AURORA Project Codex v2.0

AURORA - Project Codex v2.0

Personal Investment Intelligence System

Data: 02/01/2026 **Versione:** 2.0 (Revised & Extended) **Scopo:** Specifica tecnica completa per sviluppo sistema AURORA in locale con migrazione VPS

Focus fase iniziale (0-3/4 anni): ETF-only, PAC 150-200 €/mese, profilo aggressivo

---

Executive Summary

AURORA è un sistema operativo personale per investimenti che:

- **Definisce** una Investment Policy Statement (IPS) versionata e immutabile
- **Seleziona** ETF candidati tramite scoring quantitativo riproducibile
- **Propone** portafogli coerenti con vincoli e tolleranze definite
- **Gestisce** PAC mensile con ribilanciamento contributi-only
- **Monitora** drift e genera alert rilevanti

Nella fase successiva (3/4+ anni), AURORA estenderà l'universo a ETF + azioni con screener Quality per settore.

Principi Chiave

1. **Auditabilità**: ogni proposta ha `data_asof`, `run_id`, hash input/output
2. **Trasparenza**: niente black box - ogni decisione include rationale numerica
3. **Policy-driven**: decisioni guidate da vincoli formali, non opinioni

4. **Automazione prudente**: alert utili e rilevanti, non rumore

---

1. Architettura di Sistema

1.1 Componenti

```
┌──────────────────────────────────────────────────────────────────┐
│                         AURORA System                            │
├─────────────┬──────────────┬─────────────┬───────────────────────┤
│             │              │             │                       │
│    UI       │     API      │    Queue    │      Engine           │
│  (Next.js)  │   (NestJS)   │   (Redis)   │   (Python Worker)     │
│             │              │             │                       │
└──────┬──────┴──────┬───────┴──────┬──────┴───────┬───────────────┘
       │             │              │              │
       └─────────────┼──────────────┼──────────────┘
                     │              │
                     │              │
           ┌─────────▼──────────────▼─────────┐
           │  Database Layer                  │
           │ - Postgres 16                    │
           │ - TimescaleDB (time series)      │
           │ - Analytics: DuckDB (opz.)       │
           └──────────────────────────────────┘
```

1.2 Stack Tecnologico Dettagliato

Frontend (UI)

- **Framework**: Next.js 14+ (App Router)
- **UI Components**: shadcn/ui (Radix + Tailwind)
- **State**: TanStack Query (React Query)
- **Tables**: TanStack Table
- **Charts**: Recharts o Chart.js
- **Forms**: React Hook Form + Zod validation

Backend (API)

- **Framework**: NestJS (TypeScript)
- **ORM**: Prisma
- **Queue**: BullMQ (Redis-backed)
- **Validation**: Zod schemas
- **Auth**: Passport.js (local strategy - uso personale)
- **API Docs**: OpenAPI/Swagger auto-generato

Analytics Engine

- **Runtime**: Python 3.11+
- **Framework**: FastAPI (health endpoints)
- **Worker**: BullMQ consumer (Python)
- **Data**: pandas, numpy
- **Finance**: yfinance, pandas-datareader
- **Quality**: pydantic per validation

Infrastructure

- **Orchestrazione**: Docker Compose
- **Database**: PostgreSQL 16 + TimescaleDB extension
- **Cache/Queue**: Redis 7+
- **Reverse Proxy (VPS)**: Caddy o Nginx
- **Observability**: Grafana Loki + Prometheus (opzionale VPS)

1.3 Flusso di Dati

```
User Action (UI)
    ↓
API Endpoint (NestJS)
    ↓
Job Enqueue (Redis/BullMQ)
    ↓
Python Worker consuma job
    ↓
Calcoli + Data Fetch
    ↓
Risultati salvati (Postgres)
    ↓
UI polling/webhook aggiorna vista
```

2. Investment Policy Statement (IPS)

2.1 Struttura IPS v1

L’IPS è la “costituzione” del sistema. Ogni versione è immutabile e versionata semanticamente.

Parametri IPS v1:

Parametro	Valore	Note
Profilo	Aggressivo	Orizzonte 4-5 anni minimo
Strumenti iniziali	Solo ETF	ETF UCITS armonizzati
Orizzonte fase 1	3-4 anni	Poi estensione a ETF + azioni
PAC mensile	150-200 EUR	Default: 200 EUR
Frequenza acquisti	Mensile	Primo giorno lavorativo del mese
Ribilanciamento	Contributi-only	Vendite solo se drift > 2× banda
Banda asset class	±5% assoluto	Configurabile
Banda strumento	±3% assoluto	Configurabile
Min strumenti	1	Semplicità preferita
Max strumenti	3	Fase iniziale
Max peso singolo	100%	Permette Single ETF Core

2.2 Default Portfolio Strategy

Opzione A (consigliata): Single ETF Core - 100% ETF azionario globale (es. MSCI World o FTSE All-World) - Massima semplicità, minimi costi cognitivi - Tracking error minimo vs benchmark

Opzione B: Core + Stabilizzatore - 90% ETF azionario globale - 10% ETF obbligazionario globale hedged EUR o monetario - Riduce volatilità emotiva in fasi correttive

2.3 Schema IPS (JSON)

```
{
  "version": "1.0.0",
  "profile": "aggressivo",
  "phase": "ETF-only",
  "horizon_years": 4,
  "pac_monthly_eur": {
    "min": 150,
    "max": 200,
    "default": 200
  },
  "targets": [
    {
      "bucket": "equity_global",
      "weight": 1.00,
      "description": "Azionario globale diversificato"
    }
  ],
  "rebalance": {
    "mode": "contributi-only",
    "frequency": "monthly",
    "bands": {
      "asset_class_abs": 0.05,
      "instrument_abs": 0.03
    },
    "sell_threshold_multiplier": 2.0
  },
  "constraints": {
    "min_instruments": 1,
    "max_instruments": 3,
    "max_single_instrument_weight": 1.00,
    "allowed_domiciles": ["IE", "LU", "DE"],
    "required_ucits": true
  }
}
```

3. Data Sourcing Strategy

3.1 Problema Critico: Fonti Dati ETF

Dati necessari: - Prezzi giornalieri (NAV) - Metriche fondamentali: TER, AUM, Inception Date - Tracking difference vs benchmark (idealmente) - Volume e spread (proxy liquidità) - Holdings/composizione (opzionale fase 1)

3.2 Opzioni Data Provider

Opzione A: Yahoo Finance (FREE)

Pro: - Gratuito, API Python (yfinance) - Copertura ampia ETF europei - Storico prezzi affidabile

Contro: - Metriche limitate (TER spesso mancante) - No tracking difference ufficiale - Necessita mapping ISIN → ticker Yahoo

Implementazione:

```python
import yfinance as yf

def fetch_etf_data(ticker: str):
    etf = yf.Ticker(ticker)
    info = etf.info
    hist = etf.history(period="2y")

    return {
        "price_current": hist['Close'].iloc[-1],
        "ter": info.get('annualReportExpenseRatio', None),
        "aum": info.get('totalAssets', None),
        "avg_volume_3m": hist['Volume'].tail(60).mean(),
        "price_history": hist['Close'].to_dict()
    }
```

Opzione B: JustETF Scraping (FREE with limits)

Pro: - Dati europei completi (TER, TD, AUM) - Categorizzazione accurata - Tracking difference ufficiale

Contro: - Richiede scraping (fragile) - Rate limits - Possibili cambiamenti HTML

Implementazione:

```python
import requests
from bs4 import BeautifulSoup

def scrape_justetf(isin: str):
    url = f"https://www.justetf.com/en/etf-profile.html?isin={isin}"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Parsing specifico (da adattare a struttura DOM)
    ter = soup.select_one('.ter-value').text
    aum = soup.select_one('.aum-value').text

    return {"ter": float(ter), "aum": parse_aum(aum)}
```

Opzione C: EOD Historical Data (PAID - $19.99/mese)

Pro: - API strutturata - Dati completi e affidabili - Supporto ISIN diretto

Contro: - Costo mensile - Overkill per 1-3 ETF

Raccomandazione fase 1: Yahoo Finance + fallback manuale per TER/AUM da JustETF.

