Project statement (short)

The backend is a single **Cloudflare Worker (TypeScript)** that serves three endpoints and relies on **KV** (hot cache) and **D1** (persistent store). On cache miss, it calls **one HTTPS endpoint** (provided by your Snowflake teammate) and returns an **LDraw-Part Array JSON** per agent.

Endpoints:

- `POST /api/design`
- `POST /api/vote`
- `GET /api/leaderboard`

Minimal file tree (3 files total)

```
/worker
  ├─ wrangler.toml          # Worker config + KV/D1 bindings + vars
  ├─ schema.sql             # D1 schema for designs + votes
  └─ src/index.ts           # All logic (routing, KV/D1, validator,
rate limit, Snowflake call)
```

(Cloudflare will compile TS to JS for you.)

---

1) `wrangler.toml` (bindings & config)

Bindings

- **KV** namespace → `DESIGN_CACHE`
- **D1** database → `DB`

Vars (strings)

- `MODEL_MAX_ITEMS = "200"`
- `MODEL_MAX_BYTES = "1048576"` (1 MB)
- `POS_MIN = "-10000"`
- `POS_MAX = "10000"`
- `SNOWFLAKE_TIMEOUT_MS = "5000"`
- `RATE_LIMIT_REQUESTS = "10"`
- `RATE_LIMIT_WINDOW_S = "60"`
- `FRONTEND_ORIGIN = "https://<your-pages-domain>"`

Secrets (set via `wrangler secret put`)

- `SNOWFLAKE_ENDPOINT` (HTTPS URL your teammate exposes)
- Optional: `SNOWFLAKE_API_KEY` (if that endpoint requires one)

CLI quickstart

```
wrangler d1 create lego_db

wrangler kv namespace create DESIGN_CACHE

wrangler secret put SNOWFLAKE_ENDPOINT

wrangler d1 execute lego_db --file=./schema.sql

wrangler dev
```

```
wrangler deploy
```

---

2) `schema.sql` (D1 schema)

```
CREATE TABLE IF NOT EXISTS designs (
  hash_id    TEXT PRIMARY KEY,
  prompt     TEXT NOT NULL,
  agent_type TEXT NOT NULL,
  model_json TEXT NOT NULL,
  created_at TEXT DEFAULT (strftime('%Y-%m-%dT%H:%M:%fZ','now'))
);

CREATE TABLE IF NOT EXISTS votes (
  agent_type TEXT NOT NULL,
  prompt     TEXT NOT NULL,
  count      INTEGER NOT NULL DEFAULT 0,
  PRIMARY KEY (agent_type, prompt)
);
```

---

3) `src/index.ts` (single file Worker)

3.1 Types (shared)

```
export interface LDrawPart {
  ldraw_part: string;              // e.g. "3001.dat"
  color: number;                   // LDraw color code, e.g. 16
  pos: [number, number, number];   // [x, y, z] in LDU
  rot: [0|90|180|270, 0|90|180|270, 0|90|180|270];
}

export interface Env {
  DESIGN_CACHE: KVNamespace;
  DB: D1Database;
  MODEL_MAX_ITEMS: string;
  MODEL_MAX_BYTES: string;
  POS_MIN: string;
  POS_MAX: string;
  SNOWFLAKE_TIMEOUT_MS: string;
  RATE_LIMIT_REQUESTS: string;
  RATE_LIMIT_WINDOW_S: string;
  FRONTEND_ORIGIN: string;
  SNOWFLAKE_ENDPOINT: string;
  SNOWFLAKE_API_KEY?: string;
}
```

3.2 Routing (one `fetch` handler)

Routes:

- `POST /api/design`
- `POST /api/vote`
- `GET /api/leaderboard`
- `GET /healthz` (returns `{ok:true}`)

Always add:

- `request_id` (`ISO + "_" + random`)
- CORS headers (allow `env.FRONTEND_ORIGIN`)
- Body size guard (~256 KB)

3.3 Core logic per endpoint

A) `POST /api/design`

Request

```
{ "prompt": "space rover", "agent_types": ["A","B"] }
```

Flow (for each agent_type)

1. `hash_id = sha256(lower(prompt) + '|' + agent_type)`

2. **KV get** `DESIGN_CACHE.get(hash_id)`
    - If hit → `designs[agent] = JSON.parse(value)`, `cache[agent] = true`
3. Else **D1 read**:
    - `SELECT model_json FROM designs WHERE hash_id=?`
    - If hit → parse, set `cache[agent]=true`, `KV.put(hash_id, model_json)` best-effort
4. Else **Snowflake HTTPS**:
    - `model = await generateModel(env, prompt, agent_type, seed, timeout=SNOWFLAKE_TIMEOUT_MS)`
    - **validateModel(model, env)** (shape/limits)
    - **D1 insert** row with `model_json = JSON.stringify(model)`
    - **KV put** same JSON
    - `designs[agent] = model`, `cache[agent] = false`

Response (success)

```
{
  "designs": { "A": [/* LDrawPart */], "B": [/* LDrawPart */] },
  "cache":   { "A": true, "B": false },
  "request_id": "..."
}
```

Errors

- `400 VALIDATION_FAILED` (bad `ldraw_part/color/pos/rot`, too big, too many items)
- `502 GENERATION_TIMEOUT` (Snowflake call exceeded timeout)
- `502 GENERATION_FAILED` (non-200/invalid JSON)
- `429 RATE_LIMITED` (see limiter below)

Rate limit

- Enforce **10 req/min/IP** on `/api/design` using KV counter: key `rl:<ip>:<minuteISO>`, TTL = `RATE_LIMIT_WINDOW_S`.

