MLB Unicorn Engine — Full Project Specification (Master Document)

This is the **complete, consolidated, Codex-ready specification** for the entire MLB Unicorn Engine project. It defines all rules, data models, scoring logic, ETL pipeline, APIs, and frontend behavior required to implement a working system that can be backtested on the full **2025 MLB season**.

This document is the **single source of truth**. Code must follow it unless explicitly superseded by a later version.

---

0. High-Level Goals

1. Ingest MLB Statcast-style pitch/PA data for the **2025 season**.
2. Compute a curated set of **fan-friendly unicorn patterns** (no ultra-nerd noise).
3. Score unicorns using a **balanced scoring model** that favors:
4. statistical rarity (primary)
5. pattern importance (holy-grail metrics)
6. public appeal (general > nerd)
7. light market influence (attendance + media)
8. Produce a **Top 50 unicorns per day** list with **unique players**.
9. Keep **14 days of unicorn history** easily queryable.
10. Serve unicorns via a **backend API** and render them on a **frontend website**.
11. Backtest the full 2025 season to tune:
12. which patterns are worth keeping
13. weightings
14. which classes of unicorns to drop entirely.

Implementation should be done so that **pattern definitions and weights are data-driven**, not hardcoded, enabling future expansion.

---

1. Global Constraints & Philosophy

These apply to the entire project.

1.1 Counts — Hard Global Rule

Only the following pitch counts may ever appear in unicorn definitions: - `3-0` - `0-2` - `3-2`

Any pattern that references a count outside this set is **invalid** and must: - Either be rejected at pattern definition time, or - Be ignored by the evaluation pipeline.

1.2 Maximum Pattern Complexity

Each unicorn pattern may include **at most 4 logical conditions**. Conditions include: - EV threshold (e.g., `ev >= 100`, `is_hard_hit`, `is_barrel`) - Direction (pulled, opposite-field, center) - Pitch type (fastball, slider, curve, etc.) - Pitch location (high/low, inside/outside, quadrant-like buckets) - Count (3-0, 0-2, 3-2 only) - Broad context flags (e.g., day/night OR home/away) — ONLY if not layered with other context.

> **Rule:** Condition count = number of independent boolean filters. If a pattern would require more than 4, it is **not allowed**.

1.3 Banned Concepts (Never Allowed in Patterns)

Patterns **must not** use: - Pitch sequence logic ("after fouling off", "after two sliders", etc.). - Inning-based conditions ("in the 7th inning", "late innings", etc.). - Weather conditions (temperature, wind, humidity, etc.). - Complex environment hybrids: environment + pitch type + location + direction. - Home/road stacked together with pitch-type & location (home/road can be used sparingly at PA/summary level, not deeply embedded in pitch-level multi-condition patterns). - Any count besides 3-0, 0-2, 3-2.

If in doubt, **err on the side of disallowing** rather than including.

1.4 Allowed Pattern Categories (High-Level)

We conceptually divided unicorns into categories A–L earlier. For implementation, only the following **classes of patterns** are in-scope for v1:

- **Hitter contact/quality patterns (last 50 PA)**
- barrels, hard-hit balls, xwOBA, direction, launch-angle windows.
- **Pitcher contact suppression / whiff patterns (last 3 starts or last 5 relief apps)**
- whiff%, hard-hit% allowed, xwOBA allowed.
- **Count-specific patterns using ONLY 3-0, 0-2, 3-2**.
- **Simple day/night and home/road splits (no extra stacking)**.
- **Park-specific performance (per hitter or pitcher)**.
- **Fatigue-based patterns (pitch count thresholds) that don't violate complexity rule**.

Deep hybrid Category L type patterns are **only allowed if they comply with the 4-condition limit and all bans above**. If they can't be simplified, they are omitted.

---

2. Core Metrics (Holy-Grail Stat Sets)

2.1 Hitters – Evaluated over last 50 PA

These are the "holy grail" predictive metrics shown on hitter pages: - **Barrel %** = barrels / batted balls - **Hard-Hit %** = hard-hit (EV $\geq$95) / batted balls - **xwOBA** over last 50 PA - **Contact %** = swings that put ball in play / swings - **Chase %** = swings at pitches outside the zone / total pitches outside the zone

2.2 Starting Pitchers – Last 3 Starts

- **xwOBA allowed**
- **Whiff %** = whiffs / swings
- **K %** = strikeouts / batters faced
- **BB %** = walks / batters faced
- **Hard-Hit % allowed**

2.3 Relievers – Last 5 Appearances

- **xwOBA allowed**
- **Whiff %**
- **K %**
- **BB %**
- **Hard-Hit % allowed**

These 5 metrics per role must be computed nightly and stored in a **player_summary**-like table to power player pages and some unicorns.