3.3 Data Adapter Pattern

```python
# services/engine/src/data/providers/base.py
from abc import ABC, abstractmethod
from typing import Dict, Optional

class ETFDataProvider(ABC):
    @abstractmethod
    def fetch_price_history(self, isin: str, period: str) -> Dict:
        pass

    @abstractmethod
    def fetch_fundamentals(self, isin: str) -> Dict:
        pass

# services/engine/src/data/providers/yahoo.py
class YahooFinanceProvider(ETFDataProvider):
    def __init__(self, isin_mapper: ISINMapper):
        self.mapper = isin_mapper

    def fetch_price_history(self, isin: str, period: str) -> Dict:
        ticker = self.mapper.isin_to_ticker(isin)
        etf = yf.Ticker(ticker)
        return etf.history(period=period)['Close'].to_dict()

    def fetch_fundamentals(self, isin: str) -> Dict:
        ticker = self.mapper.isin_to_ticker(isin)
        etf = yf.Ticker(ticker)
        info = etf.info

        return {
            "ter": info.get('annualReportExpenseRatio'),
            "aum": info.get('totalAssets'),
            "currency": info.get('currency'),
            "provider": info.get('fundFamily'),
            "inception_date": info.get('fundInceptionDate')
        }

# Usage con fallback
class CompositeProvider(ETFDataProvider):
    def __init__(self, primary: ETFDataProvider, fallback: ETFDataProvider):
        self.primary = primary
        self.fallback = fallback

    def fetch_fundamentals(self, isin: str) -> Dict:
        try:
            data = self.primary.fetch_fundamentals(isin)
            if data.get('ter') is None:
                fallback_data = self.fallback.fetch_fundamentals(isin)
                data['ter'] = fallback_data.get('ter')
            return data
        except Exception as e:
            logger.warning(f"Primary provider failed: {e}")
            return self.fallback.fetch_fundamentals(isin)
```

3.4 ISIN Mapping Table

Necessaria per Yahoo Finance (usa ticker, non ISIN).

Schema DB:

```
CREATE TABLE isin_mapping (
    isin VARCHAR(12) PRIMARY KEY,
    yahoo_ticker VARCHAR(20),
    exchange VARCHAR(10),
    verified BOOLEAN DEFAULT false,
    last_verified_at TIMESTAMP
```

```sql
);

-- Esempi
INSERT INTO isin_mapping (isin, yahoo_ticker, exchange, verified) VALUES
('IE00B4L5Y983', 'IWDA.AS', 'AMS', true),  -- iShares Core MSCI World
('IE00BK5BQT80', 'VWCE.DE', 'ETR', true),  -- Vanguard FTSE All-World
('LU1781541179', 'EUNL.DE', 'ETR', true);  -- Amundi MSCI World
```

---

4. Universe Management & ETF Scoring

4.1 Bucket Strategy

Organizzazione per asset class e geografia:

Fase 1 (ETF-only): - `equity_global`: Azionario globale (MSCI World, FTSE All-World) - `equity_us`: Azionario USA (S&P 500) - `equity_europe`: Azionario Europa - `equity_emerging`: Mercati emergenti - `bond_global_hedged`: Obbligazionario globale hedged EUR - `bond_government_eur`: Governativo Eurozona - `money_market_eur`: Monetario EUR

4.2 Filtri Hard (Must-Have)

```python
def apply_hard_filters(etf_universe: List[ETF]) -> List[ETF]:
    """Filtri di ammissibilità non negoziabili"""

    filtered = []
    for etf in etf_universe:
        # 1. UCITS compliance
        if not etf.is_ucits:
            continue

        # 2. Dimensione minima
        if etf.aum_eur < 100_000_000:  # 100M EUR
            continue

        # 3. Anzianità minima
        if etf.age_years < 1:
            continue

        # 4. Domicilio autorizzato
        if etf.domicile not in ['IE', 'LU', 'DE', 'FR']:
            continue

        # 5. Liquidità minima
        if etf.avg_daily_volume < 10_000:  # shares
            continue

        # 6. Dati sufficienti
        if etf.data_completeness < 0.8:  # 80% campi popolati
            continue

        filtered.append(etf)

    return filtered
```

4.3 Scoring Model v1 (0-100)

Componenti:

Componente	Descrizione	Peso	Range
Costi	TER + stima costi impliciti	35%	0-35
Tracking	Tracking Difference vs benchmark	30%	0-30
Liquidità	AUM + volume + spread proxy	20%	0-20
Robustezza	Provider, anzianità, metodo replica	10%	0-10
Preferenze	Accumulo vs Dist, fisica vs sintetica	5%	0-5

Implementazione:

```python
from dataclasses import dataclass
from typing import Optional

@dataclass
class ScoringWeights:
    costs: float = 0.35
    tracking: float = 0.30
    liquidity: float = 0.20
    robustness: float = 0.10
    preferences: float = 0.05

def score_etf(etf: ETF, weights: ScoringWeights = ScoringWeights()) -> dict:
    """
    Calcola score 0-100 per ETF con breakdown dettagliato
    """

    # 1. Costi (0-35): TER più basso = score più alto
    # TER best-in-class: 0.05% → 35 pts
    # TER accettabile: 0.30% → 25 pts
    # TER alto: 0.60%+ → 10 pts
    ter_score = max(0, 35 - (etf.ter * 100 - 0.05) * 2)
    ter_score = min(35, ter_score)

    # 2. Tracking (0-30): tracking difference più bassa = meglio
    # TD negativa (outperform) → bonus
    # TD < 0.10% → 30 pts
    # TD 0.30% → 20 pts
    # TD > 0.60% → 5 pts
    if etf.tracking_difference is not None:
        if etf.tracking_difference < 0:
            tracking_score = 30  # Outperform
        else:
            tracking_score = max(0, 30 - etf.tracking_difference * 100 * 0.5)
            tracking_score = min(30, tracking_score)
    else:
        tracking_score = 15  # Default se dato mancante

    # 3. Liquidità (0-20)
    aum_score = min(10, etf.aum_eur / 1_000_000_000 * 10)  # 10 pts @ 1B EUR
    volume_score = min(10, etf.avg_daily_volume / 100_000 * 10)  # 10 pts @ 100k shares
    liquidity_score = aum_score + volume_score

    # 4. Robustezza (0-10)
    provider_score = {
        'iShares': 4,
        'Vanguard': 4,
        'Xtrackers': 3,
        'Amundi': 3,
        'Invesco': 2
    }.get(etf.provider, 1)

    age_score = min(3, etf.age_years / 5 * 3)  # 3 pts @ 5 anni

    replica_score = 3 if etf.replication_method == 'Physical' else 2

    robustness_score = provider_score + age_score + replica_score

    # 5. Preferenze (0-5)
    pref_score = 0
    pref_score += 3 if etf.distribution_policy == 'Accumulating' else 0
    pref_score += 2 if etf.replication_method == 'Physical' else 0

    # Total
    total_score = (
        ter_score * weights.costs / 0.35 +
        tracking_score * weights.tracking / 0.30 +
        liquidity_score * weights.liquidity / 0.20 +
        robustness_score * weights.robustness / 0.10 +
        pref_score * weights.preferences / 0.05
    )

    return {
        "total": round(total_score, 2),
        "breakdown": {
            "costs": round(ter_score, 2),
            "tracking": round(tracking_score, 2),
            "liquidity": round(liquidity_score, 2),
            "robustness": round(robustness_score, 2),
            "preferences": round(pref_score, 2)
        },
        "red_flags": identify_red_flags(etf)
    }

def identify_red_flags(etf: ETF) -> List[str]:
    """Identifica warning espliciti"""
    flags = []

    if etf.ter > 0.50:
        flags.append(f"TER elevato: {etf.ter:.2%}")

    if etf.aum_eur < 100_000_000:
        flags.append(f"AUM sotto soglia: {etf.aum_eur/1e6:.1f}M EUR")

    if etf.tracking_difference and etf.tracking_difference > 0.40:
        flags.append(f"Tracking difference elevata: {etf.tracking_difference:.2%}")

    if etf.avg_daily_volume < 50_000:
        flags.append("Liquidità bassa")

    if etf.replication_method == 'Synthetic':
        flags.append("Replica sintetica (rischio controparte)")

    return flags
```

