DJZS Anytype Profile JSON Schema + README

File 1 — `anytype-userprofile.schema.json`

This is a JSON Schema (Draft-07) for storing profile data inside Anytype as a `UserProfile` object type. It balances a local-first model (private fields) with public metadata (IRYS receipts, ENS, NFTs).

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "DJZS Anytype UserProfile",
  "description": "Schema for the DJZS UserProfile object stored in Anytype.
Supports local/private fields, public IRYS references, ENS identity and NFT
links.",
  "type": "object",
  "required": ["id","username","createdAt"],
  "properties": {
    "id": { "type": "string", "description": "UUID or Anytype object id" },
    "username": { "type": "string", "description": "Display username (e.g.
Damon)" },
    "displayName": { "type": "string" },
    "bio": { "type": "string" },
    "avatar": { "type": "object", "properties": { "type": { "type": "string" },
"url": { "type": "string", "format": "uri" } }, "additionalProperties": false },
    "ens": { "type": "string", "description": "ENS subname (e.g.
username.djzs.eth)" },
    "ethAddress": { "type": "string", "description": "Ethereum address
(0x...)" },
    "links": { "type": "array", "items": { "type": "object", "properties": {
"label": { "type": "string" }, "url": { "type": "string", "format": "uri" } },
"required": ["label","url"], "additionalProperties": false } },
    "zones": {
      "type": "array",
      "description": "High-level counts and metadata for each DJZS zone",
      "items": {
        "type": "object",
        "properties": {
          "zoneId": { "type": "string" },
          "title": { "type": "string" },
          "entryCount": { "type": "integer", "minimum": 0 },
          "publicCount": { "type": "integer", "minimum": 0 },
          "lastUpdatedAt": { "type": "string", "format": "date-time" }
        },
        "required": ["zoneId","title"]
        }
      },
      "journals": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "id": { "type": "string" },
            "title": { "type": "string" },
            "summary": { "type": "string" },
            "zoneId": { "type": "string" },
            "isPublic": { "type": "boolean" },
            "irys": {
              "type": "object",
              "properties": {
                "hash": { "type": "string" },
                "url": { "type": "string", "format": "uri" },
                "publishedAt": { "type": "string", "format": "date-time" }
              },
              "additionalProperties": false
            },
            "mint": {
              "type": "object",
              "properties": {
                "minted": { "type": "boolean" },
                "tokenId": { "type": ["string","integer"] },
                "contract": { "type": "string" },
                "chain": { "type": "string" }
              },
              "additionalProperties": false
            },
            "createdAt": { "type": "string", "format": "date-time" },
            "updatedAt": { "type": "string", "format": "date-time" }
          },
          "required": ["id","title","createdAt"]
        }
      },
      "agent": {
        "type": "object",
        "properties": {
          "name": { "type": "string" },
          "ens": { "type": "string" },
          "xp": { "type": "number" },
          "level": { "type": "integer" },
          "mood": { "type": "string" },
          "lastTrainedAt": { "type": "string", "format": "date-time" },
          "modelRef": { "type": "string", "description": "Reference to agent
weights or model snapshot (if published)" }
```

```
      },
      "additionalProperties": false
    },
    "achievements": { "type": "array", "items": { "type": "object",
"properties": { "id": { "type": "string" }, "title": { "type": "string" },
"description": { "type": "string" }, "awardedAt": { "type": "string", "format":
"date-time" } }, "required": ["id","title"] } },
    "nfts": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "tokenId": { "type": ["string","integer"] },
          "contract": { "type": "string" },
          "chain": { "type": "string" },
          "metadataUrl": { "type": "string", "format": "uri" },
          "mintedAt": { "type": "string", "format": "date-time" }
        },
        "required": ["tokenId","contract"]
      }
    },
    "irysReceipts": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "hash": { "type": "string" },
          "url": { "type": "string", "format": "uri" },
          "type": { "type": "string", "description": "e.g. profile, journal,
research" },
          "publishedAt": { "type": "string", "format": "date-time" }
        },
        "required": ["hash","url"]
      }
    },
    "publicProfile": { "type": "boolean", "description": "Whether the user has
published a public profile via IRYS" },
    "createdAt": { "type": "string", "format": "date-time" },
    "updatedAt": { "type": "string", "format": "date-time" },
    "version": { "type": "string", "description": "Schema/version string" }
  },
  "additionalProperties": false
}
```

File 2 — `README.md` (copy into your Replit project root)

```
# DJZS — Anytype UserProfile Schema

This README explains how to use `anytype-userprofile.schema.json` inside your
DJZS Replit project and how the schema maps to Anytype object types, MCP, and
the IRYS publishing flow.

