Realtime Scoreboard Module — API Service Specification

Backend Specification

September 17, 2025

1 Goal

Provide a backend module that maintains a **top-10 scoreboard** with **live updates** while preventing unauthorized or fraudulent score increases.

This document defines API contracts, data model, execution flow, security controls, and operational concerns for the engineering team to implement.

2 High-level Architecture

- **API Service (REST + WebSocket/SSE)** — accepts authenticated action completion events and exposes read endpoints for the scoreboard.
- **Auth Service** — issues short-lived JWT access tokens.
- **Persistent Store (PostgreSQL)** — source of truth for users, score events, and running totals.
- **Cache/Leaderboard (Redis)** — Redis Sorted Sets compute and serve the top–N; also used for pub/sub.
- **Async Workers** — consume a queue of validated action events, apply scoring rules, update DB + Redis atomically/idempotently, and broadcast updates.
- **Message/Task Queue** (e.g., SQS, RabbitMQ, or Postgres SKIP LOCKED) — buffers validated events and enables retries without double counting.

3 Execution Flow (Happy Path)

1. User performs an action in the client.
2. Client submits `POST /v1/actions/complete` with JWT and an Idempotency-Key.
3. API validates JWT, schema, timestamp window, and rate limits.
4. API enqueues a normalized action event into the message queue.
5. Worker consumes the job, inserts an immutable `score_event` if new, updates running totals, mirrors to Redis, and publishes a leaderboard update.
6. API immediately returns `202 Accepted`; clients receive live updates via WebSocket/SSE.

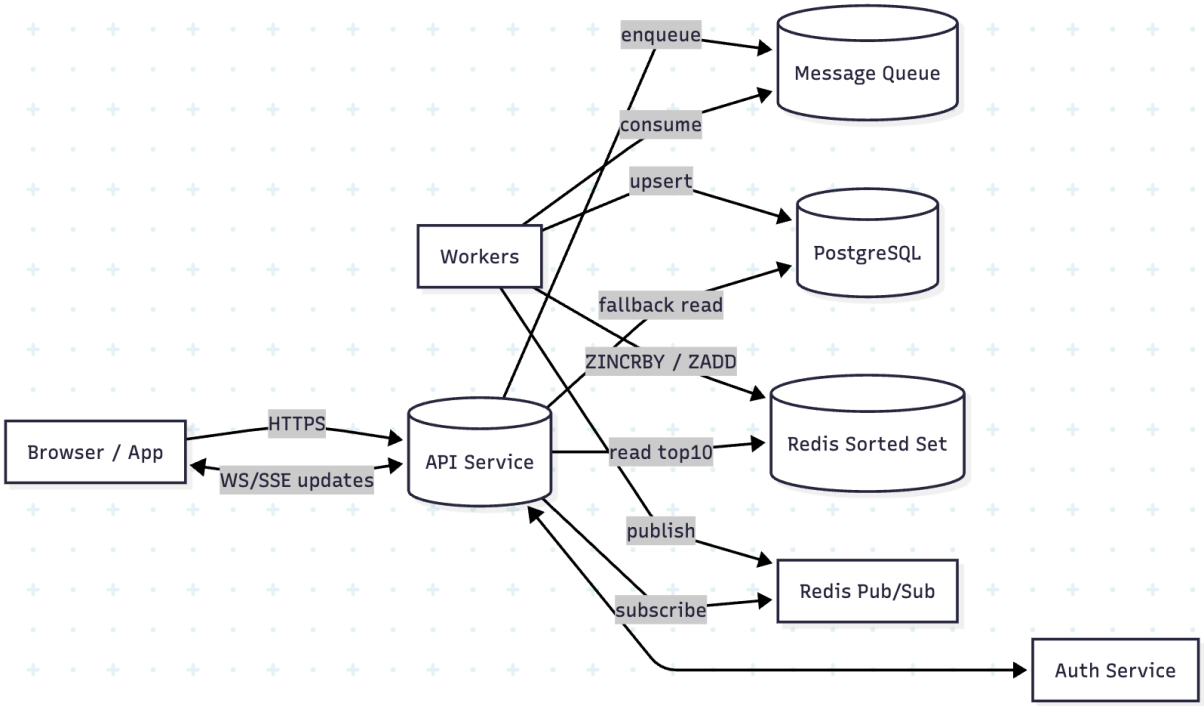

Figure 1: Component Diagram.

