**Research Agent Workflow**

**1. User sends a query for citation**

* The [**main.py**](http://main.py) in the **app/** folder receives the user’s request to generate citations for the research.

**2. Configuration settings**

* [**config.py**](http://config.py) provides necessary settings and API keys (like Google Docs permission or language model keys) handling secret info securely.

**3. Preparing data formats**

* **models/schemas.py** structures the query and response into neat formats so the system can process them without confusion.

**4. Running the research workflow**

* Inside **research\_agent/**:
  + [**graph.py**](http://graph.py) manages the citation workflow steps, deciding when to start citation formatting.
  + [**nodes.py**](http://nodes.py) contains the function that specifically handles citation generation (like formatting bibliography entries).
  + [**state.py**](http://state.py) tracks the citation step's progress.
  + [**tools.py**](http://tools.py) includes helpers such as citation style format converters.

**5. Using external services to output citation**

* From **services/**:
  + **google\_docs.py** creates or updates the Google Doc report with formatted citations.
  + **n8n\_client.py** sends notifications to the user once citation and report generation are completed.
  + **llm\_service.py** allows the AI to help generate citation text properly.

**6. Smooth operations helpers**

* **utils/logger.py** records each action, including the citation step.
* **utils/error\_handlers.py** ensures any errors in citation formatting or saving are handled gracefully.

**7. API routes**

* **api/routes.py** exposes an endpoint to trigger citation creation or fetch citation status.

**8. Testing**

* **tests/** contains test scripts to verify that citation formatting and integration with Google Docs works without bugs.

**How to run the robot (Your project)**

1. Preparation
   * Activate the environment from venv/ to ensure all necessary tools are ready.
   * Use setup.sh to install all required packages from requirements.txt automatically for ease.
2. Starting the Robot
   * Run the main.py file, which starts a FastAPI web server.
   * This web server listens for user queries (questions) via API endpoints defined in api/routes.py.
3. Handling a User Query
   * When a query arrives, main.py routes it to services inside research\_agent/.
   * graph.py plans the steps for the query, like searching, reading, summarizing.
   * nodes.py executes each step: one node may perform web search, another extracts content, another generates the summary.
   * state.py tracks progress so the system knows what's left to do and handles errors with help from error\_handlers.py.
4. Citation Handling (If requested)
   * The pipeline includes steps specifically for formatting citations using tools in tools.py.
   * The generated citations are composed, formatted, and stored in the report.
5. Report Generation and Sharing
   * google\_docs.py creates or updates the output report in Google Docs for easy access.
   * n8n\_client.py sends notifications (like emails or Slack messages) when work is done.
   * llm\_service.py helps compose smart summaries or caption texts.
6. Logging and Debugging
   * Throughout the process, logger.py keeps detailed logs of all activities.
   * Errors or warnings are caught and handled to prevent crashes.
7. Testing
   * Run test scripts inside tests/ to verify each part works properly before deployment.

**Sample Query Execution Order**

1. **Client Request**:  
   User sends a research query like "What is AI?" through the API endpoint.
2. **API Validation:**
   * The request is validated to ensure it is in the correct format using Pydantic models.
   * A unique job ID is generated and returned with status 202 (Accepted).
3. **Background Processing Starts**:  
   The research workflow begins executing the LangGraph workflow.
4. **1. Query Intake Node:**
   * Validates the query format again.
   * Initializes the job state and logs the start.
5. **2. Web Search Node:**
   * Uses DuckDuckGo search to retrieve top N relevant results.
   * Extracts titles, URLs, and snippets.
6. **3. Content Filter Node:**
   * Ranks the search results by relevance.
   * Filters out low-quality and irrelevant results.
7. **4. Content Extraction Node**:
   * Visits each filtered URL to scrape full page content.
   * Converts HTML to text, handles possible errors or timeouts.
8. **5. Synthesizer Node:**
   * Uses a large language model to synthesize summaries from extracted content.
   * Combines multiple sources into a coherent summary.
9. **6. Citation Handler Node:**
   * Formats citations in proper style (e.g., APA).
   * Creates a list of references matching the summary.
10. **7. Report Generator Node:**
    * Compiles the summary, citations, and metadata into a markdown report.
11. **8. Error Handler Node:**
    * Catches and recovers from any errors during the workflow.
    * Generates error reports and logs failures.
12. **Google Docs Creation:**
    * Authenticates with Google API and creates a new document.
    * Inserts the formatted report content.
    * Sets permissions and generates a shareable link.
13. **n8n Workflow Trigger:**
    * Sends a webhook with job metadata to trigger post-processing.
    * Post-processes include storing data, sending email or Slack notifications.
14. **Update Job Status:**
    * Marks the job as completed in the system.
    * Stores the Google Doc URL and updates completion timestamps.
15. **Client Polling:**
    * The user can query job status or retrieve the final report link.

**How graph.py Manages Workflow**

1. **Purpose:**
   * It organizes and manages the complete research workflow for answering a user query.
   * It decides what tasks to do and in what order, like a smart planner deciding your day’s schedule.
2. **What it controls:**
   * Query Intake: Starting the process with the user’s query.
   * Search: Deciding when and how to do web searches.
   * Content Extraction: Scheduling web scraping of important content.
   * Synthesis: Summarizing collected data through AI models.
   * Citation Handling: Deciding when citations are generated and formatted.
   * Report Generation: Final creation of the report.
   * Error Handling: Managing fallback steps if something fails.
3. **How it works**:
   * It models the workflow as a graph or flowchart, where each node is a task, and edges are transitions between tasks.
   * When one task finishes, it automatically triggers the next appropriate task.
   * It tracks which tasks are completed with the help of state.py.
4. **Why it’s important:**
   * Keeps the system organized and efficient.
   * Ensures no step is missed.
   * Helps handle complex workflows with many steps in an automated way.

