Quick Start Guide

Get your Markup.io automation up and running with Supabase integration, smart screenshot matching, and comprehensive error logging in minutes!

1. Prerequisites

- Node.js (v16 or higher)
- Supabase account and project
- Basic understanding of REST APIs

2. Quick Setup

Clone and Install

```
git clone <your-repo>
cd markupio-clickup-automation
npm install
```

Environment Configuration

```
cp .env.template .env
# Edit .env with your credentials
```

Required environment variables:

```
SUPABASE_URL=your_supabase_url
SUPABASE_ANON_KEY=your_supabase_anon_key
SUPABASE_SERVICE_KEY=your_supabase_service_key
SUPABASE_BUCKET=screenshots
SCRAPER_TIMEOUT=90000
SCRAPER_RETRY_ATTEMPTS=3
SCRAPER_DEBUG_MODE=false
PORT=3000
```

Supabase Setup

1. Go to Supabase Dashboard
2. Create a new project or select existing
3. In SQL Editor, paste and run the content from `setup_database.sql`
4. This creates:
    - `scraped_data` table
    - `scraping_error_logs` table

    - `markup_projects` table
    - `markup_threads` table
    - `markup_comments` table
5. Create a storage bucket named `screenshots`
6. Copy your project URL and API keys to `.env`

3. Test Run

Start the Server

```
npm start
```

Server will start on `http://localhost:3000`

Test Complete Payload Extraction

```
curl -X POST http://localhost:3000/complete-payload \
  -H "Content-Type: application/json" \
  -d '{
    "url": "https://app.markup.io/markup/your-project-id"
  }'
```

What Happens:

1. ✅ Extracts thread data from Markup.io
2. ✅ Identifies thread names automatically
3. ✅ Uses smart matching to capture only relevant images
4. ✅ Skips images without comments
5. ✅ Saves to Supabase with URL deduplication
6. ✅ Logs any errors to error table

4. Verify Integration

Check Database Tables

Scraped Data:

```
SELECT * FROM scraped_data ORDER BY created_at DESC LIMIT 5;
```

Markup Projects:

```
SELECT * FROM markup_projects ORDER BY created_at DESC LIMIT 5;
```

Threads and Screenshots:

```
SELECT
  t.thread_name,
  t.image_path,
  COUNT(c.id) as comment_count
FROM markup_threads t
LEFT JOIN markup_comments c ON t.id = c.thread_id
GROUP BY t.id, t.thread_name, t.image_path;
```

Error Logs:

```
SELECT * FROM scraping_error_logs ORDER BY failed_at DESC LIMIT 10;
```

Check Screenshots in Storage

1. Go to Supabase Storage
2. Open `screenshots` bucket
3. Verify images are uploaded with session IDs

5. Understanding Smart Matching

Example Output

```
🏷️ Thread names extracted: 01. hero.jpg, 03. footer.jpg
📸 Using smart capture with 2 thread names
📝 Current image name: hero.jpg
✅ Image 1 matches thread: "01. hero.jpg"
⏭️  Image 2 ("sidebar.jpg") doesn't match any remaining thread, skipping...
📝 Current image name: footer.jpg
✅ Image 3 matches thread: "03. footer.jpg"
🎉 All 2 threads matched!
```

Check Logs for Matching Details

Enable debug mode to see detailed matching:

```
SCRAPER_DEBUG_MODE=true
```

6. Monitor Errors

View Recent Errors

```
SELECT
  error_message,
  error_details->>'operation' as operation,
  failed_at
FROM scraping_error_logs
ORDER BY failed_at DESC
LIMIT 10;
```

Check Incomplete Matches

```
SELECT
  url,
  error_details->>'matchedCount' as matched,
  error_details->>'expectedCount' as expected,
  error_details->>'unmatchedThreads' as unmatched
FROM scraping_error_logs
WHERE error_details->>'operation' = 'captureImagesMatchingThreads -
incomplete'
ORDER BY failed_at DESC;
```

7. Next Steps

- ✅ Review SMART_SCREENSHOT_MATCHING.md for matching details
- ✅ Read ERROR_LOGGING.md for monitoring strategies
- ✅ Check DATABASE_SETUP.md for schema details
- ✅ Integrate with ClickUp or other automation tools
- ✅ Set up monitoring alerts for error rates
- ✅ Customize options based on your needs

8. Troubleshooting

Database connection fails?

- Double-check SUPABASE_URL and API keys
- Ensure all tables are created correctly
- Verify storage bucket exists

Smart matching not working?

- Enable debug mode to see matching attempts
- Check error logs for incomplete matches
- Verify image names are accessible in fullscreen view

Incomplete matches reported?

```
-- Check what went wrong
SELECT
```

```
  error_message,
  error_details
FROM scraping_error_logs
WHERE error_details->>'operation' LIKE '%incomplete%'
ORDER BY failed_at DESC LIMIT 1;
```

Scraping fails?

- Enable debug mode: `SCRAPER_DEBUG_MODE=true`
- Check error logs table for detailed context
- Verify Markup.io URL is accessible
- Increase timeout if needed: `SCRAPER_TIMEOUT=120000`

Images not appearing?

- Check Supabase storage bucket permissions
- Verify bucket name matches .env configuration
- Check error logs for upload failures

Quick Commands

```
# Start server
npm start

# Test with debug mode
SCRAPER_DEBUG_MODE=true npm start

# Run direct script test
npm run payload

# Test URL checking
npm run test
```

Useful Queries

```
-- Today's activity
SELECT COUNT(*) FROM scraped_data
WHERE DATE(created_at) = CURRENT_DATE;

-- Error rate today
SELECT COUNT(*) FROM scraping_error_logs
WHERE DATE(failed_at) = CURRENT_DATE;

-- Most active URLs
SELECT url, COUNT(*) as count
FROM scraped_data
GROUP BY url
ORDER BY count DESC
```

```
LIMIT 10;

-- Recent incomplete matches
SELECT
  url,
  error_details->>'unmatchedThreads' as unmatched
FROM scraping_error_logs
WHERE error_details->>'operation' = 'captureImagesMatchingThreads -
incomplete'
ORDER BY failed_at DESC
LIMIT 5;
```

Need more help? Check the full README.md for detailed documentation!