4.4 Output Scoring

Per ogni bucket, AURORA genera:

```
{
  "bucket": "equity_global",
  "run_id": "uuid",
  "data_asof": "2026-01-02",
  "candidates_evaluated": 47,
  "candidates_passed_filters": 12,
  "top_3": [
    {
      "rank": 1,
      "isin": "IE00B4L5Y983",
      "name": "iShares Core MSCI World UCITS ETF USD (Acc)",
      "ticker": "IWDA",
      "score": 87.5,
      "breakdown": {
        "costs": 33.0,
        "tracking": 28.5,
        "liquidity": 19.0,
        "robustness": 9.0,
        "preferences": 5.0
      },
      "fundamentals": {
        "ter": 0.20,
        "aum_eur": 62_000_000_000,
        "tracking_difference": 0.05,
        "provider": "iShares",
        "replication": "Physical"
      },
      "red_flags": []
    },
    {
      "rank": 2,
      "isin": "IE00BK5BQT80",
      "name": "Vanguard FTSE All-World UCITS ETF USD (Acc)",
      "ticker": "VWCE",
      "score": 86.0,
      "breakdown": {...},
      "red_flags": []
    },
    {
      "rank": 3,
      "isin": "LU1781541179",
      "name": "Amundi MSCI World UCITS ETF EUR (C)",
      "ticker": "EUNL",
      "score": 82.5,
      "breakdown": {...},
      "red_flags": ["TER leggermente più alto rispetto a top 2"]
    }
  ]
}
```

5. Portfolio Management & PAC Engine

5.1 Algoritmo PAC v1: Contributi-Only

Principio: investire il contributo mensile nello strumento più sotto-target rispetto all'allocazione IPS.

Input: - Targets da IPS: `{instrument_id: target_weight}` - Posizioni correnti: `{instrument_id: {qty, value_eur}}` - Contributo mensile: `contribution_eur`

Output: - Trade list: `[{instrument_id, side, amount_eur}]` - Rationale: drift snapshot, constraints check

Algoritmo:

```
from typing import Dict, List
from dataclasses import dataclass

@dataclass
class Position:
    instrument_id: str
    quantity: float
    current_price_eur: float

    @property
    def value_eur(self) -> float:
        return self.quantity * self.current_price_eur

@dataclass
class Target:
    instrument_id: str
    weight: float

def compute_monthly_pac(
    positions: List[Position],
    targets: List[Target],
    contribution_eur: float,
    ips_bands: dict
) -> dict:
    """
    Calcola proposta PAC mensile con logica contributi-only
    """

    # 1. Calcola valore totale portfolio
    total_value = sum(p.value_eur for p in positions)

    # 2. Calcola pesi correnti e drift
    drifts = []
    for target in targets:
        position = next((p for p in positions if p.instrument_id == target.instrument_id),
        None)

        current_weight = position.value_eur / total_value if position else 0.0
        drift = target.weight - current_weight

        drifts.append({
            "instrument_id": target.instrument_id,
            "target_weight": target.weight,
            "current_weight": current_weight,
            "drift": drift,
            "drift_pct": (drift / target.weight * 100) if target.weight > 0 else 0
        })

    # 3. Seleziona strumento più sotto-target
    max_drift_item = max(drifts, key=lambda d: d["drift"])

    # 4. Genera trade
    trade_list = [{
        "instrument_id": max_drift_item["instrument_id"],
        "side": "BUY",
        "amount_eur": contribution_eur
    }]

    # 5. Verifica vincoli post-trade (simulato)
    new_total_value = total_value + contribution_eur
    simulated_weights = simulate_post_trade_weights(positions, trade_list, new_total_value)
    constraints_passed = check_constraints(simulated_weights, ips_bands)

    return {
        "trade_list": trade_list,
        "rationale": {
            "total_value_pre_trade": total_value,
            "contribution": contribution_eur,
            "drift_snapshot": drifts,
            "selected_instrument": max_drift_item["instrument_id"],
            "selected_drift": max_drift_item["drift"],
            "constraints_passed": constraints_passed
        }
    }

def check_constraints(weights: Dict[str, float], bands: dict) -> dict:
    """Verifica vincoli IPS"""
    violations = []

    # Check asset class bands (esempio semplificato)
    # Assumendo equity_global dovrebbe essere 100%
    equity_weight = sum(w for w in weights.values())
    if abs(equity_weight - 1.0) > bands['asset_class_abs']:
        violations.append({
            "type": "asset_class_drift",
            "current": equity_weight,
            "target": 1.0,
            "band": bands['asset_class_abs']
        })

    # Check singolo strumento
    for instrument_id, weight in weights.items():
        # Assumendo target 100% per single ETF
        if abs(weight - 1.0) > bands['instrument_abs']:
            violations.append({
                "type": "instrument_drift",
                "instrument_id": instrument_id,
                "current": weight,
                "target": 1.0,
                "band": bands['instrument_abs']
            })

    return {
        "passed": len(violations) == 0,
        "violations": violations
    }
```

5.2 Gestione Drift Estremo (Vendite)

Regola: vendite solo se drift > 2× banda configurata.

```
def check_extreme_drift_rebalance(
    positions: List[Position],
    targets: List[Target],
    ips_bands: dict
) -> Optional[List[dict]]:
    """
    Controlla se necessario ribilanciamento tramite vendite
    Ritorna trade list se necessario, None altrimenti
    """

    total_value = sum(p.value_eur for p in positions)
    extreme_drift_multiplier = 2.0
    threshold = ips_bands['asset_class_abs'] * extreme_drift_multiplier

    trades = []

    for target in targets:
        position = next((p for p in positions if p.instrument_id == target.instrument_id),
        None)
        if not position:
            continue

        current_weight = position.value_eur / total_value
        drift = abs(current_weight - target.weight)

        if drift > threshold:
            # Calcola importo da vendere/comprare per tornare a target
            target_value = total_value * target.weight
            current_value = position.value_eur
            rebalance_amount = current_value - target_value

            if rebalance_amount > 0:
                trades.append({
                    "instrument_id": target.instrument_id,
                    "side": "SELL",
                    "amount_eur": rebalance_amount,
                    "reason": f"Drift estremo: {drift:.2%} > {threshold:.2%}"
                })
            else:
                trades.append({
                    "instrument_id": target.instrument_id,
                    "side": "BUY",
                    "amount_eur": abs(rebalance_amount),
                    "reason": f"Drift estremo: {drift:.2%} > {threshold:.2%}"
                })

    return trades if trades else None
```

