PREDICTION MARKET SYSTEMS US

Student Course Manual

Contracts, forecasting, fees, paper execution and risk governance

Version 3.2 - Corrected US Validation Candidate

Paper trading only

No live orders, signals, profit promise or accredited qualification. Confirm current eligibility and platform rules. All exercises use public data and local simulation.

Contents

01 Course scope and release status

02 M00 operating standard

03 M01 contract literacy

04 M02 resolution engineering

05 M03 api reliability

06 M04 evidence integrity

07 M05 forecasting calibration

08 M06 microstructure liquidity

09 M07 cross market equivalence

10 M08 model building

11 M09 risk policy

12 M10 backtest integrity

13 M11 paper operations

14 M12 production assurance

15 M13 capstone

16 Worked cases and solutions

17 Assessment and certification rubric

18 US source and compliance register

19 US platform and market playbook

Module 0 - The operating standard

> Operating boundary - public data and paper simulation only. This course does not authenticate an account, request API or signing credentials, hold funds or place live orders. For the US edition, every exercise remains read-only and paper-only. The course does not teach or support VPNs, proxies, remote execution, travel workarounds or any other attempt to evade an eligibility or access restriction. Platform availability and law can change; the source and compliance register must be checked before each cohort.

Why this module comes first

A technically impressive result is worthless when nobody can establish what data produced it, when the decision was made, or whether the result came from live data, sample data or hindsight. Module 0 establishes the operating discipline used throughout the course. You will install the supplied application, prove that its tests and health endpoint run in your environment, identify its data mode, and produce a reproducible verification record.

The application is a local, quote-based paper trader. It reads public Polymarket US gateway snapshots when the network request succeeds and otherwise loads clearly labelled sample markets. It stores markets, best quotes, paper positions, cash movements and equity snapshots in SQLite. It models price-dependent fees, adverse slippage and two exposure limits. It does not reconstruct full order-book quantities, guarantee a fill, place a transaction or prove that a strategy is profitable. Those boundaries are part of the product, not defects to conceal.

Learning outcomes

By the end of this module, you can:

1. separate observations, assumptions, inferences and unknowns;
2. distinguish application health from data freshness and strategy validity;
3. install and verify the raw-source application without using credentials;
4. identify `live`, `sample`, `mixed` or `empty` market-data status;
5. preserve the commands, versions, timestamps and outputs needed to reproduce a result;
6. explain the US eligibility and market-integrity boundary; and
7. submit an installation evidence pack that another learner can audit.

1. Four records that must never be merged

Every later market decision has four logically separate records.

1. Contract record: the exact proposition, identifier, resolution rules, source, deadline and unresolved ambiguity.
2. Market record: the observed best bid/ask and derived long/short entry references, source label, source timestamp, local sync time and known limits of that observation.
3. Forecast record: your probability estimate, uncertainty range, evidence set and time frozen.
4. Decision record: the side, simulated size, cost assumptions, risk checks, action and reason.

Keeping them separate prevents a market price becoming your forecast, later information entering an earlier decision, or a favourable outcome rewriting poor process.

Use these evidence labels in notes and submissions:

- OBSERVATION: directly present in a preserved source or application output. Example: “The health endpoint reported `data_mode: sample`.”
- ASSUMPTION: deliberately chosen for modelling. Example: “The simulation uses 25 basis points of base slippage before adverse tick alignment.”
- INFERENCE: a conclusion drawn from observations and assumptions. Example: “The apparent edge is too small to survive our uncertainty allowance.”
- UNKNOWN: material information that is absent or unverified. Example: “Executable depth is unknown because this application does not collect full order-book depth.”

Do not upgrade an inference to an observation merely because it sounds plausible. Do not fill an unknown with a convenient number. An explicit unknown is professional evidence; an invented value is an integrity failure.

2. Jurisdiction, safety and ethics boundary

US learners must confirm current eligibility; all course exercises use public data and local simulation. The controlling source register records the current sources checked for this release. Recheck them before release; this lesson is scope control, not personal legal advice.

Within this course you must not:

- sign in to or fund an account;
- create, request, store or paste an account password, API key, private key, authentication token or trading credential;
- place, cancel or close a live order;
- use a referral or inducement link;
- advise another person what to buy or sell;
- present simulated returns as live performance; or
- bypass or help someone bypass an eligibility, access, platform or legal restriction.

Public technical access does not establish legal permission to transact. A successful HTTP response proves only that a server returned data. If an exercise appears to require live execution, stop: either the instruction has been misunderstood or the material must be escalated to the instructor.

3. What the verification does-and does not-prove

The test suite checks defined behaviours including parsing, accounting, risk-limit rejection, duplicate-close protection, concurrency and backtest bookkeeping. A pass means those tested behaviours matched expectations in that version and environment.

Passing tests do not establish that:

- the public API is currently reachable;
- a quote is executable;
- the fee coefficient or slippage setting matches a real market;
- the resolution wording is understood;
- a forecast has skill;
- a backtest is free of every possible bias; or
- paper performance will repeat with money at risk.

Likewise, `/api/health` shows that the local service and database respond. Its `data_mode` and `last_synced_at` fields help describe the loaded market snapshot. They do not certify the economic truth of the data or authorise trading.

4. Install and verify the application

Prerequisites

Use a local computer with Python 3.11 or later, a terminal, permission to write to a working folder, and a browser. No account, API credential or payment method is required. Work from the extracted `polymarket-us-paper-trader` directory.

