Markup.io ClickUp Automation

A comprehensive automation system for capturing screenshots and extracting thread data from Markup.io projects with intelligent URL-based deduplication, automatic image replacement, and smart screenshot matching.

🌟 Key Features

- 🎯 **Smart Screenshot Matching**: Automatically matches images with threads by filename, skipping images without comments
- 🔍 **Smart URL Checking**: Automatically detects existing records for the same URL
- 🔄 **Image Replacement**: Replaces old screenshots with new ones instead of creating duplicates
- 💾 **Supabase Integration**: Stores data and images with intelligent update/create logic
- ⚡ **Single Browser Session**: Optimized extraction using one browser for both operations
- 🌐 **REST API Server**: HTTP endpoints for easy integration
- 📊 **Comprehensive Error Logging**: All errors logged to database for monitoring and debugging
- ✅ **ClickUp Ready**: Structured data output perfect for ClickUp automation workflows

🚀 Quick Start

Prerequisites

- Node.js (v16 or higher)
- Supabase account with configured database and storage bucket

Installation

```
# Clone and install dependencies
npm install

# Set up environment variables
cp .env.example .env
# Edit .env with your Supabase credentials
```

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
```

Database Setup

```
-- Run the SQL commands from supabase_schema.sql in your Supabase SQL
editor
-- This creates the scraped_data and scraping_error_logs tables
```

🕮 Usage

1. REST API Server (Recommended)

```
# Start the server
npm start

# The server runs on http://localhost:3000
```

Complete Payload Extraction (Optimized)

```
curl -X POST http://localhost:3000/complete-payload \
  -H "Content-Type: application/json" \
  -d '{
    "url": "https://app.markup.io/markup/your-project-id",
    "options": {
      "screenshotQuality": 90,
      "debugMode": false
    }
  }'
```

Response with URL Checking Info

```
{
  "success": true,
  "data": {
    "url": "https://app.markup.io/markup/your-project-id",
    "projectName": "Your Project",
    "threads": [...],
    "totalThreads": 3,
    "totalScreenshots": 3,
    "supabaseOperation": "updated",  // "created" or "updated"
    "oldImagesDeleted": 2,
    "supabaseRecordId": 123
  },
  "message": "Successfully extracted 3 threads with 3 screenshots (Updated
existing record, replaced 2 old images)",
  "supabaseOperation": "updated",
```

```
  "oldImagesDeleted": 2
}
```

2. Direct Script Usage

```
# Extract complete payload (threads + screenshots)
npm run payload

# Capture screenshots only
npm run capture

# Test URL checking functionality
npm run test
```

3. Programmatic Usage

```
const { getCompletePayload } = require('./getpayload.js');
const { captureMarkupScreenshots } = require('./script_integrated.js');

// Complete payload with URL checking
const result = await getCompletePayload('https://app.markup.io/markup/your-
id');

if (result.success) {
  console.log(`Operation: ${result.supabaseOperation}`); // 'created' or
'updated'
  console.log(`Old images deleted: ${result.oldImagesDeleted}`);

  // Process threads
  result.threads.forEach(thread => {
    console.log(`Thread: ${thread.threadName}`);
    console.log(`Comments: ${thread.comments.length}`);
    console.log(`Screenshot: ${thread.imagePath}`);
  });
}
```

🎯 Smart Screenshot Matching

How It Works

The system intelligently matches images with their corresponding threads by filename:

1. **Thread Name Extraction**: Extracts thread names like `"01. 1234-56a TEST Folder.jpg"`
2. **Image Name Detection**: Reads image names from fullscreen view (e.g., `"1234-56a TEST Folder.jpg"`)
3. **Smart Matching**: Removes leading numbers from thread names and matches with image names
4. **Intelligent Navigation**: Clicks "next" button and checks each image, skipping non-matching ones

5. **Complete Coverage**: Ensures every thread gets its corresponding screenshot

Example Scenario

```
Images available: [image1.jpg, image2.jpg, image3.jpg, image4.jpg]
Threads with comments: [01. image1.jpg, 03. image3.jpg]

