Reverse Engineering - Polymarket Arbitrage Bot Python

Date : Janvier 2025 **Source :** `E:\developpement\workspace\crypto\polymarket-arbitrage-bot`

---

1. Vue d'Ensemble

1.1 Structure du Projet Python

```
polymarket-arbitrage-bot/
├── src/                        # Core library
│   ├── __init__.py
│   ├── bot.py                  # TradingBot - Interface principale
│   ├── client.py               # ClobClient, RelayerClient - API REST
│   ├── websocket_client.py     # MarketWebSocket - Streaming temps réel
│   ├── signer.py               # OrderSigner - EIP-712 signatures
│   ├── config.py               # Configuration YAML/env
│   ├── crypto.py               # Encryption clés (PBKDF2 + Fernet)
│   ├── gamma_client.py         # Client API Gamma (markets)
│   ├── http.py                 # HTTP utilities
│   └── utils.py                # Utilitaires divers
├── apps/                       # Applications/Stratégies
│   ├── base_strategy.py        # Classe de base stratégies
│   ├── flash_crash_strategy.py # Stratégie Flash Crash
│   ├── flash_crash_runner.py   # Runner pour Flash Crash
│   └── orderbook_viewer.py     # Visualisation orderbook
├── lib/                        # Composants partagés
│   ├── market_manager.py       # Gestion marchés + WebSocket
│   ├── position_manager.py     # Tracking positions + TP/SL
│   ├── price_tracker.py        # Historique prix + détection patterns
│   └── terminal_utils.py       # TUI utilities
├── config.example.yaml
└── requirements.txt
```

1.2 Statistiques Code

Module	Lignes	Fonction
`client.py`	769	API REST (CLOB + Relayer)
`websocket_client.py`	692	WebSocket streaming
`bot.py`	646	Interface trading
`base_strategy.py`	460	Framework stratégies
`config.py`	447	Configuration
`signer.py`	349	Signatures EIP-712
`position_manager.py`	333	Gestion positions

`flash_crash_strategy.py`	225	Stratégie Flash Crash
Total	**~4,000**	Codebase complète

---

2. APIs Polymarket

2.1 REST API - CLOB

Base URL: `https://clob.polymarket.com`

Endpoint	Méthode	Description	Auth
`/book`	GET	Orderbook d'un token	Non
`/price`	GET	Prix marché	Non
`/data/orders`	GET	Ordres ouverts	L2 HMAC
`/data/order/{id}`	GET	Détail ordre	L2 HMAC
`/data/trades`	GET	Historique trades	L2 HMAC
`/order`	POST	Placer ordre	L2 HMAC + Builder
`/order`	DELETE	Annuler ordre	L2 HMAC
`/orders`	DELETE	Annuler plusieurs	L2 HMAC
`/cancel-all`	DELETE	Annuler tout	L2 HMAC
`/cancel-market-orders`	DELETE	Annuler par marché	L2 HMAC
`/auth/derive-api-key`	GET	Dériver API key	L1 EIP-712
`/auth/api-key`	POST	Créer API key	L1 EIP-712

2.2 REST API - Relayer (Gasless)

Base URL: `https://relayer-v2.polymarket.com`

Endpoint	Méthode	Description
`/deploy`	POST	Déployer Safe wallet
`/approve-usdc`	POST	Approuver USDC
`/approve-token`	POST	Approuver token ERC-1155

2.3 WebSocket API

URL: `wss://ws-subscriptions-clob.polymarket.com/ws/market`

Message Subscribe:

```
{
  "assets_ids": ["token_id_1", "token_id_2"],
  "type": "MARKET"
}
```

Events reçus:

Event	Description	Données
`book`	Snapshot orderbook	`asset_id`, `bids`, `asks`, `timestamp`
`price_change`	Changement prix	`asset_id`, `price`, `size`, `side`, `best_bid`, `best_ask`
`last_trade_price`	Dernier trade	`asset_id`, `price`, `size`, `side`, `timestamp`
`tick_size_change`	Changement tick	Paramètres tick

---

3. Authentification

3.1 L1 Authentication (EIP-712)

Utilisée pour créer/dériver les API credentials.

Domain:

```
{
  "name": "ClobAuthDomain",
  "version": "1",
  "chainId": 137
}
```

Message Types:

```
{
  "ClobAuth": [
    {"name": "address", "type": "address"},
    {"name": "timestamp", "type": "string"},
    {"name": "nonce", "type": "uint256"},
    {"name": "message", "type": "string"}
  ]
}
```

Message Data:

```
{
  "address": "0xYourAddress",
  "timestamp": "1234567890",
  "nonce": 0,
  "message": "This message attests that I control the given wallet"
}
```

3.2 L2 Authentication (HMAC)

Utilisée pour les requêtes API authentifiées.

