PerpX 完全実装仕様書（業者向け・本番環境対応版）

バージョン: 3.0

作成日: 2025年12月9日

対象: 開発業者・実装チーム

目的: 本番環境で公開可能なperpDEXの完全な実装仕様を提供する

エグゼクティブサマリー

本書は、PerpX分散型パーペチュアル取引所の**本番環境での公開に必要なすべての要素**を網羅した実装仕様書である。GMX V2、dYdX V4、Hyperliquidなど既存のperpDEXが実施している**セキュリティ監査、運用監視、テスト戦略、コンプライアンス対応**を含む包括的な内容となっている。

既存のPerpXフロントエンド（React 19 + Wagmi + Viem）との完全な統合を前提とし、業者が実装すべきバックエンドシステムの全体像を明確に定義している。

目次

---

1. MVP機能の明確化

1.1 MVP（最小実行可能製品）の定義

MVPは、PerpXの最初のリリースで提供する最小限の機能セットである。GMX V2、dYdX V4、Hyperliquidの初期バージョンと同等の機能を目指す。

1.2 MVP機能リスト

必須機能（Phase 1: 初回リリース）

取引機能

機能	説明	参考プロトコル	優先度
BTC/USDT取引	ビットコインのパーペチュアル取引	GMX V2, Hyperliquid	🔴 必須
ETH/USDT取引	イーサリアムのパーペチュアル取引	GMX V2, Hyperliquid	🔴 必須
ロング/ショート	両方向のポジション開設	全プロトコル共通	🔴 必須
レバレッジ調整	1倍〜100倍のレバレッジ選択	dYdX: 20倍, GMX: 50倍, Hyperliquid: 100倍	🔴 必須
成行注文	即座に実行される注文	全プロトコル共通	🔴 必須
ポジションクローズ	全量または部分クローズ	全プロトコル共通	🔴 必須
取引履歴	過去の取引記録	全プロトコル共通	🔴 必須

流動性提供

機能	説明	参考プロトコル	優先度
USDC預金	USDCでの流動性提供	GMX V2（GLP）	🔴 必須
USDT預金	USDTでの流動性提供	GMX V2（GLP）	🔴 必須
引き出し	プールトークンの償還	GMX V2（GLP）	🔴 必須

損益表示	リアルタイムのLP損益	GMX V2ダッシュボード	🔴 必須
APY表示	年間利回りの表示	GMX V2ダッシュボード	🔴 必須

価格フィード

機能	説明	参考プロトコル	優先度
Chainlink統合	BTC/USDTとETH/USDTの価格取得	GMX V2	🔴 必須
1秒更新	1秒以内の価格更新	Hyperliquid（0.2秒）	🔴 必須
価格履歴	過去24時間の価格データ	TradingView統合	🔴 必須

清算

機能	説明	参考プロトコル	優先度
自動清算	維持証拠金を下回ったポジションの清算	全プロトコル共通	🔴 必須
清算報酬	清算者への報酬	GMX: 1%, dYdX: 5%, Hyperliquid: 2.5%	🔴 必須
清算通知	清算リスクのアラート	GMX V2	🔴 必須

ユーザーインターフェース

機能	説明	参考プロトコル	優先度
ウォレット接続	MetaMask, WalletConnect, Coinbase Wallet	全プロトコル共通	🔴 必須
リアルタイム価格表示	右上の価格表示（既存フロントエンド）	全プロトコル共通	🔴 必須
TradingViewチャート	チャート統合（既存フロントエンド）	全プロトコル共通	🔴 必須

ポジション管理	オープン、履歴、損益表示	全プロトコル共通	🔴 必須
流動性管理	預金、引き出し、APY表示	GMX V2	🔴 必須

除外機能（Phase 2以降）

以下の機能はMVPには含めず、Phase 2以降で実装する。

- 指値注文（Limit Order）
- ストップロス/テイクプロフィット（Stop Loss/Take Profit）
- 追加の取引ペア（SOL、BNB、AVAX、MATIC等）
- ファンディングレート（Funding Rate）
- 流動性マイニング報酬（Liquidity Mining）
- ガバナンストークン（Governance Token）
- モバイルアプリ
- ソーシャルトレーディング
- コピートレード

1.3 MVP成功基準

機能要件

- BTC/USDTとETH/USDTで成行注文が実行できる
- レバレッジ1倍〜100倍で取引できる
- LPがUSDC/USDTを預金・引き出しできる
- 清算が自動的に実行される
- リアルタイム価格が1秒以内に更新される
- ポジションと取引履歴が正確に表示される

パフォーマンス要件

- 注文配置から確認まで2秒以内（Hyperliquid: 0.2秒、GMX: 1-2秒）
- 価格更新が1秒以内
- 同時100ユーザーをサポート
- API応答時間が500ms以内

- WebSocket遅延が100ms以内

セキュリティ要件

- 最低2社の外部監査を完了（GMX: Peckshield + ABDK、dYdX: Trail of Bits + Certora）
- テストネットで4週間の運用実績
- バグバウンティプログラムの開始
- 保険基金の設立（初期資金: $100,000）
- マルチシグガバナンス（3-of-5）

