# **Met to Shopify Importer: Complete User Manual**

## **Table of Contents**

1. Introduction
2. System Architecture
3. Installation Guide
4. Using the Application
   * Configuration
   * Creating Import Jobs
   * Monitoring Progress
   * Managing Results
5. Technical Features
6. Maintenance and Troubleshooting
7. Customization Options
8. Appendix: Code Reference

## **1. Introduction**

The Met to Shopify Importer is a comprehensive web application designed to automate the process of importing artwork from the Metropolitan Museum of Art's Open Access collection directly to your Shopify store. This application streamlines what was previously a manual, time-consuming process by handling everything from fetching artwork data to generating descriptions and uploading products to Shopify.

### **Key Benefits**

* **Time Savings**: Automate the entire workflow from the Met Museum to your Shopify store
* **Intelligent Categorization**: Automatically organize artwork into appropriate collections
* **AI-Generated Descriptions**: Create professional product descriptions using OpenAI
* **Rate Limit Management**: Handle API restrictions gracefully without manual intervention
* **Flexible Import Options**: Use Met Museum search URLs or custom category filters

## **2. System Architecture**

The Met to Shopify Importer consists of three main components:

1. **Backend API Server** (Node.js/Express)
   * Met Museum API integration
   * OpenAI description generation
   * Shopify product uploads
   * Job management system
   * Rate limit handling
2. **Frontend Interface** (React)
   * Dashboard for monitoring import jobs
   * Search URL and category-based imports
   * Job details and previews
   * Settings management
3. **Database** (MongoDB)
   * Store job information
   * Track processed artwork
   * Maintain application state

All components are containerized with Docker for easy deployment and maintenance.

## **3. Installation Guide**

### **Prerequisites**

* A server or VPS with:
  + At least 2GB RAM
  + 1+ CPU cores (2+ recommended)
  + 20GB+ storage
  + Linux operating system (Ubuntu 20.04+ recommended)
* Docker and Docker Compose installed
* API credentials for:
  + OpenAI (for description generation)
  + Shopify (for product uploads)

### **Docker Deployment (Recommended)**

**Prepare your server**:  
 bash  
*# Create a directory for the application*

mkdir -p ~/met-to-shopify

1. cd ~/met-to-shopify

**Copy application files**: Ensure all code files are copied to your server with the following structure:  
 met-to-shopify/

├── client/ # React frontend

│ ├── public/

│ ├── src/

│ └── package.json

├── models/ # MongoDB models

├── routes/ # API routes

├── services/ # Business logic

├── utils/ # Helper functions

├── server.js # Main server file

├── package.json # Backend dependencies

├── docker-compose.yml # Docker Compose configuration

├── Dockerfile.server # Backend Dockerfile

├── Dockerfile.client # Frontend Dockerfile

├── nginx.conf # Nginx configuration

1. └── .env # Environment variables

**Configure environment variables**:  
 bash  
*# Create .env file from template*

cp .env.template .env

nano .env  
 Add your API keys and credentials:  
 # MongoDB

MONGODB\_URI=mongodb://root:example@mongo:27017/met-shopify?authSource=admin

# OpenAI API (for artwork descriptions)

OPENAI\_API\_KEY=your\_openai\_api\_key\_here

OPENAI\_MODEL=gpt-3.5-turbo

# Shopify API (for direct uploads)

SHOPIFY\_SHOP\_NAME=your-store.myshopify.com

SHOPIFY\_API\_KEY=your\_shopify\_api\_key\_here

1. SHOPIFY\_API\_PASSWORD=your\_shopify\_api\_password\_here
2. **Start the application**:  
    bash  
   docker-compose up -d
3. **Verify deployment**:  
    bash  
   docker-compose ps  
    You should see three containers running:
   * met-to-shopify\_mongo\_1
   * met-to-shopify\_api\_1
   * met-to-shopify\_web\_1
4. **Access the application**: Open a web browser and navigate to your server's IP address or domain.

## **4. Using the Application**

### **Configuration**

1. **Access the Settings page** from the navigation menu
2. **Configure OpenAI API** for description generation:
   * API Key
   * Model selection (GPT-3.5-Turbo or GPT-4)
3. **Configure Shopify API** for direct product uploads:
   * Shop name (your-store.myshopify.com)
   * API Key
   * API Password/Access Token
4. **Set Default Import Options**:
   * Maximum items per import
   * Default pricing
   * CSV-only mode preference

### **Creating Import Jobs**

#### **Option 1: Met Museum URL Import**

1. Navigate to the **New Import** page
2. Select the **Met Search URL** tab
3. Paste a search URL from the Met Museum website:  
    https://www.metmuseum.org/art/collection/search?showOnly=openAccess&department=11&era=19th+Century
4. Add a descriptive job name (optional)
5. Configure advanced options if needed:
   * Maximum items
   * CSV-only mode
   * Default price
6. Click **Start Import**

#### **Option 2: Category-Based Import**

1. Navigate to the **New Import** page
2. Select the **Categories & Filters** tab
3. Choose artwork types (Paintings, Drawings, etc.)
4. Select time periods
5. Enter keywords for specific themes or subjects
6. Add a descriptive job name (optional)
7. Configure advanced options if needed
8. Click **Start Import**

### **Monitoring Progress**

1. **Dashboard** - View all active, completed, and failed jobs
2. **Job Details** - Click on any job to see:
   * Overall progress
   * Processed artwork preview
   * Status information
   * Rate limit information (if applicable)
3. **Job Controls**:
   * **Pause**: Temporarily stop processing
   * **Resume**: Continue a paused job
   * **Cancel**: Stop the job permanently