B) `POST /api/vote`

Request

```
{ "agent_type": "B", "prompt": "space rover" }
```

Flow

1. Validate fields exist.

Upsert D1:

```
INSERT INTO votes (agent_type, prompt, count)
VALUES (?1, ?2, 1)
ON CONFLICT(agent_type, prompt) DO UPDATE SET count = count + 1;
```

2.

Aggregate:

```
SELECT agent_type, SUM(count) AS total FROM votes GROUP BY agent_type;
```

3.
4. Return:

```
{ "leaderboard": { "A": 21, "B": 22 }, "request_id": "..." }
```

C) `GET /api/leaderboard`

1. Same aggregate query as above.
2. Respond with:

```
{ "leaderboard": { "A": n, "B": n }, "updated_at": "ISO",
"request_id": "..." }
```

3.4 Model validation (inline, minimal)

Input is an array of `LDrawPart`. Enforce:

- Array length: `1 .. Number(env.MODEL_MAX_ITEMS)` (default 200)
- JSON size: `<= Number(env.MODEL_MAX_BYTES)` (default 1 MB)
- Each item:
    - `ldraw_part`: `string` and matches `/^[^\s]{1,64}\.dat$/`
    - `color`: `integer` in `0..1023`
    - `pos`: 3 numbers, each in `[Number(env.POS_MIN) .. Number(env.POS_MAX)]`

- `rot`: exactly 3 ints, each in `{0,90,180,270}`

Error shape (on failure)

```
{
  "error": {
    "code": "VALIDATION_FAILED",
    "message": "Invalid LDraw part array",
    "details": [{ "path": "[i].rot[1]", "msg": "must be one of 0,90,180,270" }]
  },
  "request_id": "..."
}
```

3.5 Snowflake HTTP adapter (single, simple)

Contract: your teammate's endpoint **must** return:

```
{ "model": [ { "ldraw_part":"3001.dat", "color":16, "pos":[0,24,0], "rot":[0,0,0] }, ... ] }
```

Signature

```
async function generateModel(
  env: Env,
  prompt: string,
  agentType: string,
```

```
  seed: number,

  signal: AbortSignal

): Promise<{ model: LDrawPart[] } | { error: { code: string; message: string } }>;
```

Behavior

- `fetch(env.SNOWFLAKE_ENDPOINT, { method: "POST", headers: { "content-type": "application/json", ...(env.SNOWFLAKE_API_KEY ? {"x-api-key": env.SNOWFLAKE_API_KEY} : {}) }, body: JSON.stringify({ prompt, agent_type: agentType, seed }), signal })`
- If `!res.ok` → `{ error: { code: "GENERATION_FAILED", message: "upstream <status>" } }`
- Parse JSON and assert the presence of a **top-level `model` array** (no other shapes allowed).
- Return `{ model }`
- Timeout: use `AbortController` with `Number(env.SNOWFLAKE_TIMEOUT_MS)`

3.6 Utilities (inline in `index.ts`)

- `makeRequestId(): string` → `new Date().toISOString() + "_" + Math.random().toString(36).slice(2,8)`
- `sha256Hex(input: string): Promise<string>` → Web Crypto
- `designKey(prompt: string, agentType: string)` → `sha256Hex(prompt.trim().toLowerCase() + "|" + agentType)`

- `ok(data, request_id)` → success Response with CORS + JSON
- `err(code, message, details, request_id, status)` → error Response with CORS + JSON
- `limitBodySize(request, maxBytes)` → reject oversized bodies
- `rateLimit(env, ip, windowS, max)` → KV counter; return boolean allow/deny

I/O quick reference (unchanged)

`POST /api/design` request

```
{ "prompt": "space rover", "agent_types": ["A","B"] }
```

`POST /api/design` response (success)

```
{
  "designs": {
    "A":
[{"ldraw_part":"3001.dat","color":16,"pos":[0,24,0],"rot":[0,0,0]}],
    "B":
[{"ldraw_part":"3020.dat","color":14,"pos":[-20,24,0],"rot":[0,0,0]}]
  },
  "cache": { "A": true, "B": false },
  "request_id": "..."
}
```

`POST /api/vote` request

```
{ "agent_type": "B", "prompt": "space rover" }
```

`POST /api/vote` response

```
{ "leaderboard": { "A": 21, "B": 22 }, "request_id": "..." }
```

`GET /api/leaderboard` response

```
{ "leaderboard": { "A": 21, "B": 22 }, "updated_at": "ISO",
"request_id": "..." }
```

Error envelope (any endpoint)

```
{ "error": { "code": "SOMETHING_BAD", "message": "…", "details": {} },
"request_id": "..." }
```

Test plan (compact)

- **Cold design**: KV miss → D1 miss → Snowflake → validate → D1 insert + KV put → `{cache:false}`
- **Warm design**: KV hit → `{cache:true}`
- **Validation**: bad `rot`/oversize → `400 VALIDATION_FAILED`
- **Rate limit**: >10 requests/min/IP to `/api/design` → `429 RATE_LIMITED`
- **Votes**: post then check leaderboard; persists across deploys (D1)

Definition of done

- Single TypeScript Worker handles all 3 endpoints.
- KV + D1 wired; cache hits observable in logs.
- Validation & error envelopes enforced.
- Rate limiting + CORS active.
- Snowflake HTTPS call integrated with 5s timeout.
- Smoke tests via `curl` succeed.