import os import json import time import logging import threading import schedule from datetime import datetime from flask import Flask, request, jsonify from pymongo import MongoClient import redis import requests from bs4 import BeautifulSoup import pandas as pd import numpy as np from dotenv import load_dotenv

Load environment variables

load_dotenv()

Configure logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s') logger = logging.getLogger(**name**)

Initialize Flask app

app = Flask(**name**)

Connect to MongoDB

mongo_uri = os.getenv("MONGODB_URI", "mongodb://localhost:27017/amgf") mongo_client = MongoClient(mongo_uri) db = mongo_client.amgf opportunities_collection = db.opportunities platforms_collection = db.platforms metrics_collection = db.metrics

Connect to Redis

redis_uri = os.getenv("REDIS_URI", "redis://localhost:6379") redis_client = redis.from_url(redis_uri)

Constants

SCAN_INTERVAL = int(os.getenv("SCAN_INTERVAL", "3600")) # Default: scan every hour OPPORTUNITY_TTL = int(os.getenv("OPPORTUNITY_TTL", "86400")) # Default: 24 hours MAX_OPPORTUNITIES = int(os.getenv("MAX_OPPORTUNITIES", "100")) # Max opportunities to store

class OpportunityScanner: """ Main class for scanning different platforms for opportunities. """

```
def __init__(self):
    self.platforms = {
        "service": ["fiverr", "upwork"],
        "content": ["medium", "youtube"],
        "marketplace": ["etsy", "creative_market", "gumroad"]
    }
    self.scanners = {
        "fiverr": self.scan_fiverr,
        "upwork": self.scan_upwork,
        "medium": self.scan_medium,
        "youtube": self.scan_youtube,
        "etsy": self.scan_etsy,
        "creative_market": self.scan_creative_market,
        "gumroad": self.scan_gumroad
    }
```

```
def scan_all_platforms(self):
    """Scan all platforms for opportunities."""
    logger.info("Starting scan of all platforms")
    all_opportunities = []

    for category, platforms in self.platforms.items():
        for platform in platforms:
            try:
                logger.info(f"Scanning {platform} for opportunities")
                opportunities = self.scanners[platform]()
                if opportunities:
                    for opp in opportunities:
                        opp["category"] = category
                        opp["platform"] = platform
                        opp["timestamp"] = datetime.utcnow().isoformat()
                        opp["ttl"] = int(time.time()) + OPPORTUNITY_TTL
                    all_opportunities.extend(opportunities)
                    logger.info(f"Found {len(opportunities)} opportunities
on {platform}")
                else:
                    logger.info(f"No opportunities found on {platform}")
            except Exception as e:
                logger.error(f"Error scanning {platform}: {str(e)}")

    # Store opportunities in MongoDB
    if all_opportunities:
        # Remove old opportunities
        opportunities_collection.delete_many({"ttl": {"$lt":
int(time.time())}})

        # Insert new opportunities
        opportunities_collection.insert_many(all_opportunities)
        logger.info(f"Stored {len(all_opportunities)} new opportunities")

        # Update metrics
        metrics_collection.update_one(
            {"metric": "opportunity_scan"},
            {"$set": {
                "last_scan": datetime.utcnow().isoformat(),
                "opportunities_found": len(all_opportunities),
                "platforms_scanned": len(self.platforms)
            }},
            upsert=True
        )

        # Publish event to Redis
        redis_client.publish(
            "events:opportunity_scan_complete",
            json.dumps({
                "timestamp": datetime.utcnow().isoformat(),
                "opportunities_found": len(all_opportunities)
            })
        )

    return all_opportunities

def scan_fiverr(self):
    """Scan Fiverr for service opportunities."""
    logger.info("Scanning Fiverr for opportunities")
    opportunities = []

    # Categories to scan
    categories = [
        "data-entry",
        "virtual-assistant",
        "data-processing",
        "web-research",
        "social-media-management",
        "content-writing",
        "translation",
        "graphic-design",
        "video-editing"
    ]
```

```
    for category in categories:
        try:
            # In a real implementation, this would use the Fiverr API
            # For now, we'll simulate finding opportunities

            # Simulate 2-5 opportunities per category
            num_opportunities = np.random.randint(2, 6)

            for i in range(num_opportunities):
                opportunity = {
                    "id": f"fiverr_{category}_{i}",
                    "title": f"{category.replace('-', ' ').title()}
Service",
                    "description": f"Opportunity to provide
{category.replace('-', ' ')} services",
                    "url":
f"https://www.fiverr.com/categories/{category}",
                    "estimated_value": round(np.random.uniform(5, 50), 2),
                    "competition_level": np.random.choice(["low",
"medium", "high"]),
                    "implementation_time": np.random.randint(1, 5),  #
hours
                    "skills_required": [category.replace('-', ' ')],
                    "source": "fiverr",