**How nodes.py manages workflow**

**Purpose:**

1. Executes individual tasks needed for research, such as searching, extracting, summarizing, or formatting.
2. Breaks down complex research into small, manageable jobs.

**What it controls:**

1. Web search task – fetching relevant pages based on the user's query.
2. Content extraction task – scraping and reading web page content.
3. Synthesis task – summarizing collected content using AI.
4. Citation formatting task – creating formatted citation entries.
5. Report building task – assembling final report pieces.

**How it works:**

1. Contains functions or classes where each performs one task (called a node).
2. Nodes are executed in sequence or parallel as planned by the workflow manager (graph.py).
3. Passes the output from one node to the next to build the final result step-by-step.
4. Handles task-specific errors and returns useful feedback to the workflow controller.

**Why it’s important:**

1. Modularizes the research process into clear, testable operations.
2. Makes development, debugging, and maintenance easier by isolating logic.
3. Enables flexible workflows by adding or removing nodes without affecting others.

**How state.py Manages Workflow**

**Purpose:**

1. Tracks and stores the current status of every task in the research workflow.
2. Maintains intermediate data like search results, extracted content, and partial summaries.
3. Provides a centralized place for managing workflow progress and retrying failed steps.

**What it controls:**

1. Records when a task (node) starts and completes successfully.
2. Captures errors or exceptions raised during task execution.
3. Stores final and intermediate outputs for each step.
4. Helps resume workflow after interruption or failure.

**How it works:**

1. Updates state objects or records in a database/file storage after each task execution.
2. Accepts queries to report current progress or status of the entire job or individual nodes.
3. Supports rollback or retry by storing task checkpoints.
4. Provides information to error handler modules for recovery actions.

**Why it’s important:**

1. Ensures reliable and fault-tolerant workflow execution.
2. Allows detailed monitoring and debugging of workflow.
3. Enables users to track the progress of their research requests.
4. Prevents redundant re-work by keeping track of completed tasks.

**How tools.py Manages Workflow**

**Purpose:**

1. Provides helper functions and utilities that assist the main tasks in the workflow.
2. Ensures common operations like citation formatting and content cleaning are done efficiently and consistently.

**What it controls:**

1. Citation style formatting (e.g., APA, MLA).
2. Text preprocessing such as removing unwanted characters, trimming, or splitting content.
3. Wrappers around external services like search API calls.
4. Utility functions to support complex node operations.

**How it works:**

1. Offers reusable functions called by nodes during workflow execution.
2. Keeps the code DRY (Don’t Repeat Yourself) by consolidating shared operations.
3. Helps maintain consistency and reliability across various tasks.

**Why it’s important**:

1. Reduces code duplication and bugs.
2. Speeds up development by providing tested helpers.
3. Improves maintainability and readability of the project.

**How google\_docs.py Manages Workflow**

**Purpose:**

1. Connects your project to Google Docs for creating, updating, and sharing research reports.
2. Automates converting the final summary, citations, and metadata into a well-formatted document.

**What it controls:**

1. Authenticates with Google API using secure tokens.
2. Creates new Google Docs for each research job or updates existing ones.
3. Inserts main content, citations, and formatting into the document.
4. Sets permissions and generates shareable links for user access.
5. Handles errors related to document creation, update, or API access.

**How it works**:

1. Receives finalized text content from the research workflow.
2. Uses Google Docs API calls to create or update a document with multiple sections.
3. Applies styles, headings, and formatting to ensure readability.
4. Produces a share link URL which is stored for user retrieval.

**Why it’s important:**

1. Provides a professional and accessible output format for the final report.
2. Saves users time by automating document creation and prevents manual formatting.
3. Enables easy sharing and collaboration through Google Docs links.

**How n8n\_client.py Manages Workflow**

**Purpose:**

1. Handles sending notifications and triggering automated workflows after the research report is ready.
2. Integrates with external automation tools like n8n to streamline post-processing and communication.

**What it controls:**

1. Sends webhook requests to n8n or other automation platforms with job metadata.
2. Sends notifications via email, Slack, or other channels to inform users about report completion.
3. Triggers further post-processing workflows like storing metadata or analytics.

**How it works:**

1. Listens for job completion events from the main workflow.
2. Constructs and sends HTTP webhook requests to n8n with detailed job info.
3. Receives responses and logs notification success or failure.

**Why it’s important:**

1. Keeps the user informed in real-time without manual checking.
2. Automates downstream processes that improve user experience.
3. Enables seamless integration with other software tools and communication platforms.

**How llm\_service.py works:**

1. **Purpose:**
   * Acts as a bridge connecting your project to a large AI model (LLM) that understands and generates human-like text.
   * Helps in summarizing research, creating answers, and formatting outputs.
2. **How it connects:**
   * Sends the AI model your processed query (with gathered content).
   * Receives AI-generated text summaries or responses.
3. **Key tasks:**
   * Prepares prompts to ask the AI to summarize or explain content.
   * Sends these prompts securely using API keys (from config.py).
   * Receives the text generated by the AI and sends it back to the workflow.
4. Why it's useful:
   * Automates creating human-readable, concise reports instead of raw data.
   * Makes the output natural and understandable, improving user experience.
   * Handles tasks like citation text generation and report narration.



