Code Walkthrough

This Flask application integrates multiple components for dynamic content creation, including news retrieval, summarization, article generation, and social media post creation. It uses YAML configurations, error handling, and structured agents/tasks via the **crewai framework.**

Key Components

1. Environment Setup

python Copy code

```python
from dotenv import load_dotenv
import os

load_dotenv()
```

- **Purpose:** Loads environment variables from a `.env` file to configure API keys and other sensitive data.
- **Key Variable:** `OPENAI_API_KEY` for OpenAI integration.

2. Logging Setup

python Copy code

```python
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s'
logger = logging.getLogger(__name__)
```

- **Purpose:** Logs application activity for debugging and monitoring.
- Logs are timestamped, include log levels (e.g., INFO, ERROR), and messages.

3. Pydantic Models

```python
class SocialMediaPost(BaseModel):
    platform: str = Field(..., description="Social media platform")
    content: str = Field(..., description="Post content")

class ContentOutput(BaseModel):
    article: str = Field(..., description="Generated article in markdown")
    social_media_posts: List[SocialMediaPost] = Field(..., description="Related social med
```

- **Purpose:** Defines structured data models using `pydantic` for validation and type safety.
- **Models:**
  - `SocialMediaPost`: Represents a post for a specific platform.
  - `ContentOutput`: Represents the generated article and associated social media posts.

4. YAML Configuration Loader

```python
def load_yaml_config(file_path):
    try:
        with open(file_path, 'r') as file:
            return yaml.safe_load(file)
    except FileNotFoundError:
        logger.error(f"Configuration file not found: {file_path}")
        return {}
    except yaml.YAMLError as e:
        logger.error(f"Error parsing YAML file {file_path}: {e}")
        return {}
```

- **Purpose:** Reads and parses YAML files to load configurations for agents and tasks.
- Handles errors gracefully if the file is missing or invalid.

5. Tools Initialization

```python
search_tool = SerperDevTool()
scrape_tool = ScrapeWebsiteTool()
```

- Initializes tools (`SerperDevTool` and `ScrapeWebsiteTool`) for:
  - Fetching news data.
  - Scraping websites (if required).

---

6. Flask Application

```python
app = Flask(__name__)
```

- Initializes the Flask application.

---

7. Routes

Home Route

```python
@app.route("/", methods=["GET", "POST"])
def index():
    return render_template('index.html')
```

- **Purpose:** Renders the homepage where users can input a subject for content creation.

Error Handlers

```python
@app.errorhandler(404)
def not_found_error(error):
    return render_template('error.html', error_code=404, error_message="Page not found"),

@app.errorhandler(500)
def internal_error(error):
    return render_template('error.html', error_code=500, error_message="Internal Server Er
```

- **Purpose:** Displays custom error pages for 404 (Not Found) and 500 (Internal Server Error).

Content Creation Route

```python
@app.route('/create_content', methods=['POST'])
def create_content():
```

- **Purpose:** Handles the main functionality of content creation.
- Steps:
  1. **Input Validation:**

     ```python
     subject = request.form.get('subject')
     if not subject:
         return render_template('error.html', message="Subject is required"), 400
     ```

     - Ensures a subject is provided.

  2. **Agent and Task Setup:**

     ```python
     agents_config = configs.get('agents', {})
     tasks_config = configs.get('tasks', {})
     ```

     - Loads configurations for agents and tasks.
     - Initializes agents for retrieving news, summarizing, creating content, and generating social media posts.

3. **Crew Execution:**

```python
content_creation_crew = Crew(
    agents=[...],
    tasks=[...],
    verbose=False  # For development, verbose logging can be set to True
)
result = content_creation_crew.kickoff(inputs={'subject': subject})
```

- Orchestrates agents and tasks using `Crew`.
- Executes tasks in sequence:
  - **Retrieve News**
  - **Summarize News**
  - **Create Article**
  - **Generate Social Media Posts**

4. **Output Processing:**

```python
article = result.get('article', 'No article generated')
social_media_posts = result.get('social_media_posts', [])
```

- Extracts the article and social media posts from the result.
- Formats posts using `textwrap` for improved readability.

