```
import concurrent.futures
import logging
import os
import uuid
from dataclasses import dataclass
from io import BytesIO
from typing import List

import backoff
import openai
import requests
from cachetools import TTLCache
from dotenv import load_dotenv
from openai import OpenAI
from PIL import Image
from pydantic import field_validator
from termcolor import colored

load_dotenv()

# Configure Logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)



def handle_errors(self, function):
```

```
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except Exception as error:
            logger.error(error)
            raise

    return wrapper


@dataclass
class Dalle3:
    """
    Dalle3 model class

    Attributes:
    -----------
    image_url: str
        The image url generated by the Dalle3 API

    Methods:
    --------
    __call__(self, task: str) -> Dalle3:
        Makes a call to the Dalle3 API and returns the image url

    Example:
```

```
--------

>>> dalle3 = Dalle3()

>>> task = "A painting of a dog"

>>> image_url = dalle3(task)

>>> print(image_url)

https://cdn.openai.com/dall-e/encoded/feats/feats_01J9J5ZKJZJY9.png


"""


model: str = "dall-e-3"

img: str = None

size: str = "1024x1024"

max_retries: int = 3

quality: str = "standard"

openai_api_key: str = None or os.getenv("OPENAI_API_KEY")

n: int = 1

save_path: str = "images"

max_time_seconds: int = 60

save_folder: str = "images"

image_format: str = "png"

client = OpenAI(

    api_key=openai_api_key,

)

cache = TTLCache(maxsize=100, ttl=3600)

dashboard: bool = False
```

```
def __post_init__(self):
    """Post init method"""
    if self.openai_api_key is None:
        raise ValueError("Please provide an openai api key")
    if self.img is not None:
        self.img = self.convert_to_bytesio(self.img)

    os.makedirs(self.save_path, exist_ok=True)

class Config:
    """Config class for the Dalle3 model"""

    arbitrary_types_allowed = True

@field_validator("max_retries", "time_seconds")
@classmethod
def must_be_positive(cls, value):
    if value <= 0:
        raise ValueError("Must be positive")
    return value

def read_img(self, img: str):
    """Read the image using pil"""
    img = Image.open(img)
    return img
```

```
def set_width_height(self, img: str, width: int, height: int):
    """Set the width and height of the image"""
    img = self.read_img(img)
    img = img.resize((width, height))
    return img

def convert_to_bytesio(self, img: str, format: str = "PNG"):
    """Convert the image to an bytes io object"""
    byte_stream = BytesIO()
    img.save(byte_stream, format=format)
    byte_array = byte_stream.getvalue()
    return byte_array

@backoff.on_exception(
    backoff.expo, Exception, max_time=max_time_seconds
)
def __call__(self, task: str):
    """
    Text to image conversion using the Dalle3 API

    Parameters:
    ----------
    task: str
        The task to be converted to an image

    Returns:
```

```
--------
Dalle3:
    An instance of the Dalle3 class with the image url generated by the Dalle3 API


Example:
--------
>>> dalle3 = Dalle3()
>>> task = "A painting of a dog"
>>> image_url = dalle3(task)
>>> print(image_url)
https://cdn.openai.com/dall-e/encoded/feats/feats_01J9J5ZKJZJY9.png
"""
if self.dashboard:
    self.print_dashboard()
if task in self.cache:
    return self.cache[task]
try:
    # Making a call to the the Dalle3 API
    response = self.client.images.generate(
        model=self.model,
        prompt=task,
        size=self.size,
        quality=self.quality,
        n=self.n,
    )
    # Extracting the image url from the response
```

```
        img = response.data[0].url

        filename = f"{self._generate_uuid()}.{self.image_format}"

        # Download and save the image
        self._download_image(img, filename)

        img_path = os.path.join(self.save_path, filename)
        self.cache[task] = img_path

        return img_path
    except openai.OpenAIError as error:
        # Handling exceptions and printing the errors details
        print(
            colored(
                (
                    f"Error running Dalle3: {error} try"
                    " optimizing your api key and or try again"
                ),
                "red",
            )
        )
        raise error

def _generate_image_name(self, task: str):
    """Generate a sanitized file name based on the task"""
```

```
        sanitized_task = "".join(
            char for char in task if char.isalnum() or char in " _ -"
        ).rstrip()
        return f"{sanitized_task}.{self.image_format}"

    def _download_image(self, img_url: str, filename: str):
        """
            Download the image from the given URL and save it to a specified filename within self.save_path.

        Args:
        img_url (str): URL of the image to download.
        filename (str): Filename to save the image.
        """
        full_path = os.path.join(self.save_path, filename)
        response = requests.get(img_url)
        if response.status_code == 200:
            with open(full_path, "wb") as file:
                file.write(response.content)
        else:
            raise ValueError(
                f"Failed to download image from {img_url}"
            )

    def create_variations(self, img: str):
        """
```