---

3. Data Model

Assume PostgreSQL as the primary data store.

3.1 `players`

```
CREATE TABLE players (
  player_id      BIGINT PRIMARY KEY,
  mlb_id         BIGINT UNIQUE,
  full_name      TEXT NOT NULL,
  bat_side       CHAR(1),     -- 'L','R','S'
  throw_side     CHAR(1),
  primary_pos    TEXT,
  current_team_id INT REFERENCES teams(team_id)
);
```

3.2 `teams`

```
CREATE TABLE teams (
  team_id   INT PRIMARY KEY,
  team_name TEXT NOT NULL,
  abbrev    TEXT NOT NULL
);
```

3.3 games

```
CREATE TABLE games (
  game_id       BIGINT PRIMARY KEY,
  game_date     DATE NOT NULL,
  home_team_id  INT REFERENCES teams(team_id),
  away_team_id  INT REFERENCES teams(team_id),
  venue_id      INT,
  is_day_game   BOOLEAN,
  is_night_game BOOLEAN,
  -- optional: temperature, roof_state, etc., but not used in v1 patterns
  UNIQUE(game_date, home_team_id, away_team_id)
);
```

3.4 pitch_facts

One row per pitch. Denormalized for unicorn computation.

```
CREATE TABLE pitch_facts (
  pitch_id          BIGINT PRIMARY KEY,
  game_id           BIGINT REFERENCES games(game_id),
  pa_id             BIGINT,
  inning            INT,
  top_bottom        CHAR(1), -- 'T' or 'B'
  batter_id         BIGINT REFERENCES players(player_id),
  pitcher_id        BIGINT REFERENCES players(player_id),
  pitch_number_pa   INT,
  pitch_number_game INT,

  pitch_type        TEXT,            -- 'FF','SL','CU','CH','SI','FC', etc.
  vel               NUMERIC(5,2),
  spin_rate         NUMERIC(7,2),

  count_balls_before  INT,
  count_strikes_before INT,

  is_in_zone        BOOLEAN,
  result_pitch      TEXT,            -- called_strike, swinging_strike, ball,
foul, in_play, etc.
  is_last_pitch_of_pa BOOLEAN,

  -- Batted ball info if in play
  launch_speed      NUMERIC(5,2),    -- EV
  launch_angle      NUMERIC(5,2),
  spray_angle       NUMERIC(5,2),
```

```
  is_barrel         BOOLEAN,
  is_hard_hit       BOOLEAN,
  batted_ball_type  TEXT,            -- GB, FB, LD, PU
  hit_direction     TEXT,            -- 'pull','oppo','center'

  -- Derived buckets
  loc_high_mid_low  TEXT,            -- 'high','mid','low'
  loc_in_mid_out    TEXT,            -- 'in','mid','out'
  loc_region        TEXT,            -- e.g., 'high_in','low_away'

  -- Outcome from PA (duplicated for convenience)
  pa_outcome        TEXT,            -- single, double, HR, K, BB, etc.
  is_hr             BOOLEAN,
  is_hit            BOOLEAN,
  is_walk           BOOLEAN,

  -- Count string for convenience
  count_str         TEXT             -- e.g., '3-2' representing end-of-count
when pitch is thrown
);

CREATE INDEX idx_pitch_facts_batter ON pitch_facts (batter_id);
CREATE INDEX idx_pitch_facts_pitcher ON pitch_facts (pitcher_id);
CREATE INDEX idx_pitch_facts_game_date ON pitch_facts (game_id);
CREATE INDEX idx_pitch_facts_count_str ON pitch_facts (count_str);
```