5.3 Snapshot Mensili

Salvare stato portfolio ogni mese per tracking storico.

```
from datetime import datetime

def create_position_snapshot(
    portfolio_id: str,
    positions: List[Position],
    targets: List[Target]
) -> dict:
    """Crea snapshot mensile posizioni"""

    total_value = sum(p.value_eur for p in positions)
    snapshot_items = []
    for position in positions:
        target = next((t for t in targets if t.instrument_id == position.instrument_id),
        None)
        target_weight = target.weight if target else 0.0
        current_weight = position.value_eur / total_value

        snapshot_items.append({
            "instrument_id": position.instrument_id,
            "quantity": position.quantity,
            "price_eur": position.current_price_eur,
            "value_eur": position.value_eur,
            "weight": current_weight,
            "target_weight": target_weight,
            "drift": target_weight - current_weight
        })

    return {
        "portfolio_id": portfolio_id,
        "snapshot_date": datetime.now().isoformat(),
        "total_value_eur": total_value,
        "items": snapshot_items
    }
```

6. Alert System

6.1 Tipi di Alert

Priority High: 1. Drift asset class > banda configurata 2. Violazione constraint IPS 3. Data quality issue critico (prezzo mancante giorno PAC)

Priority Medium: 4. Concentrazione involontaria (strumento > 80% se target < 80%) 5. Nuova versione IPS disponibile ma non attivata

Priority Low: 6. ETF candidato con score significativamente migliore disponibile 7. TER aumentato su strumento in portfolio

6.2 Implementazione Alert

```
from enum import Enum
from typing import List

class AlertPriority(Enum):
    HIGH = "high"
    MEDIUM = "medium"
    LOW = "low"

@dataclass
class Alert:
    id: str
    priority: AlertPriority
    type: str
    title: str
    message: str
    data: dict
    created_at: datetime
    acknowledged: bool = False

def generate_alerts(
    portfolio_id: str,
    positions: List[Position],
    targets: List[Target],
    ips: dict
) -> List[Alert]:
    """Genera alert rilevanti per portfolio corrente"""

    alerts = []
    total_value = sum(p.value_eur for p in positions)

    # 1. Drift alert
    for target in targets:
        position = next((p for p in positions if p.instrument_id == target.instrument_id),
        None)
        if not position:
            continue

        current_weight = position.value_eur / total_value
        drift = abs(current_weight - target.weight)

        if drift > ips['rebalance']['bands']['asset_class_abs']:
            alerts.append(Alert(
                id=f"drift_{target.instrument_id}_{datetime.now().timestamp()}",
                priority=AlertPriority.HIGH,
                type="drift_breach",
                title="Drift oltre banda configurata",
                message=f"Strumento {target.instrument_id} ha drift {drift:.2%}, banda:
        {ips['rebalance']['bands']['asset_class_abs']:.2%}",
                data={
                    "instrument_id": target.instrument_id,
                    "current_weight": current_weight,
                    "target_weight": target.weight,
                    "drift": drift
                },
                created_at=datetime.now()
            ))

    # 2. Concentrazione alert
    for position in positions:
        weight = position.value_eur / total_value
        if weight > 0.80:  # Concentrazione > 80%
            alerts.append(Alert(
                id=f"concentration_{position.instrument_id}_{datetime.now().timestamp()}",
                priority=AlertPriority.MEDIUM,
                type="concentration",
                title="Concentrazione elevata",
                message=f"Strumento {position.instrument_id} rappresenta {weight:.1%} del
        portfolio",
                data={"instrument_id": position.instrument_id, "weight": weight},
                created_at=datetime.now()
            ))

    return alerts
```

