Quick Start Guide - Spotify Follow-Swarm

🚀 Get Started in 15 Minutes

Prerequisites

- Node.js 18+ or Python 3.9+
- PostgreSQL 14+
- Redis 7+
- Spotify Developer Account
- Git

1. Clone and Setup (2 minutes)

```bash
# Clone the repository
git clone https://github.com/yourusername/spotify-follow-swarm.git
cd spotify-follow-swarm

# Install dependencies
npm install  # or pip install -r requirements.txt

# Copy environment template
cp .env.example .env
```

2. Spotify App Setup (3 minutes)

1. Go to Spotify Developer Dashboard
2. Click "Create App"
3. Name: "Spotify Follow Swarm"
4. Redirect URI: `http://localhost:3000/auth/callback`
5. Save Client ID and Secret to `.env`

3. Database Setup (2 minutes)

```bash
```

```
# Create database
createdb spotify_swarm

# Run migrations
npm run migrate  # or python manage.py migrate

# Seed test data (optional)
npm run seed  # or python manage.py seed
```

4. Configure Environment (2 minutes)

Edit `.env` file:

```env
# Required
SPOTIFY_CLIENT_ID=your_client_id_here
SPOTIFY_CLIENT_SECRET=your_client_secret_here
DATABASE_URL=postgresql://localhost/spotify_swarm
REDIS_URL=redis://localhost:6379

# Security (generate random strings)
JWT_SECRET=generate_random_string_here
ENCRYPTION_KEY=another_random_string_here

# Optional
PORT=3001
NODE_ENV=development
```

5. Start Development (1 minute)

```bash
# Using Docker (recommended)
docker-compose up

# Or manually in separate terminals:
npm run dev:api      # Start API server
npm run dev:worker   # Start queue worker
npm run dev:frontend # Start React app
```

6. Test the Application (5 minutes)

1. Open http://localhost:3000
2. Click "Connect with Spotify"
3. Authorize the application
4. Watch the magic happen! ✨

🎯 Core Functionality Test

Manual API Test

```bash
# Test health endpoint
curl http://localhost:3001/health

# Test OAuth flow (open in browser)
open http://localhost:3001/auth/spotify

# Test follow sync (requires auth token)
curl -X POST http://localhost:3001/api/follows/sync \
  -H "Authorization: Bearer YOUR_TOKEN"
```

Quick Database Check

```sql
-- Check users
SELECT * FROM users;

-- Check follow queue
SELECT status, COUNT(*)
FROM follows
GROUP BY status;

-- Check recent activity
SELECT * FROM analytics
ORDER BY created_at DESC
LIMIT 10;
```

🔧 Common Issues & Solutions

Issue: "Spotify API returns 401"

Solution: Token expired. Implement token refresh:

```javascript
if (error.statusCode === 401) {
  await refreshUserToken(userId);
  // Retry the request
}
```

Issue: "Database connection failed"

Solution: Check PostgreSQL is running:

```bash
# macOS
brew services start postgresql

# Linux
sudo systemctl start postgresql

# Docker
docker-compose up postgres
```

Issue: "Redis connection refused"

Solution: Start Redis:

```bash
# macOS
brew services start redis

# Linux
sudo systemctl start redis

# Docker
docker-compose up redis
```

Issue: "Rate limit exceeded"

Solution: Adjust throttling in config:

```javascript
// config/limits.js
module.exports = {
  maxFollowsPerHour: 20,  // Reduce this
  delayBetweenFollows: 180000  // Increase this (3 minutes)
};
```

📊 Monitoring Your Development

Simple Logging

```javascript
// Add to your follow engine
console.log(`[${new Date().toISOString()}] Processing user ${userId}`);
console.log(`[${new Date().toISOString()}] Followed ${count} artists`);
```

Basic Metrics

```javascript
// Track in Redis
await redis.incr('follows:total');
await redis.incr(`follows:${userId}:today`);
await redis.expire(`follows:${userId}:today`, 86400);
```

Development Dashboard

Create a simple dashboard at `/admin`:

```javascript
```

```
app.get('/admin', async (req, res) => {
  const stats = {
    totalUsers: await db.query('SELECT COUNT(*) FROM users'),
    totalFollows: await db.query('SELECT COUNT(*) FROM follows WHERE status = completed'),
    queueSize: await redis.llen('follow-queue'),
    todayFollows: await redis.get('follows:today') || 0
  };
  res.json(stats);
});
```