ビジネス要件

- 初期流動性: $500,000以上
- 初日取引高: $1,000,000以上
- 初月ユーザー数: 500人以上
- 清算成功率: 99%以上

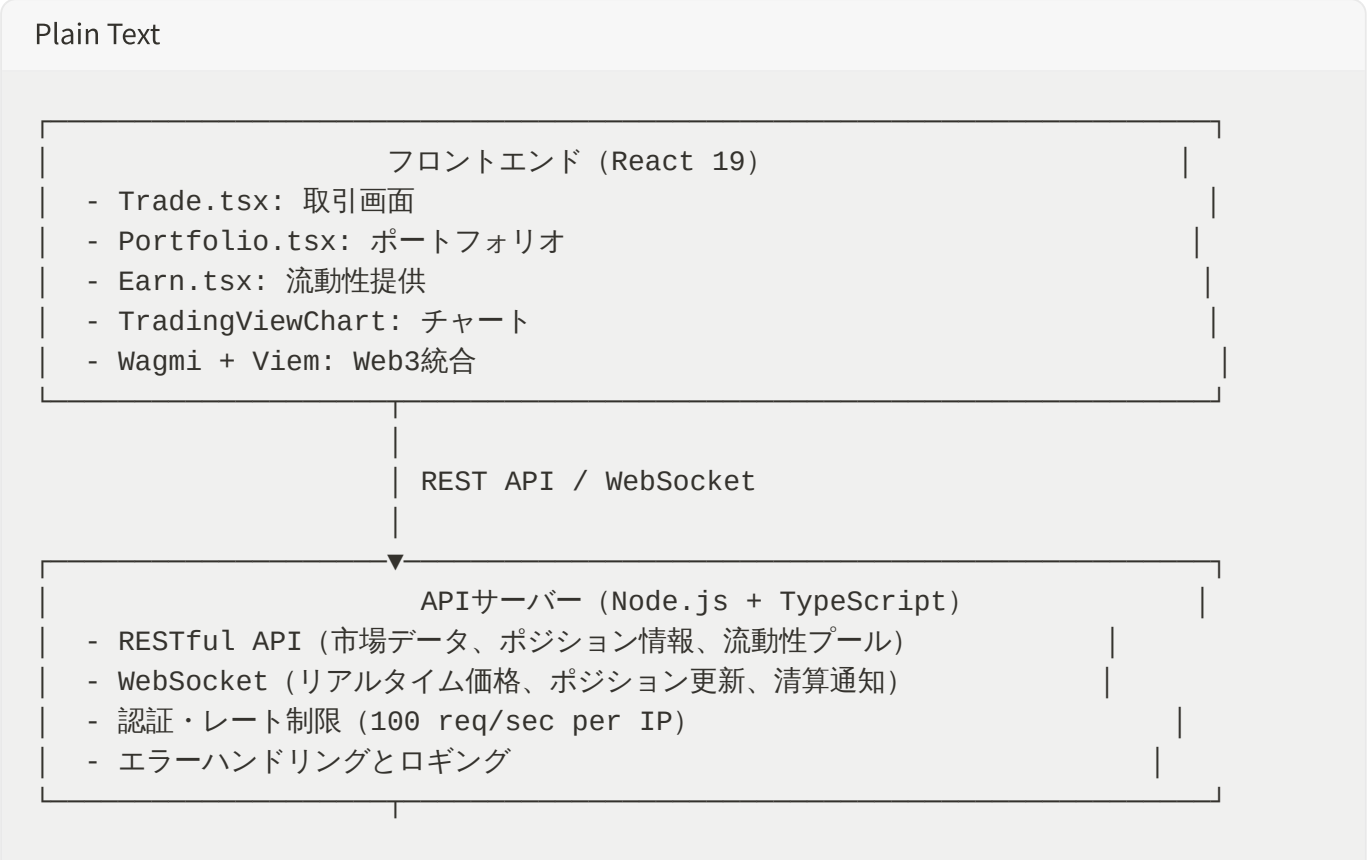
---

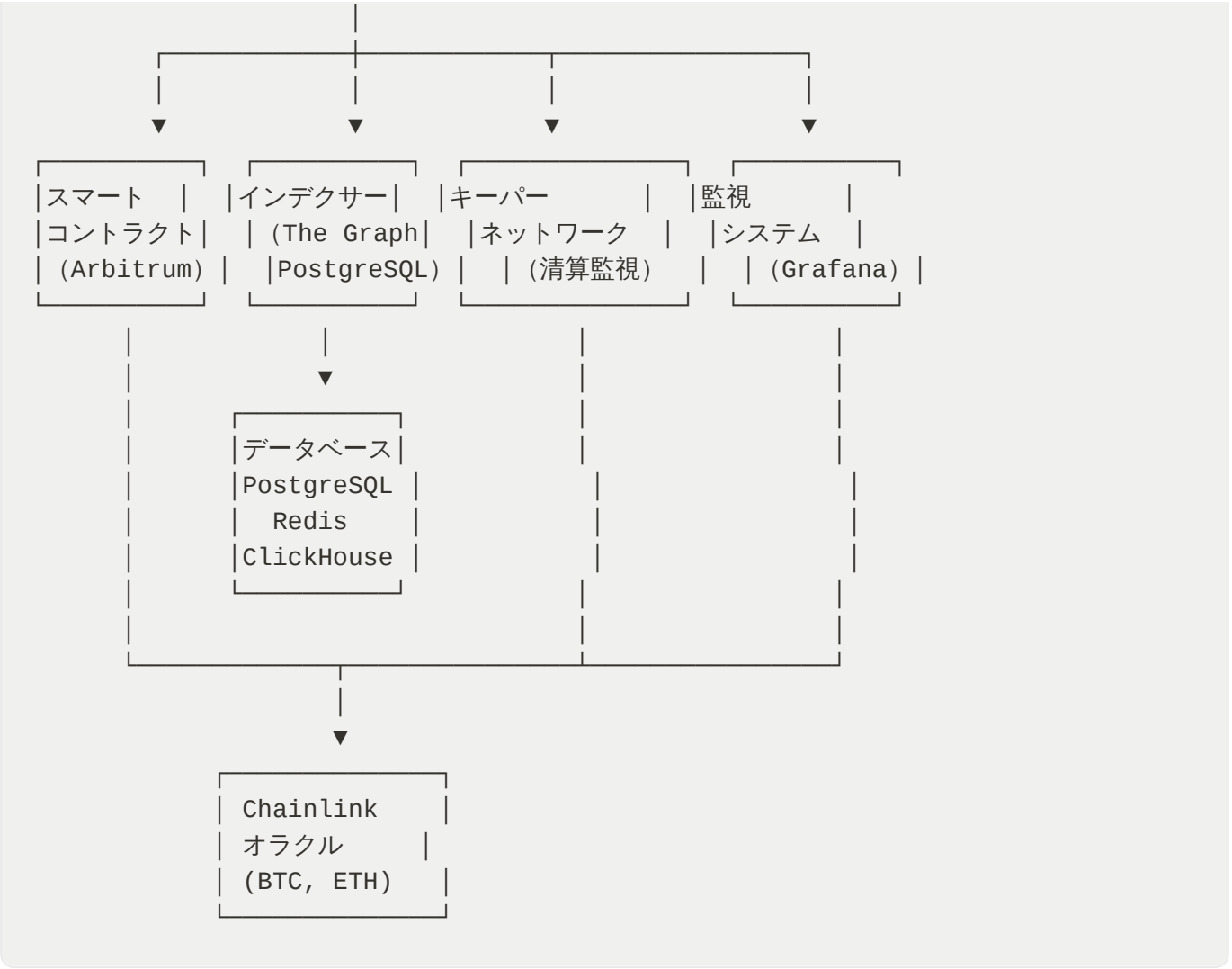
2. バックエンドシステム構成

2.1 システム全体像

PerpXのバックエンドは、以下の主要コンポーネントで構成される。

Plain Text





2.2 各コンポーネントの詳細

2.2.1 スマートコントラクト（Arbitrum One）

役割: オンチェーンでの取引実行、流動性管理、清算処理

デプロイ先: Arbitrum One（メインネット）、Arbitrum Sepolia（テストネット）

ガス最適化:

- バッチ処理による複数操作の統合
- ストレージの効率的な使用
- 不要な計算の削減

アップグレード戦略:

- Transparent Proxy Patternを使用
- マルチシグによるアップグレード承認（3-of-5）
- 48時間のタイムロック

参考実装:

- GMX V2: 独立した流動性プール、デュアルトークン担保
- dYdX V4: Cosmos SDKベースのL1（オフチェーン注文マッチング）
- Hyperliquid: カスタムL1、超高速決済

PerpXの選択: Arbitrum OneのEVMスマートコントラクト（GMX V2モデル）

2.2.2 インデクサー（The Graph + PostgreSQL）

役割: ブロックチェーンイベントをリアルタイムで処理し、データベースに保存

技術スタック:

- The Graph Protocol: サブグラフによるイベント処理
- PostgreSQL: リレーショナルデータベース
- Redis: キャッシュとセッション管理
- ClickHouse: 分析用データウェアハウス

処理するイベント:

- `PositionOpened`: ポジション開設
- `PositionClosed`: ポジションクローズ
- `PositionModified`: ポジション変更
- `Liquidation`: 清算実行
- `Deposit`: 流動性預金
- `Withdraw`: 流動性引き出し
- `PriceUpdated`: 価格更新

データ処理フロー:

1. The Graphサブグラフがイベントを検出
2. イベントデータをPostgreSQLに保存
3. Redisキャッシュを更新
4. WebSocketで接続中のクライアントに通知
5. ClickHouseに分析用データを保存

パフォーマンス要件:

- イベント処理遅延: 5秒以内
- データベースクエリ: 100ms以内
- キャッシュヒット率: 90%以上

参考実装:

- GMX: Subgraph（The Graph Protocol）
- dYdX: カスタムインデクサー（Kafka + PostgreSQL）
- Hyperliquid: 独自のインデクサー

2.2.3 APIサーバー（Node.js + TypeScript）

役割: フロントエンドへのデータ提供、WebSocketリアルタイム通信

技術スタック:

- Node.js 20 LTS
- TypeScript 5.x
- Express.js: RESTful API
- Socket.IO: WebSocket通信
- Joi: バリデーション
- Winston: ロギング

提供するAPI:

- 市場データ（価格、出来高、オープンインタレスト）
- ポジション情報（オープン、履歴）
- 流動性プール情報
- リアルタイム価格配信（WebSocket）
- 取引履歴
- 統計データ

セキュリティ機能:

- レート制限: 100 req/sec per IP
- CORS設定: フロントエンドドメインのみ許可
- 入力バリデーション: 全エンドポイント
- エラーハンドリング: 詳細なエラーメッセージ
- ロギング: 全リクエスト/レスポンス

パフォーマンス要件:

- API応答時間: 500ms以内
- WebSocket遅延: 100ms以内
- 同時接続数: 1,000以上

- スループット: 10,000 req/sec

参考実装:

- GMX: Node.js + Express
- dYdX: Go言語 + gRPC
- Hyperliquid: Rust + WebSocket

2.2.4 キーパーネットワーク

役割: 清算監視、自動清算実行、システムメンテナンス

技術スタック:

- Node.js + TypeScript
- Ethers.js: ブロックチェーン通信
- Cron: 定期実行

キーパーの役割:

- 5秒ごとにポジションの健全性をチェック
- 維持証拠金を下回ったポジションを清算
- 清算報酬（1%）を獲得
- 価格フィードの更新（バックアップ）

分散化戦略:

- 誰でもキーパーボットを実行可能
- 清算競争による効率化
- 報酬メカニズムによるインセンティブ

参考実装:

- GMX: Keeper Network（分散型ボット）
- dYdX: Liquidation Bots（誰でも実行可能）
- Hyperliquid: 内蔵清算エンジン

2.2.5 監視システム（Grafana + Prometheus）

役割: システムの健全性監視、アラート、ダッシュボード

技術スタック:

- Grafana: ダッシュボード
- Prometheus: メトリクス収集

- Loki: ログ集約
- Alertmanager: アラート管理

監視項目:

- API応答時間
- WebSocket接続数
- データベースクエリ時間
- ブロックチェーン同期状態
- キーパーボットの稼働状況
- エラー率
- トランザクション成功率

アラート条件:

- API応答時間 > 1秒
- エラー率 > 1%
- データベース接続失敗
- ブロックチェーン同期遅延 > 10ブロック
- キーパーボット停止

---

3. スマートコントラクト仕様

3.1 コントラクト構成

PerpXは以下のスマートコントラクトで構成される。

コントラクト名	役割	参考プロトコル	ファイル数
LiquidityPool	流動性プール管理	GMX V2のGLPプール	3
PerpetualTrading	取引エンジン	GMX V2のPositionManager	5
PriceOracle	価格フィード統合	GMX V2のOracle	2
LiquidationEngine	清算エンジン	GMX V2のLiquidationHandler	2
OrderBook	注文管理（Phase 2）	dYdX V4のOrderBook	-

FundingRateCalculator	ファンディングレート（Phase 2）	dYdX V4のFunding	-
PerpXGovernance	ガバナンス（Phase 2）	GMX V2のGovToken	-
合計			**12ファイル**

3.2 LiquidityPool（流動性プール）

機能概要

GMX V2のGLP（GMX Liquidity Provider）プールと同様の仕組みを実装する。LPはUSDCまたはUSDTを預金し、プールトークン（PLP: PerpX Liquidity Provider Token）を受け取る。プールトークンの価値は、トレーダーの損益によって変動する。

主要機能

預金（Deposit）

LPがUSDCまたはUSDTを預金し、プールトークン（PLP）を受け取る。

処理フロー:

1. LPがUSDCまたはUSDTを承認（ERC-20 approve）
2. `deposit(collateralToken, amount)` 関数を呼び出し
3. 担保トークンをLPからプールに転送
4. 現在のプール価値を計算
5. 発行するプールトークン数を計算: `poolTokens = amount × totalPoolTokens / poolValue`
6. プールトークンをミントしてLPに送付
7. `Deposit` イベントを発行

計算例:

- プール価値: $1,000,000
- 総プールトークン数: 1,000,000 PLP
- LP預金額: $10,000 USDC
- 発行プールトークン数: 10,000 × 1,000,000 / 1,000,000 = 10,000 PLP

引き出し（Withdraw）

LPがプールトークンを償還し、USDCまたはUSDTを受け取る。

処理フロー:

1. LPが `withdraw(poolTokenAmount)` 関数を呼び出し
2. 引き出し手数料（0.1%）を計算
3. 返却する担保額を計算: `collateral = poolTokenAmount × poolValue / totalPoolTokens - fee`
4. プールトークンをバーン
5. 担保トークンをLPに転送
6. `Withdraw` イベントを発行

計算例:

- プール価値: $1,100,000（トレーダーが損失を出したため増加）
- 総プールトークン数: 1,000,000 PLP
- LP償還額: 10,000 PLP
- 引き出し手数料: 0.1%
- 返却担保額: 10,000 × 1,100,000 / 1,000,000 × 0.999 = $10,989 USDC

プール価値計算

プール価値は、総流動性とトレーダーの未実現損益の合計である。

計算式:

```
Plain Text

プール価値 = 総流動性 + Σ(トレーダーの未実現損益)
```

例:

- 総流動性: $1,000,000
- トレーダーA: ロング、未実現損益 = -$5,000（損失）
- トレーダーB: ショート、未実現損益 = $3,000（利益）
- プール価値 = $1,000,000 + (-$5,000) + $3,000 = $998,000

トレーダーが損失を出すとプール価値は増加し、トレーダーが利益を出すとプール価値は減少する。

重要なパラメータ

パラメータ	値	参考（GMX V2）	理由
最小預金額	100 USDC/USDT	1 USDC	スパム防止
引き出し手数料	0.1%	0.3%	LP離脱コスト

最大利用率	90%	85%	流動性確保
最小プール流動性	10,000 USDC	100,000 USDC	MVP初期値

フロントエンド統合（Earn.tsx）

既存のEarn.tsxは、以下の機能を提供している。

表示項目:

- 総流動性（Total Liquidity）
- 利用可能な流動性（Available Liquidity）
- 利用率（Utilization Rate）
- APY（年間利回り）
- 自分の預金額
- 自分のプールトークン数
- 自分の損益

必要なAPI:

- `GET /api/v1/pools`: プール情報
- `GET /api/v1/pools/{address}/position`: LP個人のポジション