7. Database Schema Dettagliato

7.1 Prisma Schema Completo

```
// packages/db/prisma/schema.prisma

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

// ==================== IPS & Policy ====================

model IpsPolicy {
  id        String   @id @default(uuid())
  userId    String   @unique
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  versions  IpsPolicyVersion[]

  @@map("ips_policy")
}

model IpsPolicyVersion {
  id         String   @id @default(uuid())
  policyId   String
  version    String   // Semantic version: 1.0.0
  config     Json     // IPS JSON structure
  isActive   Boolean  @default(false)
  createdAt  DateTime @default(now())
  activatedAt DateTime?

  policy     IpsPolicy @relation(fields: [policyId], references: [id], onDelete: Cascade)

  @@unique([policyId, version])
  @@index([policyId, isActive])
  @@map("ips_policy_version")
}

// ==================== Instruments ====================

model Instrument {
  id             String   @id @default(uuid())
  isin           String   @unique
  name           String
  ticker         String?
  currency       String   @default("EUR")
  type           String   // "ETF", "STOCK", "BOND"
  category       String?  // "equity_global", "bond_government", etc.
  domicile       String?  // "IE", "LU", etc.
  provider       String?  // "iShares", "Vanguard", etc.
  isUcits        Boolean  @default(false)

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

  etfMetrics     EtfMetrics?
  isinMapping    IsinMapping?
  transactions   Transaction[]
  positions      Position[]
  priceHistory   PriceHistory[]

  @@index([isin])
  @@index([type, category])
  @@map("instrument")
}

model IsinMapping {
  isin            String   @id
  yahooTicker     String?
  exchange        String?
  verified        Boolean  @default(false)
  lastVerifiedAt  DateTime?

  instrument      Instrument @relation(fields: [isin], references: [isin], onDelete:
Cascade)

  @@map("isin_mapping")
}

model EtfMetrics {
  id                    String   @id @default(uuid())
  instrumentId          String   @unique

  ter                   Float?   // Total Expense Ratio (%)
  aum                   Float?   // Assets Under Management (EUR)
  inceptionDate         DateTime?
  replicationMethod     String?  // "Physical", "Synthetic"
  distributionPolicy    String?  // "Accumulating", "Distributing"

  trackingDifference    Float?   // vs benchmark (%)
  trackingError         Float?   // volatility of TD

  avgDailyVolume        Float?   // Last 3 months
  avgSpread             Float?   // Bid-ask spread (%)

  dataCompleteness      Float?   // 0.0 - 1.0
  lastUpdated           DateTime @default(now())

  instrument            Instrument @relation(fields: [instrumentId], references: [id],
onDelete: Cascade)

  @@index([instrumentId])
  @@map("etf_metrics")
}

model PriceHistory {
  id             String   @id @default(uuid())
  instrumentId   String
  date           DateTime @db.Date
  close          Float
  open           Float?
  high           Float?
  low            Float?
  volume         Float?

  instrument     Instrument @relation(fields: [instrumentId], references: [id], onDelete:
Cascade)

  @@unique([instrumentId, date])
  @@index([instrumentId, date])
  @@map("price_history")
}
```

```
// ==================== Portfolio & Transactions ====================

model Portfolio {
  id          String   @id @default(uuid())
  name        String
  type        String   @default("paper") // "paper" | "real"
  userId      String

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

  transactions Transaction[]
  positions    Position[]
  snapshots    PositionSnapshot[]
  proposals    Proposal[]

  @@index([userId, type])
  @@map("portfolio")
}

model Transaction {
  id             String   @id @default(uuid())
  portfolioId    String
  instrumentId   String

  side           String   // "BUY" | "SELL"
  quantity       Float
  priceEur       Float
  feeEur         Float    @default(0)
  totalEur       Float    // quantity * price + fee

  executedAt     DateTime @default(now())
  note           String?

  portfolio      Portfolio  @relation(fields: [portfolioId], references: [id], onDelete:
Cascade)
  instrument     Instrument @relation(fields: [instrumentId], references: [id])

  @@index([portfolioId, executedAt])
  @@index([instrumentId])
  @@map("transaction")
}

model Position {
  id             String   @id @default(uuid())
  portfolioId    String
  instrumentId   String

  quantity       Float
  avgCostEur     Float    // Average purchase price

  updatedAt      DateTime @updatedAt

  portfolio      Portfolio  @relation(fields: [portfolioId], references: [id], onDelete:
Cascade)
  instrument     Instrument @relation(fields: [instrumentId], references: [id])

  @@unique([portfolioId, instrumentId])
  @@index([portfolioId])
  @@map("position")
}

model PositionSnapshot {
  id             String   @id @default(uuid())
  portfolioId    String
  snapshotDate   DateTime @db.Date
  totalValueEur  Float

  items          Json     // Array of {instrument_id, qty, price, value, weight, drift}

  createdAt      DateTime @default(now())

  portfolio      Portfolio @relation(fields: [portfolioId], references: [id], onDelete:
Cascade)

  @@unique([portfolioId, snapshotDate])
  @@index([portfolioId, snapshotDate])
  @@map("position_snapshot")
}

// ==================== Proposals & Engine Runs ====================

model Proposal {
  id              String   @id @default(uuid())
  runId           String   @unique
  portfolioId     String

  type            String   // "ETF_SCORING" | "PORTFOLIO_PROPOSAL" | "MONTHLY_PAC"
  dataAsof        DateTime @db.Date

  tradeList       Json     // Array of {instrument_id, side, amount_eur}
  rationale       Json     // {targets, drift_snapshot, constraints_passed, ...}

  inputHash       String?
  outputHash      String?

  status          String   @default("pending") // "pending" | "executed" | "rejected"

  createdAt       DateTime @default(now())
  executedAt      DateTime?

  portfolio       Portfolio @relation(fields: [portfolioId], references: [id], onDelete:
Cascade)

  @@index([portfolioId, dataAsof])
  @@index([runId])
  @@map("proposal")
}

model EngineRun {
  id              String   @id @default(uuid())
  runId           String   @unique

  type            String   // "ETF_SCORING" | "MONTHLY_PAC" | etc.
  status          String   @default("queued") // "queued" | "running" | "completed" |
"failed"

  inputParams     Json
  result          Json?
  error           String?

  startedAt       DateTime?
  completedAt     DateTime?
  durationMs      Int?

  createdAt       DateTime @default(now())

  @@index([runId])
  @@index([type, status])
  @@map("engine_run")
}

// ==================== Alerts ====================

model Alert {
  id             String   @id @default(uuid())
  portfolioId    String?

  priority       String   // "high" | "medium" | "low"
  type           String   // "drift_breach" | "concentration" | etc.
  title          String
  message        String
  data           Json?

  acknowledged   Boolean  @default(false)
  acknowledgedAt DateTime?

  createdAt      DateTime @default(now())

  @@index([portfolioId, acknowledged])
  @@index([priority, acknowledged])
  @@map("alert")
}

// ==================== ETF Scoring Results ====================

model EtfScoringResult {
  id             String   @id @default(uuid())
  runId          String
  instrumentId   String
  bucket         String   // "equity_global", etc.

  score          Float
  breakdown      Json     // {costs: X, tracking: Y, ...}
  redFlags       Json     // Array of strings

  dataAsof       DateTime @db.Date
  createdAt      DateTime @default(now())

  @@index([runId])
  @@index([bucket, score])
  @@map("etf_scoring_result")
}
```

7.2 Migrazioni TimescaleDB

Per `price_history` ottimizzato per time series:

```sql
-- migrations/001_enable_timescale.sql
CREATE EXTENSION IF NOT EXISTS timescaledb;

-- Converti price_history in hypertable
SELECT create_hypertable('price_history', 'date', if_not_exists => TRUE);

-- Crea index per query comuni
CREATE INDEX IF NOT EXISTS price_history_instrument_date_idx
  ON price_history (instrument_id, date DESC);

-- Retention policy: mantieni solo ultimi 5 anni
SELECT add_retention_policy('price_history', INTERVAL '5 years', if_not_exists => TRUE);
```

8. API Endpoints Specification

8.1 IPS Management

```
// POST /api/ips
interface CreateIpsRequest {
  userId: string;
  initialVersion: IpsConfig;
}

interface IpsConfig {
  version: string;
  profile: 'conservativo' | 'moderato' | 'aggressivo';
  phase: 'ETF-only' | 'ETF-stocks';
  horizon_years: number;
  pac_monthly_eur: {
    min: number;
    max: number;
    default: number;
  };
  targets: Array<{
    bucket: string;
    weight: number;
  }>;
  rebalance: {
    mode: 'contributi-only' | 'full';
    frequency: 'monthly' | 'quarterly';
    bands: {
      asset_class_abs: number;
      instrument_abs: number;
    };
  };
  constraints: {
    min_instruments: number;
    max_instruments: number;
    max_single_instrument_weight: number;
  };
}

// POST /api/ips/:id/versions
interface CreateIpsVersionRequest {
  config: IpsConfig;
}

// POST /api/ips/:id/activate/:versionId
interface ActivateIpsVersionResponse {
  success: boolean;
  activeVersion: IpsPolicyVersion;
}
```

8.2 Portfolio Management

```
// GET /api/portfolios/:id
interface GetPortfolioResponse {
  portfolio: Portfolio;
  positions: Array<{
    instrument: Instrument;
    quantity: number;
    avgCost: number;
    currentPrice: number;
    currentValue: number;
    totalReturn: number;
    totalReturnPct: number;
  }>;
  totalValue: number;
  totalInvested: number;
  totalReturn: number;
  totalReturnPct: number;
}

// POST /api/portfolios/:id/transactions
interface CreateTransactionRequest {
  instrumentId: string;
  side: 'BUY' | 'SELL';
  quantity: number;
  priceEur: number;
  feeEur?: number;
  executedAt?: string; // ISO date
  note?: string;
}

// GET /api/portfolios/:id/snapshots
interface GetSnapshotsResponse {
  snapshots: Array<{
    date: string;
    totalValue: number;
    items: Array<{
      instrumentId: string;
      instrumentName: string;
      quantity: number;
      price: number;
      value: number;
      weight: number;
      targetWeight: number;
      drift: number;
    }>;
  }>;
}
```

