PRD.md - 차트 모듈 (상세 코드 예제 포함)

1. 프로젝트 개요

1.1 프로젝트 배경

DeltaX는 금 기반 토큰(PAX Gold)과 비트코인의 변동성 대비 상승률을 비교하여 사용자가 직관적으로 참여할 수 있는 베팅형 분석 플랫폼입니다. 차트 모듈은 이러한 비교 분석의 핵심 시각화 도구로서, 사용자의 의사결정을 돕는 중요한 역할을 수행합니다.

1.2 차트 모듈의 역할

- 실시간 가격 데이터 시각화
- 변동성 지표 계산 및 표시
- 베팅 시스템과의 데이터 연동
- 다중 암호화폐 비교 분석 지원

---

2. 핵심 기능 요구사항

2.1 기본 차트 기능

2.1.1 실시간 가격 차트

- **PAXG(금)와 BTC(비트코인) 가격 추적**
  - 실시간 가격 업데이트 (WebSocket 연동)
  - 시간대별 필터링: 1시간, 24시간, 7일, 30일, 전체
  - 캔들스틱 차트 또는 라인 차트 지원

2.1.2 변동성 지표

- **기존 변동성 대비 상승률 계산**
  - 표준편차 기반 변동성 측정
  - 이동평균 대비 변동률
  - 볼린저 밴드 (추가)

- ATR(Average True Range) 지표 (추가)

2.1.3 비교 시각화

- **듀얼 차트 뷰**
  - PAXG vs BTC 나란히 비교
  - 오버레이 모드: 정규화된 데이터로 한 차트에 표시
  - 상승률/변동성 델타 하이라이트

2.2 확장 기능

2.2.1 다중 암호화폐 지원

- **비교 가능한 자산 추가**
  - ETH(이더리움)
  - SOL(솔라나)
  - USDT/USDC (스테이블코인 비교용)
  - 사용자 정의 토큰 추가 기능 (추가)

2.2.2 고급 분석 도구

- **기술적 분석 지표** (추가)
  - RSI (Relative Strength Index)
  - MACD (Moving Average Convergence Divergence)
  - 거래량 분석
  - 변동성 히트맵

2.2.3 예측 모델 시각화 (추가)

- **AI 기반 가격 예측 표시**
  - 단기 예측 라인 (옵션)
  - 신뢰 구간 표시
  - 과거 예측 정확도 시각화

---

3. 베팅 시스템 연동

3.1 실시간 데이터 제공

- **베팅 기준 시점 데이터**
    - 베팅 시작 시점의 가격 스냅샷
    - 현재 가격과의 비교
    - 변동성 지표 전달

3.2 베팅 결과 시각화

- **차트 상 베팅 마커 표시**
    - 사용자의 베팅 시점 표시
    - 베팅 결과 (승/패) 색상 구분
    - 베팅 통계 오버레이 (추가)

3.3 라이브 베팅 현황 (추가)

- **실시간 베팅 분포 표시**
    - PAXG vs BTC 베팅 비율 차트
    - 배당률 변화 그래프
    - 베팅 풀 크기 시각화

---

4. 기술 스펙

4.1 프론트엔드

typescript

```typescript
// 기술 스택
- React 18+
- Next.js 14+ (App Router)
- TypeScript
- Chart Library: Recharts / TradingView Lightweight Charts
- State Management: Zustand (useChartStore)
- Styling: Tailwind CSS + shadcn/ui
- WebSocket: socket.io-client
```

4.2 데이터 소싱

typescript

```typescript
// API 엔드포인트
- 실시간 가격: WebSocket (Binance API, CoinGecko)
- 과거 데이터: REST API
- PAXG 데이터: Paxos API / CoinGecko
- BTC 데이터: Binance / CoinGecko
```

4.3 백엔드 API

typescript

```typescript
// pages/api/chart/ 구조
pages/api/chart/
├── realtime.ts         // WebSocket 핸들러
├── historical.ts       // GET: 과거 데이터
├── volatility.ts       // GET: 변동성 계산
├── compare.ts          // GET: 다중 자산 비교
└── [id].ts            // GET: 특정 자산 상세
```

---

5. 데이터 모델 및 타입 정의

5.1 Core Types

typescript

```typescript
// types/chart.ts

/**
 * 지원하는 자산 타입
 */
export type AssetType = 'PAXG' | 'BTC' | 'ETH' | 'SOL' | 'USDT' |
'USDC';

/**
 * 시간 범위 타입
 */
export type TimeRange = '1h' | '24h' | '7d' | '30d' | 'all';

/**
 * 차트 타입
 */
export type ChartType = 'candlestick' | 'line' | 'area';

/**
 * 차트 뷰 모드
 */
export type ViewMode = 'dual' | 'overlay' | 'single';

/**
 * 기본 가격 데이터
 */
export interface PriceData {
  asset: AssetType;
  timestamp: number;
  price: number;
  volume: number;
  change24h: number;
  changePercent24h: number;
}

/**
 * 캔들스틱 데이터
 */
export interface CandlestickData {
  asset: AssetType;
  timestamp: number;
  open: number;
  high: number;
  low: number;
  close: number;
  volume: number;
}

/**
 * 변동성 지표
 */
export interface VolatilityMetrics {
  asset: AssetType;
  timestamp: number;

  // 기본 지표
  stdDev: number;                // 표준편차
  percentChange: number;         // 변동률

  // 고급 지표
  atr?: number;                  // Average True Range
  bollingerBands?: {
    upper: number;
    middle: number;
    lower: number;
    bandwidth: number;           // 밴드폭
  };

  // 추가 지표
  rsi?: number;                  // Relative Strength Index
  macd?: {
    macd: number;
    signal: number;
    histogram: number;
  };
}

/**
 * 비교 분석 데이터
 */
export interface ComparisonData {
  assets: AssetType[];
  timeRange: TimeRange;
  // 정규화된 가격 (0-100 스케일)
  normalizedPrices: {
    [key in AssetType]?: number[];
  };

  // 변동성 비율
  volatilityRatio: number;     // PAXG/BTC

  // 상승률 비교
  priceChangeComparison: {
    [key in AssetType]?: {
      absolute: number;        // 절대값
      percentage: number;      // 퍼센트
      volatilityAdjusted: number; // 변동성 조정 상승률
    };
  };

  // AI 추천 (추가)
  recommendation?: {
    asset: AssetType;
    confidence: number;
    reason: string;
  };
}

/**
 * 베팅 마커 데이터
 */
export interface BettingMarker {
  id: string;
  userId: string;
  asset: AssetType;
  timestamp: number;
  betAmount: number;
  entryPrice: number;
  currentPrice?: number;
  result?: 'win' | 'lose' | 'pending';
  profit?: number;
}
```

```
/**
 * 차트 설정
 */
export interface ChartConfig {
  viewMode: ViewMode;
  chartType: ChartType;
  timeRange: TimeRange;
  selectedAssets: AssetType[];

  // 표시 옵션
  showVolume: boolean;
  showVolatility: boolean;
  showBettingMarkers: boolean;
  showTechnicalIndicators: boolean;

  // 색상 설정
  colors: {
    [key in AssetType]?: string;
  };
}
```

5.2 데이터베이스 스키마

```typescript
// prisma/schema.prisma

model ChartData {
  id           String   @id @default(cuid())
  asset        String   // 'PAXG', 'BTC', etc.
  timestamp    DateTime

  // OHLCV 데이터
  open         Float
  high         Float
  low          Float
  close        Float
  volume       Float
  // 계산된 지표
  volatility  Float?
  rsi         Float?

  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  @@index([asset, timestamp])
  @@unique([asset, timestamp])
}

model VolatilitySnapshot {
  id              String   @id @default(cuid())
  asset           String
  timestamp       DateTime

  // 변동성 지표
  stdDev          Float    // 표준편차
  percentChange   Float    // 변동률
  atr             Float?   // ATR

  // 볼린저 밴드
  bollingerUpper  Float?
  bollingerMiddle Float?
  bollingerLower  Float?

  createdAt       DateTime @default(now())

  @@index([asset, timestamp])
  @@unique([asset, timestamp])
}

model BettingMarker {
  id          String   @id @default(cuid())
  userId      String
  asset       String
  timestamp   DateTime

  betAmount   Float
  entryPrice  Float
  exitPrice   Float?

  result      String?  // 'win', 'lose', 'pending'
  profit      Float?

  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  @@index([userId, asset])
  @@index([timestamp])
}
```

---

6. 상태 관리 (Zustand Store)

