.)

// components/gold/VerificationBadge.tsx
// Shows XRF verified status

// components/gold/PriceBreakdown.tsx
// Table showing price components (gold + workmanship + tax)

// components/gold/BarterMatchCard.tsx
// Card showing barter match with score

// components/gold/SavingsProgress.tsx
// Progress bar for savings plan
```

معايير القبول:

- ✅ جميع الصفحات responsive
- ✅ Arabic RTL support
- ✅ Real-time price updates
- ✅ Dark/Light mode support
- ✅ Loading states
- ✅ Error handling
- ✅ Accessibility (a11y)

المرحلة 10: Testing & Quality Assurance

المهمة: اختبارات شاملة لضمان الجودة.

أنواع الاختبارات:

1. **Unit Tests:**

```bash
# Test all services
npm run test:unit

# Coverage should be >80%
npm run test:coverage
```

2. **Integration Tests:**

```bash
# Test API endpoints
npm run test:integration
```

3. **E2E Tests:**
   - استخدم Playwright أو Cypress
   - اختبر User Journeys الكاملة:
     - شراء سبيكة ذهب
     - إنشاء مقايضة
     - برنامج ادخار
4. **Load Testing:**
   - استخدم k6 أو Artillery
   - اختبر 1000 concurrent users
   - تأكد من استجابة <500ms

معايير القبول:

- ✅ Unit test coverage >80%

- ✅ جميع Integration tests تمر
- ✅ E2E tests للـ critical paths تمر
- ✅ No performance regressions

🔐 متطلبات الأمان والجودة

Security Checklist:

- [ ] Authentication محمية بـ جميع الـ endpoints
- [ ] Authorization محكمة (RBAC)
- [ ] Input validation على كل endpoint
- [ ] SQL Injection prevention (Prisma handles this)
- [ ] XSS prevention (sanitize inputs)
- [ ] Rate limiting على Pricing API
- [ ] Sensitive data encrypted (Damgha numbers, certificates)
- [ ] Audit logging للعمليات الحرجة (XRF verification, بيع/شراء)
- [ ] HTTPS only في production
- [ ] Environment variables محمية

Code Quality:

- [ ] TypeScript strict mode enabled
- [ ] ESLint configured and passing
- [ ] Prettier for code formatting
- [ ] No console.logs في production code
- [ ] Proper error handling (try/catch)
- [ ] Meaningful variable names
- [ ] Code comments للأجزاء المعقدة
- [ ] Follow DRY principle
- [ ] Modular code (single responsibility)

Performance:

- [ ] Database queries optimized (use indexes)
- [ ] N+1 query problem avoided
- [ ] Caching strategy implemented (Redis)
- [ ] Image optimization (Next.js Image component)
- [ ] Lazy loading للمكونات الثقيلة

- [ ] Code splitting
- [ ] Bundle size <500KB (gzipped)

📝 معايير قبول المشروع النهائية

قبل اعتبار المشروع "مكتمل"، تأكد من:

✅ Functional Requirements:

1. يمكن للمستخدم تصفح منتجات الذهب مع فلترة متقدمة
2. الأسعار تُحدث كل 60 ثانية من Metals-API
3. نظام XRF Verification يعمل بشكل كامل
4. شهادات الأصالة تُنشأ تلقائياً مع QR Code
5. نظام المقايضة يربط بين Gold ↔ Mobile ↔ Car
6. برنامج الادخار يسمح بالدفعات الشهرية
7. نظام Buyback يقدم عروض أسعار دقيقة
8. جميع الحسابات السعرية دقيقة 100%
9. Integration مع Payment Gateways المصرية
10. Arabic language support كامل

✅ Non-Functional Requirements:

1. Response time <500ms للصفحات
2. Database query time <100ms
3. Uptime >99.9%
4. Security vulnerabilities = 0
5. Test coverage >80%
6. Mobile responsive (100% من الصفحات)
7. Accessibility score >90 (Lighthouse)
8. SEO optimized

🚀 أوامر مفيدة