🚢 Deploy to Production (Quick)

Using Heroku (Fastest)

bash

```
# Install Heroku CLI
heroku create spotify-follow-swarm

# Add database
heroku addons:create heroku-postgresql:hobby-dev
heroku addons:create heroku-redis:hobby-dev

# Set environment variables
heroku config:set SPOTIFY_CLIENT_ID=your_id
heroku config:set SPOTIFY_CLIENT_SECRET=your_secret

# Deploy
git push heroku main

# Run migrations
heroku run npm run migrate
```

Using Railway (Alternative)

bash

```
# Install Railway CLI
npm i -g @railway/cli

# Deploy
railway login
railway init
railway up

# Add services via dashboard
# - PostgreSQL
# - Redis
# - Set environment variables
```

📈 Growth Hacking Tips

1. Early User Acquisition

- **Reddit:** Post in r/WeAreTheMusicMakers
- **Discord:** Join music production servers
- **Twitter:** Target indie artists with #spotifyartist

2. Viral Features to Add

javascript

```
// Referral system
app.post('/api/referral', async (req, res) => {
  const { referrerId, newUserId } = req.body;

  // Give both users bonus follows
  await grantBonusFollows(referrerId, 100);
  await grantBonusFollows(newUserId, 50);

  res.json({ success: true });
});
```

3. Quick Analytics

javascript

```
// Track everything
function trackEvent(userId, event, data) {
  analytics.track({
    userId,
    event,
    properties: data,
    timestamp: new Date()
  });
}

// Key events to track
trackEvent(userId, 'Signup', { source: 'organic' });
trackEvent(userId, 'FirstFollow', { count: 1 });
trackEvent(userId, 'Upgrade', { plan: 'pro' });
```

🎓 Learning Resources

Spotify API

- Web API Documentation
- Rate Limits Guide
- Authorization Guide

Tech Stack

- Node.js Best Practices
- PostgreSQL Optimization
- Redis Patterns
- React Performance

Growth Hacking

- Viral Coefficient Calculator
- Growth Hacking Tactics
- Music Marketing Guide

💡 Pro Tips

1. Start Small

Don't try to follow 10,000 artists on day one. Start with 100 and gradually increase.

2. Monitor Everything

```javascript
// Add this to every critical function
const startTime = Date.now();
// ... your code ...
console.log(`Function took ${Date.now() - startTime}ms`);
```

3. Cache Aggressively

```javascript
// Cache user data for 1 hour
const userData = await redis.get(`user:${userId}`);
if (!userData) {
  const user = await db.query('SELECT * FROM users WHERE id = $1', [userId]);
  await redis.setex(`user:${userId}`, 3600, JSON.stringify(user));
}
```

4. Handle Errors Gracefully

```javascript
// Never let the app crash
process.on('unhandledRejection', (error) => {
  console.error('Unhandled rejection:', error);
  // Log to error tracking service
});
```

5. Test Rate Limits Safely

```javascript
// Use a test account first
if (process.env.NODE_ENV === 'development') {
  MAX_FOLLOWS_PER_HOUR = 5;  // Much lower in dev
}
```

🆘 Get Help

Community Support

- **Discord:** Join our server
- **GitHub Issues:** Report bugs and request features
- **Stack Overflow:** Tag with `spotify-api`

Professional Support

- **Email:** support@spotifyswarm.com
- **Documentation:** Full docs
- **API Status:** status.spotifyswarm.com

✅ Launch Checklist

Before going live, ensure:

- [ ] Spotify API credentials are production-ready
- [ ] Database has proper indexes
- [ ] Redis is configured for persistence
- [ ] Rate limiting is properly configured
- [ ] Error tracking is set up (Sentry)
- [ ] Analytics are implemented
- [ ] Terms of Service are in place
- [ ] Privacy Policy is published
- [ ] SSL certificate is installed
- [ ] Monitoring is active

🎉 You're Ready!

Congratulations! You now have everything you need to build and launch your Spotify Follow-Swarm service. Remember:

1. **Start simple** - Get the MVP working first
2. **Test thoroughly** - Especially rate limits
3. **Monitor everything** - Catch issues early
4. **Listen to users** - They'll guide your roadmap
5. **Stay compliant** - Follow Spotify's guidelines

Happy building! 🚀