6.1 Chart Store

typescript

```typescript
// store/useChartStore.ts

import { create } from 'zustand';
import { devtools, persist } from 'zustand/middleware';
import type {
  AssetType,
  TimeRange,
  ChartType,
  ViewMode,
  PriceData,
  VolatilityMetrics,
  ComparisonData,
  BettingMarker,
  ChartConfig,
} from '@/types/chart';

interface ChartState {
  // 설정
  config: ChartConfig;
  // 실시간 데이터
  realtimeData: Map<AssetType, PriceData>;
  historicalData: Map<AssetType, PriceData[]>;

  // 변동성 데이터
  volatilityData: Map<AssetType, VolatilityMetrics>;

  // 비교 데이터
  comparisonData: ComparisonData | null;

  // 베팅 마커
  bettingMarkers: BettingMarker[];

  // 로딩 상태
  isLoading: boolean;
  error: string | null;

  // WebSocket 연결 상태
  wsConnected: boolean;
  wsReconnecting: boolean;
}

interface ChartActions {
  // 설정 변경
  setViewMode: (mode: ViewMode) => void;
  setChartType: (type: ChartType) => void;
  setTimeRange: (range: TimeRange) => void;
  setSelectedAssets: (assets: AssetType[]) => void;
  toggleVolatility: () => void;
  toggleBettingMarkers: () => void;

  // 데이터 업데이트
  updateRealtimeData: (asset: AssetType, data: PriceData) => void;
  setHistoricalData: (asset: AssetType, data: PriceData[]) => void;
  updateVolatilityData: (asset: AssetType, metrics: VolatilityMetrics)
=> void;
  setComparisonData: (data: ComparisonData) => void;

  // 베팅 마커
  addBettingMarker: (marker: BettingMarker) => void;
  updateBettingMarker: (id: string, updates: Partial<BettingMarker>) =>
void;
  removeBettingMarker: (id: string) => void;

  // 상태 관리
  setLoading: (loading: boolean) => void;
  setError: (error: string | null) => void;
  setWsConnected: (connected: boolean) => void;
  setWsReconnecting: (reconnecting: boolean) => void;

  // 유틸리티
  reset: () => void;
  fetchHistoricalData: (asset: AssetType, range: TimeRange) =>
Promise<void>;
  fetchVolatilityData: (asset: AssetType) => Promise<void>;
  fetchComparisonData: (assets: AssetType[], range: TimeRange) =>
Promise<void>;
}

type ChartStore = ChartState & ChartActions;

const initialState: ChartState = {
  config: {
    viewMode: 'dual',
    chartType: 'candlestick',
    timeRange: '24h',
    selectedAssets: ['PAXG', 'BTC'],
    showVolume: true,
    showVolatility: true,
    showBettingMarkers: true,
    showTechnicalIndicators: false,
    colors: {
      PAXG: '#FFD700',
      BTC: '#F7931A',
      ETH: '#627EEA',
      SOL: '#14F195',
    },
  },
  realtimeData: new Map(),
  historicalData: new Map(),
  volatilityData: new Map(),
  comparisonData: null,
  bettingMarkers: [],
  isLoading: false,
  error: null,
  wsConnected: false,
  wsReconnecting: false,
};

export const useChartStore = create<ChartStore>()(
  devtools(
    persist(
      (set, get) => ({
        ...initialState,

        // 설정 변경
        setViewMode: (mode) =>
          set((state) => ({
            config: { ...state.config, viewMode: mode },
          })),

        setChartType: (type) =>
          set((state) => ({
            config: { ...state.config, chartType: type },
          })),

        setTimeRange: (range) =>
          set((state) => ({
            config: { ...state.config, timeRange: range },
          })),

        setSelectedAssets: (assets) =>
          set((state) => ({
            config: { ...state.config, selectedAssets: assets },
          })),

        toggleVolatility: () =>
          set((state) => ({
            config: {
              ...state.config,
              showVolatility: !state.config.showVolatility,
            },
          })),

        toggleBettingMarkers: () =>
          set((state) => ({
            config: {
              ...state.config,
              showBettingMarkers: !state.config.showBettingMarkers,
            },
          })),

        // 데이터 업데이트
        updateRealtimeData: (asset, data) =>
          set((state) => {
            const newMap = new Map(state.realtimeData);
            newMap.set(asset, data);
            return { realtimeData: newMap };
          }),

        setHistoricalData: (asset, data) =>
          set((state) => {
            const newMap = new Map(state.historicalData);
            newMap.set(asset, data);
            return { historicalData: newMap };
          }),

        updateVolatilityData: (asset, metrics) =>
          set((state) => {
            const newMap = new Map(state.volatilityData);
            newMap.set(asset, metrics);
            return { volatilityData: newMap };
          }),

        setComparisonData: (data) =>
          set({ comparisonData: data }),

        // 베팅 마커
        addBettingMarker: (marker) =>
          set((state) => ({
            bettingMarkers: [...state.bettingMarkers, marker],
          })),

        updateBettingMarker: (id, updates) =>
          set((state) => ({
            bettingMarkers: state.bettingMarkers.map((marker) =>
              marker.id === id ? { ...marker, ...updates } : marker
            ),
          })),

        removeBettingMarker: (id) =>
          set((state) => ({
            bettingMarkers: state.bettingMarkers.filter((m) => m.id !==
  id),
          })),

        // 상태 관리
        setLoading: (loading) => set({ isLoading: loading }),
        setError: (error) => set({ error }),
        setWsConnected: (connected) => set({ wsConnected: connected }),
        setWsReconnecting: (reconnecting) => set({ wsReconnecting:
  reconnecting }),

        // 유틸리티
        reset: () => set(initialState),

        // API 호출
        fetchHistoricalData: async (asset, range) => {
          set({ isLoading: true, error: null });
          try {
            const response = await fetch(
              `/api/chart/historical?asset=${asset}&range=${range}`
            );
            if (!response.ok) throw new Error('Failed to fetch
  historical data');
            const data = await response.json();
            get().setHistoricalData(asset, data);
          } catch (error) {
            set({ error: (error as Error).message });
          } finally {
            set({ isLoading: false });
          }
        },

        fetchVolatilityData: async (asset) => {
          try {
            const response = await fetch(`/api/chart/volatility?
asset=${asset}`);
            if (!response.ok) throw new Error('Failed to fetch
volatility data');
            const data = await response.json();
            get().updateVolatilityData(asset, data);
          } catch (error) {
            console.error('Volatility fetch error:', error);
          }
        },

        fetchComparisonData: async (assets, range) => {
          set({ isLoading: true, error: null });
          try {
            const response = await fetch(
              `/api/chart/compare?
assets=${assets.join(',')}&range=${range}`
            );
            if (!response.ok) throw new Error('Failed to fetch
comparison data');
            const data = await response.json();
            set({ comparisonData: data });
          } catch (error) {
            set({ error: (error as Error).message });
          } finally {
            set({ isLoading: false });
          }
        },
      }),
      {
        name: 'chart-storage',
        partialize: (state) => ({
          config: state.config,
        }),
      }
    )
  )
);
```

7. API 구현

7.1 Historical Data API

typescript

```typescript
// pages/api/chart/historical.ts

import type { NextApiRequest, NextApiResponse } from 'next';
import { prisma } from '@/lib/db';
import type { AssetType, TimeRange, PriceData } from '@/types/chart';

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  if (req.method !== 'GET') {
    return res.status(405).json({ error: 'Method not allowed' });
  }

  try {
    const { asset, range } = req.query;

    if (!asset || !range) {
      return res.status(400).json({ error: 'Missing required parameters'
});
    }

    // 시간 범위 계산
    const endTime = new Date();
    const startTime = getStartTime(range as TimeRange);

    // DB에서 데이터 조회
    const data = await prisma.chartData.findMany({
      where: {
        asset: asset as string,
        timestamp: {
          gte: startTime,
          lte: endTime,
        },
      },
      orderBy: {
        timestamp: 'asc',
      },
    });

    // 응답 포맷 변환
    const formattedData: PriceData[] = data.map((item) => ({
      asset: item.asset as AssetType,
      timestamp: item.timestamp.getTime(),
      price: item.close,
      volume: item.volume,
      change24h: calculateChange24h(item.close, data),
      changePercent24h: calculateChangePercent24h(item.close, data),
    }));

    res.status(200).json(formattedData);
  } catch (error) {
    console.error('Historical data API error:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
}

function getStartTime(range: TimeRange): Date {
  const now = new Date();
  switch (range) {
    case '1h':
      return new Date(now.getTime() - 60 * 60 * 1000);
    case '24h':
      return new Date(now.getTime() - 24 * 60 * 60 * 1000);
    case '7d':
      return new Date(now.getTime() - 7 * 24 * 60 * 60 * 1000);
    case '30d':
      return new Date(now.getTime() - 30 * 24 * 60 * 60 * 1000);
    case 'all':
      return new Date(0); // 전체 기간
    default:
      return new Date(now.getTime() - 24 * 60 * 60 * 1000);
  }
}

function calculateChange24h(currentPrice: number, data: any[]): number {
  if (data.length < 2) return 0;
  const price24hAgo = data[data.length - 24]?.close || data[0].close;
  return currentPrice - price24hAgo;
}

function calculateChangePercent24h(currentPrice: number, data: any[]):
number {
  if (data.length < 2) return 0;
  const price24hAgo = data[data.length - 24]?.close || data[0].close;
  return ((currentPrice - price24hAgo) / price24hAgo) * 100;
}
```

