Jumpshot PRD v2 – full build spec  
 29 May 2025

contents  
 → overview  
 → data source precedence

→ production accepts input only from the demo‑parser worker.  
 → csv loaders are permitted exclusively in local development; staging and production reject them in the ingest function.  
 → any new source must be declared in this section before code reaches main.

**database connection (replit & ci)** Supabase’s raw Postgres endpoint:

Request the API keys you need.

Database Modification Restrictions:

"NEVER INSERT, UPDATE, or DELETE any data in production database under any circumstances"

"NEVER populate tables with sample, mock, test, or synthetic data"

"NEVER assume database is empty or safe to modify without explicit written permission"

"ANY database write operation requires explicit approval in writing before execution"

"Development environment ONLY for data seeding - production databases are READ-ONLY"

Developer Constraints:

"Treat all database connections as production regardless of environment"

"Build interfaces with empty states FIRST - never populate data to demonstrate functionality"

"Database modifications are EXCLUSIVELY through approved ingestion pipeline"

"No direct SQL operations on tables containing business data"

**connection usage pattern**

// RPC / Row‑level‑secure queries

import { createClient } from '@supabase/supabase-js';

export const supabase = createClient(process.env.SUPABASE\_URL!, process.env.SUPABASE\_ANON\_KEY!);

// Heavy ingest or migrations

import { Pool } from 'pg';

export const sql = new Pool({ connectionString: process.env.SUPABASE\_DB\_URI! });

This dual‑path keeps the app lightweight while bulk COPY operations bypass PostgREST rate limits.

ingestion pipeline

→ external worker parses .dem at 128 hz, samples every eighth tick for Postgres, archives full fidelity parquet.  
 → sampled snapshot is pushed to Supabase Storage under telemetry/snapshots/<match\_id>.ndjson.  
 → Storage webhook triggers edge\_copy() which streams ndjson into Postgres using copy from stdin inside an unlogged staging table, then upserts into fact tables.  
 → on success the function refreshes every materialised view, runs post‑load procs, moves the object to telemetry/processed, writes ingest\_log, and broadcasts supabase.realtime channel ingest\_status.  
 → latency target from upload to dashboard refresh < 60 s.  
 → rollback on failure leaves the snapshot in telemetry/failed and records the error in ingest\_log.error\_txt.

Below is the schema definition for the tables:

kill\_stats:

create table public.kill\_stats (

steam\_id bigint not null,

kills integer null,

headshots integer null,

wallbang\_kills integer null,

no\_scope integer null,

through\_smoke integer null,

airbone\_kills integer null,

blind\_kills integer null,

victim\_blind\_kills integer null,

awp\_kills integer null,

pistol\_kills integer null,

first\_kills integer null,

ct\_first\_kills integer null,

t\_first\_kills integer null,

first\_deaths integer null,

ct\_first\_deaths integer null,

t\_first\_deaths integer null,

event\_id integer not null,

constraint kill\_stats\_pkey primary key (steam\_id, event\_id),

constraint unique\_kill\_event unique (steam\_id, event\_id),

constraint unique\_steam\_event\_kill\_stats unique (steam\_id, event\_id),

constraint fk\_kill\_stats\_event foreign KEY (event\_id) references events (event\_id),

constraint kill\_stats\_steam\_id\_fkey foreign KEY (steam\_id) references players (steam\_id)

) TABLESPACE pg\_default;

genral\_stats:

create table public.general\_stats (

steam\_id bigint not null,

assists double precision null,

deaths double precision null,

trade\_kills double precision null,

trade\_deaths double precision null,

kd double precision null,

k\_d\_diff double precision null,

adr\_total double precision null,

adr\_ct\_side double precision null,

adr\_t\_side double precision null,

kast\_total double precision null,

kast\_ct\_side double precision null,

kast\_t\_side double precision null,

total\_rounds\_won double precision null,

t\_rounds\_won double precision null,

ct\_rounds\_won double precision null,

event\_id integer not null,

constraint general\_stats\_pkey primary key (steam\_id, event\_id),

constraint unique\_general\_event unique (steam\_id, event\_id),

constraint unique\_steam\_event\_general\_stats unique (steam\_id, event\_id),

constraint fk\_general\_stats\_event foreign KEY (event\_id) references events (event\_id),

constraint general\_stats\_steam\_id\_fkey foreign KEY (steam\_id) references players (steam\_id)