Traditional approach (WRONG):
  ✖ Would capture image1.jpg and image2.jpg

Smart matching approach (CORRECT):
  ✅ Captures image1.jpg (matches thread "01. image1.jpg")
  ⏭️  Skips image2.jpg (no matching thread)
  ✅ Captures image3.jpg (matches thread "03. image3.jpg")
  ⏭️  Skips image4.jpg (no matching thread)
```

Benefits

- ✅ **Accurate**: Only captures images that have threads/comments
- ✅ **Efficient**: Automatically skips irrelevant images
- ✅ **Flexible**: Works regardless of image order
- ✅ **Robust**: Handles missing images gracefully

🔄 URL Checking & Image Replacement Logic

How It Works

1. **URL Check**: Before saving, system checks if URL already exists in database
2. **Update vs Create**:
   - If URL exists → Updates existing record + deletes old images
   - If URL is new → Creates new record
3. **Image Management**:
   - Old screenshots are deleted from Supabase storage
   - New screenshots are uploaded with fresh session ID paths
   - No orphaned files left behind

Benefits

- **No Duplicates**: Same URL never creates multiple database records
- **Storage Efficiency**: Old images are automatically cleaned up
- **Data Freshness**: Latest screenshots always replace older versions
- **Cost Optimization**: Prevents Supabase storage bloat
- **Smart Matching**: Only relevant images are captured and stored

📊 Error Logging & Monitoring

Comprehensive Error Tracking

All errors during execution are logged to the `scraping_error_logs` table with:

- Full error messages and stack traces
- Operation context (what was happening when it failed)
- Progress information (matched/expected counts)
- Thread names and image details
- Session IDs for tracking

Error Categories

- **Image Name Extraction Errors**: When image name can't be read from fullscreen
- **Navigation Errors**: When clicking "next" button fails
- **Incomplete Matching**: When not all threads can be matched with images
- **Thread Extraction Errors**: When thread data can't be extracted
- **Database Errors**: When saving to Supabase fails

Monitoring Queries

```
-- Get all incomplete matches
SELECT * FROM scraping_error_logs
WHERE error_details->>'operation' = 'captureImagesMatchingThreads -
incomplete'
ORDER BY failed_at DESC;

-- Get errors by URL
SELECT * FROM scraping_error_logs
WHERE url = 'your-markup-url'
ORDER BY failed_at DESC;

-- Error rate by day
SELECT DATE(failed_at), COUNT(*)
FROM scraping_error_logs
WHERE failed_at > NOW() - INTERVAL '7 days'
GROUP BY DATE(failed_at);
```

See ERROR_LOGGING.md for complete documentation.

📊 Database Schema

`scraped_data` Table

```
- id (Primary Key)
- session_id (UUID)
- url (Text) - Used for deduplication
- title (Text)
- number_of_images (Integer)
- screenshot_metadata (JSONB)
- screenshots_paths (Text Array) - URLs to images in storage
- duration_seconds (Numeric)
```

```
- success (Boolean)
- options (JSONB)
- created_at (Timestamp)
- updated_at (Timestamp) - Auto-updated on record changes
```

`scraping_error_logs` Table

```
- id (Primary Key)
- session_id (UUID)
- url (Text)
- title (Text)
- error_message (Text)
- number_of_images (Integer)
- error_details (JSONB) - Detailed context including operation, progress,
thread names
- options (JSONB)
- failed_at (Timestamp)
- status (Text: 'failed', 'retrying', 'resolved')
```

Normalized Schema Tables

```
- markup_projects: Project-level information
- markup_threads: Individual threads with image references
- markup_comments: Comments within threads
```

See DATABASE_SETUP.md for complete schema.

🛠️ API Endpoints

Method	Endpoint	Description
GET	`/`	API documentation
GET	`/health`	Health check
POST	`/complete-payload`	**Recommended**: Extract threads + screenshots (optimized)
POST	`/capture`	Screenshot capture with JSON payload
GET	`/capture`	Simple screenshot capture via query params
POST	`/diagnose`	Debug mode capture with detailed logs

🔧 Configuration Options

```
const options = {
  numberOfImages: 'auto',        // Auto-detects from thread count
```

```
  threadNames: null,              // Optional: Array of thread names for
smart matching
  screenshotQuality: 90,          // JPEG quality (1-100)
  timeout: 90000,                 // Request timeout in ms
  retryAttempts: 3,               // Number of retry attempts
  debugMode: false,               // Enable detailed logging
  waitForFullscreen: true,        // Try to activate fullscreen mode
  screenshotFormat: 'jpeg'        // Image format
};
```

Smart Matching Options

When using `getCompletePayload()`, thread names are automatically extracted and used for smart matching. You can also manually provide thread names:

```
const screenshotter = new MarkupScreenshotter({
  numberOfImages: 5,
  threadNames: [
    "01. hero-section.jpg",
    "03. features-grid.jpg",
    "05. cta-banner.jpg"
  ],
  screenshotQuality: 90
});
```

🧪 Testing

```
# Test URL checking and update functionality
npm run test

# This will:
# 1. Check for existing records for test URL
# 2. Run extraction and show operation type (created/updated)
# 3. Run again immediately to test update behavior
# 4. Display before/after database records
```

🎯 ClickUp Integration

The extracted data structure is optimized for ClickUp automation:

```
{
  "projectName": "Project Name",
  "threads": [
    {
      "threadName": "Header Navigation",
      "imageIndex": 1,
```

```
      "imagePath": "https://supabase-url/image1.jpg",
      "comments": [
        {
          "pinNumber": "1",
          "userName": "John Doe",
          "messageContent": "Fix the navigation alignment",
          "threadId": "thread-123"
        }
      ]
    }
  ]
}
```

🔍 Monitoring & Debugging

View Recent Activities