3.5 `pa_facts` (Plate Appearances)

```
CREATE TABLE pa_facts (
  pa_id             BIGINT PRIMARY KEY,
  game_id           BIGINT REFERENCES games(game_id),
  inning            INT,
  top_bottom        CHAR(1),
  batter_id         BIGINT REFERENCES players(player_id),
  pitcher_id        BIGINT REFERENCES players(player_id),

  result            TEXT,           -- single,double,HR,K,BB,etc.
  is_hit            BOOLEAN,
  is_hr             BOOLEAN,
  is_bb             BOOLEAN,

  xwoba             NUMERIC(5,3),   -- xwOBA for this PA

  -- leverage / context fields can exist but are not used for v1 unicorns
  bases_state_before TEXT,
  outs_before       INT,
```

```
  score_diff_before INT,

  bat_order        INT,
  is_risp          BOOLEAN,

  PRIMARY KEY (pa_id)
);

CREATE INDEX idx_pa_facts_batter ON pa_facts (batter_id);
CREATE INDEX idx_pa_facts_pitcher ON pa_facts (pitcher_id);
CREATE INDEX idx_pa_facts_game ON pa_facts (game_id);
```

3.6 `team_market_context`

```
CREATE TABLE team_market_context (
  team_id          INT PRIMARY KEY REFERENCES teams(team_id),
  season_year      INT NOT NULL,
  attendance_score NUMERIC(5,3),  -- avg home attendance / league avg
  media_score      NUMERIC(5,3),  -- Market Power Index (MPI)
  market_weight    NUMERIC(5,3),  -- blended (0.85–1.30)
  market_weight_adj NUMERIC(5,3)  -- final adjusted factor used in scoring
);
```

- `market_weight = attendance_score * 0.70 + media_score * 0.30`, then clamped to `[0.85, 1.30]`.
- `market_weight_adj = 1 + (market_weight - 1) * 0.10` (so effect is capped at ±10%).

3.7 `pattern_templates`

```
CREATE TABLE pattern_templates (
  pattern_id        TEXT PRIMARY KEY,  -- e.g. 'UNQ-0101'
  name              TEXT NOT NULL,
  description_template TEXT NOT NULL,  -- e.g. '{{player_name}} has the most
100+ EV oppo hits on low-away sliders.'

  entity_type       TEXT NOT NULL,     -- 'batter' or 'pitcher'
  base_table        TEXT NOT NULL,     -- 'pitch_facts' or 'pa_facts'

  category          TEXT,              -- e.g. 'A_BARRELS','B_DIRECTION', etc.
  enabled           BOOLEAN DEFAULT TRUE,

  filters_json      JSONB NOT NULL,    -- list of filter conditions

  order_direction   TEXT NOT NULL,     -- 'asc' or 'desc'
  metric            TEXT NOT NULL,     -- logical metric name
```

```
  metric_expr       TEXT,              -- optional explicit SQL expression

  target_sample     INT,               -- T for sample penalty
  min_sample        INT DEFAULT 10,

  unicorn_weight    NUMERIC(4,2) DEFAULT 1.0,
  public_weight     NUMERIC(4,2) DEFAULT 1.0,

  -- complexity + validation
  complexity_score  INT NOT NULL,
  requires_count    BOOLEAN DEFAULT FALSE,
  count_value       TEXT,              -- if used, must be '3-0','0-2','3-2'

  CHECK (count_value IS NULL OR count_value IN ('3-0','0-2','3-2'))
);

CREATE INDEX idx_pattern_enabled ON pattern_templates (enabled);
```

3.8 unicorn_results

```
CREATE TABLE unicorn_results (
  run_date      DATE NOT NULL,
  pattern_id    TEXT REFERENCES pattern_templates(pattern_id),
  entity_type   TEXT NOT NULL,              -- 'batter'/'pitcher'
  entity_id     BIGINT NOT NULL,           -- player_id
  rank          INT NOT NULL,

  metric_value  NUMERIC(14,6) NOT NULL,
  sample_size   INT NOT NULL,

  z_raw         NUMERIC(14,6),
  z_adjusted    NUMERIC(14,6),
  score         NUMERIC(14,6) NOT NULL,

  PRIMARY KEY (run_date, pattern_id, entity_id)
);

CREATE INDEX idx_unicorn_results_date ON unicorn_results (run_date);
CREATE INDEX idx_unicorn_results_score ON unicorn_results (run_date, score
DESC);
```