8.3 Engine Operations

```
// POST /api/engine/run
interface RunEngineRequest {
  type: 'ETF_SCORING' | 'PORTFOLIO_PROPOSAL' | 'MONTHLY_PAC';
  portfolioId?: string;
  params: Record<string, any>;
}

interface RunEngineResponse {
  runId: string;
  status: 'queued' | 'running';
  estimatedCompletionMs: number;
}

// GET /api/engine/runs/:runId
interface GetEngineRunResponse {
  runId: string;
  type: string;
  status: 'queued' | 'running' | 'completed' | 'failed';
  result?: any;
  error?: string;
  startedAt?: string;
  completedAt?: string;
  durationMs?: number;
}

// GET /api/proposals/:portfolioId/latest
interface GetLatestProposalResponse {
  proposal: Proposal;
  tradeList: Array<{
    instrument: Instrument;
    side: 'BUY' | 'SELL';
    amountEur: number;
  }>;
  rationale: {
    totalValuePreTrade: number;
    contribution: number;
    driftSnapshot: Array<{
      instrumentId: string;
      instrumentName: string;
      targetWeight: number;
      currentWeight: number;
      drift: number;
      driftPct: number;
    }>;
    constraintsPassed: boolean;
    violations?: Array<any>;
  };
}

// POST /api/proposals/:proposalId/execute
interface ExecuteProposalResponse {
  success: boolean;
  transactions: Transaction[];
  newPositions: Position[];
}
```

8.4 ETF Universe & Scoring

```
// POST /api/universe/instruments
interface AddInstrumentRequest {
  isin: string;
  name: string;
  ticker?: string;
  category: string;
  yahooTicker?: string; // For mapping
}

// POST /api/scoring/etf/:bucket
interface ScoreETFBucketRequest {
  bucket: string; // "equity_global", etc.
  maxCandidates?: number;
}

interface ScoreETFBucketResponse {
  runId: string;
  bucket: string;
  dataAsof: string;
  candidatesEvaluated: number;
  top3: Array<{
    rank: number;
    instrument: Instrument;
    score: number;
    breakdown: {
      costs: number;
      tracking: number;
      liquidity: number;
      robustness: number;
      preferences: number;
    };
    fundamentals: EtfMetrics;
    redFlags: string[];
  }>;
}
```

8.5 Alerts

```
// GET /api/alerts
interface GetAlertsRequest {
  portfolioId?: string;
  priority?: 'high' | 'medium' | 'low';
  acknowledged?: boolean;
}

interface GetAlertsResponse {
  alerts: Alert[];
  unacknowledgedCount: number;
}

// POST /api/alerts/:id/acknowledge
interface AcknowledgeAlertResponse {
  success: boolean;
  alert: Alert;
}
```

---

9. Testing Strategy

9.1 Livelli di Testing

1. Unit Tests - Algoritmi core (PAC, scoring, drift calculation) - Data providers (mock responses) - Utilities (hash, validation)

2. Integration Tests - API endpoints (E2E con DB test) - Worker job processing - Database queries

3. Snapshot Tests - Output scoring deterministico - Proposal rationale format

4. Manual Testing - UI workflows - Data quality alerts

9.2 Test Examples

```
// apps/api/src/pac/pac.service.spec.ts
import { describe, it, expect } from 'vitest';
import { PacService } from './pac.service';
```

```
describe('PacService', () => {
  describe('computeMonthlyPac', () => {
    it('should allocate contribution to most under-target instrument', () => {
      const positions = [
        { instrumentId: 'A', quantity: 10, currentPrice: 50 } // 500 EUR
      ];
      const targets = [
        { instrumentId: 'A', weight: 1.0 }
      ];
      const contribution = 200;

      const result = pacService.computeMonthlyPac(positions, targets, contribution,
        defaultBands);

      expect(result.tradeList).toHaveLength(1);
      expect(result.tradeList[0]).toMatchObject({
        instrumentId: 'A',
        side: 'BUY',
        amountEur: 200
      });
      expect(result.rationale.constraintsPassed).toBe(true);
    });

    it('should flag constraint violation if drift exceeds band', () => {
      // Test con drift > banda
      const positions = [
        { instrumentId: 'A', quantity: 10, currentPrice: 50 },
        { instrumentId: 'B', quantity: 5, currentPrice: 20 }
      ];
      const targets = [
        { instrumentId: 'A', weight: 0.90 },
        { instrumentId: 'B', weight: 0.10 }
      ];

      const result = pacService.computeMonthlyPac(positions, targets, 200, {
        asset_class_abs: 0.05, instrument_abs: 0.03 });

      // A = 500/600 = 0.833, target 0.90 → drift -0.067 → under
      // B = 100/600 = 0.167, target 0.10 → drift +0.067 → over
      // Dovrebbe comprare A
      expect(result.tradeList[0].instrumentId).toBe('A');

      // Ma se drift B > banda dovrebbe flaggare
      if (Math.abs(0.167 - 0.10) > 0.03) {
        expect(result.rationale.constraintsPassed).toBe(false);
      }
    });
  });
});

# services/engine/tests/test_scoring.py
import pytest
from src.scoring.etf_scorer import score_etf, ScoringWeights
from src.models.etf import ETF

def test_score_etf_low_ter_high_score():
    etf = ETF(
        isin="IE00B4L5Y983",
        name="Test ETF",
        ter=0.05,
        aum_eur=1_000_000_000,
        tracking_difference=0.05,
        avg_daily_volume=100_000,
        provider="iShares",
        age_years=5,
        replication_method="Physical",
        distribution_policy="Accumulating"
    )

    result = score_etf(etf)

    assert result['total'] > 80  # High score expected
    assert result['breakdown']['costs'] > 30  # Low TER → high cost score
    assert len(result['red_flags']) == 0

def test_score_etf_high_ter_red_flag():
    etf = ETF(
        isin="TEST",
        name="Expensive ETF",
        ter=0.75,  # Very high TER
        aum_eur=500_000_000,
        tracking_difference=0.30,
        avg_daily_volume=50_000,
        provider="Unknown",
        age_years=2,
        replication_method="Synthetic",
        distribution_policy="Distributing"
    )

    result = score_etf(etf)

    assert result['total'] < 50  # Low score
    assert any("TER elevato" in flag for flag in result['red_flags'])
    assert any("Replica sintetica" in flag for flag in result['red_flags'])

def test_scoring_deterministic():
    """Score deve essere deterministico dato stesso input"""
    etf = ETF(isin="TEST", ter=0.20, aum_eur=1e9, tracking_difference=0.10, ...)

    score1 = score_etf(etf)
    score2 = score_etf(etf)

    assert score1 == score2
```

---

10. Implementation Roadmap

Sprint 0: Bootstrap (2-3 giorni)

Obiettivo: ambiente funzionante end-to-end

- □ Setup monorepo (Turborepo o Nx)
- □ Docker Compose con tutti i servizi
- □ Postgres + Timescale + Redis containers
- □ Healthcheck API (NestJS)
- □ Healthcheck Engine (Python FastAPI)
- □ Healthcheck UI (Next.js placeholder)
- □ Schema DB base (Prisma migrate)

- □ CI/CD locale (pre-commit hooks, linting)

Deliverable: `docker compose up` → tutti i servizi green

Sprint 1: IPS + Paper Portfolio (1 settimana)

Obiettivo: gestione policy e portfolio paper funzionante

Backend: - [ ] IPS CRUD endpoints (create, get, list versions) - [ ] IPS versioning logic - [ ] IPS activation endpoint - [ ] Portfolio CRUD (create paper portfolio) - [ ] Transaction endpoints (add buy/sell) - [ ] Position calculation logic - [ ] Snapshot generation (mensile)

Frontend: - [ ] IPS Wizard (form multi-step) - [ ] IPS version list + attivazione - [ ] Portfolio dashboard (posizioni correnti) - [ ] Transaction form manuale - [ ] Snapshot history view

Testing: - [ ] Unit tests per IPS versioning - [ ] Integration tests per transaction flow - [ ] Snapshot determinism test

Deliverable: utente può creare IPS, portfolio paper, aggiungere transazioni manuali, vedere snapshot mensili

---

Sprint 2: Data Sourcing + Universe (1 settimana)