) TABLESPACE pg\_default;

utility\_stats:

create table public.utility\_stats (

steam\_id bigint not null,

assisted\_flashes integer null,

flahes\_thrown integer null,

ct\_flahes\_thrown integer null,

t\_flahes\_thrown integer null,

flahes\_thrown\_in\_pistol\_round integer null,

he\_thrown integer null,

ct\_he\_thrown integer null,

t\_he\_thrown integer null,

he\_thrown\_in\_pistol\_round integer null,

infernos\_thrown integer null,

ct\_infernos\_thrown integer null,

t\_infernos\_thrown integer null,

infernos\_thrown\_in\_pistol\_round integer null,

smokes\_thrown integer null,

ct\_smokes\_thrown integer null,

t\_smokes\_thrown integer null,

smokes\_thrown\_in\_pistol\_round integer null,

util\_in\_pistol\_round integer null,

total\_util\_thrown integer null,

total\_util\_dmg integer null,

ct\_total\_util\_dmg integer null,

t\_total\_util\_dmg integer null,

event\_id integer not null,

constraint utility\_stats\_pkey primary key (steam\_id, event\_id),

constraint unique\_steam\_event\_utility\_stats unique (steam\_id, event\_id),

constraint unique\_utility\_event unique (steam\_id, event\_id),

constraint fk\_utility\_stats\_event foreign KEY (event\_id) references events (event\_id),

constraint utility\_stats\_steam\_id\_fkey foreign KEY (steam\_id) references players (steam\_id)

) TABLESPACE pg\_default;

teams:

create table public.teams (

id serial not null,

team\_clan\_name text not null,

constraint teams\_pkey primary key (id),

constraint teams\_team\_clan\_name\_key unique (team\_clan\_name)

) TABLESPACE pg\_default;

players:

create table public.players (

steam\_id bigint not null,

user\_name text null,

constraint players\_pkey primary key (steam\_id)

) TABLESPACE pg\_default;

player\_match\_summary:

create table public.player\_match\_summary (

steam\_id bigint not null,

file\_id integer not null,

team\_id integer null,

event\_id integer not null,

constraint player\_match\_summary\_pkey primary key (steam\_id, file\_id, event\_id),

constraint unique\_steam\_file\_event unique (steam\_id, file\_id, event\_id),

constraint fk\_player\_match\_summary\_event foreign KEY (event\_id) references events (event\_id),

constraint player\_match\_summary\_event\_id\_fkey foreign KEY (event\_id) references events (event\_id),

constraint player\_match\_summary\_file\_id\_fkey foreign KEY (file\_id) references matches (file\_id),

constraint player\_match\_summary\_steam\_id\_fkey foreign KEY (steam\_id) references players (steam\_id)

) TABLESPACE pg\_default;

rounds:

create table public.rounds (

id serial not null,

round\_num integer null,

start integer null,

freeze\_end integer null,

"end" integer null,

official\_end integer null,

winner text null,

reason text null,

bomb\_plant double precision null,

bomb\_site text null,

ct\_team\_clan\_name text null,

t\_team\_clan\_name text null,

winner\_clan\_name text null,

ct\_team\_current\_equip\_value double precision null,

t\_team\_current\_equip\_value double precision null,

ct\_losing\_streak integer null,

t\_losing\_streak integer null,

ct\_buy\_type text null,

t\_buy\_type text null,

advantage\_5v4 text null,

file\_id integer null,

event\_id integer null,

match\_name text null,

constraint rounds\_pkey primary key (id),

constraint unique\_round\_per\_match unique (round\_num, match\_name),

constraint rounds\_file\_id\_fkey foreign KEY (file\_id) references matches (file\_id)

