```
import json
import os
import platform
import sys
import traceback
from dataclasses import dataclass
from datetime import datetime
from typing import Any, Dict, List, Optional, Tuple

import psutil
import requests
from loguru import logger
from swarm_models import OpenAIChat

from swarms.structs.agent import Agent


@dataclass
class SwarmSystemInfo:
    """System information for Swarms issue reports."""

    os_name: str
    os_version: str
    python_version: str
    cpu_usage: float
    memory_usage: float
    disk_usage: float
    swarms_version: str  # Added Swarms version tracking
    cuda_available: bool  # Added CUDA availability check
    gpu_info: Optional[str]  # Added GPU information


class SwarmsIssueReporter:
    """
    Production-grade GitHub issue reporter specifically designed for the Swarms library.
    Automatically creates detailed issues for the https://github.com/kyegomez/swarms repository.

    Features:
    - Swarms-specific error categorization
    - Automatic version and dependency tracking
    - CUDA and GPU information collection
    - Integration with Swarms logging system
    - Detailed environment information
    """

    REPO_OWNER = "kyegomez"
    REPO_NAME = "swarms"
    ISSUE_CATEGORIES = {
        "agent": ["agent", "automation"],
        "memory": ["memory", "storage"],
        "tool": ["tools", "integration"],
        "llm": ["llm", "model"],
        "performance": ["performance", "optimization"],
        "compatibility": ["compatibility", "environment"],
    }

    def __init__(
        self,
        github_token: str,
        rate_limit: int = 10,
        rate_period: int = 3600,
        log_file: str = "swarms_issues.log",
        enable_duplicate_check: bool = True,
    ):
        """
        Initialize the Swarms Issue Reporter.

        Args:
            github_token (str): GitHub personal access token
            rate_limit (int): Maximum number of issues to create per rate_period
            rate_period (int): Time period for rate limiting in seconds
            log_file (str): Path to log file
            enable_duplicate_check (bool): Whether to check for duplicate issues
        """
        self.github_token = github_token
        self.rate_limit = rate_limit
        self.rate_period = rate_period
        self.enable_duplicate_check = enable_duplicate_check
        self.github_token = os.getenv("GITHUB_API_KEY")

        # Initialize logging
        log_path = os.path.join(os.getcwd(), "logs", log_file)
        os.makedirs(os.path.dirname(log_path), exist_ok=True)
        logger.add(
            log_path,
            rotation="1 day",
            retention="1 month",
            compression="zip",
        )

        # Issue tracking
        self.issues_created = []
        self.last_issue_time = datetime.now()

    def _get_swarms_version(self) -> str:
        """Get the installed version of Swarms."""
        try:
            import swarms

            return swarms.__version__
        except:
            return "Unknown"

    def _get_gpu_info(self) -> Tuple[bool, Optional[str]]:
        """Get GPU information and CUDA availability."""
        try:
            import torch

            cuda_available = torch.cuda.is_available()
            if cuda_available:
                gpu_info = torch.cuda.get_device_name(0)
                return cuda_available, gpu_info
            return False, None
        except:
            return False, None

    def _get_system_info(self) -> SwarmSystemInfo:
        """Collect system and Swarms-specific information."""
        cuda_available, gpu_info = self._get_gpu_info()

        return SwarmSystemInfo(
            os_name=platform.system(),
            os_version=platform.version(),
            python_version=sys.version,
            cpu_usage=psutil.cpu_percent(),
            memory_usage=psutil.virtual_memory().percent,
            disk_usage=psutil.disk_usage("/").percent,
            swarms_version=self._get_swarms_version(),
            cuda_available=cuda_available,
            gpu_info=gpu_info,
        )


    def _categorize_error(
        self, error: Exception, context: Dict
    ) -> List[str]:
        """Categorize the error and return appropriate labels."""
        error_str = str(error).lower()
        type(error).__name__

        labels = ["bug", "automated"]

        # Check error message and context for category keywords
        for (
            category,
            category_labels,
        ) in self.ISSUE_CATEGORIES.items():
            if any(
                keyword in error_str for keyword in category_labels
            ):
                labels.extend(category_labels)
                break

        # Add severity label based on error type
        if issubclass(type(error), (SystemError, MemoryError)):
            labels.append("severity:critical")
        elif issubclass(type(error), (ValueError, TypeError)):
            labels.append("severity:medium")
        else:
            labels.append("severity:low")

        return list(set(labels))  # Remove duplicates

    def _format_swarms_issue_body(
        self,
        error: Exception,
        system_info: SwarmSystemInfo,
        context: Dict,
    ) -> str:
        """Format the issue body with Swarms-specific information."""
