Prompt to generate an identical Coinbase handler

Role: You are an expert Python engineer. Generate production-ready code.

Goal: Create an **async Coinbase Advanced Trade API v3 client** and a small connector wrapper that mirror the following architecture and behaviors:

Files to output

1. `coinbase_api.py`
2. `coinbase_connector.py`

High-level design

- Language: Python 3.11
- Async HTTP: `aiohttp` with a single shared `ClientSession` and `ClientTimeout(total=Config.REQUEST_TIMEOUT)`
- Rate limiting: `ratelimit` decorators — `@sleep_and_retry` + `@limits(calls=100, period=10)`
- Logging: `structlog.get_logger("CoinbaseAPI")` / `structlog.get_logger("CoinbaseConnector")`
- Config: read from a `config.py` module exposing:
  - `class ApiKeys:` `COINBASE_API_KEY`, `COINBASE_PRIVATE_KEY` (PEM string), `COINBASE_ORG_ID`
  - `class Config:` flags & tunables used below (e.g., `REQUEST_TIMEOUT`, `RETRY_ATTEMPTS`, `USE_LIVE_MODE`)
- Symbol mapping utility: reuse a function `map_symbol_for_exchange(symbol, "coinbase")` if present; otherwise implement internal mapping (see below).

Authentication (Coinbase Advanced / CDP)

- Use **JWT (ES256)** with **ECDSA** signing.
- JWT headers: `kid` must be `organizations/{COINBASE_ORG_ID}/apiKeys/{COINBASE_API_KEY}`, and include a random `nonce`.
- JWT payload:
  - `iss: "cdp"`
  - `sub: "organizations/{COINBASE_ORG_ID}/apiKeys/{COINBASE_API_KEY}"`
  - `nbf: now`
  - `exp: now + 120`
  - `uri: "{METHOD} api.coinbase.com{endpoint}"`
- Send as `Authorization: Bearer <jwt>`.
- Validate presence of all three creds; log an error and disable if missing.

Endpoints (constants)

- `COINBASE_BASE = "https://api.coinbase.com"`
- `COINBASE_ACCOUNTS = "/api/v3/brokerage/accounts"`
- `COINBASE_ORDER = "/api/v3/brokerage/orders"`
- `COINBASE_PRICE = "/api/v3/brokerage/products/{}/ticker"` (format with product_id)
- `COINBASE_DEPTH = "/api/v3/brokerage/market_data/book"`
- `COINBASE_KLINES = "/api/v3/brokerage/products/{product_id}/candles"`

Class: `CoinbaseAPI`

Constructor

- Set `self.base_url`, load API key/private key from `ApiKeys`, parse PEM (no logging of secrets).

- Flags: `self.coinbase_enabled: bool` (True when creds are valid), `self.initialized: bool`, caches: `self.valid_symbols: set[str]`, `self.product_info: dict`.

Public async methods (exact names):

- `initialize(self)`
  - Build `self.valid_symbols` and `self.product_info` cache.
  - If product discovery fails, **fallback** with a small dict of safe defaults (e.g., BTC-USD, ETH-USD, DOGE-USD, ADA-USD with reasonable `min_notional`, `base_increment`, etc.). Log the fallback once.
- `_make_request(self, method, endpoint, params=None, data=None, retry=Config.RETRY_ATTEMPTS)`
  - Compose URL, generate JWT (`_generate_jwt` helper), set headers.
  - `aiohttp` request with retries on 429/5xx, exponential backoff jitter.
  - Return parsed JSON or `None` with a clear log line on failure.
- `get_all_tickers(self) -> list | dict`
  - Fetch a small universe of actively tradable products (or use the fallback set).
  - Return normalized list/dict.
- `get_price(self, symbol: str) -> float | None`
  - Map internal symbol to Coinbase product id (see mapping below).
  - GET ticker; return `float(last_trade_price)`; in sim fallback, optionally return a mocked price if API unavailable.
- `get_order_book(self, symbol: str, level: int = 1) -> dict | None`
  - Use `COINBASE_DEPTH`, pass product_id and book level.
- `get_klines(self, symbol: str, interval: str) -> list`

    - Map intervals: `1m, 5m, 15m, 1h, 4h, 1d → ONE_MINUTE, FIVE_MINUTE, FIFTEEN_MINUTE, ONE_HOUR, SIX_HOUR, ONE_DAY`.