) TABLESPACE pg\_default;

matches:

create table public.matches (

file\_id integer not null default nextval('matches\_file\_id\_seq'::regclass),

match\_name text not null,

event\_id integer null,

constraint matches\_pkey primary key (file\_id),

constraint unique\_file\_per\_event unique (match\_name, event\_id),

constraint matches\_event\_id\_fkey foreign KEY (event\_id) references events (event\_id)

) TABLESPACE pg\_default;

events:

create table public.events (

event\_id integer not null,

event\_name text not null,

constraint events\_pkey primary key (event\_id)

) TABLESPACE pg\_default;

player\_history:

create table public.player\_history (

id serial not null,

steam\_id bigint not null,

team\_id integer not null,

constraint player\_history\_pkey primary key (id),

constraint unique\_steam\_team unique (steam\_id, team\_id),

constraint fk\_steam foreign KEY (steam\_id) references players (steam\_id),

constraint fk\_team foreign KEY (team\_id) references teams (id)

) TABLESPACE pg\_default;

metrics and algorithms

→ piv weighting: 0.35 kills, 0.25 survival, 0.2 trades, 0.1 util damage, 0.1 bomb impact.  
 → z‑scores: z = (x − µ) ÷ σ event and global stored in columns.  
 → bayesian icf: prior µ₀ from metric\_global\_stats, k = 500 rounds.

w := n\_rounds::real / (n\_rounds + 500);

icf\_mean := w \* raw\_contribution + (1 - w) \* mu0;

std\_err := sd / sqrt(n\_rounds + 500);

ci\_lower := icf\_mean - 1.96 \* std\_err;

ci\_upper := icf\_mean + 1.96 \* std\_err;

→ util roi: util\_damage ÷ util\_spend where util\_spend > 0.  
 → synergy echo: max\_streak ÷ rounds\_together; calculated in sp\_refresh\_synergy() nightly.  
 → peek latency index: median( sqrt(vel\_x² + vel\_y²) ) from frame ‑1 before each kill.

observability stack

→ node exposes /metrics via prom‑client default and custom gauges.  
 → hosted prometheus scrapes the api and ingest worker.  
 → grafana dashboard "piv health" charts request p95, queue depth, compute latency, error rate.  
 → alertmanager routes breaches to slack #jumpshot‑alerts.  
 → local dev runs same stack via docker compose (appendix a).

front‑end data layer

→ tanstack react query client in \_app.tsx; default staleTime 30 s; retry 3.  
 → rpcs:

supabase.rpc('fn\_player\_round\_stats',{scope:'event'})

supabase.rpc('peek\_latency\_v',{match\_id,steam\_id})

→ recharts heatmap overlays svg minimap.  
 → realtime channel invalidates queries on ingest broadcast.

contributor workflow

→ branch → environment mapping: dev personal, staging shared, main production.  
 → migrations via supabase db commit; lint in pre‑push hook.  
 → secrets .env driven; pre‑commit hook prevents accidental add.  
 → seed fixtures in prisma/seed.sql generate sample event with ten rounds for ui.  
 → project uses conventional commits (feat:,fix:,chore:).

roadmap and kpis

phase 1 weeks 1‑2 → core schema, ingest, basic dashboards.  
 phase 2 week 3 → visuals and positional telemetry.  
 phase 2 week 4 → advanced metrics and bayesian icf.  
 phase 3 weeks 5‑6 → ai test‑case generator and config optimiser.  
 phase 4 q4 → snippet suggestions and map‑specific optimiser pilots.

kpi targets  
 → ingest latency < 60 s  
 → api p95 < 300 ms  
 → scenario generation success > 95 %  
 → scrim win uplift +3 pp after optimiser.

appendix a docker compose metrics

version: '3'

services:

prometheus:

image: prom/prometheus

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml

ports:

- "9090:9090"

grafana:

image: grafana/grafana

ports:

- "3001:3000"

environment:

- GF\_SECURITY\_ADMIN\_PASSWORD=admin

volumes:

- grafana-storage:/var/lib/grafana

volumes:

grafana-storage:

prometheus.yml example scrape:

scrape\_configs:

- job\_name: jumpshot-api

static\_configs:

- targets: ['host.docker.internal:4000']

- job\_name: ingest-worker

static\_configs:

- targets: ['