3.9 unicorn_top50_daily

```
CREATE TABLE unicorn_top50_daily (
  run_date      DATE NOT NULL,
```

```
  rank          INT NOT NULL,
  entity_type   TEXT NOT NULL,
  entity_id     BIGINT NOT NULL,
  pattern_id    TEXT NOT NULL,

  metric_value  NUMERIC(14,6) NOT NULL,
  sample_size   INT NOT NULL,
  score         NUMERIC(14,6) NOT NULL,

  description   TEXT NOT NULL,

  PRIMARY KEY (run_date, rank)
);

CREATE INDEX idx_top50_player ON unicorn_top50_daily (run_date, entity_id);
```

---

4. ETL & BACKTESTING FLOW (2025 Season)

The system must support: - **Full-season backfill** for 2025. - **Nightly incremental updates** once in production.

4.1 Backfill 2025

1. Ingest all 2025 Statcast pitch + PA data (source: CSVs, pybaseball, or API).
2. Populate `games`, `players`, `teams` from that data.
3. Populate `pitch_facts` and `pa_facts` with all derived fields.
4. Populate `team_market_context` for 2025 using attendance + precomputed media scores.
5. Run the **unicorn evaluation pipeline** for each day in 2025 season:
6. For each `game_date`:
    - Filter `pitch_facts` and `pa_facts` to data through that date.
    - Evaluate all enabled `pattern_templates`.
    - Compute scores and write into `unicorn_results` with `run_date = game_date`.
    - Generate `unicorn_top50_daily` for that `run_date`.

4.2 Daily Production ETL

Once in production:

- Scheduled job (e.g., 3–5am ET) runs nightly.
- Steps:
- Load yesterday's Statcast data.
- Normalize and insert into `pitch_facts`, `pa_facts`, `games`.
- Update `team_market_context` periodically (weekly or monthly).
- Run unicorn evaluation pipeline for `run_date = yesterday`.
- Generate `unicorn_top50_daily` for that date.

5. Unicorn Evaluation Pipeline — Detailed Logic

5.1 Filter & Validate Patterns

For each row in `pattern_templates` where `enabled = TRUE`:

1. Ensure `complexity_score <= 4`.
2. If `requires_count = TRUE` then `count_value` must be one of `('3-0','0-2','3-2')`.
3. Inspect `filters_json` for banned concepts (sequence, inning, weather, etc.). Patterns that violate rules are skipped.

5.2 Build SQL Query from Template

General structure:

```
SELECT
  {group_by} AS entity_id,
  {metric_expr} AS metric_value,
  COUNT(*) AS sample_size
FROM
  {base_table}
JOIN games USING (game_id)
WHERE
  games.game_date <= :as_of_date
  AND {filter_conditions_from_filters_json}
GROUP BY
  {group_by}
HAVING
  COUNT(*) >= {min_sample}
ORDER BY
  metric_value {order_direction}
LIMIT {limit_per_pattern};
```

- `group_by` = `batter_id` or `pitcher_id` depending on `entity_type`.
- `metric_expr` comes from a metric registry in code keyed by `metric`.
- `filter_conditions` are built from `filters_json` entries like:

```
{
  "base_table": "pitch_facts",
  "conditions": [
    {"field": "pitch_type", "op": "IN", "value": ["SL"]},
    {"field": "loc_region", "op": "=", "value": "low_away"},
    {"field": "is_hard_hit", "op": "=", "value": true},
    {"field": "hit_direction", "op": "=", "value": "oppo"}
```

```
  ]
}
```

5.3 Metric Registry (Code-Level)

Example mapping in the application (Python pseudocode):

```
METRICS = {
  "count_hr": "SUM(CASE WHEN is_hr THEN 1 ELSE 0 END)",
  "hr_rate": "SUM(CASE WHEN is_hr THEN 1 ELSE 0 END)::float / COUNT(*)",
  "hard_hit_rate": "SUM(CASE WHEN is_hard_hit THEN 1 ELSE 0 END)::float /
COUNT(*)",
  "avg_ev": "AVG(launch_speed)",
  "xwoba_avg": "AVG(xwoba)",
  "whiff_rate": "SUM(CASE WHEN result_pitch = 'swinging_strike' THEN 1 ELSE 0
END)::float / NULLIF(SUM(CASE WHEN result_pitch IN
('swinging_strike','foul','in_play') THEN 1 ELSE 0 END), 0)",
}
```

Codex must implement this metric registry and extend it as needed.