7.2 Volatility Calculation API

typescript

```
// pages/api/chart/volatility.ts

import type { NextApiRequest, NextApiResponse } from 'next';
import { prisma } from '@/lib/db';
import type { AssetType, VolatilityMetrics } from '@/types/chart';

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  if (req.method !== 'GET') {
    return res.status(405).json({ error: 'Method not allowed' });
  }
  try {
    const { asset, period = '24h' } = req.query;

    if (!asset) {
      return res.status(400).json({ error: 'Asset parameter required'
});
    }

    // 최근 데이터 조회
    const data = await prisma.chartData.findMany({
      where: {
        asset: asset as string,
        timestamp: {
          gte: new Date(Date.now() - 24 * 60 * 60 * 1000),
        },
      },
      orderBy: {
        timestamp: 'desc',
      },
      take: 100,
    });

    if (data.length === 0) {
      return res.status(404).json({ error: 'No data found' });
    }

    // 변동성 지표 계산
    const metrics = calculateVolatilityMetrics(data);

    const response: VolatilityMetrics = {
      asset: asset as AssetType,
      timestamp: Date.now(),
      stdDev: metrics.stdDev,
      percentChange: metrics.percentChange,
      atr: metrics.atr,
      bollingerBands: metrics.bollingerBands,
      rsi: metrics.rsi,
      macd: metrics.macd,
    };
    res.status(200).json(response);
  } catch (error) {
    console.error('Volatility API error:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
}

function calculateVolatilityMetrics(data: any[]) {
  const prices = data.map((d) => d.close);

  // 표준편차 계산
  const mean = prices.reduce((a, b) => a + b, 0) / prices.length;
  const variance = prices.reduce((sum, price) => {
    return sum + Math.pow(price - mean, 2);
  }, 0) / prices.length;
  const stdDev = Math.sqrt(variance);

  // 변동률 계산
  const percentChange = ((prices[0] - prices[prices.length - 1]) /
prices[prices.length - 1]) * 100;

  // ATR 계산 (Average True Range)
  const atr = calculateATR(data);

  // 볼린저 밴드 계산
  const bollingerBands = calculateBollingerBands(prices, 20, 2);

  // RSI 계산
  const rsi = calculateRSI(prices, 14);

  // MACD 계산
  const macd = calculateMACD(prices);

  return {
    stdDev,
    percentChange,
    atr,
    bollingerBands,
    rsi,
    macd,
  };
}

function calculateATR(data: any[], period: number = 14): number {
  const trueRanges = data.slice(1).map((item, index) => {
    const prevClose = data[index].close;
    const high = item.high;
    const low = item.low;

    return Math.max(
      high - low,
      Math.abs(high - prevClose),
      Math.abs(low - prevClose)
    );
  });

  const atr = trueRanges.slice(0, period).reduce((a, b) => a + b, 0) /
period;
  return atr;
}

function calculateBollingerBands(prices: number[], period: number,
stdDevMultiplier: number) {
  const sma = prices.slice(0, period).reduce((a, b) => a + b, 0) /
period;
  const variance = prices.slice(0, period).reduce((sum, price) => {
    return sum + Math.pow(price - sma, 2);
  }, 0) / period;
  const stdDev = Math.sqrt(variance);

  return {
    upper: sma + stdDevMultiplier * stdDev,
    middle: sma,
    lower: sma - stdDevMultiplier * stdDev,
    bandwidth: (stdDevMultiplier * stdDev * 2) / sma * 100,
  };
}

function calculateRSI(prices: number[], period: number = 14): number {
  const changes = prices.slice(1).map((price, i) => price - prices[i]);
  const gains = changes.map(change => change > 0 ? change : 0);
  const losses = changes.map(change => change < 0 ? -change : 0);

  const avgGain = gains.slice(0, period).reduce((a, b) => a + b, 0) /
period;
  const avgLoss = losses.slice(0, period).reduce((a, b) => a + b, 0) /
period;

  if (avgLoss === 0) return 100;

  const rs = avgGain / avgLoss;
  const rsi = 100 - (100 / (1 + rs));

  return rsi;
}

function calculateMACD(prices: number[]) {
  const ema12 = calculateEMA(prices, 12);
  const ema26 = calculateEMA(prices, 26);
  const macdLine = ema12 - ema26;
  const signalLine = calculateEMA([macdLine], 9);
  const histogram = macdLine - signalLine;

  return {
    macd: macdLine,
    signal: signalLine,
    histogram,
  };
}

function calculateEMA(prices: number[], period: number): number {
  const multiplier = 2 / (period + 1);
  let ema = prices[0];

  for (let i = 1; i < prices.length; i++) {
    ema = (prices[i] - ema) * multiplier + ema;
  }

  return ema;
}
```

7.3 Comparison API

typescript

```typescript
// pages/api/chart/compare.ts

import type { NextApiRequest, NextApiResponse } from 'next';
import { prisma } from '@/lib/db';
import type { AssetType, TimeRange, ComparisonData } from
'@/types/chart';

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  if (req.method !== 'GET') {
    return res.status(405).json({ error: 'Method not allowed' });
  }

  try {
    const { assets, range = '24h' } = req.query;

    if (!assets) {
      return res.status(400).json({ error: 'Assets parameter required'
});
    }

    const assetList = (assets as string).split(',') as AssetType[];
    const startTime = getStartTime(range as TimeRange);

    // 각 자산의 데이터 조회
    const dataPromises = assetList.map((asset) =>
      prisma.chartData.findMany({
        where: {
          asset,
          timestamp: {
            gte: startTime,
          },
        },
        orderBy: {
          timestamp: 'asc',
        },
      })
    );

    const allData = await Promise.all(dataPromises);

    // 데이터 정규화 (0-100 스케일)
    const normalizedPrices: { [key in AssetType]?: number[] } = {};
    const priceChangeComparison: ComparisonData['priceChangeComparison']
= {};

    assetList.forEach((asset, index) => {
      const data = allData[index];
      if (data.length === 0) return;

      const prices = data.map((d) => d.close);
      const minPrice = Math.min(...prices);
      const maxPrice = Math.max(...prices);

      // 정규화
      normalizedPrices[asset] = prices.map(
        (price) => ((price - minPrice) / (maxPrice - minPrice)) * 100
      );

      // 변동 계산
      const firstPrice = prices[0];
      const lastPrice = prices[prices.length - 1];
      const absolute = lastPrice - firstPrice;
      const percentage = (absolute / firstPrice) * 100;

      // 변동성 조정 상승률
      const volatility = calculateVolatility(prices);
      const volatilityAdjusted = percentage / (volatility || 1);

      priceChangeComparison[asset] = {
        absolute,
        percentage,
        volatilityAdjusted,
      };
    });

    // 변동성 비율 계산 (PAXG/BTC)
    const paxgVolatility = calculateVolatility(
      allData[assetList.indexOf('PAXG')]?.map((d) => d.close) || []
    );
    const btcVolatility = calculateVolatility(
      allData[assetList.indexOf('BTC')]?.map((d) => d.close) || []
    );
    const volatilityRatio = paxgVolatility / (btcVolatility || 1);

    // AI 추천 생성 (간단한 로직)
    const recommendation = generateRecommendation(priceChangeComparison,
volatilityRatio);

    const response: ComparisonData = {
      assets: assetList,
      timeRange: range as TimeRange,
      normalizedPrices,
      volatilityRatio,
      priceChangeComparison,
      recommendation,
    };

    res.status(200).json(response);
  } catch (error) {
    console.error('Comparison API error:', error);
    res.status(500).json({ error: 'Internal server error' });
  }
}

function calculateVolatility(prices: number[]): number {
  if (prices.length < 2) return 0;

  const mean = prices.reduce((a, b) => a + b, 0) / prices.length;
  const variance = prices.reduce((sum, price) => {
    return sum + Math.pow(price - mean, 2);
  }, 0) / prices.length;
  return Math.sqrt(variance);
}

function generateRecommendation(
  comparison: ComparisonData['priceChangeComparison'],
  volatilityRatio: number
): ComparisonData['recommendation'] {
  const paxgData = comparison['PAXG'];
  const btcData = comparison['BTC'];

  if (!paxgData || !btcData) return undefined;

  // 변동성 조정 상승률 비교
  if (paxgData.volatilityAdjusted > btcData.volatilityAdjusted) {
    return {
      asset: 'PAXG',
      confidence: Math.min(
        (paxgData.volatilityAdjusted / btcData.volatilityAdjusted) *
0.5,
        0.95
      ),
      reason: 'PAXG shows higher volatility-adjusted returns',
    };
  } else {
    return {
      asset: 'BTC',
      confidence: Math.min(
        (btcData.volatilityAdjusted / paxgData.volatilityAdjusted) *
0.5,
        0.95
      ),
      reason: 'BTC shows higher volatility-adjusted returns',
    };
  }
}

function getStartTime(range: TimeRange): Date {
  const now = new Date();
  switch (range) {
    case '1h':
      return new Date(now.getTime() - 60 * 60 * 1000);
    case '24h':
      return new Date(now.getTime() - 24 * 60 * 60 * 1000);
    case '7d':
      return new Date(now.getTime() - 7 * 24 * 60 * 60 * 1000);
    case '30d':
      return new Date(now.getTime() - 30 * 24 * 60 * 60 * 1000);
    case 'all':
      return new Date(0);
    default:
      return new Date(now.getTime() - 24 * 60 * 60 * 1000);
  }
}
```