必要なスマートコントラクト関数:

- `deposit(collateralToken, amount)`: 預金
- `withdraw(poolTokenAmount)`: 引き出し
- `getPoolInfo()`: プール情報取得
- `getLPPosition(address)`: LP個人のポジション取得

3.3 PerpetualTrading（取引エンジン）

機能概要

GMX V2のPositionManagerと同様の仕組みを実装する。トレーダーは担保を預けてレバレッジポジションを開設し、価格変動に応じて損益が発生する。

主要機能

ポジション開設（Open Position）

トレーダーが担保を預けてレバレッジポジションを開設する。

処理フロー:

1. トレーダーが担保トークン（USDC/USDT）を承認
2. `openPosition(market, isLong, size, leverage, collateral)` 関数を呼び出し
3. 入力検証
   - `size > 0`
   - `leverage >= 1 && leverage <= maxLeverage`
   - `collateral >= size / leverage` （初期証拠金チェック）
   - 市場がアクティブ
4. 取引手数料を計算: `fee = size × tradingFee / 10000`
5. 必要担保を計算: `requiredCollateral = size / leverage + fee`
6. 担保トークンをトレーダーからプールに転送
7. 流動性プールから利用可能な流動性をチェック
8. オラクルから現在価格を取得
9. ポジションを作成し、ストレージに保存
10. `PositionOpened` イベントを発行

計算例:

- 市場: BTC/USDT
- 現在価格: $90,000
- ポジションサイズ: $50,000
- レバレッジ: 10倍
- 必要担保: $50,000 / 10 = $5,000
- 取引手数料: $50,000 × 0.0008 = $40
- 合計必要額: $5,040

ポジションクローズ（Close Position）

トレーダーがポジションをクローズし、損益を確定する。

処理フロー:

1. トレーダーが `closePosition(positionId)` 関数を呼び出し
2. ポジションの存在と所有権を確認
3. オラクルから現在価格を取得
4. 損益を計算

- ロング: `pnl = size × (currentPrice - entryPrice) / entryPrice`
- ショート: `pnl = size × (entryPrice - currentPrice) / entryPrice`

5. クローズ手数料を計算: `fee = size × tradingFee / 10000`
6. 返却額を計算: `returnAmount = collateral + pnl - fee`
7. ポジションをクローズ（`isOpen = false`）
8. 担保トークンをトレーダーに返却
9. 流動性プールの利用率を更新
10. `PositionClosed` イベントを発行

計算例（ロング、利益）:

- エントリー価格: $88,500
- 現在価格: $90,337.33
- ポジションサイズ: $50,000
- 担保: $5,000
- 損益: $50,000 × ($90,337.33 - $88,500) / $88,500 = $1,038
- クローズ手数料: $50,000 × 0.0008 = $40
- 返却額: $5,000 + $1,038 - $40 = $5,998

計算例（ショート、損失）:

- エントリー価格: $88,500
- 現在価格: $90,337.33
- ポジションサイズ: $50,000
- 担保: $5,000
- 損益: $50,000 × ($88,500 - $90,337.33) / $88,500 = -$1,038
- クローズ手数料: $50,000 × 0.0008 = $40
- 返却額: $5,000 - $1,038 - $40 = $3,922

ポジション調整

トレーダーがポジションのサイズを増減できる。

サイズ増加（Increase Position）:

- 追加担保を預けてサイズを増やす
- 平均エントリー価格を再計算

サイズ減少（Decrease Position）:

- 部分的にポジションをクローズ
- 部分損益を確定

重要なパラメータ

パラメータ	値	参考（GMX V2）	参考（Hyperliquid）	理由
最大レバレッジ	100倍	50倍	100倍	競争力
取引手数料	0.08%	0.05-0.07%	0.02-0.05%	収益性
維持証拠金率	5%	1%	3.33%	リスク管理
最小ポジションサイズ	10 USD	10 USD	10 USD	スパム防止
最大ポジションサイズ	プール流動性の10%	プール流動性の30%	制限なし	リスク管理

フロントエンド統合（Trade.tsx）

既存のTrade.tsxは、以下の機能を提供している。

表示項目:

- リアルタイム価格（右上）
- TradingViewチャート
- 注文フォーム（ロング/ショート、サイズ、レバレッジ）
- オープンポジション一覧
- 取引履歴

必要なAPI:

- `GET /api/v1/markets`：市場データ
- `GET /api/v1/positions/{address}`：ポジション情報
- `WebSocket /ws`：リアルタイム価格更新

必要なスマートコントラクト関数:

- `openPosition(market, isLong, size, leverage, collateral)`：ポジション開設
- `closePosition(positionId)`：ポジションクローズ
- `getPosition(positionId)`：ポジション情報取得

- `getUnrealizedPnL(positionId)` : 未実現損益取得
- `getLiquidationPrice(positionId)` : 清算価格取得

3.4 PriceOracle（価格フィード）

機能概要

GMX V2のOracleと同様に、Chainlinkから価格を取得する。Phase 2では、Pyth NetworkやAPI3など複数のオラクルソースを統合し、価格操作耐性を向上させる。

主要機能

価格取得（Get Price）

Chainlink Price Feedから最新価格を取得する。

処理フロー:

1. `getPrice(market)` 関数を呼び出し
2. Chainlink Price Feedから最新価格を取得
3. 価格の鮮度を確認（60秒以内）
4. 価格を8桁固定小数点に正規化
5. 価格とタイムスタンプを返す

Chainlink Price Feed:

- BTC/USD: `0x6ce185860a4963106506C203335A2910413708e9`（Arbitrum One）
- ETH/USD: `0x639Fe6ab55C921f74e7fac1ee960C0B6293ba612`（Arbitrum One）

価格集約（Phase 2）

複数のオラクルソースから価格を取得し、加重平均を計算する。

オラクルソース:

- Chainlink: 50%
- Pyth Network: 30%
- API3: 20%

価格乖離チェック:

- 各ソースの価格差が2%以内であることを確認
- 2%を超える場合、サーキットブレーカーを発動

サーキットブレーカー

価格変動が異常な場合、取引を一時停止する。

発動条件:

- 価格変動が10%を超える（1分以内）
- オラクルソース間の価格乖離が2%を超える
- オラクル更新が60秒以上遅延

再開条件:

- 管理者（マルチシグ）が手動で再開
- 価格が正常に戻ったことを確認

重要なパラメータ

パラメータ	値	参考（GMX V2）	理由
価格更新の最大遅延	60秒	60秒	鮮度確保
価格乖離の閾値	2%	2.5%	価格操作防止
サーキットブレーカー	10%変動	10%変動	異常検知

フロントエンド統合

リアルタイム価格表示:

- WebSocketで1秒ごとに価格を配信
- Trade.tsxの右上に表示
- TradingViewチャートと連動

必要なWebSocketメッセージ:

JSON

```
{
  "type": "price_update",
  "data": {
    "symbol": "BTCUSDT",
    "price": 90337.33,
    "change24h": 2.45,
    "timestamp": 1702098765
  }
}
```

3.5 LiquidationEngine（清算エンジン）

機能概要

GMX V2のLiquidationHandlerと同様に、維持証拠金を下回ったポジションを自動的に清算する。キーパーボットが定期的にポジションの健全性をチェックし、清算可能なポジションを検出して清算を実行する。

主要機能

清算チェック（Check Liquidation）

ポジションが清算可能かどうかをチェックする。

処理フロー:

1. キーパーボットが5秒ごとに全オープンポジションをチェック
2. 各ポジションの現在価格を取得
3. 未実現損益を計算
4. 証拠金率を計算: `marginRatio = (collateral + unrealizedPnL) / size`
5. 維持証拠金率（5%）と比較
6. 証拠金率 < 5%の場合、清算可能

計算例:

- ポジションサイズ: $50,000
- 担保: $5,000
- エントリー価格: $90,000
- 現在価格: $85,500（ロング、価格下落）
- 未実現損益: $50,000 × ($85,500 - $90,000) / $90,000 = -$2,500
- 証拠金率: ($5,000 - $2,500) / $50,000 = 5%
- 維持証拠金率: 5%
- **清算可能**

清算実行（Liquidate Position）

清算可能なポジションをクローズする。

処理フロー:

1. キーパーボットが `liquidatePosition(positionId)` 関数を呼び出し
2. ポジションが清算可能であることを再確認

3. ポジションをクローズ
4. 清算ペナルティ（2%）を計算: `penalty = size × 0.02`
5. 清算報酬（1%）をキーパーに送付: `reward = size × 0.01`
6. 残りのペナルティ（1%）を流動性プールに送付: `toPool = size × 0.01`
7. 残りの担保をトレーダーに返却（あれば）
8. `Liquidation` イベントを発行

計算例:

- ポジションサイズ: $50,000
- 担保: $5,000
- 未実現損益: -$2,500
- 清算ペナルティ: $50,000 × 0.02 = $1,000
- 清算報酬（キーパー）: $50,000 × 0.01 = $500
- プールへの送付: $50,000 × 0.01 = $500
- トレーダーへの返却: $5,000 - $2,500 - $1,000 = $1,500

重要なパラメータ

パラメータ	値	参考（GMX V2）	参考（dYdX）	理由
維持証拠金率	5%	1%	3%	リスク管理
清算ペナルティ	2%	5%	7.5%	清算インセンティブ
清算報酬	1%	1%	5%	キーパー報酬
チェック頻度	5秒	5秒	1秒	効率性

フロントエンド統合

清算リスクアラート:

- 証拠金率が10%を下回った場合、警告を表示
- 証拠金率が7%を下回った場合、危険アラートを表示
- 清算価格を常に表示

必要なAPI:

- `GET /api/v1/positions/{address}`: ポジション情報（証拠金率、清算価格を含む）

必要なWebSocketメッセージ:

JSON

```
{
  "type": "liquidation_warning",
  "data": {
    "positionId": "12345",
    "marginRatio": 0.07,
    "liquidationPrice": "80265.00"
  }
}
```

---

4. API仕様

4.1 API概要

PerpX APIは、フロントエンドとバックエンド間の通信を提供する。RESTful APIとWebSocket APIの2種類を提供する。

ベースURL

- テストネット: `https://testnet-api.perpx.fi`
- メインネット: `https://api.perpx.fi`

認証

- 公開エンドポイント: 認証不要
- ウォレット署名: EIP-712署名による認証（Phase 2）

レート制限

- 公開エンドポイント: 100 req/sec per IP
- 認証済みエンドポイント: 1000 req/sec per address

エラーハンドリング

すべてのエラーレスポンスは以下の形式を返す。

JSON

```
{
  "success": false,
  "error": {
    "code": "INVALID_PARAMETER",
```

```
    "message": "Invalid leverage value. Must be between 1 and 100.",
    "details": {
      "parameter": "leverage",
      "value": 150,
      "min": 1,
      "max": 100
    }
  },
  "timestamp": 1702098765
}
```