- `get_account(self)` / `get_cash_balance(self) -> float` / `get_total_cash_balance(self) -> float` / `get_total_portfolio_value(self) -> float`
    - Use brokerage accounts; sum USD cash and asset values when available; guard with `self.coinbase_enabled`.
- `place_order(self, symbol, side, qty, price=None, order_type="limit", post_only=False, time_in_force="GTC") -> dict | None`
    - Map symbol, round size to step increments (see below).
    - For **limit**: set limit price, `time_in_force` (accept `"GTC"`, `"IOC"`, `"FOK"`, and `"GTX"` if post-only), enforce `post_only` when requested.
    - For **market**: send notional/size accordingly.
    - Return parsed order response or `None` with reason logged.
- `get_exchange_info(self)` / `get_historical_trades(self)` (light wrappers; may return `None` if not needed).
- `get_min_notional(self, symbol) -> float`
    - From `self.product_info` if available; otherwise fallback (e.g., `$12`).
- `round_to_lot_size(self, symbol, qty, price) -> float`
    - Use Decimal with `ROUND_DOWN` against `base_increment`. Return float.
- `_get_available_usd_balance(self) -> float`
    - Internal helper to pick the available USD cash from accounts.
- `close(self)`
    - Close any internal `aiohttp` session if owned.

Helpers inside `CoinbaseAPI`:

- `_generate_jwt(self, method, endpoint) -> str | None` (as described in Auth above).
- `_map_to_coinbase_symbol(self, internal_symbol: str) -> str | None`
    - Filter unsupported symbols (e.g., `CLV*`).
    - Convert internal `"BTCUSDT"` → `"BTC-USD"` (replace `USDT` with `-USD`).
    - If `self.product_info` is populated, validate; otherwise return the transformed id.
    - Return `None` if not supported.
- `get_product_info(self, coinbase_symbol: str) -> dict`
    - Pull cached increments, min sizes, min notional, quote currency; fallback to safe defaults with a warning.

Class: `CoinbaseConnector`

- `__init__`: `self.api = None`, `self.connected = False`
- `connect(self)` async:
    - Instantiate `CoinbaseAPI`, call `get_all_tickers()` to validate connectivity.
    - On success: `self.connected = True`, return `self.api`, log a ✅ line.
    - On failure: log ❌ and return `None`.

Behavior & guards

- If `coinbase_enabled` is False (missing creds), all API methods should **short-circuit** with a clear INFO/WARN and return `None`.
- In **simulation** (`Config.USE_LIVE_MODE == False`), avoid spamming Coinbase: don't run background Coinbase tasks; allow **optional one-time** balance sync only when a separate flag `USE_LIVE_BALANCE_IN_SIM` is True.

- Robust error handling: catch `aiohttp.ClientError` and `asyncio.TimeoutError`, log succinctly, return `None`.
- Never log secrets; log minimal object counts or symbols instead.

Symbol & precision

- Use **exchange increments** for `round_to_lot_size` (`Decimal.quantize(..., ROUND_DOWN)`).
- Store & use `quote_increment`, `base_increment`, `base_min_size`, `min_notional`.
- Always return notional in **USD** for Coinbase.

Acceptance checklist (must pass)

- `CoinbaseAPI` exposes all methods named above.
- JWT uses **ES256**, correct `kid`, payload, and `Authorization: Bearer`.
- `place_order` supports **post-only** w/ `time_in_force="GTX"` when requested.
- `round_to_lot_size` rounds **down** to increments.
- Mapping `"DOGEUSDT" → "DOGE-USD"` and `"SHIBUSDT" → "SHIB-USD"` works; `CLV*` is filtered.
- On init failure, fallback product cache is used and logged once.
- All network calls are rate-limited and retried on 429/5xx.
- Methods cleanly return `None` on failure (with a single, actionable log line).

If you paste that into me, I'll generate code that mirrors your handler's architecture and behavior—down to JWT signing, endpoints, symbol mapping, rounding, rate limits, and the connector's connectivity probe.