# Spotify Genre Sorter API Documentation

## Overview

The Spotify Genre Sorter API provides endpoints for authenticating users, analysing their Spotify library genres, and creating genre-based playlists.

**Base URL:** `https://your-worker.workers.dev`

## Authentication

The API uses two authentication modes:

### Spotify-Only Mode (Default)

Users authenticate directly with Spotify OAuth. Enabled when `SPOTIFY_ONLY_AUTH=true` or GitHub credentials are not configured.

### GitHub + Spotify Mode

Users first authenticate with GitHub (for access control), then connect their Spotify account.

## Rate Limiting

All `/api/*` endpoints are rate-limited:

- **30 requests per minute** per IP address
- Returns `429 Too Many Requests` when exceeded
- `Retry-After` header indicates when to retry

---

## Public Endpoints

### GET `/`

Returns the main application UI (HTML).

### GET `/health`

Health check endpoint.

**Response:**

```json
{
  "status": "ok"
}
```

### GET `/setup`

Check if required secrets are configured.

**Response:**

```json
{
  "configured": true,
  "spotifyOnly": true,
  "hasGitHub": false,
  "hasSpotify": true
}
```

### GET `/session`

Check current session status.

**Response (authenticated):**

```json
{
  "authenticated": true,
  "user": {
    "id": "spotify_user_id",
    "display_name": "User Name"
  }
}
```

**Response (not authenticated):**

```json
{
  "authenticated": false
}
```

### GET `/stats`

Get application statistics and Hall of Fame.

**Response:**

```json
{
  "userCount": 42,
  "hallOfFame": [
    {
      "position": 1,
      "spotifyName": "Early Adopter",
      "registeredAt": "2025-01-01T00:00:00Z"
    }
  ]
}
```

---

## Authentication Endpoints

### GET `/auth/github`

Initiates GitHub OAuth flow.

**Redirects to:** GitHub authorisation page

**Notes:**

- Only available in GitHub + Spotify mode
- Redirects to `/auth/spotify` if in Spotify-only mode

### GET `/auth/github/callback`

GitHub OAuth callback handler.

**Query Parameters:**

| Parameter | Type | Description |
|---|---|---|
| `code` | string | OAuth authorisation code |
| `state` | string | CSRF state token |

**Redirects to:** `/` on success, `/?error=<code>` on failure

**Error Codes:**

- `github_denied` - User denied access
- `invalid_request` - Missing code/state
- `invalid_state` - State verification failed
- `not_allowed` - User not in allowlist
- `auth_failed` - OAuth exchange failed

### GET `/auth/spotify`

Initiates Spotify OAuth flow.

**Redirects to:** Spotify authorisation page

**Required Scopes:**

- `user-library-read` - Read liked songs
- `playlist-modify-public` - Create public playlists
- `playlist-modify-private` - Create private playlists
- `user-read-private` - Read user profile

### GET `/auth/spotify/callback`

Spotify OAuth callback handler.

**Query Parameters:**

| Parameter | Type | Description |
|---|---|---|
| `code` | string | OAuth authorisation code |
| `state` | string | CSRF state token |

**Redirects to:** `/` on success, `/?error=<code>` on failure

**Error Codes:**

- `spotify_denied` - User denied access
- `invalid_request` - Missing code/state
- `invalid_state` - State verification failed
- `not_logged_in` - GitHub session required (non-Spotify-only mode)
- `spotify_auth_failed` - OAuth exchange failed

### GET `/auth/logout`

Clears the current session and logs out.

**Redirects to:** `/`

---

## API Endpoints (Authenticated)

All `/api/*` endpoints require a valid Spotify session. Returns `401 Unauthorized` if not authenticated.

### GET `/api/me`

Get current user information.

**Response:**

```json
{
  "github": {
    "username": "octocat",
    "avatar": "https://github.com/octocat.png"
  },
  "spotify": {
    "id": "spotify_user_id",
    "name": "Display Name",
    "avatar": "https://i.scdn.co/image/..."
  }
}
```

### GET `/api/genres`

Get all genres from the user's liked tracks.

**Query Parameters:**

| Parameter | Type | Description |
|---|---|---|
| `refresh` | boolean | Force cache refresh (default: false) |

**Response:**

```json
{
  "totalTracks": 500,
  "totalGenres": 85,
  "totalArtists": 200,
  "genres": [
    {
      "name": "rock",
      "count": 150,
      "trackIds": ["abc123...", "def456..."]
    },
    {
      "name": "pop",
      "count": 100,
      "trackIds": ["ghi789..."]
    }
  ],
  "cachedAt": 1699876543210,
  "fromCache": false,
  "truncated": false,
  "totalInLibrary": 500
}
```

**Notes:**

- Results are cached in KV for 1 hour
- Library is limited to 1000 tracks on free tier to avoid subrequest limits
- `truncated: true` indicates more tracks exist than were processed
- Genre order is by track count (descending)

### GET `/api/genres/chunk`

Progressive loading for large libraries. Fetches genres in chunks to stay under Cloudflare's 50 subrequest limit.