5.4 Scoring per Pattern

For each pattern on a given `run_date`:

1. Execute the pattern SQL, producing rows:
2. `entity_id`
3. `metric_value`
4. `sample_size`
5. Compute mean and stddev of `metric_value` across rows.
6. Compute raw z-score:
7. If `order_direction = 'desc'`:
    - `z_raw = (metric_value - mean) / stddev`
8. If `order_direction = 'asc'`:
    - `z_raw = (mean - metric_value) / stddev`
9. If `stddev = 0` or `n < 3`, fallback to rank-based z:
    - Sort by metric respecting direction; assign `z_raw` in [0,1] based on rank.
10. Apply sample size penalty:

```
sample_weight = sqrt(n / (n + T))  # T = target_sample
z_adjusted = z_raw * sample_weight
```

1. Apply weights:
2. Look up **unicorn_weight** from `pattern_templates`.
3. Look up **public_weight** based on category (e.g., barrels, HR, etc. get higher).
4. Look up **team market_weight_adj** by joining player → team → team_market_context.
5. Final score (multiplicative form):

```
score = z_adjusted * unicorn_weight * public_weight * market_weight_adj
```

1. Insert one row per `(pattern_id, entity_id)` into `unicorn_results` with:
2. `run_date`, `pattern_id`, `entity_type`, `entity_id`, `metric_value`, `sample_size`, `z_raw`, `z_adjusted`, `score`, and derived `rank` (per pattern).

5.5 Pattern Weight & Public Weight Defaults

Codex should implement configurable defaults:

- `unicorn_weight` default = 1.0 for all patterns.
- `public_weight` based on category tier:
- Tier 1 (barrels, HR, xwOBA, fatigue, day/night, park splits) → 1.2
- Tier 2 (pitch type matchups, simple directional patterns) → 1.0
- Tier 3 (nerdier but allowed patterns) → 0.8

Values can be stored directly in `pattern_templates` and tuned after backtesting.

---

6. Daily Top 50 Selection

After all `unicorn_results` are inserted for a given `run_date`:

1. Query all results for that date:

```
SELECT *
FROM unicorn_results
WHERE run_date = :run_date;
```

1. In application code:
2. Sort rows by `score DESC`.
3. Iterate and pick the first occurrence for each unique `entity_id`.

4. Continue until 50 unique players are selected.

5. For each selected row:

6. Join `players` + `teams` to build human-readable description using `description_template`.

7. Insert into `unicorn_top50_daily` with final `rank = 1..50`.

8. Keep at least 14 days of history; optionally keep full-season.

---

7. Market Influence (Attendance + Media)

Implementation details:

1. **Attendance Score**:

2. `attendance_score = team_avg_home_attendance_2025 / league_avg_home_attendance_2025`

3. **Media Score**:

4. `media_score` from pre-defined Market Power Index per team (seeded into DB).

5. **Base Market Weight**:

```
market_weight = attendance_score * 0.70 + media_score * 0.30
clamp to [0.85, 1.30]
```

1. **Adjusted Market Weight for Scoring**:

```
market_weight_adj = 1 + (market_weight - 1) * 0.10
```

This ensures market influence has **at most ±10% effect** on final scores.

Codex must: - Seed `team_market_context` with 30 teams. - Update `attendance_score` once real 2025 attendance is known.

---

8. API Specification

REST-style JSON API.

8.1 `GET /api/unicorns/top50/today`

Returns the 50 unicorns for the latest available `run_date`.

Response shape:

```
[
  {
    "rank": 1,
    "player_id": 12345,
    "player_name": "Juan Soto",
    "team_name": "New York Yankees",
    "pattern_id": "UNQ-0081",
    "unicorn_title": "Most Barrels Last 50 PA",
    "description": "Juan Soto leads MLB in barrels over his last 50 PA (12
barrels).",
    "metric_value": 12,
    "sample_size": 50,
    "score": 3.21,
    "run_date": "2025-08-14"
  }
]
```

8.2 `GET /api/unicorns/top50/:date`

Same as above but for a specific `run_date` (YYYY-MM-DD).