```bash
# Development
npm run dev            # Start dev server
npm run dev:api        # Start API only
npm run dev:web        # Start Next.js only

# Database
npm run db:migrate     # Run migrations
npm run db:seed        # Seed database
npm run db:studio      # Open Prisma Studio
npm run db:reset       # Reset database

# Testing
npm run test           # Run all tests
npm run test:unit      # Unit tests only
npm run test:e2e       # E2E tests
npm run test:coverage  # Coverage report

# Build
npm run build          # Build for production
npm run start          # Start production server

# Code Quality
npm run lint           # Run ESLint
npm run format         # Format with Prettier
npm run type-check     # TypeScript type checking
```

📊 تتبع التقدم

أنشئ ملف `PROGRESS.md` وحدثه بعد كل مرحلة:

```markdown
```

```
# Gold Marketplace Development Progress

## Phase 1: Database & Core Models
- [x] Prisma schema added
- [x] Migrations created
- [x] Seed data working
- [x] All relationships tested

## Phase 2: Gold Pricing Service
- [x] Metals-API integration
- [x] Caching implemented
- [x] Pricing endpoints
- [x] WebSocket real-time updates
- [ ] Price alert notifications

## Phase 3: Product Management
- [x] CRUD operations
- [x] Filtering & sorting
- [x] Validation
- [ ] Image upload

...
```

💡 نصائح مهمة

1. **اقرأ حزمة التطوير أولاً:**
   - لا تبدأ الكود قبل فهم كامل الـ architecture
   - استخدم الأمثلة كـ reference
2. **اتبع المراحل بالترتيب:**
   - لا تقفز للمرحلة 6 قبل إتمام المراحل 1-5
   - كل مرحلة تبني على السابقة
3. **اختبر باستمرار:**
   - لا تكتب 1000 سطر ثم تختبر
   - اختبر بعد كل feature صغير
4. **استخدم TypeScript بشكل صحيح:**
   - Define types لكل شيء

- لا تستخدم `any`

5. **Documentation:**
   - اكتب JSDoc comments للدوال المعقدة
   - حدث README.md باستمرار
6. **Git Commits:**
   - Commit messages واضحة
   - Commit بعد كل feature/fix
7. **استفد من الكود الموجود:**
   - راجع كيف تم تطوير Mobile/Scrap marketplaces
   - Reuse components و services

🎯 الأولويات

High Priority (افعلها أولاً):

1. Database Schema
2. Gold Pricing Service
3. Product Management APIs
4. Pricing Calculator
5. XRF Verification

Medium Priority:

6. Barter System
7. Savings Program
8. Buyback System
9. Frontend

Low Priority (يمكن تأجيلها):

10. Insurance System

11. Advanced Analytics
12. Blockchain integration

❓ في حالة مواجهة مشاكل

1. **راجع حزمة التطوير** - الحل غالباً موجود
2. **تحقق من الأمثلة** في Marketplaces الأخرى
3. **اقرأ Prisma/Next.js documentation**
4. **اسألني** إذا احتجت توضيح

🏁 الخلاصة

أنت تبني **أفضل منصة لتجارة الذهب الإلكترونية في مصر**.

المميزات الفريدة:

- ✨ نظام مقايضة متعدد الفئات (لا يوجد لدى أي منافس)
- ✨ XRF verification إلزامي (أعلى مستوى ثقة)
- ✨ شفافية تسعير كاملة (breakdown لكل مكون)
- ✨ برنامج ادخار ذهبي (فريد في السوق)
- ✨ ضمان إعادة شراء (لا خسارة مصنعية)

تذكر:

- اتبع المراحل بالترتيب
- اقرأ حزمة التطوير باستمرار
- اختبر كل feature
- اكتب كود نظيف وموثق

Good luck! 🚀