        return f"""
        ## Swarms Error Report
        - **Error Type**: {type(error).__name__}
        - **Error Message**: {str(error)}
        - **Swarms Version**: {system_info.swarms_version}

        ## Environment Information
        - **OS**: {system_info.os_name} {system_info.os_version}
        - **Python Version**: {system_info.python_version}
        - **CUDA Available**: {system_info.cuda_available}
        - **GPU**: {system_info.gpu_info or "N/A"}
        - **CPU Usage**: {system_info.cpu_usage}%
        - **Memory Usage**: {system_info.memory_usage}%
        - **Disk Usage**: {system_info.disk_usage}%


        ## Stack Trace
        {traceback.format_exc()}


        ## Context
        {json.dumps(context, indent=2)}


        ## Dependencies
        {self._get_dependencies_info()}


        ## Time of Occurrence
        {datetime.now().isoformat()}


        ---
        *This issue was automatically generated by SwarmsIssueReporter*
        """


    def _get_dependencies_info(self) -> str:
        """Get information about installed dependencies."""
        try:
            import pkg_resources


            deps = []
            for dist in pkg_resources.working_set:
                deps.append(f"- {dist.key} {dist.version}")
            return "\n".join(deps)
        except:
            return "Unable to fetch dependency information"


    # First, add this method to your SwarmsIssueReporter class
    def _check_rate_limit(self) -> bool:
        """Check if we're within rate limits."""
        now = datetime.now()
        time_diff = (now - self.last_issue_time).total_seconds()

        if (
            len(self.issues_created) >= self.rate_limit
            and time_diff < self.rate_period
        ):
            logger.warning("Rate limit exceeded for issue creation")
            return False

        # Clean up old issues from tracking
        self.issues_created = [
            time
            for time in self.issues_created
            if (now - time).total_seconds() < self.rate_period
        ]

        return True
```

```
def report_swarms_issue(
    self,
    error: Exception,
    agent: Optional[Agent] = None,
    context: Dict[str, Any] = None,
    priority: str = "normal",
) -> Optional[int]:
    """
    Report a Swarms-specific issue to GitHub.

    Args:
        error (Exception): The exception to report
        agent (Optional[Agent]): The Swarms agent instance that encountered the error
        context (Dict[str, Any]): Additional context about the error
        priority (str): Issue priority ("low", "normal", "high", "critical")

    Returns:
        Optional[int]: Issue number if created successfully
    """
    try:
        if not self._check_rate_limit():
            logger.warning(
                "Skipping issue creation due to rate limit"
            )
            return None
```

```
# Collect system information
system_info = self._get_system_info()

# Prepare context with agent information if available
full_context = context or {}
if agent:
    full_context.update(
        {
            "agent_name": agent.agent_name,
            "agent_description": agent.agent_description,
            "max_loops": agent.max_loops,
            "context_length": agent.context_length,
        }
    )

# Create issue title
title = f"[{type(error).__name__}] {str(error)[:100]}"
if agent:
    title = f"[Agent: {agent.agent_name}] {title}"

# Get appropriate labels
labels = self._categorize_error(error, full_context)
labels.append(f"priority:{priority}")

# Create the issue
url = f"https://api.github.com/repos/{self.REPO_OWNER}/{self.REPO_NAME}/issues"
```

```
    data = {
        "title": title,
        "body": self._format_swarms_issue_body(
            error, system_info, full_context
        ),
        "labels": labels,
    }

    response = requests.post(
        url,
        headers={
            "Authorization": f"token {self.github_token}"
        },
        json=data,
    )
    response.raise_for_status()

    issue_number = response.json()["number"]
    logger.info(
        f"Successfully created Swarms issue #{issue_number}"
    )

    return issue_number

except Exception as e:
    logger.error(f"Error creating Swarms issue: {str(e)}")
```

```
            return None



# Setup the reporter with your GitHub token
reporter = SwarmsIssueReporter(
    github_token=os.getenv("GITHUB_API_KEY")
)



# Force an error to test the reporter
try:
    # This will raise an error since the input isn't valid
    # Create an agent that might have issues
    model = OpenAIChat(model_name="gpt-4o")
    agent = Agent(agent_name="Test-Agent", max_loops=1)

    result = agent.run(None)

    raise ValueError("test")
except Exception as e:
    # Report the issue
    issue_number = reporter.report_swarms_issue(
        error=e,
        agent=agent,
        context={"task": "test_run"},
        priority="high",
)

print(f"Created issue number: {issue_number}")
```