8.3 `GET /api/players/:player_id`

Returns player profile with holy-grail metrics and recent unicorns.

```
{
  "player_id": 12345,
  "player_name": "Juan Soto",
  "team_id": 10,
  "team_name": "New York Yankees",
  "role": "hitter",
  "metrics": {
    "barrel_pct_last_50": 0.24,
    "hard_hit_pct_last_50": 0.52,
    "xwoba_last_50": 0.420,
    "contact_pct_last_50": 0.78,
    "chase_pct_last_50": 0.19
  },
  "recent_unicorns": [
```

```
    {
      "run_date": "2025-08-14",
      "pattern_id": "UNQ-0081",
      "description": "Juan Soto leads MLB in barrels over his last 50 PA (12
barrels).",
      "metric_value": 12,
      "score": 3.21
    }
  ]
}
```

8.4 `GET /api/teams`

List all teams.

8.5 `GET /api/teams/:team_id`

Return: - team info - roster split into hitters/starters/relievers

8.6 `GET /api/unicorns/history?days=14`

Return Top 50 per day for the last N days.

---

9. Frontend Specification (Minimal v1)

9.1 Tech Stack (Suggested)

- React or Next.js
- TailwindCSS (or equivalent utility-first CSS)
- Simple REST calls to backend

9.2 Pages & Layout

1. **Homepage**
2. Title: "Top 50 Unicorn Stats Today"
3. Date selector (default = latest `run_date`).
4. List of 50 unicorns, each row showing:
    - Rank
    - Player name + team
    - Short description
    - Metric value
    - Link to player page
5. **Teams Page**

6. Grid/list of 30 MLB teams.

7. **Team Detail Page**

8. Team info.
9. Buttons/tabs: Hitters / Starters / Relievers.
10. Each tab: list of player names.

11. **Player Detail Page**

12. Player name, team, role.
13. Display 5 holy-grail metrics.
14. List recent unicorns for that player.

9.3 No Articles / Editorial Content for v1

Focus is purely data + unicorns.

---

10. Implementation Order for Codex Max

1. **Create database schemas** (sections 3.1–3.9).
2. **Implement ETL ingestion for 2025 data** into `games`, `players`, `teams`, `pitch_facts`, `pa_facts`.
3. **Seed** `team_market_context` with media scores and placeholder attendance scores.
4. **Implement** `pattern_templates` **table** + load a **small initial set** of 20–50 well-defined patterns covering:
5. barrels last 50 PA,
6. hard-hit rate,
7. xwOBA vs pitch types,
8. day/night splits,
9. park splits,
10. fatigue (simple pitch count thresholds).
11. **Implement the unicorn evaluation engine** (section 5).
12. **Implement scoring** (z-scores, penalties, weights, market adjustment).
13. **Implement Top 50 daily generation** (section 6).
14. **Backfill entire 2025 season** and store results.
15. **Implement APIs** (section 8).
16. **Implement minimal frontend** (section 9).

---

11. Backtesting & Tuning Plan

After 2025 data is loaded and the system is running:

1. Inspect several days of `unicorn_top50_daily` outputs.
2. Identify:
3. patterns that never produce interesting unicorns
4. patterns that appear too often
5. patterns that are confusing or feel too nerdy
6. Adjust:
7. `enabled` flags in `pattern_templates`
8. `unicorn_weight` per pattern
9. `public_weight` per category
10. Re-run the evaluation for selected test dates.
11. Iterate until the Top 50 page is consistently:
12. readable,
13. interesting,
14. not dominated by noise,
15. not flooded by obscure conditions.

---

12. Notes for Developers

- All unicorn definitions must be **data-driven** via `pattern_templates` — no hardcoded queries.
- All scoring must be **per-pattern normalized** before global comparisons.
- Market influence must remain **light** (±10%) and must not dominate rankings.
- Global rules (counts, complexity, bans) are **non-negotiable**.
- The system must be built so that new patterns can be added and tested without redesigning the core.

This completes the **full project specification** for the MLB Unicorn Engine v1, suitable for Codex Max to implement end-to-end.