7.4 WebSocket Handler

typescript

```typescript
// pages/api/chart/realtime.ts

import type { NextApiRequest } from 'next';
import type { NextApiResponse } from 'next';
import { Server as SocketIOServer } from 'socket.io';
import type { Server as HTTPServer } from 'http';
import type { Socket as NetSocket } from 'net';
import type { PriceData, AssetType } from '@/types/chart';

interface SocketServer extends HTTPServer {
  io?: SocketIOServer | undefined;
}

interface SocketWithIO extends NetSocket {
  server: SocketServer;
}

interface NextApiResponseWithSocket extends NextApiResponse {
  socket: SocketWithIO;
}
```

```
export default function handler(
  req: NextApiRequest,
  res: NextApiResponseWithSocket
) {
  if (res.socket.server.io) {
    console.log('Socket.io already initialized');
    res.end();
    return;
  }

  console.log('Initializing Socket.io');
  const io = new SocketIOServer(res.socket.server);
  res.socket.server.io = io;

  io.on('connection', (socket) => {
    console.log('Client connected:', socket.id);

    // 자산 구독
    socket.on('subscribe', (assets: AssetType[]) => {
      console.log('Client subscribed to:', assets);
      assets.forEach((asset) => {
        socket.join(`asset:${asset}`);
      });

      // 초기 데이터 전송
      assets.forEach((asset) => {
        fetchAndEmitPriceData(io, asset);
      });
    });

    // 자산 구독 해제
    socket.on('unsubscribe', (assets: AssetType[]) => {
      console.log('Client unsubscribed from:', assets);
      assets.forEach((asset) => {
        socket.leave(`asset:${asset}`);
      });
    });

    socket.on('disconnect', () => {
      console.log('Client disconnected:', socket.id);
    });
  });

  // 실시간 데이터 업데이트 (1초마다)
  setInterval(() => {
    const assets: AssetType[] = ['PAXG', 'BTC', 'ETH', 'SOL'];
    assets.forEach((asset) => {
      fetchAndEmitPriceData(io, asset);
    });
  }, 1000);

  res.end();
}

async function fetchAndEmitPriceData(io: SocketIOServer, asset:
AssetType) {
  try {
    // 실제로는 외부 API에서 데이터를 가져와야 함
    // 여기서는 예시로 랜덤 데이터 생성
    const priceData: PriceData = await fetchPriceFromExternalAPI(asset);

    // 해당 자산을 구독한 클라이언트에게 전송
    io.to(`asset:${asset}`).emit('price-update', priceData);
  } catch (error) {
    console.error(`Error fetching price for ${asset}:`, error);
  }
}

async function fetchPriceFromExternalAPI(asset: AssetType):
Promise<PriceData> {
  // 실제 구현에서는 CoinGecko, Binance 등의 API 사용
  // 예시 구현
  const basePrice = asset === 'PAXG' ? 2000 : asset === 'BTC' ? 45000 :
3000;
  const randomChange = (Math.random() - 0.5) * 100;

  return {
    asset,
    timestamp: Date.now(),
    price: basePrice + randomChange,
    volume: Math.random() * 1000000,
    change24h: randomChange,
    changePercent24h: (randomChange / basePrice) * 100,
  };
}
```

8. 컴포넌트 구현

8.1 Main Chart Container

typescript

```typescript
// components/chart/ChartContainer.tsx

'use client';

import { useEffect } from 'react';
import { useChartStore } from '@/store/useChartStore';
import { ChartHeader } from './ChartHeader';
import { PriceChart } from './PriceChart';
import { VolatilityPanel } from './VolatilityPanel';
import { BettingWidget } from './BettingWidget';
import { useWebSocket } from '@/hooks/useWebSocket';

export function ChartContainer() {
  const {
    config,
    isLoading,
    error,
    fetchHistoricalData,
    fetchComparisonData,
  } = useChartStore();

  const { connect, disconnect } = useWebSocket();

  useEffect(() => {
    // 초기 데이터 로드
    config.selectedAssets.forEach((asset) => {
      fetchHistoricalData(asset, config.timeRange);
    });

    fetchComparisonData(config.selectedAssets, config.timeRange);

    // WebSocket 연결
    connect(config.selectedAssets);

    return () => {
      disconnect();
    };
  }, [config.selectedAssets, config.timeRange]);

  if (error) {
    return (
      <div className="flex items-center justify-center h-96 text-red-
500">
        Error: {error}
      </div>
    );
  }

  return (
    <div className="w-full space-y-4">
      <ChartHeader />

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-4">
        {config.viewMode === 'dual' ? (
          <>
            {config.selectedAssets.map((asset) => (
              <PriceChart key={asset} asset={asset} />
            ))}
          </>
        ) : config.viewMode === 'overlay' ? (
          <div className="lg:col-span-2">
            <PriceChart assets={config.selectedAssets} overlay />
          </div>
        ) : (
          <div className="lg:col-span-2">
            <PriceChart asset={config.selectedAssets[0]} />
          </div>
        )}
      </div>

      {config.showVolatility && <VolatilityPanel />}

      {config.showBettingMarkers && <BettingWidget />}
    </div>
  );
}
```

8.2 Chart Header

```typescript
// components/chart/ChartHeader.tsx

'use client';

import { useChartStore } from '@/store/useChartStore';
import { Button } from '@/components/ui/button';
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from '@/components/ui/select';
import type { AssetType, TimeRange, ViewMode, ChartType } from
'@/types/chart';

const ASSETS: AssetType[] = ['PAXG', 'BTC', 'ETH', 'SOL'];
const TIME_RANGES: TimeRange[] = ['1h', '24h', '7d', '30d', 'all'];
const VIEW_MODES: ViewMode[] = ['dual', 'overlay', 'single'];
const CHART_TYPES: ChartType[] = ['candlestick', 'line', 'area'];

export function ChartHeader() {
  const {
    config,
    setViewMode,
    setChartType,
    setTimeRange,
    setSelectedAssets,
    toggleVolatility,
    toggleBettingMarkers,
  } = useChartStore();

  return (
    <div className="flex flex-wrap items-center justify-between gap-4 p-
4 bg-card rounded-lg border">
      {/* 자산 선택 */}
      <div className="flex items-center gap-2">
        <span className="text-sm font-medium">Assets:</span>
        <div className="flex gap-2">
          {ASSETS.map((asset) => (
            <Button
              key={asset}
              variant={
                config.selectedAssets.includes(asset) ? 'default' :
'outline'
              }
              size="sm"
              onClick={() => {
                const newAssets = config.selectedAssets.includes(asset)
                  ? config.selectedAssets.filter((a) => a !== asset)
                  : [...config.selectedAssets, asset];
                setSelectedAssets(newAssets);
              }}
            >
              {asset}
            </Button>
          ))}
        </div>
      </div>

      {/* 시간 범위 */}
      <div className="flex items-center gap-2">
        <span className="text-sm font-medium">Time:</span>
        <Select
```

```
          value={config.timeRange}
          onValueChange={(value) => setTimeRange(value as TimeRange)}
        >
          <SelectTrigger className="w-[100px]">
            <SelectValue />
          </SelectTrigger>
          <SelectContent>
            {TIME_RANGES.map((range) => (
              <SelectItem key={range} value={range}>
                {range}
              </SelectItem>
            ))}
          </SelectContent>
        </Select>
      </div>

      {/* 뷰 모드 */}
      <div className="flex items-center gap-2">
        <span className="text-sm font-medium">View:</span>
        <Select
          value={config.viewMode}
          onValueChange={(value) => setViewMode(value as ViewMode)}
        >
          <SelectTrigger className="w-[120px]">
            <SelectValue />
          </SelectTrigger>
          <SelectContent>
            {VIEW_MODES.map((mode) => (
              <SelectItem key={mode} value={mode}>
                {mode}
              </SelectItem>
            ))}
          </SelectContent>
        </Select>
      </div>

      {/* 차트 타입 */}
      <div className="flex items-center gap-2">
        <span className="text-sm font-medium">Type:</span>
        <Select
```

```
          value={config.chartType}
          onValueChange={(value) => setChartType(value as ChartType)}
        >
          <SelectTrigger className="w-[140px]">
            <SelectValue />
          </SelectTrigger>
          <SelectContent>
            {CHART_TYPES.map((type) => (
              <SelectItem key={type} value={type}>
                {type}
              </SelectItem>
            ))}
          </SelectContent>
        </Select>
      </div>

      {/* 토글 옵션 */}
      <div className="flex gap-2">
        <Button
          variant={config.showVolatility ? 'default' : 'outline'}
          size="sm"
          onClick={toggleVolatility}
        >
          Volatility
        </Button>
        <Button
          variant={config.showBettingMarkers ? 'default' : 'outline'}
          size="sm"
          onClick={toggleBettingMarkers}
        >
          Bets
        </Button>
      </div>
    </div>
  );
}
```