Obiettivo: importare ETF da fonti esterne e popolare universe

Backend: - [ ] ISIN mapping table + seed iniziale (top 20 ETF europei) - [ ] Yahoo Finance provider implementation - [ ] JustETF scraper (opzionale, fallback) - [ ] Composite provider con fallback logic - [ ] Instrument CRUD endpoints - [ ] ETF metrics update job (BullMQ) - [ ] Price history fetch job (giornaliero)

Frontend: - [ ] Instrument search/add form (by ISIN) - [ ] Instrument detail page (metrics, chart) - [ ] Universe explorer (lista ETF per bucket)

Testing: - [ ] Mock Yahoo Finance responses - [ ] Test ISIN mapping lookup - [ ] Test fallback mechanism

Deliverable: sistema può importare dati ETF da Yahoo Finance, salvare metriche e storico prezzi, visualizzare in UI

Sprint 3: ETF Scoring (1-2 settimane)

Obiettivo: scoring engine funzionante con output top 3 per bucket

Engine (Python): - [ ] Hard filters implementation - [ ] Scoring algorithm v1 (costs, tracking, liquidity, robustness, preferences) - [ ] Red flags identification - [ ] BullMQ consumer per job `ETF_SCORING` - [ ] Salvataggio risultati in `etf_scoring_result`

Backend: - [ ] Endpoint POST /api/scoring/etf/:bucket - [ ] Endpoint GET /api/scoring/results/:runId - [ ] Job enqueue logic

Frontend: - [ ] Scoring trigger page (seleziona bucket, lancia scoring) - [ ] Scoring results view (top 3 con breakdown) - [ ] Red flags display

Testing: - [ ] Scoring determinism tests - [ ] Hard filters edge cases - [ ] Mock ETF data con vari scenari (high TER, low AUM, etc.)

Deliverable: utente può lanciare scoring per bucket, vedere top 3 ETF candidati con motivazione numerica e red flags

---

Sprint 4: PAC Engine (1 settimana)

Obiettivo: algoritmo PAC mensile contributi-only

Engine (Python): - [ ] PAC algorithm implementation (contributi-only) - [ ] Drift calculation - [ ] Constraint checking - [ ] Extreme drift detection (vendite se necessario) - [ ] BullMQ consumer per job `MONTHLY_PAC`

Backend: - [ ] Endpoint POST /api/engine/pac/monthly - [ ] Proposal CRUD endpoints - [ ] Proposal execution endpoint (applica trade list → crea transactions)

Frontend: - [ ] PAC proposal view (trade list + rationale) - [ ] Drift visualization (gauge chart) - [ ] Execute proposal button - [ ] Proposal history

Testing: - [ ] PAC algorithm unit tests (vari scenari drift) - [ ] Constraint violation detection - [ ] Extreme drift sell trigger test

Deliverable: sistema genera proposta PAC mensile, mostra drift, permette esecuzione con creazione transactions

---

Sprint 5: Alerts & Monitoring (3-5 giorni)

Obiettivo: alert system funzionante

Backend: - [ ] Alert generation logic (drift, concentrazione, IPS change) - [ ] Alert endpoints (GET, acknowledge) - [ ] Alert cleanup job (rimuove acknowledged vecchi)

Frontend: - [ ] Alert badge in header (count unacknowledged) - [ ] Alert drawer/modal - [ ] Alert priority colors (high: red, medium: yellow, low: blue)

Testing: - [ ] Alert generation tests (drift > banda → alert HIGH) - [ ] Alert acknowledgment flow

Deliverable: sistema genera alert rilevanti, utente vede notifiche e può acknowledgeùare

---

Sprint 6: UI Polish & UX (1 settimana)

Obiettivo: dashboard professionale e usabile

- ☐ Dashboard home (summary portfolio, alert, next PAC)
- ☐ Performance chart (valore portfolio nel tempo)
- ☐ Mobile responsive
- ☐ Dark mode (opzionale)
- ☐ Loading states e skeleton screens
- ☐ Error boundaries
- ☐ Toast notifications

Fase 2: ETF + Azioni (anno 3/4+)

Quando portfolio ETF > 10k EUR e comfort con sistema:

- □ Security master azioni (ticker, exchange, sector)
- □ Fundamentals data provider (Financial Modeling Prep API o simile)
- □ Quality screener (ROIC, ROE, accruals, debt/equity, earnings stability)
- □ Sector allocation constraints
- □ Stock position sizing (max 2-3% per singola azione)
- □ Satellite allocation (10-20% azioni, 80-90% ETF core)

11. VPS Migration Plan

11.1 Quando Migrare

- Dopo Sprint 5 completato
- Quando sistema stabile in locale (> 1 mese uso)
- Quando necessario accesso remoto o automazione scheduling

11.2 VPS Requirements

Minimo: - 2 vCPU - 4 GB RAM - 40 GB SSD - Ubuntu 22.04 LTS

Provider suggeriti: - Hetzner Cloud (CX21: €5.83/mese) - DigitalOcean (Basic Droplet: $12/mese) - Linode (Nanode 4GB: $12/mese)

11.3 Migration Checklist

Preparazione: - [ ] Backup completo DB locale - [ ] Export docker images o rebuild su VPS - [ ] Secrets management (env file protetto o Vault)

Setup VPS: - [ ] SSH key auth (disable password) - [ ] Firewall (ufw): only 22, 80, 443 - [ ] Docker + Docker Compose install - [ ] Reverse proxy (Caddy con auto-HTTPS) - [ ] Domain setup (aurora.tuodominio.com)

Deploy: - [ ] Clone repo su VPS - [ ] .env.production con secrets - [ ] docker compose -f compose.prod.yml up -d - [ ] Restore DB backup - [ ] Verificare healthcheck

Hardening: - [ ] Auto-updates (unattended-upgrades) - [ ] Postgres backup giornaliero (pg_dump + S3/Backblaze) - [ ] Log rotation - [ ] Monitoring (opzionale: Grafana + Prometheus) - [ ] Fail2ban (protezione brute-force SSH)

Compose production:

```yaml
# infra/compose.prod.yml
version: '3.9'

services:
  db:
    image: timescale/timescaledb:latest-pg16
    environment:
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: aurora
    volumes:
      - db-data:/var/lib/postgresql/data
      - ./backups:/backups
    restart: unless-stopped
    networks:
      - aurora-net

  redis:
    image: redis:7-alpine
    restart: unless-stopped
    networks:
      - aurora-net

  api:
    build:
      context: .
      dockerfile: apps/api/Dockerfile
    environment:
      DATABASE_URL: postgresql://postgres:${DB_PASSWORD}@db:5432/aurora
      REDIS_URL: redis://redis:6379
      NODE_ENV: production
    depends_on:
      - db
      - redis
    restart: unless-stopped
    networks:
      - aurora-net

  engine:
    build:
      context: .
      dockerfile: services/engine/Dockerfile
    environment:
      DATABASE_URL: postgresql://postgres:${DB_PASSWORD}@db:5432/aurora
      REDIS_URL: redis://redis:6379
    depends_on:
      - db
      - redis
    restart: unless-stopped
    networks:
      - aurora-net

  ui:
    build:
      context: .
      dockerfile: apps/ui/Dockerfile
    environment:
      NEXT_PUBLIC_API_URL: https://aurora.tuodominio.com/api
    depends_on:
      - api
    restart: unless-stopped
    networks:
      - aurora-net

  caddy:
    image: caddy:2-alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./infra/Caddyfile:/etc/caddy/Caddyfile
      - caddy-data:/data
      - caddy-config:/config
    restart: unless-stopped
    networks:
```

```yaml
      - aurora-net

volumes:
  db-data:
  caddy-data:
  caddy-config:

networks:
  aurora-net:
    driver: bridge
```