---

13. Tech Stack & Repository Structure (Recommended)

Codex should assume the following **baseline stack** unless explicitly changed later:

13.1 Backend

- Language: **Python 3.11+**
- Web framework: **FastAPI** (or similar async framework)
- DB: **PostgreSQL**
- Migrations: **Alembic**
- Task scheduler: **Celery + Redis** or cron-like scheduled jobs (for ETL + daily unicorn runs)

13.2 Frontend

- Framework: **Next.js (React)**
- Styling: **TailwindCSS**
- Data fetching: standard REST calls to the backend API

13.3 Repository Layout (Monorepo)

Suggested structure:

```
mlb-unicorn-engine/
  backend/
    app/
      api/
      core/
      db/
      etl/
      unicorns/
      schemas/
    alembic/
    tests/
  frontend/
    app/  (Next.js app dir)
    components/
    lib/
  infra/
    docker-compose.yml
    k8s-manifests/ (optional)
  docs/
    mlb_unicorn_engine_spec.md  (this document)
```

Codex should: - Create this structure. - Place this spec into `docs/mlb_unicorn_engine_spec.md`.

---

14. Initial Pattern Templates (Seed Set for v1)

Codex must define and insert an initial seed set of **20–50 unicorn patterns** into `pattern_templates`. Here is a **concrete starter set** of patterns that obey all global rules.

14.1 Hitters — Quality of Contact (Last 50 PA)

All use `entity_type = 'batter'` and `base_table = 'pa_facts'` (with joins to pitch_facts where needed).

1. **UNQ-H-0001 — Most Barrels Last 50 PA**

2. Description: `{{player_name}} has the most barrels in MLB over his last 50 PA ({{metric_value}} barrels).`
3. Metric: `count_barrels_last_50_pa`
4. Order: `desc`
5. Filters: last 50 PA per player, count barrels
6. Complexity: 1 (time window only)
7. **UNQ-H-0002 — Highest Hard-Hit Rate Last 50 PA**
8. Metric: `hard_hit_rate_last_50_pa`
9. Order: `desc`
10. Complexity: 1
11. **UNQ-H-0003 — Highest xwOBA Last 50 PA**
12. Metric: `xwoba_last_50_pa`
13. Order: `desc`
14. Complexity: 1
15. **UNQ-H-0004 — Lowest Chase % Last 50 PA**
16. Metric: `chase_pct_last_50_pa`
17. Order: `asc`
18. Complexity: 1
19. **UNQ-H-0005 — Highest Contact % Last 50 PA**
20. Metric: `contact_pct_last_50_pa`
21. Order: `desc`
22. Complexity: 1

14.2 Hitters — Directional Power Unicorns

Use `base_table = pitch_facts`, `entity_type = 'batter'`.

1. **UNQ-H-0010 — Most Opposite-Field HR (Season-To-Date)**
2. Filters: `is_hr = TRUE AND hit_direction = 'oppo'`
3. Metric: `COUNT(*)`
4. Order: `desc`
5. Complexity: 1 (direction)
6. **UNQ-H-0011 — Most Pulled HR (Season-To-Date)**

7. Filters: `is_hr = TRUE AND hit_direction = 'pull'`
8. Metric: `COUNT(*)`
9. Order: `desc`
10. Complexity: 1 (direction)

11. **UNQ-H-0012 — Most 100+ EV Opposite-Field Hits on Low-Away Sliders (Season-To-Date)**

12. Filters:
    - `launch_speed >= 100`
    - `hit_direction = 'oppo'`
    - `loc_region = 'low_away'`
    - `pitch_type = 'SL'`
13. Metric: `COUNT(*)`
14. Order: `desc`
15. Complexity: 4 (EV + direction + location + pitch type)

14.3 Hitters — Count-Based (Only Allowed Counts)

1. **UNQ-H-0020 — Most HR in 3–0 Counts (Season-To-Date)**
2. Filters:
    - `is_hr = TRUE`
    - `count_str = '3-0'`
3. Metric: `COUNT(*)`
4. Order: `desc`
5. Complexity: 2 (HR + count)

6. **UNQ-H-0021 — Highest xwOBA in 3–2 Counts (Season-To-Date, min PA threshold)**

    - Filters:
    - `count_str = '3-2'`
    - Metric: `AVG(xwoba)` over relevant PAs
    - Order: `desc`
    - Complexity: 1 (count)

14.4 Starters — Last 3 Starts

Use `entity_type = 'pitcher'`, combine `pa_facts` + metadata about last 3 starts.