8.3 Price Chart Component

typescript

```typescript
// components/chart/PriceChart.tsx

'use client';

import { useMemo } from 'react';
import {
  LineChart,
  Line,
  AreaChart,
  Area,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer,
  ReferenceLine,
} from 'recharts';
import { useChartStore } from '@/store/useChartStore';
import type { AssetType } from '@/types/chart';
import { formatPrice, formatTimestamp } from '@/lib/utils';

interface PriceChartProps {
  asset?: AssetType;
  assets?: AssetType[];
  overlay?: boolean;
}

export function PriceChart({ asset, assets, overlay = false }:
PriceChartProps) {
  const {
    config,
    historicalData,
    realtimeData,
    bettingMarkers,
  } = useChartStore();

  const displayAssets = overlay ? assets || [] : asset ? [asset] : [];
  const chartData = useMemo(() => {
    if (displayAssets.length === 0) return [];

    const firstAsset = displayAssets[0];
    const firstData = historicalData.get(firstAsset) || [];

    return firstData.map((point, index) => {
      const dataPoint: any = {
        timestamp: point.timestamp,
        time: formatTimestamp(point.timestamp),
      };

      displayAssets.forEach((assetKey) => {
        const assetData = historicalData.get(assetKey);
        if (assetData && assetData[index]) {
          dataPoint[assetKey] = assetData[index].price;
        }
      });

      return dataPoint;
    });
  }, [displayAssets, historicalData]);

  const ChartComponent = config.chartType === 'area' ? AreaChart :
LineChart;

  return (
    <div className="p-4 bg-card rounded-lg border">
      <div className="mb-4">
        <h3 className="text-lg font-semibold">
          {overlay
            ? `${displayAssets.join(' vs ')} Comparison`
            : `${asset} Price Chart`}
        </h3>
        {!overlay && asset && realtimeData.has(asset) && (
          <div className="flex items-center gap-4 mt-2 text-sm">
            <span className="font-medium">
              ${formatPrice(realtimeData.get(asset)!.price)}
            </span>
            <span
              className={
                realtimeData.get(asset)!.changePercent24h >= 0
                  ? 'text-green-500'
                  : 'text-red-500'
              }
            >
              {realtimeData.get(asset)!.changePercent24h >= 0 ? '+' :
''}
              {realtimeData.get(asset)!.changePercent24h.toFixed(2)}%
            </span>
          </div>
        ))}
      </div>

      <ResponsiveContainer width="100%" height={400}>
        <ChartComponent data={chartData}>
          <CartesianGrid strokeDasharray="3 3" />
          <XAxis
            dataKey="time"
            tick={{ fontSize: 12 }}
            tickFormatter={(value) => value.split(' ')[1] || value}
          />
          <YAxis
            tick={{ fontSize: 12 }}
            tickFormatter={(value) => `$${formatPrice(value)}`}
          />
          <Tooltip
            content={({ active, payload }) => {
              if (!active || !payload || payload.length === 0) return
null;
              return (
                <div className="bg-background p-3 border rounded-lg
shadow-lg">
                  <p className="text-sm font-medium mb-2">
                    {payload[0].payload.time}
                  </p>
                  {payload.map((entry: any) => (
                    <p
                      key={entry.dataKey}
                      className="text-sm"
                      style={{ color: entry.color }}
                    >
                      {entry.dataKey}: ${formatPrice(entry.value)}
                    </p>
                  ))}
                </div>
              );
            }}
          />
          <Legend />

          {displayAssets.map((assetKey) => {
            const color = config.colors[assetKey] || '#8884d8';
            return config.chartType === 'area' ? (
              <Area
                key={assetKey}
                type="monotone"
                dataKey={assetKey}
                stroke={color}
                fill={color}
                fillOpacity={0.3}
              />
            ) : (
              <Line
                key={assetKey}
                type="monotone"
                dataKey={assetKey}
                stroke={color}
                strokeWidth={2}
                dot={false}
              />
            );
          })}

          {/* 베팅 마커 표시 */}
          {config.showBettingMarkers &&
            !overlay &&
            asset &&
            bettingMarkers
```

```
              .filter((marker) => marker.asset === asset)
              .map((marker) => (
                <ReferenceLine
                  key={marker.id}
                  x={marker.timestamp}
                  stroke={marker.result === 'win' ? '#22c55e' :
'#ef4444'}
                  strokeDasharray="3 3"
                  label={{
                    value: marker.result === 'win' ? '✓' : '✗',
                    position: 'top',
                  }}
                />
              ))}
        </ChartComponent>
      </ResponsiveContainer>
    </div>
  );
}
```

8.4 Volatility Panel

```typescript
// components/chart/VolatilityPanel.tsx

'use client';

import { useChartStore } from '@/store/useChartStore';
import { Card, CardContent, CardHeader, CardTitle } from
'@/components/ui/card';
import { Progress } from '@/components/ui/progress';
import type { AssetType } from '@/types/chart';

export function VolatilityPanel() {
  const { config, volatilityData, comparisonData } = useChartStore();

  return (
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-
```

```
4">
      {config.selectedAssets.map((asset) => {
        const metrics = volatilityData.get(asset);
        if (!metrics) return null;

        return (
          <Card key={asset}>
            <CardHeader>
              <CardTitle className="text-lg">{asset}
Volatility</CardTitle>
            </CardHeader>
            <CardContent className="space-y-4">
              <div>
                <div className="flex justify-between text-sm mb-1">
                  <span>Standard Deviation</span>
                  <span className="font-medium">
                    {metrics.stdDev.toFixed(2)}
                  </span>
                </div>
                <Progress value={Math.min(metrics.stdDev / 10, 100)} />
              </div>

              <div>
                <div className="flex justify-between text-sm mb-1">
                  <span>Price Change</span>
                  <span
                    className={`font-medium ${
                      metrics.percentChange >= 0
                        ? 'text-green-500'
                        : 'text-red-500'
                    }`}
                  >
                    {metrics.percentChange >= 0 ? '+' : ''}
                    {metrics.percentChange.toFixed(2)}%
                  </span>
                </div>
                <Progress
                  value={Math.min(Math.abs(metrics.percentChange), 100)}
                  className={
                    metrics.percentChange >= 0 ? 'bg-green-200' : 'bg-
```

```
red-200'
                  }
                />
              </div>

              {metrics.rsi && (
                <div>
                  <div className="flex justify-between text-sm mb-1">
                    <span>RSI (14)</span>
                    <span className="font-medium">
{metrics.rsi.toFixed(2)}</span>
                  </div>
                  <Progress
                    value={metrics.rsi}
                    className={
                      metrics.rsi > 70
                        ? 'bg-red-200'
                        : metrics.rsi < 30
                        ? 'bg-green-200'
                        : ''
                    }
                  />
                  <div className="flex justify-between text-xs text-
muted-foreground mt-1">
                    <span>Oversold</span>
                    <span>Neutral</span>
                    <span>Overbought</span>
                  </div>
                </div>
              )}

              {metrics.bollingerBands && (
                <div className="text-sm space-y-1">
                  <div className="flex justify-between">
                    <span>Upper Band:</span>
                    <span className="font-medium">
                      ${metrics.bollingerBands.upper.toFixed(2)}
                    </span>
                  </div>
                  <div className="flex justify-between">
```

```
                    <span>Middle Band:</span>
                    <span className="font-medium">
                      ${metrics.bollingerBands.middle.toFixed(2)}
                    </span>
                  </div>
                  <div className="flex justify-between">
                    <span>Lower Band:</span>
                    <span className="font-medium">
                      ${metrics.bollingerBands.lower.toFixed(2)}
                    </span>
                  </div>
                </div>
              )}
            </CardContent>
          </Card>
        );
      })}

      {/* 비교 카드 */}
      {comparisonData && (
        <Card className="md:col-span-2 lg:col-span-1">
          <CardHeader>
            <CardTitle className="text-lg">Comparison
Analysis</CardTitle>
          </CardHeader>
          <CardContent className="space-y-4">
            <div>
              <div className="flex justify-between text-sm mb-1">
                <span>Volatility Ratio (PAXG/BTC)</span>
                <span className="font-medium">
                  {comparisonData.volatilityRatio.toFixed(2)}
                </span>
              </div>
              <Progress value={comparisonData.volatilityRatio * 50} />
            </div>

            {comparisonData.recommendation && (
              <div className="p-3 bg-primary/10 rounded-lg">
                <div className="text-sm font-medium mb-1">
                  AI Recommendation
                </div>
                <div className="text-lg font-bold text-primary mb-1">
                  {comparisonData.recommendation.asset}
                </div>
                <div className="text-xs text-muted-foreground">
                  Confidence: {(comparisonData.recommendation.confidence
* 100).toFixed(0)}%
                </div>
                <div className="text-xs mt-2">
                  {comparisonData.recommendation.reason}
                </div>
              </div>
            )}

            {Object.entries(comparisonData.priceChangeComparison).map(
              ([asset, data]) => (
                <div key={asset} className="text-sm">
                  <div className="font-medium mb-1">{asset}</div>
                  <div className="flex justify-between">
                    <span>Change:</span>
                    <span
                      className={
                        data.percentage >= 0 ? 'text-green-500' : 'text-
red-500'
                      }
                    >
                      {data.percentage >= 0 ? '+' : ''}
                      {data.percentage.toFixed(2)}%
                    </span>
                  </div>
                  <div className="flex justify-between">
                    <span>Vol-Adjusted:</span>
                    <span className="font-medium">
                      {data.volatilityAdjusted.toFixed(2)}
                    </span>
                  </div>
                </div>
              )
            )}
          </CardContent>
```

```
        </Card>
      )}
    </div>
  );
}
```

