Caption-Generator

Alhaan

2025-05-06 19:10:09

Contents

Project Overview

No project description provided.

Codebase Statistics

- **Total files:** 11
- **Total lines of code:** 453
- **Languages used:** plaintext (2), markdown (1), python (7), json (1)
- **Generated by:** code2pdf v0.2

Directory Structure

```
blip testing/
    └── .gitignore
    └── hf_status_notes.txt
    └── README.md
    └── requirements.txt
    └── run.py
    └── vercel.json
    ├── app/
        └── config.py
        └── models.py
        └── routes.py
        └── utils.py
        └── __init__.py
        ├── static/
            ├── css/
            ├── images/
        ├── templates/
    ├── instance/
    ├── test/
        └── test_captioning.py
    ├── uploads/
```

D:/Programming/blip testing/.gitignore

```
# System files
.DS_Store
```

```
Thumbs.db
.vscode

# Environment files
.env
.env.*

# Virtual env
venv



# Python bytecode
*.pyc
__pycache__/

# Other
uploads
test_.py

metrics.db
*.css
*.html
static/
```

D:/Programming/blip testing/hf_status_notes.txt

```
HuggingFace Inference API Status Notes
====================================

Current Status: DOWN (As of April 2024)
Issue: HuggingFace Inference API is experiencing widespread issues affecting model
 ↪  inference.

Monitoring Setup:
- Automated status checks every 5 minutes via frontend
- Backend verification before each API call
- Warning banner displays when service is down

Recovery Steps:
1. API will automatically resume when HuggingFace services are restored
2. No manual intervention needed
3. Warning banner will automatically hide when service is back online
```

```
Alternative Options During Downtime:
1. Consider implementing local model inference
2. Look into HuggingFace Inference Endpoints (paid service)
3. Explore alternative API providers

References:
- HuggingFace Status Page: https://status.huggingface.co/
- Official Issue Thread:
↪  https://discuss.huggingface.co/t/inference-api-stopped-working/
```

D:/Programming/blip testing/README.md

D:/Programming/blip testing/run.py

```python
from app import create_app

app = create_app()

if __name__ == '__main__':
    app.run(debug=True)
```

D:/Programming/blip testing/vercel.json

```json
{
    "version": 2,
    "builds": [
        {
            "src": "run.py",
            "use": "@vercel/python"
        }
    ],
    "routes": [
        {
            "src": "/(.*)",
            "dest": "run.py"
        }
    ]
}
```

D:/Programming/blip testing/app/config.py

```python
from dotenv import load_dotenv
import os

load_dotenv()

# API Keys
token_key = os.getenv('TOKEN_KEY')
cerebras_api_key = os.getenv('CEREBRAS_API_KEY')
huggingface_token = os.getenv('HUGGINGFACE_TOKEN')
phosus_api_key = os.getenv('PHOSUS_API_KEY')
phosus_key_id = int(os.getenv('PHOSUS_KEY_ID'))
```

D:/Programming/blip testing/app/models.py

```python
import json
import os
from datetime import datetime
from threading import Lock

class MetricsManager:
    def __init__(self):
        self.lock = Lock()
        self.metrics_file = '/tmp/metrics.json'
        self.default_metrics = {
            'total_images': 0,
            'total_users': set(),
            'visits': []
        }
        self._ensure_metrics_file()

    def _ensure_metrics_file(self):
        if not os.path.exists(self.metrics_file):
            with open(self.metrics_file, 'w') as f:
                json.dump(self.default_metrics, f, default=self._serialize)

    def _serialize(self, obj):
        if isinstance(obj, set):
            return list(obj)
        if isinstance(obj, datetime):
            return obj.isoformat()
        return obj
    def _deserialize(self, data):
        data['total_users'] = set(data['total_users'])
        return data

    def track_visit(self, ip_address, endpoint, image_processed=False):
        with self.lock:
            try:
                with open(self.metrics_file, 'r') as f:
                    data = json.load(f)
                    data = self._deserialize(data)

                if image_processed:
                    data['total_images'] += 1

                data['total_users'].add(ip_address)
                data['visits'].append({
                    'timestamp': datetime.utcnow().isoformat(),
                    'endpoint': endpoint,
                    'ip': ip_address
                })

                # Keep only last 1000 visits to avoid file size issues
                data['visits'] = data['visits'][-1000:]

                with open(self.metrics_file, 'w') as f:
                    json.dump(data, f, default=self._serialize)

            except Exception as e:
                print(f"Error tracking metrics: {e}")

    def get_metrics(self):
        try:
            with open(self.metrics_file, 'r') as f:
                data = json.load(f)
                data = self._deserialize(data)

                # Calculate last 24h visits
                now = datetime.utcnow()
                day_ago = (now - datetime.timedelta(days=1)).isoformat()
                last_24h = sum(1 for visit in data['visits']
                             if visit['timestamp'] > day_ago)
                return {
                    'total_images': data['total_images'],
                    'total_users': len(data['total_users']),
                    'last_24h': last_24h
                }
        except Exception as e:
            print(f"Error getting metrics: {e}")
            return {
                'total_images': 0,
                'total_users': 0,
                'last_24h': 0
            }

metrics = MetricsManager()
```