1. **UNQ-P-0100 — Lowest xwOBA Allowed Over Last 3 Starts**

    - Filters: PAs faced in last 3 games started by pitcher
    - Metric: `AVG(xwoba)`
    - Order: `asc`
    - Complexity: 1 (time window)

2. **UNQ-P-0101 — Highest Whiff % Over Last 3 Starts**
    - Metric: `whiff_rate_last_3_starts`
    - Order: `desc`
3. **UNQ-P-0102 — Lowest Hard-Hit % Allowed Over Last 3 Starts**
    - Metric: `hard_hit_allowed_pct_last_3_starts`
    - Order: `asc`
4. **UNQ-P-0103 — Most Strikeouts Over Last 3 Starts**
    - Metric: `total_k_last_3_starts`
    - Order: `desc`

14.5 Relievers — Last 5 Appearances

1. **UNQ-R-0200 — Lowest xwOBA Allowed Over Last 5 Relief Appearances**
2. **UNQ-R-0201 — Highest Whiff % Over Last 5 Relief Appearances**
3. **UNQ-R-0202 — Lowest Hard-Hit % Allowed Over Last 5 Relief Appearances**

14.6 Fatigue Patterns (Pitcher)

1. **UNQ-P-0300 — Most HR Allowed After 75+ Pitches (Season-To-Date)**
    - Filters:
    - `is_hr = TRUE`
    - `pitch_number_game >= 75`
    - Metric: `COUNT(*)`
    - Order: `desc`
    - Complexity: 2
2. **UNQ-P-0301 — Lowest xwOBA Allowed After 70+ Pitches (min PA)**
    - Filters:
    - `pitch_number_game >= 70`
    - Metric: `AVG(xwoba)`
    - Order: `asc`

14.7 Park-Specific Unicorns

1. **UNQ-H-0400 — Most HR in Coors Field (Season-To-Date)**
    - Filters:
    - `games.venue_id = {COORS_FIELD_ID}`
    - `is_hr = TRUE`
    - Metric: `COUNT(*)`
    - Order: `desc`

Codex must: - Implement a **seeding script** to insert these patterns into `pattern_templates` with appropriate `filters_json`, `unicorn_weight`, `public_weight`, `target_sample`, and `min_sample`. - Ensure all seeded patterns obey global rules (no banned counts, no >4 conditions).

---

15. Local Development & Running the System

Codex should provide basic scripts / commands:

15.1 Backend

- `make migrate` → run Alembic migrations
- `make seed-patterns` → insert initial `pattern_templates`
- `make etl-backfill-2025` → run backfill ETL for 2025 season
- `make run-unicorns date=YYYY-MM-DD` → compute unicorns and Top 50 for given date
- `make api` → run FastAPI server

15.2 Frontend

- `npm install`
- `npm run dev` → start Next.js dev server

---

16. How to Use This Spec with Codex Max

When handing this to Codex Max, the user should:

1. Ensure this document is accessible as `docs/mlb_unicorn_engine_spec.md` in the repo or pasted into the Codex context.
2. Give Codex a **clear, constrained instruction**, e.g.:

   "You are building the MLB Unicorn Engine backend according to `docs/mlb_unicorn_engine_spec.md`. Start by implementing the PostgreSQL schema (all tables in section 3) and Alembic migrations in the `backend` project. Then implement the ETL pipeline to ingest 2025 Statcast data into `pitch_facts` and `pa_facts`. Next, implement the unicorn evaluation engine as described in section 5, including scoring and the daily Top 50 generation. Do not start frontend work yet."

1. After backend is working and backtesting is confirmed, the next Codex instruction could be:

   "Using the same spec, implement the FastAPI routes described in section 8 and the minimal Next.js frontend described in section 9, using the existing backend as the data source."

Codex must always: - Treat this spec as the **authoritative source**. - Ask for clarification only when something is truly underspecified.

---

17. Ready State

This document now defines **everything needed** for Codex Max to: - Scaffold the repo - Create the DB schema - Build ETL to ingest 2025 data - Implement unicorn computation + scoring - Generate Top 50 daily unicorns - Expose APIs - Build a minimal but functional website.

No additional architectural decisions are required to begin implementation.