Headers:

```
POLY_ADDRESS: Safe address
POLY_API_KEY: API key (de derive_api_key)
POLY_TIMESTAMP: Unix timestamp
POLY_PASSPHRASE: Passphrase
POLY_SIGNATURE: HMAC-SHA256(timestamp + method + path + body, secret)
```

3.3 Builder Authentication (HMAC)

Pour les transactions gasless via Builder Program.

Headers:

```
POLY_BUILDER_API_KEY: Builder API key
POLY_BUILDER_TIMESTAMP: Unix timestamp
POLY_BUILDER_PASSPHRASE: Builder passphrase
POLY_BUILDER_SIGNATURE: HMAC-SHA256(timestamp + method + path + body, api_secret)
```

---

4. Signature d'Ordres (EIP-712)

4.1 Order Types

```
{
  "Order": [
    {"name": "salt", "type": "uint256"},
    {"name": "maker", "type": "address"},
    {"name": "signer", "type": "address"},
    {"name": "taker", "type": "address"},
    {"name": "tokenId", "type": "uint256"},
    {"name": "makerAmount", "type": "uint256"},
    {"name": "takerAmount", "type": "uint256"},
    {"name": "expiration", "type": "uint256"},
    {"name": "nonce", "type": "uint256"},
    {"name": "feeRateBps", "type": "uint256"},
    {"name": "side", "type": "uint8"},
    {"name": "signatureType", "type": "uint8"}
  ]
}
```

4.2 Calcul des Montants

```
# USDC = 6 decimals
USDC_DECIMALS = 6

# Pour un ordre BUY de 10 shares à 0.65
```

```
size = 10.0
price = 0.65

maker_amount = int(size * price * 10**USDC_DECIMALS)  # 6500000
taker_amount = int(size * 10**USDC_DECIMALS)          # 10000000

side_value = 0  # BUY = 0, SELL = 1
```

4.3 Structure Ordre Signé

```
{
  "order": {
    "tokenId": "123456789...",
    "price": 0.65,
    "size": 10.0,
    "side": "BUY",
    "maker": "0xSafeAddress",
    "nonce": 1706123456,
    "feeRateBps": 0,
    "signatureType": 2
  },
  "signature": "0x...",
  "signer": "0xSignerAddress"
}
```

---

5. Stratégies Trading

5.1 Base Strategy (Pattern)

```
┌─────────────────────────────────────────────────────────┐
│                    BaseStrategy                         │
├─────────────────────────────────────────────────────────┤
│ Components:                                             │
│   • MarketManager  → WebSocket + market discovery       │
│   • PriceTracker   → Historique prix + pattern detection│
│   • PositionManager → Positions + TP/SL                 │
│   • TradingBot     → Order execution                    │
├─────────────────────────────────────────────────────────┤
│ Lifecycle:                                              │
│   start() → run() → stop()                              │
│                                                         │
│ Callbacks (à implémenter):                              │
│   • on_book_update(snapshot)  → Chaque update orderbook │
│   • on_tick(prices)           → Chaque tick (100ms)     │
│   • render_status(prices)     → Affichage TUI           │
│                                                         │
│ Hooks optionnels:                                       │
│   • on_market_change()                                  │
```

```
│   • on_connect() / on_disconnect()                          │
└─────────────────────────────────────────────────────────────┘
```

5.2 Flash Crash Strategy

Logique:

1. Surveille les marchés 15min (BTC, ETH, SOL, XRP)
2. Détecte drops de probabilité > seuil dans fenêtre temps
3. Achète le côté qui a crashé (mean reversion)
4. Gère position avec TP/SL

Configuration:

```python
@dataclass
class FlashCrashConfig:
    coin: str = "ETH"              # Crypto à trader
    drop_threshold: float = 0.30   # Drop 30% pour trigger
    size: float = 5.0              # $5 par trade
    take_profit: float = 0.10      # +$0.10 TP
    stop_loss: float = 0.05        # -$0.05 SL
    price_lookback_seconds: int = 10  # Fenêtre détection
    max_positions: int = 1         # 1 position max
```

Flow:

```
WebSocket → book event
    ↓
PriceTracker.record(side, mid_price)
    ↓
on_tick() → PriceTracker.detect_flash_crash()
    ↓
Si flash crash détecté:
    execute_buy(side, price)
    PositionManager.open_position()
    ↓
on_tick() → PositionManager.check_all_exits()
    ↓
Si TP ou SL atteint:
    execute_sell(position, price)
    PositionManager.close_position()
```