エラーコード一覧:

- `INVALID_PARAMETER`: 無効なパラメータ
- `NOT_FOUND`: リソースが見つからない
- `RATE_LIMIT_EXCEEDED`: レート制限超過
- `INTERNAL_SERVER_ERROR`: サーバーエラー
- `BLOCKCHAIN_ERROR`: ブロックチェーンエラー
- `INSUFFICIENT_LIQUIDITY`: 流動性不足
- `POSITION_NOT_FOUND`: ポジションが見つからない
- `UNAUTHORIZED`: 認証エラー

4.2 RESTful API

4.2.1 市場データ

GET /api/v1/markets

すべての取引可能な市場の情報を取得する。

リクエスト:

Plain Text

```
GET https://api.perpx.fi/api/v1/markets
```

レスポンス:

JSON

```
{
  "success": true,
  "data": {
    "markets": [
```

```
      {
        "symbol": "BTCUSDT",
        "name": "Bitcoin",
        "address": "0x1234567890abcdef1234567890abcdef12345678",
        "price": 90337.33,
        "change24h": 2.45,
        "high24h": 91250.00,
        "low24h": 88500.00,
        "volume24h": "2500000000",
        "openInterest": "1250000000",
        "fundingRate": 0.0001,
        "nextFundingTime": 1702123200,
        "maxLeverage": 100,
        "tradingFee": 0.0008,
        "maintenanceMargin": 0.05,
        "isActive": true
      },
      {
        "symbol": "ETHUSDT",
        "name": "Ethereum",
        "address": "0xabcdef1234567890abcdef1234567890abcdef12",
        "price": 3842.50,
        "change24h": 1.85,
        "high24h": 3900.00,
        "low24h": 3750.00,
        "volume24h": "1200000000",
        "openInterest": "600000000",
        "fundingRate": 0.0001,
        "nextFundingTime": 1702123200,
        "maxLeverage": 100,
        "tradingFee": 0.0008,
        "maintenanceMargin": 0.05,
        "isActive": true
      }
    ]
  },
  "timestamp": 1702098765
}
```


フィールド説明:

フィールド	型	説明	計算方法
symbol	string	取引ペアシンボル	-
name	string	市場名	-

address	string	スマートコントラクトアドレス	-
price	number	現在価格	Chainlinkオラクル
change24h	number	24時間変動率（%）	(現在価格 - 24時間前価格) / 24時間前価格 × 100
high24h	number	24時間最高値	-
low24h	number	24時間最安値	-
volume24h	string	24時間取引高（USD）	Σ(ポジションサイズ)
openInterest	string	オープンインタレスト（USD）	Σ(オープンポジションサイズ)
fundingRate	number	ファンディングレート（Phase 2）	(ロング - ショート) / 総OI
nextFundingTime	number	次回ファンディング時刻（Unix timestamp）	-
maxLeverage	number	最大レバレッジ	-
tradingFee	number	取引手数料（小数）	-
maintenanceMargin	number	維持証拠金率（小数）	-
isActive	boolean	市場がアクティブか	-

キャッシング:

- TTL: 5秒
- Redisキャッシュ

フロントエンド統合:

- Trade.tsxの市場選択ドロップダウン
- 価格表示
- 24時間変動率表示

4.2.2 ポジション管理

GET /api/v1/positions/{address}

特定のアドレスの全ポジションを取得する。

リクエスト:

Plain Text

```
GET
https://api.perpx.fi/api/v1/positions/0xABCDEF1234567890ABCDEF1234567890ABCDE
F01
```

レスポンス:

JSON


```
{
  "success": true,
  "data": {
    "positions": [
      {
        "id": "12345",
        "trader": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01",
        "market": "BTCUSDT",
        "marketAddress": "0x1234567890abcdef1234567890abcdef12345678",
        "isLong": true,
        "size": "50000",
        "collateral": "5000",
        "leverage": 10,
        "entryPrice": "88500.00",
        "currentPrice": "90337.33",
        "unrealizedPnL": "1038.00",
        "unrealizedPnLPercent": 20.76,
        "liquidationPrice": "80265.00",
        "marginRatio": 0.1207,
        "openTimestamp": 1702012345,
        "isOpen": true
      },
      {
        "id": "12346",
        "trader": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01",
        "market": "ETHUSDT",
        "marketAddress": "0xabcdef1234567890abcdef1234567890abcdef12",
        "isLong": false,
        "size": "20000",
        "collateral": "1000",
        "leverage": 20,
        "entryPrice": "3800.00",
        "currentPrice": "3842.50",
        "unrealizedPnL": "-223.68",
        "unrealizedPnLPercent": -22.37,
```

```
        "liquidationPrice": "3990.00",
        "marginRatio": 0.0388,
        "openTimestamp": 1702023456,
        "isOpen": true
      }
    ],
    "totalUnrealizedPnL": "814.32",
    "totalCollateral": "6000",
    "accountValue": "6814.32",
    "marginRatio": 0.0974
  },
  "timestamp": 1702098765
}
```


フィールド説明:

フィールド	型	説明	計算方法
id	string	ポジションID	-
trader	string	トレーダーアドレス	-
market	string	市場シンボル	-
marketAddress	string	市場コントラクトアドレス	-
isLong	boolean	ロング/ショート	-
size	string	ポジションサイズ（USD）	-
collateral	string	担保額（USD）	-
leverage	number	レバレッジ	size / collateral
entryPrice	string	エントリー価格	-
currentPrice	string	現在価格	Chainlinkオラクル
unrealizedPnL	string	未実現損益（USD）	ロング: size × (currentPrice - entryPrice) / entryPrice ショート: size × (entryPrice - currentPrice) / entryPrice

unrealizedPnLPercent	number	未実現損益率（%）	unrealizedPnL / collateral × 100
liquidationPrice	string	清算価格	ロング: entryPrice × (1 - 1 / leverage + maintenanceMargin) ショート: entryPrice × (1 + 1 / leverage - maintenanceMargin)
marginRatio	number	証拠金率	(collateral + unrealizedPnL) / size
openTimestamp	number	オープン時刻（Unix timestamp）	-
isOpen	boolean	オープン中かどうか	-

集計フィールド:

フィールド	型	説明	計算方法
totalUnrealizedPnL	string	全ポジションの未実現損益合計	Σ(unrealizedPnL)
totalCollateral	string	全ポジションの担保合計	Σ(collateral)
accountValue	string	アカウント価値	totalCollateral + totalUnrealizedPnL
marginRatio	number	全体の証拠金率	accountValue / Σ(size)

キャッシング:

- TTL: 3秒
- Redisキャッシュ

フロントエンド統合:

- Trade.tsxのポジション一覧
- Portfolio.tsxのポートフォリオ表示
- リアルタイム損益更新

4.2.3 取引履歴

GET /api/v1/trades/{address}

特定のアドレスの取引履歴を取得する。

リクエスト:

Plain Text

```
GET
https://api.perpx.fi/api/v1/trades/0xABCDEF1234567890ABCDEF1234567890ABCDEF01
?limit=50&offset=0
```

クエリパラメータ:

- `limit`: 取得件数（デフォルト: 50、最大: 100）
- `offset`: オフセット（デフォルト: 0）
- `market`: 市場フィルター（オプション）
- `type`: タイプフィルター（`open`, `close`, `liquidation`）

レスポンス:

JSON

```
{
  "success": true,
  "data": {
    "trades": [
      {
        "id": "67890",
        "positionId": "12345",
        "trader": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01",
        "market": "BTCUSDT",
        "type": "close",
        "isLong": true,
        "size": "50000",
        "entryPrice": "88500.00",
        "exitPrice": "90337.33",
        "realizedPnL": "1038.00",
        "fee": "40.00",
        "timestamp": 1702098765,
        "txHash":
"0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef"
      }
    ],
    "total": 127,
    "limit": 50,
```

```
    "offset": 0
  },
  "timestamp": 1702098765
}
```

フロントエンド統合:

- Trade.tsxの取引履歴タブ
- Portfolio.tsxの履歴表示

4.2.4 流動性プール

GET /api/v1/pools

すべての流動性プールの情報を取得する。

リクエスト:

Plain Text

```
GET https://api.perpx.fi/api/v1/pools
```

レスポンス:

JSON

```
{
  "success": true,
  "data": {
    "pools": [
      {
        "address": "0x111122223333444455556666777788899990000",
        "market": "BTCUSDT",
        "totalLiquidity": "50000000",
        "availableLiquidity": "35000000",
        "utilizationRate": 0.30,
        "totalPoolTokens": "48500000",
        "poolTokenPrice": 1.0309,
        "apy": 0.2547,
        "tradingVolume24h": "2500000000",
        "fees24h": "2000000",
        "lpCount": 1247
      },
      {
        "address": "0xaaabbbcccdddeeefff000111222333444555666",
        "market": "ETHUSDT",
        "totalLiquidity": "30000000",
        "availableLiquidity": "22000000",
```

```
        "utilizationRate": 0.27,
        "totalPoolTokens": "29700000",
        "poolTokenPrice": 1.0101,
        "apy": 0.1823,
        "tradingVolume24h": "1200000000",
        "fees24h": "960000",
        "lpCount": 823
      }
    ]
  },
  "timestamp": 1702098765
}
```

フィールド説明:

フィールド	型	説明	計算方法
address	string	プールアドレス	-
market	string	市場シンボル	-
totalLiquidity	string	総流動性（USD ）	Σ(LP預金額)
availableLiquidity	string	利用可能な流動性（USD）	totalLiquidity - 使用中の流動性
utilizationRate	number	利用率	使用中の流動性 / totalLiquidity
totalPoolTokens	string	総プールトークン数	-
poolTokenPrice	number	プールトークン価格	プール価値 / totalPoolTokens
apy	number	年間利回り（APY）	(fees24h × 365) / totalLiquidity
tradingVolume24h	string	24時間取引高（USD）	-
fees24h	string	24時間手数料（USD）	tradingVolume24h × tradingFee
lpCount	number	LP数	-

キャッシング:

- TTL: 10秒

- Redisキャッシュ

フロントエンド統合:

- Earn.tsxのプール情報表示
- APY表示
- 流動性提供フォーム

GET /api/v1/pools/{address}/position

特定のアドレスのLPポジションを取得する。

リクエスト:

Plain Text

```
GET
https://api.perpx.fi/api/v1/pools/0x111122223333444455556666777788889999000/
position?lpAddress=0xABCDEF1234567890ABCDEF1234567890ABCDEF01
```

レスポンス:

JSON

```
{
  "success": true,
  "data": {
    "lpAddress": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01",
    "poolAddress": "0x111122223333444455556666777788889999000",
    "market": "BTCUSDT",
    "poolTokens": "10000",
    "depositValue": "10000",
    "currentValue": "10309",
    "unrealizedPnL": "309",
    "unrealizedPnLPercent": 3.09,
    "depositTimestamp": 1701012345,
    "apy": 0.2547
  },
  "timestamp": 1702098765
}
```