Caddyfile:

```
aurora.tuodominio.com {
    reverse_proxy ui:3000

    handle /api/* {
        reverse_proxy api:3000
    }
}
```

---

12. Security Considerations

12.1 Uso Personale - Threat Model

Assunzioni: - Singolo utente (te) - No dati sensibili terze parti - No obbligo GDPR - Accesso da device fidati

Rischi principali: 1. Accesso non autorizzato a VPS 2. Esposizione credenziali (DB, API keys) 3. Data loss (DB corruption, VPS down)

12.2 Mitigazioni

Autenticazione: - Fase 1 (locale): nessuna auth necessaria - Fase VPS: basic auth (Caddy) o OAuth2 Proxy (Google) - API keys per accesso programmatico (se necessario)

Secrets: - `.env.local` in .gitignore - VPS: env file con chmod 600 - Rotate DB password periodicamente

Backup: - Automated daily DB dump - Retention 30 giorni - Off-site storage (Backblaze B2, AWS S3)

Monitoring: - Alert se servizi down (UptimeRobot free tier) - Disk space monitoring - Failed login attempts (Fail2ban)

---

13. Cost Breakdown

13.1 Fase Locale (Sviluppo)

- **Costi diretti**: €0
- **Tempo sviluppo**: ~4-6 settimane part-time

13.2 Fase VPS (Produzione)

Infrastruttura: - VPS (Hetzner CX21): €5.83/mese - Domain (.com): ~€12/anno (€1/mese) - Backup storage (Backblaze B2 25GB): €0.12/mese

Data providers: - Yahoo Finance: FREE - JustETF scraping: FREE (con limiti) - Opzionale EOD Historical: $19.99/mese (solo se necessario)

Totale mensile: - Minimo: ~€7/mese (VPS + domain + backup) - Con EOD: ~€27/mese

Totale annuale: €84-324/anno

14. Success Metrics

14.1 Fase 1 (ETF-only, primi 6 mesi)

Metriche tecniche: - [ ] Sistema disponibile 99%+ (locale o VPS) - [ ] PAC eseguito entro 1° giorno lavorativo del mese - [ ] Data quality > 95% (campi popolati ETF) - [ ] Zero transazioni manuali errate (rollback se necessario)

Metriche finanziarie: - [ ] Tracking error portfolio vs benchmark < 0.30% - [ ] Costi totali (TER + commissioni broker) < 0.40%/anno - [ ] Drift medio mensile < banda configurata

Metriche comportamentali: - [ ] Zero trading emotivo (solo PAC + ribilanciamento policy-driven) - [ ] 100% aderenza a IPS - [ ] Alert actionable ratio > 80% (alert utili vs rumore)

14.2 Fase 2 (ETF + Azioni, anno 3+)

- ☐ Quality stock portfolio outperformance vs MSCI World > 1%/anno (al netto costi)
- ☐ Max drawdown satellite < 1.5× core ETF
- ☐ Turnover azioni < 30%/anno

15. Known Limitations & Future Work

15.1 Limitazioni Fase 1

Data: - Tracking difference calcolato (se benchmark non disponibile) - Spread bid-ask non real-time (approssimato da volume) - Fundamentals update manuale/settimanale (non real-time)

Features: - No tax-loss harvesting (dipende da broker) - No dividend tracking dettagliato - No multi-currency (solo EUR)

Scalability: - Non testato per > 50 strumenti - No concurrent users (uso personale)

15.2 Future Enhancements

Short-term (anno 2): - [ ] Backtesting framework (test strategie storiche) - [ ] Scenario analysis (stress test portfolio) - [ ] Dividend calendar & reinvestment tracking - [ ] Mobile app (React Native o PWA)

Long-term (anno 3+): - [ ] Tax optimization suggestions (Italia-specific) - [ ] Broker API integration (auto-execute trades) - [ ] Machine learning per tracking difference prediction - [ ] Multi-portfolio support (aggressive + conservative)

16. References & Resources

16.1 Financial Theory

- **Modern Portfolio Theory**: Markowitz (1952)
- **Passive Investing**: Bogle “Common Sense on Mutual Funds”
- **Factor Investing**: Fama-French models
- **Quality Investing**: Piotroski F-Score, Greenblatt Magic Formula

16.2 Data Sources

- **ETF Screeners**: JustETF, ETF.com, Morningstar
- **Benchmarks**: MSCI, FTSE Russell
- **Research**: Vanguard Research, BlackRock Investment Institute

16.3 Technical Stack

- **NestJS**: https://nestjs.com
- **Next.js**: https://nextjs.org
- **Prisma**: https://prisma.io
- **BullMQ**: https://docs.bullmq.io
- **TimescaleDB**: https://docs.timescale.com
- **yfinance**: https://github.com/ranaroussi/yfinance

Appendix A: Sample ETF Universe (Starter Kit)

Azionario Globale (equity_global): - IE00B4L5Y983 - iShares Core MSCI World UCITS ETF USD (Acc) - IWDA - IE00BK5BQT80 - Vanguard FTSE All-World UCITS ETF USD (Acc) - VWCE - LU1781541179 - Amundi MSCI World UCITS ETF EUR (C) - EUNL

Azionario USA (equity_us): - IE00B5BMR087 - iShares Core S&P 500 UCITS ETF USD (Acc) - CSPX - IE00BFM15T99 - Vanguard S&P 500 UCITS ETF USD (Acc) - VUAA

Obbligazionario Globale Hedged (bond_global_hedged): - IE00BG47KH54 - Vanguard Global Aggregate Bond UCITS ETF EUR Hedged (Acc) - VAGF - LU1407890620 - Amundi Global Aggregate Bond UCITS ETF EUR Hedged (C) - AGGH

Monetario EUR (money_market_eur): - LU0290358497 - Xtrackers II EUR Overnight Rate Swap UCITS ETF (C) - XEON

Appendix B: ISIN Mapping Initial Seed

```sql
-- Seed per top ETF europei
INSERT INTO isin_mapping (isin, yahoo_ticker, exchange, verified) VALUES
('IE00B4L5Y983', 'IWDA.AS', 'AMS', true),
('IE00BK5BQT80', 'VWCE.DE', 'XETRA', true),
('LU1781541179', 'EUNL.DE', 'XETRA', true),
('IE00B5BMR087', 'CSPX.L', 'LSE', true),
('IE00BFM15T99', 'VUAA.L', 'LSE', true),
('IE00BG47KH54', 'VAGF.L', 'LSE', true),
('LU0290358497', 'XEON.DE', 'XETRA', true);
```

Conclusioni

AURORA è progettato per essere:

1. **Semplice**: inizia con 1 ETF, cresce con te
2. **Trasparente**: ogni decisione è tracciabile e motivata
3. **Robusto**: policy formale previene errori emotivi
4. **Scalabile**: architettura pronta per ETF → ETF+Azioni
5. **Personale**: nessuna distribuzione, zero compromessi UX enterprise

Next Steps: 1. Review questo documento 2. Setup repository monorepo 3. Iniziare Sprint 0 (bootstrap)

Questo codex è versionato e vivo: aggiorna quando necessario, conserva vecchie versioni per reference.

Fine documento - AURORA Project Codex v2.0

Generato il 02/01/2026 - Uso esclusivamente personale