Record the first two commands and their outputs:

```
python --version
python -c "import platform; print(platform.platform())"
```

If your system uses `python3`, use it consistently in every later command. Do not alternate interpreters without recording the change.

Configuration truth

Direct Python reads operating-system environment variables; it does not automatically load `.env`. The example documents supported names and Docker Compose can use `.env`, but copying it alone does not change a direct run.

Defaults are adequate for this lab. To isolate your evidence database on macOS or Linux, you may prefix a command:

```
PAPER_DB_PATH=data/module0_verification.db python -m polymarket_app.cli init
```

In PowerShell, set the variable for the terminal session first:

```
$env:PAPER_DB_PATH = "data/module0_verification.db"
python -m polymarket_app.cli init
```

Use a distinct path so the run is repeatable and earlier evidence remains preserved.

Run the regression suite

From the application root:

```
python -m unittest discover -s tests -v
```

Preserve the complete output, including the number of tests, failures or errors, elapsed time and final status. Do not submit only the final `OK` line. If any test fails, the verification gate remains failed until the cause and remediation are documented.

Initialise and sync

```
python -m polymarket_app.cli init
python -m polymarket_app.cli sync
python -m polymarket_app.cli portfolio
python -m polymarket_app.cli markets
```

Sync reports its source: `live Polymarket US gateway` or `offline sample` with an exception class. Sample mode is valid for learning, but every derived record must remain labelled synthetic.

The initial portfolio should show configured cash and no positions. Market output should include identifier, question, prices, source and timestamps. Record missing or inconsistent fields as incidents.

Start and inspect the local service

```
python -m polymarket_app.server
```

Open `http://127.0.0.1:8080` in a browser. The interface should identify paper mode and the active market-data mode. Do not expose the development server to the public internet. The default bind address is local-only.

In a second terminal, capture the health response:

```
curl http://127.0.0.1:8080/api/health
```

PowerShell users may use:

```
Invoke-RestMethod http://127.0.0.1:8080/api/health
```

A valid response includes application status, paper mode, version, data mode, source counts and last sync time. Compare the API response with the dashboard label. A disagreement is an incident; do not proceed as though one of them is automatically correct.

Stop the server with `Ctrl+C` after capturing the evidence. Module 0 does not require opening a paper position.

5. Reproducibility standard

Your verification record must let a second person reproduce the run and contain:

- application/package version or supplied archive name;
- operating system and Python version;
- exact working directory and database path, excluding personal secrets;
- configuration values that differ from defaults;
- commands in execution order;
- unedited stdout and error output;
- local timestamp with time zone and, where available, UTC timestamp;
- test count and result;
- sync source and data mode;
- health response;
- any deviation, retry or manual intervention; and
- a conclusion limited to what the evidence proves.

Use sortable names such as `M00_2026-09-25T133000Z_test-output.txt`. Preserve failed and passing runs and explain the change. You may redact usernames, but not results, source mode or timestamps.

Lab 0 - Installation and evidence gate

Task

Complete the installation workflow above using a fresh database path. Produce one compact verification pack containing:

1. environment record;
2. complete automated-test output;
3. initialisation and sync output;
4. initial portfolio output;
5. health-endpoint response;
6. one dashboard screenshot showing paper mode and data mode;
7. an evidence classification table with at least two observations, two assumptions, one inference and two unknowns; and
8. a 150-250 word verification conclusion.

Your conclusion must state whether the installation gate passed, whether the run used live or sample data, what passing tests demonstrate, and at least three things they do not demonstrate.

Pass conditions

- Python 3.11 or later is evidenced.
- All supplied tests pass, or a failed gate is honestly reported with reproducible diagnostics.
- The database initialises and portfolio output reconciles to its untouched starting state.
- Dashboard and health response agree on paper mode and data mode.
- No credential, account action, live order or circumvention technique is used.
- Claims remain inside the evidence captured.

Common failure modes

1. Running from the wrong directory: `ModuleNotFoundError` usually means the package root is not the current directory.
2. Mistaking `.env.example` for active configuration: direct Python reads exported environment variables, not that file automatically.
3. Calling sample data live: the source label controls the claim, not the realism of the question.
4. Treating health as freshness: a responsive service can contain old or sample data.
5. Showing only a passing screenshot: it omits commands, versions and prior failures needed for reproduction.
6. Changing multiple variables at once: the successful cause becomes unknowable.
7. Using test success as profitability evidence: software correctness and forecasting edge are different claims.
8. Exposing the local server: keep the default local bind unless a separately reviewed deployment plan exists.

Submission checklist

- [ ] Environment and Python version recorded
- [ ] Fresh database path recorded
- [ ] Full test output preserved
- [ ] Sync source explicitly labelled
- [ ] Portfolio starting state captured
- [ ] Health JSON captured
- [ ] Dashboard paper/data mode visible
- [ ] Observations separated from assumptions and inferences
- [ ] Unknowns retained rather than invented
- [ ] No secrets or personal identifiers included
- [ ] US eligibility boundary acknowledged
- [ ] Conclusion makes no performance claim

Knowledge check

Choose one answer for each question.

1. A passing test suite proves that:

A. the strategy is profitable; B. the tested behaviours matched expected results in that environment; C. US gateway quotes were executable; D. the market will resolve correctly.