フロントエンド統合:

- Earn.tsxの個人ポジション表示
- 損益表示

4.3 WebSocket API

4.3.1 接続

エンドポイント: `wss://api.perpx.fi/ws`

接続フロー:

1. クライアントがWebSocketに接続
2. サーバーが接続確認メッセージを送信
3. クライアントがチャネルを購読
4. サーバーがリアルタイムデータを送信

接続確認メッセージ:

JSON

```
{
  "type": "connected",
  "data": {
    "message": "Connected to PerpX WebSocket API",
    "version": "1.0.0"
  },
  "timestamp": 1702098765
}
```

4.3.2 価格更新の購読

購読リクエスト:

JSON

```
{
  "type": "subscribe",
  "channel": "price",
  "symbols": ["BTCUSDT", "ETHUSDT"]
}
```

購読確認メッセージ:

JSON

```
{
  "type": "subscribed",
  "channel": "price",
  "symbols": ["BTCUSDT", "ETHUSDT"],
```

```
  "timestamp": 1702098765
}
```

価格更新メッセージ:

JSON

```
{
  "type": "price_update",
  "data": {
    "symbol": "BTCUSDT",
    "price": 90337.33,
    "change24h": 2.45,
    "timestamp": 1702098765
  }
}
```

更新頻度: 1秒ごと（Hyperliquid: 0.2秒、GMX: 1秒 ）

フロントエンド統合:

- Trade.tsxの右上価格表示
- リアルタイム更新

実装例（Trade.tsx）:

TypeScript

```
// 既存のTrade.tsxに実装済み
useEffect(() => {
  const ws = new WebSocket('wss://api.perpx.fi/ws');

  ws.onopen = () => {
    ws.send(JSON.stringify({
      type: 'subscribe',
      channel: 'price',
      symbols: ['BTCUSDT', 'ETHUSDT']
    }));
  };

  ws.onmessage = (event) => {
    const message = JSON.parse(event.data);
    if (message.type === 'price_update') {
      setCurrentPrice(message.data.price);
      setPriceChange(message.data.change24h);
    }
  };
```

```
  return () => ws.close();
}, []);
```

4.3.3 ポジション更新の購読

購読リクエスト:

JSON

```
{
  "type": "subscribe",
  "channel": "positions",
  "address": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01"
}
```

ポジション更新メッセージ:

JSON

```
{
  "type": "position_update",
  "data": {
    "id": "12345",
    "trader": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01",
    "market": "BTCUSDT",
    "currentPrice": "90337.33",
    "unrealizedPnL": "1038.00",
    "unrealizedPnLPercent": 20.76,
    "marginRatio": 0.1207,
    "timestamp": 1702098765
  }
}
```

更新頻度: 価格変動時（1秒ごと）

フロントエンド統合:

- Trade.tsxのポジション一覧
- Portfolio.tsxのポートフォリオ
- リアルタイム損益更新

4.3.4 清算警告の購読

購読リクエスト:

JSON

```
{
  "type": "subscribe",
  "channel": "liquidation_warning",
  "address": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01"
}
```

清算警告メッセージ:

JSON

```
{
  "type": "liquidation_warning",
  "data": {
    "positionId": "12345",
    "market": "BTCUSDT",
    "marginRatio": 0.07,
    "liquidationPrice": "80265.00",
    "currentPrice": "81000.00",
    "severity": "high",
    "timestamp": 1702098765
  }
}
```

severity レベル:

- `low` : 証拠金率 < 15%
- `medium` : 証拠金率 < 10%
- `high` : 証拠金率 < 7%
- `critical` : 証拠金率 < 5.5%

フロントエンド統合:

- Trade.tsxの警告バナー
- ブラウザ通知
- 音声アラート

4.3.5 清算通知の購読

購読リクエスト:

JSON

```
{
  "type": "subscribe",
  "channel": "liquidations",
```

```
  "address": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01"
}
```

清算通知メッセージ:

JSON

```
{
  "type": "liquidation",
  "data": {
    "positionId": "12345",
    "trader": "0xABCDEF1234567890ABCDEF1234567890ABCDEF01",
    "market": "BTCUSDT",
    "size": "50000",
    "collateral": "5000",
    "liquidationPrice": "80265.00",
    "penalty": "1000",
    "returnedCollateral": "1500",
    "timestamp": 1702098765,
    "txHash":
"0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef"
  }
}
```

フロントエンド統合:

- Trade.tsxの通知
- Portfolio.tsxの履歴更新

---

5. フロントエンド完全統合

5.1 既存フロントエンドの構成

既存のPerpXフロントエンドは以下の技術スタックで構築されている。

技術	バージョン	用途
React	19	UIフレームワーク
TypeScript	5.x	型安全性
Tailwind CSS	4	スタイリング
Wagmi	最新	Web3統合

Viem	最新	Ethereumライブラリ
TanStack Query	最新	状態管理とキャッシング
Wouter	最新	ルーティング
Sonner	最新	トースト通知

5.2 主要ページと統合要件

5.2.1 Trade.tsx（取引画面）

表示項目:

- リアルタイム価格（右上）
- 24時間変動率
- TradingViewチャート
- 市場選択ドロップダウン
- 注文フォーム（ロング/ショート、サイズ、レバレッジ、担保）
- オープンポジション一覧
- 取引履歴

必要なAPI:

- `GET /api/v1/markets` : 市場データ
- `GET /api/v1/positions/{address}` : ポジション情報
- `GET /api/v1/trades/{address}` : 取引履歴
- `WebSocket /ws` : リアルタイム価格更新、ポジション更新、清算警告

必要なスマートコントラクト関数:

- `openPosition(market, isLong, size, leverage, collateral)` : ポジション開設
- `closePosition(positionId)` : ポジションクローズ
- `increasePosition(positionId, additionalSize, additionalCollateral)` : ポジション増加
- `decreasePosition(positionId, reduceSize)` : ポジション減少

統合チェックリスト:

- [ ] リアルタイム価格がWebSocketで1秒ごとに更新される
- [ ] 注文フォームが正しく動作する

- [ ] ポジション開設トランザクションが成功する
- [ ] オープンポジションがリアルタイムで更新される
- [ ] 未実現損益が正確に計算される
- [ ] 清算価格が正確に表示される
- [ ] 清算警告が適切に表示される
- [ ] ポジションクローズが正しく動作する
- [ ] 取引履歴が正確に表示される

5.2.2 Portfolio.tsx（ポートフォリオ）

表示項目:

- アカウント価値
- 総未実現損益
- 総担保
- 証拠金率
- オープンポジション一覧（詳細）
- 取引履歴
- パフォーマンスチャート

必要なAPI:

- `GET /api/v1/positions/{address}` : ポジション情報
- `GET /api/v1/trades/{address}` : 取引履歴
- `WebSocket /ws` : ポジション更新

統合チェックリスト:

- [ ] アカウント価値が正確に計算される
- [ ] 総未実現損益が正確に表示される
- [ ] ポジション一覧がリアルタイムで更新される
- [ ] 取引履歴が正確に表示される
- [ ] パフォーマンスチャートが正しく描画される

5.2.3 Earn.tsx（流動性提供）

表示項目:

- プール一覧（BTC/USDT、ETH/USDT）
- 総流動性
- 利用可能な流動性
- 利用率
- APY
- 自分の預金額
- 自分のプールトークン数
- 自分の損益
- 預金フォーム
- 引き出しフォーム

必要なAPI:

- `GET /api/v1/pools` : プール情報
- `GET /api/v1/pools/{address}/position?lpAddress={lpAddress}` : LP個人のポジション

必要なスマートコントラクト関数:

- `deposit(collateralToken, amount)` : 預金
- `withdraw(poolTokenAmount)` : 引き出し
- `getPoolInfo()` : プール情報取得
- `getLPPosition(address)` : LP個人のポジション取得

統合チェックリスト:

- [ ] プール情報が正確に表示される
- [ ] APYが正確に計算される
- [ ] 預金フォームが正しく動作する
- [ ] 預金トランザクションが成功する
- [ ] プールトークンが正しくミントされる
- [ ] 引き出しフォームが正しく動作する
- [ ] 引き出しトランザクションが成功する
- [ ] 担保が正しく返却される
- [ ] 個人の損益が正確に表示される

5.3 Web3統合（Wagmi + Viem）

5.3.1 ウォレット接続

既存のフロントエンドは、Wagmiを使用してウォレット接続を管理している。

サポートするウォレット:

- MetaMask
- WalletConnect
- Coinbase Wallet
- Rabby Wallet

実装例:

TypeScript

```
import { useAccount, useConnect, useDisconnect } from 'wagmi';

function WalletButton() {
  const { address, isConnected } = useAccount();
  const { connect, connectors } = useConnect();
  const { disconnect } = useDisconnect();

  if (isConnected) {
    return (
      <button onClick={() => disconnect()}>
        {address?.slice(0, 6)}...{address?.slice(-4)}
      </button>
    );
  }

  return (
    <button onClick={() => connect({ connector: connectors[0] })}>
      Connect Wallet
    </button>
  );
}
```

5.3.2 スマートコントラクト呼び出し

既存のフロントエンドは、Viemを使用してスマートコントラクトと通信する。

必要な情報:

- スマートコントラクトのアドレス
- スマートコントラクトのABI（Application Binary Interface）
- 関数名とパラメータ

実装例（ポジション開設）:

TypeScript

```
import { useWriteContract, useWaitForTransactionReceipt } from 'wagmi';
import { parseUnits } from 'viem';

function OpenPositionButton() {
  const { writeContract, data: hash } = useWriteContract();
  const { isLoading, isSuccess } = useWaitForTransactionReceipt({ hash });

  const openPosition = async () => {
    await writeContract({
      address: PERPETUAL_TRADING_ADDRESS,
      abi: PERPETUAL_TRADING_ABI,
      functionName: 'openPosition',
      args: [
        MARKET_ADDRESS, // market
        true, // isLong
        parseUnits('50000', 6), // size (50,000 USDC)
        10, // leverage
        parseUnits('5000', 6) // collateral (5,000 USDC)
      ]
    });
  };

  return (
    <button onClick={openPosition} disabled={isLoading}>
      {isLoading ? 'Opening...' : 'Open Position'}
    </button>
  );
}
```

5.3.3 トランザクション状態管理

既存のフロントエンドは、トランザクションの状態を管理している。

状態:

- `idle` : 待機中
- `pending` : トランザクション送信中
- `confirming` : ブロックチェーン確認中
- `success` : 成功
- `error` : エラー

実装例:

TypeScript

```
import { toast } from 'sonner';

function TransactionHandler() {
  const { writeContract, data: hash, error } = useWriteContract();
  const { isLoading, isSuccess } = useWaitForTransactionReceipt({ hash });

  useEffect(() => {
    if (isLoading) {
      toast.loading('Transaction confirming...', { id: hash });
    }
    if (isSuccess) {
      toast.success('Transaction successful!', { id: hash });
    }
    if (error) {
      toast.error(`Transaction failed: ${error.message}`, { id: hash });
    }
  }, [isLoading, isSuccess, error, hash]);

  return null;
}
```

5.4 必要な環境変数

バックエンドは、フロントエンドが以下の環境変数を設定できるようにする必要がある。

Plain Text

```
# API設定
VITE_API_BASE_URL=https://api.perpx.fi
VITE_WS_BASE_URL=wss://api.perpx.fi/ws

# スマートコントラクトアドレス
VITE_PERPETUAL_TRADING_ADDRESS=0x1234567890abcdef1234567890abcdef12345678
VITE_LIQUIDITY_POOL_BTCUSDT_ADDRESS=0xabcdef1234567890abcdef1234567890abcdef1
2
VITE_LIQUIDITY_POOL_ETHUSDT_ADDRESS=0x9876543210fedcba9876543210fedcba9876543
2
VITE_PRICE_ORACLE_ADDRESS=0xfedcba9876543210fedcba9876543210fedcba98
VITE_LIQUIDATION_ENGINE_ADDRESS=0x1111222233334444555566667777888899990000

# ブロックチェーン設定
VITE_CHAIN_ID=42161
VITE_RPC_URL=https://arb1.arbitrum.io/rpc
```

```
# 機能フラグ
VITE_ENABLE_TESTNET=false
VITE_ENABLE_ANALYTICS=true
```

5.5 エラーハンドリング

既存のフロントエンドは、以下のエラーを適切に処理する必要がある 。

エラータイプ:

- ウォレット接続エラー
- トランザクション拒否
- ガス不足
- 流動性不足
- 無効なパラメータ
- ネットワークエラー
- APIエラー

実装例:

TypeScript

```
try {
  await writeContract({...});
} catch (error) {
  if (error.message.includes('User rejected')) {
    toast.error('Transaction rejected by user');
  } else if (error.message.includes('insufficient funds')) {
    toast.error('Insufficient funds for gas');
  } else if (error.message.includes('INSUFFICIENT_LIQUIDITY')) {
    toast.error('Insufficient liquidity in pool');
  } else {
    toast.error(`Transaction failed: ${error.message}`);
  }
}
```

5.6 ローディング状態

既存のフロントエンドは、以下のローディング状態を表示する。

ローディング状態:

- データ取得中
- トランザクション送信中

- ブロックチェーン確認中

実装例:

TypeScript

```
import { Skeleton } from '@/components/ui/skeleton';

function PositionList() {
  const { data: positions, isLoading } = useQuery({
    queryKey: ['positions', address],
    queryFn: () => fetch(`/api/v1/positions/${address}`)
  });

  if (isLoading) {
    return (
      <div className="space-y-4">
        <Skeleton className="h-20 w-full" />
        <Skeleton className="h-20 w-full" />
        <Skeleton className="h-20 w-full" />
      </div>
    );
  }

  return (
    <div>
      {positions.map(position => (
        <PositionCard key={position.id} position={position} />
      ))}
    </div>
  );
}
```

---

6. セキュリティと監査

6.1 スマートコントラクト監査

6.1.1 監査要件

PerpXは、本番環境デプロイ前に**最低2社の外部監査**を完了する必要がある。

監査会社の選定基準:

- DeFiプロトコルの監査実績
- パーペチュアルDEXの監査経験

- 業界での評判
- 監査レポートの公開

推奨監査会社:

監査会社	実績	費用目安	期間
Trail of Bits	dYdX, Compound, Uniswap	$50,000 - $100,000	4-6週間
OpenZeppelin	Aave, Chainlink, Coinbase	$40,000 - $80,000	3-5週間
Certora	dYdX, Aave, Compound	$60,000 - $120,000	5-7週間
Peckshield	GMX, Binance, Huobi	$30,000 - $60,000	3-4週間
ABDK	GMX, Balancer, Gnosis	$40,000 - $70,000	4-5週間

GMX V2の監査実績:

- Peckshield: 2023年6月
- ABDK: 2023年7月

dYdX V4の監査実績:

- Trail of Bits: 2023年8月
- Certora: 2023年9月

6.1.2 監査スコープ

監査は以下のコントラクトを対象とする。

対象コントラクト:

- LiquidityPool.sol（3ファイル）
- PerpetualTrading.sol（5ファイル）
- PriceOracle.sol（2ファイル）
- LiquidationEngine.sol（2ファイル）
- **合計**: 12ファイル、約3,000行

監査項目:

- 再入攻撃（Reentrancy）

- 整数オーバーフロー/アンダーフロー
- アクセス制御
- 価格操作
- フロントランニング
- ガス最適化
- ロジックエラー
- 状態管理
- イベント発行
- エラーハンドリング

6.1.3 監査プロセス

ステップ1: 準備（1週間）

- コードのフリーズ
- ドキュメントの準備
- テストカバレッジの確認（>90%）
- 監査会社への提出

ステップ2: 初回監査（3-5週間）

- 監査会社がコードをレビュー
- 脆弱性の発見と報告
- 初回監査レポートの受領

ステップ3: 修正（1-2週間）

- 発見された脆弱性の修正
- 修正コードのテスト
- 修正内容の文書化

ステップ4: 再監査（1-2週間）

- 修正コードの再レビュー
- 最終監査レポートの受領
- 公開用レポートの作成

ステップ5: 公開（1週間）

- 監査レポートの公開

- コミュニティへの報告
- メディアへの発表

合計期間: 7-11週間

6.1.4 監査後の対応

監査レポートの公開:

- GitHubリポジトリに公開
- 公式ウェブサイトに掲載
- Twitterで発表

継続的な監査:

- コード変更時の再監査
- 定期的なセキュリティレビュー（6ヶ月ごと）

6.2 バグバウンティプログラム

6.2.1 プログラム概要

PerpXは、セキュリティ研究者がバグを報告するためのバグバウンティプログラムを開始する。

プラットフォーム: Immunefi（GMX、dYdX、Hyperliquidも使用）

報酬総額: $500,000

報酬レベル:

深刻度	説明	報酬	例
Critical	資金の直接的な損失	$50,000 - $100,000	再入攻撃、価格操作
High	資金の間接的な損失	$10,000 - $50,000	アクセス制御の欠陥
Medium	プロトコルの機能不全	$2,000 - $10,000	ロジックエラー
Low	軽微な問題	$500 - $2,000	ガス最適化

対象範囲:

- スマートコントラクト
- APIサーバー
- インデクサー

- キーパーボット
- フロントエンド（XSS、CSRF等）

除外範囲:

- 既知の問題
- 監査レポートに記載された問題
- DoS攻撃
- ソーシャルエンジニアリング

6.2.2 報告プロセス

ステップ1: 報告

- Immunefiプラットフォームで報告
- 脆弱性の詳細を記載
- 再現手順を提供
- 影響範囲を説明

ステップ2: トリアージ（24時間以内）

- PerpXチームが報告を確認
- 深刻度を評価
- 報告者に確認メッセージを送信

ステップ3: 修正（1-7日）

- 脆弱性を修正
- テストを実施
- デプロイを準備

ステップ4: 報酬支払い（7-14日）

- 報酬額を決定
- 報告者に支払い
- 公開（報告者の同意があれば）

GMX V2のバグバウンティ:

- 報酬総額: $1,000,000
- 最高報酬: $250,000

dYdX V4のバグバウンティ:

- 報酬総額: $2,000,000

- 最高報酬: $1,000,000

6.3 保険基金

6.3.1 保険基金の目的

保険基金は、予期しない損失をカバーするための準備金である。

カバーする損失:

- 清算失敗による損失
- オラクル障害による損失
- スマートコントラクトのバグによる損失

初期資金: $100,000（USDC）

資金源:

- プロトコル収益の10%
- 清算ペナルティの一部
- ガバナンストークンの販売（Phase 2）

6.3.2 保険基金の管理

管理方法:

- マルチシグウォレット（3-of-5）
- 透明性のある資金管理
- 定期的な報告（月次）

使用条件:

- ガバナンス投票による承認
- 損失の証明
- 補償額の決定

GMX V2の保険基金:

- 初期資金: $1,000,000
- 現在残高: $5,000,000以上

6.4 緊急停止メカニズム

6.4.1 サーキットブレーカー

異常な状況を検知した場合、プロトコルを一時停止する。

発動条件:

- 価格変動が10%を超える（1分以内）
- オラクル障害
- 大量の清算
- スマートコントラクトのバグ検知

停止する機能:

- 新規ポジションの開設
- ポジションの増加
- 流動性の引き出し

継続する機能:

- ポジションのクローズ
- 清算

再開条件:

- マルチシグによる承認
- 問題の解決確認

6.4.2 マルチシグガバナンス

重要な操作は、マルチシグウォレットによる承認が必要である。

マルチシグ構成: 3-of-5

署名者:

- PerpXチーム: 2名
- 外部アドバイザー: 2名
- コミュニティ代表: 1名

承認が必要な操作:

- スマートコントラクトのアップグレード
- パラメータの変更（レバレッジ、手数料等）
- サーキットブレーカーの再開
- 保険基金の使用

タイムロック: 48時間

---

7. テストと品質保証

7.1 テスト戦略

7.1.1 テストレベル

PerpXは、以下のテストレベルを実施する。

テストレベル	目的	カバレッジ目標	実施頻度
ユニットテスト	個別関数の動作確認	>90%	コミットごと
統合テスト	コントラクト間の連携確認	>80%	プルリクエストごと
E2Eテスト	フロントエンドからの完全なフロー確認	主要フロー100%	デプロイ前
ロードテスト	パフォーマンスとスケーラビリティ確認	-	週次
セキュリティテスト	脆弱性の検出	-	月次

7.1.2 ユニットテスト

対象: スマートコントラクトの個別関数

フレームワーク: Hardhat + Chai

テストケース例（LiquidityPool.deposit）:

- 正常系: 預金が成功し、プールトークンが発行される
- 異常系: 預金額が0の場合、エラーが発生する
- 異常系: 担保トークンが無効な場合、エラーが発生する
- 境界値: 最小預金額（100 USDC）で預金できる
- 境界値: 最小預金額未満（99 USDC）で預金できない

カバレッジ目標: >90%

実施頻度: コミットごと

7.1.3 統合テスト

対象: コントラクト間の連携

テストシナリオ例:

1. LPが流動性を預金
2. トレーダーがポジションを開設
3. 価格が変動
4. トレーダーがポジションをクローズ
5. LPが流動性を引き出し

検証項目:

- プール流動性が正しく更新される
- ポジションの損益が正確に計算される
- LPの損益が正確に計算される
- イベントが正しく発行される

カバレッジ目標: >80%

実施頻度: プルリクエストごと

7.1.4 E2Eテスト

対象: フロントエンドからの完全なフロー

フレームワーク: Playwright

テストシナリオ例:

1. ウォレット接続
2. 市場選択（BTC/USDT）
3. ポジション開設（ロング、$50,000、10倍レバレッジ）
4. トランザクション確認
5. ポジション一覧に表示されることを確認
6. 価格更新を確認
7. 未実現損益を確認
8. ポジションクローズ
9. 取引履歴に表示されることを確認

検証項目:

- UIが正しく表示される
- トランザクションが成功する
- データがリアルタイムで更新される

- エラーが適切に処理される

カバレッジ目標: 主要フロー100%

実施頻度: デプロイ前

7.1.5 ロードテスト

対象: APIサーバー、WebSocket、データベース

ツール: k6

テストシナリオ:

- 同時接続数: 1,000
- リクエスト数: 10,000 req/sec
- 継続時間: 10分

検証項目:

- API応答時間 < 500ms
- WebSocket遅延 < 100ms
- エラー率 < 1%
- CPU使用率 < 80%
- メモリ使用率 < 80%

実施頻度: 週次

7.1.6 セキュリティテスト

対象: スマートコントラクト、APIサーバー、フロントエンド

ツール:

- Slither: 静的解析
- Mythril: シンボリック実行
- Echidna: ファジングテスト
- OWASP ZAP: Webアプリケーションスキャン

検証項目:

- 再入攻撃
- 整数オーバーフロー
- アクセス制御
- XSS、CSRF

- SQLインジェクション

実施頻度: 月次

7.2 テスト環境

7.2.1 ローカル環境

構成:

- Hardhat Network: ローカルブロックチェーン
- PostgreSQL: ローカルデータベース
- Redis: ローカルキャッシュ

用途: 開発中のユニットテスト、統合テスト

7.2.2 テストネット環境

構成:

- Arbitrum Sepolia: テストネットブロックチェーン
- テストネットAPI: `https://testnet-api.perpx.fi`
- テストネットフロントエンド: `https://testnet.perpx.fi`

用途: E2Eテスト、ユーザーテスト

テストネット運用期間: 4週間以上

7.2.3 ステージング環境

構成:

- Arbitrum One: メインネットブロックチェーン
- ステージングAPI: `https://staging-api.perpx.fi`
- ステージングフロントエンド: `https://staging.perpx.fi`

用途: 本番環境前の最終確認

制限:

- 限定ユーザーのみアクセス可能
- 実際の資金を使用

7.3 継続的インテグレーション（CI）

7.3.1 CI/CDパイプライン

ツール: GitHub Actions

パイプライン:

1. コードのチェックアウト
2. 依存関係のインストール
3. リンター実行（Solhint, ESLint）
4. ユニットテスト実行
5. 統合テスト実行
6. カバレッジレポート生成
7. セキュリティスキャン実行
8. ビルド
9. デプロイ（テストネット/ステージング/本番）

実行トリガー:

- プルリクエスト作成時
- メインブランチへのマージ時
- タグプッシュ時

7.3.2 品質ゲート

マージ条件:

- [ ] すべてのテストが成功
- [ ] カバレッジ >90%（ユニットテスト）
- [ ] カバレッジ >80%（統合テスト）
- [ ] セキュリティスキャンで重大な問題なし
- [ ] コードレビュー承認（2名以上）

---

8. 運用とインフラ

8.1 インフラ構成

8.1.1 クラウドプロバイダー

推奨: AWS（Amazon Web Services）

理由:

- 高い可用性（99.99%）
- グローバルなリージョン展開
- DeFiプロトコルでの実績（GMX、dYdX）
- 豊富なマネージドサービス

代替: Google Cloud Platform、Azure

8.1.2 サーバー構成

本番環境:

コンポーネント	インスタンスタイプ	台数	用途
APIサーバー	t3.xlarge（4 vCPU、16GB RAM）	3	ロードバランサー配下
WebSocketサーバー	t3.large（2 vCPU、8GB RAM）	2	WebSocket専用
インデクサー	t3.xlarge（4 vCPU、16GB RAM）	2	イベント処理
PostgreSQL	db.r5.xlarge（4 vCPU、32GB RAM）	1（マスター）+1（レプリカ）	データベース
Redis	cache.r5.large（2 vCPU、13GB RAM）	1（マスター）+1（レプリカ）	キャッシュ
ClickHouse	r5.2xlarge（8 vCPU、64GB RAM）	1	分析用DB
Grafana/Prometheus	t3.medium（2 vCPU、4GB RAM）	1	監視
合計		**12台**	

月額費用目安: $3,000 - $5,000

8.1.3 ネットワーク構成

ロードバランサー: AWS Application Load Balancer

CDN: CloudFlare（フロントエンド配信）

DNS: Route 53

SSL証明書: Let's Encrypt（自動更新）

8.1.4 ストレージ

データベースバックアップ: S3（日次、7日間保持）

ログ保存: S3（30日間保持）

監査ログ: S3 Glacier（永久保持）

8.2 監視とアラート

8.2.1 監視項目

システムメトリクス:

- CPU使用率
- メモリ使用率
- ディスク使用率
- ネットワーク帯域幅

アプリケーションメトリクス:

- API応答時間
- WebSocket接続数
- データベースクエリ時間
- キャッシュヒット率
- エラー率
- トランザクション成功率

ビジネスメトリクス:

- 取引高（24時間）
- オープンインタレスト
- ユーザー数（アクティブ）
- 流動性（総額）
- 清算件数

8.2.2 アラート設定

Critical（即座に対応）:

- API応答時間 > 2秒
- エラー率 > 5%

- データベース接続失敗
- ブロックチェーン同期遅延 > 20ブロック
- キーパーボット停止

Warning（1時間以内に対応）:

- API応答時間 > 1秒
- エラー率 > 1%
- CPU使用率 > 80%
- メモリ使用率 > 80%
- ディスク使用率 > 80%

Info（監視のみ）:

- 取引高の急増
- ユーザー数の急増
- 清算件数の増加

8.2.3 通知チャネル

Slack: リアルタイム通知

PagerDuty: オンコール対応

Email: 日次レポート

8.3 ログ管理

8.3.1 ログレベル

ERROR: エラー発生時

WARN: 警告発生時

INFO: 重要なイベント

DEBUG: デバッグ情報（本番環境では無効）

8.3.2 ログ形式

構造化ログ（JSON形式）:

JSON

```
{
  "timestamp": "2025-12-09T11:30:00Z",
```

```
  "level": "INFO",
  "service": "api-server",
  "message": "Position opened",
  "data": {
    "positionId": "12345",
    "trader": "0xABCD...EF01",
    "market": "BTCUSDT",
    "size": "50000",
    "leverage": 10
  }
}
```

8.3.3 ログ保存

短期保存（30日）: CloudWatch Logs

長期保存（1年）: S3

監査ログ（永久）: S3 Glacier

8.4 バックアップとリカバリ

8.4.1 バックアップ戦略

データベース:

- 日次フルバックアップ（S3）
- 1時間ごとの増分バックアップ
- 7日間保持

設定ファイル:

- Gitリポジトリで管理
- 変更時に自動コミット

スマートコントラクト:

- Etherscanで検証済みコード公開
- GitHubリポジトリで管理

8.4.2 リカバリ手順

データベース障害:

1. レプリカに切り替え（自動、30秒以内）
2. マスターを復旧

3. レプリケーションを再開

APIサーバー障害:

1. ロードバランサーが自動的に切り離し
2. 新しいインスタンスを起動
3. ヘルスチェック通過後、ロードバランサーに追加

ブロックチェーン同期遅延:

1. RPCエンドポイントを切り替え
2. 同期を再開
3. データの整合性を確認

目標復旧時間（RTO）: 1時間以内

目標復旧時点（RPO）: 1時間以内

8.5 インシデント対応

8.5.1 インシデントレベル

P0（Critical）:

- プロトコルが完全に停止
- 資金の損失リスク
- セキュリティ侵害

P1（High）:

- 主要機能が停止
- パフォーマンス著しく低下
- 一部のユーザーに影響

P2（Medium）:

- 軽微な機能が停止
- 一部のユーザーに影響

P3（Low）:

- 軽微な問題
- ユーザーへの影響なし

8.5.2 対応手順

ステップ1: 検知（5分以内）

- アラートを受信
- インシデントレベルを判定
- オンコール担当者に通知

ステップ2: トリアージ（15分以内）

- 影響範囲を確認
- 原因を特定
- 対応方針を決定

ステップ3: 対応（1時間以内）

- 問題を修正
- サービスを復旧
- 動作確認

ステップ4: 報告（24時間以内）

- インシデントレポート作成
- コミュニティへの報告
- 再発防止策の策定

---

9. キーパーネットワーク

9.1 キーパーの役割

キーパーボットは、PerpXプロトコルの重要なインフラである。主な役割は以下の通り。

清算監視:

- 5秒ごとに全オープンポジションをチェック
- 証拠金率が維持証拠金率（5%）を下回ったポジションを検出
- 清算可能なポジションを清算

価格フィード更新（バックアップ）:

- Chainlinkオラクルの更新を監視
- 更新が遅延した場合、手動で更新

システムメンテナンス:

- プールのリバランス
- 古いデータのアーカイブ

9.2 キーパーボットの実装要件

9.2.1 技術スタック

言語: Node.js + TypeScript

ライブラリ:

- Ethers.js: ブロックチェーン通信
- Axios: API通信
- Winston: ロギング

インフラ:

- AWS EC2: ボット実行
- AWS CloudWatch: 監視

9.2.2 清算ボットの動作フロー

ステップ1: ポジション取得（5秒ごと）

```
Plain Text

GET /api/v1/positions/all?status=open
```

ステップ2: 清算可能性チェック

- 各ポジションの現在価格を取得
- 未実現損益を計算
- 証拠金率を計算
- 証拠金率 < 5%の場合、清算可能

ステップ3: 清算実行

```
Plain Text

liquidatePosition(positionId)
```

ステップ4: 報酬受領

- 清算報酬（1%）を受領
- ガス代を差し引いた純利益を確認

9.2.3 収益性の計算

清算報酬: ポジションサイズの1%

ガス代: 約0.001 ETH（Arbitrum One）

損益分岐点:

- ポジションサイズ > $50（ETH = $3,000の場合）
- 実際には、より大きなポジション（$1,000以上）が清算対象

期待収益:

- 1日あたり清算件数: 10件
- 平均ポジションサイズ: $10,000
- 清算報酬: $10,000 × 1% × 10 = $1,000
- ガス代: 0.001 ETH × 10 × $3,000 = $30
- 純利益: $1,000 - $30 = $970/日