D:/Programming/blip testing/app/routes.py

```
from flask import Blueprint, render_template, request, jsonify, url_for, redirect,
 ↪  flash, send_from_directory
import requests
import os
from werkzeug.utils import secure_filename
from .utils import call_phosus_api, process_image, generate_cerebras_captions
from datetime import datetime

main = Blueprint('main', __name__)

UPLOAD_FOLDER = '/tmp/uploads'

# Ensure upload directory exists
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)



@main.route('/', methods=['GET'])
def index():
    return render_template('index.html')



# Serve uploaded files
@main.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory('/tmp/uploads', filename)
```

```
# Web Output Endpoint
@main.route('/generate-caption', methods=['POST'])
def generate_caption():
    try:
        if 'image' not in request.files:
            flash('No image file found', 'error')
            return redirect(url_for('main.index'))

        file = request.files['image']
        tone = request.form.get('tone', 'professional')
        file_path, filename = process_image(file)

        print(f"Processing image: {filename}")

        # Get base caption from Phosus
        base_caption, error = call_phosus_api(file_path)
        print(f"Base caption: {base_caption}")
        print(f"Phosus error: {error}")

        # Generate creative variations using Cerebras
        captions, cerebras_error = generate_cerebras_captions(file_path, tone,
 ↪  base_caption)

        image_url = url_for('main.uploaded_file', filename=filename)

        return render_template('result.html',
                               base_caption=base_caption or "Image processing
 ↪  incomplete",
                               captions=captions,
                               error_message=error or cerebras_error,
                               image_url=image_url,
                               tone=tone)

    except Exception as e:
        print(f"Critical Error in generate_caption: {str(e)}")
        flash("Unable to process image. Please try again.", 'error')
        return redirect(url_for('main.index'))


# api-endpoint
@main.route('/generate-caption-api', methods=['POST'])
```

```python
def generate_caption_api():
    try:
        if 'image' not in request.files:
            return jsonify({'error': 'No image file found'}), 400

        file = request.files['image']
        file_path = process_image(file)
        caption = call_phosus_api(file_path)

        return jsonify({'caption': caption})
    except ValueError as e:
        return jsonify({'error': str(e)}), 400
    except Exception as e:
        return jsonify({'error': f"Error: {e}"}), 500


@main.route('/api/status')
def api_status():
    from .utils import check_huggingface_status
    status = check_huggingface_status()
    return jsonify({
        'operational': status,
        'service': 'Phosus API',
        'timestamp': datetime.utcnow().isoformat()
    })


@main.route('/timeline')
def timeline():
    return render_template('timeline.html')
```