9. WebSocket Hook

typescript

```typescript
// hooks/useWebSocket.ts

'use client';

import { useEffect, useRef, useCallback } from 'react';
import { io, Socket } from 'socket.io-client';
import { useChartStore } from '@/store/useChartStore';
import type { AssetType, PriceData } from '@/types/chart';

export function useWebSocket() {
  const socketRef = useRef<Socket | null>(null);
  const {
    updateRealtimeData,
    setWsConnected,
    setWsReconnecting,
  } = useChartStore();

  const connect = useCallback((assets: AssetType[]) => {
    if (socketRef.current?.connected) {
      socketRef.current.emit('subscribe', assets);
      return;
    }

    // Socket.IO 연결
    socketRef.current = io({
      path: '/api/chart/realtime',
    });
    socketRef.current.on('connect', () => {
      console.log('WebSocket connected');
      setWsConnected(true);
      setWsReconnecting(false);
      socketRef.current?.emit('subscribe', assets);
```

typescript

```typescript
    socketRef.current.on('disconnect', () => {
      console.log('WebSocket disconnected');
      setWsConnected(false);
    });

    socketRef.current.on('reconnect_attempt', () => {
      console.log('WebSocket reconnecting...');
      setWsReconnecting(true);
    });

    socketRef.current.on('price-update', (data: PriceData) => {
      updateRealtimeData(data.asset, data);
    });

    socketRef.current.on('error', (error) => {
      console.error('WebSocket error:', error);
    });
  }, [updateRealtimeData, setWsConnected, setWsReconnecting]);

  const disconnect = useCallback(() => {
    if (socketRef.current) {
      socketRef.current.disconnect();
      socketRef.current = null;
      setWsConnected(false);
    }
  }, [setWsConnected]);

  return { connect, disconnect };
}
```

10. Utility Functions

typescript

```
// lib/utils/chart.ts

import type { AssetType, TimeRange } from '@/types/chart';

/**
 * 가격 포맷팅
 */
export function formatPrice(price: number): string {
  if (price >= 1000) {
    return price.toLocaleString('en-US', {
      minimumFractionDigits: 2,
      maximumFractionDigits: 2,
    });
  }
  return price.toFixed(2);
}

/**
 * 타임스탬프 포맷팅
 */
export function formatTimestamp(timestamp: number, range?: TimeRange):
string {
  const date = new Date(timestamp);

  if (range === '1h' || range === '24h') {
    return date.toLocaleTimeString('en-US', {
      hour: '2-digit',
      minute: '2-digit',
    });
  }

  if (range === '7d' || range === '30d') {
    return date.toLocaleDateString('en-US', {
      month: 'short',
      day: 'numeric',
    });
  }

  return date.toLocaleDateString('en-US', {
    year: 'numeric',
    month: 'short',
    day: 'numeric',
  });
}

/**
 * 퍼센트 변화 계산
 */
export function calculatePercentChange(
  currentPrice: number,
  previousPrice: number
): number {
  if (previousPrice === 0) return 0;
  return ((currentPrice - previousPrice) / previousPrice) * 100;
}

/**
 * 변동성 계산
 */
export function calculateVolatility(prices: number[]): number {
  if (prices.length < 2) return 0;

  const mean = prices.reduce((sum, price) => sum + price, 0) /
prices.length;
  const variance = prices.reduce((sum, price) => {
    return sum + Math.pow(price - mean, 2);
  }, 0) / prices.length;

  return Math.sqrt(variance);
}

/**
 * 이동평균 계산
 */
export function calculateSMA(prices: number[], period: number): number[]
{
  const sma: number[] = [];

  for (let i = 0; i < prices.length; i++) {
    if (i < period - 1) {
      sma.push(NaN);
      continue;
    }

    const sum = prices.slice(i - period + 1, i + 1).reduce((a, b) => a +
b, 0);
    sma.push(sum / period);
  }

  return sma;
}

/**
 * 지수이동평균 계산
 */
export function calculateEMA(prices: number[], period: number): number[]
{
  const ema: number[] = [];
  const multiplier = 2 / (period + 1);

  // 첫 번째 EMA는 SMA로 시작
  const firstSMA = prices.slice(0, period).reduce((a, b) => a + b, 0) /
period;
  ema.push(firstSMA);

  for (let i = period; i < prices.length; i++) {
    const currentEMA = (prices[i] - ema[ema.length - 1]) * multiplier +
ema[ema.length - 1];
    ema.push(currentEMA);
  }

  return ema;
}

/**
 * RSI 계산
 */
export function calculateRSI(prices: number[], period: number = 14):
number[] {
  const rsi: number[] = [];
  const changes = prices.slice(1).map((price, i) => price - prices[i]);

  for (let i = 0; i < changes.length; i++) {
    if (i < period - 1) {
      rsi.push(NaN);
      continue;
    }

    const recentChanges = changes.slice(i - period + 1, i + 1);
    const gains = recentChanges.filter(c => c > 0);
    const losses = recentChanges.filter(c => c < 0).map(c =>
Math.abs(c));

    const avgGain = gains.length > 0 ? gains.reduce((a, b) => a + b, 0)
/ period : 0;
    const avgLoss = losses.length > 0 ? losses.reduce((a, b) => a + b,
0) / period : 0;

    if (avgLoss === 0) {
      rsi.push(100);
    } else {
      const rs = avgGain / avgLoss;
      rsi.push(100 - (100 / (1 + rs)));
    }
  }

  return rsi;
}

/**
 * MACD 계산
 */
export function calculateMACD(
  prices: number[],
  fastPeriod: number = 12,
  slowPeriod: number = 26,
  signalPeriod: number = 9
): { macd: number[]; signal: number[]; histogram: number[] } {
  const fastEMA = calculateEMA(prices, fastPeriod);
  const slowEMA = calculateEMA(prices, slowPeriod);
  const macdLine = fastEMA.map((fast, i) => fast - slowEMA[i]);
  const signalLine = calculateEMA(macdLine.filter(v => !isNaN(v)),
signalPeriod);
  const histogram = macdLine.map((macd, i) => macd - (signalLine[i] ||
0));

  return {
    macd: macdLine,
    signal: signalLine,
    histogram,
  };
}

/**
 * 볼린저 밴드 계산
 */
export function calculateBollingerBands(
  prices: number[],
  period: number = 20,
  stdDevMultiplier: number = 2
): { upper: number[]; middle: number[]; lower: number[] } {
  const middle = calculateSMA(prices, period);
  const upper: number[] = [];
  const lower: number[] = [];

  for (let i = 0; i < prices.length; i++) {
    if (i < period - 1) {
      upper.push(NaN);
      lower.push(NaN);
      continue;
    }

    const slice = prices.slice(i - period + 1, i + 1);
    const sma = middle[i];
    const variance = slice.reduce((sum, price) => {
      return sum + Math.pow(price - sma, 2);
    }, 0) / period;
    const stdDev = Math.sqrt(variance);
    upper.push(sma + stdDevMultiplier * stdDev);
    lower.push(sma - stdDevMultiplier * stdDev);
  }

  return { upper, middle, lower };
}

/**
 * ATR (Average True Range) 계산
 */
export function calculateATR(
  high: number[],
  low: number[],
  close: number[],
  period: number = 14
): number[] {
  const trueRanges: number[] = [];

  for (let i = 1; i < high.length; i++) {
    const tr = Math.max(
      high[i] - low[i],
      Math.abs(high[i] - close[i - 1]),
      Math.abs(low[i] - close[i - 1])
    );
    trueRanges.push(tr);
  }

  return calculateSMA(trueRanges, period);
}

/**
 * 데이터 정규화 (0-100 스케일)
 */
export function normalizeData(data: number[]): number[] {
  const min = Math.min(...data);
  const max = Math.max(...data);
  const range = max - min;

  if (range === 0) return data.map(() => 50);
  return data.map(value => ((value - min) / range) * 100);
}

/**
 * 자산 색상 가져오기
 */
export function getAssetColor(asset: AssetType): string {
  const colors: Record<AssetType, string> = {
    PAXG: '#FFD700',
    BTC: '#F7931A',
    ETH: '#627EEA',
    SOL: '#14F195',
    USDT: '#26A17B',
    USDC: '#2775CA',
  };

  return colors[asset] || '#8884d8';
}

/**
 * 시간 범위를 밀리초로 변환
 */
export function timeRangeToMs(range: TimeRange): number {
  const ranges: Record<TimeRange, number> = {
    '1h': 60 * 60 * 1000,
    '24h': 24 * 60 * 60 * 1000,
    '7d': 7 * 24 * 60 * 60 * 1000,
    '30d': 30 * 24 * 60 * 60 * 1000,
    'all': Number.MAX_SAFE_INTEGER,
  };

  return ranges[range];
}
```

