🚀Spectre Guild Platform — README

A fresh, opinionated blueprint to buikd rhw Star Citizen guild site with **clarity, fewer bugs, and a straight path to MVP**. Keep this README as the single source of truth.

---

0) Goals & Guardrails

- **Three roles**: `GUEST` (unauthenticated), `USER` (default after login), `MEMBER` (org-verified), `ADMIN` (full control).
- **MVP features**:
- Auth via **Discord OAuth2 ➜ JWT** (+ refresh tokens).
- Public read-only pages + **member-only tools**.
- **Posts/Blog** (create/edit/delete by MEMBER; ADMIN can moderate all).
- **Image Gallery** (upload/delete own; ADMIN full moderation).
- **Ship data & Tools** (comparison, commodities, routes) behind role checks.
- **Non-goals (for MVP)**: fancy admin dashboard (add later), complicated CMS, multi-tenant.
- **Simplicity first**: small modules, strict linting, defensive error handling, consistent naming.

---

1) Architecture Overview

```
monorepo/
├─ backend/ (Spring Boot 3, Java 21)
│  ├─ src/main/java/com/spectre/
│  │  ├─ controller/            # REST controllers
│  │  ├─ service/               # business logic
│  │  ├─ repository/            # JPA repos
│  │  ├─ model/                 # JPA entities
│  │  ├─ security/              # OAuth2, JWT, filters
│  │  ├─ dto/                   # request/response payloads
│  │  └─ config/                # CORS, rate limiting, cache
│  ├─ src/main/resources/
│  │  └─ application.yml
│  └─ pom.xml
│
├─ frontend/ (React 19 + Vite + TS)
│  ├─ src/
│  │  ├─ app/                   # router, providers
│  │  ├─ components/
│  │  ├─ pages/                 # Login, Home, Tools, Gallery, Posts, Admin
│  │  ├─ features/              # posts, gallery, auth, tools
│  │  ├─ api/                   # axios client, endpoints
│  │  ├─ hooks/                 # useAuth, useRole, useQuery
│  │  └─ store/                 # Context or Zustand
│  └─ index.html
│
```

```
├─ docker/
│  ├─ docker-compose.yml        # mysql + backend + frontend
│  └─ nginx.conf                # prod reverse proxy (later)
│
└─ README.md (this file)
```

Data Flow (happy path)

1. User clicks **Login with Discord** → backend OAuth2 callback → create/find `User` → issue **JWT** (+ Refresh).
2. Frontend stores **access token** in memory (and refresh token in HttpOnly cookie) → attaches via interceptor.
3. Protected API routes validate JWT → apply `@PreAuthorize` based on roles.
4. Tools fetch/calc data. External APIs cached in DB or in-memory to avoid rate limits.

---

2) Role & Access Matrix (MVP)

Feature / Route	GUEST	USER	MEMBER	ADMIN
View posts	✅	✅	✅	✅
Create/edit/delete own posts	❌	❌	✅	✅
Moderate all posts	❌	❌	❌	✅
View gallery	✅	✅	✅	✅
Upload/delete own images	❌	✅	✅	✅
Moderate all images	❌	❌	❌	✅
Ship tools (info, compare)	❌	✅	✅	✅
Member-only tools (trade, etc)	❌	❌	✅	✅
User management	❌	❌	❌	✅

Implementation hint: `@PreAuthorize("hasRole('MEMBER')")` etc. Keep roles as `ROLE_USER`, `ROLE_MEMBER`, `ROLE_ADMIN` in DB; map to simpler flags on frontend.

---

3) Backend — Spring Boot

3.1 Dependencies (pom.xml key ones)

- Spring Web, Spring Security, Spring Data JPA
- OAuth2 Client
- MySQL + HikariCP
- jjwt (or io.jsonwebtoken via Spring), MapStruct (DTOs), Validation API
- Bucket4j (rate limiting), Caffeine or Spring Cache
- Lombok, Testcontainers (tests)

3.2 Configuration (`application.yml`)

```
server:
  port: 8080
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/website?
useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC
    username: root
    password: root
    hikari:
      maximum-pool-size: 10
  jpa:
    hibernate:
      ddl-auto: update   # prod: validate
    properties:
      hibernate.dialect: org.hibernate.dialect.MySQL8Dialect
  cache:
    type: caffeine
security:
  cors:
    allowed-origins: [http://localhost:5173]
    allowed-methods: [GET,POST,PUT,DELETE]
    allowed-headers: [Authorization,Content-Type]
    allow-credentials: true
app:
  jwt:
    secret: ${JWT_SECRET}
    expiration: 900000       # 15 min
    refresh-expiration: 604800000 # 7 days
  discord:
    client-id: ${DISCORD_CLIENT_ID}
    client-secret: ${DISCORD_CLIENT_SECRET}
    redirect-uri: http://localhost:8080/api/auth/discord/callback
    guild-id: ${DISCORD_GUILD_ID}
uex:
  api-key: ${UEX_API_KEY}
```