```
const supabaseService = new SupabaseService();
const recent = await supabaseService.getRecentActivities(10);
console.log(recent);
```

Check Failed Scrapings

```
const failed = await supabaseService.getFailedScrapings(10);
console.log(failed);
```

Check Incomplete Matches

```
-- See which threads couldn't be matched
SELECT
  error_message,
  error_details->>'unmatchedThreads' as unmatched,
  error_details->>'matchedCount' as matched,
  error_details->>'expectedCount' as expected,
  failed_at
FROM scraping_error_logs
WHERE error_details->>'operation' = 'captureImagesMatchingThreads -
incomplete'
ORDER BY failed_at DESC;
```

Debug Mode

Set `SCRAPER_DEBUG_MODE=true` or use `debugMode: true` in options for detailed logs including:

- Image name extraction attempts

- Thread matching details
- Navigation button searches
- Screenshot capture progress

🚦 Error Handling

- **Automatic Retries**: Failed operations retry with exponential backoff
- **Graceful Degradation**: Screenshot failures don't break thread extraction
- **Comprehensive Logging**: All errors logged to Supabase with full context
- **Smart Matching Fallback**: Falls back to sequential capture if thread names unavailable
- **Incomplete Match Warnings**: Logs when not all threads can be matched
- **Recovery Options**: Failed operations can be manually retriggered with logged context

📁 File Structure

```
├── server.js                      # REST API server
├── db_helper.js                   # Screenshot capture with smart matching
├── getpayload.js                  # Thread extraction + screenshot
combination
├── supabase-service.js            # Database operations with URL checking
├── db_response_helper.js          # Database response utilities
├── supabase_schema.sql            # Database schema
├── setup_database.sql             # Complete database setup
├── test_url_checking.js           # Test script for URL checking
├── package.json                   # Dependencies and scripts
├── README.md                      # This file
├── DATABASE_SETUP.md              # Database setup guide
├── SMART_SCREENSHOT_MATCHING.md   # Smart matching documentation
├── ERROR_LOGGING.md               # Error logging documentation
└── ERROR_LOGGING_SUMMARY.md       # Error logging quick reference
```

🔄 Workflow Examples

New Project Workflow

1. Extract markup → **Creates new record with smart matching**
2. Re-extract same markup → **Updates existing record, replaces images**
3. **Smart matching** ensures only images with threads are captured
4. No duplicate records, storage stays clean

Smart Matching Workflow

1. Thread extraction identifies 5 threads from images: 1, 2, 4, 6, 8
2. Smart matching navigates through all images in fullscreen
3. Captures only images that match thread names (skips 3, 5, 7, 9, 10)
4. Each thread gets its correct corresponding screenshot
5. Incomplete matches logged for monitoring

Production Integration

1. Webhook receives markup URL
2. API call to `/complete-payload`
3. Smart matching automatically used for screenshot capture
4. Response indicates if record was created or updated
5. ClickUp tasks created/updated accordingly
6. Old screenshots automatically cleaned up
7. Error logs available for monitoring

🛡️ Security & Best Practices

- Use Supabase Service Key for server environments
- Use Supabase Anon Key for client environments
- Enable RLS (Row Level Security) on Supabase tables
- Set up proper CORS policies
- Monitor storage usage and set up cleanup jobs
- Use environment variables for all sensitive configuration

📈 Performance Optimizations

- **Single Browser Session**: Complete payload extraction uses one browser instance
- **Smart Navigation**: Only navigates to and captures relevant images
- **Parallel Processing**: Screenshots captured in parallel when possible
- **Smart Caching**: URL checking prevents unnecessary duplicate processing
- **Automatic Cleanup**: Old images deleted to prevent storage bloat
- **Connection Pooling**: Efficient Supabase connections
- **Intelligent Matching**: Skips non-relevant images automatically

📚 Additional Documentation

- **SMART_SCREENSHOT_MATCHING.md**: Complete guide to smart matching feature
- **ERROR_LOGGING.md**: Comprehensive error logging documentation
- **ERROR_LOGGING_SUMMARY.md**: Quick reference for error logging
- **DATABASE_SETUP.md**: Database schema and setup guide
- **QUICKSTART.md**: Quick start guide for new users

🤝 Contributing

1. Fork the repository
2. Create a feature branch
3. Test URL checking functionality with the test script
4. Submit a pull request with clear description of changes

📄 License

ISC License - see package.json for details.

🆘 Support

For issues related to:

- **Smart Matching**: Check SMART_SCREENSHOT_MATCHING.md and error logs
- **Error Logging**: See ERROR_LOGGING.md for monitoring queries
- **URL Checking**: Run `npm run test` to verify functionality
- **Database Issues**: Check Supabase logs and connection settings
- **Screenshot Problems**: Enable debug mode and check detailed logs
- **API Integration**: Check server logs and endpoint documentation
- **Incomplete Matches**: Query `scraping_error_logs` table for details