9.3 分散化戦略

9.3.1 誰でも実行可能

キーパーボットは、誰でも実行できるように設計する。

公開情報:

- ボットのソースコード（GitHub）
- 実行手順（ドキュメント）
- 必要な環境変数

参入障壁の低減:

- 最小資金: 0.1 ETH（ガス代）
- 技術的知識: Node.js、Ethers.js

9.3.2 清算競争

複数のキーパーが同時に清算を試みる場合、最初に成功したキーパーが報酬を受け取る。

競争メカニズム:

- ガス価格の入札
- 最速のRPCエンドポイント
- 効率的なアルゴリズム

期待される効果:

- 清算の迅速化

- プロトコルの安定性向上
- 分散化の促進

9.3.3 報酬メカニズム

清算報酬: ポジションサイズの1%

報酬の分配:

- キーパー: 1%
- プール: 0%（清算ペナルティの残り1%はプールに送付）

インセンティブ設計:

- 報酬が十分に高い（1%）
- ガス代を上回る利益
- 競争による効率化

---

10. 流動性ブートストラップ

10.1 初期流動性の確保

10.1.1 目標

MVP成功基準:

- 初期流動性: $500,000以上
- 初日取引高: $1,000,000以上
- 初月ユーザー数: 500人以上

10.1.2 流動性提供者の募集

Phase 1: プライベートラウンド（ローンチ前2週間）

- 対象: VC、エンジェル投資家、戦略的パートナー
- 目標: $300,000
- インセンティブ: 追加APY +5%（3ヶ月間）

Phase 2: パブリックラウンド（ローンチ時）

- 対象: 一般ユーザー
- 目標: $200,000
- インセンティブ: 追加APY +3%（1ヶ月間）

10.1.3 マーケットメイカー

役割:

- 初期流動性の提供
- 取引の活性化
- スプレッドの縮小

契約条件:

- 最小流動性: $100,000
- ロック期間: 3ヶ月
- 報酬: 取引手数料の一部

候補:

- Wintermute
- Jump Trading
- GSR Markets

10.2 インセンティブプログラム

10.2.1 流動性マイニング（Phase 2）

報酬: ガバナンストークン（PerpX Token）

配分:

- 総供給量の10%を流動性マイニングに配分
- 4年間で配布

計算方法:

```
Plain Text

報酬 = (個人の流動性 / 総流動性) × 期間の報酬
```

10.2.2 取引報酬（Phase 2）

報酬: ガバナンストークン（PerpX Token）

配分:

- 総供給量の5%を取引報酬に配分
- 2年間で配布

計算方法:

```
報酬 = (個人の取引高 / 総取引高) × 期間の報酬
```

10.2.3 紹介プログラム

報酬: 取引手数料の10%

条件:

- 紹介されたユーザーが取引を実行
- 紹介者が報酬を受け取る

期間: 永久

10.3 マーケティング戦略

10.3.1 ローンチ前（2週間）

活動:

- Twitterでのティーザー投稿
- Discordコミュニティの構築
- インフルエンサーとの提携
- メディアへのプレスリリース

目標:

- Twitterフォロワー: 5,000人
- Discordメンバー: 1,000人

10.3.2 ローンチ時

活動:

- ローンチイベント（オンライン）
- エアドロップキャンペーン
- 取引コンテスト
- メディア露出

目標:

- 初日ユーザー数: 200人

- 初日取引高: $1,000,000

10.3.3 ローンチ後（1ヶ月）

活動:

- コミュニティイベント
- パートナーシップ発表
- 機能追加（指値注文、追加ペア）
- ガバナンス投票の開始

目標:

- 月間ユーザー数: 500人
- 月間取引高: $50,000,000

---

11. コンプライアンスと法務

11.1 法的構造

11.1.1 事業体

推奨: ケイマン諸島財団（Foundation Company）

理由:

- DeFiプロトコルでの実績（GMX、dYdX）
- 税制上の優遇
- 規制の柔軟性

代替: スイス財団、シンガポール財団

11.1.2 知的財産権

スマートコントラクト: MIT License（オープンソース）

フロントエンド: MIT License（オープンソース）

ブランド: 商標登録（米国、EU、日本）

11.2 規制対応

11.2.1 証券法

判断: PerpXのプールトークン（PLP）は証券に該当する可能性がある

対応:

- 米国ユーザーへの販売制限（Regulation D適用外）
- 適格投資家（Accredited Investor）のみに制限（Phase 2）
- 証券弁護士との相談

11.2.2 AML/KYC

MVP: KYC不要（分散型プロトコル）

Phase 2: 任意KYC（高額取引のみ）

Phase 3: 強制KYC（規制対応）

11.2.3 地域制限

制限対象:

- 米国（証券法）
- 中国（暗号資産禁止）
- 北朝鮮、イラン等（OFAC制裁対象）

実装方法:

- IPアドレスによるブロック
- VPN検知
- ウォレットアドレスのブラックリスト

11.3 利用規約とプライバシーポリシー

11.3.1 利用規約

含める内容:

- サービスの説明
- リスクの開示
- 責任の制限
- 紛争解決
- 準拠法

参考: GMX、dYdX、Hyperliquidの利用規約

11.3.2 プライバシーポリシー

含める内容:

- 収集する情報（ウォレットアドレス、IPアドレス）
- 情報の使用目的
- 第三者への開示
- データの保存期間
- ユーザーの権利

準拠法: GDPR（EU）、CCPA（カリフォルニア州）

---

12. デプロイメント戦略

12.1 デプロイメントフェーズ

12.1.1 Phase 1: テストネット（4週間）

目的: 機能検証、バグ修正

環境: Arbitrum Sepolia

参加者: 限定ユーザー（50人）

活動:

- 全機能のテスト
- バグ報告の収集
- フィードバックの収集
- パフォーマンステスト

成功基準:

- 重大なバグなし
- パフォーマンス目標達成
- ユーザーフィードバック良好

12.1.2 Phase 2: 限定メインネット（2週間）

目的: 本番環境での検証

環境: Arbitrum One

参加者: 限定ユーザー（200人）

制限:

- 最大ポジションサイズ: $10,000

- 最大流動性: $100,000

活動:

- 実際の資金での取引
- 清算の検証
- 流動性提供の検証

成功基準:

- 重大なバグなし
- 清算成功率 >99%
- ユーザーフィードバック良好

12.1.3 Phase 3: 公開ローンチ

目的: 一般公開

環境: Arbitrum One

参加者: 全ユーザー

制限解除:

- ポジションサイズ制限なし
- 流動性制限なし

活動:

- マーケティングキャンペーン
- メディア露出
- コミュニティイベント

成功基準:

- 初日ユーザー数: 200人
- 初日取引高: $1,000,000
- 初月ユーザー数: 500人

12.1.4 Phase 4: マルチチェーン展開（3-6ヶ月後）

目的: 他のチェーンへの展開

対象チェーン:

- Optimism
- Base

- Polygon zkEVM

活動:

- スマートコントラクトの移植
- 監査の実施
- デプロイ

12.2 デプロイメントチェックリスト

12.2.1 テストネットデプロイ前

- [ ] すべてのテストが成功
- [ ] カバレッジ >90%（ユニットテスト）
- [ ] カバレッジ >80%（統合テスト）
- [ ] セキュリティスキャンで重大な問題なし
- [ ] ドキュメント完成
- [ ] フロントエンドビルド成功

12.2.2 メインネットデプロイ前

- [ ] 2社以上の監査完了
- [ ] 監査レポート公開
- [ ] バグバウンティプログラム開始
- [ ] 保険基金設立（$100,000）
- [ ] マルチシグウォレット設定（3-of-5）
- [ ] テストネットで4週間の運用実績
- [ ] 利用規約とプライバシーポリシー公開
- [ ] 地域制限の実装
- [ ] 監視システムの構築
- [ ] インシデント対応手順の策定

12.2.3 公開ローンチ前

- [ ] 限定メインネットで2週間の運用実績
- [ ] 初期流動性確保（$500,000）

- [ ] マーケットメイカー契約
- [ ] マーケティング準備完了
- [ ] メディア露出計画
- [ ] コミュニティ構築（Twitter 5,000人、Discord 1,000人）

---

13. 他のperpDEXとの比較

13.1 機能比較

機能	PerpX（MVP）	GMX V2	dYdX V4	Hyperliquid
取引ペア数	2（BTC, ETH）	10+	35+	50+
最大レバレッジ	100倍	50倍	20倍	100倍
取引手数料	0.08%	0.05-0.07%	0.02-0.05%	0.02-0.05%
維持証拠金率	5%	1%	3%	3.33%
清算ペナルティ	2%	5%	7.5%	2.5%
注文タイプ	成行のみ	成行、指値	成行、指値、ストップ	成行、指値、ストップ
ファンディングレート	Phase 2	あり	あり	あり
流動性提供	あり（GLP型）	あり（GLP）	なし	なし
ブロックチェーン	Arbitrum One	Arbitrum One	Cosmos L1	カスタムL1
オラクル	Chainlink	Chainlink	Pyth + Chainlink	独自オラクル
監査	2社	Peckshield + ABDK	Trail of Bits + Certora	非公開
バグバウンティ	$500,000	$1,000,000	$2,000,000	非公開
保険基金	$100,000	$5,000,000+	$10,000,000+	非公開

13.2 アーキテクチャ比較

項目	PerpX	GMX V2	dYdX V4	Hyperliquid
流動性モデル	独立プール	独立プール	オーダーブック	オーダーブック
注文マッチング	オンチェーン	オンチェーン	オフチェーン	オフチェーン
決済	オンチェーン	オンチェーン	オンチェーン（L1）	オンチェーン（L1）
オラクル	Chainlink	Chainlink	Pyth + Chainlink	独自オラクル
インデクサー	The Graph	The Graph	カスタム	カスタム
レイテンシー	1-2秒	1-2秒	0.5-1秒	0.2秒
ガス代	$0.01-0.05	$0.01-0.05	$0.001-0.01	$0.001-0.01
スループット	1,000 tx/sec	1,000 tx/sec	10,000 tx/sec	100,000 tx/sec

13.3 パフォーマンス比較

項目	PerpX（目標）	GMX V2（実績）	dYdX V4（実績）	Hyperliquid（実績）
注文配置	< 2秒	1-2秒	0.5-1秒	0.2秒
価格更新	< 1秒	1秒	0.5秒	0.2秒
同時ユーザー	100人	10,000人+	50,000人+	100,000人+
日次取引高	$1,000,000（初日）	$500,000,000+	$2,000,000,000+	$1,000,000,000+
総流動性	$500,000（初期）	$500,000,000+	N/A（オーダーブック）	N/A（オーダーブック）

13.4 セキュリティ比較

| 項目 | PerpX | GMX V2 | dYdX V4 | Hyperliqui