                    "opportunity_type": "service"
                }
                opportunities.append(opportunity)
        except Exception as e:
            logger.error(f"Error scanning Fiverr category {category}:
{str(e)}")

    return opportunities

def scan_upwork(self):
    """Scan Upwork for service opportunities."""
    logger.info("Scanning Upwork for opportunities")
    opportunities = []

    # Categories to scan
    categories = [
        "data-entry",
        "virtual-assistant",
        "data-processing",
        "web-research",
        "social-media-management",
        "content-writing",
        "translation",
        "graphic-design",
        "video-editing"
    ]

    for category in categories:
        try:
            # In a real implementation, this would use the Upwork API
            # For now, we'll simulate finding opportunities

            # Simulate 2-5 opportunities per category
            num_opportunities = np.random.randint(2, 6)

            for i in range(num_opportunities):
                opportunity = {
                    "id": f"upwork_{category}_{i}",
                    "title": f"{category.replace('-', ' ').title()}
Project",
                    "description": f"Opportunity to provide
{category.replace('-', ' ')} services",
                    "url": f"https://www.upwork.com/search/jobs/?q=
{category}",
                    "estimated_value": round(np.random.uniform(20, 100),
2),
                    "competition_level": np.random.choice(["low",
"medium", "high"]),
                    "implementation_time": np.random.randint(2, 8),  #
```

```
hours
                    "skills_required": [category.replace('-', ' ')],
                    "source": "upwork",
                    "opportunity_type": "service"
                }
                opportunities.append(opportunity)
        except Exception as e:
            logger.error(f"Error scanning Upwork category {category}:
{str(e)}")

    return opportunities

def scan_medium(self):
    """Scan Medium for content opportunities."""
    logger.info("Scanning Medium for opportunities")
    opportunities = []

    # Topics to scan
    topics = [
        "technology",
        "programming",
        "data-science",
        "artificial-intelligence",
        "productivity",
        "self-improvement",
        "entrepreneurship",
        "marketing",
        "design"
    ]

    for topic in topics:
        try:
            # In a real implementation, this would use the Medium API or
web scraping
            # For now, we'll simulate finding opportunities

            # Simulate 1-3 opportunities per topic
            num_opportunities = np.random.randint(1, 4)

            for i in range(num_opportunities):
                opportunity = {
                    "id": f"medium_{topic}_{i}",
                    "title": f"{topic.replace('-', ' ').title()} Content",
                    "description": f"Opportunity to create content about
{topic.replace('-', ' ')}",
                    "url": f"https://medium.com/tag/{topic}",
                    "estimated_value": round(np.random.uniform(10, 30),
2),
                    "competition_level": np.random.choice(["low",
"medium", "high"]),
                    "implementation_time": np.random.randint(2, 6),  #
hours
                    "skills_required": ["writing", topic.replace('-', '
')],
                    "source": "medium",
                    "opportunity_type": "content"
                }
                opportunities.append(opportunity)
        except Exception as e:
            logger.error(f"Error scanning Medium topic {topic}: {str(e)}")

    return opportunities

def scan_youtube(self):
    """Scan YouTube for content opportunities."""
    logger.info("Scanning YouTube for opportunities")
    opportunities = []

    # Topics to scan
    topics = [
        "technology",
        "programming",
        "data-science",
        "artificial-intelligence",
```

```
        "productivity",
        "self-improvement",
        "entrepreneurship",
        "marketing",
        "design"
    ]

    for topic in topics:
        try:
            # In a real implementation, this would use the YouTube API
            # For now, we'll simulate finding opportunities

            # Simulate 1-3 opportunities per topic
            num_opportunities = np.random.randint(1, 4)

            for i in range(num_opportunities):
                opportunity = {
                    "id": f"youtube_{topic}_{i}",
                    "title": f"{topic.replace('-', ' ').title()} Video",
                    "description": f"Opportunity to create video content
about {topic.replace('-', ' ')}",
                    "url": f"https://www.youtube.com/results?search_query=
{topic}",
                    "estimated_value": round(np.random.uniform(20, 50),
2),
                    "competition_level": np.random.choice(["low",
"medium", "high"]),
                    "implementation_time": np.random.randint(3, 8),  #
hours
                    "skills_required": ["video editing", topic.replace('-
', ' ')],
                    "source": "youtube",
                    "opportunity_type": "content"
                }
                opportunities.append(opportunity)
        except Exception as e:
            logger.error(f"Error scanning YouTube topic {topic}:
{str(e)}")

    return opportunities

def scan_etsy(self):
    """Scan Etsy for marketplace arbitrage opportunities."""
    logger.info("Scanning Etsy for opportunities")
    opportunities = []

    # Categories to scan
    categories = [
        "digital-planners",
        "printables",
        "social-media-templates",
        "resume-templates",
        "presentation-templates",
        "digital-art",
        "ebook-templates",
        "website-templates",
        "graphic-design-elements"
    ]

    for category in categories:
        try:
            # In a real implementation, this would use the Etsy API
            # For now, we'll simulate finding opportunities

            # Simulate 1-4 opportunities per category
            num_opportunities = np.random.randint(1, 5)

            for i in range(num_opportunities):
                opportunity = {
                    "id": f"etsy_{category}_{i}",
                    "title": f"{category.replace('-', ' ').title()}
Product",
                    "description": f"Opportunity to sell
{category.replace('-', ' ')} products",
```

```
                    "url": f"https://www.etsy.com/search?q={category}",
                    "estimated_value": round(np.random.uniform(15, 40),
2),
                    "competition_level": np.random.choice(["low",
"medium", "high"]),
                    "implementation_time": np.random.randint(2, 6),  #
hours
                    "skills_required": ["design", category.replace('-', '
')],
                    "source": "etsy",
                    "opportunity_type": "marketplace"
                }
                opportunities.append(opportunity)
        except Exception as e:
            logger.error(f"Error scanning Etsy category {category}:
{str(e)}")

    return opportunities

def scan_creative_market(self):
    """Scan Creative Market for marketplace arbitrage opportunities."""
    logger.info("Scanning Creative Market for opportunities")
    opportunities = []

    # Categories to scan
    categories = [
        "templates",
        "graphics",
        "fonts",
        "add-ons",
        "photos",
        "themes",
        "3d",
        "web-elements",
        "ui-kits"
    ]

    for category in categories:
        try:
            # In a real implementation, this would use the Creative Market
API or web scraping
            # For now, we'll simulate finding opportunities

            # Simulate 1-3 opportunities per category
            num_opportunities = np.random.randint(1, 4)

            for i in range(num_opportunities):
                opportunity = {
                    "id": f"creative_market_{category}_{i}",
                    "title": f"{category.replace('-', ' ').title()}
Product",
                    "description": f"Opportunity to sell
{category.replace('-', ' ')} products",
                    "url": f"https://creativemarket.com/search?q=
{category}",
                    "estimated_value": round(np.random.uniform(20, 60),
2),
                    "competition_level": np.random.choice(["low",
"medium", "high"]),
                    "implementation_time": np.random.randint(3, 7),  #
hours
                    "skills_required": ["design", category.replace('-', '
')],
                    "source": "creative_market",
                    "opportunity_type": "marketplace"
                }
                opportunities.append(opportunity)
        except Exception as e:
            logger.error(f"Error scanning Creative Market category
{category}: {str(e)}")

    return opportunities

def scan_gumroad(self):
```

```
    """Scan Gumroad for marketplace arbitrage opportunities."""
    logger.info("Scanning Gumroad for opportunities")
    opportunities = []

    # Categories to scan
    categories = [
        "templates",
        "ebooks",
        "courses",
        "software",
        "design-assets",
        "music",
        "photography",
        "writing",
        "coaching"
    ]

    for category in categories:
        try:
            # In a real implementation, this would use the Gumroad API or
web scraping
            # For now, we'll simulate finding opportunities

            # Simulate 1-3 opportunities per category
            num_opportunities = np.random.randint(1, 4)

            for i in range(num_opportunities):
                opportunity = {
                    "id": f"gumroad_{category}_{i}",
                    "title": f"{category.replace('-', ' ').title()}
Product",
                    "description": f"Opportunity to sell
{category.replace('-', ' ')} products",
                    "url": f"https://gumroad.com/discover?query=
{category}",
                    "estimated_value": round(np.random.uniform(25, 70),
2),
                    "competition_level": np.random.choice(["low",
"medium", "high"]),
                    "implementation_time": np.random.randint(3, 8),  #
hours
                    "skills_required": ["digital product creation",
category.replace('-', ' ')],
                    "source": "gumroad",
                    "opportunity_type": "marketplace"
                }
                opportunities.append(opportunity)
        except Exception as e:
            logger.error(f"Error scanning Gumroad category {category}:
{str(e)}")

    return opportunities
```