### **Managing Results**

1. **CSV Download**:
   * For any completed job, download a CSV file ready for Shopify import
   * Contains all product details and image URLs
2. **Direct Shopify Upload**:
   * For jobs run in CSV-only mode, click "Upload to Shopify" to start the upload process
   * Monitor upload progress like any other job
3. **Results View**:
   * See all processed artwork
   * View collections and tags assigned to each item
   * Check for any processing errors

## **5. Technical Features**

### **URL Parsing**

The application intelligently parses Met Museum search URLs to extract:

* Department IDs
* Time periods
* Search terms
* Filter options

### **Smart Categorization**

Artwork is automatically categorized based on two systems:

**Time Period/Era**:  
 'Prehistoric / Ancient': { start: -10000, end: 499 },

'Classical Antiquity': { start: -800, end: 499 },

'Medieval': { start: 500, end: 1399 },

'Renaissance': { start: 1400, end: 1599 },

'Baroque / Rococo': { start: 1600, end: 1749 },

'Enlightenment': { start: 1750, end: 1799 },

'19th Century': { start: 1800, end: 1899 },

'Early 20th Century': { start: 1900, end: 1945 },

1. 'Mid to Late 20th Century': { start: 1946, end: 1999 }

**Theme/Subject** (partial list):  
 'People': ['people', 'person', 'human', 'figure', 'portrait', 'man', 'woman', 'boy', 'girl'],

'Nature': ['nature', 'natural', 'landscape', 'outdoor', 'garden', 'field'],

'Animals': ['animal', 'beast', 'bird', 'fish', 'dog', 'cat', 'horse', 'lion', 'tiger', 'creature'],

1. 'Landscapes': ['landscape', 'scenery', 'vista', 'mountain', 'river', 'lake', 'ocean', 'sea', 'forest'],

### **Rate Limit Management**

The application includes a sophisticated rate limit management system:

* Tracks API usage for Met Museum, OpenAI, and Shopify
* Automatically pauses when limits are reached
* Displays countdown to resumption
* Continues processing from where it left off

### **Job Management**

The background job processor:

* Manages job queues and priorities
* Handles job state (pending, processing, paused, completed, failed)
* Provides real-time progress updates
* Recovers from errors and interruptions

## **6. Maintenance and Troubleshooting**

### **Regular Maintenance**

**Monitoring Logs**:  
 bash  
*# View application logs*

1. docker-compose logs -f api

**Database Backups**:  
 bash  
*# Backup MongoDB data*

1. docker exec met-to-shopify\_mongo\_1 sh -c 'mongodump --archive' > backup\_$(date +%Y%m%d).archive

**Updates**:  
 bash  
*# Update application with new code*

docker-compose down

*# Replace updated files*

1. docker-compose up -d --build

### **Troubleshooting Common Issues**

1. **API Rate Limits**
   * **Symptom**: Jobs frequently pause with "Rate limit reached" messages
   * **Solution**:
     + Reduce the number of concurrent jobs
     + Increase pauses between requests in the code
2. **OpenAI API Issues**
   * **Symptom**: Missing descriptions or errors in job details
   * **Solution**:
     + Verify API key in Settings
     + Check OpenAI account billing status
     + Try a different model (GPT-3.5-Turbo instead of GPT-4)
3. **Shopify API Problems**
   * **Symptom**: Products not appearing in Shopify
   * **Solution**:
     + Verify API credentials in Settings
     + Check API permission scopes
     + Try CSV download and manual import
4. **Performance Issues**
   * **Symptom**: Slow processing or high server load
   * **Solution**:
     + Reduce maximum items per job
     + Increase server resources
     + Use CSV-only mode for large imports

## **7. Customization Options**

### **Modifying Categorization Logic**

You can customize how artwork is categorized by editing:

1. **Time Period Ranges**:
   * Edit eraPeriods object in services/metService.js
2. **Theme Keywords**:
   * Modify themeCollections object in services/metService.js
   * Add or remove keywords associated with each theme

### **Adjusting Description Generation**

1. **Change the AI Model**:
   * Update the OPENAI\_MODEL environment variable
   * Options: gpt-3.5-turbo (faster, cheaper) or gpt-4 (higher quality)
2. **Modify the Prompt**:
   * Edit the prompt template in the generateDescription function in services/openaiService.js

### **Frontend Customization**

1. **Branding**:
   * Update logos and colors in the React components
   * Modify the header component to include your branding
2. **Additional Features**:
   * Extend the React components to add new functionality
   * Modify the dashboard to include additional metrics

## **8. Appendix: Code Reference**

The complete application code is organized as follows:

### **Backend (Node.js/Express)**

* server.js - Main server file
* models/Job.js - MongoDB job model
* routes/jobRoutes.js - API routes for job management
* routes/metRoutes.js - API routes for Met Museum data
* services/metService.js - Met Museum API integration
* services/openaiService.js - OpenAI integration
* services/shopifyService.js - Shopify API integration
* services/rateLimitManager.js - API rate limit handling
* services/jobProcessor.js - Background job processing
* utils/metUrlParser.js - Met Museum URL parsing

### **Frontend (React)**

* client/src/App.js - Main React component
* client/src/pages/Dashboard.js - Dashboard page
* client/src/pages/NewImport.js - Import creation page
* client/src/pages/JobDetails.js - Job details page
* client/src/pages/Settings.js - Settings page
* client/src/components/JobCard.js - Job card component
* client/src/components/Header.js - Header component
* client/src/components/LoadingSpinner.js - Loading spinner component

### **Deployment**

* docker-compose.yml - Docker Compose configuration
* Dockerfile.server - Backend Dockerfile
* Dockerfile.client - Frontend Dockerfile
* nginx.conf - Nginx configuration

*This document serves as a comprehensive guide for deploying, using, and maintaining the Met to Shopify Importer application. For additional assistance, refer to the logs and code comments, or seek support from your development team.*