---

6. Position Management

6.1 Position Structure

```python
@dataclass
class Position:
    id: str
    side: str              # "up" ou "down"
    token_id: str
```

```
    entry_price: float
    size: float
    entry_time: float
    order_id: Optional[str]

    # TP/SL
    take_profit_delta: float = 0.10
    stop_loss_delta: float = 0.05

    # Calculés
    take_profit_price = entry_price + take_profit_delta
    stop_loss_price = entry_price - stop_loss_delta
```

6.2 Exit Logic

```
def check_exit(position, current_price):
    if current_price >= position.take_profit_price:
        return "take_profit"
    if current_price <= position.stop_loss_price:
        return "stop_loss"
    return None
```

---

7. Configuration

7.1 Variables d'Environnement

Variable	Description
`POLY_PRIVATE_KEY`	Clé privée (hex)
`POLY_PROXY_WALLET`	Adresse Safe/Proxy
`POLY_RPC_URL`	URL RPC Polygon
`POLY_BUILDER_API_KEY`	Builder API key
`POLY_BUILDER_API_SECRET`	Builder API secret
`POLY_BUILDER_API_PASSPHRASE`	Builder passphrase
`POLY_CLOB_HOST`	Host CLOB API
`POLY_CHAIN_ID`	Chain ID (137)
`POLY_DATA_DIR`	Dossier credentials
`POLY_LOG_LEVEL`	Niveau log

7.2 Config YAML

```yaml
safe_address: "0x..."
rpc_url: "https://polygon-rpc.com"

clob:
  host: "https://clob.polymarket.com"
  chain_id: 137
  signature_type: 2

relayer:
  host: "https://relayer-v2.polymarket.com"
  tx_type: "SAFE"

builder:
  api_key: "..."
  api_secret: "..."
  api_passphrase: "..."

data_dir: "credentials"
log_level: "INFO"
```

---

8. Mapping Python → Rust

8.1 Modules

Python	Rust Crate	Description
`src/client.py`	`trading-engine/api`	REST clients
`src/websocket_client.py`	`trading-engine/websocket`	WebSocket client
`src/bot.py`	`trading-engine/bot`	Trading interface
`src/signer.py`	`trading-engine/crypto`	EIP-712 signing
`src/config.py`	`common/config`	Configuration
`apps/base_strategy.py`	`trading-engine/strategy`	Strategy trait
`apps/flash_crash_strategy.py`	`trading-engine/strategy/flash_crash`	Flash crash impl
`lib/position_manager.py`	`trading-engine/position`	Position management

8.2 Types

Python	Rust
`dataclass`	`struct + derive`
`Dict[str, Any]`	`HashMap<String, Value>` / struct
`List[T]`	`Vec<T>`
`Optional[T]`	`Option<T>`

`float`	`Decimal` (rust_decimal)
`asyncio`	`tokio`
`websockets`	`tokio-tungstenite`
`requests`	`reqwest`
`eth_account`	`alloy`

8.3 Traits Rust

```rust
// Strategy trait (équivalent BaseStrategy)
#[async_trait]
pub trait Strategy: Send + Sync {
    async fn on_book_update(&mut self, snapshot: &OrderbookSnapshot);
    async fn on_tick(&mut self, prices: &HashMap<String, Decimal>);
    fn render_status(&self, prices: &HashMap<String, Decimal>);

    // Optionnel
    fn on_market_change(&mut self, _old: &str, _new: &str) {}
    fn on_connect(&mut self) {}
    fn on_disconnect(&mut self) {}
}
```

---

9. Points d'Attention Migration

9.1 Précision Numérique

Python: `float` (imprécis) **Rust:** `rust_decimal::Decimal` (précision financière)

```rust
use rust_decimal::Decimal;
use rust_decimal_macros::dec;

let price = dec!(0.65);
let size = dec!(10.0);
let cost = price * size; // Exact: 6.50
```

9.2 Gestion Erreurs

Python: Exceptions **Rust:** `Result<T, E>` + `?` operator

```rust
pub async fn place_order(&self, order: Order) -> Result<OrderResult, TradingError> {
    let signed = self.signer.sign_order(&order)?;
    let response = self.client.post_order(signed).await?;
    Ok(OrderResult::from_response(response))
}
```