## Purpose

The schema defines a `UserProfile` object for Anytype that stores both private/
local fields (agent memory, drafts) and public metadata (IRYS receipts, ENS
identity, NFTs). The design is: **Local-first by default — publish selectively
to IRYS when the user chooses.**

## Files

- `anytype-userprofile.schema.json` — JSON Schema (Draft-07)
- `README.md` — this document

## How to use

### 1. Add to your Replit project
Copy both files into the root of your Replit project. Use the schema to validate
payloads from your backend or MCP tools before writing to Anytype.

### 2. Anytype object type mapping
In Anytype, create a new Object Type `UserProfile` with fields corresponding to
the schema. Recommended field types:

- `id` — Text (unique)
- `username` — Text
- `displayName` — Text
- `bio` — Long Text
- `avatar` — File or URL
- `ens` — Text
- `ethAddress` — Text
- `links` — Relation to `Link` objects or JSON field
- `zones` — Relation to `Zone` objects (with counters)
- `journals` — Relation to `Journal` objects (store full entries elsewhere)
- `agent` — Relation to `Agent` object or nested JSON
- `achievements` — Relation or JSON
- `nfts` — Relation to `NFT` objects
- `irysReceipts` — Relation to `IRYSReceipt` objects

> Note: Prefer relations in Anytype for rich querying. Store bulky content in
```

~~~
Journal objects and keep references in the profile.

### 3. MCP agent read/write patterns
Your MCP tool should:

- Read `UserProfile` from Anytype Vault when the agent starts.
- Update `agent.lastTrainedAt`, `agent.xp`, and `journals` summaries after
training.
- Prepare `irys` bundle when the user clicks `Publish`.
- Store IRYS receipt back into `irysReceipts` and set `publicProfile: true`
where applicable.

### 4. IRYS publishing flow (recommended)

1. MCP builds a publish bundle (profile metadata + selected journal summaries +
chosen assets).
2. Upload bundle to IRYS and receive `hash` + `url` + `receipt`.
3. Store `irysReceipts[]` entry in the Anytype `UserProfile` object with `type:
"profile"`.
4. If minting is desired, call your minting service (wagmi/wallet connect) with
`metadata: { irysUrl, irysHash }`.

### 5. Minting tips (Ethereum via Wagmi)
Use the IRYS `url` as the canonical `metadata.uri` of your token. Example
metadata JSON structure for minting:

```json
{
  "name": "DJZS Journal — 2025-11-24",
  "description": "Immutable DJZS journal published by username.djzs.eth",
  "external_url": "https://ea5d0ede-
c0b4-4e6e-8aab-622338eae763-00-1hla9paj6syzi.kirk.replit.dev/profile",
  "irys": {
    "hash": "<IRYS_HASH>",
    "url": "<IRYS_URL>"
  },
  "attributes": [
    { "trait_type": "Zone", "value": "Research" }
  ]
}
~~~

6. Example: Writing a new journal via Replit backend

1. Frontend posts journal to Replit backend.
2. Backend validates payload against `anytype-userprofile.schema.json` (the `journals` item structure).
3. Backend writes journal to Anytype Vault via MCP tool / Anytype SDK.

4. Agent optionally trains on journal and updates `agent.xp` and `lastTrainedAt`.

Versioning & schema upgrades

- Bump `version` field when you change the schema. Keep migration scripts in `/migrations` to upgrade older UserProfile objects.

Security & Privacy

- Sensitive data (private journals, agent internal state) should be kept local in Anytype and **not** included in IRYS bundles unless intentionally published.
- Sign IRYS bundles with the user's ENS-linked key to prove authorship.

Next Steps — Automation ideas

- Add a `publishIntent` field (timestamp + checklist) that MCP reads to determine whether to auto-publish.
- Add a `visibility` field per journal (private / followers / public) and use MCP to batch publish followable updates.
- Add an `auditTrail` relation to store revision IDs for each change.

Contact

If you want, I can also produce:

- TypeScript interfaces derived from this schema
- A migration script to map existing Replit profile shape to the new schema
- Sample MCP tool code (Node + Anytype SDK) to read/write the `UserProfile`

~~~

~~~

```

---

*End of document.*
```