Bot Service - Project Overview & Run Instructions

1) Purpose

This repository implements a decision microservice + supporting infrastructure for automated auction bidding (bot).

It keeps decision logic (persona selection, fuzzy/hybrid rules) separate from runtime workers (publisher + player).

Admins can inspect bot assignments and replay decisions. Use for UAT with admin-visible bots; production must disclose bots to users.

2) High-level architecture / flow

- Auction API (external): provides live auction data and accepts bid submissions.
- AucApi.py: HTTP wrapper to call external auction API endpoints (get snapshots, submit bids).
- AuctionInfoDistributor.py: ZMQ PUB process. Polls AucApi.get_all_live_auction_data() and publishes JSON snapshots on TCP port (default 9505).
- AuctionPlayer.py: ZMQ SUB process. Subscribes to distributor stream, for each auction:
    * Ensures a bot is assigned (state_store.assign_bot)
    * Builds a DecideRequest-like object
    * Calls app.services.decision_service.DecisionService.decide(req, state)
    * Sleeps for delay_seconds returned by decision, then calls AucApi.submit_bid(...)
    * Persists updated state via state_store.persist_bot_state
- app/ (FastAPI decision microservice):
    * app.main: wires routers
    * app.api.decide: /decide endpoint for programmatic decisions (used in tests or admin replay)
    * app.api.admin: admin endpoints to list assigned bots and replay decisions (protect these in prod)

* app.services.decision_service: orchestrates persona selection, fuzzy category, base heuristics, persona adjustments

* chooser.py, fuzzy_logic.py, hybrid_core.py, persona.py: modular components used by DecisionService

- state_store.py: in-memory store for assignment and per-bot state (UAT). Replace with state_store_redis.py or DbQueries for production.

3) File layout (root)

- app/
  - api/decide.py, api/admin.py
  - services/decision_service.py
  - chooser.py, fuzzy_logic.py, hybrid_core.py, persona.py, schemas.py, utils.py
- AucApi.py
- AuctionInfoDistributor.py
- AuctionPlayer.py
- state_store.py
- config.py
- Dockerfile
- docker-compose.yml
- requirements.txt
- .env (local, DO NOT commit)
- .env.example (commit)

4) How the decision is made (summary)

- Persona selection: deterministic weighted scoring using WEIGHT_MATRIX in config.py based on features (phase_ratio, time_left_pct, recent_bid_rate, last_bid_secs, leader_dominance, volatility).

- Fuzzy category: fuzzy_logic computes category ("low","medium","high") and a score to nudge amount.
- Hybrid core: computes base_amount and base_delay heuristics using best_price, threshold_price, min_inc.
- Persona adjustments: persona rules shift amount and delay (patient waits longer, snipe bids late, aggressive bids higher/faster).
- DecisionService combines these, quantizes to min_inc_price, applies max/inc/threshold safety, randomizes slightly, returns:

```
{status, persona, bid_amount, delay_seconds, metadata, updated_state}
```

5) Run locally (no Docker)

- Create venv and install deps:

```
python3 -m venv .venv
source .venv/bin/activate   # Windows: .venv\Scripts\activate
pip install --upgrade pip
pip install -r requirements.txt
```

- Create .env from .env.example and fill values (project root).

- Start decision API:

```
uvicorn app.main:app --reload --port 8080
```

- Start distributor (in another terminal):

```
python AuctionInfoDistributor.py
```

- Start player (another terminal):

```
python AuctionPlayer.py
```

6) Run with Docker Compose (recommended for UAT)

- Ensure .env is in project root, then:

```
docker-compose up --build
```

- Services:

   decision (FastAPI), distributor (publisher), player (subscriber), optional redis

7) Admin & audit

- Admin endpoints (app/api/admin.py) let you inspect assigned bots and bot_state.
- DecisionService attaches metadata (is_automated, agent, persona, decision_inputs_hash) - persist these to DB for audit.
- For production, replace state_store with Redis/DB and record every decision in a 'bot_decisions' table with request/response JSON.

8) Notes & next steps

- Replace in-memory store with Redis for multi-process coordination in production.
- Replace time.sleep scheduling with APScheduler or job queue to avoid blocking player loop.
- Add authentication for admin endpoints and secure API credentials.
- Add unit tests for chooser, persona, fuzzy_logic and integration tests simulating snapshots.

9) Quick troubleshooting

- Missing imports: pip install -r requirements.txt (pyzmq, requests, python-dotenv)
- If pyzmq fails in Docker build, use Dockerfile variant that installs libzmq3-dev and build-essential.