D:/Programming/blip testing/app/utils.py

```python
import requests
import time
import base64
from datetime import datetime, UTC, timedelta
from jwt import encode
from .config import token_key, cerebras_api_key, phosus_api_key, phosus_key_id
from cerebras.cloud.sdk import Cerebras
from werkzeug.utils import secure_filename
import os
```

```python
UPLOAD_FOLDER = '/tmp/uploads'

_cached_token = None
_token_expiry = None

def get_phosus_token():
    """Generate JWT token for Phosus API with caching"""
    global _cached_token, _token_expiry

    current_time = datetime.now(UTC)

    if _cached_token and _token_expiry and _token_expiry > current_time +
    ↪   timedelta(minutes=5):
        return _cached_token

    expiry_time = current_time + timedelta(days=1)
    payload = {
        'account_key_id': phosus_key_id,
        'exp': expiry_time,
        'iat': current_time
    }

    _cached_token = encode(payload, key=phosus_api_key, algorithm='HS256')
    _token_expiry = expiry_time
    return _cached_token

def call_phosus_api(image_path):
    """Get caption from Phosus API"""
    try:
        jwt_token = get_phosus_token()
        headers = {"authorizationToken": jwt_token}

        with open(image_path, "rb") as f:
            base64_img_str = base64.b64encode(f.read()).decode('utf-8')

        payload = {"image_b64": base64_img_str}

        print("Calling Phosus API...")
        response = requests.post(
            "https://api.phosus.com/icaption/v1",
            headers=headers,
            json=payload,
            timeout=30
        )

        if response.status_code == 200:
            return response.json()["prediction"], None

        try:
            error_data = response.json()
            if error_data.get("error", {}).get("msg") == "Insufficient Credit":
                print("Phosus API Credit Exhausted")
                return "An interesting image", "API credit limit reached. Please
                ↪ try again later."
        except:
            pass

        return "An interesting image", f"Error: Status {response.status_code}"

    except Exception as e:
        print(f"Phosus API Error: {str(e)}")
        return "An interesting image", str(e)

def check_huggingface_status(timeout=10):
    """Check if Phosus API is operational"""
    try:
        jwt_token = get_phosus_token()
        headers = {"authorizationToken": jwt_token}

        # Simple test request to Phosus API
        response = requests.get(
            "https://api.phosus.com/status",  # Assuming there's a status endpoint
            headers=headers,
            timeout=timeout
        )
        return response.status_code == 200
    except Exception:
        return False

def process_image(file):
    """Process the uploaded image: save it and return its path."""
    if file.filename == '':
        raise ValueError('No selected file')

    filename = secure_filename(file.filename)
    file_path = os.path.join(UPLOAD_FOLDER, filename)
    file.save(file_path)
    return file_path, filename

def get_emotion_prompt(emotion):
    prompts = {
        'happy': "Generate an upbeat and cheerful caption that emphasizes joy and
        ↪  positivity",
        'sad': "Create a thoughtful, melancholic caption that captures emotional
        ↪  depth",
        'thoughtful': "Write a philosophical and contemplative caption that
        ↪  provokes deep thinking",
        'geeky': "Generate a technically-oriented caption with subtle tech/geek
        ↪  culture references",
        'romantic': "Create a romantic and poetic caption that emphasizes beauty
        ↪  and emotion",
        'funny': "Write a humorous and witty caption that makes people smile"
    }
    return prompts.get(emotion, "Generate a creative caption")

def generate_cerebras_captions(image_path, emotion, base_caption):
    """Generate multiple captions using Cerebras API"""
    try:
        client = Cerebras(api_key=cerebras_api_key)

        messages = [
            {
                "role": "system",
                "content": (
                    f"You are a creative caption generator.
                    ↪  {get_emotion_prompt(emotion)} "
                    "Format each caption on a new line starting with a number and a
                    ↪  period. "
                    "Keep responses clean and concise without additional formatting
                    ↪  or quotes."
                )
            },
            {
                "role": "user",
                "content": f"Based on this image description: '{base_caption}',
                ↪  generate 5 unique captions."
            }
        ]
```

```python
        stream = client.chat.completions.create(
            messages=messages,
            model="llama3.1-8b",
            stream=True,
            max_completion_tokens=1024,
            temperature=0.8
        )

        full_response = ""
        for chunk in stream:
            full_response += (chunk.choices[0].delta.content or "")

        # Clean up the response
        captions = []
        for line in full_response.split('\n'):
            line = line.strip()
            if line and any(line.startswith(f"{i}.") for i in range(1, 6)):
                caption = line[line.find('.')+1:].strip()
                caption = caption.strip('"').strip("'")  # Remove quotes if present
                captions.append(caption)

        return captions[:5], None

    except Exception as e:
        error_message = f"Creative caption generation failed: {str(e)}"
        return [], error_message
```

D:/Programming/blip testing/app/__init__.py

```python
from flask import Flask

def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = 'your_secret_key'

    from .routes import main
    app.register_blueprint(main)

    return app
```

D:/Programming/blip testing/test/test_captioning.py

```python
import unittest
import sys
import os

# Ensure the `app` module is accessible
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from app import create_app  # Now it should work


class TestImageCaptioning(unittest.TestCase):

    def setUp(self):
        self.app = create_app()
        self.client = self.app.test_client()

    def test_image_upload_and_caption_generation(self):
        with open(r'uploads\29699386d0ad9d42d27d76d2ba584adb.jpg', 'rb') as img:
            response = self.client.post('/generate-caption',
                data={'image': (img, 'sample.jpg')},
                content_type='multipart/form-data')

            self.assertEqual(response.status_code, 200)
            self.assertIn('caption', response.json)
            self.assertIsInstance(response.json['caption'], str)

if __name__ == '__main__':
    unittest.main()
```