3.3 Entities (minimal)

- `User(id, discordId, username, avatar, roles, createdAt)`
- `Role(id, name)` with values: `ROLE_USER`, `ROLE_MEMBER`, `ROLE_ADMIN`
- `RefreshToken(id, userId, token, expiry)`
- `Post(id, userId, title, content, createdAt, updatedAt)`
- `Image(id, userId, title, content[LONGBLOB], uploadedAt)`
- `Ship(id, uuid, name, manufacturer, size, cargoCapacity, speed, pitch, yaw, roll, hp, updatedAt, image[BLOB])`
- `CommodityPrice(id, commodity, location, buy, sell, timestamp)`

3.4 Security Pipeline

- **/api/auth** routes: allow anonymous
- **/api/public** routes: allow anonymous
- **/api/** routes: require JWT
- `JwtAuthFilter` reads `Authorization: Bearer <token>`
- `@PreAuthorize` on controller methods for role checks

3.5 OAuth2 + JWT Flow (backend-only)

1. `/api/auth/discord/login` → redirect to Discord OAuth2
2. `/api/auth/discord/callback` → exchange code → fetch user + guild membership
3. Create/update `User` + infer role `ROLE_MEMBER` if in guild; default `ROLE_USER`
4. Issue Access & Refresh tokens (Return: access in body, refresh as **HttpOnly cookie**)
5. `/api/auth/refresh` issues new access token if refresh is valid
6. `/api/auth/logout` invalidates refresh token (delete DB entry)

3.6 API Surface (MVP)

Auth

- `GET /api/auth/discord/login` → 302
- `GET /api/auth/discord/callback` → 200 {accessToken}
- `POST /api/auth/refresh` → 200 {accessToken}
- `POST /api/auth/logout` → 204

Users

- `GET /api/me` → current profile + roles

Posts

- `GET /api/posts` (public)
- `POST /api/posts` (MEMBER+)
- `PUT /api/posts/{id}` (owner or ADMIN)
- `DELETE /api/posts/{id}` (owner or ADMIN)

Gallery

- `GET /api/images` (public)
- `POST /api/images` (USER+)
- `DELETE /api/images/{id}` (owner or ADMIN)
- `GET /api/images/{id}/raw` (public stream)

Ships & Tools

- `GET /api/ships/names` (USER+)
- `GET /api/ships/info?name=...` (USER+)
- `POST /api/tools/compare` (USER+)
- `GET /api/tools/commodities` (MEMBER+; cached)
- `POST /api/tools/routes` (MEMBER+)

3.7 Rate Limiting & Caching

- **Bucket4j** per IP for `/api/auth/*` (strict) and generic for others.
- **Caffeine** in-memory cache for hot reads (e.g., `/api/ships/names`).
- Persist external API snapshots (UEX, Wiki) in DB; refresh via **scheduled job**.

3.8 Error Handling

- `@ControllerAdvice` with `GlobalExceptionHandler` → normalized JSON error envelope:

```
{ "timestamp": "...", "status": 400, "error": "Bad Request", "message":
"...", "path": "/api/..." }
```

- Log correlation ID (MDC) on each request; return `x-request-id` header.

---

4) Frontend — React 19 + Vite + TS

4.1 Key Choices

- Router with protected routes (guards by role)
- **AuthContext** stores access token in memory; refresh via HttpOnly cookie
- Axios interceptor adds `Authorization` header
- UI lib optional (shadcn/ui) + Recharts for charts

4.2 Routes (MVP)

- `/` (public Home)
- `/login` → kicks off OAuth2 (redirect)
- `/posts` (public list)
- `/posts/new` (MEMBER+)
- `/gallery` (public list)
- `/tools/ships` (USER+)
- `/tools/trade` (MEMBER+)
- `/admin` (ADMIN only; simple table views)

4.3 Minimal AuthContext (pseudo-code)

```
const AuthContext = createContext(...);
// store access in memory; rely on refresh cookie
// on 401: call /api/auth/refresh, retry once
```