5. **Rendering Results:**

```python
return render_template(
    'result.html',
    article=article,
    social_media_posts=formatted_posts
)
```

8. Error Handling

- **At Agent/Task Level:**

```python
except Exception as agent_error:
    logger.error(f"Error in agent setup or execution: {agent_error}")
    return render_template('error.html', message="Failed to create content"), 500
```

- **Global Exception Handling:**

```python
except Exception as e:
    logger.error(f"Unexpected error in content creation: {e}")
    return render_template('error.html', message="An unexpected error occurred"), 500
```

Execution

Local Development

```python
if os.getenv('FLASK_ENV', 'development') == 'development':
    app.run(debug=False)
else:
    logger.warning("This script should not be run directly in production.")
```

- Runs the app locally with appropriate logging.
- Warns against using `app.run()` directly in production.

Key Features

1. **Dynamic Content Creation**:
    - Generates articles and social media posts based on user input.
2. **Agent-Based Architecture**:
    - Modular, with distinct agents for different tasks.
3. **Robust Error Handling**:
    - Comprehensive logging and custom error pages.
4. **YAML Configuration**:
    - Centralized and flexible task/agent settings.
5. **Extensibility**:
    - Can integrate additional tools or agents as needed.

PROJECT STRUCTURE:

```
crewai_app/
├── app.py                # Main Flask app
├── templates/
│   ├── index.html        # Homepage template
│   ├── result.html       # Results page
├── static/
│   ├── styles.css        # Optional CSS for styling
├── config/
│   ├── agents.yaml       # Agent configurations
│   ├── tasks.yaml        # Task configurations
├── requirements.txt      # Python dependencies
└── helper.py             # Helper functions (optional)
```

AGENTS (YAML)

```yaml
agents:
  - name: news_retriever_agent
    role: "News Retriever"
    goal: "Fetch the latest and most relevant news articles from trusted sources."
    backstory: "Specialized in accessing and extracting up-to-date information from various news outlets. Ensures that only relevant and reliable articles are retrieved."
    verbose: true
    allow_delegation: false
    llm: "openai/gpt-4o"
    tools:
      - "search_tool"
      - "scrape_tool"

  - name: summarizer_agent
    role: "News Summarizer"
    goal: "Condense articles into concise, readable summaries without losing key information."
    backstory: "A linguistic expert trained in identifying the core message of lengthy texts. Highly efficient in summarization for quick consumption."
    verbose: true
    allow_delegation: true
    llm: "openai/gpt-4o"
    tools:
      - "search_tool"
      - "scrape_tool"

  - name: content_creator_agent
    role: "Content Creator"
    goal: "Generate engaging content for daily news updates, including formatted articles and blog content."
    backstory: "A creative specialist adept at crafting user-friendly news content for blogs and newsletters. Ensures tone and style are audience-appropriate."
    verbose: true
    allow_delegation: true
    llm: "openai/gpt-4o"
    tools:
      - "search_tool"
      - "scrape_tool"

  - name: social_media_posts_agent
    role: "Social Media Post Creator"
    goal: "Design attention-grabbing and audience-specific social media posts from summarized news."
    backstory: "Focused on delivering concise, engaging, and visually appealing posts tailored for various social media platforms like Twitter, Instagram, and LinkedIn."
    verbose: true
    allow_delegation: true
    llm: "openai/gpt-4o"
    tools:
      - "search_tool"
      - "scrape_tool"
```

TASKS (YAML)

```yaml
tasks:
  - name: retrieve_news
    description: "Fetches the latest news articles from various sources based on the input topic or keyword
    input:
      - "topic"
    output:
      - "articles"

  - name: summarize_news
    description: "Summarizes the fetched articles into concise and readable key points."
    input:
      - "articles"
    output:
      - "summaries"

  - name: create_content
    description: "Generates engaging content for daily news updates based on the summaries and articles."
    input:
      - "articles"
      - "summaries"
    output:
      - "formatted_news_content"

  - name: generate_social_media_posts
    description: "Creates social media posts for promoting the daily news content."
    input:
      - "formatted_news_content"
    output:
      - "social_media_posts"
```