**Query Parameters:**

| Parameter | Type | Default | Description |
|---|---|---|---|
| `offset` | number | 0 | Starting track offset |
| `limit` | number | 500 | Number of tracks per chunk (max 500) |

**Response:**

```
{
  "chunk": {
    "genres": [
      { "name": "rock", "count": 75, "trackIds": ["..."] }
    ],
    "trackCount": 500,
    "artistCount": 150,
    "cachedAt": 1699876543210
  },
  "pagination": {
    "offset": 0,
    "limit": 500,
    "hasMore": true,
    "nextOffset": 500,
    "totalInLibrary": 2500
  },
  "progress": 20,
  "fromCache": false
}
```

**Usage Example:**

```
let allGenres = [];
let offset = 0;
const limit = 500;
let hasMore = true;

while (hasMore) {
  const response = await fetch(`/api/genres/chunk?offset=${offset}&limit=${limit}`);
  const data = await response.json();

  allGenres = mergeGenres(allGenres, data.chunk.genres);
  hasMore = data.pagination.hasMore;
  offset = data.pagination.nextOffset;

  updateProgressBar(data.progress);
}
```

### POST `/api/playlist`

Create a playlist for a specific genre.

**Request Body:**

```
{
  "genre": "rock",
  "trackIds": ["abc123", "def456", "ghi789"],
  "force": false
}
```

| Field | Type | Required | Description |
|---|---|---|---|
| `genre` | string | Yes | Genre name (max 100 chars) |
| `trackIds` | string[] | Yes | Spotify track IDs (max 10,000) |
| `force` | boolean | No | Create even if duplicate exists |

**Response (success):**

```
{
  "success": true,
  "playlist": {
    "id": "playlist_id",
    "url": "https://open.spotify.com/playlist/...",
    "name": "rock (from Likes)",
    "trackCount": 150
  }
}
```

**Response (duplicate found, force=false):**

```json
{
  "success": false,
  "duplicate": true,
  "existingPlaylist": {
    "id": "existing_playlist_id",
    "name": "rock (from Likes)",
    "trackCount": 100
  },
  "message": "A playlist named \"rock (from Likes)\" already exists with 100 tracks"
}
```

**Errors:**

- `400` - Invalid genre name or track IDs
- `401` - Not authenticated
- `500` - Spotify API error

### POST `/api/playlists/bulk`

Create playlists for multiple genres at once.

**Request Body:**

```json
{
  "genres": [
    { "name": "rock", "trackIds": ["abc123", "def456"] },
    { "name": "pop", "trackIds": ["ghi789"] }
  ],
  "skipDuplicates": true
}
```

| Field | Type | Required | Description |
|---|---|---|---|
| `genres` | array | Yes | Array of genres (max 50) |
| `genres[].name` | string | Yes | Genre name |
| `genres[].trackIds` | string[] | Yes | Track IDs for this genre |
| `skipDuplicates` | boolean | No | Skip genres with existing playlists |

**Response:**

```json
{
  "total": 3,
  "successful": 2,
  "skipped": 1,
  "results": [
    { "genre": "rock", "success": true, "url": "https://open.spotify.com/playlist/..." },
    { "genre": "pop", "success": true, "url": "https://open.spotify.com/playlist/..." },
    { "genre": "jazz", "success": false, "skipped": true, "error": "Playlist already exists"
}
  ]
}
```

## Error Responses

All error responses follow this format:

```
{
  "error": "Human-readable error message",
  "details": "Optional technical details",
  "step": "Optional step where error occurred"
}
```

### HTTP Status Codes

| Code | Description |
|---|---|
| `200` | Success |
| `400` | Bad Request - Invalid parameters |
| `401` | Unauthorized - Authentication required |
| `429` | Too Many Requests - Rate limited |
| `500` | Internal Server Error |

## Data Validation

### Track IDs

- Must match pattern: `/^[a-zA-Z0-9]{22}$/`
- Maximum 10,000 per request

### Genre Names

- Maximum 100 characters
- Dangerous characters ( `<` , `>` , `"` , `'` , `&` ) are stripped

## Caching

| Data | TTL | Cache Key Format |
|---|---|---|
| Genre data | 1 hour | `genre_cache_{userId}` |
| Genre chunks | 1 hour | `genre_chunk_{userId}_{offset}_{limit}` |

|  |  |  |
|---|---|---|
| Sessions | 7 days | `session_{sessionId}` |
| OAuth state | 10 minutes | `state_{stateToken}` |

Cache is invalidated automatically when:

- A new playlist is created
- User explicitly requests refresh ( `?refresh=true` )

---

## Examples

### Fetch genres and create a playlist

```
// 1. Get all genres
const genresRes = await fetch('/api/genres');
const { genres } = await genresRes.json();

// 2. Find rock tracks
const rockGenre = genres.find(g => g.name === 'rock');

// 3. Create playlist
const createRes = await fetch('/api/playlist', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    genre: 'rock',
    trackIds: rockGenre.trackIds
  })
});

const { playlist } = await createRes.json();
console.log(`Created: ${playlist.url}`);
```

### Handle large library with progressive loading

```
async function loadAllGenres(onProgress) {
  const mergedGenres = new Map();
  let offset = 0;
  let hasMore = true;

  while (hasMore) {
    const res = await fetch(`/api/genres/chunk?offset=${offset}&limit=500`);
    const data = await res.json();

    // Merge genres from this chunk
    for (const genre of data.chunk.genres) {
      const existing = mergedGenres.get(genre.name);
      if (existing) {
        existing.count += genre.count;
```

```
          existing.trackIds.push(...genre.trackIds);
        } else {
          mergedGenres.set(genre.name, { ...genre });
        }
      }

      onProgress?.(data.progress);
      hasMore = data.pagination.hasMore;
      offset = data.pagination.nextOffset ?? offset + 500;
    }

    return Array.from(mergedGenres.values())
      .sort((a, b) => b.count - a.count);
  }
```

---

*Last updated: December 2025*