11. API Service Layer

typescript

```
// services/api/chartApi.ts

import type {
  AssetType,
  TimeRange,
  PriceData,
  VolatilityMetrics,
  ComparisonData,
  CandlestickData,
} from '@/types/chart';

class ChartApiService {
  private baseUrl = '/api/chart';

  /**
   * 과거 가격 데이터 조회
   */
  async getHistoricalData(
    asset: AssetType,
    range: TimeRange
  ): Promise<PriceData[]> {
    const response = await fetch(
      `${this.baseUrl}/historical?asset=${asset}&range=${range}`
    );

    if (!response.ok) {
      throw new Error(`Failed to fetch historical data:
${response.statusText}`);
    }

    return response.json();
  }

  /**
   * 캔들스틱 데이터 조회
   */
  async getCandlestickData(
    asset: AssetType,
    range: TimeRange,
    interval: string = '1h'
  ): Promise<CandlestickData[]> {
    const response = await fetch(
      `${this.baseUrl}/candlestick?
asset=${asset}&range=${range}&interval=${interval}`
    );

    if (!response.ok) {
      throw new Error(`Failed to fetch candlestick data:
${response.statusText}`);
    }

    return response.json();
  }

  /**
   * 변동성 지표 조회
   */
  async getVolatilityMetrics(asset: AssetType):
Promise<VolatilityMetrics> {
    const response = await fetch(
      `${this.baseUrl}/volatility?asset=${asset}`
    );

    if (!response.ok) {
      throw new Error(`Failed to fetch volatility metrics:
${response.statusText}`);
    }

    return response.json();
  }

  /**
   * 비교 데이터 조회
   */
  async getComparisonData(
    assets: AssetType[],
    range: TimeRange
  ): Promise<ComparisonData> {
    const response = await fetch(
      `${this.baseUrl}/compare?
```

```
assets=${assets.join(',')}&range=${range}`
    );

    if (!response.ok) {
      throw new Error(`Failed to fetch comparison data:
${response.statusText}`);
    }

    return response.json();
  }

  /**
   * 특정 자산 상세 정보 조회
   */
  async getAssetDetails(asset: AssetType): Promise<any> {
    const response = await fetch(`${this.baseUrl}/${asset}`);

    if (!response.ok) {
      throw new Error(`Failed to fetch asset details:
${response.statusText}`);
    }

    return response.json();
  }

  /**
   * 외부 API에서 실시간 가격 조회 (CoinGecko)
   */
  async getExternalPrice(asset: AssetType): Promise<PriceData> {
    const coinGeckoIds: Record<AssetType, string> = {
      PAXG: 'pax-gold',
      BTC: 'bitcoin',
      ETH: 'ethereum',
      SOL: 'solana',
      USDT: 'tether',
      USDC: 'usd-coin',
    };

    const coinId = coinGeckoIds[asset];
    const response = await fetch(
      `https://api.coingecko.com/api/v3/simple/price?
ids=${coinId}&vs_currencies=usd&include_24hr_change=true&include_24hr_vo
l=true`
    );

    if (!response.ok) {
      throw new Error(`Failed to fetch external price:
${response.statusText}`);
    }

    const data = await response.json();
    const priceData = data[coinId];

    return {
      asset,
      timestamp: Date.now(),
      price: priceData.usd,
      volume: priceData.usd_24h_vol || 0,
      change24h: priceData.usd_24h_change || 0,
      changePercent24h: priceData.usd_24h_change || 0,
    };
  }

  /**
   * Binance API에서 캔들스틱 데이터 조회
   */
  async getBinanceKlines(
    symbol: string,
    interval: string = '1h',
    limit: number = 100
  ): Promise<CandlestickData[]> {
    const response = await fetch(
      `https://api.binance.com/api/v3/klines?
symbol=${symbol}&interval=${interval}&limit=${limit}`
    );

    if (!response.ok) {
      throw new Error(`Failed to fetch Binance data:
${response.statusText}`);
    }
    const data = await response.json();

    return data.map((kline: any) => ({
      asset: symbol.replace('USDT', '') as AssetType,
      timestamp: kline[0],
      open: parseFloat(kline[1]),
      high: parseFloat(kline[2]),
      low: parseFloat(kline[3]),
      close: parseFloat(kline[4]),
      volume: parseFloat(kline[5]),
    }));
  }
}

export const chartApi = new ChartApiService();
```

---

12. Betting Widget Component

typescript

```typescript
// components/chart/BettingWidget.tsx

'use client';

import { useState } from 'react';
import { useChartStore } from '@/store/useChartStore';
import { Card, CardContent, CardHeader, CardTitle } from
'@/components/ui/card';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Label } from '@/components/ui/label';
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from '@/components/ui/select';
import type { AssetType } from '@/types/chart';

export function BettingWidget() {
  const { config, realtimeData, comparisonData } = useChartStore();
  const [selectedAsset, setSelectedAsset] = useState<AssetType>('PAXG');
  const [betAmount, setBetAmount] = useState<string>('');

  const currentPrice = realtimeData.get(selectedAsset);
  const recommendation = comparisonData?.recommendation;

  const handlePlaceBet = () => {
    // 베팅 로직 - 베팅 시스템 API 호출
    console.log('Placing bet:', { selectedAsset, betAmount });
    // TODO: 베팅 시스템과 연동
  };

  return (
    <Card>
      <CardHeader>
        <CardTitle>Quick Bet</CardTitle>
      </CardHeader>
      <CardContent className="space-y-4">
        <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
          {/* 자산 선택 */}
          <div className="space-y-2">
            <Label>Select Asset</Label>
            <Select
              value={selectedAsset}
              onValueChange={(value) => setSelectedAsset(value as
AssetType)}
            >
              <SelectTrigger>
                <SelectValue />
              </SelectTrigger>
              <SelectContent>
                {config.selectedAssets.map((asset) => (
                  <SelectItem key={asset} value={asset}>
                    {asset}
                  </SelectItem>
                ))}
              </SelectContent>
            </Select>
          </div>

          {/* 베팅 금액 */}
          <div className="space-y-2">
            <Label>Bet Amount (Points)</Label>
            <Input
              type="number"
              placeholder="Enter amount"
              value={betAmount}
              onChange={(e) => setBetAmount(e.target.value)}
            />
          </div>
        </div>

        {/* 현재 가격 정보 */}
        {currentPrice && (
          <div className="p-3 bg-muted rounded-lg space-y-1">
            <div className="flex justify-between text-sm">
              <span>Current Price:</span>
              <span className="font-
  medium">${currentPrice.price.toFixed(2)}</span>
            </div>
            <div className="flex justify-between text-sm">
              <span>24h Change:</span>
              <span
                className={
                  currentPrice.changePercent24h >= 0
                    ? 'text-green-500 font-medium'
                    : 'text-red-500 font-medium'
                }
              >
                {currentPrice.changePercent24h >= 0 ? '+' : ''}
                {currentPrice.changePercent24h.toFixed(2)}%
              </span>
            </div>
          </div>
        )}
```

```
        {/* AI 추천 */}
        {recommendation && (
          <div className="p-3 bg-primary/10 rounded-lg">
            <div className="text-sm font-medium mb-1">AI
Recommendation</div>
            <div className="flex items-center justify-between">
              <div>
                <div className="text-lg font-bold text-primary">
                  {recommendation.asset}
                </div>
                <div className="text-xs text-muted-foreground">
                  {recommendation.reason}
                </div>
              </div>
              <div className="text-right">
                <div className="text-2xl font-bold">
                  {(recommendation.confidence * 100).toFixed(0)}%
                </div>
                <div className="text-xs text-muted-
foreground">Confidence</div>
              </div>
            </div>
          </div>
        )}

        {/* 베팅 버튼 */}
        <div className="grid grid-cols-2 gap-2">
          <Button
            variant="default"
            className="bg-green-600 hover:bg-green-700"
            onClick={handlePlaceBet}
            disabled={!betAmount || parseFloat(betAmount) <= 0}
          >
            Bet {selectedAsset} Wins
          </Button>
          <Button
            variant="outline"
            onClick={() => {
              const otherAsset = config.selectedAssets.find(
                (a) => a !== selectedAsset
              );
              if (otherAsset) setSelectedAsset(otherAsset);
            }}
          >
            Switch to Other
          </Button>
        </div>

        {/* 베팅 통계 */}
        <div className="text-xs text-muted-foreground text-center">
          View your betting history in the My Page section
        </div>
      </CardContent>
    </Card>
  );
}
```

13. Page Implementation

```typescript
// pages/chart/index.tsx (또는 app/chart/page.tsx for App Router)

'use client';

import { ChartContainer } from '@/components/chart/ChartContainer';
import { Suspense } from 'react';

export default function ChartPage() {
  return (
    <div className="container mx-auto py-8 px-4">
      <div className="mb-8">
        <h1 className="text-4xl font-bold mb-2">DeltaX Chart
Analysis</h1>
        <p className="text-muted-foreground">
          Compare PAXG (Gold) and BTC volatility-adjusted returns
        </p>
      </div>

      <Suspense
        fallback={
          <div className="flex items-center justify-center h-96">
            <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-primary" />
          </div>
        }
      >
        <ChartContainer />
      </Suspense>
    </div>
  );
}
```