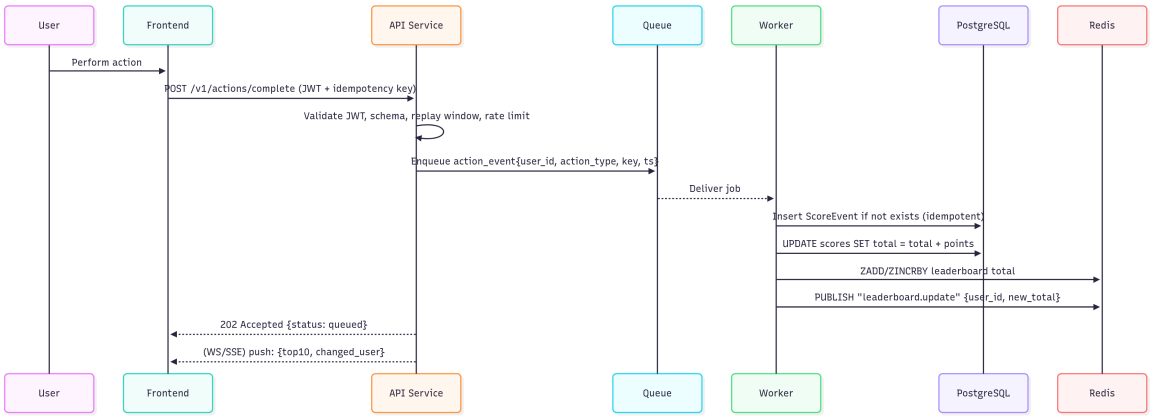

Figure 2: Sequence Diagram.

Sequence Diagram

Notes. Points are computed *server-side* from `action_type` using a scoring rules table; clients never choose the points value.

4 API Design

Base URL: `/v1`

4.1 Submit an action completion

`POST /v1/actions/complete`

Headers

- `Authorization: Bearer <access_token>` (JWT, 5–15 min TTL)

- `Idempotency-Key: <uuidv4>`
- `Content-Type: application/json`

Body

```
{
  "action_type": "watch_video",
  "client_ts": "2025-09-17T12:34:56Z",
  "metadata": {
    "action_instance_id": "abc123",
    "extra": "opaque client data"
  }
}
```

Behavior

- Validate token, schema, and rate limit per user + IP.
- Reject if request timestamp is outside a rolling window (e.g., ±10 min).
- Compute `points = scoring_rules[action_type]` (server-side).
- Generate `event_hash = sha256(user_id + action_type + idempotency_key)` for dedupe.
- Enqueue normalized event (user, action, points, idempotency key, client ts, ip, ua).
- Respond `202 Accepted` with a processing handle.

Responses

- `202 Accepted` `{"status": "queued", "event_id": "<opaque-id>"}`
- `400 Bad Request` (schema, unknown action)
- `401 Unauthorized`
- `409 Conflict` (duplicate idempotency key for user)
- `429 Too Many Requests`

4.2 Read the Top 10 Leaderboard

`GET /v1/leaderboard?limit=10`

Response

```
{
  "as_of": "2025-09-17T12:40:00Z",
  "entries": [
    {"rank": 1,"user_id": "u_1", "display_name": "Alex", "score": 999}
  ]
}
```

4.3 Read My Score

`GET /v1/me/score`

Response

```
{"user_id":"u_1", "score": 1234}
```

4.4 Realtime stream (live updates)

Prefer WebSocket; provide SSE fallback.

WebSocket: `GET /v1/leaderboard/stream` (JWT on upgrade)

Push payload:

```
{"type":"leaderboard.update","top10":[...],
 "changed":{"user_id":"u_1","new_total":1240}}
```

SSE: `GET /v1/leaderboard/stream.sse` (same payload via events). Coalesce updates (250–500 ms). Heartbeat every 20–30 s.

