```
// .env.local COINGECKO_API_KEY=CG-Yqmsms6TW3PiBUd6V3SdEbQL// package.json {
"name": "midas-x-mvp", "version": "1.0.0", "private": true, "scripts": { "dev": "next dev", "build":
"next build", "start": "next start" }, "dependencies": { "axios": "^1.7.2", "@solana/web3.js":
"^1.95.3", "next": "^15.2.4", "react": "^18.2.0", "react-dom": "^18.2.0", "tailwindcss": "^3.4.0" },
"devDependencies": { "postcss": "^8.4.38", "autoprefixer": "^10.4.19" } }// tsconfig.json {
"compilerOptions": { "baseUrl": "src", "paths": { "@/lib/": ["lib/"] }, "target": "es5", "module":
"esnext", "jsx": "preserve", "strict": true } }// tailwind.config.js module.exports = { content:
["./src/**/*.{js,jsx,ts,tsx}"], theme: { extend: {} }, plugins: [] };// src/lib/trading.js export const
calculateSignals = (prices, buyThresholds = [0.001, 0.002, 0.005, 0.008], sellTrigger = 0.40) => {
const signals = []; let holding = false; let entryPrice = 0; let trades = 0;for (let i = 1; i <
prices.length; i++) { const price = prices[i].current_price; const prevPrice = prices[i -
1].current_price; const priceChange = (price - prevPrice) / prevPrice;if} return signals; };//
src/pages/api/coin-prices.js import axios from 'axios';export default async function handler(req,
res) { if (req.method === 'GET') { try { const response = await
axios.get('https://api.coingecko.com/api/v3/coins/markets', { params: { vs_currency: 'usd', ids:
'bitcoin,ethereum', x_cg_demo_api_key: process.env.COINGECKO_API_KEY, }, });
res.status(200).json({ success: true, prices: response.data }); } catch (error) {
res.status(500).json({ error: 'Failed to fetch prices' }); } } else { res.status(405).json({ error:
'Method not allowed' }); } }// src/pages/api/mint-get.js import { Connection, Keypair } from
'@solana/web3.js';export default async function handler(req, res) { if (req.method === 'POST') {
const { user, joules } = req.body; if (joules >= 100) { const gets = joules / 100; const connection
= new Connection('https://api.devnet.solana.com'); const keypair = Keypair.generate();
res.status(200).json({ success: true, gets, value: gets * 10, mint: keypair.publicKey.toString() }); }
else { res.status(400).json({ error: 'Need 100 J' }); } } else { res.status(405).json({ error: 'Method
not allowed' }); } }// src/pages/index.js import { useState, useEffect } from 'react'; import {
calculateSignals } from '@/lib/trading';export default function Dashboard() { const [prices,
setPrices] = useState([]); const [signals, setSignals] = useState([]); const [getBalance,
setGetBalance] = useState(0); const [capital, setCapital] = useState(1000); const [trades,
setTrades] = useState([]);const fetchPrices = async () => { const res = await
fetch('/api/coin-prices'); const data = await res.json(); if (data.success) { setPrices((prev) => {
const newPrices = [...prev, ...data.prices].slice(-100); // Keep last 100 prices const btcPrices =
newPrices.filter((p) => p.id === 'bitcoin'); const newSignals = calculateSignals(btcPrices);
setSignals(newSignals); updateTrades(newSignals, btcPrices); return newPrices; }); } };const
mintGet = async () => { const res = await fetch('/api/mint-get', { method: 'POST', headers: {
'Content-Type': 'application/json' }, body: JSON.stringify({ user: 'randall', joules: 100 }), }); const
data = await res.json(); if (data.success) setGetBalance(getBalance + data.gets); };const
updateTrades = (signals, prices) => { let currentCapital = capital; const newTrades =
[];signals};useEffect(() => { fetchPrices(); const interval = setInterval(fetchPrices, 60000); //
Update every 60s return () => clearInterval(interval); }, []);return (  Midas-x MVP Dashboard
Crypto Prices (CoinGecko)  {prices.slice(-2).map((coin) => (  {coin.name}: ${coin.current_price}
({coin.price_change_percentage_24h.toFixed(2)}% 24h)  ))}    GET Balance {getBalance} GET
($10/GET = ${getBalance * 10})  Mint GET (100 J)    Trades (1,000x ROI Potential) Capital:
${capital.toFixed(2)}  {trades.map((trade) => (  {trade.type} @ ${trade.price.toFixed(2)}{' '}
{trade.profit ? (Profit: $${trade.profit.toFixed(2)}) : (${trade.units.toFixed(4)} units)}  ))}     ); }
```