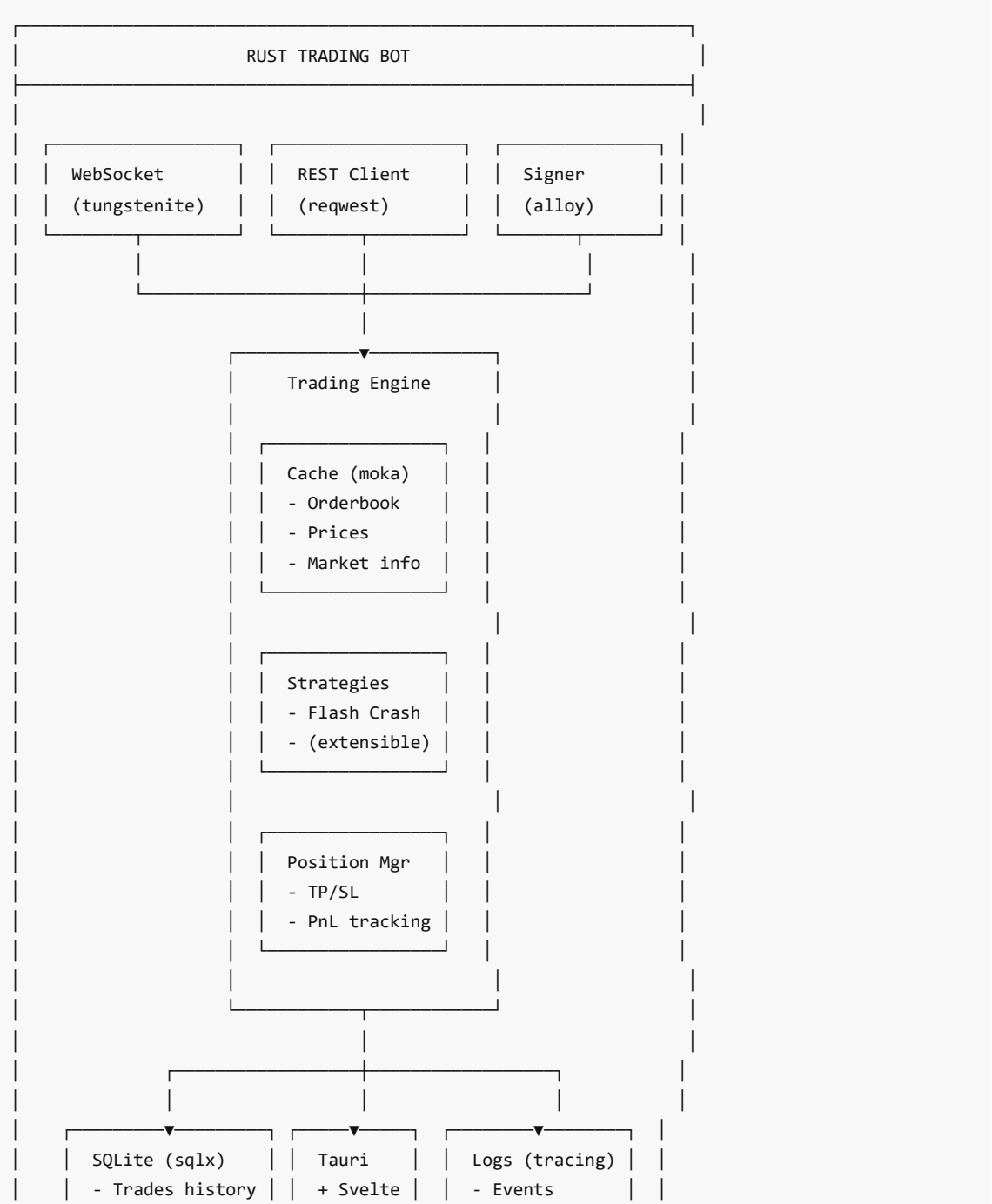
9.3 Concurrence

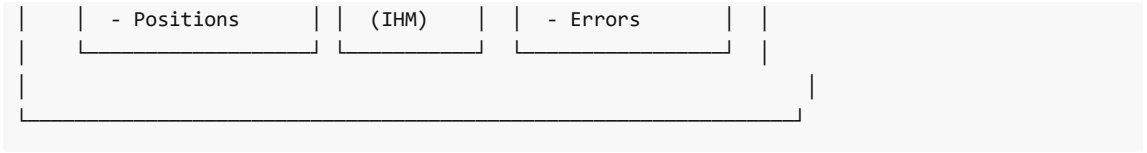
Python: `asyncio` (single-threaded) **Rust:** `tokio` (multi-threaded par défaut)

```rust
// Rust permet le vrai parallélisme
let (orderbook, positions) = tokio::join!(
    self.fetch_orderbook(token_id),
    self.fetch_positions()
);
```

---

10. Résumé Architecture Cible Rust




- Positions
(IHM)
- Errors