Create variations of an image using the Dalle3 API

Parameters:

---

img: str

    The image to be used for the API request

Returns:

---

img: str

    The image url generated by the Dalle3 API

Example:

>>> dalle3 = Dalle3()

>>> img = "https://cdn.openai.com/dall-e/encoded/feats/feats_01J9J5ZKJZJY9.png"

>>> img = dalle3.create_variations(img)

>>> print(img)

"""

try:

    response = self.client.images.create_variation(

        img=open(img, "rb"), n=self.n, size=self.size

)

    img = response.data[0].url

```
            return img
        except (Exception, openai.OpenAIError) as error:
            print(
                colored(
                    (
                        f"Error running Dalle3: {error} try"
                        " optimizing your api key and or try again"
                    ),
                    "red",
                )
            )
            print(
                colored(
                    f"Error running Dalle3: {error.http_status}",
                    "red",
                )
            )
            print(
                colored(f"Error running Dalle3: {error.error}", "red")
            )
            raise error

    def print_dashboard(self):
        """Print the Dalle3 dashboard"""
        print(
```

```
        colored(
            f"""Dalle3 Dashboard:
                --------------------


                Model: {self.model}
                Image: {self.img}
                Size: {self.size}
                Max Retries: {self.max_retries}
                Quality: {self.quality}
                N: {self.n}
                Save Path: {self.save_path}
                Time Seconds: {self.time_seconds}
                Save Folder: {self.save_folder}
                Image Format: {self.image_format}
                --------------------



                """,
            "green",
        )
    )

def process_batch_concurrently(
    self, tasks: List[str], max_workers: int = 5
):
    """
```

Process a batch of tasks concurrently

Args:

tasks (List[str]): A list of tasks to be processed

max_workers (int): The maximum number of workers to use for the concurrent processing

Returns:

----

results (List[str]): A list of image urls generated by the Dalle3 API

Example:

>>> dalle3 = Dalle3()

>>> tasks = ["A painting of a dog", "A painting of a cat"]

>>> results = dalle3.process_batch_concurrently(tasks)

>>> print(results)

['https://cdn.openai.com/dall-e/encoded/feats/feats_01J9J5ZKJZJY9.png',

"""

```
with concurrent.futures.ThreadPoolExecutor(
    max_workers=max_workers
) as executor:
    future_to_task = {
        executor.submit(self, task): task for task in tasks
    }
```

```
results = []
for future in concurrent.futures.as_completed(
    future_to_task
):
    task = future_to_task[future]
    try:
        img = future.result()
        results.append(img)

        print(f"Task {task} completed: {img}")
    except Exception as error:
        print(
            colored(
                (
                    f"Error running Dalle3: {error} try"
                    " optimizing your api key and or try"
                    " again"
                ),
                "red",
            )
        )
        print(
            colored(
                (
                    "Error running Dalle3:"
                    f" {error.http_status}"
```

```
                ),
                "red",
            )
        )
        print(
            colored(
                f"Error running Dalle3: {error.error}",
                "red",
            )
        )
        raise error

def _generate_uuid(self):
    """Generate a uuid"""
    return str(uuid.uuid4())

def __repr__(self):
    """Repr method for the Dalle3 class"""
    return f"Dalle3(image_url={self.image_url})"

def __str__(self):
    """Str method for the Dalle3 class"""
    return f"Dalle3(image_url={self.image_url})"

@backoff.on_exception(
    backoff.expo, Exception, max_tries=max_retries
```

```
)

def rate_limited_call(self, task: str):
    """Rate limited call to the Dalle3 API"""
    return self.__call__(task)
```