4.4 UI/UX Guidelines

- Responsive grid, clear nav with role-based items
- Toasts for success/error
- Disable buttons on pending, show skeletons

---

5) External APIs Strategy

5.1 Star Citizen Wiki API

- Ingest by UUID; store normalized `Ship`
- Nightly sync + manual sync endpoint (ADMIN)

5.2 UEX Commodities API

- Hourly fetch to `CommodityPrice` (keep latest per (commodity, location))
- Tools use DB snapshot; never block on live calls

5.3 Discord API

- After OAuth2, verify guild membership → grant `ROLE_MEMBER`
- Re-check membership on login & daily scheduled job; downgrade if left

---

6) Local Development

6.1 Prereqs

- Java 21, Node 20, Docker, MySQL 8 (or via docker-compose)

6.2 Environment

Create `.env` in repo root:

```
JWT_SECRET=dev-super-secret
DISCORD_CLIENT_ID=...
DISCORD_CLIENT_SECRET=...
DISCORD_GUILD_ID=...
UEX_API_KEY=...
```

6.3 Start Everything

```
# DB + services
docker compose -f docker/docker-compose.yml up -d

# backend
cd backend && mvn spring-boot:run

# frontend
cd frontend && npm i && npm run dev
```

---

7) Testing & Quality

- **Unit**: services + utils; mock repos
- **WebMvcTest**: controllers with security
- **Testcontainers**: MySQL integration tests
- **ESLint/Prettier** + **Checkstyle/Spotless**
- **CI**: build, tests, lint on PR

---

8) Security Best Practices (MVP)

- HttpOnly, Secure cookies for refresh token
- Access token in memory only (not localStorage)
- Strict CORS (exact origins), CSRF disabled for APIs
- Validate uploads (size/type) and store as compressed JPEG/WebP
- Limit body size, sanitize inputs (server-side validation)

---

9) Deployment (later)

- Docker images for backend & frontend
- Nginx reverse proxy with TLS
- Externalized configs via env vars
- Fly.io/Render/VPS acceptable; attach managed MySQL if possible

---

10) Step-by-Step Build Plan (Do-this-Next)

Day 1: Auth core

1. Scaffold backend, add security config, JWT service, filters
2. Implement Discord OAuth2 login & callback
3. Implement refresh token flow + `/api/me`

Day 2: Roles & Posts 4. Entities & repos for `Post`, `Image` 5. CRUD for posts with ownership checks + tests

Day 3: Gallery & Uploads 6. Image upload (size/type limits), list, delete own; stream raw endpoint

Day 4: Ships & Tools 7. Ship ingest service (Wiki API), `/ships/names` & `/ships/info` 8. Trading endpoints using cached UEX data

Day 5: Frontend MVP 9. AuthContext + protected routes + role menu 10. Pages: Posts, Gallery, Ships, Trade

Hardening 11. Add rate limiting, caching, error handler, logs, request-id

---

11) Minimal Endpoint Contract (for frontend wiring)

```
GET /api/me                          -> 200 { id, username, roles }
POST /api/auth/refresh               -> 200 { accessToken }
POST /api/posts                      -> 201 { id }
GET  /api/posts                      -> 200 [ ... ]
PUT  /api/posts/{id}                 -> 200
DELETE /api/posts/{id}               -> 204
GET  /api/images                     -> 200 [ {id,title,url} ]
POST /api/images (multipart/form)    -> 201 { id }
DELETE /api/images/{id}              -> 204
GET  /api/images/{id}/raw            -> 200 (image stream)
GET  /api/ships/names                -> 200 ["Carrack", ...]
GET  /api/ships/info?name=Carrack    -> 200 { ...specs }
POST /api/tools/compare              -> 200 { diff, radarData }
GET  /api/tools/commodities          -> 200 { items: [...] }
POST /api/tools/routes               -> 200 { bestRoute: ... }
```

---

12) Admin (thin slice for now)

- `/admin/users` list + change role (ADMIN)
- `/admin/sync/ships` trigger re-sync (ADMIN)
- `/admin/mod/*` basic moderation endpoints

---

13) Troubleshooting Checklist

- 401 on API → check `Authorization` header, refresh flow, CORS
- OAuth2 redirect loop → redirect URI mismatch in Discord application
- Image upload fails → increase max-file-size; verify multipart config
- Ship names empty → run sync; verify API key/rate limit; check logs

---

14) License & Credits

Internal guild project. Star Citizen data courtesy of community APIs.