Initialize scanner

scanner = OpportunityScanner()

API Routes

@app.route(‘/health’, methods=[‘GET’]) def health_check(): “““Health check endpoint.””” return jsonify({“status”: “healthy”, “timestamp”: datetime.utcnow().isoformat()})

@app.route(‘/scan’, methods=[‘POST’]) def trigger_scan(): “““Trigger a scan of all platforms.””” try: opportunities = scanner.scan_all_platforms() return jsonify({ “status”: “success”, “message”: f”Scan completed. Found {len(opportunities)} opportunities.”, “timestamp”: datetime.utcnow().isoformat() }) except Exception as e: logger.error(f”Error during scan: {str(e)}“) return jsonify({”status”: “error”, “message”: str(e), “timestamp”:

datetime.utcnow().isoformat() }), 500

@app.route(‘/opportunities’, methods=[‘GET’]) def get_opportunities(): “””Get all opportunities.””” try: # Get query parameters platform = request.args.get(‘platform’) category = request.args.get(‘category’) min_value = request.args.get(‘min_value’) max_value = request.args.get(‘max_value’) max_implementation_time = request.args.get(‘max_implementation_time’) competition_level = request.args.get(‘competition_level’) limit = int(request.args.get(‘limit’, 100))

```
    # Build query
    query = {}
    if platform:
        query["platform"] = platform
    if category:
        query["category"] = category
    if min_value:
        query["estimated_value"] = {"$gte": float(min_value)}
    if max_value:
        if "estimated_value" in query:
            query["estimated_value"]["$lte"] = float(max_value)
        else:
            query["estimated_value"] = {"$lte": float(max_value)}
    if max_implementation_time:
        query["implementation_time"] = {"$lte":
int(max_implementation_time)}
    if competition_level:
        query["competition_level"] = competition_level

    # Get opportunities from MongoDB
    opportunities =
list(opportunities_collection.find(query).limit(limit))

    # Convert ObjectId to string
    for opp in opportunities:
        if "_id" in opp:
            opp["_id"] = str(opp["_id"])

    return jsonify({
        "status": "success",
        "count": len(opportunities),
        "opportunities": opportunities,
        "timestamp": datetime.utcnow().isoformat()
    })
except Exception as e:
    logger.error(f"Error getting opportunities: {str(e)}")
    return jsonify({
        "status": "error",
        "message": str(e),
        "timestamp": datetime.utcnow().isoformat()
    }), 500
```

@app.route(‘/opportunities/top’, methods=[‘GET’]) def get_top_opportunities(): “””Get top opportunities based on estimated value.””” try: # Get query parameters limit = int(request.args.get(‘limit’, 10))

```
    # Get top opportunities from MongoDB
    opportunities =
list(opportunities_collection.find().sort("estimated_value", -
1).limit(limit))

    # Convert ObjectId to string
    for opp in opportunities:
        if "_id" in opp:
            opp["_id"] = str(opp["_id"])

    return jsonify({
        "status": "success",
        "count": len(opportunities),
        "opportunities": opportunities,
        "timestamp": datetime.utcnow().isoformat()
    })
except Exception as e:
    logger.error(f"Error getting top opportunities: {str(e)}")
    return jsonify({
        "status": "error",
        "message": str(e),
        "timestamp": datetime.utcnow().isoformat()
    }), 500
```

@app.route(‘/metrics’, methods=[‘GET’]) def get_metrics(): “””Get scanner metrics.””” try: metrics = metrics_collection.find_one({“metric”: “opportunity_scan”}) if metrics: metrics[“_id”] = str(metrics[“_id”]) else: metrics = { “metric”: “opportunity_scan”, “last_scan”: None, “opportunities_found”: 0, “platforms_scanned”: 0 }

```
    return jsonify({
        "status": "success",
        "metrics": metrics,
        "timestamp": datetime.utcnow().isoformat()
    })
except Exception as e:
    logger.error(f"Error getting metrics: {str(e)}")
    return jsonify({
        "status": "error",
        "message": str(e),
        "timestamp": datetime.utcnow().isoformat()
    }), 500
```

def run_scheduler(): “””Run the scheduler in a separate thread.””” logger.info(“Starting scheduler”)

```
# Schedule regular scans
schedule.every(SCAN_INTERVAL).seconds.do(scanner.scan_all_platforms)

# Run the scheduler
while True:
    schedule.run_pending()
    time.sleep(1)
```

if **name** == ‘**main**’: # Start scheduler in a separate thread scheduler_thread = threading.Thread(target=run_scheduler) scheduler_thread.daemon = True scheduler_thread.start()

```
# Run initial scan
scanner.scan_all_platforms()

# Start Flask app
app.run(host='0.0.0.0', port=5000)
```