14. Environment Variables

```bash
# .env.local

# Database
DATABASE_URL="postgresql://user:password@localhost:5432/deltax"

# External APIs
COINGECKO_API_KEY="your_coingecko_api_key"
BINANCE_API_KEY="your_binance_api_key"
BINANCE_API_SECRET="your_binance_api_secret"

# WebSocket
WS_PORT=3001

# Next.js
NEXT_PUBLIC_API_URL="http://localhost:3000"
```

15. Package Dependencies

json

```json
// package.json

{
  "dependencies": {
    "next": "^14.0.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "typescript": "^5.0.0",

    "zustand": "^4.4.0",

    "recharts": "^2.10.0",
    "socket.io": "^4.6.0",
    "socket.io-client": "^4.6.0",

    "@prisma/client": "^5.0.0",
    "prisma": "^5.0.0",

    "@radix-ui/react-select": "^2.0.0",
    "@radix-ui/react-progress": "^1.0.0",
    "class-variance-authority": "^0.7.0",
    "clsx": "^2.0.0",
    "tailwind-merge": "^2.0.0",
    "tailwindcss": "^3.3.0"
  },
  "devDependencies": {
    "@types/node": "^20.0.0",
    "@types/react": "^18.2.0",
    "@types/react-dom": "^18.2.0",
    "eslint": "^8.0.0",
    "eslint-config-next": "^14.0.0"
  }
}
```

16. Testing Examples

typescript

```typescript
// __tests__/chart/utils.test.ts

import {
  calculateVolatility,
  calculateSMA,
  calculateRSI,
  normalizeData,
} from '@/lib/utils/chart';

describe('Chart Utilities', () => {
  describe('calculateVolatility', () => {
    it('should calculate volatility correctly', () => {
      const prices = [100, 102, 98, 103, 97];
      const volatility = calculateVolatility(prices);
      expect(volatility).toBeGreaterThan(0);
    });

    it('should return 0 for insufficient data', () => {
      const prices = [100];
      const volatility = calculateVolatility(prices);
      expect(volatility).toBe(0);
    });
  });

  describe('calculateSMA', () => {
    it('should calculate simple moving average', () => {
      const prices = [10, 20, 30, 40, 50];
      const sma = calculateSMA(prices, 3);
      expect(sma[2]).toBe(20); // (10+20+30)/3
      expect(sma[3]).toBe(30); // (20+30+40)/3
      expect(sma[4]).toBe(40); // (30+40+50)/3
    });
  });

  describe('calculateRSI', () => {
    it('should calculate RSI correctly', () => {
      const prices = Array.from({ length: 20 }, (_, i) => 100 + i);
      const rsi = calculateRSI(prices, 14);
      expect(rsi[rsi.length - 1]).toBeGreaterThan(50);
    });
  });

  describe('normalizeData', () => {
    it('should normalize data to 0-100 scale', () => {
      const data = [10, 20, 30, 40, 50];
      const normalized = normalizeData(data);
      expect(normalized[0]).toBe(0);
      expect(normalized[4]).toBe(100);
      expect(normalized[2]).toBe(50);
    });
  });
});
```

typescript

```typescript
// __tests__/chart/store.test.ts

import { renderHook, act } from '@testing-library/react';
import { useChartStore } from '@/store/useChartStore';

describe('Chart Store', () => {
  beforeEach(() => {
    const { result } = renderHook(() => useChartStore());
    act(() => {
      result.current.reset();
    });
  });

  it('should set view mode', () => {
    const { result } = renderHook(() => useChartStore());

    act(() => {
      result.current.setViewMode('overlay');
    });

    expect(result.current.config.viewMode).toBe('overlay');
  });
  it('should update realtime data', () => {
    const { result } = renderHook(() => useChartStore());

    const mockData = {
      asset: 'BTC' as const,
      timestamp: Date.now(),
      price: 45000,
      volume: 1000000,
      change24h: 500,
      changePercent24h: 1.11,
    };

    act(() => {
      result.current.updateRealtimeData('BTC', mockData);
    });

    expect(result.current.realtimeData.get('BTC')).toEqual(mockData);
  });

  it('should add betting marker', () => {
    const { result } = renderHook(() => useChartStore());

    const marker = {
      id: '1',
      userId: 'user1',
      asset: 'PAXG' as const,
      timestamp: Date.now(),
      betAmount: 100,
      entryPrice: 2000,
    };

    act(() => {
      result.current.addBettingMarker(marker);
    });

    expect(result.current.bettingMarkers).toHaveLength(1);
    expect(result.current.bettingMarkers[0]).toEqual(marker);
  });
});
```

17. 성능 최적화 체크리스트

17.1 프론트엔드 최적화

- [ ] React.memo()로 불필요한 리렌더링 방지
- [ ] useMemo/useCallback 활용
- [ ] 차트 데이터 가상화 (react-window)
- [ ] 이미지 최적화 (next/image)
- [ ] Code splitting (dynamic import)

17.2 데이터 최적화

- [ ] Redis 캐싱 구현
- [ ] API 응답 압축 (gzip)
- [ ] 데이터베이스 인덱싱
- [ ] 쿼리 최적화 (N+1 문제 해결)
- [ ] CDN 활용

17.3 WebSocket 최적화

- [ ] 연결 풀링
- [ ] 메시지 배칭
- [ ] 압축 전송
- [ ] 자동 재연결 로직
- [ ] Heartbeat 구현

18. 보안 고려사항

18.1 API 보안

typescript

```
// middleware/rateLimit.ts

import { NextApiRequest, NextApiResponse } from 'next';
import { LRUCache } from 'lru-cache';

type Options = {
  uniqueTokenPerInterval?: number;
  interval?: number;
};

export default function rateLimit(options?: Options) {
  const tokenCache = new LRUCache({
    max: options?.uniqueTokenPerInterval || 500,
    ttl: options?.interval || 60000,
  });

  return {
    check: (res: NextApiResponse, limit: number, token: string) =>
      new Promise<void>((resolve, reject) => {
        const tokenCount = (tokenCache.get(token) as number[]) || [0];
        if (tokenCount[0] === 0) {
          tokenCache.set(token, tokenCount);
        }
        tokenCount[0] += 1;

        const currentUsage = tokenCount[0];
        const isRateLimited = currentUsage >= limit;
        res.setHeader('X-RateLimit-Limit', limit);
        res.setHeader(
          'X-RateLimit-Remaining',
          isRateLimited ? 0 : limit - currentUsage
        );

        return isRateLimited ? reject() : resolve();
      }),
  };
}
```

18.2 입력 검증

typescript

```typescript
// lib/validation/chart.ts

import { z } from 'zod';

export const assetSchema = z.enum(['PAXG', 'BTC', 'ETH', 'SOL', 'USDT',
'USDC']);

export const timeRangeSchema = z.enum(['1h', '24h', '7d', '30d',
'all']);

export const historicalDataRequestSchema = z.object({
  asset: assetSchema,
  range: timeRangeSchema,
});

export const comparisonRequestSchema = z.object({
  assets: z.array(assetSchema).min(2).max(4),
  range: timeRangeSchema,
});
```

---

19. 배포 체크리스트

19.1 프로덕션 준비

- [ ] 환경 변수 설정 확인
- [ ] 데이터베이스 마이그레이션
- [ ] API 키 보안 확인
- [ ] CORS 설정
- [ ] SSL/TLS 인증서
- [ ] 로깅 시스템 구축
- [ ] 모니터링 도구 설정 (Sentry, DataDog)
- [ ] 백업 전략 수립

19.2 성능 테스트

- ☐ 로드 테스트 (Artillery, k6)
- ☐ WebSocket 부하 테스트
- ☐ 데이터베이스 쿼리 성능 측정
- ☐ 메모리 누수 확인
- ☐ 브라우저 호환성 테스트

---

20. 향후 개선 사항 (추가)

20.1 단기 목표 (1-2개월)

- ☐ 모바일 앱 개발 (React Native)
- ☐ 다크/라이트 테마 전환
- ☐ 차트 스냅샷 공유 기능
- ☐ 알림 시스템 구축
- ☐ 사용자 커스텀 지표 추가

20.2 중기 목표 (3-6개월)

- ☐ AI 기반 가격 예측 모델 고도화
- ☐ 소셜 트레이딩 기능
- ☐ 자동 매매 봇 연동
- ☐ 포트폴리오 분석 도구
- ☐ 교육 콘텐츠 플랫폼

20.3 장기 목표 (6개월+)

- ☐ 다른 블록체인 자산 지원 확대
- ☐ DeFi 프로토콜 연동
- ☐ DAO 거버넌스 구현
- ☐ 크로스체인 브릿지
- ☐ 모바일 네이티브 앱

이 PRD는 차트 모듈의 완전한 구현 가이드입니다. 모든 코드 예제와 속성이 포함되어 있으며, 실제 프로젝트에 바로 적용할 수 있습니다. 추가 질문이나 특정 부분에 대한 더 자세한 설명이 필요하시면 말씀해주세요!