5 Data Model (PostgreSQL)

```sql
-- Users
CREATE TABLE users (
  id BIGSERIAL PRIMARY KEY,
  display_name TEXT NOT NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- Scoring rules (server-controlled)
CREATE TABLE scoring_rules (
  action_type TEXT PRIMARY KEY,
  points INT NOT NULL CHECK (points > 0),
  active BOOLEAN NOT NULL DEFAULT TRUE,
  updated_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- Running totals
CREATE TABLE scores (
  user_id BIGINT PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,
  total BIGINT NOT NULL DEFAULT 0,
  updated_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- Immutable event log (for audit and anti-fraud)
CREATE TABLE score_events (
  id BIGSERIAL PRIMARY KEY,
  event_hash BYTEA UNIQUE NOT NULL, -- sha256(user_id, action_type, idempotency_key)
  user_id BIGINT NOT NULL REFERENCES users(id),
  action_type TEXT NOT NULL REFERENCES scoring_rules(action_type),
  points INT NOT NULL CHECK (points > 0),
  idempotency_key UUID NOT NULL,
  client_ts TIMESTAMPTZ,
  source_ip INET,
  user_agent TEXT,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

CREATE INDEX idx_scores_total_desc ON scores(total DESC);
CREATE INDEX idx_events_user_ts ON score_events(user_id, created_at DESC);
```

Redis keys

- `lb:global` (ZSET) — member: user_id, score: total

- `user:display:<id>` (STRING) — cached display name (TTL 1h)
- `pubsub:leaderboard.update` — channel for push updates

6 Idempotency, Consistency & Concurrency

- Each client request must include an Idempotency-Key.
- Worker transaction:
  1. Insert into `score_events` with `ON CONFLICT DO NOTHING`.
  2. If inserted, upsert `scores` and return new total.
  3. Mirror the new total to Redis and publish update.
- Retries are safe; at-least-once delivery is acceptable due to deduplication.

7 Security & Anti-Fraud

1. OAuth2/OIDC with short-lived JWTs (5–15 min).
2. Never trust client-provided points; derive from `action_type`.
3. Replay protection: Idempotency-Key + timestamp window + event hash.
4. Rate limits per user and per IP (Redis token bucket).
5. Abuse heuristics: velocity caps, device/IP reputation, ASN denylist, optional CAPTCHA.
6. HTTPS only, HSTS, least-privilege DB/Redis roles, immutable audit log ($\geq$ 90 days).
7. WebSocket auth on upgrade; short max connection lifetime.

8 Performance Targets

- Read: `GET /leaderboard` P95 $\leq$ 20 ms from Redis.
- Write: action $\rightarrow$ visible update P95 $\leq$ 500 ms.
- Throughput: $\geq$ 1k events/s with horizontal workers.

9 Observability

- Metrics: request rate, latency, error codes, queue depth, worker lag, Redis ops, pub/sub fan-out, dropped frames.
- Logs: structured JSON including user_id, event_id, idempotency_key, outcome.
- Tracing: propagate trace IDs across API $\rightarrow$ queue $\rightarrow$ worker $\rightarrow$ DB/Redis.
- Alerts: spike in 5xx/429, duplicate surge, top-10 staleness $>$ 2 s, retry storms.

10 SDK / Integration Notes (Frontend)

- Prefer WebSocket; fallback SSE; last resort polling every $\sim$5 s.
- Always send Idempotency-Key; cache recent keys locally.
- On `202 Accepted`, optimistic UI update, reconcile with stream.

Example cURL

```
curl -X POST https://api.example.com/v1/actions/complete \
  -H "Authorization: Bearer $TOKEN" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{"action_type":"watch_video","client_ts":"2025-09-17T12:34:56Z","metadata":{"
      action_instance_id":"abc123"}}'
```

11 Testing Strategy

Unit, integration, property, load, security, and contract tests ensure correctness and resilience (dedupe, replay, skew, hot keys, etc.).

12 Operational Runbook

- Nightly rebuild check of Redis ZSET from DB totals.
- Cache-loss recovery: rebuild from `scores`.
- Shard Redis if needed; aggregate cross-shard top-10 in API or worker.
- Disaster recovery: DB PITR + durable queue; RPO $\leq$ 1 min, RTO $\leq$ 30 min.

13 Future Improvements

Scoped leaderboards (per region/season), heuristic/ML cheating detection, webhooks, privacy controls (opt-out or initials).

14 Non-Goals

Defining the action itself; admin UI.

15 Acceptance Criteria

- Top-10 accurate within $\sim$1 s under normal load.
- Duplicate submissions never inflate scores.
- Unauthorized clients cannot submit actions.
- REST & realtime contracts implemented and documented (OpenAPI/AsyncAPI).